Universidad Autónoma de Querétaro
Facultad de Informática

DESARROLLO DE UN FRAMEWORK DE PRUEBAS AUTOMATIZADAS PARA LA VERIFICACIÓN DE APLICACIONES WEB.

Tesis
Que como parte de los requisitos para obtener el grado de
Maestro en Sistemas Computacionales.

Presenta
Dante Spindola Hernández.

Santiago de Querétaro, Agosto, 2021.

Universidad Autónoma de Querétaro
Facultad de Informática
Maestría en Sistemas Computacionales

DESARROLLO DE UN FRAMEWORK DE PRUEBAS AUTOMATIZADAS PARA LA VERIFICACIÓN DE APLICACIONES WEB.

TESIS

Que como parte de los requisitos para obtener el grado de
Maestro en Sistemas Computacionales.

Presenta:
Dante Spindola Hernández.

Dirigido por:
Carlos Alberto Olmos Trejo.

SINODALES

MSID. Carlos Alberto Olmos Trejo Presidente	_____ Firma
MSI. José Alejandro Vargas Días Secretario	_____ Firma
DRA. Gabriela Xicoténcatl Ramírez Vocal	_____ Firma
MSI. Diego Octavio Ibarra Corona Suplente	_____ Firma
DR. Alberto Vázquez Cervantes Suplente	_____ Firma

_____ MSI Juan Salvador Hernández Valerio Director de la Facultad de Informática	_____ Dra. Ma. Guadalupe Flavia Loarca Piña Directora de Investigación y Posgrado

Centro Universitario
Santiago de Querétaro
Agosto, 2021
México

RESUMEN

El propósito de todo framework es facilitar las cosas a la hora de desarrollar una aplicación, para enfocarse en el problema a resolver y olvidándonos de implementar funcionalidades que son de uso común. Con base en esta problemática, el presente documento tiene como objetivo el desarrollo de un framework gráfico, que permita implementar pruebas automatizadas de manera eficiente, sin la necesidad de tener una habilidad de programación avanzada, dado que la herramienta solicita parámetros específicos a través de una interfaz gráfica, esto ahorra tiempo y costos en el proceso de desarrollo de software, específicamente en el área de pruebas. La metodología empleada en este trabajo consta de siete etapas diferenciadas, las cuales son: documentación, análisis, diseño, desarrollo, implementación, evaluación y pruebas, todo esto en un orden secuencial dado que la metodología se basa en el modelo de desarrollo de software en cascada, por lo tanto, puesto que es una herramienta basada en software que evalúa el correcto comportamiento de una aplicación, debe seguir las pautas correspondientes de cualquier modelo de desarrollo de software. Los criterios de aceptación fueron la reducción del tiempo de ejecución e implementación, siendo que el primero se da de manera intrínseca en cualquier herramienta de automatización y no representa ninguna ventaja competitiva, sin embargo, el segundo es una ventaja agregada sobre el primero, por tener una interfaz gráfica que guía al usuario en el momento de la implementación, generando un desarrollo transparente y una curva de aprendizaje considerablemente más baja que cualquier método convencional y en consecuencia de forma natural aumenta la cobertura de las pruebas automatizadas, luego el tiempo ahorrado se aprovecha de manera efectiva para diferentes fines, como el mantenimiento, desarrollo o implementación de nuevas funciones o mejoras en el framework. Como resultado, se obtuvo una reducción del 90% en el tiempo de ejecución con respecto al manual, y en cuanto al tiempo de implementación el ahorro fue del 66% en escenarios específicos, en vista de que este puede variar en función de diferentes factores como la experiencia, conocimientos y habilidades del desarrollador.

Palabras clave: Framework, Automatización, Pruebas de Software.

SUMMARY

The purpose of all frameworks is to make things easier at the moment of developing an application, focusing on the real problem, and forgetting to implement functionalities that are of common use. Based on this problem, the present document has as objective the development of a graphical framework, that allows us to implement automated tests in an efficient manner, without the necessity to have an advanced programming skill, since tool only request specific parameter thought a graphical interface, this saves time and costs in the software development process, specifically in the testing area. The methodology used in this work consists of seven different stages, which are: documentation, analysis, design, development, implementation, evaluation and testing, all this in a sequential order since methodology is based on the waterfall software development model, therefore, since it is a software-based tool that evaluate the correct behavior of an application, it must follow the corresponding guidelines of any software development model. The acceptance criteria were the reduction of the execution and implementation time, being that the first is given in an intrinsic way in any automation tool and does not represent any competitive advantage, however, the second is an added advantage over the first, due to having a graphical interface that guides the user to the moment of implementation, it generates a transparent development and a learning curve considerably lower than any conventional method and consequently in a natural way increases the coverage of automated tests, Then the saved time is used effectively for different purposes, such as the maintenance, development or implementation of new functions or improvements in the framework. As a result, a 90% reduction in execution time was obtained with respect to the manual, and regarding the implementation time the savings was 66% in specific scenarios, since this may vary depending on different factors such as experience, knowledge and skills of the developer.

Key words: Framework, Automation, Software Testing.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Definición del Problema

Al momento de automatizar pruebas, siempre surge el dilema de que framework utilizar para llevar a cabo esta tarea de la manera más adecuada, por lo que es necesario buscar alternativas que permitan implementar pruebas automatizadas de una manera eficiente. En este momento, no se cuenta con una herramienta de pruebas oficial para entornos web específicamente, pues las opciones para este tipo de aplicaciones son diversas y usualmente están montadas sobre lenguajes de programación de alto nivel como los son Java, Python, C# entre otros, además para poder utilizar estas herramientas es necesario tener conocimiento sobre los lenguajes de programación anteriormente mencionados, lo cual hace que se vuelva más difícil el poder automatizar en tiempos reducidos, dado que sin experiencia previa esto ralentiza los tiempos de implementación y por lo tanto la cobertura de pruebas automatizadas, generando que diferentes tipos de clientes no acepten este tipo de propuestas debido a que están ensambladas en aplicaciones diseñadas localmente sobre los ambientes descritos anteriormente, generando cierto tipo de desconfianza, ya que es importante mencionar que la calidad de su producto, así como la reputación de este, depende de la fiabilidad de las pruebas aplicadas.

1.2 Objetivos generales

Desarrollar un framework gráfico de automatización de pruebas funcionales para la verificación de aplicaciones web.

1.3 Objetivos parciales o específicos

- Identificar las necesidades de uso común entre los desarrolladores de pruebas automatizadas.
- Diseñar las interfaces gráficas que serán utilizadas para la configuración de parámetros.
- Desarrollar la integración de las interfaces gráficas con el secuenciador y el motor principal de la automatización web.

- Desarrollar un instalador que permita distribuir la herramienta de una manera práctica.
- Evaluar la rapidez y sencillez al momento de implementar pruebas automatizadas con respecto a los métodos tradicionales.
- Evaluar la reducción de tiempos de ejecución.

1.4 Justificación

La automatización de pruebas de software involucra el uso de códigos para desarrollar aplicaciones que ejecutan y supervisan las pruebas por sí mismas, de esta forma se reduce el tiempo de ejecución de manera drástica, sin embargo, el uso de códigos genera naturalmente que las personas encargadas de implementar las pruebas automatizadas deban tener conocimientos sobre los lenguajes de programación utilizados, por consiguiente cuando se tiene que contratar personal para atacar este tipo de problemática se buscan perfiles que manejen este tipo de lenguajes, lo cual hace que el abanico de opciones se vea reducido de manera considerable, generando que se contraten expertos en herramientas más que en pruebas de software, por eso la necesidad de contar con frameworks de automatización que requieran las mínimas habilidades de programación necesarias, de esta manera se propiciara a que se enfoque la mayoría del esfuerzo en el diseño así como en la aplicación de técnicas de prueba, y de este modo garantizar una mayor confiabilidad en las pruebas efectuadas, dado que con este tipo de herramientas será más sencillo el traducir el caso de prueba manual a su forma automatizada y por lo tanto ahorrar tiempo a la hora de la implementación.

1.5 Hipótesis

Un Framework de pruebas con un entorno de configuración gráfico y modular permitirá generar casos de prueba en menor tiempo, lo cual incrementara la cobertura de pruebas automatizadas, así como la reducción de tiempos de ejecución, además de consentir el reusó de módulos de prueba.

2. ASPECTOS TEÓRICOS

2.1 Antécédentes

2.1.1 Pruebas de software

Las pruebas de software se definen como "*Una investigación empírica y técnica realizada para proporcionar a las partes interesadas información sobre la calidad del producto o servicio bajo prueba*". Por lo tanto, los principales objetivos de esta actividad son detectar y prevenir defectos, así como asegurar el comportamiento del software probado (Bhondokar, Ranawade, Jadhav and Vibhute, 2015).

De acuerdo con la International Software Testing Qualification Board (2018) se define que los niveles de prueba son grupos de actividades que se organizan y gestionan en conjunto, cada nivel es una instancia del proceso de pruebas, que consta de diferentes actividades que están realizadas en relación con cada nivel del desarrollo de software.

Los niveles de prueba son:

- Pruebas de componente: Las pruebas de componente (conocidas como pruebas de módulo o unidad) se centran en componentes que están separados.
- Pruebas de integración: Las pruebas de integración se enfocan en la interacción entre sistemas o componentes.
- Pruebas de sistema: Las pruebas de sistema se centran en el comportamiento y las capacidades de un sistema o producto completo, considerando las tareas de un extremo a otro que el sistema puede realizar, así como los comportamientos no funcionales que presenta mientras realiza esas tareas.
- Pruebas de aceptación: Las pruebas de aceptación, así como las pruebas del sistema, se enfocan en el comportamiento y las características de un conjunto, sistema o producto, estas producen información para evaluar la capacidad del sistema para su implementación y uso por parte del usuario final (cliente).

El nivel de prueba que es soportado por la herramienta desarrollada en este trabajo es el de "*pruebas de sistema*" a su vez también podría aplicar para "*pruebas de aceptación*" dependiendo directamente de quien utilice esta herramienta (cliente o proveedor).

La ISTQB define que los tipos de prueba son un conjunto de actividades predestinadas a probar características específicas de un sistema, o una parte particular de un sistema basado en objetivos de prueba específicos.

Los tipos de prueba son:

- Pruebas funcionales: Las pruebas funcionales de un sistema implican pruebas que evalúan las funciones que un sistema debe realizar. Las pruebas funcionales toman en cuenta el comportamiento del software, por lo que estas utilizan técnicas de caja negra para derivar condiciones de prueba y casos de prueba para la funcionalidad correspondiente al componente o sistema.
- Pruebas no funcionales: Las pruebas no funcionales de un sistema o software evalúan las características de estos, como la usabilidad, rendimiento o seguridad. Se pueden utilizar técnicas de caja negra para derivar condiciones de prueba y casos de prueba para las pruebas no funcionales.
- Pruebas de caja blanca: Las pruebas de caja blanca derivan pruebas que están basadas en la implementación o estructura interna de un sistema o software. La estructura puede incluir arquitectura, código, flujos de trabajo y/o flujos de datos dentro del sistema.
- Pruebas relacionadas con el cambio: Cuando se realizan cambios en un sistema o software, ya sea para arreglar un defecto o debido a cambios nuevos en una funcionalidad, se deben ejecutar pruebas para confirmar que los cambios han corregido el defecto o implementado la funcionalidad correctamente y no ha causado consecuencias adversas imprevistas, este tipo de pruebas se clasifican en:
    - Pruebas de confirmación: Una vez reparado un defecto, el software debe evaluarse con los casos de prueba que fallaron debido a este, los casos de prueba deben volver a ejecutarse en la nueva versión del software.

- Pruebas de regresión: Es posible que un cambio realizado en una parte del código ya sea una corrección u otro tipo de ajuste, pueda afectar de manera imprevista el comportamiento de otras partes del código, ya sea dentro del componente o en otros componentes del mismo sistema, incluso en diferentes sistemas.

Los tipos de pruebas que son automatizables por la herramienta desarrollada en este trabajo son las "*pruebas funcionales*", "*no funcionales*", así como las pruebas relacionadas con el cambio, específicamente las "*pruebas de regresión*".

Uno de los tipos de pruebas más comunes en la actualidad son las pruebas de regresión, las pruebas de regresión son caras, pero una actividad esencial en el mantenimiento del software. Las pruebas de regresión validan el software modificado y aseguran que las partes modificadas del programa no introduzcan errores inesperados (Amir Ngah, Malcolm Munro and Mohammad Abdallah, 2017).

Las pruebas de regresión usualmente están basadas en pruebas funcionales, las pruebas funcionales se realizan para garantizar que el software funcione según los requisitos del usuario, así mismo basan sus casos de prueba en las especificaciones del componente de software, alimentándolas con entradas y examinando las salidas obtenidas en comparación con la salida esperada que está dada por el mismo criterio de aceptación.

Las pruebas funcionales, no funcionales y de regresión consumen bastante tiempo, son repetitivas y requieren de máxima eficiencia, aunado a la creciente demanda de disminuir los tiempos de desarrollo, se busca automatizar esta clase de pruebas con algún tipo de herramienta, pues de esta manera se reducirán los tiempos de ejecución y se obtendrá una ventaja competitiva en cuanto a los tiempos de desarrollo con respecto a la competencia.

2.1.2 Automatización de pruebas de software

Las pruebas automatizadas son un proceso que utiliza software separado del software bajo prueba para controlar la ejecución de pruebas y la comparación de los resultados reales con los resultados esperados (D. Huizinga and A. Kolawa, 2007).

Las pruebas automatizadas generalmente ahorran tiempo, así el evaluador puede ejecutar de manera eficiente una gran cantidad de pruebas en un período corto, y de la misma manera las tareas importantes y repetitivas, así como las pruebas que serían difíciles de realizar manualmente se pueden automatizar.

Además de ahorrar tiempo, las pruebas de automatización ahorran dinero y esfuerzo, aumentan la calidad de las tareas de prueba y también ayudan a mejorar la precisión del software (M. Polo, P. Reales, M. Piattini, and C. Ebert, 2013).

En la Tabla 1 se muestran algunas ventajas y desventajas de la automatización de pruebas:

Ventajas	*Desventajas*
Mejora la precisión y la búsqueda rápida de errores en comparación con las pruebas manuales.	*Elegir la herramienta adecuada requiere de esfuerzo, tiempo y un plan de evolución considerable.*
Ahorra tiempo y esfuerzo al hacer que las pruebas sean más eficientes.	*Requiere conocimiento de la herramienta de prueba.*
Aumenta la cobertura de la prueba porque se pueden utilizar varias herramientas a la vez, lo que permite realizar pruebas en paralelo con diferentes escenarios de prueba.	*El costo de comprar la herramienta de prueba y en el caso de los métodos de reproducción, el mantenimiento de la prueba es caro.*
El script de prueba de automatización es repetible.	*Se requiere competencia para escribir los scripts de prueba de automatización.*

Tabla 1. Tabla comparativa ventajas/desventajas de la automatización de pruebas. Fuente: Mubarak Albarka Umar and Chen Zhanfang.

La ejecución de pruebas sobre software nuevo o prefabricado ha sido un reto para las organizaciones, dado que por lo general esto implica la contratación de personal extra y de disponer de un tiempo considerable para lograr una cobertura de pruebas óptima. Dicha situación puede volverse complicada en organizaciones que implementan soluciones tipo ERP (Enterprise Resource Planning) o CRM (Customer Relationship Management), dónde en varias ocasiones se realizan desarrollos a la medida y/o configuraciones sobre las soluciones adquiridas, esto pueden llegar a afectar el funcionamiento normal de dichas plataformas. Frente a dicha problemática, algunas organizaciones han optado por implementar pruebas automatizadas, aunque estas al principio arrojan resultados positivos, al largo plazo implica de una alta mantenibilidad de los desarrollos realizados para dicho fin, de hecho, uno de los impedimentos para implementar la automatización es la inflexibilidad y el bajo reusó (Salazar, 2016).

Así mismo la ISTQB menciona que los sistemas de software son una parte integral de la vida, desde las aplicaciones comerciales (Bancos) hasta los productos de servicio (Automóviles). No obstante, el propósito de este trabajo es desarrollar una solución dirigida específicamente a las aplicaciones web y sus pruebas correspondientes.

La importancia de las pruebas automatizadas de aplicaciones web se deriva de la creciente dependencia de estos sistemas para las funciones empresariales, sociales, organizativas y gubernamentales (Alshahwan N and Harman M, 2015).

Por otra parte, las empresas que han intentado realizar una implementación de automatización de pruebas se enfrentan al hecho de que posteriormente el mantenimiento de las pruebas automatizadas tiene un costo elevado, además de que para su elaboración es necesario contar con personal adecuado con el conocimiento y experiencia en el manejo de las herramientas que permiten la automatización (Salazar, 2016).

Todos los proyectos, por muy pequeños que sean, pueden llegar a tener una cantidad de casos de pruebas muy elevado, sin contar que las pruebas se repetirán varias veces debido a las pruebas de regresión. Estos proyectos, necesitan de una administración, planificación y ejecución, así como de herramientas que permitan realizar pruebas automatizadas.

Algunas de las herramientas de automatización de pruebas más utilizadas son:

- Selenium: Es un framework portátil para probar aplicaciones web, también proporciona un lenguaje de prueba específico del dominio (Selenese) para escribir pruebas en varios lenguajes de programación populares, incluidos C#, Groovy, Java, Perl, PHP, Python, Ruby y Scala.
- Quick Test Professional: Es el módulo de automatización de la empresa HP, permite la automatización de casos de prueba y los scripts son programados en Visual Basic Script.
- SoapUI: Permite probar, simular y generar códigos de servicio web partiendo del contrato de estos en formato WSDL (Web Services Description Language) y con vinculo SOAP (Simple Object Access Protocol) sobre http.
- Cucumber: Permite la automatización de la prueba de aceptación para aplicaciones web, está basada en BDD (Behavior Driven Development) y el lenguaje de generación de scripts es Ruby.
- IBM Rational Automation Framework: Es el módulo de generación de scripts y automatización de pruebas de la empresa IBM que se puede integrar con los módulos de gestión, además permite la generación de scripts con varios lenguajes de programación.

La selección de la herramienta de automatización depende en gran medida de la tecnología en la que se basa la aplicación bajo prueba, por lo que es de suma importancia seleccionar la opción correcta para obtener buenos resultados, así como hacer uso de frameworks que sirvan como pautas o reglas al momento de desarrollar casos de prueba automatizados.

2.1.3 Framework de automatización

Un framework de automatización de pruebas es un conjunto de pautas como estándares de codificación, manejo de datos de prueba, tratamiento de repositorios de objetos, etc., que cuando se siguen durante la secuencia de comandos de automatización producen resultados beneficiosos como una mayor reutilización de código, una mayor portabilidad, un menor costo de mantenimiento de la secuencia de comandos, etc.

También se puede definir como un conjunto de funcionalidades encapsuladas que facilitan la automatización para procesarse a sí mismo. Estas herramientas ayudan mucho no solo en controlar y monitorear la ejecución de la prueba en escenarios cotidianos, sino también en el aumento de la reutilización de las pruebas automatizadas (Hanna, Elsayed y Mostafa, 2018).

De acuerdo con las definiciones anteriores, podemos concluir que un framework de automatización es un conjunto de procedimientos, conceptos y entornos en el que las pruebas serán creadas e implementadas de una manera eficiente.

El objetivo principal de todo framework es facilitar las cosas al momento de desarrollar una aplicación, haciendo que nos centremos en el verdadero problema y nos olvidemos de implementar funcionalidades que son de uso genérico como puede ser el registro de un usuario, establecer conexión con la base de datos, manejo de sesiones de usuario o el almacenamiento en base de datos de contenido cacheado (Gómez, 2014).

Actualmente existen diferentes tipos de frameworks de automatización, los cuales difieren entre si debido a su compatibilidad con respecto a factores clave para realizar una automatización exitosa, tales como la reutilización de módulos de prueba, el conocimiento intrínseco a la hora del desarrollo, así como la sencillez al momento del mantenimiento, el criterio de selección dependerá naturalmente de las condiciones que se tengan en el proyecto al cual se desee implementar una solución de este índole.

De acuerdo con Milad Hanna, Nahla El-Haggar and Mostafa Sami (2014) los frameworks de automatización de pruebas más comunes son:

- Linear Scripting: Se basa en configurar la herramienta de prueba en el modo de registro mientras se realizan acciones sobre la aplicación bajo prueba, el script de prueba generado consta de una serie de instrucciones que utilizan el lenguaje de programación compatible con la herramienta, los scripts de prueba se crean registrando las acciones que un usuario realiza manualmente en la interfaz del sistema y luego guardando las acciones como instrucciones de prueba.

En la Figura 1 se muestra la secuencia de uso de un "*Linear Scripting*" framework:

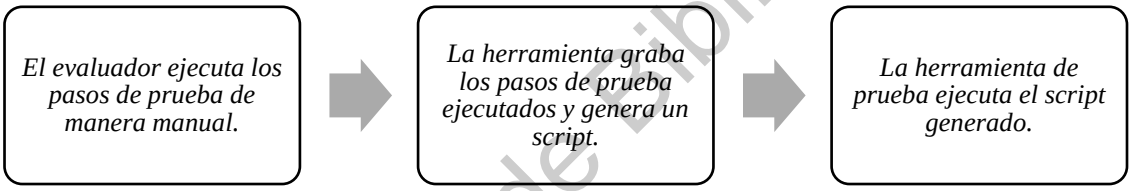

Figura 1. Secuencia de uso de un "Linear Scripting" framework. Fuente: Elaboración propia.

- Data-Driven: Se basa en almacenar los datos de prueba en un archivo externo en lugar de tenerlos estrechamente incorporados en el script de prueba, al momento de la ejecución de pruebas los datos se leen del archivo externo en lugar de tomarlos directamente del propio script, esto permite que tanto los datos de entrada como los resultados esperados se puedan almacenar juntos y por separado del propio script, así mismo es importante que el archivo de datos externo se sincronice con el script de control, esto significa que si se aplica algún cambio al formato del archivo de datos, el script de control se debe actualizar para que corresponda con él.

En la Figura 2 se muestra la arquitectura básica de un "*Data-Driven*" framework:

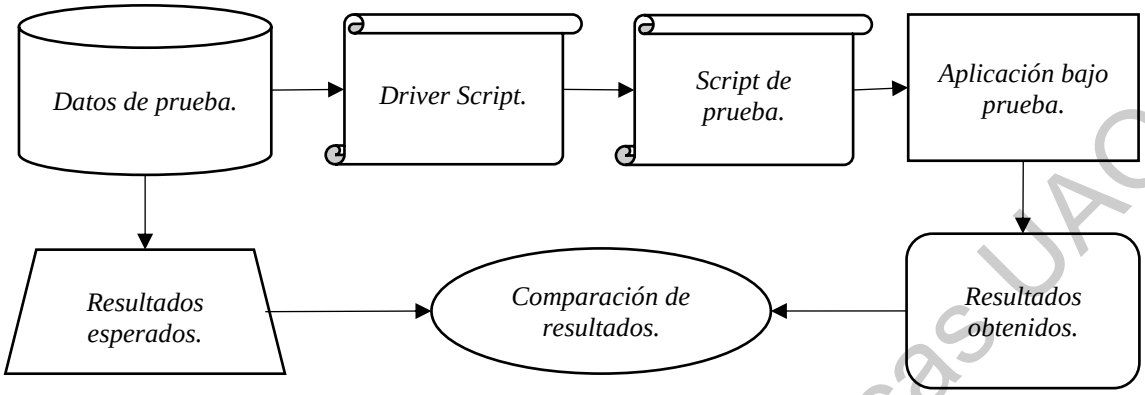

Figura 2. Arquitectura básica de un "Data-Driven" framework. Fuente: Elaboración propia.

- Keyword-Driven: Se basa en la creación de scripts de prueba a través del uso de palabras clave, este enfoque no solo separa los datos de prueba al igual que en el framework "*Data-Driven*", sino que también usa palabras clave especiales para realizar funciones comerciales, así mismo se pueden crear una gran cantidad de scripts de prueba simplemente usando las palabras clave predefinidas, todo lo que necesita saber el evaluador es qué palabras clave están disponibles actualmente para aplicarse en el script y cuáles son los datos que espera cada palabra clave.

