Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: “Pruebas Automatizadas al Módulo Base de Datos Histórico del editor Phoenix en el SCADA UX”.

Autor: Yeilin Font Falt

Tutor: Ing. Mariela Cepero Núñez

Junio 2011

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yeilin Font Falt

_____

Firma del Autor

Mariela Cepero Núñez

_____

Firma del Tutor

DATOS DE CONTACTO

Mariela Cepero Núñez: Ingeniera en Ciencias Informáticas, perteneciente al Departamento Integración y Despliegue del Centro de Desarrollo de Informática Industrial. Profesor instructor con 4 años de experiencia docente y 5 años de experiencia en la producción.

AGRADECIMIENTOS

A mi esposo Yordankis.

A mi abuelo Elsido.

A mis padres Edilia y Papito.

A José Cardoso

A mis suegros Angel y Gricel.

A Lesky.

A mi tutora Mariela.

A toda mi familia.

A todo el que de una forma u otra me ayudó en mi formación.

DEDICATORIA

A mi esposo Yordankis, por ser mí guía.

A mis abuelos del alma Elsido, Ana y Julia.

A mis padres, por apoyarme en todo momento.

A mis hermanos Dariel y Orlandito.

A toda mi familia, por confiar siempre en mí.

RESUMEN

Hoy día en el mundo de la informática es sumamente importante liberar productos software con un gran nivel de eficiencia y calidad. En el Centro de Informática Industrial (CEDIN) de La Universidad de las Ciencias Informáticas (UCI), se desarrolla el proyecto SCADA UX de las siglas en inglés *Supervisory Control and Data Adquisition* (en español: Control, Supervisión y Adquisición de Datos), el cual está conformado por varios módulos dentro de los cuales se encuentra el de Base de datos Histórico (BDH), que se encarga de almacenar los cambios de estados dentro del sistema, partiendo de una configuración previa realizada por los administradores. Las pruebas que se realizan actualmente en el SCADA UX son de forma manual, por lo que se hace necesaria la automatización de las mismas para poder detectar los errores con una mayor eficacia, disminuir el tiempo de ejecución de las pruebas, los recursos destinados, así como la reutilización de estas pruebas automatizadas durante cada iteración. En el presente trabajo de diploma se propone la ejecución de este tipo de pruebas a la configuración del módulo BDH del SCADA UX, fundamentadas en un estudio de los temas referentes a pruebas de software, actividades y resultados que la conforman. Finalmente se desarrolló el proceso de pruebas logrando la automatización mediante el uso de una herramienta, se generaron los artefactos correspondientes y se analizaron los resultados obtenidos.

Palabras clave

base de datos, calidad, pruebas automatizadas, pruebas de software.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo de un país se mide en gran medida por el estado productivo que muestren sus industrias, entre ellas la informática juega un papel muy importante. El avance apresurado de las Tecnologías de la Información Científica (TIC) trae aparejado la creación de nuevos software que permitan automatizar procesos que hasta hoy se realizan de forma manual. La Universidad de Ciencias Informáticas (UCI) no está ajena a esto y pone sus mayores esfuerzos en el desarrollo de productos y servicios informáticos basados en la investigación de una temática, para convertirla en una rama productiva.

La práctica de verificar y gestionar la calidad durante el ciclo vital del proyecto es fundamental para conseguir los resultados esperados, ya que los problemas de software son mucho más difíciles de localizar y reparar después del despliegue. La calidad del software es definida por Pressman como: “concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente”. (1)

En concreto la calidad de un software es una serie de características que darán satisfacción al cliente y para lograrla se ejecutan un conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) cumpla con los requisitos dados de calidad, asociados a elementos tales como la funcionalidad, robustez, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad y escalabilidad. Entre las acciones que pueden llevarse a cabo para garantizar el cumplimiento de estos requisitos se encuentran las inspecciones, revisiones técnicas formales, auditorías, y pruebas de software.

Particularmente la aplicación de pruebas de software posibilita la detección de errores desde etapas tempranas del desarrollo de un producto. Este proceso se realiza comúnmente de forma manual sobre versiones (release) liberadas por el equipo de desarrollo, donde el equipo de probadores lleva a cabo un conjunto de tareas (testing) a fin de garantizar el correcto desempeño de las funcionalidades de la versión del producto. Luego de entregada una nueva versión se prueban nuevamente las funcionalidades presentes en la versión anterior y aquellas que se

agregaron para la segunda, con el objetivo de garantizar la correcta integración de las funciones del sistema.

El Centro de Informática Industrial (CEDIN) de la UCI, consciente de la importancia de las pruebas de software, concibió dentro de su estructura organizativa un Grupo de Pruebas de Software que brinda este servicio para todos los productos desarrollados en el centro. Entre los software que actualmente desarrolla el CEDIN está el SCADA UX.

SCADA es una "aplicación software de control de producción, que se comunica con los dispositivos de campo y controla el proceso de forma automática desde la pantalla del ordenador, proporcionando de esta forma información del proceso a diversos usuarios entre los que se encuentran los operadores, supervisores de control de calidad, supervisión, mantenimiento y otros" (2). Su finalidad es supervisar y controlar los procesos industriales.

Dentro de los módulos que conforman SCADA UX se encuentran los de Configuración, Seguridad, Reportes, Adquisición, Visualización o HMI, y BDH (Base de Datos Histórico); este último se encarga de "almacenar los cambios de estados dentro del sistema, entiéndase variables, alarmas, eventos, bitácoras, etc., partiendo de una configuración previa realizada por los administradores posibilitando el análisis posterior de estos registros y la generación de reportes a distintos niveles" (3). Este módulo es de suma importancia ya que la información almacenada en él es utilizada por una serie de aplicaciones entre las cuales se destacan los servicios gerenciales como la gestión de producción, mantenimiento y control.

Su configuración se realiza a través del módulo HMI, el cual se encarga de representar los procesos automatizados que ocurren en el campo en tiempo real y está dividido en dos partes, una es el Ambiente de ejecución y la otra es el Ambiente de configuración (editor Phoenix) que está conformado por interfaces gráficas de usuario (GUI) que permiten configurar todos los módulos del SCADA.

Las pruebas realizadas al módulo BDH actualmente se hacen de forma manual por los miembros del Grupo de Prueba del Departamento de Integración y Despliegue del CEDIN, dedicando muchos recursos humanos y tiempo a esta fase, dando lugar a la posibilidad de no detectar todos los errores existentes por el crecimiento y variabilidad en el código de dicho módulo, los mismos

pueden traer consigo desde la insatisfacción del cliente hasta resultados fatales en tiempo de ejecución. Por lo anteriormente descrito surge el **problema científico** ¿Cómo detectar errores en la configuración del módulo Base de Datos Histórico (BDH) con mayor precisión?

Constituye **objeto de estudio** de la presente investigación, las pruebas de software. Como **objetivo** se plantea: Desarrollar un componente que permita realizar pruebas automatizadas a la configuración del módulo Base de Datos Histórico (BDH) del SCADA UX.
Las pruebas que se automaticen permitirán la detección de errores en el sistema de forma temprana enmarcándose en las principales funcionalidades asociadas al módulo, siendo el **campo de acción** las pruebas automatizadas al módulo Base de Datos Histórico del editor Phoenix en el SCADA UX.
Para lograr el total cumplimiento del objetivo se trazan las siguientes **tareas investigativas:**

1. Estudio del estado del arte centrado en pruebas de software y automatización de pruebas.
2. Descripción de las herramientas y tecnologías que permiten el desarrollo del componente para la automatización de las pruebas al módulo BDH del editor Phoenix en el SCADA UX.
3. Definición del alcance de la solución a través de un plan de pruebas.
4. Diseño de los casos de pruebas.
5. Implementación del componente para realizar pruebas automatizadas.
6. Ejecución de las pruebas automatizadas.
7. Análisis de los resultados de las pruebas.
8. Validación de la efectividad de las pruebas automatizadas.

Métodos de investigación:
Para el desarrollo de esta investigación fue necesario utilizar algunos métodos teóricos y empíricos, que se listan a continuación:

Métodos Teóricos:

- ✓ **Analítico - Sintético:** permite analizar, estudiar e interpretar las teorías y documentos relacionados con las pruebas de software, incluyendo las automatizadas, con el fin de extraer los elementos más significativos.

Métodos Empíricos:

- ✓ Con el objetivo de seleccionar la información necesaria en la investigación a partir del estudio de documentos y diferentes bibliografías se utilizó el método **Revisión de la documentación**.
- ✓ Con el objetivo de analizar las opiniones y experiencias de los especialistas que han estado realizando pruebas en el CEDIN se utilizó el método **Entrevista.** Anexo 1

Este trabajo de diploma consta de tres capítulos de contenido, los mismos están estructurados de la siguiente manera:

En el capítulo 1, “Fundamentación Teórica”, se hace un análisis bibliográfico donde se investigan las características y funcionalidades de las pruebas automatizadas de software, se realiza un estudio de los beneficios que brindan las herramientas de automatización de pruebas, se describen las tecnologías utilizadas para la implementación del componente. Además analizan las definiciones y estructura del SCADA UX centrando la atención en la información referente a la configuración del módulo de Bases de Datos Histórico (BDH).

En el capítulo 2, “Descripción de la solución”: Se describe la implementación de la solución que se propone para la automatización de las pruebas a las funcionalidades del módulo BDH en el editor Phoenix, a partir de la definición del plan de pruebas y los diseños de casos de pruebas, concluyendo con los elementos asociados a la implementación de la prueba automatizada.

En el capítulo 3, “Análisis de resultados”, se exponen los resultados de las pruebas ejecutadas, mostrando la cantidad de escenarios que se ejecutaron satisfactoriamente y los que por el contrario resultaron insatisfactorios y por lo tanto generaron no conformidades. También se valora la importancia de las pruebas automatizadas realizadas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

La fase de pruebas es considerada una de las más costosas del ciclo de vida de un software. En sentido estricto, deben realizarse pruebas de todos los artefactos generados durante la construcción de un producto, lo que incluye (siguiendo la metodología RUP) las especificaciones de requisitos, casos de uso, diagramas de diversos tipos y por supuesto, el código fuente y el resto de los elementos que forman parte de la aplicación. Obviamente, se aplican diferentes técnicas de prueba a cada tipo de producto software.

1.1 Pruebas de software

En el proceso de desarrollo de un software intervienen dinámicamente la realización de pruebas, para así garantizar que el producto sea en gran medida confiable y funcional. La importancia de las pruebas se ve reflejada en el siguiente planteamiento de Elfriede Dustin**: "**Las pruebas de software permiten pasar de forma confiable del cómodo ambiente planteado por la ingeniería de software, es decir del controlado ambiente de análisis, diseño y construcción, al exigente mundo real en el cual los entornos de producción someten los productos a todo tipo de fatiga." (4)

Algunas definiciones reconocidas se muestran a continuación:

- ✓ **Pressman:** Las Pruebas de software pueden definirse como "el proceso de evaluación de un producto desde un punto de vista crítico. Es necesario probar los nuevos programas en un entorno de pruebas separado físicamente del de producción. (5)

- ✓ **IEEE, 1990:** Una prueba puede ser: "Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente". (6)

- ✓ **RUP:** Una prueba es “una disciplina en el proceso de ingeniería de software cuyo objetivo es integrar y poner a prueba el sistema”. No es posible garantizar que un sistema esté correcto solo usando pruebas. Así que aunque se hagan esfuerzos extraordinarios, no podemos decir que el sistema está libre de defectos. (7)

Luego de haber analizado estos conceptos se concluye que la prueba es un elemento crítico para la garantía de la calidad del software antes de ser entregado al cliente, pero no puede asegurar la ausencia de errores en su totalidad, sólo puede demostrar que existen defectos. La detección temprana de defectos como resultado de las pruebas, aumenta la probabilidad de corregirlos antes de que el producto esté en manos del cliente final y por lo tanto lograr una mayor satisfacción del mismo.

Las pruebas según sus características se clasifican en distintos niveles que serán descritos a continuación.

1.2 Niveles de pruebas

El proceso de prueba es llevado a cabo en varios niveles, cada uno es realizado en un determinado momento del ciclo de desarrollo del software. Se distinguen los siguientes niveles de pruebas:

1. **Prueba de unidad:** La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo (5). Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera.

2. **Prueba de integración:** La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger los

módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. (5)

3. **Prueba de validación:** Tras la culminación de la prueba de integración, el software está completamente ensamblado como un paquete, se han encontrado y corregido los errores de interfaz y puede comenzar una serie final de pruebas del software: la prueba de validación. La validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente. Este tipo de prueba está enfocada en el nivel de requisitos, o sea se valida que el software cumpla con las especificaciones solicitadas. Esta validación puede realizarse por parte de los desarrolladores o del propio cliente, en cuyo caso se denominan pruebas de aceptación y pueden ejecutarse tanto en el ambiente de desarrollo como en el entorno real donde se implantará el software.

4. **Prueba de Sistema:** La prueba del sistema, realmente, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora (5). Son las pruebas que se hacen cuando el software está funcionando como un todo, dirigidas a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

Dentro del nivel de Sistema se encuentran los siguientes tipos de pruebas: (5)

- ✓ **Prueba de Recuperación:** Es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente.

- ✓ **Prueba de Seguridad:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán.

- ✓ **Prueba de Resistencia:** Están diseñadas para enfrentar a los programas con situaciones anormales.

- ✓ **Prueba de Rendimiento:** Está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

Luego de haber analizado los distintos niveles de pruebas se concluye que cuando se está probando un software es muy importante tener bien definido el tipo de prueba que se le va a aplicar para así obtener los resultados esperados. Según RUP un software se va perfeccionando mediante las iteraciones que se van realizando durante su ciclo de vida en varios niveles o escenarios de trabajos, que permiten probar el producto desde la menor unidad creada hasta que se ha finalizado la construcción. Para la ejecución de las pruebas se tienen en cuenta además los métodos de prueba.

