UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

"Implantación de un proceso de automatización de pruebas para una aplicación software"

TFG

Grado en Ingeniería Informática

Autor: Esteve Ambrosio, Daniel Alberto

Tutor: Letelier Torres, Patricio Orlando

2014-2015

Resumen

La automatización de pruebas es una de las alternativas más atractivas para afrontar un ritmo frecuente de entregas al cliente sin comprometer la calidad del producto software. Esto se hace más necesario cuando ya se tiene un producto en el entorno de producción y con una gran cantidad de usuarios que pueden verse afectados por un fallo. Existen diversos tipos y niveles de pruebas, entre ellos las pruebas de aceptación tienen especial importancia de cara a la conformidad del comportamiento externo del producto según las expectativas del cliente. El objetivo de este TFG es definir un proceso de automatización y aplicación de pruebas automatizadas en el contexto de un producto software real. Nos centraremos en pruebas de aceptación funcionales. Para esto, y gracias a la colaboración con la empresa en la cual disfruto de una beca, se establecerá un proceso para el equipo de testeo, trabajando con un producto software de gran envergadura en cuanto a funcionalidad y número de usuarios.

Palabras clave: Pruebas automatizadas, proceso de testeo, testeo automatizado.

Tabla de contenidos

1. Introducción

1.1. Motivación

La motivación para realizar este trabajo es definir un proceso de automatización de pruebas para una aplicación software, de esta manera ayudar tanto a que empresas que aún no tienen un proceso definido, como también ayudar a equipos de testeo a realizar esta implantación de una manera cómoda y eficiente, utilizando como ejemplo este desglose sobre el proceso.

Las pruebas de aceptación tienen un papel muy importante en el éxito de una versión que está basado en la satisfacción de PAs (pruebas de aceptación) por esto, es importante disponer de información sobre el estado del desarrollo y aplicación de las PAs.

Este trabajo se ha realizado con la colaboración con una PYME dedicada al desarrollo de un ERP, el cual está en mantenimiento continuo y se acompaña de otros productos software complementarios desarrollados por la misma empresa.

1.2. Objetivos

Los objetivos principales de este proyecto son el detallar el proceso de automatización de PAs para una aplicación software, todo mediante un ejemplo para analizar las repercusiones, beneficios y ventajas de implantar un enfoque basado en pruebas de aceptación (PAs), las herramientas integradas en el proceso son: RFT (Rational Functional Tester) para la gestión del proceso de desarrollo y mantenimiento de software, SAPI como herramienta para la gestión y desarrollo del producto y ATUN como una herramienta de gestión de pruebas.

1.3. Estructura de trabajo

Este proyecto va a ser estructurado en los siguientes capítulos:

- **Automatización:** Análisis de las pruebas que deben ser automatizadas, criterios que se deben emplear para seleccionar dichas pruebas, funcionamiento de pruebas y análisis de la herramienta de automatización que vamos a emplear (Rational Functional Tester).

- **Ejemplo de uso con RFT:** Como un ejemplo de los que se puede hacer con RFT de manera detallada.

- **Proceso de desarrollo basado en PAs**: Aquí se va a detallar por qué se realiza un desarrollo basado en pruebas de aceptación y que características debe cubrir en el proceso de implantación.

- **Diseño de PAs y pruebas automatizadas:** En este apartado se explica cómo se realiza el diseño de una prueba de aceptación y como la herramienta SAPI gestiona las incidencias.

- **Infraestructura para pruebas de regresión**: Explica detalladamente las distintas herramientas que componen la infraestructura de desarrollo, el lanzador de pruebas como la aplicación que gestiona las ejecuciones de las pruebas y el gestor de fallos donde se realiza la detección y control de las distintas pruebas de aceptación.

- **Conclusiones:** Cuestiones, valoraciones finales sobre este trabajo, así como peculiaridades sobre esta tesis.

- **Referencias:** Distintas fuentes de información utilizadas para la realización de este proyecto.

2. Automatización

El proceso de automatización se realiza para agilizar las actividades de testeo de una aplicación, empleando un proceso selectivo de las pruebas a automatizar como las pruebas que sean más frecuentemente aplicadas o que sean más difíciles de testear de manera manual, ayudando a controlar y probar una aplicación en continuo cambio, de esta manera creas un primer criterio por el cual seleccionar una pruebas y otras no, para crear un orden sobre las pruebas a automatizar se suelen reconocer las pruebas que afecten a las zonas más importantes de la aplicación, como la facturación o el sistema de control de usuarios. Ahora continuaremos explicando los conceptos básicos del proceso de automatización.

2.1. Pruebas Funcionales

Se denominan pruebas funcionales, a las pruebas de software que tienen por objetivo validar cuando el comportamiento del software probado cumple o no con sus especificaciones. La prueba funcional toma el punto de vista del usuario. Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna del programa raramente es considerada. A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra.

La automatización de las pruebas funcionales reduce significativamente el esfuerzo dedicado a las pruebas de regresión en productos que se encuentran en continuo mantenimiento. La automatización de las pruebas debe ser considerada un proyecto en sí mismo con objetivos definidos.

Conceptos

Se presentan los principales conceptos relacionados con la automatización de las pruebas funcionales. Se define prueba funcional, caso de prueba, procedimiento de prueba, script de prueba, suite de prueba y pruebas de regresión.

- Un **caso de prueba** (test case) es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y poscondiciones de ejecución, desarrollados con un objetivo particular o condición de prueba, tal como ejercitar un camino de un programa particular o para verificar que se cumple un requerimiento específico.
- Un **script de prueba** (test script) son los datos y las instrucciones escritas con una sintaxis formal, almacenado en un archivo y usado por una herramienta de automatización de las pruebas. Un script de prueba puede automatizar uno o más casos de prueba, navegación, inicialización u operaciones de configuración del entorno. Un script de prueba previsto para la ejecución manual de las pruebas es un procedimiento de prueba.
- Una **suite de pruebas** (test suite) es uno o más conjuntos de pruebas reunidos para satisfacer un objetivo de prueba. Un conjunto de prueba incluye scripts y documentación. En nuestro caso, las suites constituyen un conjunto de scripts y el orden de ejecución de los mismos.

- Las **pruebas de regresión** tienen como objetivo verificar que no ocurrió una regresión en la calidad del producto luego de un cambio, asegurándose que los cambios no introducen un comportamiento no deseado o errores adicionales. Implican la re ejecución de alguna o todas las pruebas realizadas anteriormente.

Herramientas de automatización de pruebas

Hay herramientas que apoyan diversos aspectos de la prueba. A continuación se presenta una clasificación posible para las herramientas:

- Administración de las pruebas y el proceso de pruebas: herramientas para la administración de las pruebas, para el seguimiento de incidentes, para la gestión de la configuración y para la administración de requerimientos.
- Pruebas estáticas: herramientas para apoyar el proceso de revisión, herramientas para el análisis estático y herramientas de modelado.
- Especificación de las pruebas: herramientas para el diseño de las pruebas y para la preparación de datos de prueba.
- Ejecución de las pruebas: herramientas de ejecución de casos de prueba, herramientas de pruebas unitarias, comparadores, herramientas de medición del cubrimiento, herramientas de seguridad.
- Desempeño y monitorización: herramientas de análisis dinámico, herramientas de desempeño, de carga y de estrés, herramientas de monitorización.

Para la automatización de las pruebas funcionales son especialmente indicadas las herramientas de ejecución de las pruebas de captura y reproducción. Estas herramientas permiten al tester capturar y grabar pruebas, para luego editarlas, modificarlas y reproducirlas en distintos entornos. Herramientas que graban la interfaz de usuario a nivel de componentes y no de bitmaps son más útiles. Durante la grabación se capturan las acciones realizadas por el tester, creando automáticamente un script en algún lenguaje de alto nivel. Luego el tester modifica el script para crear una prueba reusable y mantenible. Este script se vuelve la línea base y luego es reproducido en una nueva versión, contra la cual es comparado. En general estas herramientas vienen acompañadas de un comparador, que compara automáticamente la salida en el momento de ejecutar el script con la salida grabada.

Las características que una herramienta de automatización de pruebas funcionales debería cubrir son las siguientes:

- **Integración con otras herramientas de desarrollo.** El proceso de pruebas es una de las fases del ciclo de desarrollo de software. Normalmente, esta tarea de pruebas es realizada por el especialista en pruebas (o tester) que forma parte de un equipo de desarrollo de software, el cual es probable que esté utilizando una herramienta específica. Por eso, es interesante que el software utilizado por el tester esté integrado en el sistema que se esté utilizando para el desarrollo del proyecto.

- **Automatización de pruebas funcionales sobre distintos tipos de tecnología de aplicaciones.** Es conveniente que la herramienta no se limite, únicamente, a un tipo de aplicación específico. Es interesante que pueda trabajar con aplicaciones desarrolladas en distintas tecnologías como Java, .NET, etc.

- **Ejecución de pruebas de forma distribuida.** La herramienta debe permitir un proceso de pruebas distribuido. Dicho en otras palabras, debe consentir la ejecución sincronizada de pruebas en distintos equipos. Es decir, dar la opción de poder ejecutar varios proyectos de pruebas sobre varios sitios de trabajo, de forma simultánea o sincronizada, permitiendo crear una prueba realista del multi-cliente de las aplicaciones distribuidas.

- **Varias opciones de lenguajes para los scripts de pruebas.** Es interesante que la herramienta no utilice un lenguaje específico para los scripts de prueba sino que incorpore diversos lenguajes de test, también utilizados para el desarrollo de aplicaciones.
- **Opción para la configuración del entorno de trabajo.** Toda herramienta debe proporcionar una opción para configurar el entorno sobre el cuál se va a trabajar con las pruebas. Por ejemplo, qué navegador se va a utilizar para el registro y reproducción de las pruebas, el lugar donde se va a situar el repositorio en el cual se almacenarán las pruebas y los resultados, etc.

- **Creación de pruebas guiada y generación de scripts automática.** Debe permitir la creación de pruebas de una forma guiada, sin necesidad de teclear líneas de código, facilitando así el proceso de creación y manipulación de pruebas para usuarios principiantes. Es interesante que disponga de un grabador, cuya función sea registrar las acciones que realiza el usuario en la aplicación. Además, debe almacenar información de los objetos existentes. Por otra parte, ha de tener una opción para generar el script de prueba a partir del archivo que contiene el registro efectuado por el navegador.

- **Explorador de Scripts.** Se trata de un panel que contiene todos los objetos registrados durante la grabación de la prueba y dónde se puede consultar las propiedades de cada uno de ellos.