En la Figura 3 se muestra la arquitectura básica de un "Keyword-*Driven*" framework:

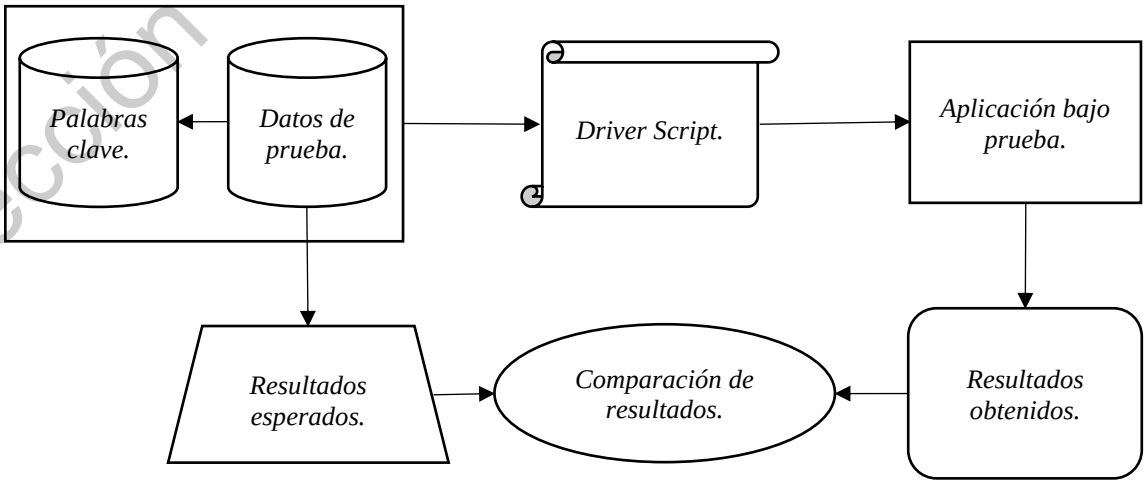

Figura 3. Arquitectura básica de un "Keyword-Driven" framework. Fuente: Elaboración propia.

3. METODOLOGÍA

3.1 Documentación

Para la realización de este trabajo fue necesario recabar información actual acerca de cualquier herramienta que comparta similitudes con la propuesta en este documento, esto para poder tomar en cuenta las lecciones aprendidas, así como para utilizarlos de guía y simplificar el desarrollo de este.

En esta sección de la metodología se omitirá el contenido de las fuentes usadas, dado que no es el propósito mostrar la información recolectada sino el hecho de la importancia de esta, ya que fue de gran utilidad para cumplir los objetivos de este proyecto, toda la información que fue utilizada se encuentra debidamente documentada en la sección de "*Referencias*" para su futura consulta si así fuese necesario.

3.2 Análisis

Resultado de la investigación realizada en la etapa de "*Documentación*", se obtuvo información relacionada con algunas propuestas de herramientas gráficas de automatización de pruebas para aplicaciones web.

En la mayoría de los casos se hace uso del "*Keyword-Driven*" framework, dado que por su naturaleza es el más recomendable para disminuir el uso de códigos de programación especializados al momento de la implementación de pruebas automatizadas.

Este tipo de framework hace uso de un archivo externo que contiene las palabras clave o acciones a realizar sobre la aplicación bajo prueba, esto se puede considerar como una representación de los casos de prueba manual, lo cual se considera una ventaja al momento de la implementación y el mantenimiento, ya que no se requiere de conocimiento especializado para dichas actividades.

Sin embargo, aunque este tipo de archivos ya se considere como una interfaz gráfica y sea más sencilla su utilización que los métodos de scripting convencional, este también tiene desventajas al momento de hacer uso de estructuras de control tales como los ciclos "*For*" o "*While*" dado que en algún escenario de prueba en el que se necesite hacer uso de estos, tendrían que ser implementados en el "*Driver Script*", el cual su función principal es la de interpretar las palabras clave así como los parámetros de prueba que estos usen y reflejarlos en manera de acciones sobre la aplicación bajo prueba.

Para hacer uso de las ventajas y corregir las deficiencias, así como también para cumplir los objetivos de esta investigación, fue necesario hacer uso de herramientas que permitieron implementar una solución gráfica al momento de la creación de la prueba automatizada, así como al momento de hacer uso de estructuras de control o resolución de problemas que impliquen el uso de lógica programable.

Las herramientas utilizadas para la elaboración de este proyecto fueron:

- Selenium: Es un proyecto que encapsula una variedad de herramientas y bibliotecas que permiten la automatización del navegador web.
- LabVIEW: Es un software que suministra un entorno de desarrollo gráfico para el diseño de aplicaciones de ingeniería de adquisición de datos, así como el análisis de medidas y la presentación de información gracias a su lenguaje de programación sin la complejidad de otras herramientas de desarrollo.
- TestStand: Es un software de gestión de pruebas y sistemas de validación, permite el desarrollo de secuencias de prueba automatizadas que integran módulos de código escritos en cualquier lenguaje de programación, estas secuencias especifican el flujo de ejecución, así como la presentación de resultados.

De las herramientas seleccionadas, las últimas dos tienen desventajas relacionadas con los altos costos en sus licencias de uso, lo cual hace que no sean una opción factible para empresas que no puedan cubrir ese tipo de gastos, pero a la vez es una ventaja por su alto reconocimiento a nivel mundial.

3.3 Diseño

Una vez identificadas las herramientas, así como el modelo de framework a utilizar, se realizó el diseño de la arquitectura de automatización, esta representa de manera visual el trabajo hecho en este documento.

La arquitectura consta de dos diferentes capas, las cuales son:

- Definición y adaptación: Se hace uso de un conjunto predefinido de palabras clave, las cuales representan las diferentes funciones disponibles para interactuar de manera directa con la aplicación web bajo prueba, cada palabra clave representa una acción a ser realizada, por lo que estas necesitan ser configuradas a través de su propia interfaz gráfica, esta tiene el propósito de configurar los parámetros necesarios para la utilización de las librerías de automatización así como el de guiar y facilitar el llenado de los mismos al implementador.
- Ejecución: Se ejecutan de manera secuencial las palabras clave utilizadas en la etapa de "*Definición y Adaptación*", generando el escenario de prueba deseado y registrando las acciones realizadas sobre la aplicación bajo prueba a manera de evidencia, una vez terminada la ejecución se genera un reporte de resultados que contiene el estatus de las pruebas realizadas.

En la Figura 4 se muestra el diagrama correspondiente a la arquitectura de automatización:

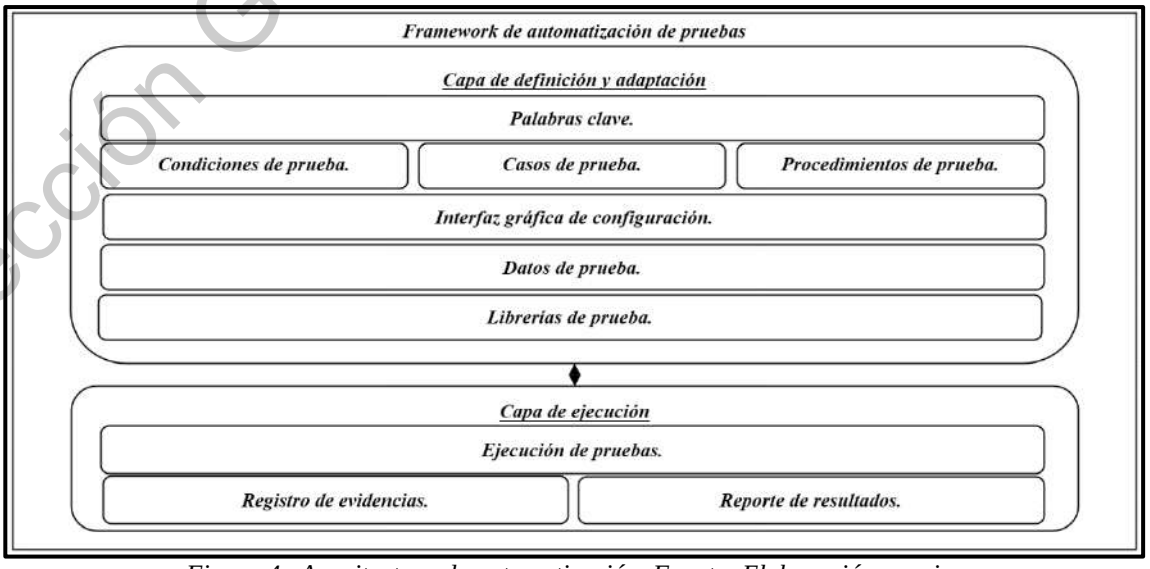

Figura 4. Arquitectura de automatización. Fuente: Elaboración propia.

3.4 Desarrollo

Una vez realizado el diseño, fue necesario convertirlo a un modelo real a través de la combinación de las herramientas previamente mencionadas, en esta etapa de la metodología se describe a manera de pasos el procedimiento requerido para su reproducción si así fuese necesario.

Selenium WebDriver mediante TestStand

Para poder llevar al cabo esta herramienta, fue necesario tener acceso a todas las funciones que tiene disponible "*Selenium WebDriver*", la cual nos permite interactuar de manera directa con la aplicación web bajo prueba, al estar está disponible para diferentes lenguajes de programación, redujo considerablemente el posible número de opciones a utilizar, dado que se tiene que seleccionar alguna que fuera compatible con los actuadores de "*TestStand*", tal fue el caso del "*WebDriver.dll*" que es la versión disponible para C#.

Una vez teniendo disponible el archivo "*WebDriver.dll*" es necesario hacer uso del actuador .NET de "*TestStand*" para poder acceder a las características de esta.

En la Figura 5 se muestra el proceso anteriormente descrito:

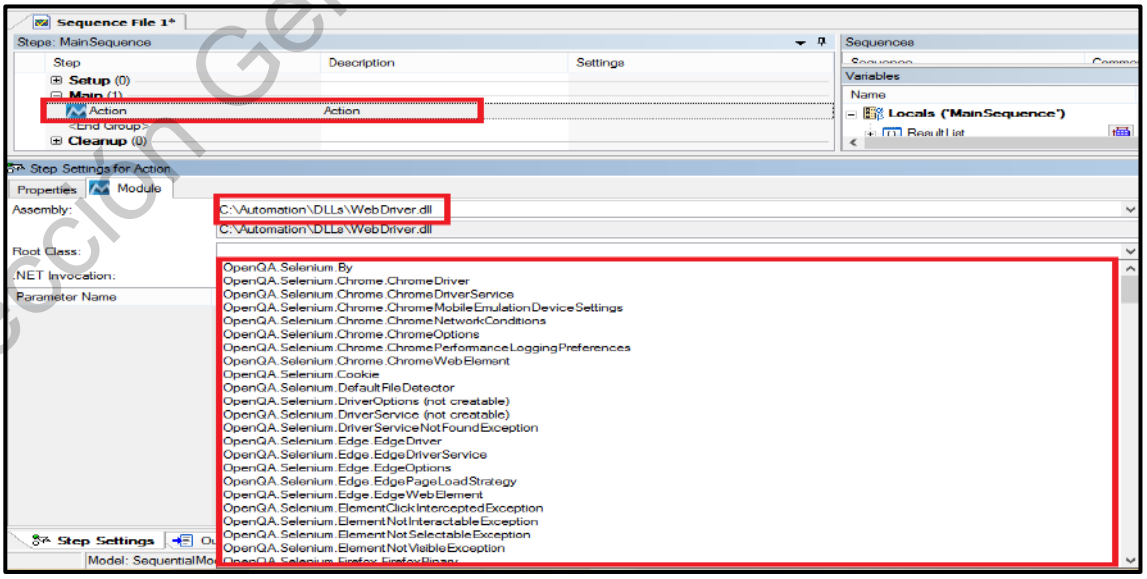

Figura 5. Acceso a "Selenium WebDriver" mediante "TestStand". Fuente: Elaboración propia.

Como se observa en la figura 5, el actuador .NET tiene acceso a las funciones de "*Selenium WebDriver*", sin embargo, esto no significa que sea el mismo procedimiento para realizar un escenario de prueba automatizado a como si se estuviera realizando por el método convencional, ya que como su contrapuesto, es necesario hacer uso de parámetros para poder ejecutar una función, así como el llevar un orden secuencial y el concatenamiento de funciones.

En la Figura 6 se muestra un ejemplo de los parámetros solicitados por una función de "*Selenium WebDriver*" mediante "*TestStand"*.

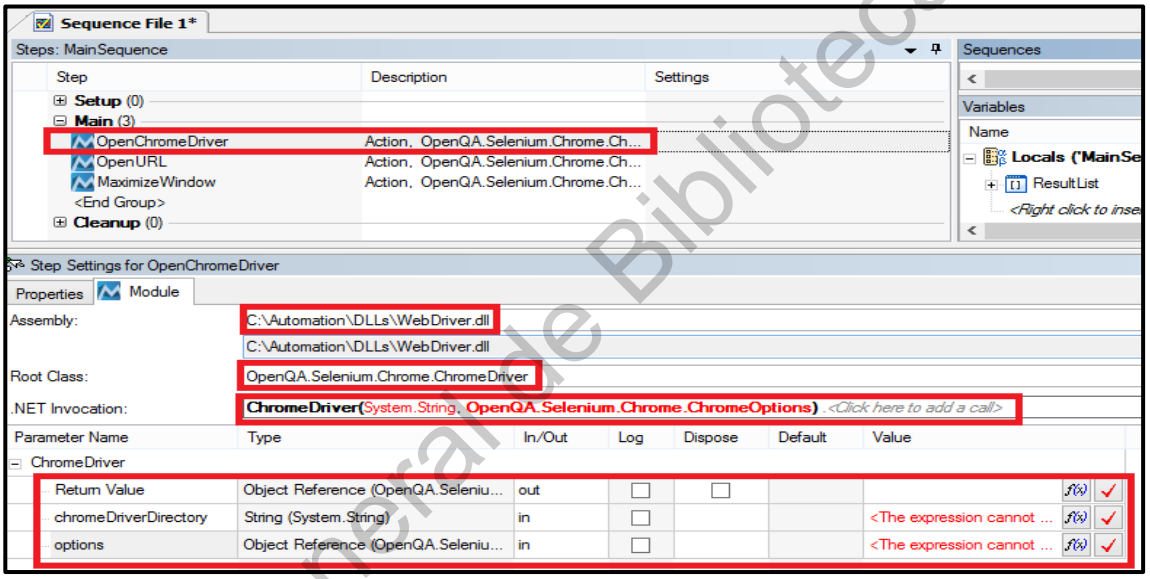

Figura 6. Parámetros solicitados por una función de "Selenium WebDriver" mediante "TestStand". Fuente: Elaboración propia.

Para poder entender en su totalidad la combinación de "*Selenium WebDriver*" con "*TestStand"* es necesario ejemplificar un escenario sencillo en donde se abre un explorador web a través de diferentes actuadores .NET que estarán haciendo uso de las funciones necesarias para llevar al cabo este ejemplo.

Para la realización de este ejemplo, así como de esta herramienta es necesario hacer usos de los drivers para el explorador web deseado, en este caso se hace uso del "*ChromeDriver.exe*".

En la Figura 7 se muestra el primer actuador "*OpenChromeDriver*" que tiene como objetivo el inicializar el controlador necesario para poder interactuar con el navegador bajo prueba, en esta se puede observar cómo se guarda un objeto de referencia en la variable "*Driver*" que será ocupada posteriormente para heredar la inicialización de este hacia los demás actuadores.

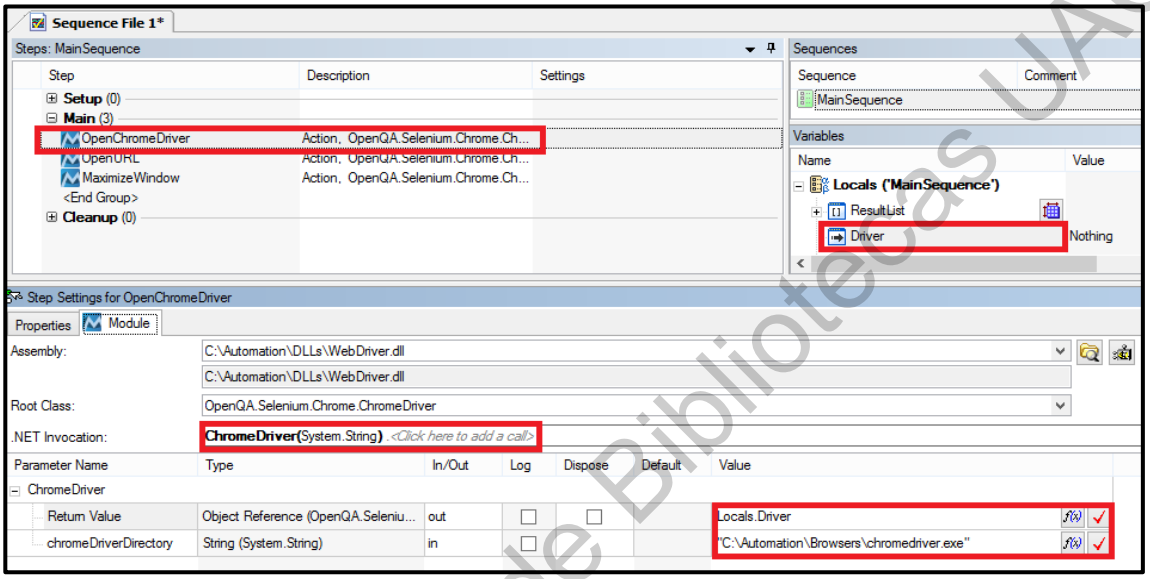

Figura 7. Inicialización del "ChromeDriver". Fuente: Elaboración propia.

En la Figura 8 se muestra como se hace uso del "*Driver*" creado anteriormente en el actuador "*OpenURL*" para poder utilizar la función "*Url*" que permite acceder hacia la página web bajo prueba.

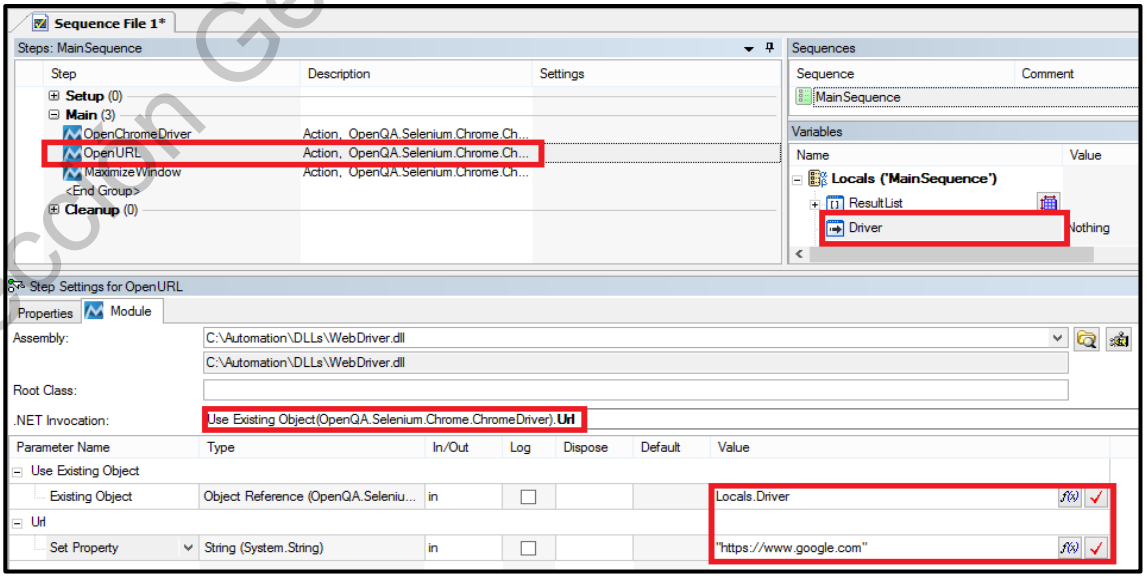

Figura 8. Selección de la "Url" bajo prueba. Fuente: Elaboración propia.

En la Figura 9 se muestra al tercer actuador "*MaximizeWindow*" que de la misma forma hace uso del "*Driver*" creado anteriormente para acceder a la función "*Maximize*" el cual su objetivo es el de maximizar la ventana del explorador a un tamaño adecuado para la realización de las pruebas.

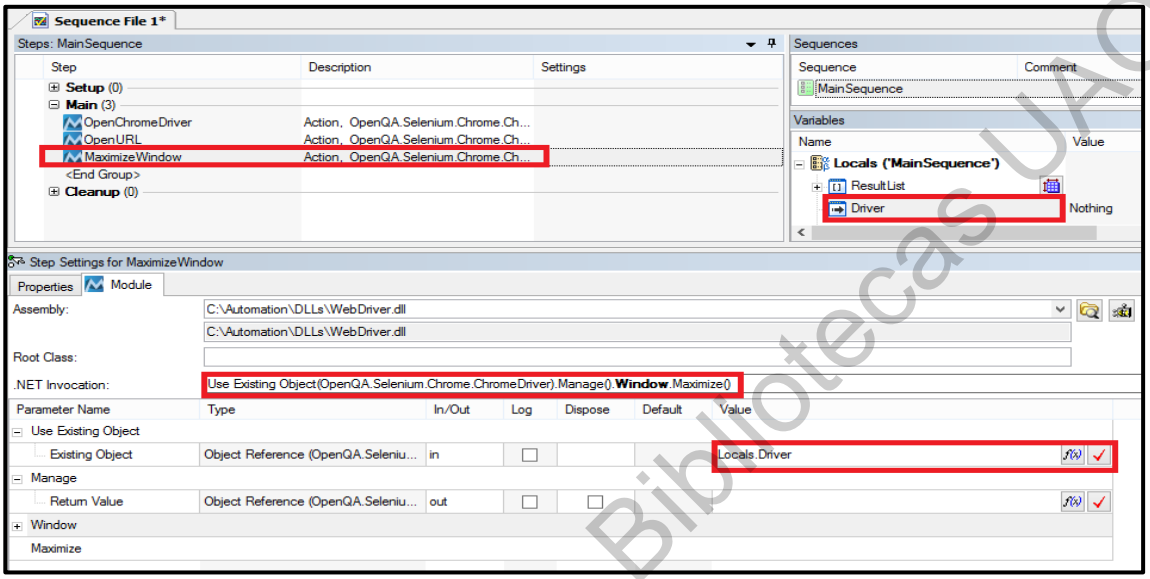

Figura 9. Maximización de la ventana bajo prueba. Fuente: Elaboración propia.

Como se muestra en el ejemplo anterior, la forma en que se hace uso de las funciones de "*Selenium WebDriver*" mediante "*TestStand"* presenta una ventaja al momento de la implementación debido a su entorno gráfico con respecto al método convencional, sin embargo, aún se requiere de conocimiento puntual en las librerías a utilizar, así como los parámetros que estas requieren para su funcionamiento.

Así mismo, esto representa un avance fundamental para la implementación de la arquitectura de automatización, dado que era primordial para la realización de las palabras clave, así como de las secuencias de prueba que forman parte de ellas.

Una vez encontrada la manera de interactuar con la aplicación web mediante la herramienta de gestión de pruebas, es necesario la implementación de secuencias de prueba que representen el conjunto de acciones correspondientes a cada palabra clave, estas reciben únicamente los parámetros necesarios para su funcionamiento.

Secuencia de prueba

Las secuencias de prueba representan los pasos internos que una palabra clave necesita para realizar una acción determinada sobre la aplicación bajo prueba, estos suelen ser repetitivos como para ser utilizados de manera directa al momento de la implementación de las pruebas automatizadas, por esta razón son agrupados en módulos reusables que únicamente solicitan parámetros específicos para su utilización.

En la Figura 10 se muestra un ejemplo de una secuencia de prueba haciendo uso de "*Selenium WebDriver*" mediante "*TestStand"*.