1.3 Métodos de pruebas de software

Dependiendo de los objetivos de prueba trazados, tiempo y recursos disponibles, nivel de prueba que se desee ejecutar y si se tiene acceso o no al código de la aplicación, se determina el método de prueba a emplear, pudiendo ser de caja blanca o de caja negra, sus principales características se abordan a continuación:

1.3.1 Prueba de caja blanca

La prueba de caja blanca se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar. (8)

La prueba de caja blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de caja blanca el ingeniero del software puede obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de caja blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad.

1.3.2 Prueba de caja negra

La prueba de caja negra se enfoca esencialmente en la interfaz del software. Los casos de prueba que se aplican tienen como objetivo probar la funcionalidad del software mediante la aceptación de la entrada y el nivel de corrección de la salida, así como la integridad de la información externa. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema, sin tener mucho en cuenta la composición interna o código del software.

Estas pruebas de funcionalidad permiten concebir un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa, ignorando las estructuras de control y concentrándose únicamente en los requisitos funcionales del sistema, de igual manera se establecen las salidas esperadas (tanto para casos válidos como inválidos).

La prueba de caja negra no es una alternativa a las técnicas de prueba de Caja Blanca, sino un complemento que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.

Dentro de los errores comúnmente encontrados durante los procesos de pruebas de caja negra se encuentran:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para guiar un proceso es necesario emplear alguna metodología de desarrollo, en el epígrafe siguiente se hará un análisis para seleccionar la que sea más conveniente para la elaboración del procedimiento.

1.4 Metodologías de Desarrollo de Software

Todo desarrollo de software es riesgoso y difícil de controlar, por lo que es recomendable seguir una metodología o procedimiento bien definido, evitando así que los resultados provoquen insatisfacción, no solo en los equipos de proyectos sino también en los clientes.

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. Por lo general definen actividades, responsables y productos de trabajo, para cada etapa del desarrollo del software, incluyendo la etapa de pruebas.

En el proceso de desarrollo de software actual es muy común el uso de metodologías ágiles, que brindan a los equipos de trabajo pequeños, herramientas factibles para el desarrollo de productos informáticos de pocas o medianas exigencias ya que no demandan la elaboración de muchos artefactos descriptivos y se centran en la implementación. Sin embargo el grupo de trabajo del proyecto SCADA UX es un equipo grande donde el cliente no es específicamente parte del mismo, además la complejidad del propio proceso de desarrollo requiere una documentación más formal; por tales razones se demanda el uso de una metodología tradicional, siendo aplicada la metodología **Rational Unified Process (RUP)**.

1.4.1 RUP

Debido a que el SCADA UX utiliza la metodología RUP se decidió mantener la misma para llevar a cabo el desarrollo del componente de prueba, a continuación se describen algunas características de esta metodología.

Fases:

RUP divide el desarrollo de software en 4 fases:

1. **Conceptualización (Concepción o Inicio):** Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.
2. **Elaboración:** Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.
3. **Construcción:** Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene 1 o varios release del producto que han pasado las pruebas. Se ponen estos release a consideración de un subconjunto de usuarios.
4. **Transición:** El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores

Adicionalmente RUP define Flujos de trabajo, los cuales tienen mayor o menor influencia en cada una de las fases, tal como lo muestra la siguiente figura.

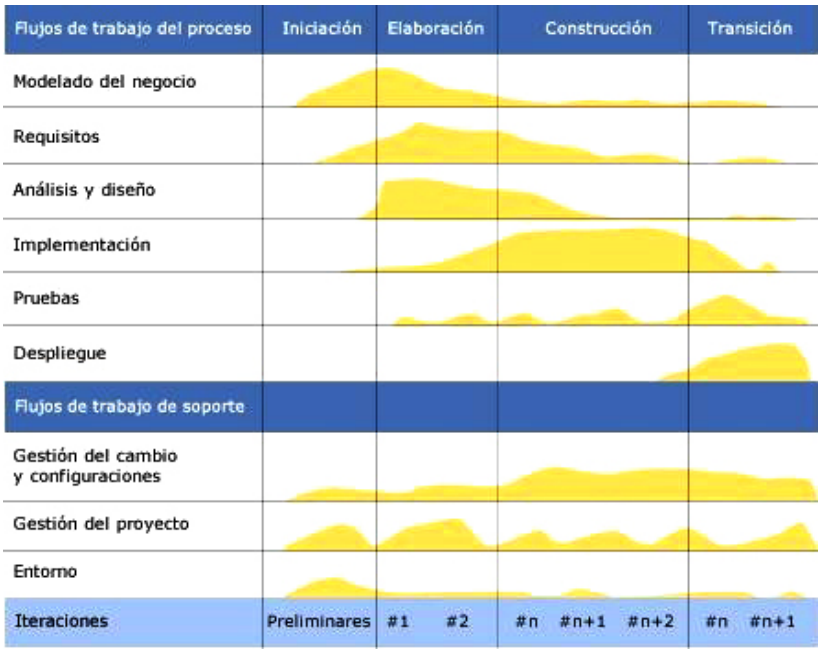

Figura 1. 1 Fases e Iteraciones de la Metodología RUP

A continuación se tratan brevemente los principales elementos que conforman cada flujo de trabajo.

- ✓ **Modelado del negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- ✓ **Requisitos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- ✓ **Análisis y diseño:** Describe cómo el sistema será realizado a partir los requerimientos especificados, por lo que indica con precisión lo que se debe programar.
- ✓ **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- ✓ **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.

- ✓ **Despliegue:** Produce “release” del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

También define los llamados flujos de soporte, entre los que se encuentran gestión de proyecto, gestión de cambios y configuración y el flujo de entorno.

RUP facilita el desarrollo de pruebas a lo largo del ciclo de vida. Los requisitos funcionales obtenidos durante el flujo de trabajo del mismo nombre, determinan la funcionalidad del sistema mediante la elaboración de casos de uso, lo cual facilita el diseño de casos de prueba y automatización de las mismas; es iterativo e incremental lo que posibilita que las pruebas no se apliquen al final del proyecto ya que en cada iteración puede ser aplicada al menos una iteración de pruebas, tiene el proceso de pruebas bien definido, generando gran cantidad de artefactos lo que permite documentar totalmente las actividades, detallando los roles que serán responsables durante este proceso, considerándose los criterios antes expuestos como suficientes para realizar su selección como metodología que da soporte completo al proceso de pruebas de software.

1.5 Artefactos de prueba

Un artefacto: (según RUP) no es más que los productos tangibles del proyecto que son producidos, modificados y usados por las actividades, pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables, generados por la metodología utilizada.
Las pruebas de software se realizan con el apoyo de varios artefactos. Igualmente se generan otros como resultado de la ejecución de estas pruebas tales como:

- Plan de prueba (Test plan).
- Matriz de trazabilidad (Traceability matrix).
- Caso de prueba (Test case).
- Script de prueba (Test script).
- Documento de No Conformidades.
- Colección de casos de prueba (Test suite).
- Datos de prueba (Test data).

Por sus características, la información que recogen y la experiencia adquirida por los especialistas del Grupo de Pruebas del Centro de Informática Industrial, se consideran significativos para esta investigación los descritos a continuación:

1.5.1 Plan de pruebas

Las pruebas deben planificarse por adelantado y llevarse a cabo sistemáticamente, es por ello que uno de los artefactos elementales es el Plan de Pruebas (PP) en el que se incluyen el propósito y alcance del proceso de pruebas, qué requerimientos se van a probar, las herramientas a utilizar, los recursos a emplear, reflejando las características de hardware y software, el cronograma, así como la documentación que será entregado.

1.5.2 Casos de pruebas

Una de las principales herramientas con que cuenta un probador para realizar las pruebas son los casos de prueba(CP), ya que mediante ellos, se formalizan las pruebas, describiendo todo lo que puede implicar llevarlas a cabo y sugiriendo los resultados que éstas deberían arrojar. Los CP y su reutilización juegan un papel fundamental a la hora de hacer que un proceso de pruebas sea eficiente y cumpla con los tiempos que el mercado demanda.

1.5.3 Procedimiento de prueba

El procedimiento específica cómo realizar uno o varios casos de pruebas ó partes de éstos. Puede ser una instrucción para un individuo sobre cómo realizar un caso de prueba manualmente, ó una especificación de cómo interactuar manualmente con una herramienta de automatización de pruebas, para crear componentes ejecutables de prueba.

1.5.4 Script de prueba

Automatiza uno o varios procedimientos de prueba o partes de ellos. Este término surgió producto a las herramientas de pruebas de regresión automatizadas. Se pueden desarrollar

utilizando un lenguaje de guiones o un lenguaje de programación, o pueden ser desarrollados con una herramienta de automatización de pruebas.

1.5.5 Datos de prueba

Los datos de prueba son valores que serán utilizados para la ejecución de las mismas. Se deben generar datos de prueba que representen entradas válidas e inválidas y debe existir constancia a partir de su almacenamiento, de igual manera los datos de prueba se revisan antes de cada iteración de prueba.

1.5.6 Documento de no conformidades

Se refiere a un documento redactado por el probador durante el proceso de pruebas a una determinada versión del software entregada por el equipo de desarrollo. En este artefacto se recogen las anomalías detectadas durante las pruebas.

Considerando lo anteriormente expuesto se puede concluir que la realización del proceso de pruebas de un software está guiado por el plan de prueba (PP), de cuya correcta confección depende en gran medida el éxito del proceso. El probador tendrá a su disposición los diseños de casos de pruebas (CP) que constituyen herramientas utilizadas para formalizar dichos test, basándose en los casos de usos, describiendo como deben desarrollarse y sugiriendo cuales son los resultados esperados. Los diseños de CP agilizan el trabajo a la hora de realizar las pruebas ya sea de forma manual o automatizada y deben contener datos de prueba. En el caso de las pruebas automatizadas los propios escenarios de prueba diseñados, guían la elaboración de los script de pruebas (SP), los cuales son determinantes para la correcta ejecución de las pruebas automatizadas. Finalmente los resultados de las pruebas se registran en el documento de no conformidades que es entregado al equipo de desarrollo, quien se encargará de solucionar los defectos detectados y conformar la próxima versión del producto que será probada en la siguiente iteración de prueba.

1.6 Programa de mejora

La UCI está inmersa en un programa de mejora desde finales del año 2008, basado en el modelo CMMI (Capability Madurity Model Integration) con el objetivo de: “Mejorar los procesos, métodos, tecnología y la calidad de los proyectos a partir de la incorporación de buenas prácticas propuestas por el modelo CMMI.”, se pretende inicialmente alcanzar el nivel 2 establecido por el modelo.

El CEDIN es uno de los centros implicados directamente en este programa, de ahí la necesidad de cumplir con las políticas establecidas, asociadas a cada una de las 7 áreas de proceso de obligatorio cumplimiento en dicho nivel y por lo tanto ajustarse a los artefactos y roles definidos, que serán evaluados; por este motivo la documentación que se genera durante esta investigación utilizará plantillas propuestas por el Programa de Mejoras.

1.7 Pruebas manuales

Las pruebas manuales son el método de pruebas de software más antiguo y riguroso. Requieren de un probador que ejecute de manera manual operaciones en la aplicación sin ayuda de herramientas de automatización. Por lo que siempre que sea posible se sugiere que sean llevadas a cabo por personas pacientes, observadoras, creativas e innovadoras. Este tipo de pruebas detecta cualquier problema relacionado con la funcionalidad del producto, especialmente defectos vinculados con la usabilidad y la interfaz gráfica de la aplicación.

1.8 Pruebas automatizadas

Las pruebas constituyen un tema polémico, no son parte de la solución ni son entregadas a los clientes. Sin embargo, por la importancia que se les confiere en el desarrollo de software de Calidad, se hace necesario tratar de encontrar vías que permitan enfrentar esta etapa de forma rápida, económica y detectando la mayor cantidad errores posibles, lo cual puede lograrse mediante la automatización.

¿Qué significa automatizar?

Automatizar es realizar procesos o tareas de manera automática mediante herramientas de software.

“Las pruebas automatizadas de software han sido consideradas fundamentales para las grandes organizaciones de desarrollo de software. Las compañías más exitosas en el mundo de la informática hacen uso de ellas, ejemplo son: Corel, Intel, Adobe, Autodesk, Intuit, McDonald's, Motorola, Sony Symantec”. (9)

Ventajas de la automatización:

Las pruebas automatizadas se consideran ventajosas ya que se caracterizan por ser:

- ✓ **Confiables:** Las pruebas realizan con exactitud diversas operaciones cada vez que se ejecutan, de tal modo que evita posibles errores humanos.
- ✓ **Recursivas:** Evidencian cómo el software reacciona bajo diferentes condiciones, cuando se está probando una misma operación repetidamente.
- ✓ **Programables:** Permiten programar pruebas sofisticadas que pongan en evidencia la robustez del software que se pruebe.
- ✓ **Abarcadoras:** Facilitan la construcción de un ambiente de pruebas que cubra cada característica del software.
- ✓ **Reutilizables:** Permite reutilizar pruebas en diversas versiones.
- ✓ **Factibles:** Se pueden ejecutar más pruebas en menos tiempo y el número de los recursos se reduce.
- ✓ **Rápidas:** Su ejecución es perceptiblemente más rápida.
- ✓ **Flexibles:** Las pruebas deben ser fáciles de entender, de modificar y de extender.
- ✓ **Se evitan pruebas obsoletas.**

Desventajas de la automatización:

- ✓ Muy pocas herramientas
- ✓ Necesitan una “base” a menudo inexistente.
- ✓ En ocasiones se gasta más en la automatización que en la pruebas en sí.

- ✓ En ocasiones pueden ser difíciles de mantener.

En este sentido es imprescindible utilizar herramientas de automatización cuyo uso no implique obviar los procedimientos de prueba establecidos, al contrario, deben tenerse en cuenta los planes y diseños de casos de prueba, así como la existencia de modelos y documentación del sistema, que posibiliten automatizar las pruebas eficientemente. Lamentablemente las herramientas ejecutan únicamente scripts diseñados para testear un requisito o funcionalidad específicos y no poseen la habilidad de la “toma de decisiones” ni grabación de discrepancias no incluidas en el script, (10) también ha de tenerse en cuenta la necesidad real de utilizar herramientas, ya que no siempre es factible automatizarlo absolutamente todo.