- **Editor de pruebas.** La herramienta debe proporcionar un editor de pruebas para la manipulación de éstas una vez finalizada la grabación. En el momento en que esté disponible el script, que contiene las acciones del usuario así como los objetos de la aplicación, se debe poder manipular el contenido de éste. Por ejemplo, introducir validaciones, puntos de verificación, etc. Por otra parte, el editor de pruebas tiene que proporcionar la opción de manipular también los scripts de prueba. Modificar los scripts trabajando directamente sobre el código fuente.

- **Generación automática de los resultados de las pruebas y su almacenamiento.** La herramienta ha de tener un mecanismo que genere informes de los resultados de la prueba después de su ejecución. Además, debe proporcionar la opción de guardar dichos informes, almacenándolos en el repositorio correspondiente.

2.2. RFT

RFT es una herramienta avanzada de testing funcional y de regresión automatizado creada para los testers y los desarrolladores de GUI que necesitan de un control superior para probar sus aplicaciones Java, Microsoft Visual Studio .NET y Web.

¿Cómo funciona?

RFT graba las interacciones del usuario con las aplicaciones Java, Web y Visual Studio .NET, creando test scripts que, al ser ejecutadas, reproducen dichas acciones. Durante la grabación, el usuario puede insertar puntos de verificación que extractan datos o propiedades específicas de la aplicación bajo prueba. Durante la reproducción, dichos puntos de verificación se usan para comparar la información grabada con la información que tiene la aplicación bajo prueba en ese momento para garantizar la consistencia.

Al desarrollar las actividades de grabación de pruebas, los testers tienen la opción de agregar código personalizado al test script para realizar una cantidad ilimitada de tareas, incluyendo la manipulación de datos y configuraciones de entorno que a menudo son necesarias para asegurar que las máquinas están constituidas adecuadamente para correr el test. Cuando se está ejecutando el test, **Rational Functional Tester** genera un registro conteniendo los resultados de las comparaciones de los puntos de verificación.

Características

- Es una herramienta de testing automatizada que es sirve para hacer pruebas funcionales y la regresión de aplicaciones Java, Microsoft Visual Studio .NET y basadas en web.
- Ofrece a los testers avanzados una selección de idiomas de script y un editor de solidez, Java en Eclipse o Microsoft Visual Basic .NET en Visual Studio .NET, para la verificación del montaje y la personalización.
- Proporciona a los testers con poca experiencia funciones automatizadas para actividades como, por ejemplo, la generación de pruebas y el testing dirigido por datos.
- Incluye la tecnología ScriptAssure y funciones de coincidencia de patrón para mejorar la capacidad de recuperación del script de verificación dado los frecuentes cambios que se producen en la interfaz del usuario de aplicaciones.
- 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.
- Permite la realización de pruebas de aplicaciones creadas con VS.NET Winforms, J2SE/J2EE, HTML/DHTML, XML, JavaScript y applets de Java e incluye soporte exclusivo para la biblioteca SWT de Java asociada con el shell de Eclipse.
- Soporta el testing de aplicaciones 3270 (zSeries) y 5250 (iSeries) que utilizan las aplicaciones basadas en IBM Rational Functional Tester Extension for Terminal.

Cuando se reproduce un script, RFT ejecuta las mismas acciones que se han grabado, con lo cual automatiza el ciclo de testo del software. Esta automatización permite probar cada una de las nuevas versiones de la aplicación rápidamente y más a fondo que las pruebas manuales, reduciendo el tiempo de testeo.

Hay dos fases generales de la reproducción de scripts:

- Fase de desarrollo de pruebas.
- Fase de testeo de regresión.

Fase de desarrollo de pruebas

En la fase de desarrollo de pruebas, se reproducen los scripts para verificar que funcionan correctamente, usando la misma versión de la aplicación bajo prueba que se utilizó para la grabación. Esta fase valida que el comportamiento de la aplicación es el esperado.

La fase de desarrollo de pruebas consiste en estos pasos:

1. Restaurar el entorno y configurar las opciones de reproducción.
2. Reproducir cada script en la misma versión de la aplicación en la que han sido grabados para garantizar el correcto funcionamiento del script.
3. Analizar los resultados en el registro.
4. Usar el comparador de puntos de verificación para determinar la causa del fallo del punto de verificación.
5. Si el script falla, editarlo o volverlo a grabar si es necesario.
6. Si se asocia un proyecto RFT con un proyecto Rational, se pueden agrupar los scripts en un test suite. Reproducir la suite para verificar el correcto funcionamiento de los scripts. Si fallan, editarlos o volverlos a grabar si es necesario.

Fase de testeo de regresión

En la fase de testeo de regresión, se reproducen los scripts para com1probar que el comportamiento de la última versión sea el mismo que en la fase de desarrollo de pruebas. Las pruebas de regresión identificarán las diferencias que han sido introducidas desde la última versión. Se evalúan esas diferencias para determinar si son defectos o cambios.

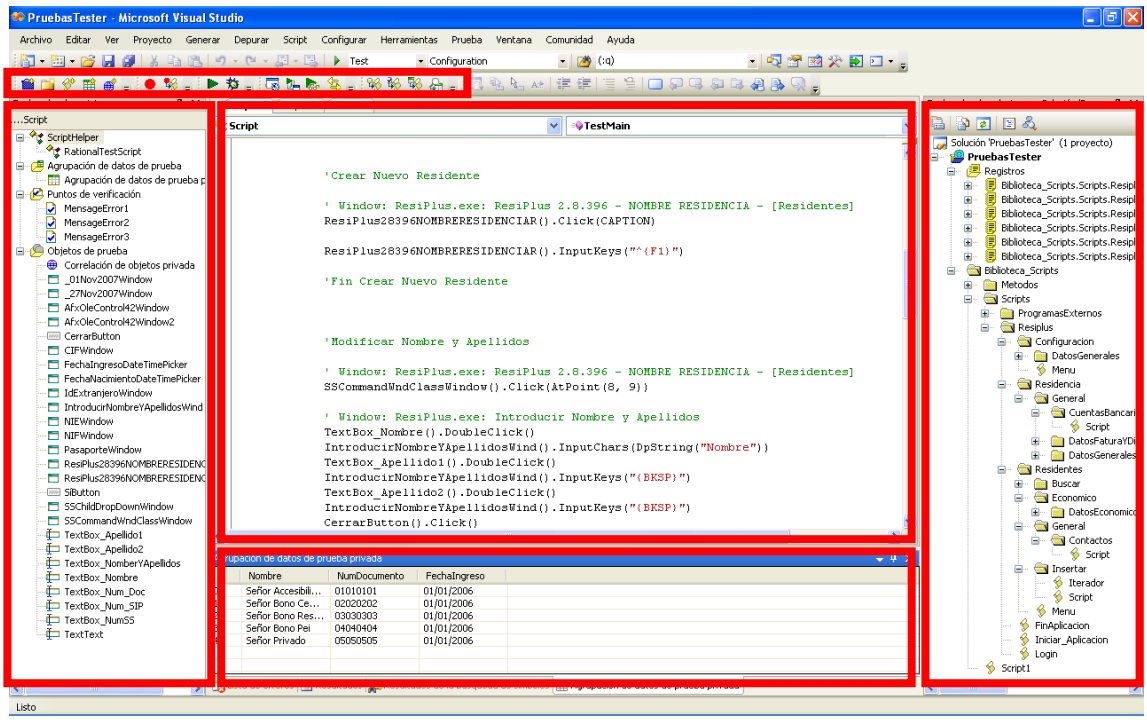
En la fase de testeo de regresión consiste en cinco pasos:

1. Restaurar el entorno y configurar las opciones de reproducción.
2. Reproducir los scripts con la nueva versión de la aplicación bajo prueba.
3. Analizar los resultados en el registro.
4. Usar el comparador de puntos de verificación de RFT para determinar la causa del fallo del punto de verificación. Si la verificación falla por cambios intencionados de la aplicación bajo prueba, actualizar los datos usando el comparador.
5. Si es necesario, revisar los scripts usar nuevas características de la aplicación bajo prueba. Después, reproducir los scripts revisados con la versión actual de la aplicación y evaluar los resultados.

En este apartado se va a explicar detalladamente las funciones principales y cómo usar esta herramienta para hacer pruebas funcionales.

2.2.1. Entorno de RFT.

Dentro de la herramienta RFT, se puede distinguir estas partes:



En la parte central está el editor de código, donde se puede ver el código de los scripts. En la parte derecha está el explorador de soluciones, que muestra las carpetas, scripts y otros archivos que contiene el proyecto actual. También se puede ver los registros creados al ejecutar scripts. En la ventana de la parte izquierda, se lista los elementos que componen el script. Estos son: los objetos de prueba, la agrupación de datos y los puntos de verificación.

Si se observa la barra de herramientas situada en la parte superior del editor de código, se pueden ver estas opciones:

	**Crear un proyecto Functional Test**	Crea un nuevo proyecto Functional Test.
	**Nueva carpeta**	Crea una nueva carpeta dentro del proyecto.
	**Crear un script vacío de Funcional Test**	Configura y crea un script en el proyecto sin código.
	**Crear una agrupación de datos de prueba.**	Crea una agrupación de datos de prueba en el proyecto.

	**Crear una correlación de objetos de prueba**	Crea un test una correlación de objetos de prueba en el proyecto.
	**Registrar un script de Functional Test**	Crea un script y muestra la ventana de grabación.
	**Insertar registro en script de Functional Test activo**	Muestra el cuadro de dialogo de grabación. El código que se genere se ubicará en el script en el que estaba el cursor antes de haber pulsado el botón.
	**Ejecutar script de Functional Test**	Ejecuta el script que se muestra en el editor de código.
	**Depurar script de Functional Test**	Ejecuta el script en modo depuración.
	**Configurar aplicaciones para prueba**	Muestra un cuadro de dialogo para añadir y configurar las aplicaciones de prueba.
	**Habilitar entornos para prueba**	Muestra un cuadro de dialogo para habilitar y configurar el entorno Java y los navegadores.
	**Abrir inspector de objetos de prueba**	Muestra el cuadro de dialogo del inspector de objetos de prueba, en el que muestra información de los objetos de prueba.
	**Sincronizar solución con el editor**	Selecciona en el explorador de soluciones el script que se muestra en el editor de código.
	**Insertar objeto de prueba en script activo de Functional Test**	Muestra un cuadro de dialogo en el que permite añadir objetos de prueba en la correlación de objetos de prueba y en el script actual.
	**Insertar punto de verificación en script activo de Functional Test**	Muestra un cuadro de dialogo para seleccionar un control de la aplicación para insertar un punto de verificación en el script actual.
	**Insertar mandatos controlados en script activo en Functional Test**	Muestra un cuadro de dialogo en el que se permite insertar variables en el en la agrupación de datos del script actual seleccionando controles de la aplicación.
	**Buscar literales y reemplazar por referencia de agrupación de datos**	Muestra un cuadro de dialogo con el que se puede reemplazar los datos del script por datos que contiene las variables de la agrupación de datos.