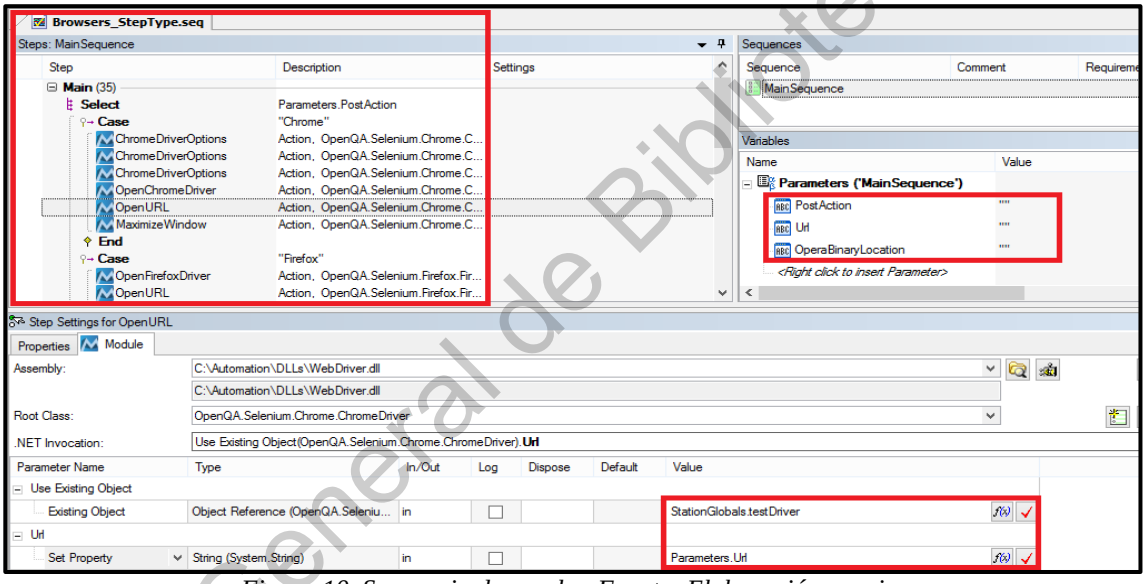

Figura 10. Secuencia de prueba. Fuente: Elaboración propia.

Como se observa en la figura 8, las secuencias de prueba pueden englobar más de una funcionalidad, dado que estas hacen uso de estructuras de control que permiten el uso de lógica programable para la toma de decisiones.

Las estructuras de control son de gran utilidad al momento de la implementación de las secuencias de prueba, dado que al hacer uso de estas se puede tener una mayor cantidad de funciones en una sola palabra clave, lo cual permite una reducción considerable en el número de estas, generando menor ruido visual al momento de la implementación de las pruebas automatizadas.

En la Figura 11 se muestran las diferentes estructuras de control disponibles en "*TestStand"* para la implementación de pruebas automatizadas, así como de secuencias de prueba.

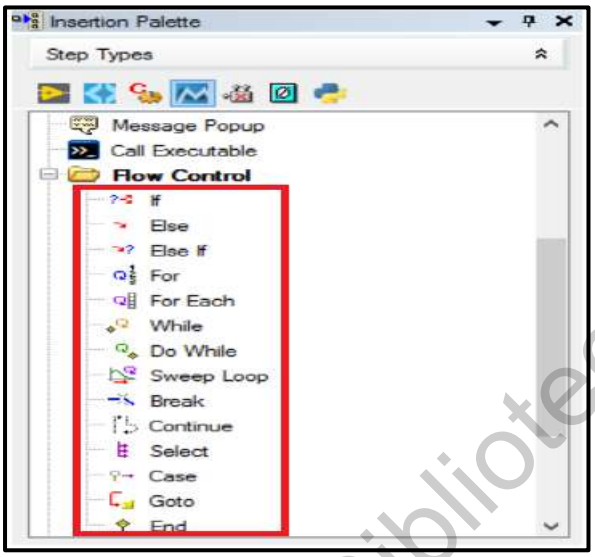

Figura 11. Estructuras de control disponibles en "TestStand". Fuente: Elaboración propia.

En la Figura 12 se muestra el actuador correspondiente a la estructura de control "*Select*", la cual tiene el propósito de almacenar en módulos los actuadores .NET correspondientes a las instrucciones de prueba, así mismo la ejecución de estos módulos depende directamente de una variable de control, la cual indica con un parámetro específico el bloque de prueba que será ejecutado.

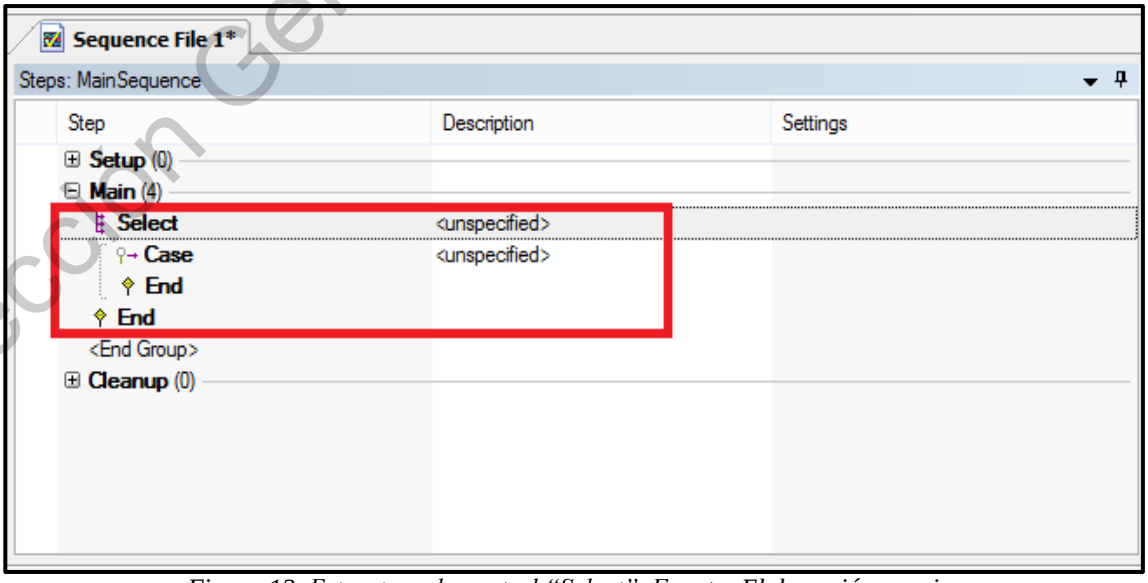

Figura 12. Estructura de control "Select". Fuente: Elaboración propia.

Para aclarar lo comentado anteriormente, es necesario retomar el ejemplo en donde se abre un explorador "*Chrome*" mediante "*TesStand*".

Tomando como base el escenario anterior, se ejemplifica una secuencia de prueba que permite la selección de dos diferentes opciones de explorador web, por lo que es necesario hacer uso de las estructuras de control "*Select*" y "*Case*", así como del conjunto de actuadores .NET correspondientes a cada explorador.

Para el propósito de este ejemplo, se utiliza "*Firefox*" como la segunda opción de explorador web, lo cual significa que se tiene que hacer uso del "*geckodriver.exe*" para poder interactuar con este.

En la Figura 13 se muestra lo comentado anteriormente.

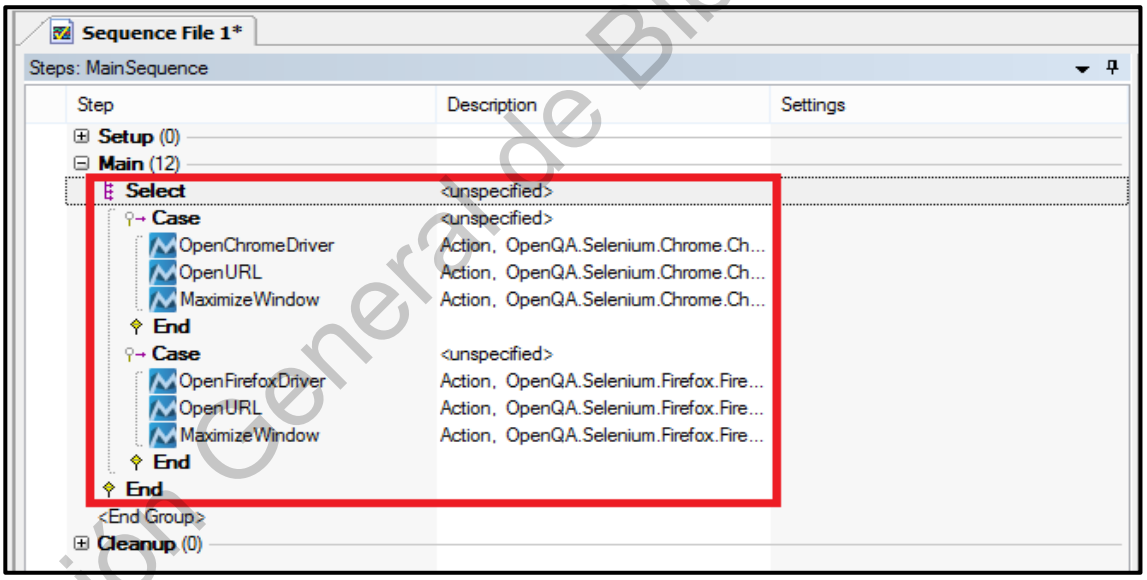

Figura 13. Secuencia de prueba - "Explorador Web". Fuente: Elaboración propia.

Como se puede observar en la figura 13, se utilizan dos estructuras de control tipo "*Case*", estas tienen el propósito de agrupar en bloques los actuadores .NET correspondientes a cada uno de los exploradores web, estos necesitan directamente de un identificador para que puedan ser utilizados al momento de la ejecución de la secuencia de prueba, este identificador necesita corresponder al módulo de prueba asociado.

En la Figura 14 se muestra la configuración del identificador "*Chrome*" correspondiente al primer bloque de prueba, este identificador tiene un valor predefinido y es comparado con el valor seleccionado por el usuario al momento de la configuración de la secuencia de prueba, si este valor es igual al predefinido entonces se ejecuta el módulo de prueba correspondiente al identificador.

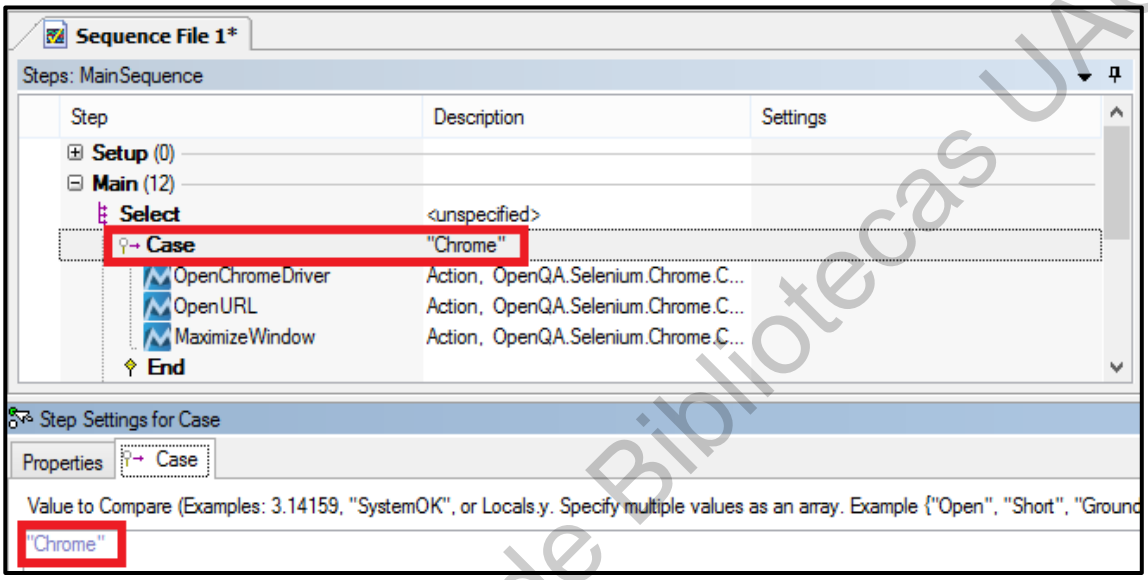

Figura 14. Configuración del identificador "Chrome". Fuente: Elaboración propia.

En la Figura 15 se muestra la configuración del identificador "*Firefox*", que al igual que el anterior este tiene el propósito de ejecutar de manera controlada el módulo de prueba correspondiente al valor predefinido.

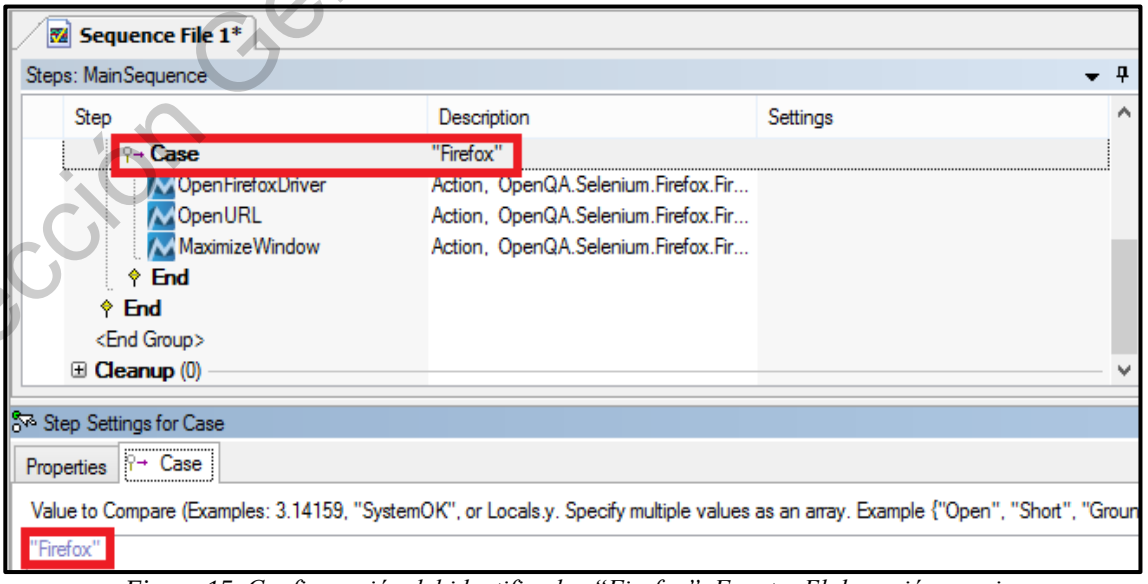

Figura 15. Configuración del identificador "Firefox". Fuente: Elaboración propia.

Una vez realizada la configuración de las estructuras de control "*Case*", es necesario hacer lo mismo con la variable de control encargada de seleccionar el bloque de prueba a ser ejecutado, esta variable está localizada directamente en la estructura de control "*Select*".

Así mismo, la ubicación de esta variable debe estar predefinida en la sección de "*Parámetros*", dado que el valor de esta es dado por el usuario que está haciendo uso de la palabra clave al momento de la implementación de las pruebas automatizadas.

Lo comentado anteriormente se puede observar en la Figura 16.

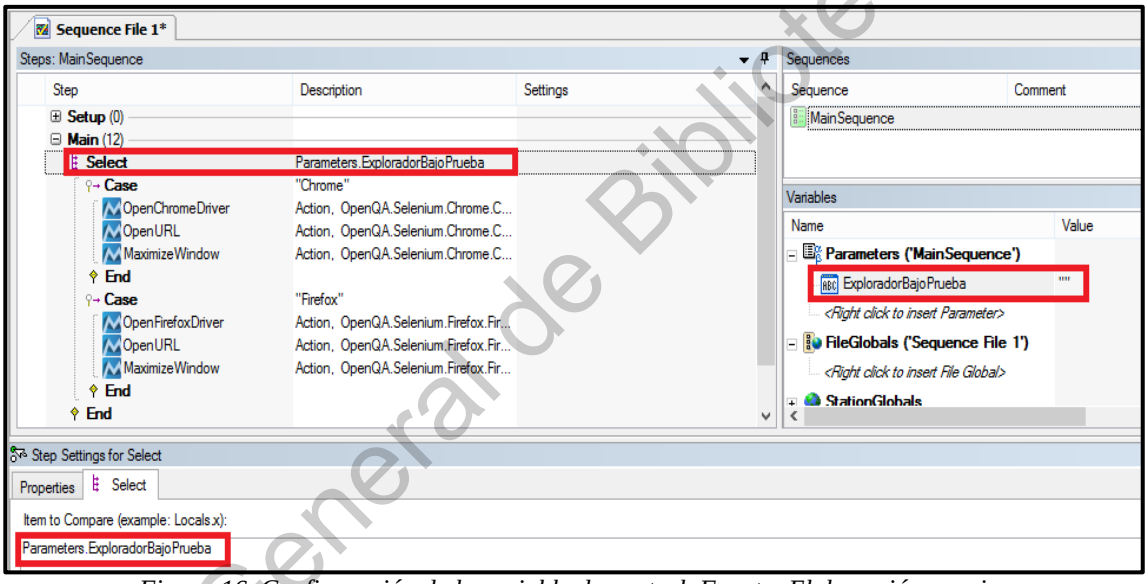

Figura 16. Configuración de la variable de control. Fuente: Elaboración propia.

Como se mencionó anteriormente, la configuración de las variables es indispensable al momento de la implementación de las secuencias de prueba, estas tienen que estar previamente asignadas según su propósito, dado que dependiendo de esto se define de manera directa su localización en "*TestStand*".

Así mismo las variables que tienen interacción directa con el usuario final deben de estar localizadas en la sección de "*Parámetros*", el resto de estas pueden ser ubicadas en el apartado de variables locales.

De la misma forma en que se configura la variable de control, es necesario establecer el parámetro correspondiente a la "*Url"* bajo prueba, esta variable es única independientemente del explorador que se esté utilizando en ese momento, sin embargo, esta debe ser indicada en cada módulo de prueba.

En la Figura 17 se muestra la configuración de la variable "*Url*" en el módulo de prueba "*Chrome*".

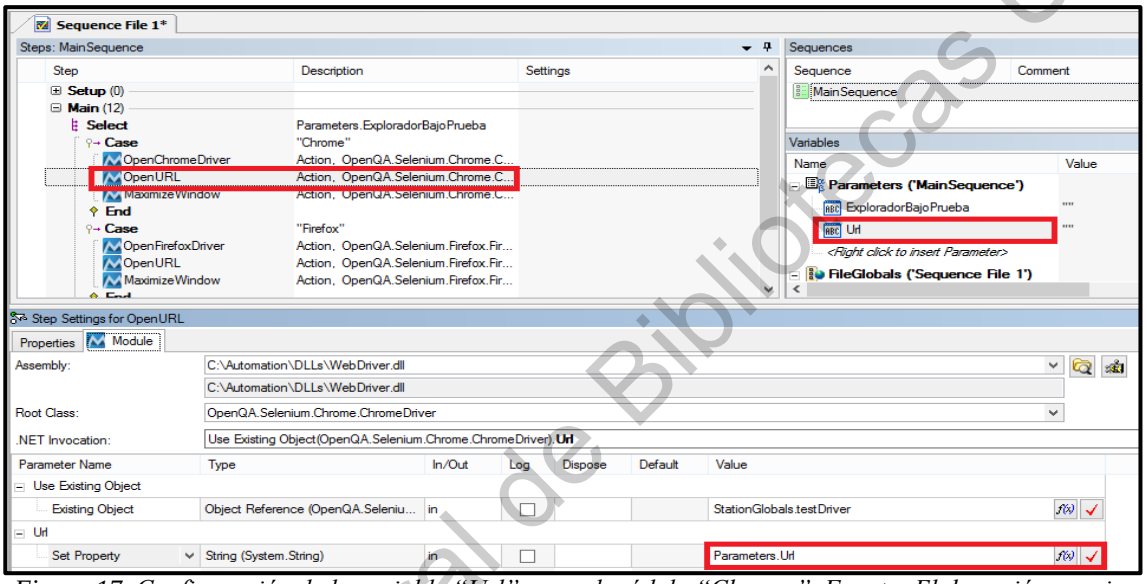

Figura 17. Configuración de la variable "Url" para el módulo "Chrome". Fuente: Elaboración propia.

En la Figura 18 se muestra la configuración de la variable "*Url*" en el módulo de prueba "*Firefox*".

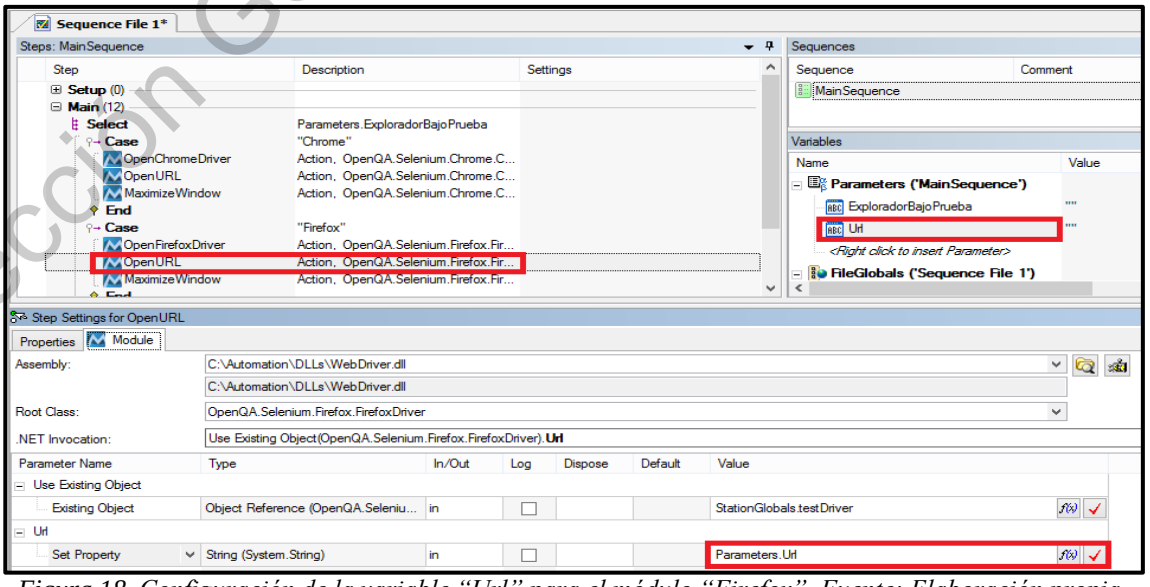

Figura 18. Configuración de la variable "Url" para el módulo "Firefox". Fuente: Elaboración propia.

Una vez terminado el desarrollo de las secuencias de prueba, es necesario implementar las interfaces gráficas correspondientes a cada palabra clave, estas tienen el propósito de guiar al usuario de manera directa al momento de la implementación de las pruebas automatizadas, y de esta forma reducir el factor de error humano que se pudiera tener si se realizara la automatización a través del método convencional.

Interfaz Gráfica

El propósito principal de toda interfaz gráfica es el de establecer conexión física y funcional entre dos aparatos, en este caso se busca instaurar comunicación entre el implementador de la prueba automatizada y la secuencia de prueba, dado que esta última requiere de parámetros específicos para que pueda funcionar, y estos tienen que ser establecidos por el propio implementador.

En la Figura 19 se muestra un ejemplo de la interfaz gráfica que corresponde a la secuencia de prueba "*Browsers*".

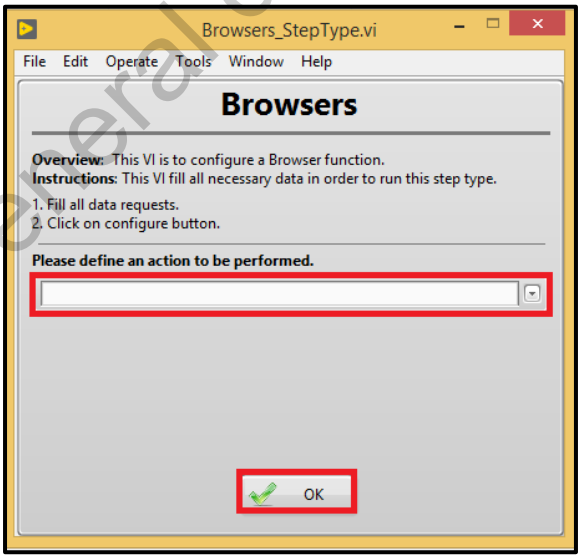

Figura 19. Interfaz Gráfica. Fuente: Elaboración propia.