1.9 Herramientas actuales para pruebas automatizadas

En la actualidad existe una gran variedad de herramientas para realizar pruebas automatizadas, éstas proporcionan una ventaja evidente partiendo de tiempo y conservación de los recursos. El uso de las mismas permite tener más comodidad en la detección de los errores y fallos que se producen antes de que el producto de software sea liberado, o en momentos en que haya necesidad de proporcionar asistencia al cliente para resolver los defectos encontrados. A continuación se mencionan algunas de estas herramientas:

- ✓ **IBM Rational Functional Tester:** permite a los probadores y a los desarrolladores automatizar las pruebas funcionales y la regresión de aplicaciones Java, .NET y basadas en Web. Proporciona a los probadores con poca experiencia funciones automatizadas para actividades como, por ejemplo, la generación de pruebas dirigidas por datos. Incorpora soporte para el control de la versión para permitir un desarrollo paralelo de los scripts de verificación y el uso simultáneo por parte de equipos distribuidos por el mundo.

- ✓ **IBM Rational Manual Tester:** es una herramienta de ejecución y automatización de pruebas manuales para probadores y analistas de negocio. Promueve la reutilización de

los pasos de pruebas para reducir la repercusión de los cambios de software realizados en las actividades de mantenimiento de pruebas. Proporciona un editor de textos avanzado que soporta adjuntos e imágenes para mejorar la legibilidad de las pruebas. Facilita la entrada de datos y verificación durante la ejecución de pruebas para reducir los posibles errores humanos. Incorpora los requisitos de equipos exclusivos a través de numerosas opciones de personalización; permite compartir contenido entre ubicaciones de pruebas distribuidas.

- ✓ **IBM Rational Test RealTime:** Permite pruebas de componentes y el análisis de la ejecución para el software incrustado y la plataforma cruzada en tiempo real. Diseñado específicamente para aquellos que escriben código para los productos incrustados u otros tipos de productos de sistema informático ubicuo. Soporta las aplicaciones incrustadas seguras más importantes del negocio.

- ✓ **Herramienta de pruebas automatizadas para el editor Phoenix:** Es un producto desarrollado en la Universidad de las Ciencias Informáticas por el equipo de trabajo del SCADA-UX del CEDIN, luego de haber analizado varias herramientas de pruebas, y comprobar que no se ajustaban a sus necesidades. Surge así el namespace PhoenixTest que no es más que una nueva versión del framework QtestLib que incorpora nuevas funcionalidades y algunas modificaciones relacionadas con el filtrado de eventos y el mecanismo para la sincronización de eventos. El objetivo es optimizar el proceso de pruebas, minimizando tiempo y recursos.

Esta última fue escogida para desarrollar las pruebas automatizadas del configurador de BDH, por ser recomendada por el equipo de desarrollo del SCADA UX, haber sido implementada totalmente sobre tecnologías libres, además de estar integrada al editor Phoenix facilitando cargar y probar cada módulo de manera independiente, adicionalmente posee las siguientes características:

- ✓ Proporciona una interfaz de prueba sencilla e intuitiva para los usuarios facilitando su utilización.
- ✓ Brinda la totalidad de funcionalidades para identificar y simular eventos asociados al movimiento del ratón, junto con las pulsaciones del teclado.
- ✓ Distingue los eventos generados por las pruebas automatizadas, de los generados por la interacción del usuario.
- ✓ Brinda total cobertura al trabajo con la arquitectura orientada a plug-ins del editor posibilitando la automatización de las pruebas de todos los módulos de SCADA por separado.
- ✓ Permite personalizar las pruebas para que puedan adaptarse ante la introducción de nuevos cambios en el código.
- ✓ Soporta pruebas tanto de caja blanca como de caja negra.
- ✓ Cobertura de código para C y C++ permitiendo la programación orientada a objetos.
- ✓ Integrada completamente al entorno de desarrollo integrado eclipse y permite el trabajo con el framework QT.

1.10 Tecnologías

Para definir las tecnologías y lenguaje a utilizar en el desarrollo del componente de pruebas automatizadas, se tuvieron en cuenta los estándares tecnológicos utilizados en la herramienta de pruebas previamente seleccionada y que se ajustan a los definidos en el proyecto SCADA. Además se adoptan las políticas de software asumidas por el CEDIN, que establecen la utilización de software libre para el desarrollo de las aplicaciones.

1.10.1 Entorno de Desarrollo Integrado

El Eclipse es un Entorno de Desarrollo Integrado (IDE) multiplataforma para crear aplicaciones clientes de cualquier tipo. Este IDE emplea módulos para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. El mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java, por ejemplo existe un módulo para dar soporte a C/C++.

Eclipse es uno de los IDE de código abierto y extensible más utilizados en los últimos tiempos, empleado para desarrollar la herramienta para la automatización de las pruebas, y en general en el CEDIN. Es completamente neutral a la plataforma y al lenguaje, siendo un IDE gratuito, libre, potente, y tiene gran soporte en la comunidad de software libre.

1.10.2 Lenguaje de programación

Como lenguaje de programación se utilizará el **C++**, coincidiendo con el utilizado no solo para la implementación del SCADA-UX, sino también por la herramienta de pruebas automatizadas seleccionada. **C++** es un potente lenguaje de programación que surgió en 1980, continuando con las ventajas, flexibilidad y eficacia del C. Se trata de un lenguaje estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, de propósito general, y ampliamente utilizado tanto en el ámbito profesional como en el educativo.

Las principales características del C++ son el soporte para programación orientada a objetos, el soporte de plantillas o programación genérica (templates), la portabilidad, brevedad, programación modular, compatibilidad con C y velocidad. Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. (11)

1.10.3 Lenguaje de modelado

El lenguaje de modelado es la notación (principalmente gráfica) utilizada para expresar un diseño. UML (Lenguaje Uniticado de Construction de Modelos) nació como una notación estándar de la construcción de modelos. (12) Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. (11) Será utilizado porque soporta la metodología RUP definida con anterioridad, ayuda a la captura de decisiones y conocimiento sobre los sistemas que se deben construir y se usa para entender, diseñar, configurar, mantener, y controlar la información sobre tales sistemas.

1.10.4 Herramienta Case

Las herramientas CASE facilitan la automatización de las actividades del ciclo de vida del software. Entre ellas se destaca el Visual Paradigm (VP) ya que es una herramienta UML profesional, que soporta el ciclo de vida completo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Se utilizará VP ya que es una herramienta multiplataforma muy potente e intuitiva, que facilita la creación de diagramas en un tiempo relativamente corto. Permite documentar con mayor exactitud todo el trabajo efectuado, realizar un diseño centrado en casos de uso y enfocado al negocio, modelar el sistema evidenciándose en los modelos de análisis y diseño que se presenta, todo esto permitiendo obtener exitosos resultados.

Otras características que la distinguen se exponen a continuación. (11)

- Ingeniería inversa - código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código - modelo a código, diagrama a código.
- Diagramas de flujo de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

1.11 Sistema SCADA UX

Como se mencionaba anteriormente, SCADA es un término que se utiliza para describir la supervisión, control y gestión de soluciones en una amplia gama de industrias, como son, los

sistemas de gestión de agua, energía eléctrica, señales de tráfico, sistemas de control ambiental, y sistemas de fabricación. (13)

El SCADA UX desarrollado en el CEDIN, está compuesto por varios módulos dentro de los cuales se encuentran el de BDH y el de Visualización. Este último permite interactuar con el sistema a través de una Interfaz Hombre Máquina (HMI), la cual está compuesta por Interfaces Gráficas de Usuario (GUI). Todos los módulos del SCADA están estrechamente relacionados, cumpliendo cada uno determinadas responsabilidades en el sistema, a continuación se abordarán algunos elementos del módulo encargado de la persistencia de datos.

1.11.1 Módulo BDH

Un SCADA es por excelencia un concentrador de datos y tareas, que muestran a los operadores de la planta el estado actual de los procesos. Es por ello que toda esa información se debería mantener en un registro histórico de manera tal que puede servir entre otras cosas para:

- ✓ Análisis de comportamientos y tendencias.
- ✓ Auditorias a la operación del sistema.
- ✓ Identificación y modelado de procesos.
- ✓ Análisis postmortem de fallas, etc.

El módulo de Base de Datos Histórica (BDH) cubre las tareas relacionadas con la persistencia de datos con carácter histórico, enmarcando tanto el almacenamiento de valores, como las relacionadas con el servicio de los mismos a los demás módulos del sistema que así lo requieran.

Servicios brindados por el módulo: (13)

- ✓ **Servicio de histórico de variables (puntos):** Almacenamiento planificado de las muestras de los puntos, en función de parámetros como la frecuencia de envío hacia los históricos o la ocurrencia de una excepción.

- ✓ **Servicio de histórico de eventos**: Almacenamiento de la ocurrencia de eventos que permitan realizar una trazabilidad del estado del sistema en el tiempo, por ejemplo caídas de módulos o usuarios intentando acceder a recursos protegidos.

- ✓ **Servicio de históricos de bitácora:** Información introducida por los operadores del sistema, acerca de situaciones que ocurrieron y que podrían repetirse a las cuales típicamente se les describe la solución dada para tenerla de referencia en el futuro.

Para que el módulo de BDH cumpla sus funciones correctamente es necesario configurarlo, lo cual se hace en el ambiente de configuración (editor Phoenix) del módulo HMI, específicamente en la extensión dedicada a BDH, que es la que precisamente será probada como resultado de la presente investigación.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN

Introducción

El presente capítulo tiene como propósito describir la solución técnica para el desarrollo de las pruebas automatizadas a la configuración del módulo Base de Datos Histórica del editor Phoenix del proyecto SCADA UX, en el mismo se describen los principales elementos de la solución del problema comenzando por la descripción del proceso de pruebas a utilizar, la definición de Plan y Diseños de prueba, así como los casos de uso que darán solución a la prueba automatizada.

2.1 Proceso de pruebas

Para llevar a cabo las pruebas automatizadas se desarrolló un proceso de prueba que consta de 6 etapas:

- ✓ **Planificar las pruebas:** En esta primera etapa se define el alcance y objetivo de las pruebas, roles involucrados, estrategia de prueba a seguir, entre otros aspectos que guían el desarrollo de las pruebas. Su principal salida es el plan de pruebas, que se encuentra en el epígrafe 2.2.
- ✓ **Diseñar las pruebas:** El diseño de las pruebas organiza el trabajo que debe realizarse, mediante la definición de diferentes escenarios de prueba que permitan encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo. La salida de esta fase son los casos de prueba, que incluyen además la especificación de datos de prueba. Una descripción más detallada se encuentra en el epígrafe 2.3.
- ✓ **Implementar el componente de pruebas automatizadas:** Luego de diseñar las pruebas se procede a la automatización de las mismas mediante la herramienta de pruebas seleccionada, incorporándole los script de pruebas que responden a los escenarios de los casos de prueba. Una descripción más detallada se encuentra en el epígrafe 2.4.
- ✓ **Ejecutar las pruebas:** Se prueban las funcionalidades del configurador de BDH, mediante la ejecución del componente de pruebas implementado epígrafe 2.5.
- ✓ **Analizar los resultados:** En esta última fase se analizan los resultados de las pruebas teniendo en cuenta los errores detectados, la cantidad de pruebas que fallaron y las que se ejecutadas satisfactoriamente. Esta fase se aborda en el capítulo 3.

2.2 Plan de pruebas

El plan de pruebas que se utiliza en el CEDIN se compone fundamentalmente por: los roles y responsabilidades, la estrategia de pruebas que se pretende aplicar donde se definen los métodos de pruebas, el alcance y el entorno (se especifica los requerimientos de Hardware y Software), una descripción de los requisitos y los casos de uso a probar**.** A continuación se describe el plan que guía las pruebas de la configuración del módulo BDH del SCADA UX.

2.2.1 Roles y responsabilidades

Para llevar a cabo las pruebas será necesario el desempeño de dos roles, los cuales son definidos en el programa de mejora y participan directamente en las pruebas de software, siendo estos ejecutados en este caso particular, por una misma persona.

El **diseñador de pruebas** participa en la confección de la estrategia y el plan de pruebas, identificando los métodos, herramientas y directrices apropiadas para implementar las pruebas necesarias. Es el encargado de establecer el ambiente de comprobación y diseñar los casos de pruebas (incluyendo datos de prueba), también debe analizar y evaluar el estado del producto de trabajo comprobado. Luego el **probador** siguiendo lo establecido en el plan de pruebas, ejecuta los casos de prueba y registra las no conformidades detectadas, analizando además los resultados de las pruebas.

2.2.2 Estrategia de pruebas

Para realizar las pruebas automatizadas al módulo BDH se ha seleccionado el nivel pruebas de unidad. Se utilizó como método de prueba el de caja negra ya que se definirán distintos valores de entradas y se probará si la salida corresponde a la funcionalidad descrita. El alcance de estas pruebas se enmarcará en probar las funcionalidades descritas en los casos de uso del configurador de BDH.

Entorno de Pruebas

Las pruebas han de realizarse en un entorno diferente al de desarrollo, contando con los recursos necesarios para que pueda ejecutarse el software (bibliotecas o librerías y el código de la versión de prueba), también debe tenerse acceso a la información y artefactos del proceso de pruebas.

Especificaciones de Hardware

La elaboración del componente exige la utilización de una computadora capaz de soportar el Editor Phoenix de ahí que se sugiere: Procesador Intel(R) Pentium(R) 4 con CPU a más de 1.8 GHz, memoria RAM de 512 MB y disco duro 80 GB.

Especificaciones de Software

Para llevar a cabo la implementación se utilizó el Sistema Operativo Linux en la distribución Ubuntu 10.04 LTS - versión Lucid Lynx ya que posibilita la instalación del framework de QT, no obstante se puede utilizar tanto la distribución Ubuntu como Debian obteniendo resultados satisfactorios.