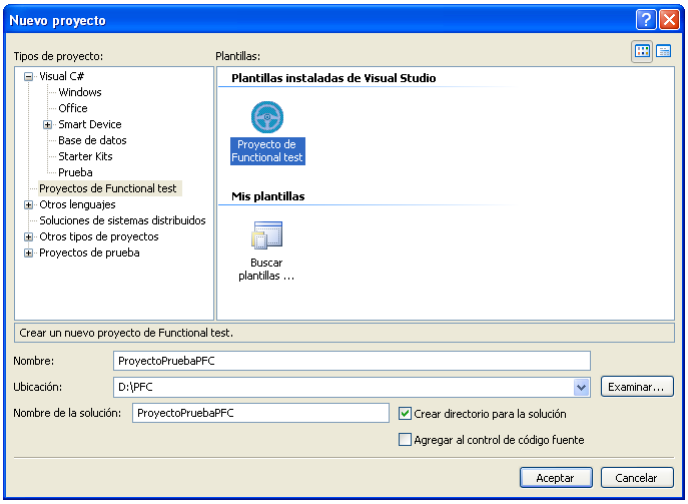
2.2.2. Creación de un proyecto RFT.

Para crear un proyecto hay que seguir estos pasos:

1. Hay que iniciar el Visual Studio, para ello ir a **Inicio → Todos los programas → Microsoft Visual Studio → Microsoft Visual Studio**.

2. Una vez arrancado el Visual Studio, ir a **Archivo → Nuevo → Proyecto de Functional Test….**

3. Escribir el nombre y elegir la ubicación del proyecto haciendo clic en **Examinar….** Hacer clic en **Aceptar**.



En esta imagen se muestra la ventana que aparece tras seleccionar la pestaña de nuevo proyecto, aquí pones nombre al proyecto, su ubicación y el tipo de proyecto.

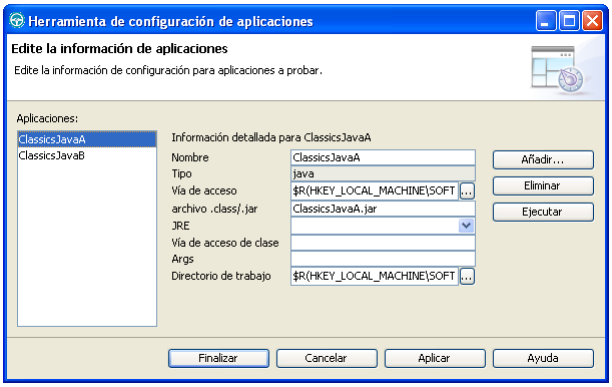
4. El proyecto será creado tras una breve espera.

2.2.3. Configuración de aplicaciones para prueba.

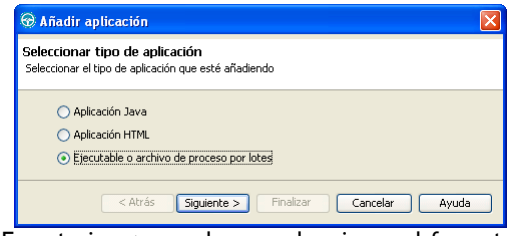
Para tener disponibles aquellas aplicaciones a las que se quiere hacer pruebas, hay que introducirlas en la lista de aplicaciones y configurarlas antes de utilizarlas, así se pueden ejecutar desde los scripts. Para hacer esto, hay que seguir estos pasos:

1. Ir a **Configurar → Configurar aplicaciones para prueba** o pulsar en la barra de herramientas.

2. Hacer clic en **Añadir…**. Tal como se muestra en la siguiente imagen:

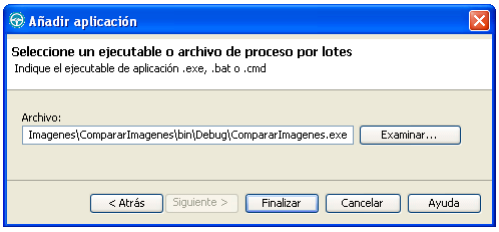


3. Si el programa es un ejecutable, escoger la opción **Ejecutable o archivo de proceso por lotes**, en caso de que sea un programa implementado en Java, seleccionar **Aplicación Java**.



En esta imagen podemos seleccionar el formato

4. Escoger la ruta de la aplicación haciendo clic en **Examinar…**. Hacer clic en **Finalizar**.



Aquí podemos ver la ventana para la selección de ruta de la aplicación

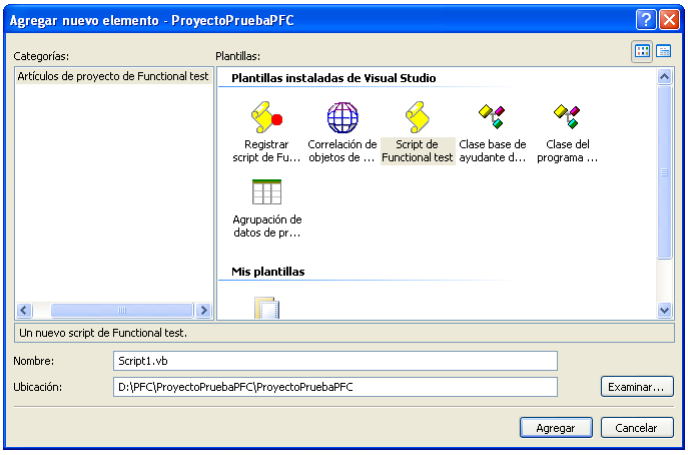
5. Aparecerá en la lista de aplicaciones. Si se quiere ejecutar con argumentos, estos se deben escribir en **Args**. Hacer clic en **Aplicar** si se quiere conservar la configuración de la aplicación para otros proyectos. Hacer clic en **Finalizar**.

2.2.4. Creación de scripts.

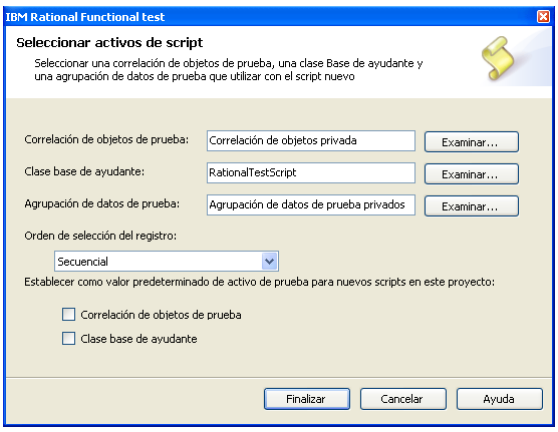
Hay dos formas de crear scripts. Una de ellas es creando el script sin código para ejecutar, es decir el método principal del script no tiene código. La otra forma de crear un script es al grabar, el código que se genere en la grabación es el que tendrá este script. A continuación se explicará cómo crearlos.

1. Hacer clic en la carpeta donde se quiere guardar el script en el explorador de soluciones. Para crear un script vacío ir a ir a **Proyecto → Añadir script vacio…** o hacer clic en de la barra de herramientas. Si se quiere crearlo al grabar, ir a **Proyecto → Añadir script utilizando el grabador…** o pulsar de la barra de herramientas.

2. Aparecerá este cuadro de dialogo para escribir el nombre del script. Hacer clic en **Agregar**.



3. La siguiente ventana es en la que se configuran los scripts. Para aceptar la configuración, hacer clic en Finalizar. Si el script se ha creado al grabar, a continuación aparecerá la pantalla de grabación. El script aparecerá en el explorador de soluciones del proyecto.



En la pantalla de configuración del script, se puede elegir que correlación de objetos de prueba y que agrupación de datos usar. Se pueden usar tanto privados como los que se tenga creados dentro del proyecto.

También se puede escoger la clase base de ayudante, es decir, la clase la cual hereda el script. Se pueden hacer clases base de ayudante propia para añadir métodos o sobrescribirlos en un script.

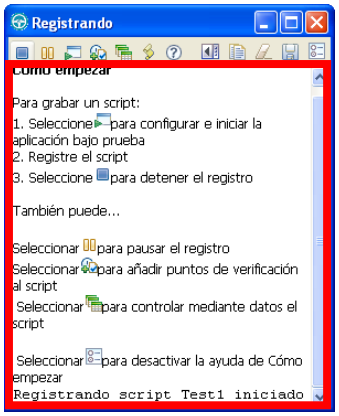
En el orden de selección del registro, indica cómo se recorrerá los registros de la agrupación de datos. El orden puede ser secuencial (los registros de la agrupación de datos se recorren de manera en que aparecen) o aleatorio (los registros se recorren aleatoriamente).

Para guardar la configuración de la clase base de ayudante o la correlación de objetos de prueba para la creación de otros scripts, basta con seleccionar los checks correspondientes.

2.2.5. Grabación de scripts.

La grabación de scripts se puede comenzar mediante el botón  de la barra de herramientas o yendo a **Script → Añadir script utilizando el grabador…**, en cuyo caso creará un script y grabara en él. Si se quiere grabar a partiendo un script existente, hay que tener el script abierto en el editor de código, situar el cursor donde se quiera que se inserte el nuevo código e ir a **Script → Insertar registro** o pulsar el botón  de la barra de herramientas.

Aparecerá la pantalla de grabación y todo lo que se haga, se grabará, excepto las acciones que se hagan sobre esta pantalla. Desde esta pantalla se pueden insertar puntos de verificación y también construir la agrupación de datos del script. La zona que está dentro del rectángulo rojo es el monitor, muestra el código y los errores que se van generando con la grabación.



La tabla siguiente explica con más detalle las opciones de la pantalla de grabación.

	**Detener registro**	Cierra la ventana de grabación y para la grabación.
	**Pausar registro**	Pausa la grabación, las acciones que se hagan en la interfaz de la aplicación no se grabarán.
	**Reanudar registro**	Reanuda la grabación.