Así mismo, haciendo uso de la interfaz gráfica se busca establecer un criterio de implementación homogéneo, dado que la misma sirve para guiar al implementador al momento de establecer los parámetros requeridos por la secuencia de prueba.

Como se puede observar en la figura 19, la interfaz gráfica cuenta con dos elementos de configuración, el primero hace referencia a un menú desplegable el cual tiene el propósito de mostrar a manera de lista los exploradores web soportados, y el segundo tiene el objetivo de configurar los parámetros previamente establecidos por el implementador de la prueba automatizada.

En la Figura 20 se muestran las opciones disponibles en el menú desplegable de la interfaz gráfica.

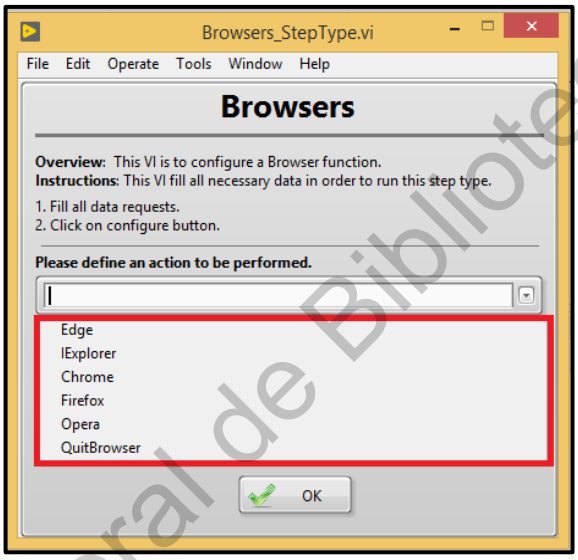

Figura 20. Menú desplegable. Fuente: Elaboración propia.

Como se puede observar en la figura 20, el menú desplegable cuenta con diversas opciones que pueden ser seleccionadas por el implementador de la prueba automatizada, al momento de seleccionar una opción de explorador web, la interfaz gráfica agrega automáticamente un campo de texto en el cual se tiene que indicar la "*Url*" a la que se desea navegar para la realización de las pruebas.

Este tipo de interacción fue posible dado a que todas las interfaces gráficas fueron desarrolladas mediante "*LabVIEW*", este entorno de programación gráfico permite realizar el diseño de la interfaz gráfica, así como el uso de lógica programable para dar solución a este y otros tipos de problemas, además de que este puede interactuar de manera directa con "*TestStand*" lo cual representa una ventaja extra al momento de la integración.

En la Figura 21 se muestra lo comentado anteriormente.

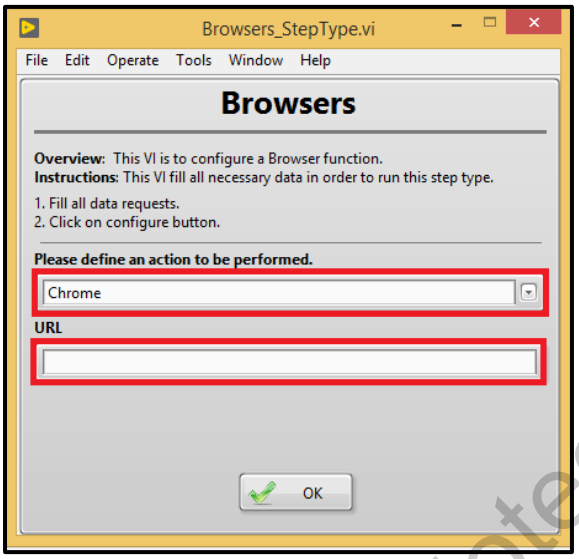

Figura 21. Campo de texto "Url". Fuente: Elaboración propia.

Para entender cómo es que funciona una interfaz gráfica desarrollada mediante "*LabVIEW*", se retoma el ejemplo en donde se implementó una secuencia de prueba para la selección de un explorador web y a esta se le diseña su interfaz gráfica correspondiente.

Este ejemplo se realizará en dos partes, la primera es específicamente el diseño de la interfaz gráfica haciendo uso del "*Panel Frontal*", y la segunda consiste en la implementación de la lógica programable requerida mediante el "*Diagrama de Bloques*".

El propósito del "*Panel Frontal*" es el de mantener una interacción con el usuario, en este caso el implementador de la prueba automatizada, mediante el muestreo de datos o la inyección de estos para realizar cierto proceso.

Así mismo, el propósito del "*Diagrama de Bloques*" es el de definir la funcionalidad que es ejecutada por el programa, esto significa que todo elemento que este definido en el "*Panel Frontal*" tiene que ser utilizado para cumplir el propósito previamente establecido.

En la Figura 22 se puede observar un ejemplo de "*Panel Frontal*" y "*Diagrama de Bloques*" en "*LabVIEW*".

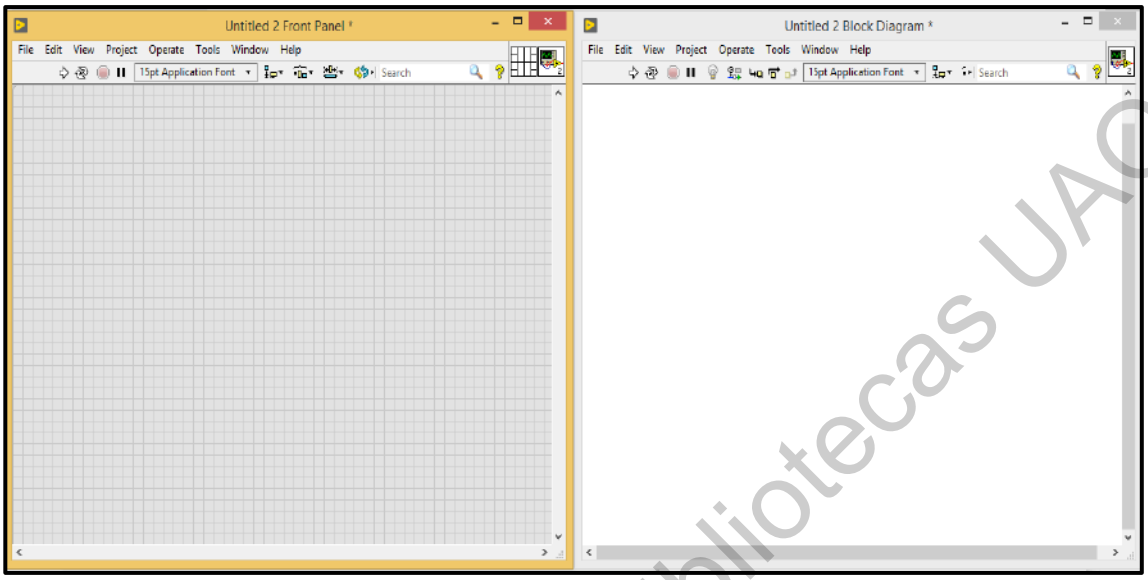

Figura 22. "Panel Frontal" y "Diagrama de Bloques" en "LabVIEW". Fuente: Elaboración propia.

Para empezar con la realización del ejemplo mencionado anteriormente, lo primero que se tiene que hacer es agregar los controles necesarios para cumplir con las características específicas que la interfaz gráfica requiere, estos deben de incluir una lista desplegable, un campo de texto y un botón de configuración.

En la Figura 23 se muestra lo comentado anteriormente.

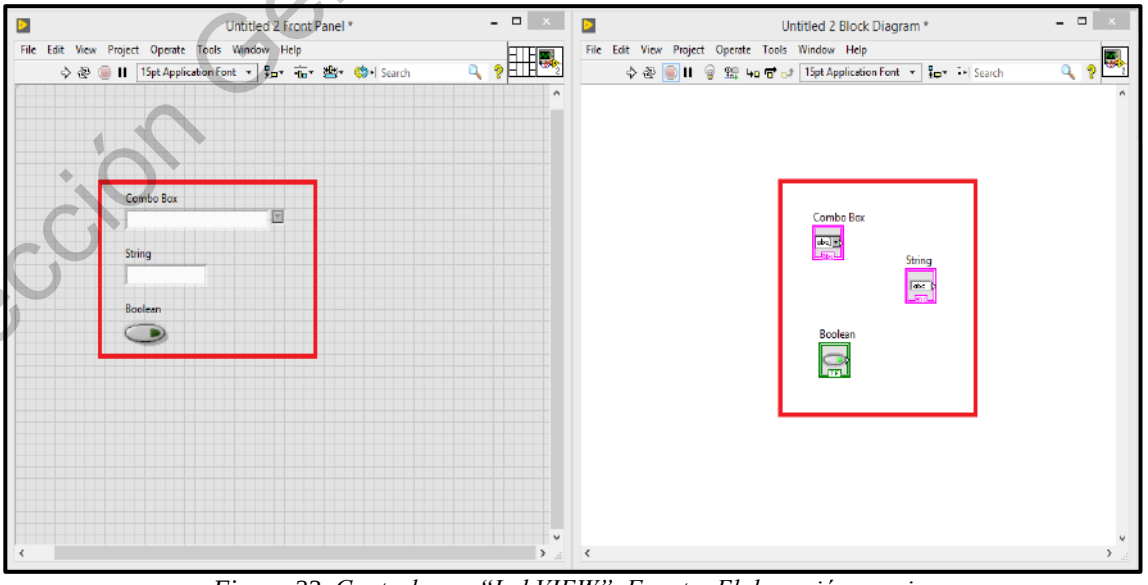

Figura 23. Controles en "LabVIEW". Fuente: Elaboración propia.

Una vez agregados los controles necesarios para cumplir con el propósito de la interfaz gráfica, es necesario ir configurando cada uno de ellos, en este caso se agregan las opciones disponibles de exploradores web en el menú desplegable, para hacer esto es necesario hacer clic derecho sobre el elemento y seleccionar la opción de editar artículo.

Esto despliega un menú de configuración, el cual tiene disponible un botón llamado insertar, en este se da clic las veces necesarias para agregar las opciones correspondientes a los exploradores "*Chrome*" y "*Firefox*".

En la Figura 24 se muestra la configuración de las opciones del menú desplegable.

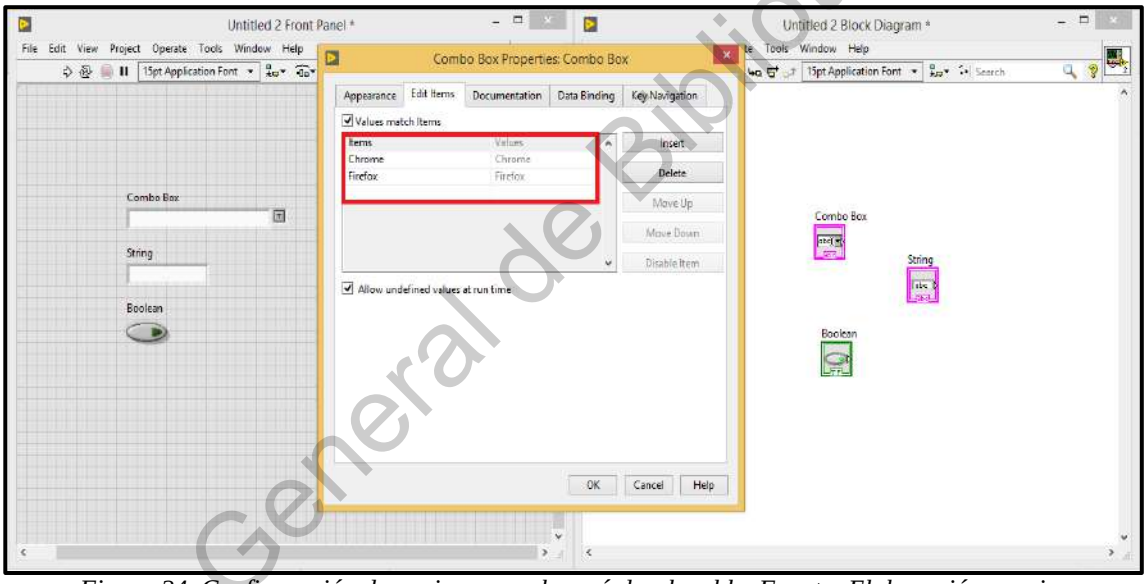

Figura 24. Configuración de opciones en el menú desplegable. Fuente: Elaboración propia.

Una vez agregadas las opciones de exploradores web, es necesario hacer clic en el botón de ok para terminar la configuración de estas, para comprobar que las opciones fueron agregadas correctamente se tiene que dar clic en el menú desplegable para confirmar que estas aparezcan de la manera deseada.

Así mismo, al terminar la configuración de las opciones de explorador web, es necesario cambiar el nombre de los controles para que estos puedan ser identificados de manera sencilla por el implementador de la prueba automatizada.

Lo comentado anteriormente se puede observar en la Figura 25.

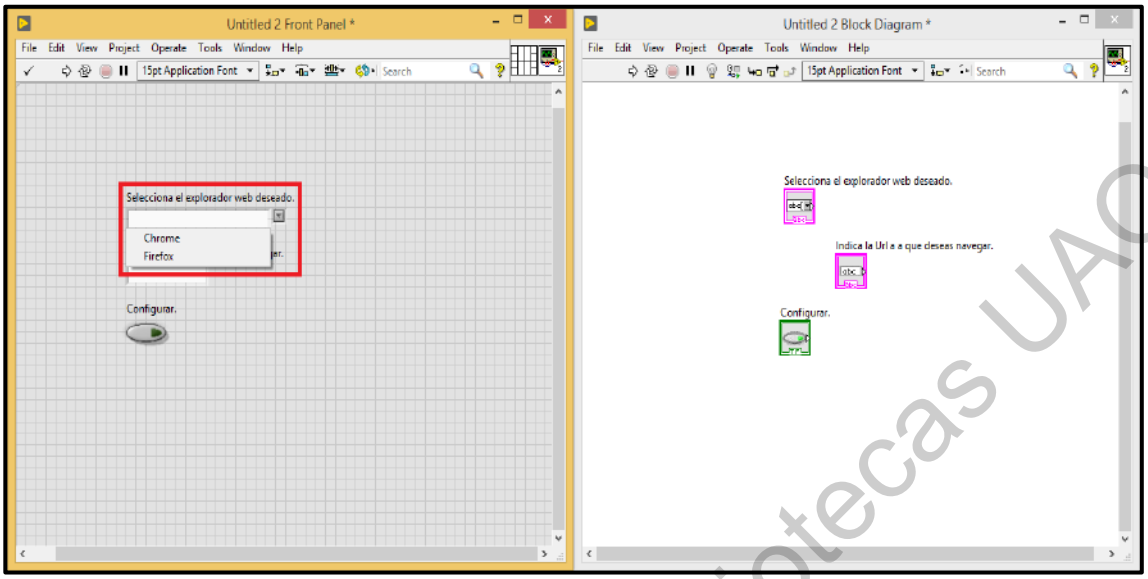

Figura 25. Opciones disponibles en el menú desplegable. Fuente: Elaboración propia.

Cambiado el nombre de los controles, es momento de hacer la configuración del "*Diagrama de Bloques*", en este momento se hace uso de la estructura de control "*While Loop*", la función de esta es ejecutar el programa de manera continua hasta que el implementador de la prueba automatizada presione el botón de configurar, para hacer esto es necesario conectar el botón de configuración con la condición de alto del mismo ciclo.

En la Figura 26 se muestra lo comentado anteriormente.

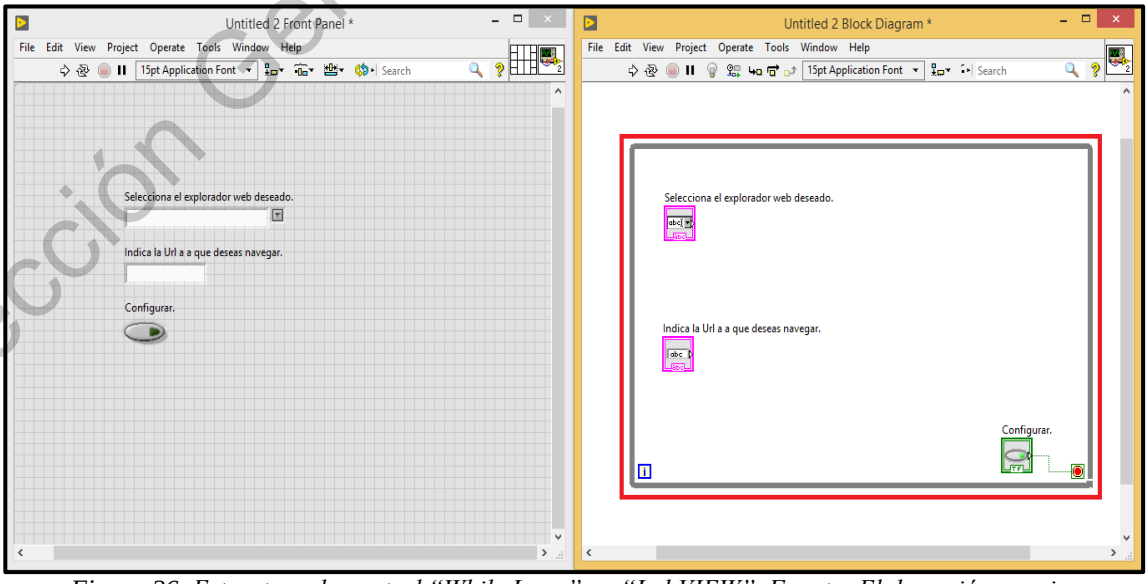

Figura 26. Estructura de control "While Loop" en "LabVIEW". Fuente: Elaboración propia.

Una vez agregado el ciclo "*While Loop*", se necesita configurar el campo de texto que corresponde a la "*Url*" bajo prueba, este campo solo debe ser visible si alguna opción del menú desplegable corresponde de manera directa con un explorador web.

Para hacer esto, se tiene que hacer uso de la estructura de control "*Case Structure*", esta ejecuta únicamente los bloques que se encuentren dentro de ella si una condición en específico se cumple.

Lo comentado anteriormente se puede observar en la Figura 27.

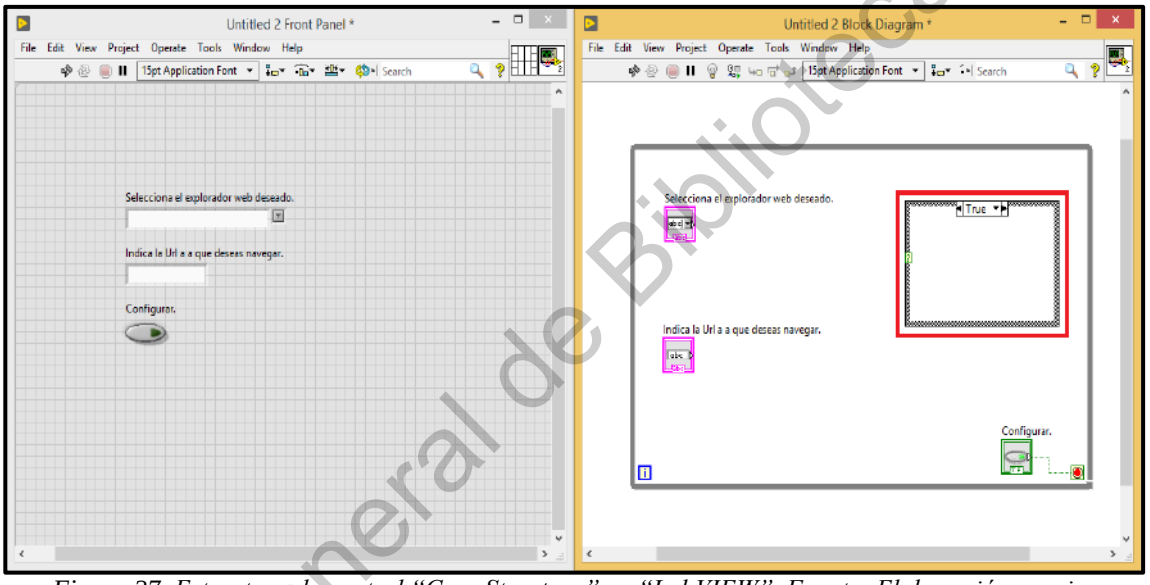

Figura 27. Estructura de control "Case Structure" en "LabVIEW". Fuente: Elaboración propia.

Como se mencionó anteriormente, para que la estructura de control "*Case Structure*" pueda ejecutarse esta necesita que una condición se cumpla, esta condición es dada por las opciones del menú desplegable, las cueles son comparadas con los exploradores web soportados por la secuencia de prueba.

Estas opciones corresponden a los exploradores "*Chrome*" y "*Firefox*", por lo que si alguna de estas es seleccionada por el implementador de la prueba automatizada se despliega el campo de texto que corresponde con la "*Url"*, para que esto sea posible es necesario utilizar atributos y elementos de comparación disponibles en "*LabVIEW*".

En la Figura 28 se muestra la condición de control necesaria para ejecutar la estructura de control "*Case Structure*".

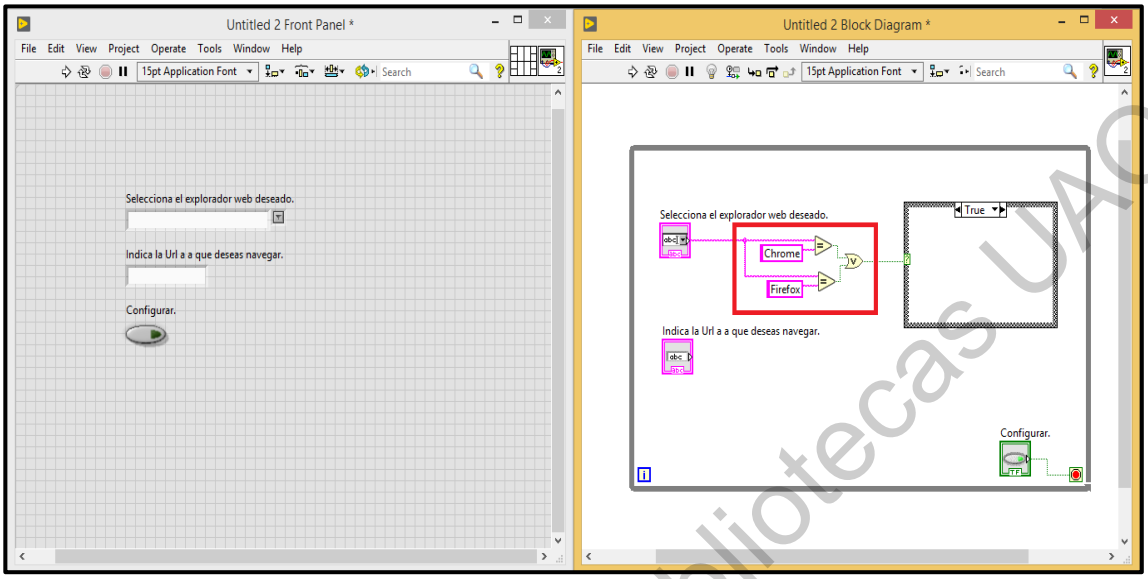

Figura 28. Condición de control - "Case Structure". Fuente: Elaboración propia.

Como se puede observar en la figura 28, cuando la condición de control sea verdadera, esta ejecuta la lógica que este dentro del "*Case Structure*", esta lógica hace referencia a desplegar el campo de texto correspondiente a la "*Url*", para hacer esto es necesario utilizar el atributo correspondiente a la visibilidad.

En la Figura 29 se muestra lo comentado anteriormente.