2.2.3 Funcionalidades del módulo Base de Datos Histórico en el HMI

El Editor Phoenix está desarrollado con una arquitectura orientada a plugins siendo los mismos una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Los plugins son cargados por el editor mediante el mecanismo de extensiones que ofrece Qt y pueden ser: el inspector de propiedades, los componentes gráficos para la edición de la configuración de los módulos y los propios módulos del SCADA, entre ellos el de BDH. Al configurarlo el sistema debe gestionar elementos tales como, el propio módulo BDH, las bases de datos, y los grupos de transferencia históricos, que pueden ser digitales y analógicos, dichas funcionalidades fueron agrupadas en 4 casos de uso, descritos en el documento de especificación de casos de uso. (14) A continuación se muestran de manera explícita las funcionalidades de configuración del editor gráfico para el módulo Base de Datos Histórico que será necesario probar para un buen funcionamiento del mismo:

2.2.3.1 Administrar módulo BDH

Descripción de la Funcionalidad:

Esta funcionalidad permite la configuración del módulo BDH.

Flujo Central:

Inicia cuando el usuario pulsa clic derecho sobre un Nodo siguiendo la ruta Proyecto->Nodo.

Sección 1: Agregar Módulo BDH.

El usuario selecciona la opción Agregar Módulo, se incorpora el módulo al árbol de dependencia con sus recursos asociados guardándose la configuración en el sistema.

Sección 2: Eliminar Módulo sin recursos asociados

El usuario selecciona la opción Eliminar Módulo, el sistema muestra un mensaje de confirmación y se confirma la misma, eliminándose el módulo y actualizando el árbol de dependencias.

Sección 3: Eliminar Módulo con recursos asociados.

El usuario selecciona la opción Eliminar Base de Datos, el sistema muestra un mensaje de alerta indicando que tiene recursos asociados y no puede ser eliminado (debería eliminar primero los recursos), el usuario acepta el mensaje.

2.2.3.2 Administrar Base de Datos

Descripción de la Funcionalidad:

La funcionalidad permite agregar y eliminar una base de datos.

Flujo Central:

Inicia cuando el usuario pulsa clic derecho sobre la carpeta BDH siguiendo la ruta Proyecto->Nodo->BDH.

Sección 1: Agregar Base de Datos

El usuario selecciona la opción Adicionar Base de Datos. El sistema muestra la ventana de propiedades de la base de datos con los campos a llenar: nombre y descripción y se pulsa aceptar, se comprueba que el campo nombre no esté vacío (Alterno 1), contenga menos de 32 caracteres (Alterno 2) y que no exista otra base de datos con el mismo nombre (Alterno 3),

guardándose la configuración y actualizando el árbol de dependencias terminando así el caso de uso.

Flujo alterno 1.1: Validar que el campo nombre no esté vacio

El usuario pulsa Aceptar en la ventana de propiedades de la base de datos sin llenar el campo nombre, el sistema muestra un mensaje de error indicando que el campo no puede estar vacío.

Flujo alterno 1.2: Validar que el campo nombre no contenga más de 32 caracteres

El usuario intenta insertar más de 32 caracteres en el campo nombre de la ventana de propiedades de la base de datos. El sistema muestra un mensaje de error indicando que el campo no debe contener más de 32 caracteres.

Flujo alterno 1.3: Validar que no existan dos bases de datos con el mismo nombre

El usuario inserta en el campo nombre de la ventana de propiedades de la base de datos, un nombre que ya está asignado a otra base de datos y pulsa aceptar, el sistema muestra un mensaje de error indicando que existe una base de datos con ese nombre.

Sección 3: Eliminar Base de Datos sin recursos asociados.

El usuario selecciona la opción Eliminar Base de Datos, el sistema muestra un mensaje de confirmación y se confirma la misma, eliminándose la base de datos y actualizando el árbol de dependencias

Sección 4: Eliminar Base de Datos con recursos asociados.

El usuario selecciona la opción Eliminar Base de Datos, el sistema muestra un mensaje de alerta indicando que tiene recursos asociados, y no puede ser eliminada (debería eliminar primero los recursos), el usuario acepta el mensaje.

2.2.3.3 Administrar grupo de Transferencia Digital

Descripción de la Funcionalidad:

La funcionalidad permite agregar, eliminar y modificar un grupo de transferencia digital.

Flujo Central:

Inicia cuando el usuario pulsa clic derecho sobre la carpeta Grupo Digitales siguiendo la ruta Proyecto->Nodo->BDH->Base de Datos->Grupo Digitales.

Sección 1: Agregar grupo de transferencia digital.

El usuario selecciona la opción Nuevo Grupo Digital. El sistema muestra la ventana de propiedades del grupo digital con los campos a llenar: nombre, descripción, tipo de almacenamiento (no jerárquico por defecto), tipo de transferencia (por muestro o por excepción), frecuencia de muestreo y banda muerta. Se verifica que los campos nombre, frecuencia de muestro (solo números) y banda muerta (solo números) no estén vacios (Alterno 1), demás no deben existir dos grupos con el mismo nombre (Alterno 2), y debe tener menos de 32 caracteres dicho campo (Alterno3). Se llenan los mismos y se pulsa aceptar, guardándose la configuración y actualizando el árbol de dependencias terminando así el caso de uso.

Flujo alterno 1.1: Validar que los campos nombre, frecuencia de muestreo y banda muerta no estén vacios

El usuario presiona la opción Aceptar en la ventana de propiedades del grupo digital sin llenar los campos: nombre, descripción, tipo de almacenamiento (no jerárquico por defecto), tipo de transferencia (por muestro o por excepción), frecuencia de muestreo y banda muerta, el sistema se muestra un mensaje de error indicando que los campos no pueden estar vacíos.

Flujo alterno 1.2: Validar que el campo nombre no contenga más de 32 caracteres

El usuario intenta insertar más de 32 caracteres en el campo nombre de la ventana de propiedades del grupo digital. El sistema muestra un mensaje de error indicando que el campo no debe contener más de 32 caracteres

Flujo alterno 1.3: Validar que no existan dos grupos digitales con el mismo nombre

El usuario inserta en el campo nombre de la ventana de propiedades del grupo digital, un nombre que ya esta asignado a otro grupo de transferencia digital y pulsa Aceptar, el sistema muestra un mensaje de error indicando que existe un grupo digital con ese nombre.

Sección 2: Modificar grupo de transferencia digital.

El operador selecciona la opción Editar Grupo de Transferencia, siguiendo la ruta Proyecto->Nodo->BDH->Base de Datos, el sistema muestra los iconos asociados a puntos digitales y puntos analógicos, el usuario da clic en el ícono del grupo digital, el sistema muestra la ventana de propiedades del punto digital con la información almacenada hasta ese momento en los

campos: nombre, descripción, tipo de almacenamiento (no jerárquico por defecto), tipo de transferencia (por muestro o por excepción), frecuencia de muestreo (solo números) y banda muerta (solo números). Se modifican los mismos guardándose automáticamente la configuración y actualizando el árbol de dependencias terminando así el caso de uso.

Sección 3: Eliminar grupo de transferencia digital.

El usuario selecciona la opción eliminar Grupo Digital, el sistema muestra un mensaje de confirmación y se confirma la misma, eliminándose el grupo digital y actualizando el árbol de dependencias.

2.2.3.4 Administrar grupo de Transferencia Analógico

Descripción de la Funcionalidad:

La funcionalidad permite agregar, eliminar y modificar un grupo de transferencia analógico.

Flujo Central:

Inicia cuando el usuario pulsa clic derecho sobre la carpeta Grupo Analógicos siguiendo la ruta Proyecto->Nodo->BDH->Base de Datos->Grupo Analógicos.

Sección 1: Agregar grupo de transferencia analógico.

El usuario selecciona la opción Nuevo Grupo Analógico. El sistema muestra la ventana de propiedades del grupo analógico con los campos a llenar: nombre, descripción, tipo de almacenamiento (jerárquico o no jerárquico), tipo de transferencia (por muestro o por excepción), frecuencia de muestreo (solo números) y banda muerta (solo números). Se verifica que los campos nombre, frecuencia de muestro y banda muerta no estén vacios (Alterno1), además no deben existir dos grupos con el mismo nombre (Alterno 3), y debe tener menos de 32 caracteres dicho campo (Alterno2). Se llenan los mismos y se pulsa aceptar, guardándose la configuración y actualizando el árbol de dependencias terminando así el caso de uso.

Flujo alterno 1.1: Validar que los campos nombre, frecuencia de muestreo y banda muerta no estén vacíos.

El usuario presiona la opción Aceptar en la ventana de propiedades del grupo analógico sin llenar los campos: nombre, descripción, tipo de almacenamiento (no jerárquico por defecto), tipo de

transferencia (por muestro o por excepción), frecuencia de muestreo y banda muerta, el sistema muestra un mensaje de error indicando que los campos no pueden estar vacíos.

Flujo alterno 1.2: Validar que el campo nombre no contenga más de 32 caracteres.

El usuario intenta insertar más de 32 caracteres en el campo nombre de la ventana de propiedades del grupo analógico. El sistema muestra un mensaje de error indicando que el campo no debe contener más de 32 caracteres

Flujo alterno 1.3: Validar que no existan dos grupos analógicos con el mismo nombre.

El usuario inserta en el campo nombre de la ventana de propiedades del grupo analógico, un nombre que ya está asignado a otro grupo de transferencia analógico y pulsa Aceptar, el sistema muestra un mensaje de error indicando que existe un grupo analógico con ese nombre.

Sección 2: Modificar grupo de transferencia analógico.

El operador selecciona la opción Editar Grupo de Transferencia, siguiendo la ruta Proyecto->Nodo->BDH->Base de Datos, el sistema muestra los iconos asociados a puntos analógicos y puntos digitales, el usuario da clic en el ícono del grupo analógico, el sistema muestra la ventana de propiedades del punto analógico con la información almacenada hasta ese momento en los campos: nombre, descripción, tipo de almacenamiento (no jerárquico por defecto), tipo de transferencia (por muestro o por excepción), frecuencia de muestreo (solo números) y banda muerta (solo números). Se modifican los mismos guardándose automáticamente la configuración y actualizando el árbol de dependencias terminando así el caso de uso.

Sección 3: Eliminar grupo de transferencia analógico.

El usuario selecciona la opción eliminar Grupo Analógico, el sistema muestra un mensaje de confirmación y se confirma la misma, eliminándose el grupo analógico y actualizando el árbol de dependencias.

De esta forma concluye la fase de planificación de las pruebas, siendo momento entonces de comenzar con los diseños de casos de prueba que respondan a las funcionalidades descritas anteriormente.

2.3 Casos de prueba

Los casos de pruebas se diseñaron teniendo en cuenta la plantilla que propone el Programa de Mejora de la Universidad (y que es la que utiliza el grupo de pruebas del CEDIN) en la que se especifica una descripción general del caso de prueba, condiciones de ejecución (precondiciones del caso de uso) y secciones del caso de prueba, en las cuales se detallan los elementos a probar referidos a los flujos básicos y alternos del los casos de uso. Fueron diseñados 4 casos de prueba (uno por cada caso de uso), que serán descritos a continuación, aunque pueden consultarse en la documentación adjunta a esta investigación:

- ✓ Gestionar Módulo BDH.

Este caso de prueba está dividido en 4 escenarios, dos de ellos referidos a la inserción del módulo BDH y los otros dos a su eliminación.

- ✓ DCP Gestionar Bases de Datos.

Este caso de prueba está dividido en 8 escenarios, en los que se verifica que el sistema permita la inserción y eliminación de bases de datos en el módulo BDH, cumpliendo condiciones tales como: no permitir que se inserten bases de datos con el mismo nombre, que hayan campos vacíos o que se introduzcan caracteres extraños.

- ✓ Gestionar Grupos de Transferencia Analógico y Gestionar Grupo de Transferencia Digital.

Ambos casos de prueba constan de 7 escenarios, 3 de ellos verifican la inserción correcta de los grupos de transferencia, garantizando que no se omita ni se repita el campo nombre, los otros se refieren a la eliminación y modificación de dichos grupos.

2.4 Implementar el componente de pruebas automatizadas:

La implementación del componente de pruebas automatizadas para probar la configuración del módulo BDH utiliza la arquitectura orientada a plugins del editor Phoenix y las funcionalidades del

namespace PhoenixTest, explotando así las ventajas que esta ofrece. En el núcleo de la aplicación se desarrolla la creación y manejo de las pruebas registrándolas en el editor que se encargará de cargar y manejar todas las pruebas contenidas en el plugin.