	**Iniciar aplicación**	Inicia la aplicación seleccionada.
	**Insertar punto de verificación y mandato de acción**	Inserta un punto de verificación.
	**Insertar mandatos controlados por datos**	Inserta variables en la agrupación de datos.
	**Insertar mandatos de soporte de script**	Inserta funciones como llamadas a otros scripts, suspensión del script, insertar comentarios en la entrada de registro, etc.
	**Mostrar ayuda**	Muestra la ayuda sobre esta ventana.
	**Mostrar solo barra de herramientas**	Oculta el monitor, dejando solo visible las opciones de la barra de herramientas.
	**Mostrar monitor**	Muestra el monitor.
	**Copiar texto seleccionado**	Copia el texto seleccionado del monitor.
	**Borrar todo el texto de monitor**	Borra todo el texto del monitor.
	**Guardar texto de monitor como**	Guarda el texto del monitor en un archivo.
	**Preferencias de mensaje de monitor**	Configura los mensajes que se muestran en el monitor.

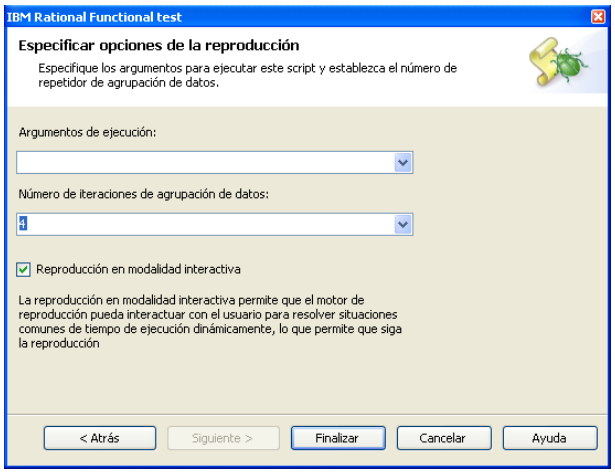
2.2.6. Ejecutar scripts.

Para ejecutar scripts:

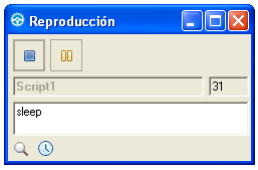
1. Pulsar el botón  de la barra de herramientas o ir a **Script → Ejecutar**, y se ejecutará el script que está actualmente en el editor de código. En caso de quererlo ejecutar en modo depurador ir a **Script → Depurar** o pulsar el botón 

2. Se abrirá una ventana para sobrescribir en un registro existente o crear uno nuevo indiciando el nombre.

3. En la pantalla de configuración de reproducción se puede indicar los argumentos que recibe el script para su ejecución y también el número de iteraciones que debe hacer sobre la agrupación de datos. Podemos ver sicha ventana a continuación:



4. Al comenzar la reproducción aparecerá la ventana de reproducción. Desde esta ventana se puede pausar la reproducción haciendo clic en o pulsando la tecla **F12** o parar haciendo clic en o pulsando la tecla **F11**. Que se puede ver así:

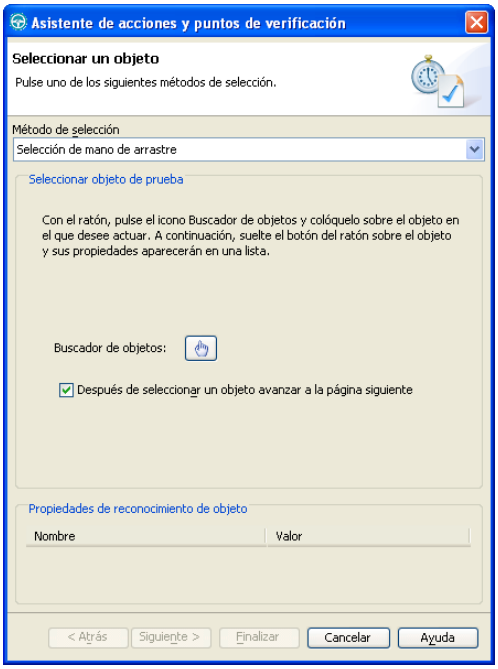


Una vez acabado la reproducción, en el navegador se abrirá el registro que se haya producido indicando el resultado de la prueba. Si se pulsa parar, la prueba fallará.

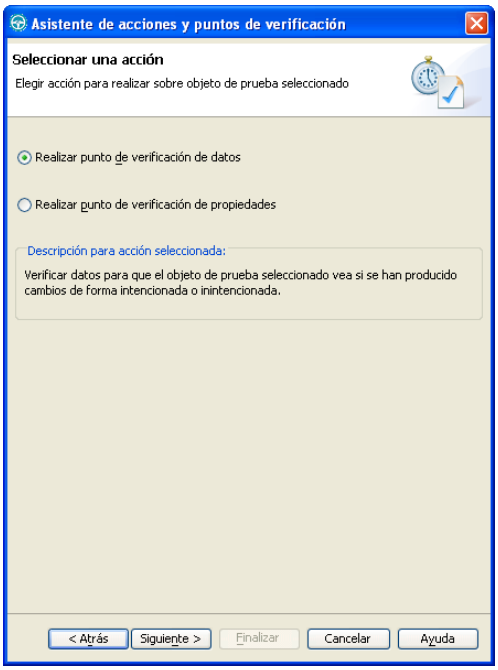
2.2.7. Crear puntos de verificación.

Los puntos de verificación se pueden crear en cualquier momento, no es necesario que se esté grabando la prueba. Para insertar un punto de verificación hay que hacer clic en el botón **Insertar punto de verificación y mandato de acción** ( ) si se está grabando el script o pulsar **Insertar punto de verificación en script activo de Functional Test** ( ) en la barra de herramientas.

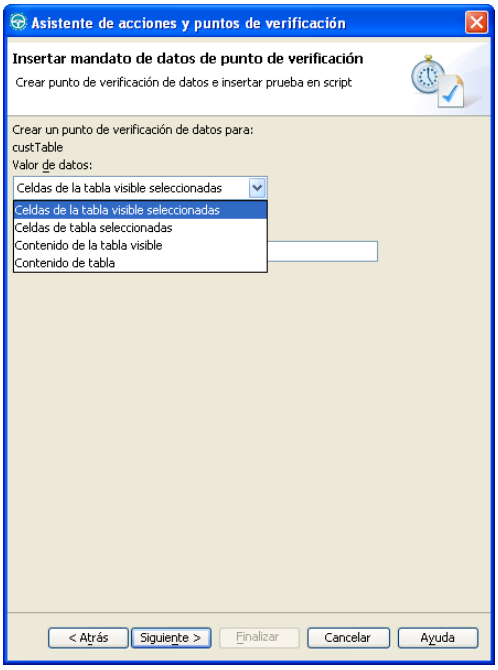
Se abría la ventana del asistente de acciones y puntos de verificación. Haciendo clic en el botón  y arrastrando saldrá un rectángulo rojo en el que muestra el objeto de la interfaz que se está seleccionando. Al soltar la pantalla del asistente cambiará. Podemos ver a continuación una captura:



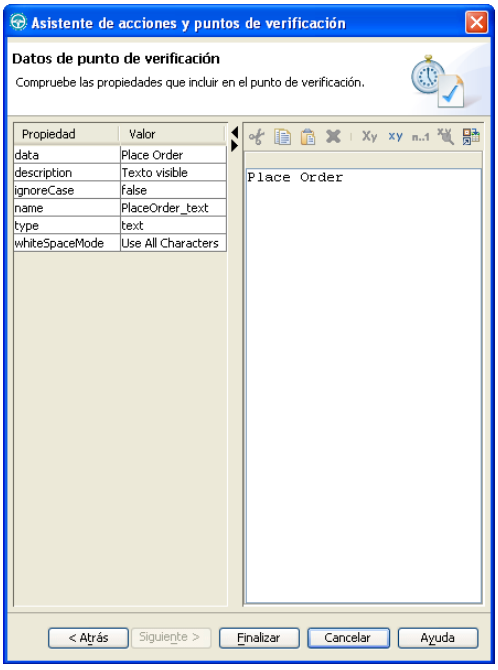
El siguiente paso es seleccionar que tipo de punto de verificación hacer, sobre los datos o sobre las propiedades del objeto, como vemos a continuación:



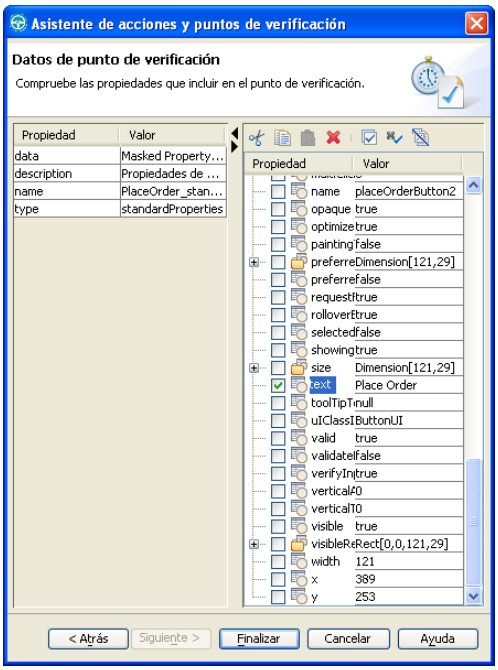
Si se selecciona **Realizar punto de verificación de datos**, se puede elegir que datos verificar, según el tipo de objeto, estos datos son distintos. Por ejemplo, si es una tabla, se puede seleccionar como datos las celdas de la tabla, los datos de las celdas seleccionadas, el contenido de la tabla visible o todo el contenido de la tabla. También se puede cambiar el nombre del punto de verificación.



En esta ventana al pulsar siguiente, se puede ver los datos que contiene el objeto, pudiendo ser modificados por el usuario. También se puede convertir los datos en expresiones regulares o introducir referencias a variables de una agrupación de datos, como vemos en la siguiente imagen:



Si se elige realizar un punto de verificación de propiedades, la siguiente pantalla se listará las propiedades del objeto, pudiendo elegir mediante checkbox las propiedades a verificar.