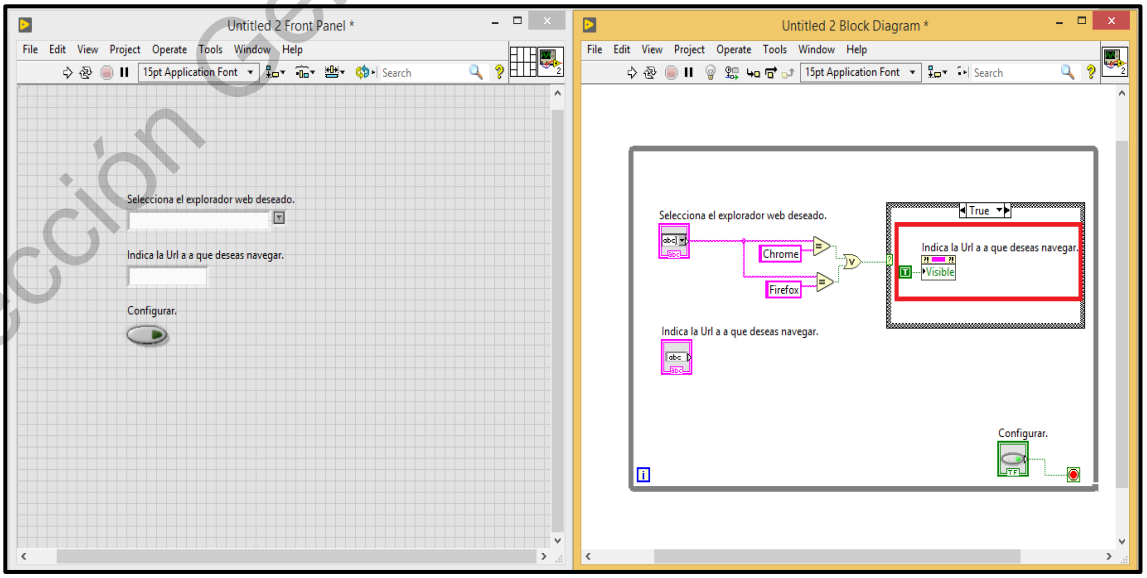

Figura 29. Atributo de visibilidad - Verdadero. Fuente: Elaboración propia.

De la misma forma que el anterior, es necesario configurar el caso falso de la estructura de control "*Case Structure*", en este escenario se tiene que establecer el atributo de visibilidad de tal forma que este no aparezca desplegado en la interfaz gráfica mientras la condición de control no se cumpla.

En la Figura 30 se muestra lo comentado anteriormente.

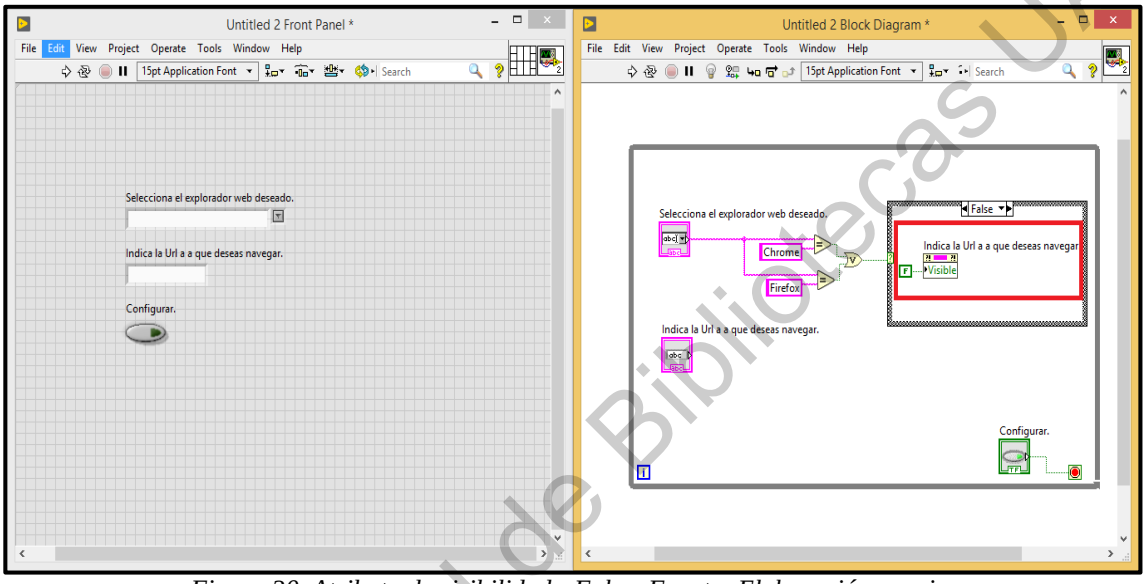

Figura 30. Atributo de visibilidad - Falso. Fuente: Elaboración propia.

Terminada la lógica requerida para el campo de texto correspondiente a la "*Url*", se requiere almacenar los valores introducidos en los controles del "*Panel Frontal*" con algún indicador, estos sirven para traspasar los parámetros hacia la secuencia de prueba de manera directa haciendo uso de la configuración apropiada en "*LabVIWE*".

Los indicadores son requeridos únicamente en los controles que reciben parámetros específicos del implementador de la prueba automatizada, casos como el botón de configuración no requiere de algún indicador debido a que sus estados no son utilizados por la secuencia de prueba.

En este caso es importante mencionar que la secuencia de prueba es la que indica cuantos elementos de la interfaz gráfica requieren de indicadores, dado que estos son los valores que reciben los parámetros de la secuencia de prueba para funcionar.

En la Figura 31 se muestran los indicadores correspondientes a los controles del "*Panel Frontal*".

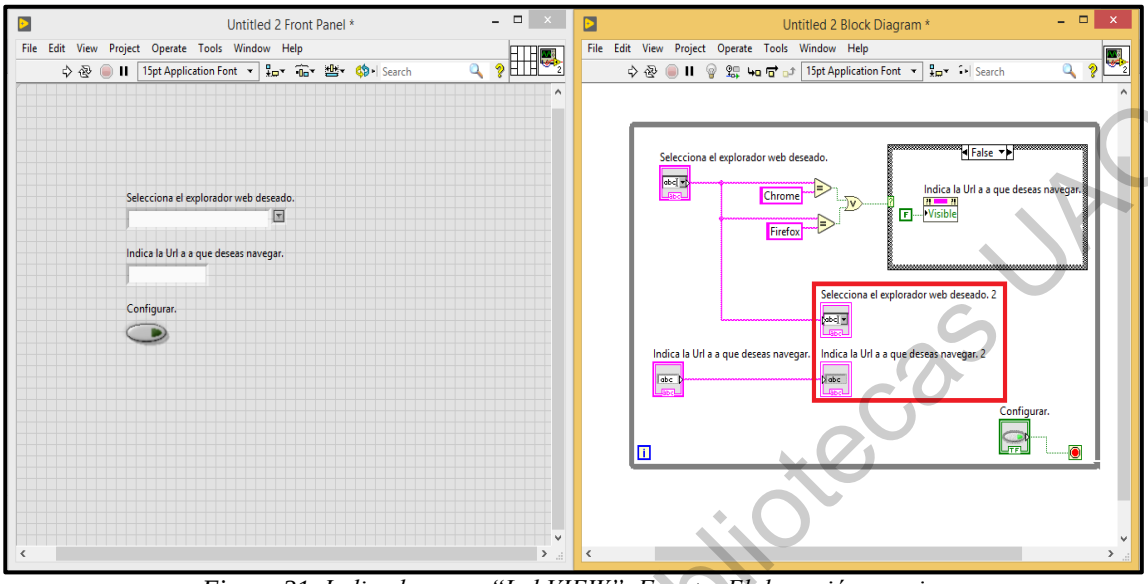

Figura 31. Indicadores en "LabVIEW". Fuente: Elaboración propia.

Una vez terminada la configuración tanto del "*Panel Frontal*" como del "*Diagrama de Bloques*", es necesario configurar los controles e indicadores como entradas y salidas respectivamente, este tipo de configuración se hace directamente en "*LabVIEW*" mediante las terminales.

En la Figura 32 se muestra la configuración de las entradas y salidas mediante "*LabVIEW*".

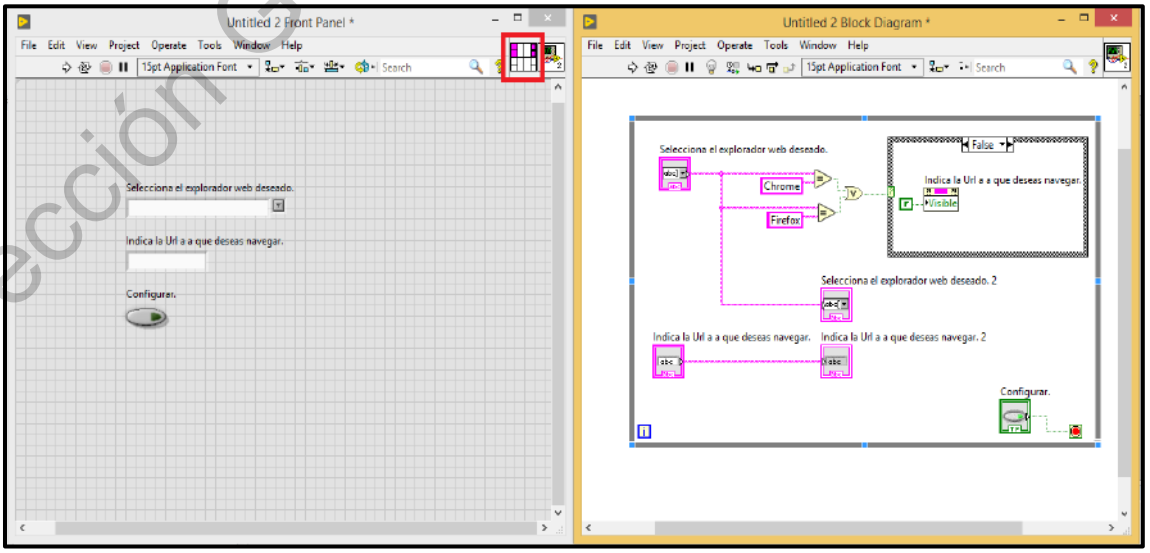

Figura 32. Configuración de entradas y salidas en "LabVIEW". Fuente: Elaboración propia.

Una vez terminada la implementación de las interfaces gráficas correspondientes a las secuencias de prueba, es necesario integrarlas y de esta forma generar las palabras clave que estarán disponibles en el framework gráfico de automatización de pruebas, estas palabras clave son representadas mediante una configuración específica en "*TestStand*" y esta es conocida con el nombre de "*Tipos de paso*".

Tipos de paso

"*TestStand*" incluye una gran variedad de tipos de pasos integrados que operan como bloques de elaboración para las secuencias de prueba, además los tipos de pasos integrados permite a los usuarios desarrollar tipos de pasos personalizados para implementar funcionalidades adicionales.

Los tipos de pasos personalizados habilitan a los usuarios a extender los pasos de prueba existentes de las siguientes formas:

- Configurando los valores de las propiedades del paso de prueba y determinar qué estas pueden ser modificadas por los usuarios.
- Agregando nuevas propiedades al paso de prueba para almacenar datos personalizados, que pueden ser registrados opcionalmente en los resultados de la prueba.
- Definiendo el código que se invocara antes o después de la ejecución del módulo de prueba principal.
- Desarrollando interfaces para habilitar a los usuarios la configuración de las propiedades del paso de prueba personalizado en el momento de la configuración.

Un tipo de paso diseñado apropiadamente puede acelerar el desarrollo de una secuencia de prueba, reducir los esfuerzos de depuración y permitir a los implementadores compartir código estandarizado y consistencia entre múltiples estaciones de prueba y grupos separados, sin embargo, los tipos de pasos personalizados pueden requerir mucho tiempo para su planificación, programación, depuración y implementación.

Antes de comenzar el desarrollo de un tipo de paso personalizado, se debe tener en cuenta enfoques que pueden ser adecuados para una nueva funcionalidad.

No se recomienda crear o modificar un tipo de paso personalizado en los siguientes casos:

- La nueva funcionalidad afecta a muchos de los tipos de pasos, en este escenario se debe considerar usar una devolución de llamada de motor, que se ejecute antes o después de cada paso de prueba.
- Si desea proveer una configuración básica de un paso de prueba existente, pero no se requiere de nuevas funciones o características, en este escenario cree una plantilla de pasos que contenga los cambios necesarios.
- Si desea proporcionar una herramienta para utilizar en el momento de la configuración, en este caso utilice un elemento del menú de herramientas para llamar a su código personalizado.

Se recomienda crear o modificar un tipo de paso en las siguientes situaciones:

- La funcionalidad no es implementable utilizando los tipos de paso integrados.
- La funcionalidad pretende cambiar características que no son configurables en las instancias de los tipos de pasos existente.
- La funcionalidad requiere que las acciones se ejecuten antes o después de llamar al módulo de prueba, la configuración del paso o el análisis de los resultados.
- Si desea facilitar la experiencia del usuario al momento de configurar los pasos de prueba con la ayuda de una interfaz de usuario.
- Si la funcionalidad es compartida con otros grupos, empresas o clientes.

Es importante diferenciar entre los términos paso y tipo de paso, un desarrollador de secuencias hace uso de pasos para la implementación de las pruebas automatizadas mediante algún tipo de paso particular en un archivo de secuencia.

A veces, los tipos de pasos predeterminados no proporcionan las funciones necesarias para una aplicación de prueba, "*TestStand*" permite crear tipos de pasos personalizados para cualquier tipo de aplicación.

Para entender de mejor manera lo que es un tipo de paso, es necesario retomar los ejemplos anteriores donde se implementó una secuencia de prueba y su respectiva interfaz gráfica para la selección de un explorador web, para la realización de este ejemplo es necesario seguir el siguiente procedimiento en "*TestStand*".

Lo primero que se necesita hacer es dar clic en el botón paleta de tipos, esta está ubicado en la barra de herramientas y permite la creación de un tipo de paso particular que está disponible para todos los archivos de secuencia.

En la Figura 33 se muestra la interfaz de configuración de la "*Paleta de tipos*" en "*TestStand*".

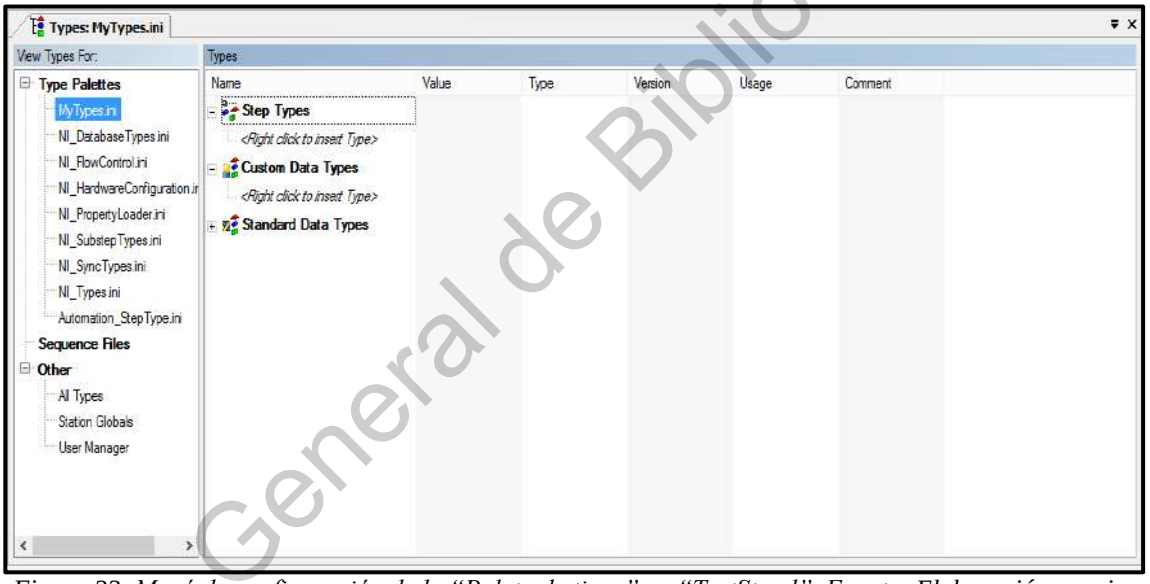

Figura 33. Menú de configuración de la "Paleta de tipos" en "TestStand". Fuente: Elaboración propia.

Una vez en la sección "*Paletas de tipos*", se debe seleccionar la opción "*MyTypes.ini"* y posteriormente en el menú de configuración "*Tipos*" se da clic derecho para insertar un nuevo tipo de paso.

Este nuevo tipo de paso debe ser nombrado de manera sencilla y representativa para facilitar al implementador de la prueba automatizada comprender el propósito que esta tiene.

En la Figura 34 se puede ver un ejemplo de nomenclatura para el tipo de paso agregado.

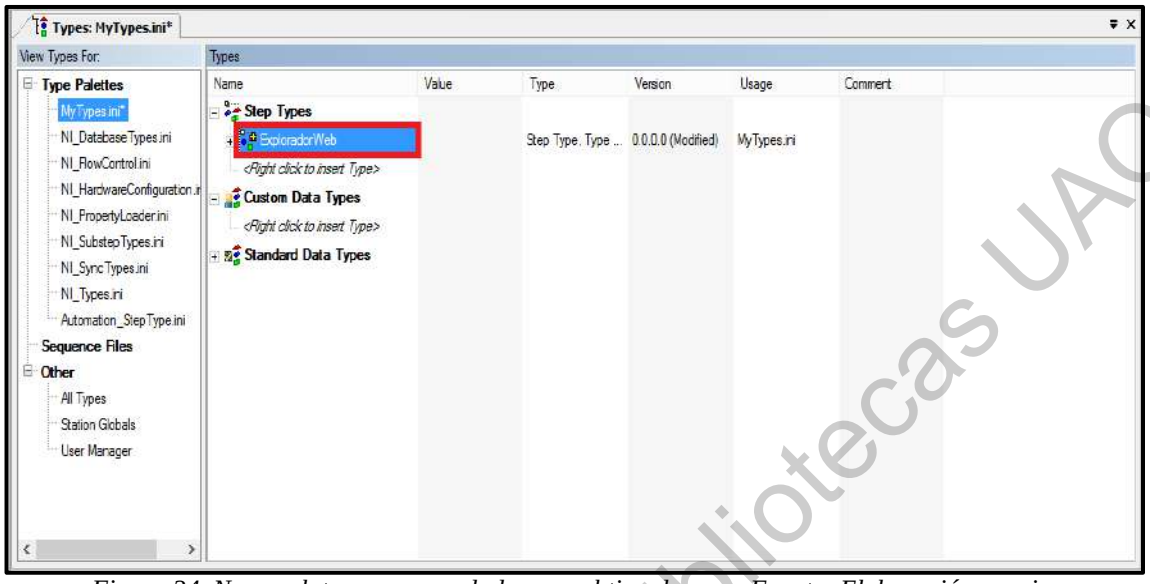

Figura 34. Nomenclatura recomendada para el tipo de paso. Fuente: Elaboración propia.

Una vez agregado el tipo de paso al menú de configuración, es necesario dar clic en el botón de + para expandir las propiedades de este, estos deben ser configurados de acuerdo con los parámetros establecidos por la secuencia de prueba.

En la Figura 35 se muestra el apartado de configuración de parámetros correspondientes al tipo de paso.

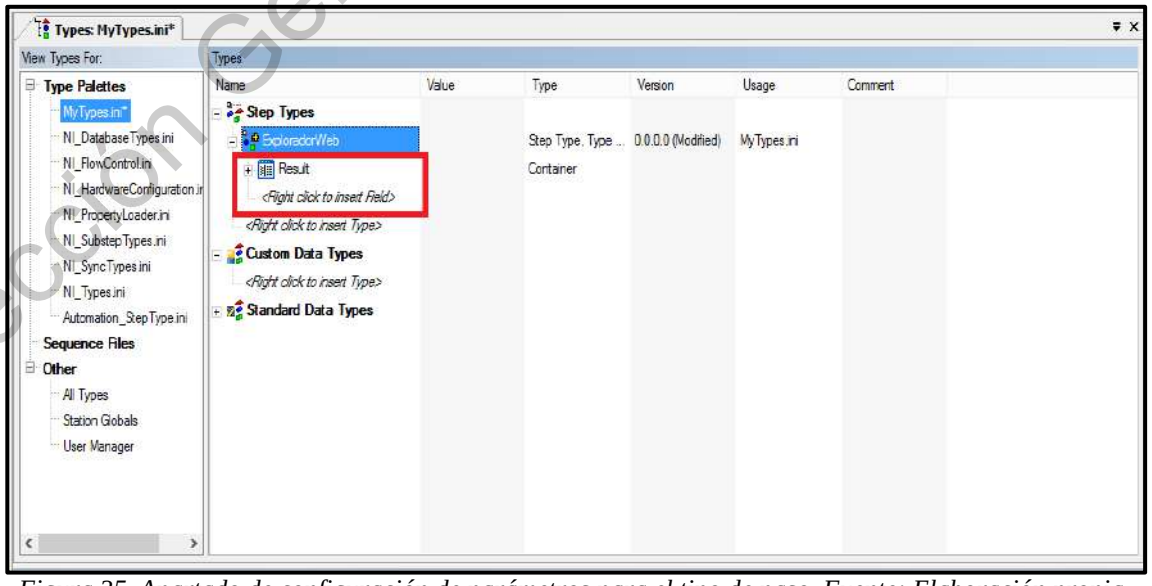

Figura 35. Apartado de configuración de parámetros para el tipo de paso. Fuente: Elaboración propia.

Como se mencionó anteriormente, estos parámetros corresponden a los establecidos por la secuencia de prueba, estos son el explorador web y la "*Url*", por lo tanto, estos tienen que ser establecidos en los parámetros del tipo de paso con el mismo nombre que en la secuencia de prueba.

En la Figura 36 se muestra lo comentado anteriormente.

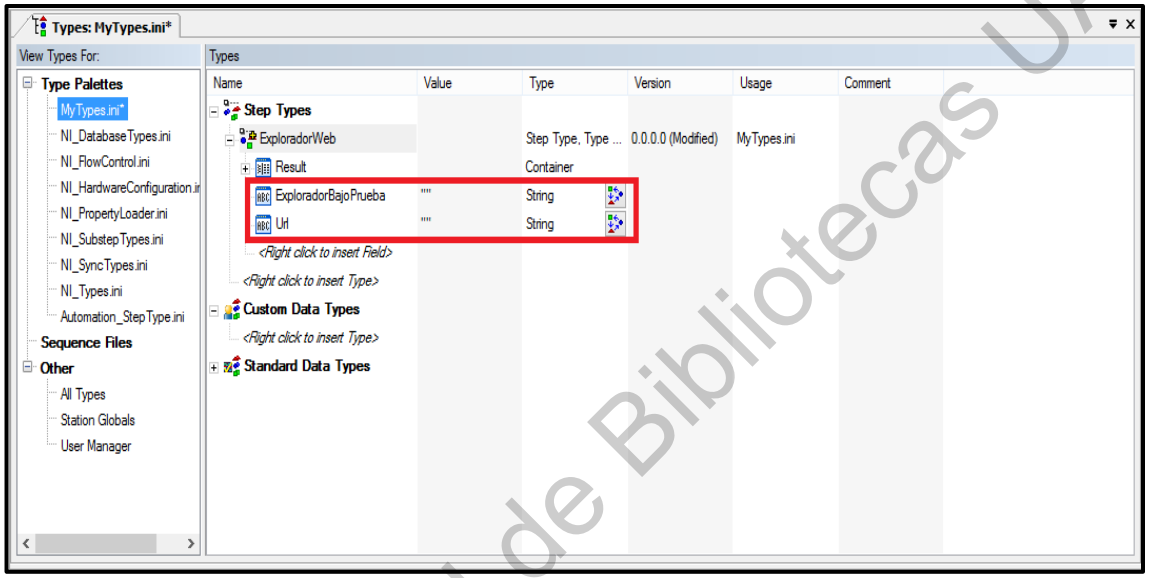

Figura 36. Parámetros para el tipo de paso. Fuente: Elaboración propia.

Como se puede observar en la figura 36, los parámetros agregados tienen que corresponder con el tipo de dato previamente establecido por la secuencia de prueba, si esto no es así, la configuración del tipo de paso marcará errores y no permitirá continuar con la misma.

Después de agregar los parámetros requeridos por la secuencia de prueba, es necesario salvar la configuración previamente establecida, sin embargo, cabe la posibilidad de recibir un mensaje de alarma, en este se tiene que seleccionar las opciones de "*No incremente las versiones de tipo*" y "*Eliminar marca modificada de tipos*", posteriormente se tiene que dar clic en el botón de ok.

Este tipo de alerta depende exclusivamente de la versión de la herramienta de desarrollo, en este caso "*TestStand*".

En la Figura 37 se muestra el mensaje de alarma comentado anteriormente.