Para implementar el componente se llevaron a cabo los siguientes pasos:

Paso1: Reconocimiento por parte de la herramienta del plugin de BDH

La clase encargada de crear al plugins debe heredar de TestExtension, clase abstracta que permite que el plugins sea cargado y registrado con sus pruebas en el sistema. Se incorpora a la clase BDHPlugin la función void registerTests (Core::TestComposite*) encargada de registrar las pruebas del plugins, como se muestra a continuación:

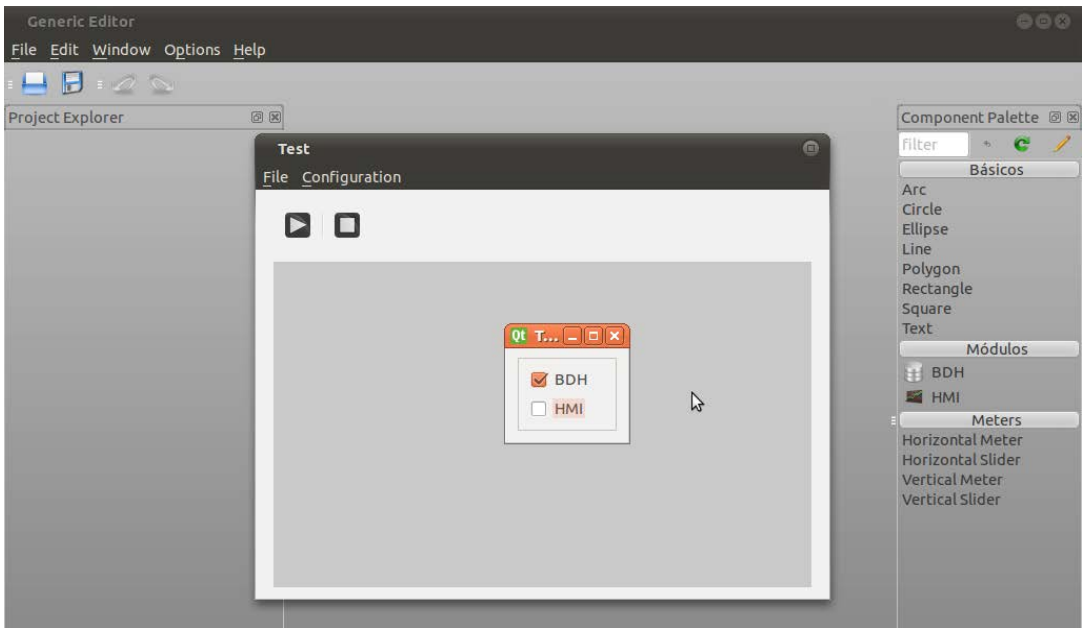

Figura 2. 1 Herramienta de Pruebas Automatizadas para el módulo HMI del editor Phoenix

Paso 2: Crear las propias pruebas del plugin BDH

Al crear las pruebas se deben simular primeramente las acciones del usuario sobre la aplicación, siendo estas funcionalidades brindadas por la herramienta de pruebas del Editor Phoenix. Luego se debe verificar si dichas acciones generan las respuestas esperadas por el sistema. Las clases que conforman el componente de pruebas se presentan a continuación:

Nombre de la clase	Descripción
BDHTest	Clase donde se manejan las pruebas al sistema, colecciona los casos de prueba.
BDHplugin	Clase que contiene la extensión de prueba que será cargada y registrada por el sistema.
SlotTimer	Clase que sincroniza el tiempo para controlar las ventanas que bloquean el resto de la aplicación.
DataCommand	Clase que controla la gestión de los grupos de transferencia, es una clase que sirve de apoyo a BDHTest.

Tabla 2. 1 Descripción de las clases

La relación que se establece entre estas clases se representa en el siguiente diagrama de clases, en el que también se visualizan los atributos y métodos contenidos en ellas. En el mismo se incluyen además las clases Test y TestExtencion que son brindadas por la herramienta de pruebas y sirven de base a sus clases hijas. A continuación se describen brevemente.

- ✓ **Test:** Clase abstracta de prueba, de la cual deben heredar todas las clases de pruebas construidas en el sistema.
- ✓ **TestExtension**: Clase abstracta, de la cual deben heredar las extensiones de pruebas que quieran ser cargadas y registradas en el sistema.

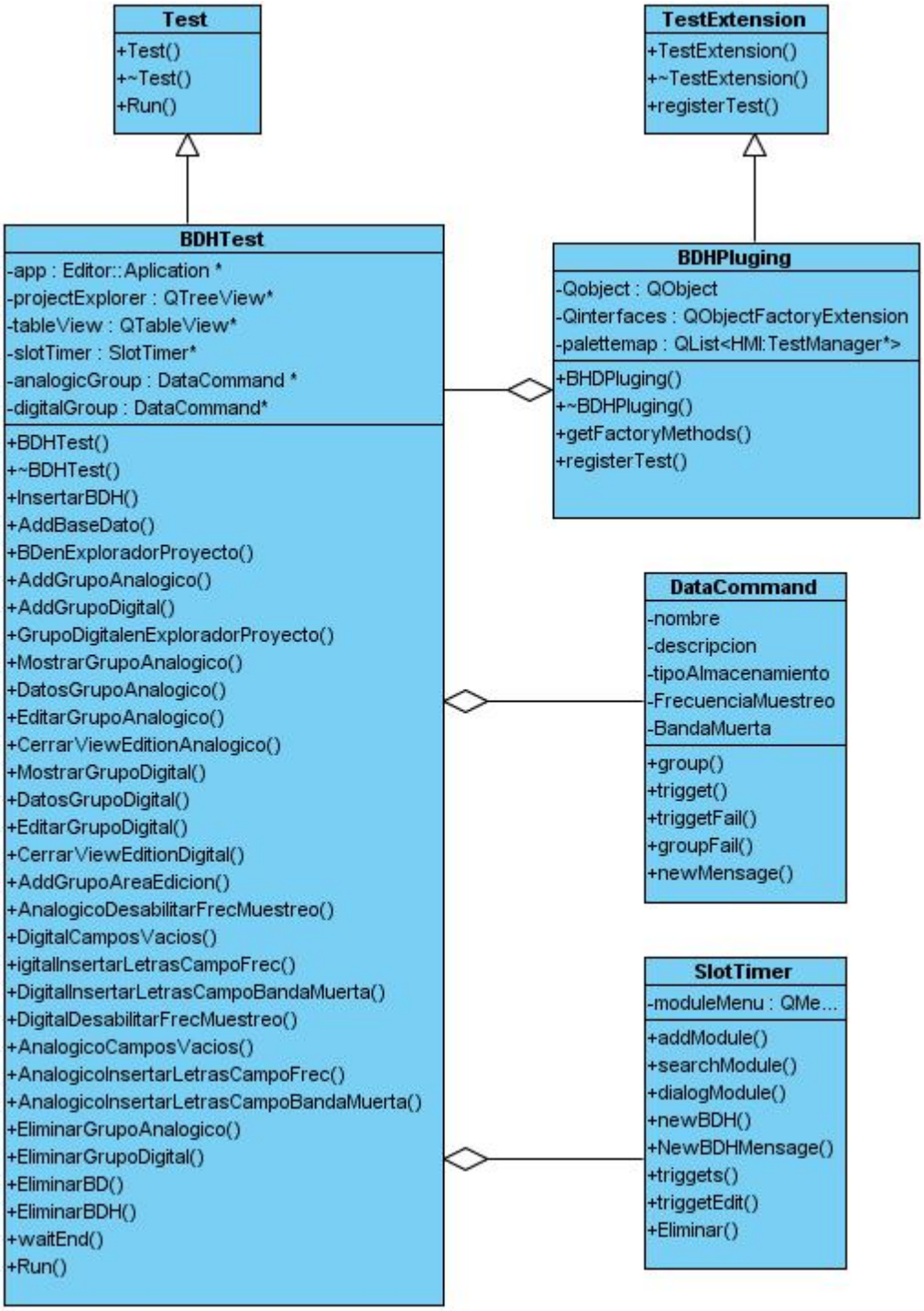

Figura 2. 2 Diagrama de Clases del Diseño

Cada método de las clases será ejecutado para determinar las no conformidades del módulo BDH, destacándose los de las clases BDHTest y BDHplugin en las que se manejan los casos de prueba. Las descripciones de los métodos de ambas clases pueden consultarse en el Anexo 2 y Anexo 3 respectivamente.

El estilo arquitectónico presente en el plugin BDH es el Modelo Vista Controlador el cual separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: el modelo, la vista y el controlador.
En el caso del componente desarrollado no se encuentra dentro de ninguna de las clases mencionadas con anterioridad pues el sistema desarrollado no aporta ni afecta las funcionalidades del plugin, solo se limita a interactuar con dicha arquitectura para verificar el correcto funcionamiento de la misma y por tanto del plugin como se muestra en el siguiente diagrama.

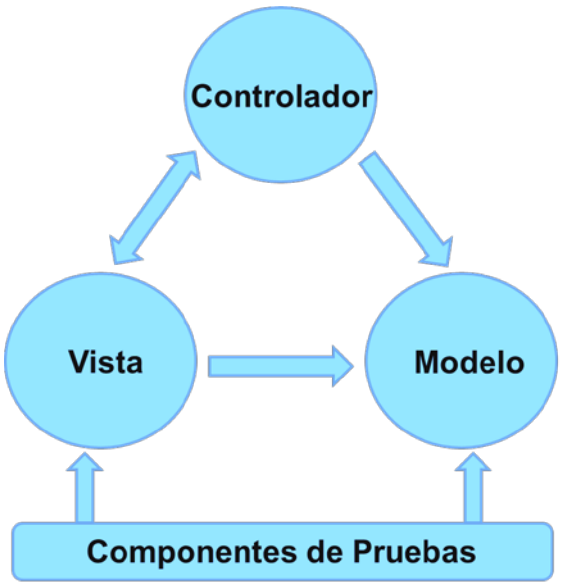

Figura 2. 3 Estilo Arquitectónico Modelo Vista Controlador

El componente de prueba interactúa con la vista para simular los eventos del ratón y el teclado o sea las acciones ejecutadas por los usuarios, por otro lado interactúa con el modelo para comprobar el correcto almacenamiento de los datos, y como se comportan los datos de entrada y salida.

Para un mejor manejo de las clases se siguieron los siguientes patrones de diseño:

De tipo GRASP el **Experto en información**: Asigna una responsabilidad a la clase que contiene toda la información necesaria para cumplir dicha responsabilidad, en el componente de pruebas implementado la clase experta es: BDHTest, la cual posee la información precisa de los casos de pruebas automatizados de las funcionalidades Administrar módulo BDH, Administrar Base de datos, Administrar Grupos de Transferencia Digital y Administrar Grupos de Transferencia Analógico.

De tipo GOF el **Patrón Fachada**: Proporciona una interfaz unificada de alto nivel que facilita su uso, la clase Test es la que funciona como fachada para todas las clases de pruebas del sistema pues sabe qué clase es responsable de ejecutar las pruebas en cada uno de los plugins manteniéndose completamente transparente para ellas, es importante destacar que aunque las funciones de probar el plugin BDH se encuentran en la clase BDHTest, es a la “Fachada” a quien le toca el trabajo de mediar y por tanto ejecutar las pruebas desde el editor. El uso de este patrón posibilita reducir el acoplamiento entre las clases.

Paso 3: Una vez implementadas las funciones de prueba se compila el plugin BDH para que los cambios realizados en el código puedan ser reconocidos por el editor a la hora de cargar el plugin.

2.5 Ejecutar las pruebas

En esta fase se ejecuta el editor Phoenix con la configuración de prueba (test) que carga el plugin de BDH con sus pruebas integradas y la interfaz de prueba del editor permitiendo ejecutar las

pruebas. Una vez ejecutadas las mismas el reporte de los resultados se encuentra documentado en la consola, mostrando la cantidad de script ejecutados satisfactoria e insatisfactoriamente, tal como se muestra en el Anexo 5. Los errores encontrados se registran en el en el documento de no conformidades, de forma tal que aporte el mayor nivel de información para su ubicación y corrección inmediata. Ante cada error es importante conocer:

1. Elemento correspondiente: Caso de prueba y escenario al que pertenece el error encontrado.
2. Número de la no conformidad: Un número consecutivo para facilitar su referencia.
3. No conformidad: Descripción del error encontrado.
4. Aspecto correspondiente: Valoración del tipo de fallo.
5. Relevancia/Nivel de falla: Importancia del error detectado según el especialista (Alta, media o baja).
6. Estado de la No conformidad: Recoge el progreso del error detectado.
7. Solución propuesta: Aspecto para recoger la propuesta de solución del equipo de prueba luego de analizar el error encontrado.

En el siguiente capítulo se abordaran con mayor detalle las no conformidades detectadas en el configurador de BDH. Luego de haber aplicado las pruebas automatizadas.

CAPÍTULO 3: ANÁLISIS DE RESULTADOS

Introducción

En el presente capítulo se describen los resultados obtenidos durante la ejecución del componente de pruebas automatizas al módulo BDH del sistema SCADA UX. Se caracterizan las no conformidades encontradas y se obtiene la respuesta de equipo de desarrollo.

3.1 Ejecución de las pruebas automatizadas

Al ejecutar las pruebas automatizadas se probaron las funcionalidades del configurador de BDH que estaban reflejadas en sus 4 casos de uso, generando 4 casos de prueba en correspondencia con estos. La distribución de escenarios por casos de prueba puede consultarse en el Anexo 4. En la gráfica que se presenta a continuación se muestra un resumen de la cantidad de escenarios por cada caso de prueba, especificando cuántos fueron ejecutados y cuantos no, los casos de prueba en la gráfica responden a la siguiente distribución:

CP 1: Gestionar módulo BDH
CP 2: Gestionar Base de datos
CP 3: Gestionar grupo de transferencia digital
CP 4: Gestionar grupo de transferencia analógico

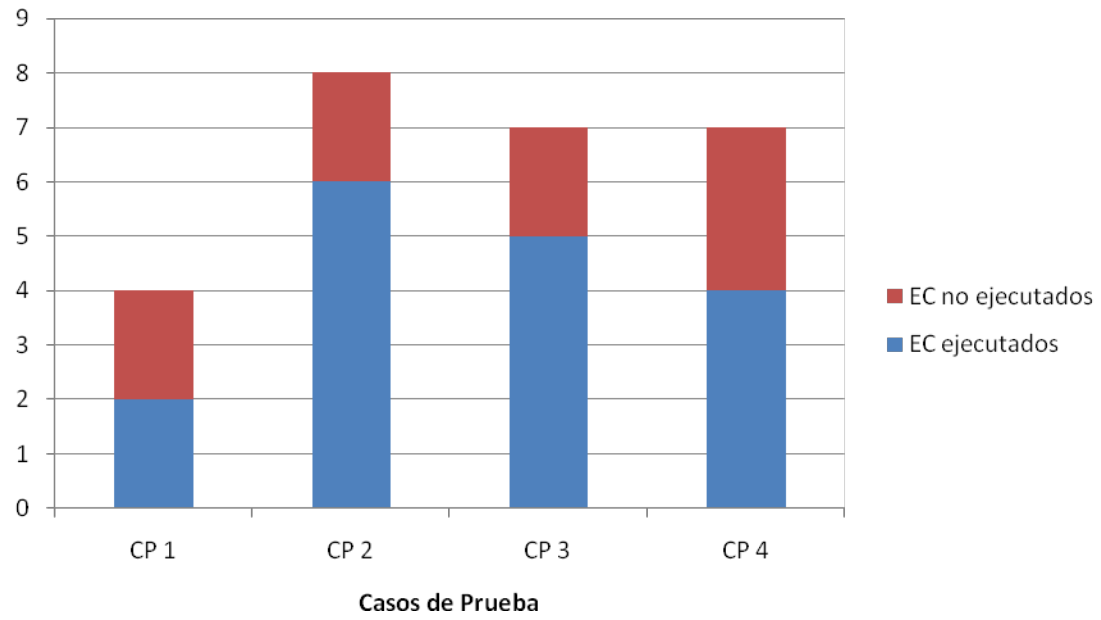

Figura 3. 1 Escenarios de pruebas

3.2 Datos de pruebas

La comparación de resultados obtenidos con los esperados, arroja los posibles fallos detectados. Dependiendo del objetivo del escenario de prueba los datos de prueba que serán utilizados pueden ser válidos (V), inválidos (I) o irrelevantes, en cuyo caso se utilizan las siglas N/A indicando que no aplica para un campo en específico. Los datos de prueba para cada escenario probado se muestran a continuación.