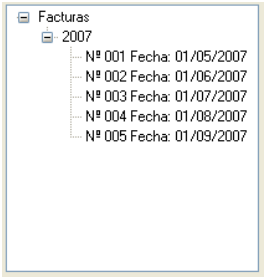


3. Ejemplo de uso de RFT

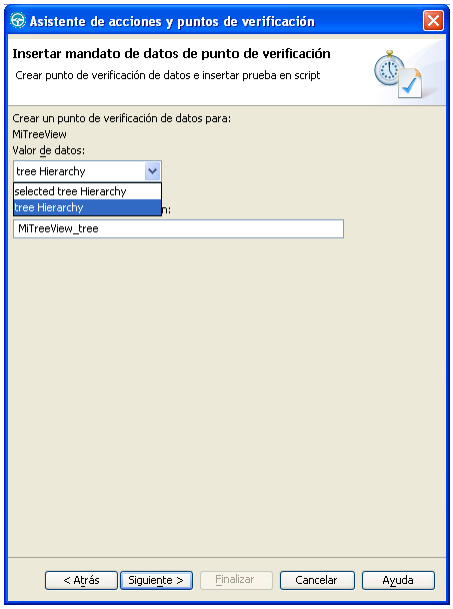
En la aplicación utilizada en el caso de estudio, no es posible utilizar los puntos de verificación de datos que nos ofrece RFT, debido a que los controles usados en la aplicación no son estándares, sino que o bien son controles personalizados o bien son controles de Infragistics. Por tanto, si no es posible usarlos, no se puede saber si la prueba ha sido pasada, así que se tiene que buscar otra solución para saber si el resultado de la aplicación al ejecutar la prueba es el esperado.

A continuación, se muestra un ejemplo de un control de la aplicacion en el que no se puede insertar un punto de verificación y otro control estándar del mismo tipo que sí que es posible.

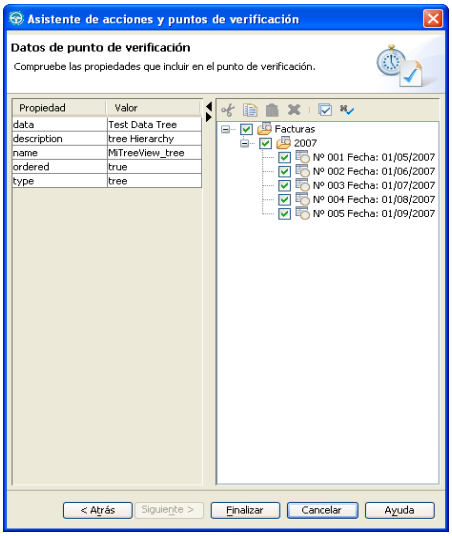
Este es un treeview estándar en Visual C#.



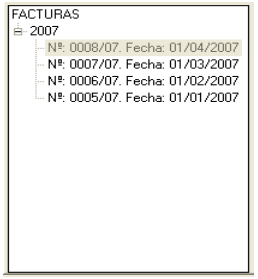
Al insertar el punto de verificación en el treeview, se puede elegir como valor de datos tree hierarchy para comprobar todos los datos de contiene el treeview, como podemos ver a continuación:



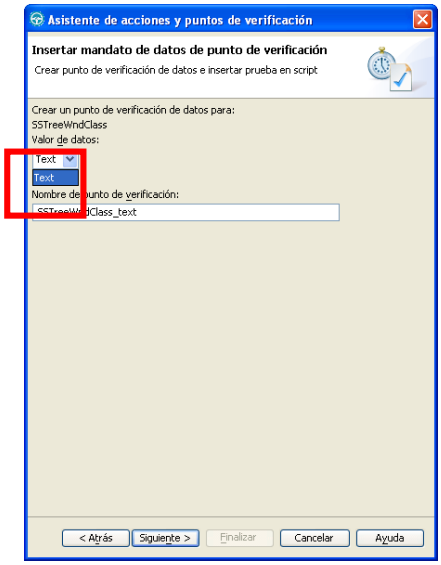
Al pulsar Siguiente, se puede ver el contenido y elegir los ítems en los que hacer la comprobación como vemos en el siguiente pantallazo:



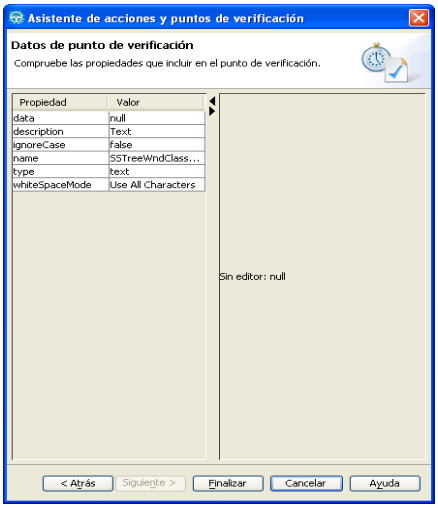
Este es un treeview utilizado en la aplicación:



Al insertar un punto de verificación para comprobar los datos, solo se puede se puede elegir como valor de datos la propiedad Text, como vemos en el siguiente desplegable:



Y como se puede ver en la siguiente imagen, no aparece el contenido del control, como aparecía en el control anterior, por lo que no es posible comprobar los datos que tiene el treeview:



Una posible solución para saber si el resultado es correcto, es haciendo una captura de pantalla al control o controles que muestran el resultado y compararlas con otras previamente capturadas y validadas por el tester, de esta manera se puede obtener puntos de verificación. Además, sería interesante poder comparar ficheros que nos genera el programa ha testear, como facturas, listados, etc, imprimiéndolas como imágenes mediante una impresora virtual. Como resultado, si la comparación falla, creará una imagen mostrando en color rojo en que se diferencian, para que el tester pueda saber rápidamente que es lo que falla.

Para hacer posible esta comparación, se ha utilizado un programa externo, es decir, una aplicación que no está en la herramienta de pruebas. Entonces, cada vez que se haga un punto de verificación, se debe hacer una captura del control e iniciar este programa para que se ejecute y nos indique si las capturas son semejantes.

En el caso de estudio, se ha implemento una prueba de sistema de una prueba de aceptación cuyo objetivo es comprobar la correcta facturación de cualquier tipo de residente (privado o beneficiario) con distintos bonos, es decir, que cualquier combinación de concierto.

Un residente es concertado, o tiene un concierto, cuando la Consellería le concede una determinada ayuda, generalmente parcial, a través de una resolución o ratificación anual con efecto retroactivo. En un día específico un residente puede ser beneficiario de una ayuda o privado.

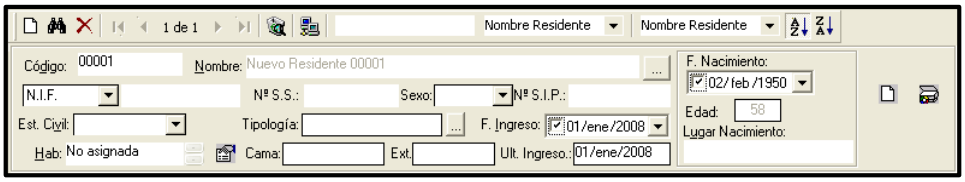
Existen las siguientes tipos de ayudas:

- Bono Residencia
- Accesibilidad Social
- PEI
- Centro de Día
- Bono Respiro Azul
- Bono Respiro Blanco
- Bono Respiro Verde

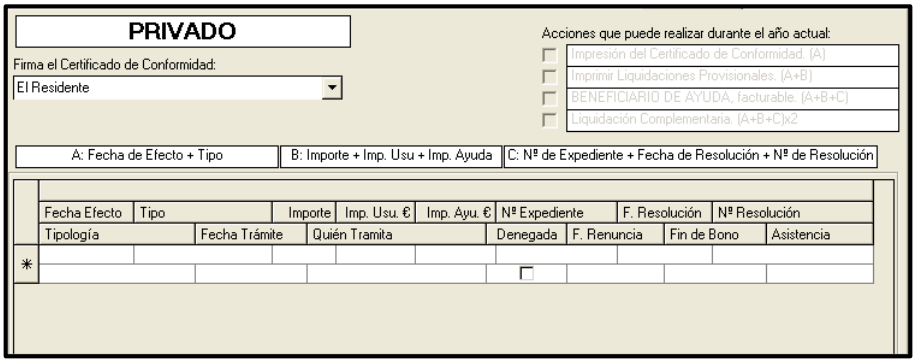
3.1. Organización del proyecto.

Antes de comenzar la grabación, hay que diseñar la organización del proyecto. Para ello, hay que dividir la aplicación en requisitos e interfaces de usuario y codificarlos. Las siguientes figuras, muestran la división y codificación de interfaces que se usaran en la prueba.

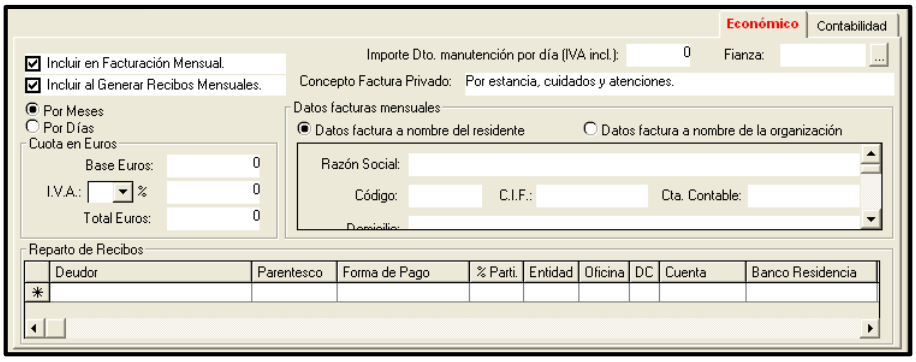
En esta interfaz es donde se realiza el logueo de la aplicación



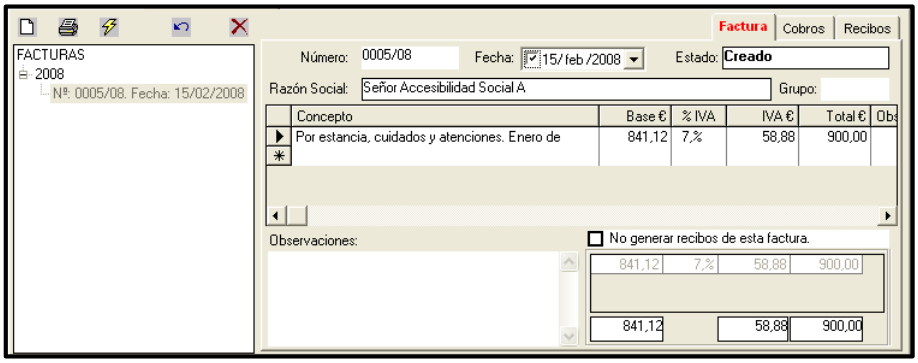
En esta ventana podemos ver que es donde se rellenan los datos del cliente, fecha de ingreso, código de identificación, DNI, etc…