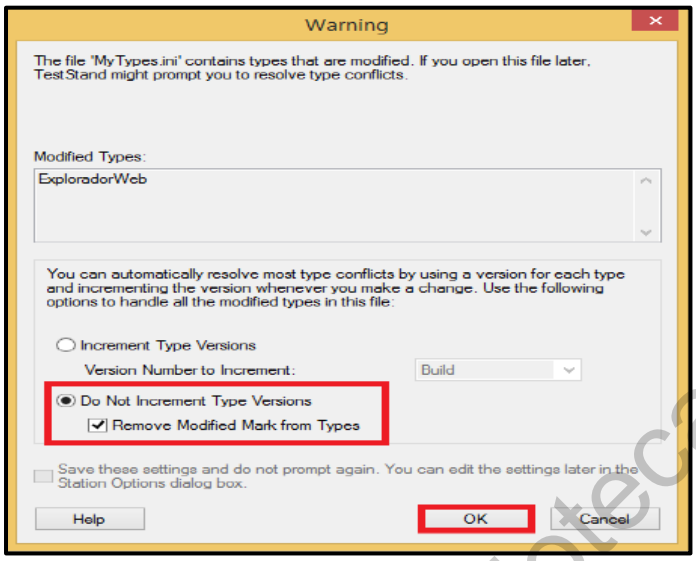

Figura 37. Mensaje de alarma posterior a salvar la configuración del tipo de paso. Fuente: Elaboración propia.

Terminada la configuración del tipo de paso, es necesario establecer algunas propiedades de este, para realizar lo anterior se tiene que dar clic derecho sobre el tipo de paso y seleccionar la opción de "*Propiedades*", esto despliega un menú de configuración, el cual permite establecer algunas propiedades, módulos de código y acciones predeterminadas para un tipo de paso.

En este menú se puede observar una gran cantidad de pestañas, pero para fines de este ejemplo nos enfocaremos en las pestañas "*General*" y "*Subpaso*", esta primera contiene el nombre, la descripción, el icono y el tipo de adaptador predeterminado.

Algunos de los elementos solicitados por este menú tienen que ser previamente establecidos, un ejemplo de esto es el icono representativo del tipo de paso, dado que "*TestStand*" no cuenta con alguna característica que permita elaborar alguno de manera directa, este tiene que ser implementado mediante algún otro método.

En la Figura 38 se muestra el menú de configuración de propiedades.

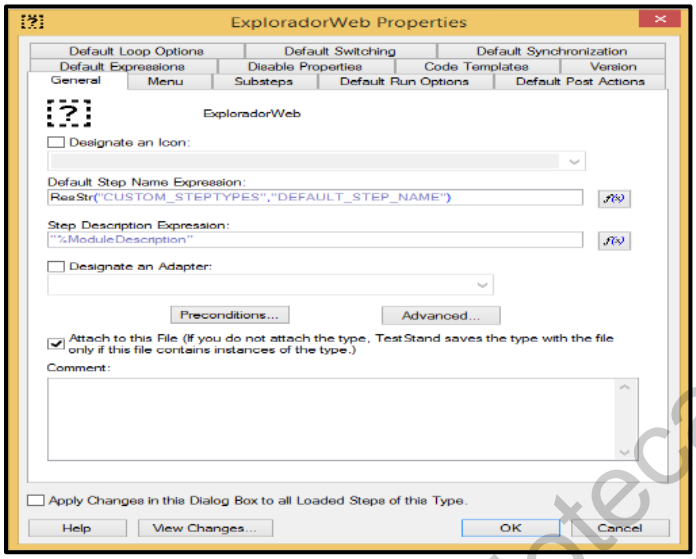

Figura 38. Menú de configuración de propiedades. Fuente: Elaboración propia.

La primera propiedad que se tiene que configurar es el icono representativo del tipo de paso, este tiene que representar de manera visual el propósito que este tiene establecido, para hacer esto se tiene que seleccionar la opción "*Designar icono*" y posteriormente elegir el icono correspondiente al tipo de paso.

En la Figura 39 se muestra lo comentado anteriormente.

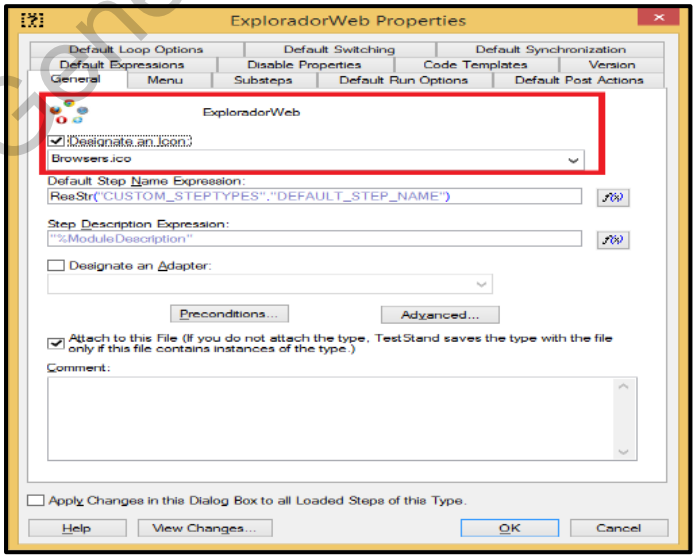

Figura 39. Configuración del icono correspondiente al tipo de paso. Fuente: Elaboración propia.

La segunda propiedad que se tiene que configurar es el nombre predeterminado del tipo de paso, este tiene que ser sencillo y representativo para el implementador de la prueba automatizada, para este ejemplo se utilizara el nombre de "*ExploradorWeb*".

En la Figura 40 se muestra la configuración del nombre predeterminado para el tipo de paso.

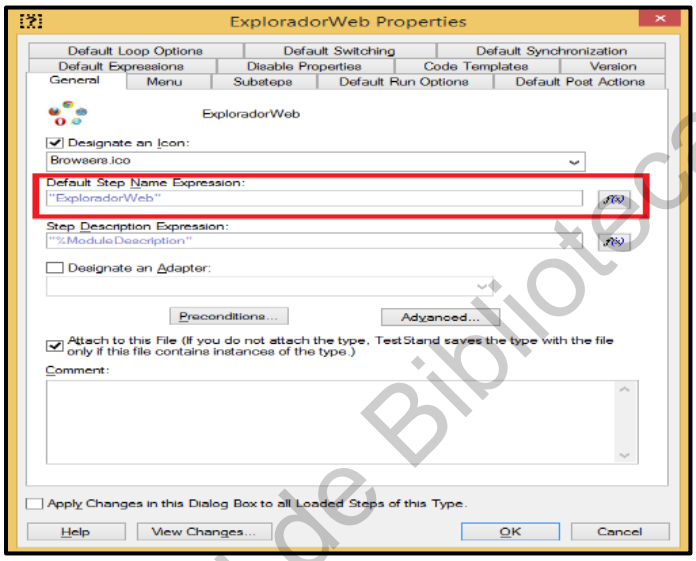

Figura 40. Configuración del nombre predeterminado para el tipo de paso. Fuente: Elaboración propia.

La tercera propiedad corresponde a la expresión de la descripción del paso, esta se mantiene configurada de manera predeterminada debido a que esta toma de manera directa la información correspondiente a la secuencia de prueba.

Por último, se selecciona la opción "*Designar adaptador*" y posteriormente se elige la opción correspondiente a "*Sequence*", esto se debe a que la secuencia de prueba que fue implementada para este ejemplo se diseñó utilizando este tipo de archivo.

Como se mencionó anteriormente, este tipo de integración fue posible debido a que tanto "*LabVIEW*" y "*TestStand*" al ser del mismo proveedor permiten una interacción directa entre plataformas, lo cual representa una ventaja con respecto a los demás módulos de programación soportados.

En la Figura 41 se muestra la configuración del adaptador correspondiente a "*Sequence*".

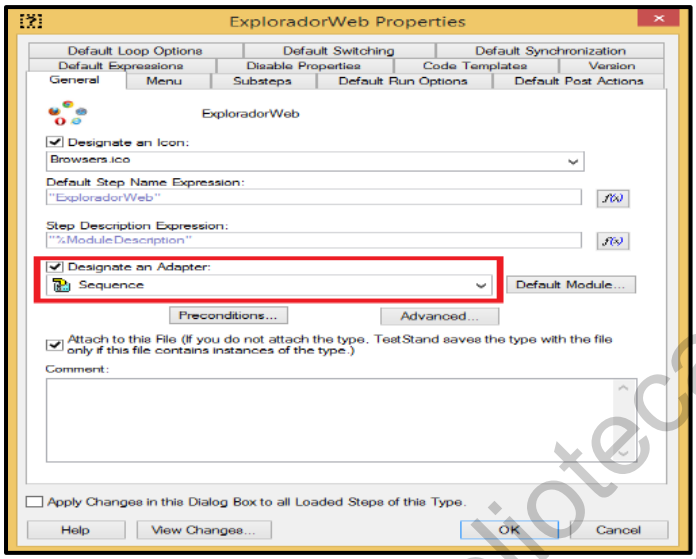

Figura 41. Configuración del adaptador predeterminado "Sequence". Fuente: Elaboración propia.

Una vez configurado el adaptador, es necesario establecer la secuencia de prueba correspondiente con el ejemplo, para realizar esto es necesario dar clic en el botón modulo predeterminado.

En la Figura 42 se muestra lo comentado anteriormente.

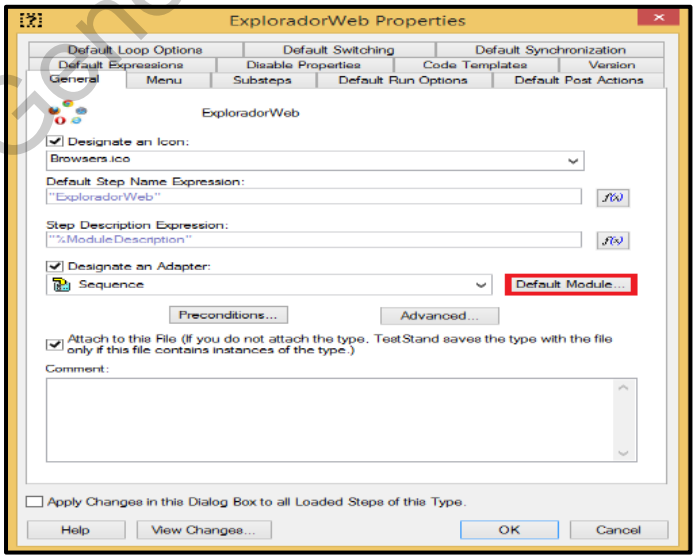

Figura 42. Botón de configuración del módulo predeterminado. Fuente: Elaboración propia.

Una vez presionado el botón, este despliega un menú de configuración que corresponde con el adaptador seleccionado anteriormente.

En la Figura 43 se muestra el menú de configuración correspondiente a la secuencia de prueba.

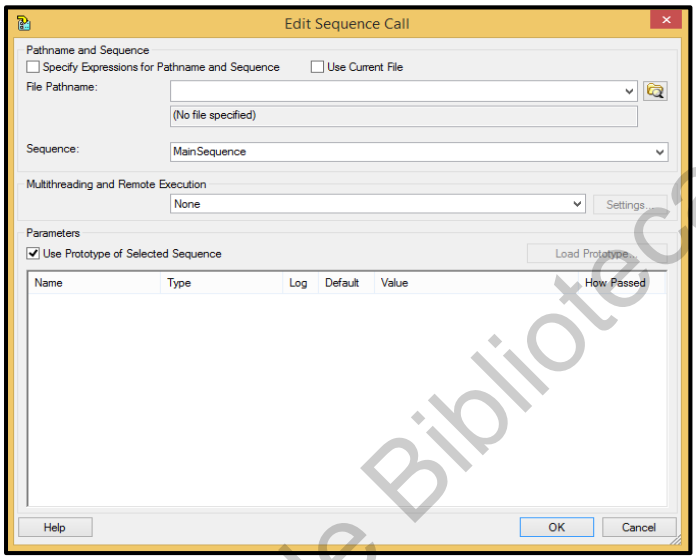

Figura 43. Menú de configuración del módulo predeterminado. Fuente: Elaboración propia.

Para seleccionar el archivo correspondiente con la secuencia de prueba, se tiene que dar clic sobre el botón de navegación y seleccionar el archivo correspondiente, esto se puede observar en la Figura 44.

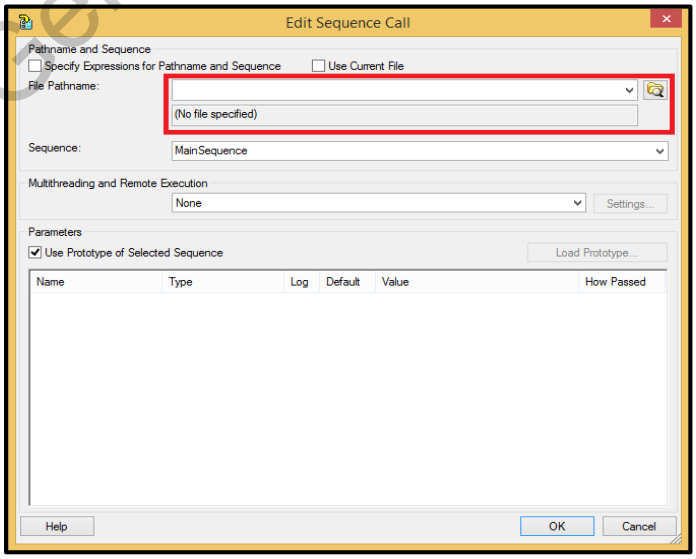

Figura 44. Botón de navegación - "Secuencia de prueba". Fuente: Elaboración propia.

Una vez seleccionado el archivo correspondiente a la secuencia de prueba, el menú de configuración despliega una previsualización de los parámetros establecidos como variables.

En la Figura 45 se muestra lo comentado anteriormente.

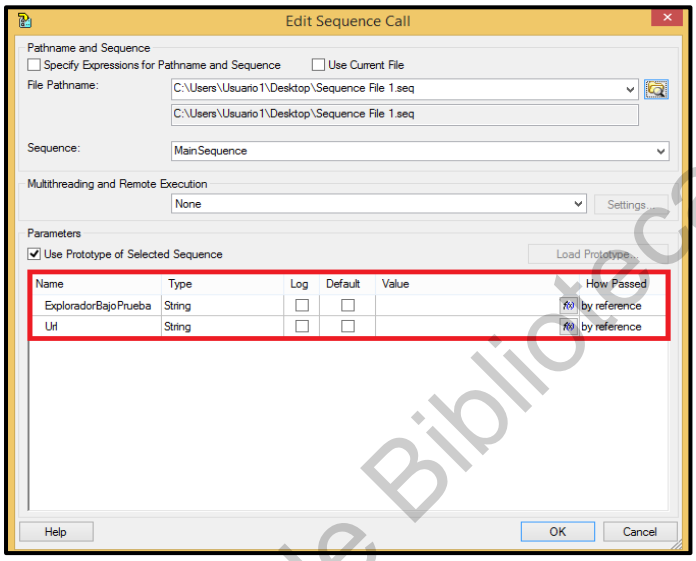

Figura 45. Parámetros correspondientes a la secuencia de prueba. Fuente: Elaboración propia.

Para realizar la configuración de estos parámetros, es necesario colocar la expresión "*Step.ExploradorBajoPrueba*" y "*Step.Url*" en su respectivo campo, esto se puede observar en la Figura 46.

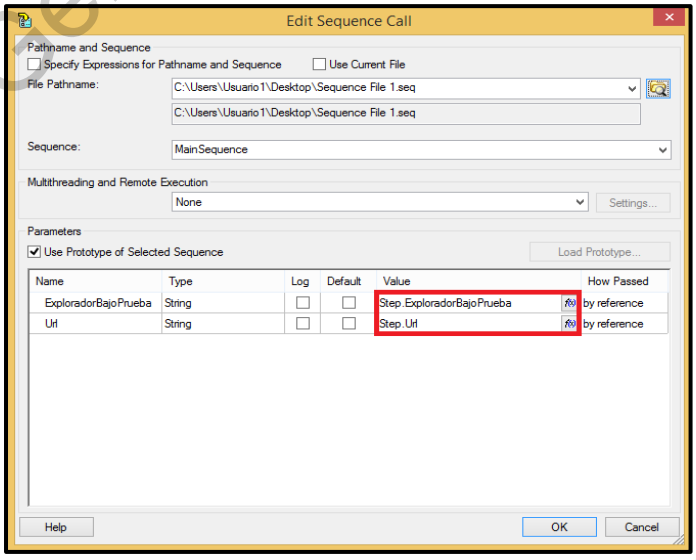

Figura 46. Expresiones de configuración correspondientes a la secuencia de prueba. Fuente: Elaboración propia.

Una vez configurados los parámetros de la secuencia de prueba, es necesario configurar la interfaz gráfica asociada a esta, para realizar esto es necesario ir a la pestaña "*Subpasos*" ubicada en el menú de configuración de propiedades del tipo de paso.

En la Figura 47 se puede observar la pestaña "*Subpasos*" en el menú de configuración de propiedades.

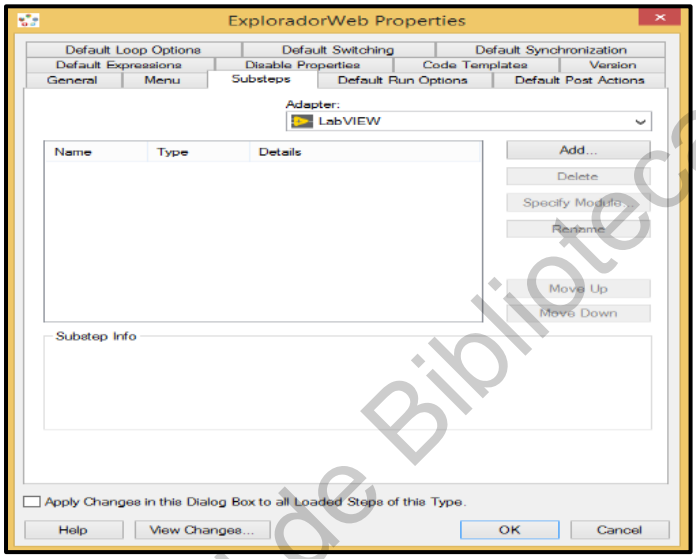

Figura 47. Pestaña de configuración - "Subpasos". Fuente: Elaboración propia.

Posterior a esto se necesita seleccionar el adaptador predeterminado correspondiente a "*LabVIEW*", esto se puede observar en la Figura 48.

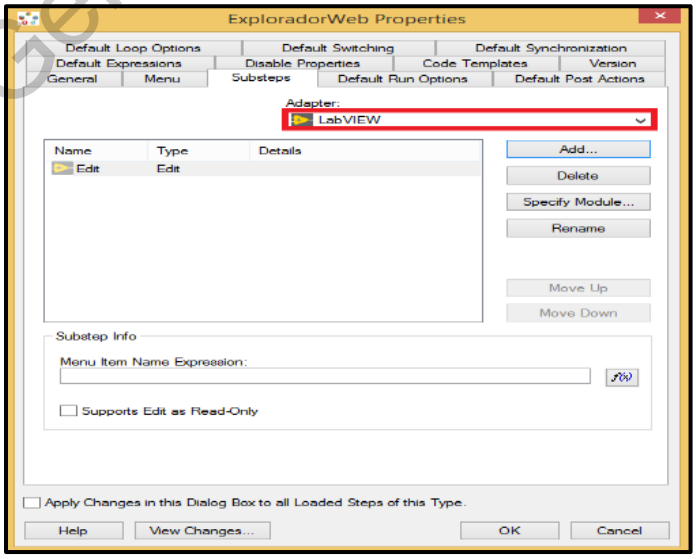

Figura 48. Configuración del adaptador "LabVIEW". Fuente: Elaboración propia.

Una vez configurado el adaptador, es necesario dar clic en el botón agregar y posteriormente seleccionar la opción de "*Editar*", esto se puede observar en la Figura 49.

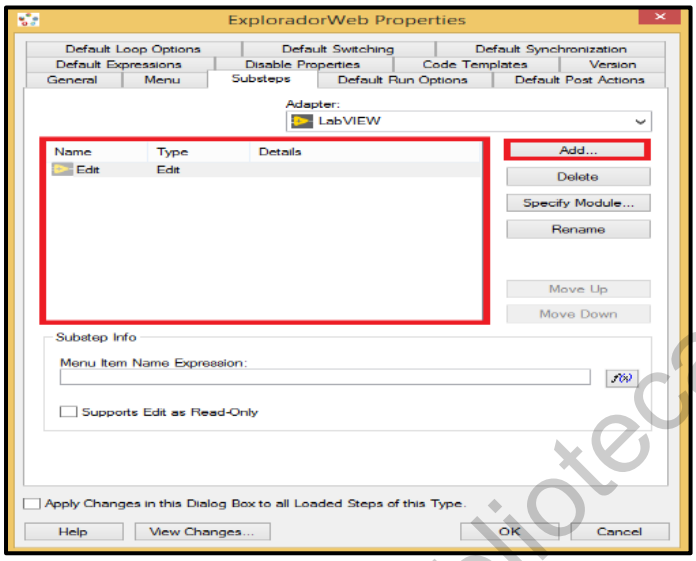

Figura 49. Configuración de la opción "Editar" correspondiente al adaptador "LabVIEW". Fuente: Elaboración propia.

Realizada la configuración, se debe establecer el módulo de prueba correspondiente a la interfaz gráfica, para realizar esto es necesario dar clic en el botón especificar módulo.

En la Figura 50 se muestra lo comentado anteriormente.

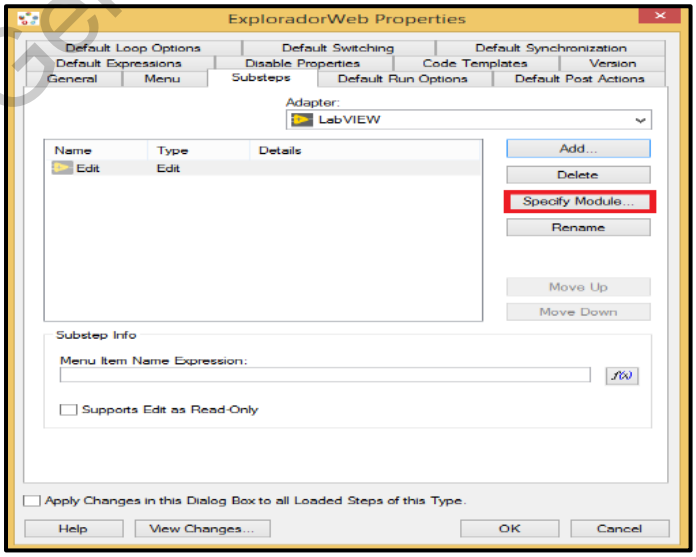

Figura 50. Botón de configuración del módulo específico. Fuente: Elaboración propia.

Una vez presionado el botón, este despliega un menú de configuración que corresponde con el módulo de prueba seleccionado anteriormente.

En la Figura 51 se muestra el menú de configuración correspondiente al adaptador previamente seleccionado.

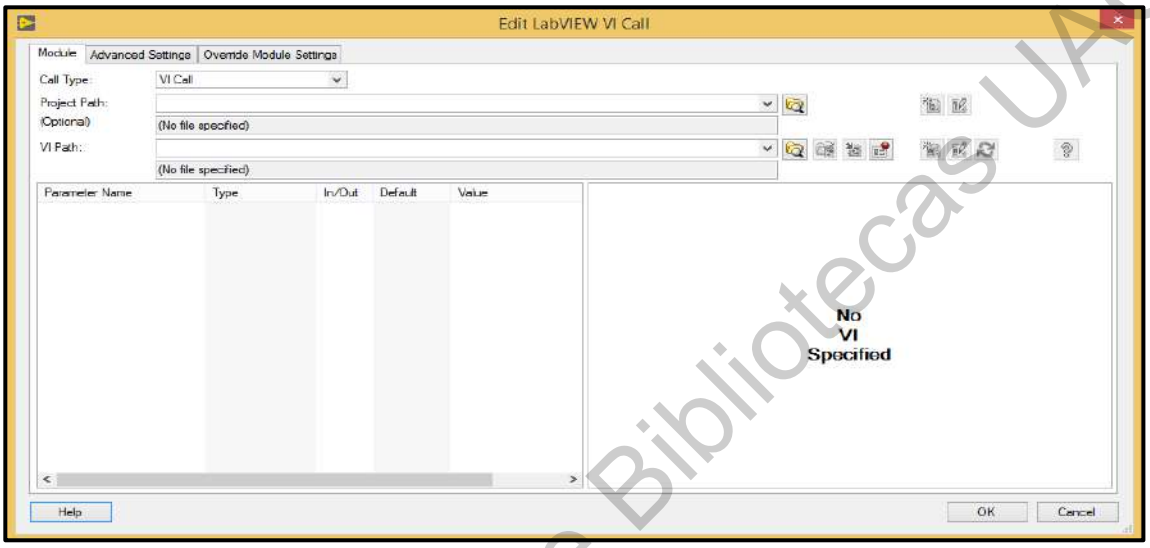

Figura 51. Menú de configuración del módulo específico. Fuente: Elaboración propia.