Gestionar módulo BDH

Escenarios	**Variable1** Módulo
EC 1.1	V BDH
EC 2.1	N/A

Tabla 3. 1 Datos de prueba: Gestionar modulo BDH

Gestionar Base de Datos

Escenarios	**Variable1** Nombre	**Variable2** Descripción
EC 1.1	V DataBase	V This is a test
EC 1.2	I (vacío)	N/A
EC 1.3	I aaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaa	N/A
EC 1.4	I DataBase	N/A
EC 1.5	I *@*	N/A
EC 2.1	N/A	N/A

Tabla 3. 2 Datos de prueba: Gestionar Base de Datos

Gestionar grupo de Transferencia Digital

Escenarios	**Var 1** Nombre	**Var 2** Descripción	**Var 3** Tipo de almacenamiento	**Var 4** Tipo de transferencia	**Var 5** Frecuencia muestreo	**Var 6** Banda muerta
EC 1.1	V Digital1	V This is a group	V No jerárquico	V muestreo	V 22	V 5
EC 1.2	I vacío	N/A	N/A	N/A	N/A	N/A
	V Digital2	I vacío	N/A	N/A	N/A	N/A
	V Digital3	V This is a group	V No jerárquico	V muestreo	I vacío	N/A
	V Digital4	V This is a group	V No jerárquico	V excepción	V 34	I vacío

EC 1.3	V Digital1	V This is a group	V No jerárquico	V excepción	V 22	V 5
EC 2.2	N/A	N/A	N/A	N/A	N/A	N/A
EC 3.1	V Digital5	V -	V No jerárquico	V muetreo	V 15	V 10

Tabla 3. 3 Datos de prueba: Gestionar grupo de Transferencia Digital

Gestionar grupo de Transferencia Analógico

Escenarios	**Var 1** Nombre	**Var 2** Descripción	**Var 3** Tipo de almacenamiento	**Var 4** Tipo de transferencia	**Var 5** Frecuencia muestreo	**Var 6** Banda muerta
EC 1.1	V Analogico1	V This is a group	V No jerárquico	V muestreo	V 15	V 10
EC 1.2	I vacío	N/A	N/A	N/A	N/A	N/A
	V Analógico2	I vacío	N/A	N/A	N/A	N/A
	V Analógico3	V This is a group	V No jerárquico	V muestreo	I vacío	N/A
	V Analógico4	V This is a group	V jerárquico	V excepción	V 34	I vacío
CP 2.1	N/A	N/A	N/A	N/A	N/A	N/A
CP 3.1	V Analogico5	V -	V jerárquico	V muestreo	V 15	V 10

Tabla 3. 4 Datos de prueba: Gestionar grupo de Transferencia Analógico

3.3 Criterios de aceptación

Los criterios de éxito de las pruebas reflejan el flujo correcto de las funcionalidades del sistema. No alcanzar este criterio implica que el sistema no se ajusta a sus especificaciones, considerándose en este caso que el escenario falló y por lo tanto debe registrarse una no conformidad. Los criterios de éxito definidos en la presente investigación son:

- ✓ Los escenarios de pruebas que evalúan la creación correcta de los componentes se consideran exitosos si permiten introducir los datos correctamente y como resultado se crea el componente en la estructura de datos correspondientes.

- ✓ Los escenarios de pruebas que evalúan la creación de los componentes con datos erróneos en los campos se consideran exitosos si muestran el mensaje de error correspondiente al campo incorrecto.

- ✓ Los escenarios de pruebas que evalúan la modificación correcta de los componentes se consideran exitosos si: la interfaz muestra los datos de los campos a modificar, permiten introducir los datos correctamente y se modifican los datos del elemento satisfactoriamente en la estructura de datos correspondientes.

- ✓ Los escenarios de pruebas que evalúan la modificación de los componentes con datos erróneos en los campos se consideran exitosos si: la interfaz muestra los datos de los campos a modificar, se introduce los datos incorrectos, mostrando el sistema un mensaje de error correspondiente al campo incorrecto.

- ✓ El criterio de prueba satisfactorio se dará para el caso de la eliminación si el sistema muestra un mensaje de confirmación y luego de aceptar el mismo se elimina el componente de la estructura de datos correspondientes.

Durante la realización de estas pruebas se probaron un total de 17 escenarios (mediante la ejecución de 36 script de pruebas), identificando, según los criterios de éxito planteados, 6

escenarios no satisfactorios y 11 satisfactorios, distribuidos por casos de prueba de la siguiente manera.

Funcionalidad: Gestionar Módulo BDH

Escenarios de casos de prueba ejecutados	Satisfactorios	No satisfactorios
2	2	0

Tabla 3. 5 Gestionar Módulo BDH

Funcionalidad: Gestionar Base de Datos

Escenarios de casos de prueba ejecutados	Satisfactorios	No satisfactorios
6	5	1

Tabla 3. 6 Gestionar Base de Datos

Funcionalidad: Gestionar Grupo de Transferencia Digital

Escenarios de casos de prueba ejecutados	Satisfactorios	No satisfactorios
5	1	4

Tabla 3. 7 Gestionar Grupo de Transferencia Digital

Funcionalidad: Gestionar Grupo de Transferencia Analógico

Escenarios de casos de prueba ejecutados	Satisfactorios	No satisfactorios
4	3	1

Tabla 3. 8 Gestionar Grupo de Transferencia Analógico

3.4 Resultado de las pruebas

Es responsabilidad del probador documentar de forma correcta y concisa el resultado obtenido durante las pruebas para ayudar a identificar las funcionalidades que no responden a los comportamientos previamente especificados en la implementación del componente, permitiendo al equipo de desarrollo erradicar las deficiencias encontradas durante las operaciones de testeo. A continuación se exponen las No Conformidades detectadas durante la aplicación de las pruebas automatizadas.

Documento de No Conformidades

Elemento	No	No conformidad	Aspecto correspondiente	Relevancia	Estado	Solución Propuesta
CP Gestionar Base de Datos EC 1.3	1	El campo nombre de la base de datos admite más de 32 caracteres	Validación	Media	Pendiente	Validar que el campo nombre de la base de datos solo permita entrar 32 caracteres
CP Grupo Digital EC 3.1	2	El sistema no permite editar un grupo de transferencia digital porque al seleccionar la opción de Editar no se visualizan los grupos de transferencia digitales creados previamente en el sistema	Funcionalidad	Alta	Pendiente	Validar que se muestren en el área de edición los grupos de transferencia digitales insertados en el sistema.
CP Grupo Digital EC 2.2	3	Al adicionar un grupo digital en el sistema se muestra en el explorador de proyecto, pero en	Funcionalidad	Alta	Pendiente	Comprobar que se adicione el grupo digital en el modelo real de datos.

		el modelo real de datos no se inserta.				
CP Grupo Transferencia digital EC 1.3	4	El sistema permite insertar un grupo de transferencia digital con el mismo nombre de uno existente	Validación	Alta	Pendiente	Validar que no se puedan agregar grupos de transferencia con el mismo nombre.
CP Grupo Transferencia digital EC 2.2.	5	El sistema no elimina un grupo de transferencia digital independiente.	Funcionalidad	Alta	Pendiente	Implementar correctamente la opción “Eliminar Histórico”
CP Grupo Transferencia analógico EC 2.1	6	El sistema no muestra un mensaje al eliminar grupo de transferencia.	Funcionalidad	Media	Pendiente	Validar que se muestre un mensaje de confirmación al eliminar un grupo de transferencia.

Tabla 3. 9 No Conformidades

3.5 Resumen de evaluación

En resumen la mayor cantidad de fallas detectadas se refieren a funcionales que no se están desempeñando tal como se definen en los casos de uso y las otras representan validaciones de campos, lo cual se puede observar en la siguiente gráfica:

Figura 3. 2 Total de escenarios probados

Finalizadas las pruebas se discuten las no conformidades con los líderes del equipo de desarrollo analizando cada una de las anomalías encontradas para agilizar su solución en la siguiente versión del sistema.

El equipo está trabajando en la nueva versión del módulo BDH, la mayoría de las no conformidades planteadas han sido resueltas, pero aún no se ha liberado el producto para realizarle una nueva iteración de pruebas automatizadas.

Aunque el costo inicial fue mayor en el caso de las pruebas automatizadas, debido a la necesidad de seleccionar una herramienta, familiarizarse con el lenguaje de programación y con la aplicación a probar, este disminuye de acuerdo al conocimiento que se va adquiriendo acerca de estos temas. En el caso del sistema SCADA UX donde el proceso de pruebas se realiza repetidamente por cada versión, la automatización es de mucha ayuda. Ésta permite realizar procesos de testeo sobre un código que cambia frecuentemente reduciendo el tiempo insumido en las pruebas de regresión.

Los beneficios que generó el proceso de pruebas automatizadas al módulo de BDH del SCADA UX lo constituye sin duda la calidad de los reportes de errores detectados, lo que contribuye a

agilizar la entrega de no conformidades al equipo de desarrollo, y por tanto al análisis y corrección de las mismas. Por otra parte el CEDIN puede contar con el componente de pruebas realizado para las próximas iteraciones de prueba al configurador de BDH.

CONCLUSIONES

Como resultado de la investigación desarrollada, se puede arribar a las siguientes conclusiones:

Del análisis realizado a las fuentes de información sobre las pruebas de software, se seleccionaron aquellas que pudiesen develar las fallas y errores del módulo BDH: las pruebas unitarias y funcionales.

Se describieron las herramientas y tecnologías que fueron utilizadas para implementar el componente de pruebas automatizadas, para la configuración del módulo BDH del editor Phoenix.

Se elaboró un Plan de Pruebas donde se describen los elementos necesarios para la ejecución del Proceso de Pruebas; definiéndose la estrategia de pruebas a seguir, así como los recursos destinados al proceso, roles involucrados y las principales funcionalidades del módulo BDH que serían objeto de prueba.

Se diseñaron los casos de pruebas viables para la detección de fallas y errores en la configuración del módulo BDH

Se implementó el componente de pruebas automatizadas, que garantizan mayor precisión en el proceso de pruebas al módulo BDH, permitiendo detectar en menor tiempo las anomalías presentes y posibilitando su depuración satisfactoria por parte del equipo de desarrollo.

RECOMENDACIONES

- ✓ Comprobar en nuevas iteraciones, mediante la ejecución de pruebas automatizadas el correcto funcionamiento de las antiguas funcionalidades y la depuración de los defectos encontrados.

- ✓ Seguir ampliando la clase que contiene los casos de pruebas en función de las nuevas funcionalidades que se le agreguen al módulo BDH.

REFERENCIAS BIBLIOGRÁFICAS

1. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* 5ta. 2005. pág. 135.

2. **Vega, Iliana Pérez Pupo. Irina Elena Argota.** *Desarrollo del flujo de requisitos para el subsistema de Gráficos Vectoriales del producto SCADA Nacional.* UCI. 2007. pág. 22, Tesis de ingeniería.

3. **García, Yuniel Sardiñas.** *Herramienta de respaldo, restauración y mantenimiento para el módulo de Base de Datos de Históricos del SCADA Guardián del ALBA.* UCI. 2010. Tesis de Ingeniería.

4. **E, Dustin.** *Effective Software testing.* s.l. : Person Education, 2003. pág. 57.

5. **S., Roger.** *Ingeniería del Software. Un enfoque práctico.* 2005. pág. 335.

6. **IEEE.** *Computer Dictionary.* [trad.] Inglés. 1990. Standard 610.

7. **Process, Rational Unified.** Ayuda del RUP. *calisoft.uci.cu.* [En línea] 2003. [Citado el: 20 de mayo de 2011.] http://10.12.63.200:5900/rup/index.htm#core.base_rup/guidances/supportingmaterials/welcome_2BC5187F.html.

8. **Software, Colectivo de ingeniería de.** Entorno Virtual de Aprendizaje. Ingeniería de Software 2. Flujo de Pruebas. [En línea] 2010. [Citado el: 25 de mayo de 2011.] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_10/Clase_Practica_6/Materiales_Basicos/Material_de_caja_b_y_caja_n.pdf.

9. **López, Abel Quintana.** *Herramienta de Pruebas Automatizadas para el módulo HMI del editor Phoenix.* UCI. 2011.

10. **René Iván Rodríguez Infante, Wilsón Alfonso González.** *"Automatización de las pruebas de interfaz de usuario para componente HMI GTK del SCADA".* UCI. 2010. pág. 21.

11. **Gómez., Yurisel Menéndez.** *Pruebas Automatizadas para el plug-ins Adquisición de Datos de la Interfaz Hombre – Máquina (HMI) QT del proyecto SCADA UX.* UCI. 2011. pág. Capítulo 1.

12. **Larman, Craig.** *UML y Patrones. Introducción al Análisis y Diseño orientado a objetos.* 1999. pág. XIX.

13. **ATC., Departamento.** *Curso del guardián del Alba. Verción Sucre.* Mérida : s.n., 2010.

14. **Cepero, Mariela.** *Modelo de Casos de Uso del sistema. Editor Base de datos Histórico .* 2011.

BIBLIOGRFÍA

1. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* 5ta. 2005. pág. 135.

2. **Vega, Iliana Pérez Pupo. Irina Elena Argota.** *Desarrollo del flujo de requisitos para el subsistema de Gráficos Vectoriales del producto SCADA Nacional.* UCI. 2007. pág. 22, Tesis de ingeniería.