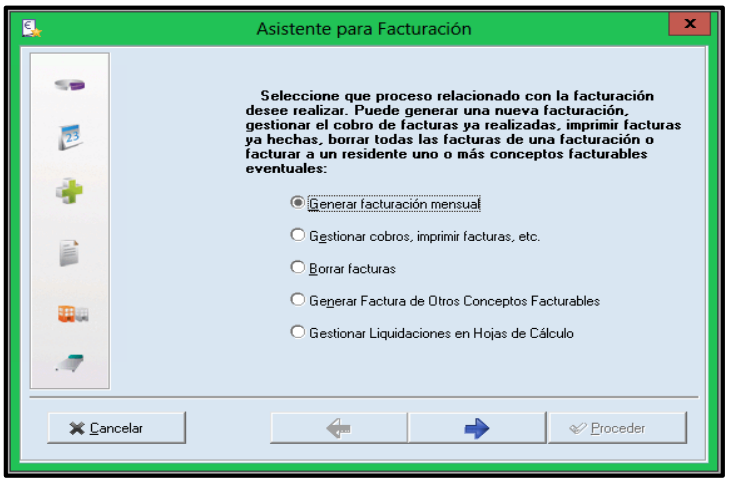
Vemos que aquí es donde se introducen las ayudas a los residentes, con datos como la fecha del trámite, tipo de ayuda, importe de ayuda, etc…



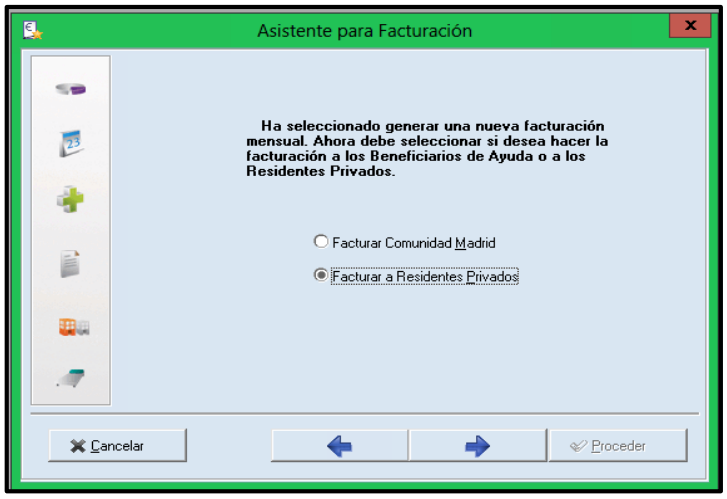
En esta interfaz de usuario encontramos distintos parámetros económicos referentes a la facturación y los recibos, podemos indicar si la factura esta nombre del residente o de una organización externa, podemos introducir datos de la organización si así lo fuera, etc…



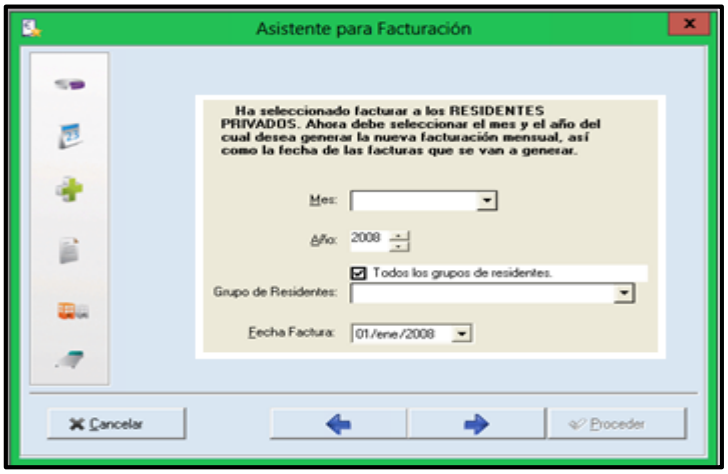
En esta interfaz es donde se crean las facturas para los residentes, donde se añaden los distintos conceptos de facturación y donde también se pueden crear nuevas facturas, entre otras acciones



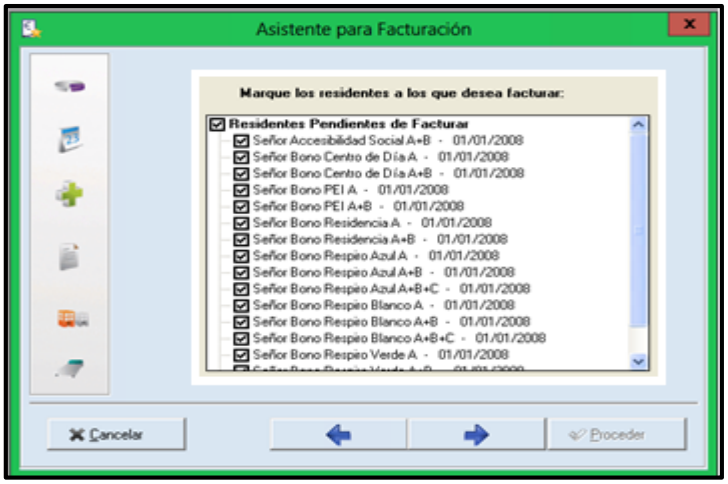
En esta ventana se selecciona el tipo de acción que deseas realizar en la facturación



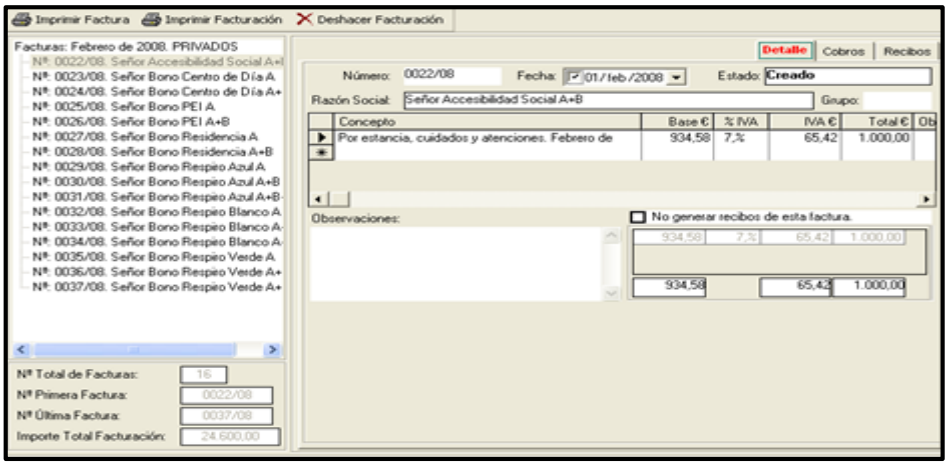
En esta interfaz de usuario se selecciona si la facturación se realiza a residentes privados o subvencionados por la comunidad en la que se encuentra la residencia, en este caso vemos la comunidad de Madrid



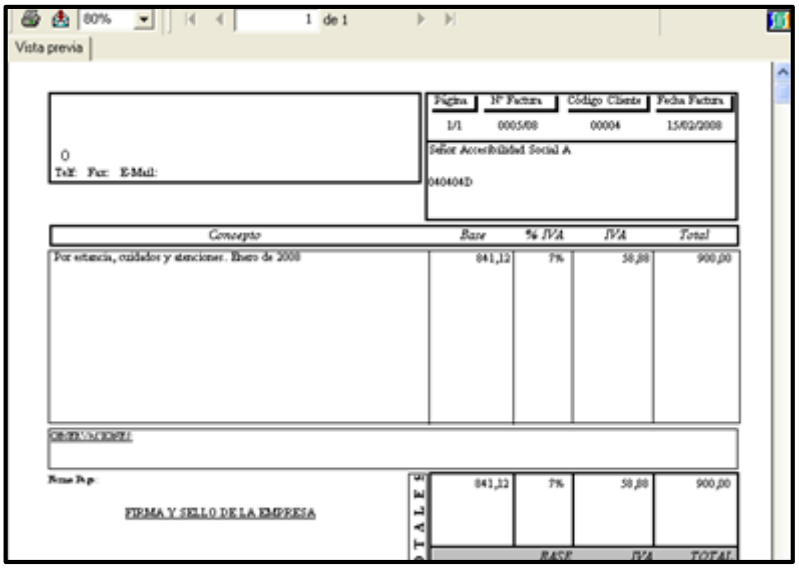
Se puede seleccionar el mes que se va a facturar, año, grupo de residentes o fecha de facturación



Aquí se pueden ver los residentes que van a ser facturados



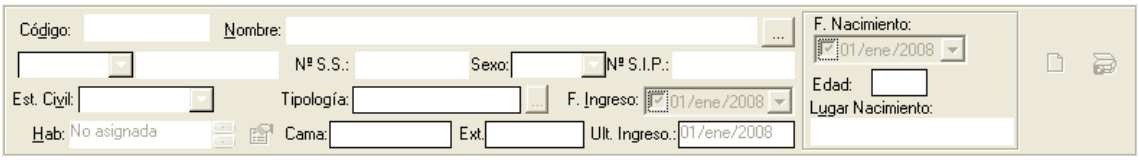
Aquí se gestionan las facturas y se ve el detalle de cada concepto individualmente

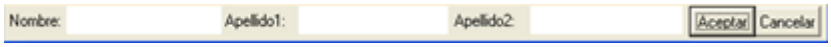


Este es un pantallazo de una factura ya impresa, pudiendo ser en papel o en formato digital

3.2. Creación de los scripts Interactuadores

Una vez organizado el proyecto, hay que hacer los scripts que interactúan. Para ilustrar la creación de estos scripts, se explica el proceso de cómo llevarlo a cabo mediante un ejemplo. El ejemplo consiste en la creación del script de interfaz de la introducción de datos personales de un residente. La interfaces correspondientes son estas:

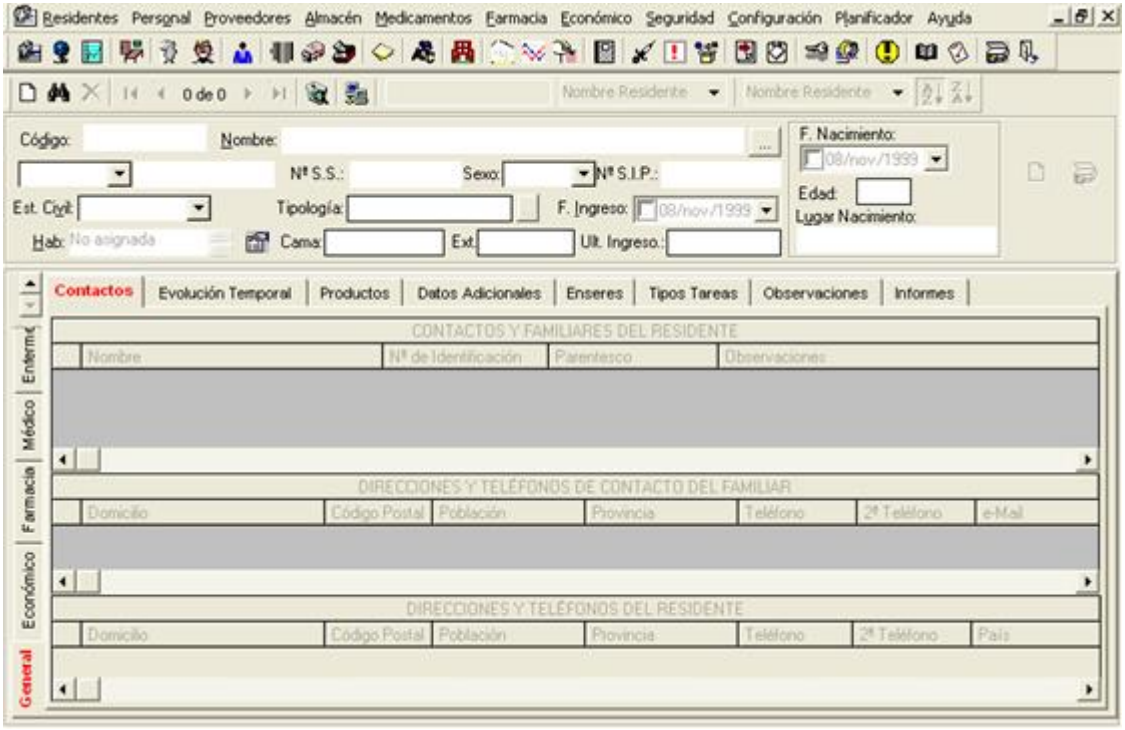




Se considera que la interfaz de introducción del nombre y apellidos formará parte de la interfaz de los datos personales de los residentes.

Grabación del script Interactuador

Antes de comenzar a grabar, hay que tener iniciado la aplicación y tener insertado un residente para poder grabar el script y poder utilizar los controles de la interfaz de datos personales de los residentes.



Cuando se tenga preparada la aplicación se puede empezar la grabación del script, como se puede ver en la imagen anterior ya está abierta la ventana de residentes.

Para hacer más sencillo el script, no se va a interactuar con todos los controles de la interfaz, solo se va a hacer con los necesarios para esta prueba. Se siguen estos pasos para hacer la grabación:

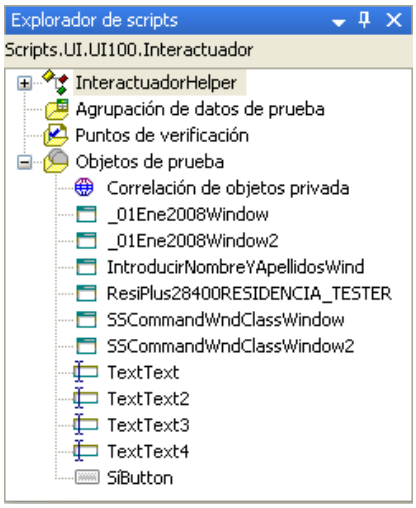
1. Seleccionar la carpeta **de scripts** en el explorador de soluciones y pulsar ● en la barra de herramientas.
2. Se abrirá la pantalla de grabación. El objetivo de la grabación es la obtención de los objetos de prueba y obtener las coordenadas de los clics en los controles necesarios. De momento, como solo necesitamos introducir el nombre, apellidos, DNI, fecha de ingreso y fecha de nacimiento de los residentes, solo se va a obtener estos objetos de prueba.

3. Se comenzará la grabación de los objetos de la aplicación.
Hacer clic en la barra de título de la aplicación para asegurarse de que se tiene el foco en la aplicación y escribir **Ctrl+F1**, así se creará el residente ya que no detecta el hacer clic en el botón **Nuevo**. Pulsar el botón… que está a la derecha del cuadro de texto del nombre, aparecerá en la ventana de introducción de nombre y apellidos. Hacer clic en el cuadro de texto del nombre y escribir cualquier cosa, luego hacer clic en los dos restantes cuadros de texto y el botón **Aceptar**. A continuación hacer clic en el cuadro de texto que está a la derecha de **N.I.F.** después hacer clic en el día en el control de la fecha de nacimiento, después pulsar dos veces la tecla del teclado de la flecha derecha para mover el foco hacia el mes y el año y luego pulsar el día de la fecha de ingreso. Pulsar con el teclado **Alt + r** y hacer clic en **Sí** cuando aparezca el mensaje "Modificar la fecha de ingreso supone cambiar la evolución temporal del residente, ¿está usted seguro de hacerlo?". Pulsar el botón  para parar la grabación.

El código que se habrá generado es el siguiente:

```
Public Function TestMain(ByVal args() As Object) As Object

        ' Window: ResiPlus.exe: ResiPlus 2.8.400 - RESIDENCIA_TESTER - [Residentes]
        ResiPlus28400RESIDENCIA_TESTER().Click(CAPTION)
        ResiPlus28400RESIDENCIA_TESTER().InputKeys("^{F1}")
        SSCommandWndClassWindow().Click(AtPoint(10,10))

        ' Window: ResiPlus.exe: Introducir Nombre y Apellidos
        TextText().Click(AtPoint(38,5))
        IntroducirNombreYApellidosWind().InputChars("blabla")
        TextText2().Click(AtPoint(68,9))
        TextText3().Click(AtPoint(65,14))
        SSCommandWndClassWindow2().Click(AtPoint(15,14))

        ' Window: ResiPlus.exe: ResiPlus 2.8.400 - RESIDENCIA_TESTER - [Residentes]
        TextText4().Click(AtPoint(12,9))
        _01Ene2008Window().Click(AtPoint(24,11))
        ResiPlus28400RESIDENCIA_TESTER().InputKeys("{ExtRight}{ExtRight}")
    _01Ene2008Window2().Click(AtPoint(21,9))
    ResiPlus28400RESIDENCIA_TESTER().InputKeys("%r")

        ' Window: ResiPlus.exe: Confirmación
    SíButton().Click(AtPoint(41,9))

    Return Nothing
End Function
```

Los objetos de prueba que habrá introducido en el script son los que se ven en esta captura:



Modificación del script

Una vez obtenidos los objetos de prueba, se tienen que modificar para poder ser identificados más fácilmente. Para poder renombrar los objetos, hacer clic derecho sobre el objeto de prueba en el explorador de scripts y elegir la opción Renombrar. La longitud máxima del nombre es treinta caracteres. Se van a renombrar de la siguiente forma:

Nombre original del objeto de prueba	Nombre modificado del objeto de prueba
_01Ene2008Window	FechaNacimiento
_01Ene2008Window2	FechaIngreso
IntroducirNombreYApellidosWind	IntroducirNombreYApellidosWind
ResiPlus28400RESIDENCIA_TESTER	ResiPlusWindow
SSCommandWndClassWindow	BotonIntroducirNombre
SSCommandWndClassWindow2	AceptarIntroducirNombre
TextText	Nombre
TextText2	Apellido1
TextText3	Apellido2
TextText4	NumDocumento
SíButton	SíButton

También se tiene que modificar el script para que se pueda introducir datos de una agrupación de datos. Cuando se ha grabado el script, se ha escrito con el teclado, la razón de esta acción es para obtener el objeto de prueba de la ventana que está dicho control, ya que detecta el control de forma general y no especifica, entonces no se puede utilizar los métodos específicos de los objetos de prueba para introducir datos. Por tanto, para poder insertar datos, hay que situar el foco en el control y escribir los datos en la ventana. A continuación, hay un ejemplo de código de cómo hacerlo:

```
'No es posible utilizar este método para introducir datos
'ya que no está disponible al detectado como un control general
'y no como un textbox
Nombre().SetText("Juan")
```

```
'Para poder introducir datos hay que situar el foco en el control
'y escribir el texto en la ventana
Nombre().Click()
IntroducirNombreYApellidosWind().InputChars("Juan")
```

En el caso del control de la fecha, como es un control no estándar y es detectado como un control general, no se puede utilizar ningún método para introducir datos. Para poderlo hacer hay que utilizar el sistema anterior. Pero este control hay insertar la fecha de forma distinta, primero hay que escribir el día, después hay que desplazar el foco a la derecha para insertar el mes y por ultimo desplazar el foco a la derecha otra vez para introducir el año. Como hay muchos controles de fecha de este tipo, se utiliza una clase para poder escribir fechas en cualquier control de la aplicación sin duplicar código. Esta clase será la siguiente:

```
Imports Rational.Test.Ft.Object.Interfaces

Public Class ResiDateTimerPicker

    'Atributos
    'Control que representa el date Timer picker
    Private datetimerpicker As GuiSubitemTestObject
    'Control que representa la ventana principal
  Private ventana As TopLevelSubitemTestObject

    'Constructor
  Public Sub New(ByVal ventana As TopLevelSubitemTestObject, ByVal
datetimerpicker As GuiSubitemTestObject)
        Me.ventana = ventana
        Me.datetimerpicker = datetimerpicker
  End Sub

    'Introduce una fecha con formato DD/MM/AAAA en el control
    Public Sub seleccionarFecha(ByVal fecha As String)

        'Obtenemos un array con el día, mes y año
        Dim array_fecha As String()
        array_fecha = fecha.Split("/")

        'Posicionamos el foco en el día
        datetimerpicker.Click(New System.Drawing.Point(23, 9))
        'Introducimos el día
        ventana.InputChars(array_fecha(0))

        'Movemos el foco en la posición del mes pulsando flecha derecha
        ventana.InputKeys("{ExtRight}")
        'Introducimos el mes
        ventana.InputChars(array_fecha(1))

        'Movemos el foco en la posición del año pulsando flecha derecha
        ventana.InputKeys("{ExtRight}")
        'Introducimos el año
```

```
            ventana.InputChars(array_fecha(2))

    End Sub
End Class
```