Para seleccionar el archivo correspondiente con el módulo de prueba, se tiene que dar clic sobre el botón de navegación y seleccionar el archivo correspondiente, esto se puede observar en la Figura 52.

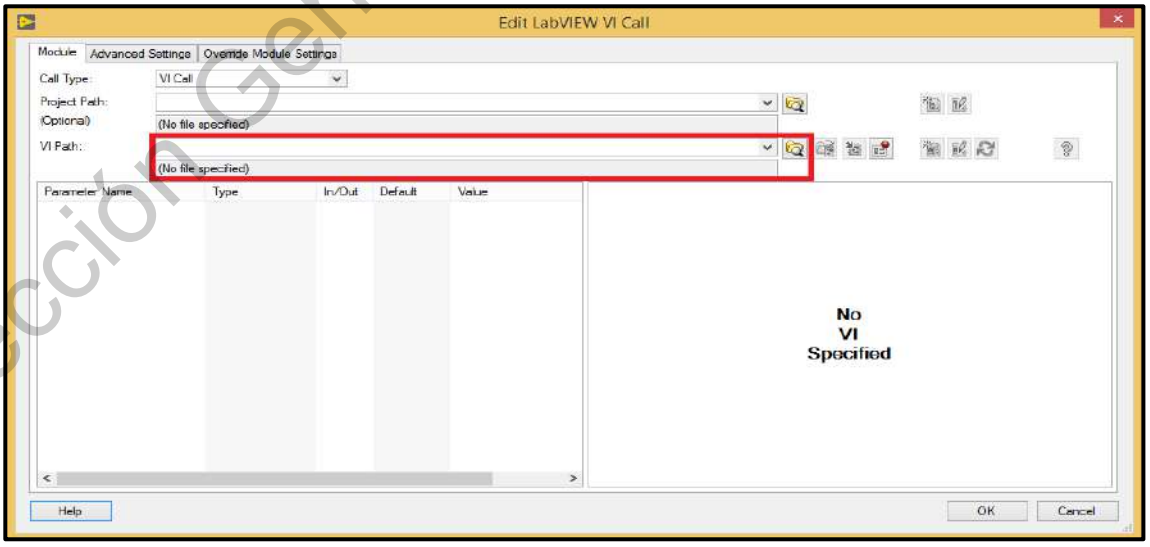

Figura 52. Botón de navegación - "Módulo de prueba". Fuente: Elaboración propia.

Una vez seleccionado el archivo correspondiente al módulo de prueba, el menú de configuración despliega una previsualización del módulo y los parámetros establecidos como variables.

En la Figura 53 se muestra lo comentado anteriormente.

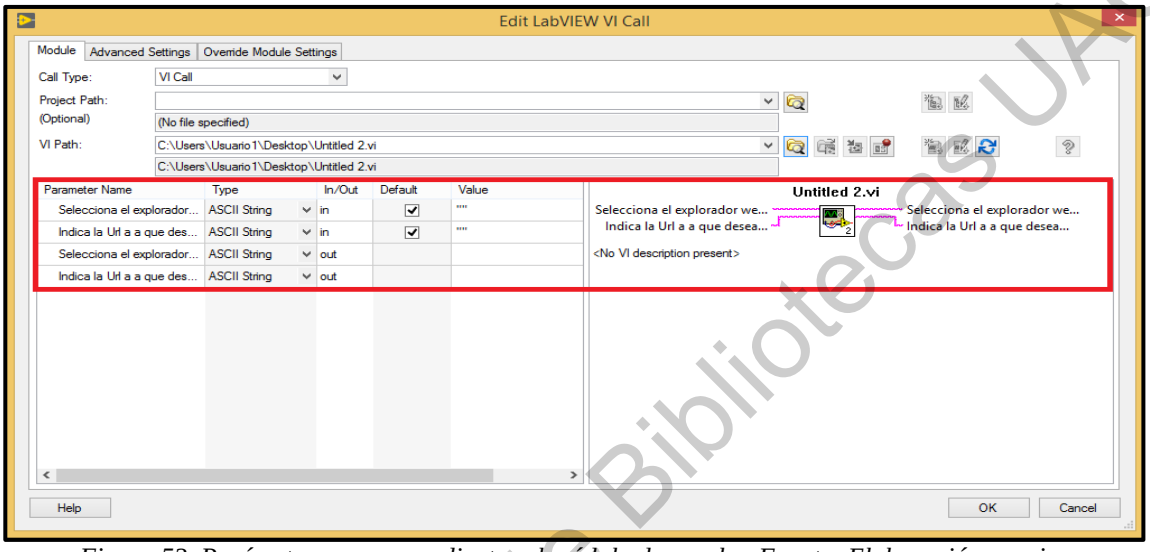

Figura 53. Parámetros correspondientes al módulo de prueba. Fuente: Elaboración propia.

Para realizar la configuración de los parámetros, se necesita colocar la expresión "*Step.ExploradorBajoPrueba*" y "*Step.Url*" en su respectivo campo, esto se puede observar en la Figura 54.

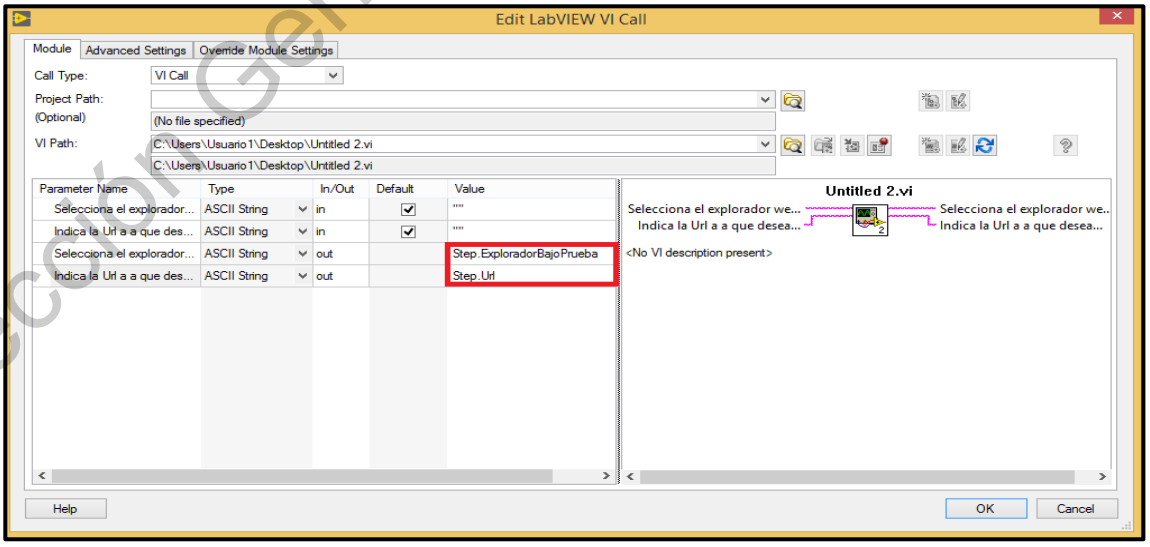

Figura 54. Expresiones de configuración correspondientes al módulo de prueba. Fuente: Elaboración propia.

Para concluir con la configuración del tipo de paso propuesto en este ejemplo, es necesario confirmar los cambios realizados en las diferentes pestañas del menú de configuración de propiedades.

Realizada la confirmación y salvados los cambios, el tipo de paso está listo para ser utilizado en cualquier momento, así mismo, este se encuentra disponible en la "*Paleta de inserción*" ubicado en la interfaz de desarrollo de "*TestStand*".

En la Figura 55 se muestra el resultado final correspondiente con el tipo de paso implementado en el ejemplo propuesto.

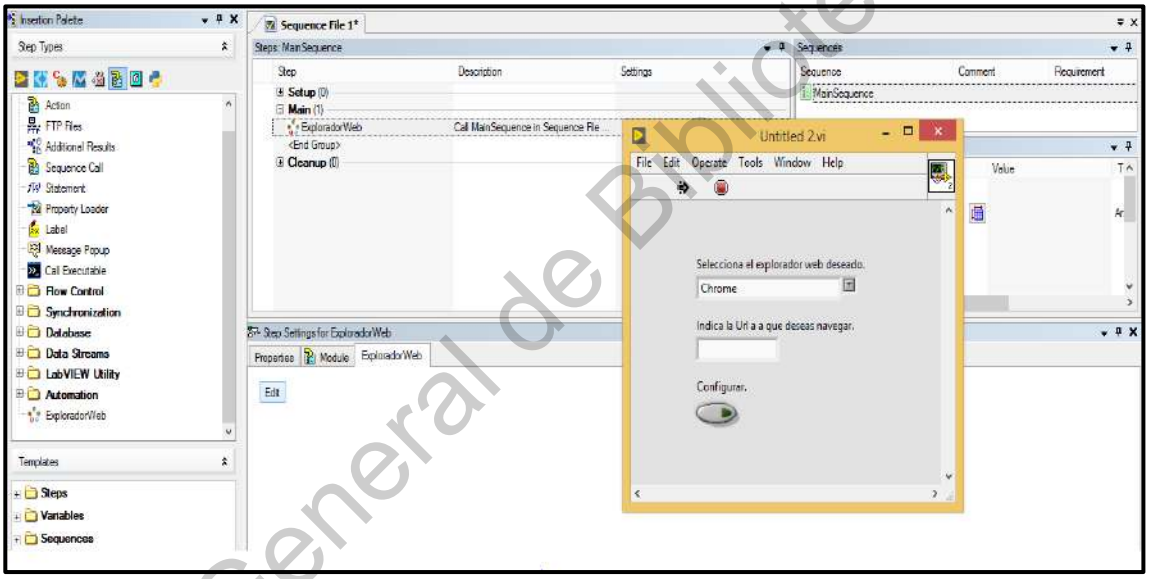

Figura 55. Tipo de paso correspondiente al ejemplo "Selección de un explorador web". Fuente: Elaboración propia.

El procedimiento realizado en esta sección del documento fue repetido para cada una de las palabras claves que forman parte del framework de automatización de pruebas, sin embargo, estas cuentan con características más elaboradas.

Estas características no afectan el procedimiento descrito anteriormente, dado que estas solo hacen referencia a algunos aspectos relacionados con el desempeño y la usabilidad de las interfaces gráficas.

3.5 Implementación

Una vez finalizado el desarrollo del framework gráfico de automatización, fue necesario buscar un método para distribuirlo de manera eficiente al equipo de pruebas y que estos pudieran hacer uso de este para buscar oportunidades de mejora.

El método seleccionado fue un instalador .exe, el cual simplemente hará una copia de los archivos y dependencias de la herramienta en la computadora en que se esté instalando, sin embargo, para que este pueda funcionar de manera correcta se necesita tener previamente instalado los programas que forman parte del framework de automatización, en este caso "*LabVIEW*" y "*TestStand*".

Estos programas no se pueden incluir en el instalador de la herramienta de automatización, debido a que este es una extensión de estos programas y por lo tanto necesitan configurase primeramente sobre ellos, además de que al ser programas de desarrollo estos tienen un peso extenso para poder ser incluidos directamente en él.

Antes de hablar de la herramienta que permitió crear el instalador, es necesario mostrar la estructura de carpetas que contienen todos los archivos, librerías y dependencias que el framework gráfico de automatización de pruebas necesita para funcionar adecuadamente.

Estas carpetas fueron definidas de acuerdo con las necesidades específicas de los proyectos bajo prueba para el cual fue desarrollado la herramienta de automatización, esto no significa que sea necesario replicarlo de manera idéntica dado que los archivos y dependencias corresponde a las necesidades específicas de ese momento.

Así mismo, es posible que la persona que quisiera retomar este trabajo pueda necesitar de diferentes librerías o incluso darle un giro diferente al área de aplicación, por lo que no se recomienda copiar la estructura sin antes definir sus necesidades.

En la Figura 56 se muestra la estructura de carpetas correspondiente a la herramienta de automatización.

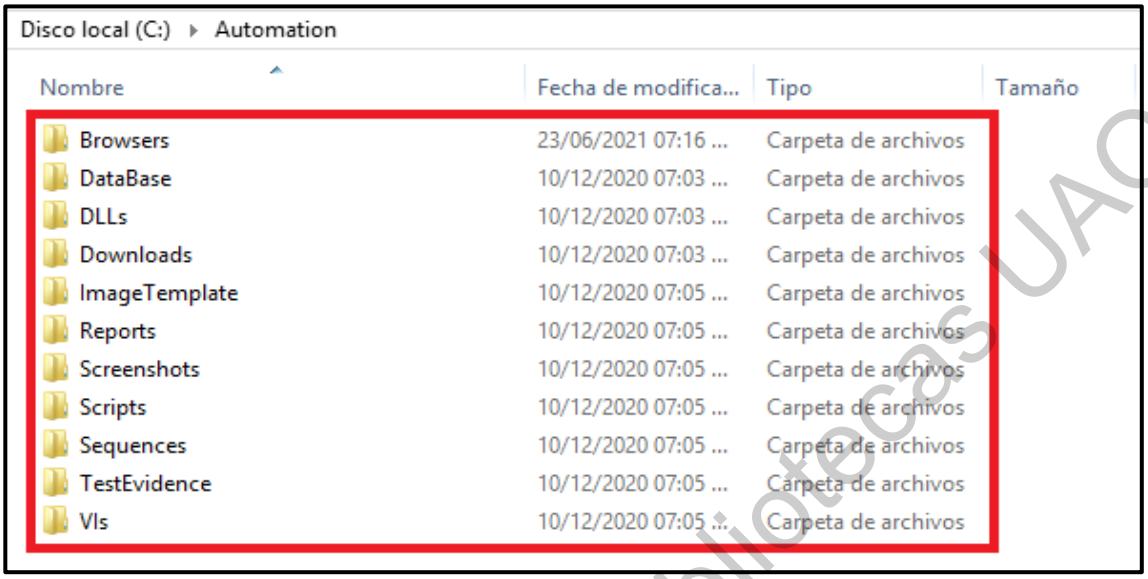

Figura 56. Estructura de carpetas pertenecientes al framework de automatización. Fuente: Elaboración propia.

Como se puede observar en la figura 56, todas las carpetas cuentan con un número importante de archivos, así mismo estos están ubicados en el disco local c, esto se debe a que en esta ubicación las rutas de los archivos no tienen ninguna dependencia del usuario de la computadora donde el framework de automatización este instalado.

Una vez comprendida la estructura de carpetas, es necesario hablar acerca de la herramienta que permitió realizar el ejecutable correspondiente al instalador del framework gráfico de automatización.

Esta herramienta lleva el nombre de "*InstallForge*", es uno de los programas de creación de configuraciones más populares para computadoras, y una de sus características más destacables es la velocidad, estabilidad y facilidad de uso.

Así mismo, esta herramienta no requiere de conocimientos específicos en algún lenguaje de programación, lo cual resulta en una ventaja añadida al momento de realizar la configuración del instalador, esto se debe a que esta cuenta con un interfaz de usuario que resulta amigable para cualquier tipo de persona que haga uso de esta.

En la Figura 57 se muestra la interfaz de usuario que corresponde a la herramienta "*InstallForge*".

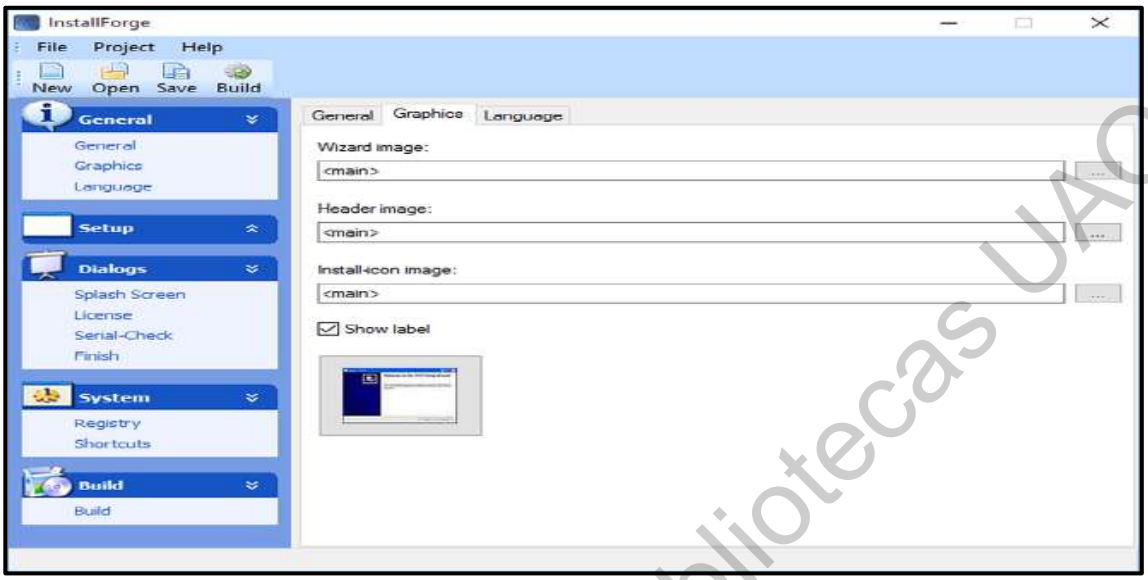

Figura 57. Interfaz de usuario de "InstallForge". Fuente: Elaboración propia.

Como se puede observar en la figura 57, esta herramienta requiere principalmente de la estructura de carpetas correspondientes a la herramienta de automatización, así como de otros parámetros de configuración que resultan esenciales para la misma.

En la Figura 58 se muestra el resultado final del ejecutable correspondiente al instalador del framework gráfico de automatización de pruebas.

Figura 58. Instalador del framework de automatización de pruebas. Fuente: Elaboración propia.

3.6 Evaluación

Para hacer una evaluación objetiva del framework gráfico de automatización de pruebas de software desarrollado en este documento, fue necesario realizar una prueba de concepto, dado que estas adecúan y mejoran las ideas para que tengan un mayor potencial de adaptación en el mercado.

De acuerdo con el Institut de Publique Sondage d'Opinion Secteur (2016) se define que la prueba de concepto consiste en la etapa del desarrollo de un producto en la que se exhibe una representación detallada de este (con características y ventajas) a los posibles usuarios o clientes con el fin de evaluar sus actitudes e impresiones respecto al producto mostrado.

Las pruebas de concepto:

- Evalúan el atractivo relativo o absoluto de un concepto, configuración o enfoque alternativo de un producto.
- Indican cuáles son los segmentos determinados de la población a los que les interesa el producto.
- Proporcionan la información requerida para la implementación del producto, así como para su promoción, distribución y determinación del precio.

Las pruebas de concepto brindan conclusiones para diseñar un producto o servicio de mejor manera, además permiten evaluar el éxito de los productos nuevos antes de su comercialización al público.

El mejor instante para realizar este tipo de pruebas es aquel en el que la idea ya ha sido desarrollada e incluye las particularidades del producto, posicionamiento deseado y la personalidad de la marca prevista, estas evalúan el concepto principal mediante un guion gráfico, bocetos o incluso una maqueta del producto.

Con este tipo de prueba se busca garantizar que esta herramienta pueda ser recibida de manera correcta por los implementadores de las pruebas automatizadas.

Otra característica importante de las pruebas de concepto es la reducción de los riesgos financieros del fabricante, dado que sería sumamente costoso transformar cada idea en un producto, comercializarlo, y que este al final fuera un fracaso, debido a las grandes pérdidas económicas que traerían consigo.

Existe una amplia cantidad de perspectivas para las pruebas de concepto, cada una de ellas tiene un objetivo totalmente diferente y puede ofrecer distintas ventajas.

A continuación, se presentan las más importantes:

- Pruebas de concepto de productos nuevos: Las características pueden clasificarse como indispensables y secundarias, las necesidades de los clientes deben identificarse y priorizarse para desarrollar el producto y distribuirse al mercado.
- Pruebas de reedición: Una reformulación, modificación o actualización pueden conceder una nueva oportunidad a los productos o servicios existentes, la prioridad de estas consiste en identificar el conjunto de características óptimas.
- Pruebas de posicionamiento: Mantener la oferta de manera constante, modificando el medio utilizado para describirla y de esta forma comprender las reacciones de los consumidores, esto constituye principalmente el enfoque de una prueba de posicionamiento.
- Pruebas de rutas de migración: Conocer las características y las ventajas principales de un producto resulta esencial a la hora de definir las necesidades del consumidor con respecto a la probabilidad de actualizar un producto existente o adoptar una nueva tecnología.
- Pruebas de precios e incentivos: Los paquetes, el precio, los incentivos, los productos de venta y los factores que reducen el precio, así como las garantías y los convenios de uso, cambian las percepciones tanto del precio como del valor del producto en general.

En resumen, las pruebas de concepto tienen la capacidad de mejorar el producto e incrementar la asociación entre mercado y producto, además de mitigar los riesgos económicos para el fabricante.

Para la herramienta desarrollada en este trabajo se aplicó la prueba de concepto de productos nuevos, esto se debe a que el framework gráfico de automatización de pruebas de software es una herramienta nueva en comparación con otras que ya están disponibles actualmente en el mercado.

Para realizar esta evaluación de manera objetiva primero se implementó un prototipo de la herramienta, la cual contaba con una cantidad limitada de palabras clave pero que permitía al usuario tener la experiencia de implementar una prueba automatizada sencilla y de esta forma obtener alguna percepción ya sea positiva o negativa de la misma.

Este prototipo fue evaluado por el equipo de pruebas conformado por cinco personas en ese momento, los cuales tuvieron la oportunidad de utilizar la herramienta en un periodo de tiempo determinada para poder definir su criterio, el cual posteriormente seria recibido a manera de crítica constructiva en una mesa redonda.

Los resultados obtenidos fueron parcialmente aceptables por la mayoría de los ingenieros que formaron parte de la mesa redonda, sin embargo, de la misma manera se obtuvieron sugerencias y recomendaciones para mejorar la experiencia del usuario final.

Algunas de las recomendaciones consistían en mejorar el diseño de las interfaces gráficas, y de esta forma simplificar el camino que el implementador de las pruebas automatizadas debería seguir al momento de estar llenando los parámetros requeridos por cada palabra clave, así mismo se sugirió que el framework gráfico de automatización debía tener compatibilidad directa con la herramienta de administración de pruebas al momento de subir los resultados obtenidos hacia la misma.

Una vez implementadas las mejoras recibidas en la primera prueba de concepto, se volvió a compartir el prototipo modificado con el equipo de pruebas para volver a realizar el proceso previamente mencionado, donde posteriormente se obtendría una aceptación unánime por parte de todo el equipo de pruebas.

3.7 Pruebas

Una vez terminado el desarrollo del framework gráfico de automatización, fue necesario realizar una serie de pruebas que validaran la completa funcionalidad de este, estas pruebas fueron necesarias debido a que esta herramienta está basada en software y por lo tanto debe seguir el mismo lineamiento de evaluación que seguiría cualquier desarrollo de software convencional.

Sin embargo, debido a que esta herramienta no contaba con requerimientos funcionales que definieran como tenía que ser implementada, así como la necesidad de que estas pruebas fueran sencillas y rápidas de ejecutar, se tuvo que hacer uso de algunas técnicas de prueba, específicamente de las pruebas basadas en la experiencia, para poder entender este tipo de técnicas es necesario definirlas.