3. **García, Yuniel Sardiñas.** *Herramienta de respaldo, restauración y mantenimiento para el módulo de Base de Datos de Históricos del SCADA Guardián del ALBA.* UCI. 2010. Tesis de Ingeniería.

4. **E, Dustin.** *Effective Software testing.* s.l. : Person Education, 2003. pág. 57.

5. **S., Roger.** *Ingeniería del Software. Un enfoque práctico.* 2005. pág. 335.

6. **IEEE.** *Computer Dictionary.* [trad.] Inglés. 1990. Standard 610.

7. **Process, Rational Unified.** Ayuda del RUP. *calisoft.uci.cu.* [En línea] 2003. [Citado el: 20 de mayo de 2011.] http://10.12.63.200:5900/rup/index.htm#core.base_rup/guidances/supportingmaterials/welcome_2BC5187F.html.

8. **Software, Colectivo de ingeniería de.** Entorno Virtual de Aprendizaje. Ingeniería de Software 2. Flujo de Pruebas. [En línea] 2010. [Citado el: 25 de mayo de 2011.] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_10/Clase_Practica_6/Materiales_Basicos/Material_de_caja_b_y_caja_n.pdf.

9. **López, Abel Quintana.** *Herramienta de Pruebas Automatizadas para el módulo HMI del editor Phoenix.* UCI. 2011.

10. **René Iván Rodríguez Infante, Wilsón Alfonso González.** *"Automatización de las pruebas de interfaz de usuario para componente HMI GTK del SCADA".* UCI. 2010. pág. 21.

11. **Gómez., Yurisel Menéndez.** *Pruebas Automatizadas para el plug-ins Adquisición de Datos de la Interfaz Hombre – Máquina (HMI) QT del proyecto SCADA UX.* UCI. 2011. pág. Capítulo 1.

BIBLIOGRAFÍA

12. **Larman, Craig.** *UML y Patrones. Introducción al Análisis y Diseño orientado a objetos.* 1999. pág. XIX.

13. **ATC., Departamento.** *Curso del guardián del Alba. Verción Sucre.* Mérida : s.n., 2010.

14. **Cepero, Mariela.** *Modelo de Casos de Uso del sistema. Editor Base de datos Histórico .* 2011.

15. **Álvarez, Camilo Javier Solis y Figueroa Díaz, Roberth Gustavo.** Metodología s Tradicionales vs. Metodología s Ágiles. [En línea] 2007. http://www.mygnet.net/manuales/software/métodologias_tradicionales_vs_dot_métodologias_agiles.1515..

16. **Delgado, Ramses.** Documento de Roles y Responsabilidades del Programa de Mejora. [En línea] 2011. http://calisoft.uci.cu/attachments/046_0516_Roles%20y%20responsabilidades.pdf.

17. **Microsoft.** Microsoft Solutions Framework. [En línea] http://www.microsoft.com/spanish/MSDN/estudiantes/ingsoft/planificacion/msf.mspx..

18. **Roberth G. Figueroa, Camilo J. Solís.** *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.*

ANEXOS

Anexo 1 Modelo de entrevista a especialista de calidad

Modelo de entrevista	
Nombre y apellidos:	
Software:	Versión:
Preguntas: 1. ¿Está bien definido el proceso de pruebas en el CEDIN? 2. ¿Qué artefactos se generan? 3. ¿Qué tipos y métodos de pruebas son los que se usan? 4. ¿Se han desarrollado pruebas automatizadas en el CEDIN? 5. ¿Se almacenan los resultados de las pruebas?	
Aspectos positivos:	
Aspectos negativos:	
Recomendaciones:	

Tabla Anexo. 1 Modelo de entrevista

Anexo 2 Descripción de la clase BDHTest

Nombre: BDHTest	
Descripción: Es la clase encargada de ejecutar los casos de pruebas implementados en el resto de las clases que componen la suite de prueba.	
Nombre:	**BDHTest** ()
Descripción:	Constructor por defecto de la clase.
Nombre:	~ **BDHTest** ()
Descripción:	Destructor de la clase.
Nombre:	run()
Descripción:	Ejecuta las pruebas en el resto de las clases.
Nombre:	QString BDHTest::getName()const
Descripción:	Devuelve el nombre de la base de datos.
Nombre:	void BDHTest::InsertarBDH()
Descripción:	Adiciona el módulo BDH en el árbol.
Nombre:	void BDHTest::FieldNameBDH_32_characters()
Descripción:	Comprueba que el campo nombre de la BD permita hasta 32 caracteres.
Nombre:	void BDHTest::AddBaseDato()
Descripción:	Adiciona una base de datos, comprueba las opciones del menú desplegable, verifica que el campo nombre no esté vacío y que no contenga caracteres extraños.
Nombre:	void BDHTest::BDenExploradorProyecto()
Descripción:	Comprueba la visibilidad en el explorador de proyecto de la BD y de los paquetes de grupos analógicos y digitales.
Nombre:	void BDHTest::AddGrupoAnalogico()
Descripción:	Comprueba la adición del grupo de trasferencia analógico.
Nombre:	void BDHTest::AddHistorico_SimilarName()
Descripción:	Verifica que no hallan dos BD con el mismo nombre.
Nombre:	void BDHTest::GrupoAnalogicoenExploradorProyecto()
Descripción:	Comprueba la visibilidad en el explorador de proyecto del grupo de trasferencia analógico.
Nombre:	void BDHTest::AddGrupoDigital()
Descripción:	Comprueba la adición del grupo de trasferencia digital.
Nombre:	void BDHTest::GrupoDigitalenExploradorProyecto()
Descripción:	Comprueba la visibilidad en el explorador de proyecto del grupo de

	transferencia digital.
Nombre:	void BDHTest::MostrarGroupAnalogico()
Descripción:	Verifica si se muestra la opción editar grupos de transferencia analógico y muestra las imágenes de ambos grupos.
Nombre:	void BDHTest::DatosGroupAnalogico()
Descripción:	Comprueba si fueron insertados los datos del grupo de transferencia analógico en el tableView.
Nombre:	void BDHTest::EditarGroupAnalogico()
Descripción:	Comprueba si se modificaron todos los campos de los grupos de transferencia analógico en el área de edición.
Nombre:	void BDHTest::CerrarViewEditionAnlogico()
Descripción:	Permite simular que se cierre la tabla de edición de los grupos de transferencia analógico.
Nombre:	void BDHTest::MostrarGroupDigital()
Descripción:	Verifica si se muestra la opción editar grupos de transferencia digital y muestra las imágenes de ambos grupos.
Nombre:	void BDHTest::DatosGropoDigital()
Descripción:	Comprueba si fueron insertados los datos del grupo de transferencia digital en el tableView.
Nombre:	void BDHTest::EditarGroupoDigital()
Descripción:	Comprueba si se modificaron todos los campos de los grupos de transferencia digital en el área de edición.
Nombre:	void BDHTest::CerrarViewEditionDigital()
Descripción:	Permite simular que se cierre la tabla de edición de los grupos de transferencia digital.
Nombre:	void BDHTest::AddGrupoAreaEdicion()
Descripción:	Permite dar clic en botón (+) del área de edición.
Nombre:	void BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto()
Descripción:	Comprueba que se insertó el grupo de transferencia digital desde el área de edición en el explorador de proyecto.
Nombre:	void BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo()
Descripción:	Permite insertar un grupo de transferencia analógico con la frecuencia de muestreo desabilitada.
Nombre:	void BDHTest::DigitalCamposVacios()
Descripción:	Se comprueba que aparezca un mensaje de error cuando todos los campos están vacíos en el grupo de transferencia digital.
Nombre:	void BDHTest::DigitalInsertarLetrasCampoFrec()
Descripción:	Se comprueba que aparezca un mensaje de error, siendo el campo Frecuencia llenado y los demás campos permanecen vacíos en el grupo de transferencia digital.

Nombre:	void BDHTest::DigitalInsertarLetrasCampoBandaMuerta()
Descripción:	Se comprueba que aparezca un mensaje de error, siendo el campo Banda Muerta llenado y los demás campos permanecen vacíos en el grupo de transferencia digital.
Nombre:	void BDHTest::DigitalDesabilitarFrecuenciaMuestreo()
Descripción:	Se inserta un grupo digital con la Frecuencia de muestreo deshabilitada en el grupo de transferencia digital.
Nombre:	void BDHTest::AnalogicoCamposVacios()
Descripción:	Se comprueba que aparezca un mensaje de error cuando todos los campos están vacíos en el grupo de transferencia analógico.
Nombre:	void BDHTest::AnalogicoInsertarLetrasCampoFrec()
Descripción:	Se comprueba que aparezca un mensaje de error, siendo el campo Frecuencia llenado y los demás campos permanecen vacíos en el grupo de transferencia analógico.
Nombre:	void BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta()
Descripción:	Se comprueba que aparezca un mensaje de error, siendo el campo Banda Muerta llenado y los demás campos permanecen vacíos en el grupo de transferencia analógico.
Nombre:	void BDHTest::EliminarHistorico()
Descripción:	Se simula que se va a eliminar un grupo de transferencia digita de forma independiente.
Nombre:	void BDHTest::MostrarMensageal_EliminarGrupos()
Descripción:	Comprueba si se muestra un mensaje de confirmación al eliminar los grupos de transferencia.
Nombre:	void BDHTest::EliminarGruposDigitales()
Descripción:	Comprueba si se eliminan correctamente los grupos digital.
Nombre:	void BDHTest::EliminarGruposAnalogicos()
Descripción:	Comprueba si se eliminan correctamente los grupos analógico.
Nombre:	void BDHTest::EliminarBaseDato()
Descripción:	Comprueba que se elimine correctamente la BD.
Nombre:	void BDHTest::EliminarModuloBDH()
Descripción:	Comprueba que se elimine correctamente el módulo BDH.
Nombre:	waitEnd()
Descripción:	Para evitar conflictos con los demás módulos del SCADA.

Tabla Anexo. 2 Descripción de la clase BDHTest

Anexo 3 Descripción de la clase BDHPlugin

Nombre: BDHPlugin	
Descripción: Es la clase que contiene la extensión de prueba que será cargada y registrada por el sistema.	
Nombre:	BDHplugin()
Descripción:	Constructor por defecto de la clase.
Nombre:	~ BDHplugin()
Descripción:	Destructor de la clase.
Nombre:	void BDHplugin::registerTests(HMI::TestManager* testManager)
Descripción:	Me permite cargar el plugin en la herramienta.

Tabla Anexo. 3 Descripción de la clase BDHPlugin

Anexo 4 Distribución de escenarios por casos de prueba

Caso de prueba	**Escenarios**	**Ejecutado**
Gestionar modulo BDH	1.1 Insertar módulo BDH correctamente.	x
	1.2 Insertar módulo BDH cuando ya hay uno en el sistema.	
	2.1 Eliminar módulo BDH correctamente.	x
	2.2 Eliminar módulo con elementos asociados	
Gestionar Base de datos	1.1Adicionar Base de Datos correctamente.	x
	1.2 Validar campo nombre vacío	x
	1.3 Adicionar Base de Datos con más de 32 caracteres	x
	1.4 Adicionar Base de Datos con el mismo nombre	x
	1.5 Adicionar introduciendo caracteres extraños	x
	1.6 Cancelar adicionar base de datos	
	2.1 Eliminar base de datos correctamente	x
	2.2 Eliminar base de datos con recursos asociados	
Gestionar grupo	1.1Adicionar grupo digital correctamente.	x

de transferencia digital	1.2 Validar que no hay campos vacíos	x
	1.3 Validar que no se repita el mismo nombre	x
	1.5 Cancelar adicionar grupo de transferencia digital	
	2.1 Eliminar grupos digitales correctamente	
	2.2 Eliminar grupo digital correctamente	x
	3.1: Modificar grupo digital correctamente	x
Gestionar grupo de transferencia analógico	1.1Adicionar grupo analógico correctamente.	x
	1.2 Validar que no hay campos vacíos	x
	1.3 Validar que no se repita el mismo nombre	
	1.5 Cancelar adicionar grupo de transferencia analógico	
	2.1 Eliminar grupos analógico correctamente	x
	2.2 Eliminar grupo analógico correctamente	
	3.1 Modificar grupo analógico correctamente	x

Tabla Anexo. 4 Distribución de escenarios por casos de prueba

Anexo 5 Reporte Final de pruebas

********* Start testing of BDHTest *********
Config: Using QTest library 4.6.2, Qt 4.6.2

PASS : BDHTest::initTestCase()

QDEBUG : BDHTest::InsertarBDH() previous child name: "Node1"

QDEBUG : BDHTest::InsertarBDH() current child name: "Node1"

QDEBUG : BDHTest::InsertarBDH() previous child name: "Node1"

QDEBUG : BDHTest::InsertarBDH() current child name: "BDH"

PASS : BDHTest::InsertarBDH()

QDEBUG : BDHTest::FieldNameBDH_32_characters() previous child name: "BDH"

QDEBUG : BDHTest::FieldNameBDH_32_characters() current child name: "BDH"

FAIL! : BDHTest::FieldNameBDH_32_characters() Compared values are not the same

Actual (fieldNameBDH): 33

Expected (32): 32

Loc: [src/Test/SlotTimer.cpp(173)]