La ISTQB define que las técnicas de prueba basadas en la experiencia utilizan la experiencia de los desarrolladores, evaluadores y usuarios para diseñar, implementar y ejecutar pruebas, estas comúnmente se combinan con técnicas de prueba de caja negra y caja blanca.

Estas técnicas pueden ser utilizadas para identificar pruebas que no fueron fácilmente identificadas por otras técnicas más sistemáticas, dependiendo del enfoque y la experiencia del evaluador, estas técnicas pueden lograr grados muy diversos de cobertura y efectividad.

De esta forma los casos de prueba se derivan de la habilidad y la intuición del evaluador y de su experiencia con aplicaciones y tecnologías similares.

Aunque este tipo de técnicas de prueba suenen en esencia bien, estas presentan un problema, la cobertura puede ser difícil de evaluar y es posible que no se pueda medir con estas técnicas, por lo tanto, no es posible realizar algún tipo de métrica sobre estas.

Las técnicas de prueba basadas en experiencia son:

- Adivinación de errores: Es utilizada para anticipar la ocurrencia de errores, defectos y fallas, esta se basa en el conocimiento del evaluador, que incluye:
    - Cómo funciono la aplicación en el pasado.
    - Qué tipo de errores cometen los desarrolladores.
    - Fallos que han ocurrido en aplicaciones similares.

Un enfoque sistemático de la técnica de adivinación de errores es crear una lista de posibles errores, defectos y fallas, y con esta diseñar pruebas que exhiban esas fallas y los defectos que las causaron, este tipo de listas de errores, defectos y fallas se pueden elaborar en base a la experiencia, los datos de defectos y fallas, o del conocimiento común sobre por qué falla el software.

- Pruebas exploratorias: Las pruebas informales o no predefinidas se diseñan, ejecutan, registran y evalúan dinámicamente durante la ejecución de la prueba.

Las pruebas exploratorias son de mayor utilidad cuando hay pocas o inadecuadas especificaciones de prueba, así como una presión de tiempo considerable al momento de la ejecución, las pruebas exploratorias también son útiles para complementar otras técnicas de prueba más formales.

- Pruebas basadas en listas de verificación: En las pruebas que están basadas en listas de verificación, los evaluadores diseñan, implementan y ejecutan pruebas para cubrir las condiciones que se encuentran en una lista de verificación.

Estas listas de verificación se pueden crear en función de la experiencia y el conocimiento sobre lo que es importante para el usuario o la comprensión de por qué y cómo falla el software, en ausencia de casos de prueba detallados, las pruebas basadas en listas de verificación pueden proporcionar pautas y cierto grado de coherencia.

Para la realización de la validación de este framework gráfico de automatización se utilizó la técnica de pruebas exploratorias, dado que estas permiten realizar las pruebas al mismo tiempo que estas se están ejecutando y esto reduce de manera considerable el tiempo que toma en llevar al cabo la ejecución de estas.

Estas pruebas fueron efectuadas sobre cada palabra clave que estuviera implementada en el lanzamiento de una nueva versión de la herramienta, estas pruebas consistían en hacer un barrido rápido de cada elemento que formara parte de la estructura de la palabra clave, esto hace referencia a la interfaz gráfica y la secuencia de prueba.

Una parte de las pruebas realizadas sobre las interfaces gráficas consistían en validar que los elementos que formaran parte de esta funcionaran sin problema alguno, por ejemplo, si un menú desplegable contaba con cuatro opciones, se buscaba que el implementador de la prueba automatizada pudiera hacer uso de las cuatro, así como si este quisiera hacer uso de un botón de configuración, este no efectuara una acción correspondiente al mismo, así como otros tipos de escenarios que pudieran salir al momento de estar realizando la validación.

De la misma forma y como parte integrada a la interfaz gráfica, se realizaban pruebas a la secuencia de prueba de manera indirecta, dado que se hacía uso de pruebas de caja negra para validar que los datos ingresados por el implementador de la prueba automatizada fueran recibidos de manera correcta por la secuencia de prueba y esta efectuara las acciones correspondientes a la misma.

Relacionado a lo comentado anteriormente, también se realizaba la validación de los valores recibidos mediante la interfaz gráfica directamente en las variables correspondientes a las secuencias de prueba, estas pruebas pudieran considerase como algún tipo de prueba de caja blanca, dado a que se están revisando a nivel de código.

Sin embargo, es importante mencionar que en ninguna de las pruebas realizadas se aplicaron técnicas que correspondieran a las pruebas de caja negra y caja blanca.

4. EXPERIMENTACIÓN Y RESULTADOS

Resultado del desarrollo del framework de automatización gráfico presentado en este documento, se obtuvo un conjunto de las palabras clave más utilizadas por los implementadores de pruebas automatizadas, esto representa una ventaja en comparación con los métodos convencionales debido a que esto permite enfocarse exclusivamente en el desarrollo de las pruebas automatizadas en vez de implementar funcionalidades que son de uso genérico.

En la Figura 59 se muestra el conjunto de palabras clave que dispone la herramienta de automatización.

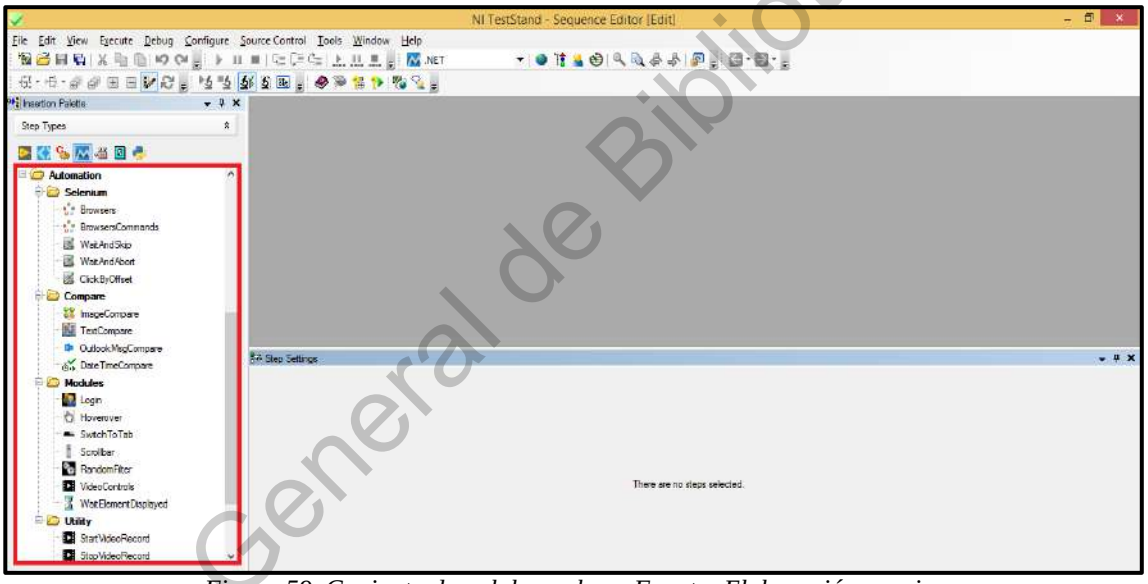

Figura 59. Conjunto de palabras clave. Fuente: Elaboración propia.

Una vez que el implementador de la prueba automatizada está enfocado exclusivamente en la solución de esta, se obtuvo una reducción de tiempo importante al momento de la implementación, ya que esta herramienta al ser gráfica solo solicita el llenado de parámetros específicos para la ejecución de las palabras clave y por lo tanto no se desperdicia tiempo en la implementación de soluciones específicas para algún tipo de problemática.

Este tiempo ahorrado puede ser utilizado tanto en la implementación de nuevas pruebas automatizadas, así como en el desarrollo de nuevas palabras claves que fueran requeridas y que no estén contempladas en el conjunto inicial.

En la Figura 60 se muestra un ejemplo de una implementación de una prueba automatizada mediante el framework gráfico.

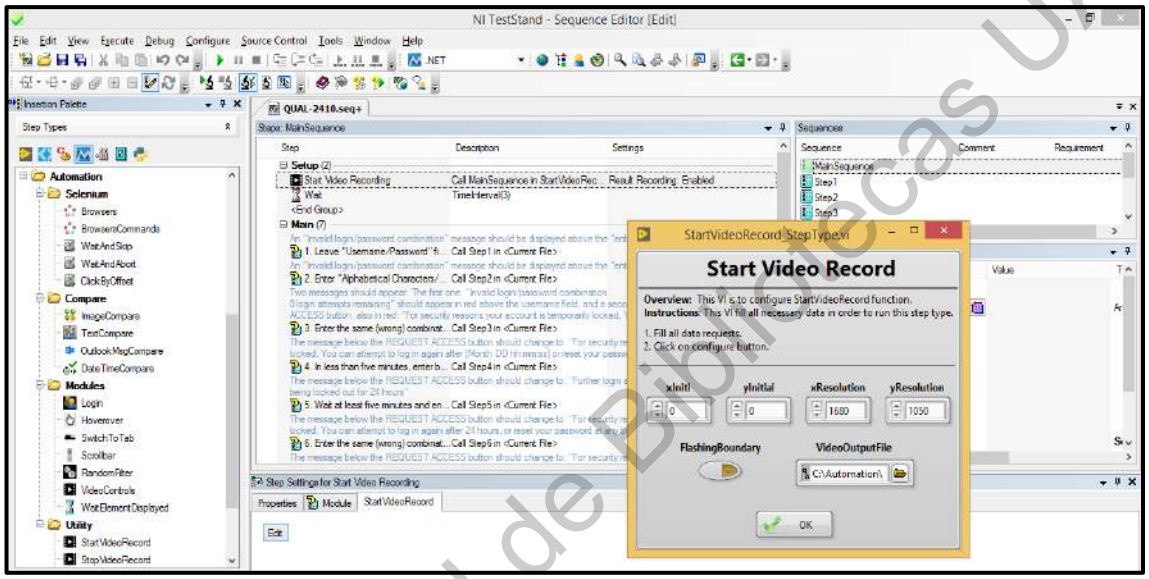

Figura 60. Prueba automatizada mediante el framework gráfico. Fuente: Elaboración propia.

Para poder medir de manera objetiva el tiempo de implementación se propuso un escenario de prueba muy común, el cual consiste en ingresar a una aplicación web con un usuario y contraseña válida, y dentro de la aplicación buscar un elemento web (botón, campo de texto etc.) y extraer el texto o un atributo de este y compararlo con un resultado esperado.

En la Tabla 2 se muestra el tiempo que tarda un ingeniero de pruebas de software en implementar el ejemplo propuesto anteriormente.

Método	*Tiempo de implementación (minutos)*
Framework Gráfico.	*5*
Convencional.	*10*

Tabla 2. Comparación de tiempo de desarrollo entre el método convencional vs el framework gráfico. Fuente: Elaboración propia.

Con la ventaja de implementar las pruebas automatizadas en menor tiempo, se obtiene la posibilidad de tener una mayor cobertura de estas, y por lo tanto de reducir el tiempo de ejecución en cualquier tipo de producto bajo prueba.

En la Tabla 3 se muestra el tiempo ahorrado en la ejecución de las pruebas automatizadas mediante el framework gráfico para un tipo de producto en específico.

Característica	*Tiempo manual (horas)*	*Tiempo automatizado (horas)*	*Tiempo ahorrado*
Login.	*6*	*1.30*	*78%*
Account settings.	*18*	*1.10*	*94%*
Home.	*6*	*1.20*	*80%*
Vehicle.	*50*	*2.30*	*95%*
Data Lab.	*26*	*1.10*	*96%*
Events.	*50*	*1.00*	*98%*
Reports.	*60*	*6.30*	*90%*

Tabla 3. Comparación de tiempo de ejecución de pruebas manual vs automatizado. Fuente: Elaboración propia.

Como se puede observar en las tablas anteriores, la reducción de los tiempos tanto de implementación como de ejecución son considerablemente buenos, sin embargo, en el caso específico del tiempo de implementación este no se puede comparar de manera exacta con su contra puesto, esto se debe a que la velocidad con la que se implemente la prueba automatizada depende directamente de las habilidades que el ingeniero de pruebas tenga en ese momento, aun así se puede asumir que el simple hecho de contar con una interfaz gráfica que guie al usuario en el transcurso de la implementación reduce considerablemente la dependencia de tener conocimientos extraordinarios en lenguajes de programación, haciendo centrarse en el verdadero objetivo de un ingeniero de pruebas, el cual es encontrar defectos en el software.

5. CONCLUSIONES

Se puede concluir que un framework gráfico de automatización de pruebas puede ser una buena solución para una empresa en la que su plantilla de ingenieros de pruebas de software no esté formada por expertos en los diferentes lenguajes de programación requeridos para la implementación de soluciones de automatización, dado que este tipo de herramienta permite desarrollar casos de prueba automatizados con una curva de aprendizaje relativamente corta en comparación con los métodos convencionales, así como el fácil mantenimiento y reusó de estas.

De esta manera, se busca reducir principalmente el tiempo de implementación y ejecución de pruebas automatizadas, todo esto se puede ver reflejado en un ahorro considerable de dinero para el departamento de pruebas de cualquier industria.

Aunque esta solución suene bastante prometedora, es importante mencionar que el framework gráfico de automatización de pruebas esta desarrollado mediante el software de "*National Instruments*", el cual es relativamente costoso en cuestión de licencias de uso, esto reduce considerablemente el número de posibles clientes que puedan acceder a esta solución, sin embargo, aún existe la posibilidad de lanzar este producto como una extensión para su software base, el cual definitivamente puede ser adquirido por "*National Instruments*" y posteriormente vendido y distribuido por el mismo.

Por otro lado, actualmente existen diversas plataformas de automatización de pruebas, muchas de esta cuentan con entornos de programación gráfica lo cual implica una severa competencia en cuestión de mercado, algunos de esto inclusive son de licencia gratuita para un número reducido de usuarios, un ejemplo notorio es el de "*Katalon Studio*", esta plataforma cuenta con un conjunto de herramientas gráficas para la elaboración de scripts de prueba.

Aunque actualmente existan otras alternativas de automatización en el mercado, no cabe duda de que "*Katalon Studio*" es actualmente una de las más utilizadas.

En la Figura 61 se muestra un ejemplo de la implementación de una prueba automatizada mediante "*Katalon Studio*".

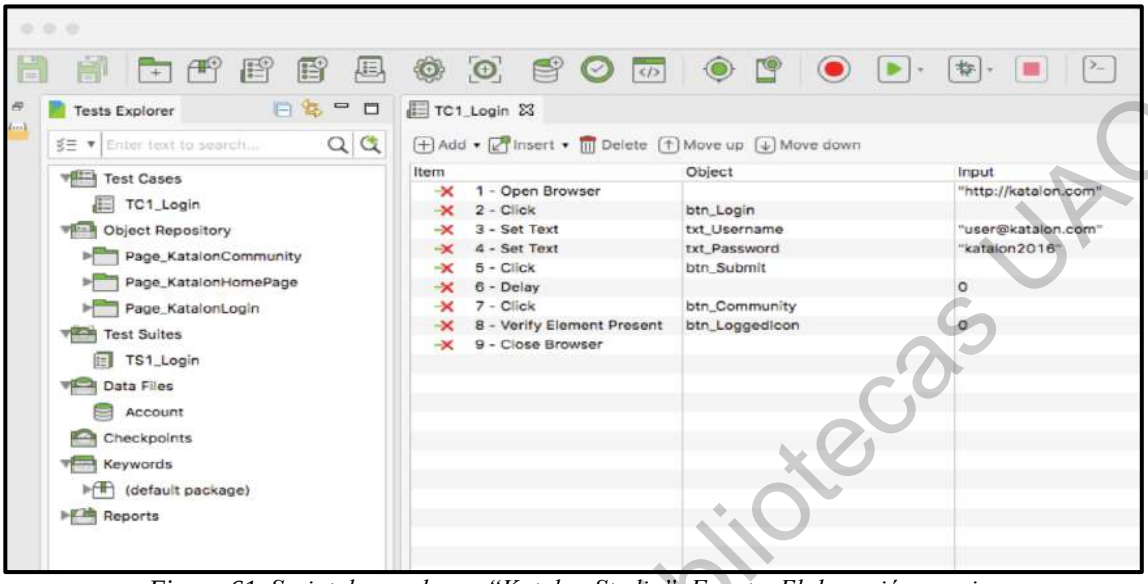

Figura 61. Script de prueba en "Katalon Studio". Fuente: Elaboración propia.

Al parecer "*Katalon Studio*" resulta ser una solución ideal, esto se debe a su licencia gratuita, así como a su capacidad de desarrollar pruebas automatizadas mediante un esquema hibrido, ya sea utilizando el método de scripting convencional o mediante su interfaz gráfica.

Sin embargo, aún existe la posibilidad de competir contra una opción tan interesante, dado a que "*National Instruments*" cuenta con una enorme reputación a nivel mundial como el software por excelencia de automatización de pruebas, y muchas empresas premium se encuentran colindadas con dicha compañía y sus productos.

Tomando en cuenta que actualmente la tendencia de cualquier negocio es brindar soluciones digitales mediante la nube, no cabe ninguna duda que existirá algún interesado en la solución propuesta en este documento.

Por último, es importante resaltar que esta herramienta está totalmente sujeta a la mejora, ya que es de suma importancia el estar realizando un mantenimiento constante debido a su dependencia en cierto tipo de librerías que están en constante actualización.

REFERENCIAS

Abhishek, J. and Sheetal, S. (2012). *AN EFFICIENT KEYWORD DRIVEN TEST AUTOMATION. INTERNATIONAL JOURNAL OF ENGINEERING SCIENCE & ADVANCED TECHNOLOGY FRAMEWORK FOR WEB PPLICATIONS*, Vol. 2, 600-604. Obtenido el 7 junio 2019 de: http://ijesat.org/Volumes/2012_Vol_02_Iss_03/IJESAT_2012_02_03_36.pdf

Alshahwan, N, Harman, M, Marchetto A. and Tonella P. (2014). *Improving Web Application Testing Using Testability Measures*. Obtenido el 7 de junio 2019 de: http://www0.cs.ucl.ac.uk/staff/M.Harman/wse09.pdf

Alshahwan, N. and Harman, M. (2015). *Automated Web Application Testing Using Search Based Software Engineering*. Obtenido el 7 junio de 2019 de: http://www0.cs.ucl.ac.uk/staff/M.Harman/ase11-na.pdf

Amir Ngah, Malcolm Munro and Mohammad Abdallah (2017). *An Overview of Regression Testing*. Obtenido el 7 junio 2019 de: https://www.researchgate.net/publication/321243328_An_Overview_of_Regression_Testing

Artiz, S, Dolby J, Holm, S, Moller, A, and Tip, F. (2012). *A Framework for Automated Testing of JavaScript Web Applications.* Obtenido el 7 de junio 2019 de: https://cs.au.dk/~amoeller/papers/artemis/paper.pdf

Bhondokar, B., Ranawade, P., Jadhav, S., and Vibhute, M. (2015). *Hybrid test automation framework for web application, International Journal of research and Technology*, Vol. 4. Obtenido el 7 junio 2019 de: https://www.ijert.org/research/hybrid-test-automation-framework-for-web-application-IJERTV4IS041292.pdf

Brühlmann, A. (2006). *Albatross: Seaside Web Applications Scenario Testing Framework. Software Composition Group, University of Bern*, 1-18. Obtenido el 7 junio 2019 de: http://scg.unibe.ch/archive/projects/Brue06a.pdf

Bucur, S, Kinder, J. and Candea, G. (2014). *Making Automated Testing of Cloud Applications an Integral Component of PaaS*. Obtenido el 7 de junio 2019 de: http://www.cs.rhul.ac.uk/home/kinder/papers/apsys13.pdf

Bustamante, Justo. (2017). *Introduction of Protractor as test automation framework for AngularJS applications*. Obtenido el 7 de junio 2019 de: https://www.theseus.fi/bitstream/handle/10024/129866/Bustamante_Justo.pdf?sequence=1&isAllowed=y

Colectiva, Nube. (2016). *Que es un Framework, Historia y Más Detalles*. Obtenido el 7 de junio 2019 de: http://blog.nubecolectiva.com/que-es-un-framework-historia-y-mas-detalles/

Divya, A. and Mahalakshmi, D. (2014). *An Efficient Framework for Unified Automation Testing: A Case*. *International Journal of Advanced Research in Computer Science & Technology*, 2(1), 15-19. Obtenido el 7 junio 2019 de: https://www.ijarcst.com/conference/first/conf2.pdf

Donley, B and Jeff Offutt, J. (2009). Web Application Testing Challenges, George Mason University, 1-26. Obtenido el 7 junio 2019 de: http://www.quaso.com/knowledge-base/Web-Application-Testing-Complexities-v1.1.pdf

Ghufran, A., Tiwari, R.G., and Kumar, P. (2017). *Web Application Testing Framework using Agents. International Journal of Computer Applications, 66(7), 10-13*. Obtenido el 7 junio 2019 de: https://pdfs.semanticscholar.org/c07d/295b34a6b731357de15b02127ad43a81f9b7.pdf

Grabinski, L. and O'Hare, J. (2014). *How-to-Implement-Efficient-Test-Automation-on-an-Agile-Project*. Obtenido el 7 de junio 2019 de: https://www.agileconference.org/wp-content/uploads/2014/10/-Project-Lukasz-Grabinski-John-OHare.pdf

Hanna, M, Elsayed, A. and Sami, Mostafa. (2014). *A Review of Scripting Techniques Used in Automated Software Testing*. Obtenido el 7 de junio 2019 de: https://www.researchgate.net/publication/307843614_A_Review_of_Scripting_Techniques_Used_in_Automated_Software_Testing

Hanna, M, Elsayed, A. and Sami, Mostafa. (2018). *Automated Software Testing Framework for Web Applications*. Obtenido el 7 de junio 2019 de: https://www.ripublication.com/ijaer18/ijaerv13n11_141.pdf

Institut de Publique Sondage d'Opinion Secteur (2016). *Prueba de concepto*. Obtenido el 7 de junio 2019 de: https://www.ipsos.com/es-es/prueba-de-concepto

International Software Testing Qualification Board (2018). *Foundation Level Syllabus*. Obtenido el 7 de junio 2019 de: https://www.istqb.org/certification-path-root/foundation-level-2018.html

Juan Camilo Salazar Rodriguez (2016). *Automatización de pruebas de software web basada en reglas de negocio*. Obtenido el 7 junio 2019 de: http://repository.udistrital.edu.co/bitstream/11349/5194/1/SalazarRodriguezJuanCamilo2016.pdf

Kalmo, M. (2009). *Automated Testing of Java Web Applications.* Obtenido el 7 de junio 2019 de: http://publications.lib.chalmers.se/records/fulltext/117314.pdf

Kurma, M. and Rai, M. (2018). *BEST PRACTICES IN AUTOMATION TESTING OF MOBILE APPLICATIONS*. Obtenido el 7 de junio 2019 de: https://www.infosys.com/digital/insights/Documents/mobile-application-landscape.pdf

Laukkanen, P. (2006). *Data-Driven and Keyword-Driven Test Automation Frameworks*. Obtenido el 7 de junio 2019 de: http://eliga.fi/Thesis-Pekka-Laukkanen.pdf

Manuel Gómez Martínez (2014). *Framework para el desarrollo ágil de aplicaciones*. Obtenido el 7 junio 2019 de: https://www.acens.com/wpcontent/images/2014/03/frameworks-white-paper-acens-.pdf

Monier, M. (2015). *Evaluation of automated web testing tools*. Obtenido el 7 de junio 2019 de: http://ijcat.com/archives/volume4/issue5/ijcatr04051014.pdf

Saluja, N. (2012). *Efficent Agent Based Testing Framework for Web Applications*. Obtenido el 7 de junio 2019 de: https://pdfs.semanticscholar.org/dff0/b5a27dff1180d16129e691584b93dfe343d8.pdf

Tonella, P. and Rica, F. (2008). *Dynamic model extraction and statistical analysis of Web applications: Follow-up after 6 years, 10th International Symposium on Web Site Evolution*. Obtenido el 7 de junio 2019 de: https://www.cs.du.edu/~sazghand/background_chap_papers/Analysis%20and%20testing%20of%20Web%20applications.pdf