QDEBUG : BDHTest::AddBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::AddBaseDato() current child name: "BDH"
QDEBUG : BDHTest::AddBaseDato() Test Case insertar caracteres extrannos en campo Nombre
QDEBUG : BDHTest::AddBaseDato() Test Case Mostrar ventana ViewNewBDH sin campo Nombre
QDEBUG : BDHTest::AddBaseDato() Test Case Mostrar ventana ViewNewBDH
QDEBUG : BDHTest::AddBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::AddBaseDato() current child name: "DataBase"
QDEBUG : BDHTest::AddBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::AddBaseDato() current child name: "DataBase"
PASS : BDHTest::AddBaseDato()
QDEBUG : BDHTest::BDenExploradorProyecto() Test Case: Mostrar la nueva bdh en el explorador de proyecto
QDEBUG : BDHTest::BDenExploradorProyecto() Test Case: Mostrar Grupo Analogicos
QDEBUG : BDHTest::BDenExploradorProyecto() Test Case: Mostrar Grupo Digitales
PASS : BDHTest::BDenExploradorProyecto()
QDEBUG : BDHTest::AddGrupoAnalogico() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AddGrupoAnalogico() current child name: "Grupo Analogicos"
WARNING: BDHTest::AddGrupoAnalogico() Test editGroupName
WARNING: BDHTest::AddGrupoAnalogico() Test editDescription
WARNING: BDHTest::AddGrupoAnalogico() Test cbStorage
WARNING: BDHTest::AddGrupoAnalogico() Test cbTypeTransferency
WARNING: BDHTest::AddGrupoAnalogico() Test editFrecuency
WARNING: BDHTest::AddGrupoAnalogico() Test editDeadBand
QDEBUG : BDHTest::AddGrupoAnalogico() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AddGrupoAnalogico() current child name: "analogico1"
PASS : BDHTest::AddGrupoAnalogico()
PASS : BDHTest::GrupoAnalogicoenExploradorProyecto()
QDEBUG : BDHTest::AddGrupoDigital() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::AddGrupoDigital() current child name: "Grupo Digitales"
WARNING: BDHTest::AddGrupoDigital() Test editGroupName
WARNING: BDHTest::AddGrupoDigital() Test editDescription
WARNING: BDHTest::AddGrupoDigital() Test cbStorage
WARNING: BDHTest::AddGrupoDigital() Test cbTypeTransferency
WARNING: BDHTest::AddGrupoDigital() Test editFrecuency

WARNING: BDHTest::AddGrupoDigital() Test editDeadBand
QDEBUG : BDHTest::AddGrupoDigital() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::AddGrupoDigital() current child name: "digital1"
PASS : BDHTest::AddGrupoDigital()
PASS : BDHTest::GrupoDigitalenExploradorProyecto()
QDEBUG : BDHTest::MostrarGroupAnalogico() previous child name: "DataBase"
QDEBUG : BDHTest::MostrarGroupAnalogico() current child name: "DataBase"
QDEBUG : BDHTest::MostrarGroupAnalogico() Test Case Opcion Editar Grupo Transferencia
QDEBUG : BDHTest::MostrarGroupAnalogico() Test Case: Mostrar imagen Puntos Analogicos
PASS : BDHTest::MostrarGroupAnalogico()
PASS : BDHTest::DatosGroupAnalogico()
PASS : BDHTest::EditarGroupAnalogico()
PASS : BDHTest::CerrarViewEditionAnlogico()
QDEBUG : BDHTest::MostrarGroupDigital() previous child name: "DataBase"
QDEBUG : BDHTest::MostrarGroupDigital() current child name: "DataBase"
QDEBUG : BDHTest::MostrarGroupDigital() Test Case Opcion Editar Grupo Transferencia
QDEBUG : BDHTest::MostrarGroupDigital() Test Case: Mostrar imagen Puntos Digitales
PASS : BDHTest::MostrarGroupDigital()
FAIL! : BDHTest::DatosGropoDigital() Compared values are not the same
Actual (dataActual): Analogic
Expected (dataExpected->nombre): digital1
Loc: [src/Test/BDHTest.cpp(451)]
PASS : BDHTest::EditarGroupoDigital()
WARNING: BDHTest::AddGrupoAreaEdicion() Test editGroupName
WARNING: BDHTest::AddGrupoAreaEdicion() Test editDescription
WARNING: BDHTest::AddGrupoAreaEdicion() Test cbStorage
WARNING: BDHTest::AddGrupoAreaEdicion() Test cbTypeTransferency
WARNING: BDHTest::AddGrupoAreaEdicion() Test editFrecuency
WARNING: BDHTest::AddGrupoAreaEdicion() Test editDeadBand
QDEBUG : BDHTest::AddGrupoAreaEdicion() previous child name: "DataBase"
QDEBUG : BDHTest::AddGrupoAreaEdicion() current child name: "digital2"
PASS : BDHTest::AddGrupoAreaEdicion()
QDEBUG : BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto() "No se encuentra digital2 en ItemModel"

QDEBUG : BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto() "Grupo Digitales"
QDEBUG : BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto() Cantidad de hijos:
QDEBUG : BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto() 1
FAIL! : BDHTest::AreaEdicionGrupoDigitalenExploradorProyecto() 'item != 0' returned FALSE. ()
Loc: [src/Test/BDHTest.cpp(522)]
PASS : BDHTest::CerrarViewEditionDigital()
QDEBUG : BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() current child name: "Grupo Analogicos"
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test editGroupName
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test editDescription
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test cbStorage
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test cbTypeTransferency
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test editFrecuency
WARNING: BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() Test editDeadBand
QDEBUG : BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo() current child name: "analogico2"
PASS : BDHTest::AnalogicoDesabilitarFrecuenciaMuestreo()
QDEBUG : BDHTest::DigitalCamposVacios() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::DigitalCamposVacios() current child name: "Grupo Digitales"
WARNING: BDHTest::DigitalCamposVacios() Test editGroupName
WARNING: BDHTest::DigitalCamposVacios() Test editDescription
WARNING: BDHTest::DigitalCamposVacios() Test cbStorage
WARNING: BDHTest::DigitalCamposVacios() Test cbTypeTransferency
WARNING: BDHTest::DigitalCamposVacios() Test editFrecuency
WARNING: BDHTest::DigitalCamposVacios() Test editDeadBand
PASS : BDHTest::DigitalCamposVacios()
QDEBUG : BDHTest::DigitalInsertarLetrasCampoFrec() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::DigitalInsertarLetrasCampoFrec() current child name: "Grupo Digitales"
WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test editGroupName
WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test editDescription
WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test cbStorage
WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test cbTypeTransferency
WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test editFrecuency

WARNING: BDHTest::DigitalInsertarLetrasCampoFrec() Test editDeadBand
PASS : BDHTest::DigitalInsertarLetrasCampoFrec()
QDEBUG : BDHTest::DigitalInsertarLetrasCampoBandaMuerta() previous child name: "Grup Digitales"
QDEBUG : BDHTest::DigitalInsertarLetrasCampoBandaMuerta() current child name: "Grupo Digitales"
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test editGroupName
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test editDescription
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test cbStorage
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test cbTypeTransferency
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test editFrecuency
WARNING: BDHTest::DigitalInsertarLetrasCampoBandaMuerta() Test editDeadBand
PASS : BDHTest::DigitalInsertarLetrasCampoBandaMuerta()
QDEBUG : BDHTest::DigitalDesabilitarFrecuenciaMuestreo() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::DigitalDesabilitarFrecuenciaMuestreo() current child name: "Grupo Digitales"
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test editGroupName
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test editDescription
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test cbStorage
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test cbTypeTransferency
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test editFrecuency
WARNING: BDHTest::DigitalDesabilitarFrecuenciaMuestreo() Test editDeadBand
QDEBUG : BDHTest::DigitalDesabilitarFrecuenciaMuestreo() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::DigitalDesabilitarFrecuenciaMuestreo() current child name: "digital"
PASS : BDHTest::DigitalDesabilitarFrecuenciaMuestreo()
QDEBUG : BDHTest::AnalogicoCamposVacios() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AnalogicoCamposVacios() current child name: "Grupo Analogicos"
WARNING: BDHTest::AnalogicoCamposVacios() Test editGroupName
WARNING: BDHTest::AnalogicoCamposVacios() Test editDescription
WARNING: BDHTest::AnalogicoCamposVacios() Test cbStorage
WARNING: BDHTest::AnalogicoCamposVacios() Test cbTypeTransferency
WARNING: BDHTest::AnalogicoCamposVacios() Test editFrecuency
WARNING: BDHTest::AnalogicoCamposVacios() Test editDeadBand
PASS : BDHTest::AnalogicoCamposVacios()
QDEBUG : BDHTest::AnalogicoInsertarLetrasCampoFrec() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AnalogicoInsertarLetrasCampoFrec() current child name: "Grupo Analogicos"

WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test editGroupName
WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test editDescription
WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test cbStorage
WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test cbTypeTransferency
WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test editFrecuency
WARNING: BDHTest::AnalogicoInsertarLetrasCampoFrec() Test editDeadBand
PASS : BDHTest::AnalogicoInsertarLetrasCampoFrec()
QDEBUG : BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() current child name: "Grupo Analogicos"
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test editGroupName
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test editDescription
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test cbStorage
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test cbTypeTransferency
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test editFrecuency
WARNING: BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta() Test editDeadBand
PASS : BDHTest::AnalogicoInsertarLetrasCampoBandaMuerta()
QDEBUG : BDHTest::AddHistorico_SimilarName() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::AddHistorico_SimilarName() current child name: "Grupo Digitales"
WARNING: BDHTest::AddHistorico_SimilarName() Test editGroupName
WARNING: BDHTest::AddHistorico_SimilarName() Test editDescription
WARNING: BDHTest::AddHistorico_SimilarName() Test cbStorage
WARNING: BDHTest::AddHistorico_SimilarName() Test cbTypeTransferency
WARNING: BDHTest::AddHistorico_SimilarName() Test editFrecuency
WARNING: BDHTest::AddHistorico_SimilarName() Test editDeadBand
QDEBUG : BDHTest::AddHistorico_SimilarName() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::AddHistorico_SimilarName() current child name: "digital"
FAIL! : BDHTest::AddHistorico_SimilarName() Compared values are not the same
Actual (actualChildcount): 3
Expected (expectedChildcount): 2
Loc: [src/Test/BDHTest.cpp(155)]
QDEBUG : BDHTest::EliminarHistorico() previous child name: "analogico2"
QDEBUG : BDHTest::EliminarHistorico() current child name: "analogico2"
QWARN : BDHTest::EliminarHistorico() El recurso "analogico2" est???? siendo utilizado.

FAIL! : BDHTest::EliminarHistorico() Compared values are not the same
Actual (actualChildcount): 3
Expected (expectedChildcount): 2
Loc: [src/Test/BDHTest.cpp(627)]
QDEBUG : BDHTest::MostrarMensageal_EliminarGrupos() previous child name: "analogico2"
QDEBUG : BDHTest::MostrarMensageal_EliminarGrupos() current child name: "analogico2"
QWARN : BDHTest::MostrarMensageal_EliminarGrupos() El recurso "analogico2" est???? siendo utilizado.
FAIL! : BDHTest::MostrarMensageal_EliminarGrupos() Compared values are not the same
Actual (showMessages): 0
Expected (true): 1
Loc: [src/Test/BDHTest.cpp(646)]
QDEBUG : BDHTest::EliminarGruposAnalogicos() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() current child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() current child name: "analogico2"
QDEBUG : BDHTest::EliminarGruposAnalogicos() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() current child name: "digital2"
QDEBUG : BDHTest::EliminarGruposAnalogicos() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() current child name: "Digital"
QDEBUG : BDHTest::EliminarGruposAnalogicos() previous child name: "Grupo Analogicos"
QDEBUG : BDHTest::EliminarGruposAnalogicos() current child name: "Grupo Analogicos"
PASS : BDHTest::EliminarGruposAnalogicos()
QDEBUG : BDHTest::EliminarGruposDigitales() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() current child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() current child name: "digital"
QDEBUG : BDHTest::EliminarGruposDigitales() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() current child name: "digital"
QDEBUG : BDHTest::EliminarGruposDigitales() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() current child name: "digital1"
QDEBUG : BDHTest::EliminarGruposDigitales() previous child name: "Grupo Digitales"
QDEBUG : BDHTest::EliminarGruposDigitales() current child name: "Grupo Digitales"
PASS : BDHTest::EliminarGruposDigitales()

QDEBUG : BDHTest::EliminarBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::EliminarBaseDato() current child name: "BDH"
QDEBUG : BDHTest::EliminarBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::EliminarBaseDato() current child name: "DataBase"
QDEBUG : BDHTest::EliminarBaseDato() previous child name: "BDH"
QDEBUG : BDHTest::EliminarBaseDato() current child name: "BDH"
PASS : BDHTest::EliminarBaseDato()
QDEBUG : BDHTest::EliminarMóduloBDH() previous child name: "BDH"
QDEBUG : BDHTest::EliminarMóduloBDH() current child name: "BDH"
QDEBUG : BDHTest::EliminarMóduloBDH() previous child name: "BDH"
QDEBUG : BDHTest::EliminarMóduloBDH() current child name: "BDH"
QDEBUG : BDHTest::EliminarMóduloBDH() previous child name: "GALBA"
QDEBUG : BDHTest::EliminarMóduloBDH() current child name: "Node1"
PASS : BDHTest::EliminarMóduloBDH()
PASS : BDHTest::waitEnd()
PASS : BDHTest::cleanupTestCase()
Totals: 30 passed, 6 failed, 0 skipped
********* Finished testing of BDHTest *******

GLOSARIO DE TÉRMINOS

B

BDH: Módulo de Bases de Datos Histórico deñ proyecto SCADA UX.

C

CMMI: Capability Maturity Model Integration. Modelo que permite catalogar a las organizaciones con el nivel de capacidad de madurez de su proceso de desarrollo.

G

GUIs: La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz

H

HMI: Human Machine Interface (Interfaces hombre-máquina).

P

Pruebas automatizadas: Es el uso por medio de software para el control de la ejecución de pruebas, la comparación actual de resultados con los resultados que se han predicho anteriormente, la configuración de las actuales pruebas y sus pre-condiciones, otros tipos de pruebas de control y pruebas de función.

Plugin: aplicación que se relaciona con otra más compleja para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal.

S

SCADA siglas en ingles Supervisory Control And Data Acquisiton" (Control y Adquisición de Datos de Supervisión): Es un sistema basado en computadores que permite supervisar y controlar variables de proceso a distancia, proporcionando comunicación con los dispositivos de campo (controladores autónomos) y controlando el proceso de forma automática por medio de un software especializado.

Scripts: Conjunto de líneas de código que permite interactuar con los objetos del proyecto, llámense objetos gráficos, drivers, alarmas, etc, para dotar al sistema de funcionalidades avanzadas que no pueden lograse con recursos, eventos y comandos que propicia el sistema.