Modificando el script para aceptar datos de una agrupación de datos, queda de la siguiente manera:

```
Public Function TestMain(ByVal args() As Object) As Object

        'Obtenermos el foco
        ResiPlusWindow().Click(CAPTION)
        'Insertamos el residente
    ResiPlusWindow().InputKeys("^{F1}")

    'Abrimos la ventana de introducción de nombre y apellidos
    BotonIntroducirNombre().Click(AtPoint(10,10))

    'Introducimos el nombre
    Nombre().DoubleClick()
    IntroducirNombreYApellidosWind().InputChars(DpString("Nombre"))
    'Introducimos el primer apellido
    Apellido1().DoubleClick()
    IntroducirNombreYApellidosWind().InputChars(DpString("Apellido1"))
    'Introducimos el segundo apellido
    Apellido2().DoubleClick()
    IntroducirNombreYApellidosWind().InputChars(DpString("Apellido2"))
    'Aceptamos
    AceptarIntrodudirNombre().Click()

    'Introducimos el número de documento
    NumDocumentacion().Click()
    ResiPlusWindow().InputChars(DpString("NumDocumento"))

    'Introducimos la fecha de nacimiento
    Dim fechaNacimientoResi As ResiDateTimerPicker = New
        ResiDateTimerPicker(ResiPlusWindow, FechaNacimiento)
    fechaNacimientoResi.seleccionarFecha(DpString("FechaNacimiento"))

    'Introducimos la fecha de ingreso
    Dim fechaIngresoResi As ResiDateTimerPicker = New
        ResiDateTimerPicker(ResiPlusWindow, FechaIngreso)
    fechaIngresoResi.seleccionarFecha(DpString("FechaIngreso"))

    'Intentaremos abrir el menú para perder el foco
    'esta acción hará que salga el mensaje de confirmación
    ResiPlusWindow().InputKeys("%r")

    'En el mensaje de confirmación se hace clic en Sí
    SíButton().Click()

    Return Nothing
End Function
```

Todo este proceso se tiene que llevar a cabo en todos los scripts que interactúan del proyecto.

3.3. Creación de la prueba de sistema.

La prueba de sistema consiste en crear veintiún residentes con distintos tipos de concierto y bonos, después hacer la facturación de un mes y luego comprobar que las facturas generadas son correctas. Pero antes, hay una precondición que se debe cumplir, y es que la aplicación no tenga insertado ningún residente o factura, para asegurar que las facturas generadas en la prueba y las facturas guardadas como punto de verificación sean iguales. Así que antes de iniciar la aplicación se ha hacer una restauración de la base de datos con una vacía. Los pasos de la ejecución de la prueba de sistema seria de la siguiente manera:

1. Restaurar base de datos por una vacía.
2. Iniciar la aplicación.
3. Creación de los veintiún residentes.
4. Para cada residente, insertar un concierto.
5. Generar la liquidación y facturación automática de los residentes beneficiarios al insertar el concierto.
6. Introducir la cuota en los residentes privados.
7. Generar las facturas de los residentes privados mediante el asistente.
8. Imprimir las facturas de los residentes en imagen.
9. Cerrar aplicación.
10. Hacer la comparación de las facturas generadas con las facturas punto de verificación.

La implementación del script Manager es como si se hiciera un puzle donde los scripts utilizados fueran las piezas y el resultado final de la combinación fuera la prueba de sistema. El código del script Manager seria este:

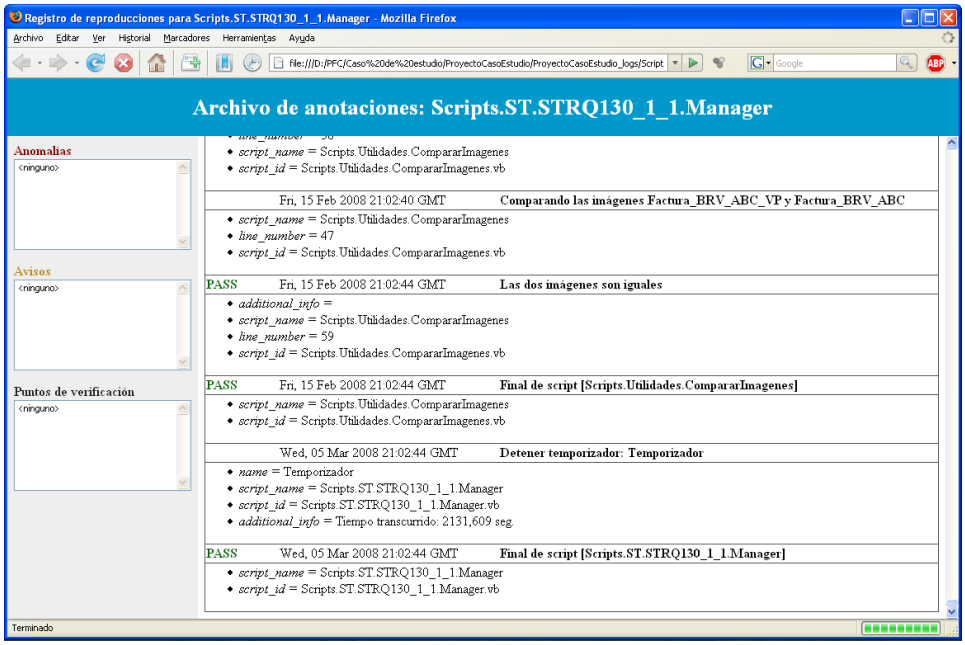
```
Public Function TestMain(ByVal args() As Object) As Object

    'Restauramos las bases de datos por unas vacias
    CallScript("Scripts.Utilidades.RestaurarDB")
    'Iniciamos la aplicación
    CallScript("Scripts.RQ.RQ001.IniciarAplicacion")
    'Hacemos el login
    CallScript("Scripts.RQ.RQ100.Login")
    'Cambiamos la fecha del sistema
    cambiarFecha(New Date(2008, 2, 15))
    'Abrimos la ventana de residentes
    CallScript("Scripts.RQ.RQ110.Menu")
    'Insertamos los residentes
    CallIterator("Scripts.UI.UI100.Interactuador",
"Scripts\ST\STRQ111_1_1\datapool.rftdp")
    'Insertamos los concertados
    CallIterator("Scripts.UI.UI101.Interactuador",
"Scripts\ST\STUI101_1_1\datapool.rftdp")
    'Insertamos las cuotas de los residentes privados
    CallIterator("Scripts.UI.UI103.Interatuador",
```

```
"Scripts\ST\STUI103_1_1\datapool.rftdp")
    'Facturamos los residentes privados con el asistente
    CallScript("Scripts.UI.UI130.Menu")
    'Elegimos la opción Generar facturación mensual
    CallScript("Scripts.UI.UI130.Interactuador", New Object() {1})
    'Elegimos la opción Facturar a residentes privados
    CallScript("Scripts.UI.UI131.Interactuador", New Object() {2})
    'Elegimos facturar los residentes del mes de enero,
    'poniendo la fecha de factura el 01/02/2008
    CallScript("Scripts.UI.UI132.Interactuador", New Object() {1, New Date(2008, 2,
15)})
    CallScript("Scripts.UI.UI133.Interactuador")
    CallScript("Scripts.UI.UI134.Salir")
    'Salimos del asistente
    CallScript("Scripts.UI.UI130.Salir")

    'Obtenemos el iterador de la agrupación de datos
    cambiarDatapool("Scripts\ST\STRQ130_1_1\datapool.rftdp")
    'Hasta que no hayamos recorrido todas las instancias
    While Not DpDone()
        'Buscamos el residente
        CallScript("Scripts.RQ.RQ113.Buscar",NewObject(){DpString("DniResidente")})
        'Seleccionamos la factura del residente y pulsamos imprimir
        CallScript("Scripts.UI.UI105.ImprimirFactura", New Object() {1})
        'Configuramos la impresora virtual

    CallScript("Scripts.Utilidades.ConfigurarImprimir",NewObject(){"Scripts\ST\STRQ1
30_1_1\VP\", DpString("NombreFactura")})
        'Imprimimos la factura y cerramos
        CallScript("Scripts.UI.UI150.ImprimiryCerrar")
        'Pasamos a la siguiente instancia
        DpNext()
    End While

    'Cerramos la aplicación
    CallScript("Scripts.RQ.RQ001.CerrarAplicacion")

    'Inicializamos el iterador de la agrupación de datos
    DpReset()
    'Hasta que no hayamos recorrido todas las instancias
    While DpDone()
        'Comparamos la factura del residente
        CompararImagenes("Scripts\ST\STRQ130_1_1\VP\",
DpString("NombreFactura"))
        'Pasamos a la siguiente instancia
        DpNext()
    End While

Return Nothing
```

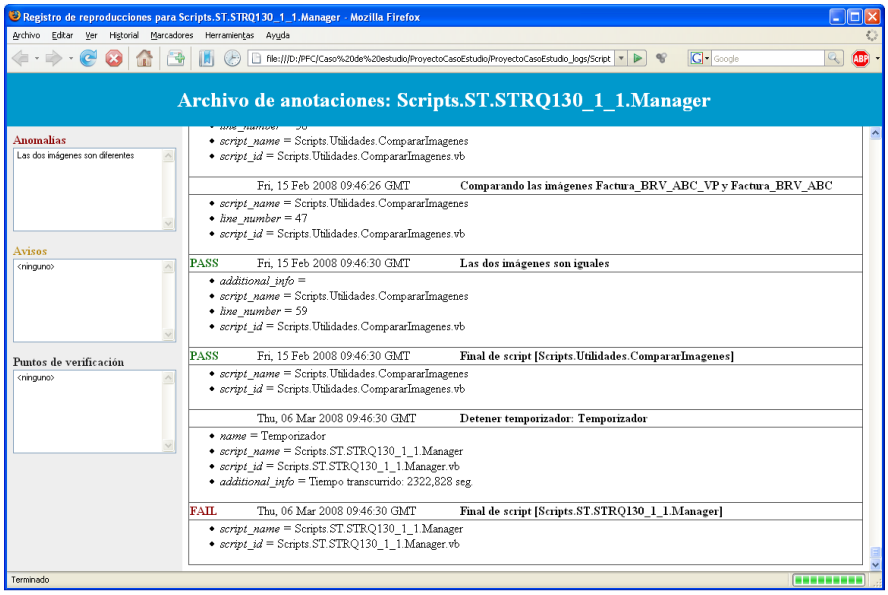
Una vez que se tenga implementado el script Manager de la prueba, hay que ejecutarlo sin comparar las imágenes de punto de verificación de las facturas, para poder generarlas y comprobar que sean correctas y también asegurarse de que el script se reproduce correctamente.

3.4. Ejecución de la prueba en RFT.

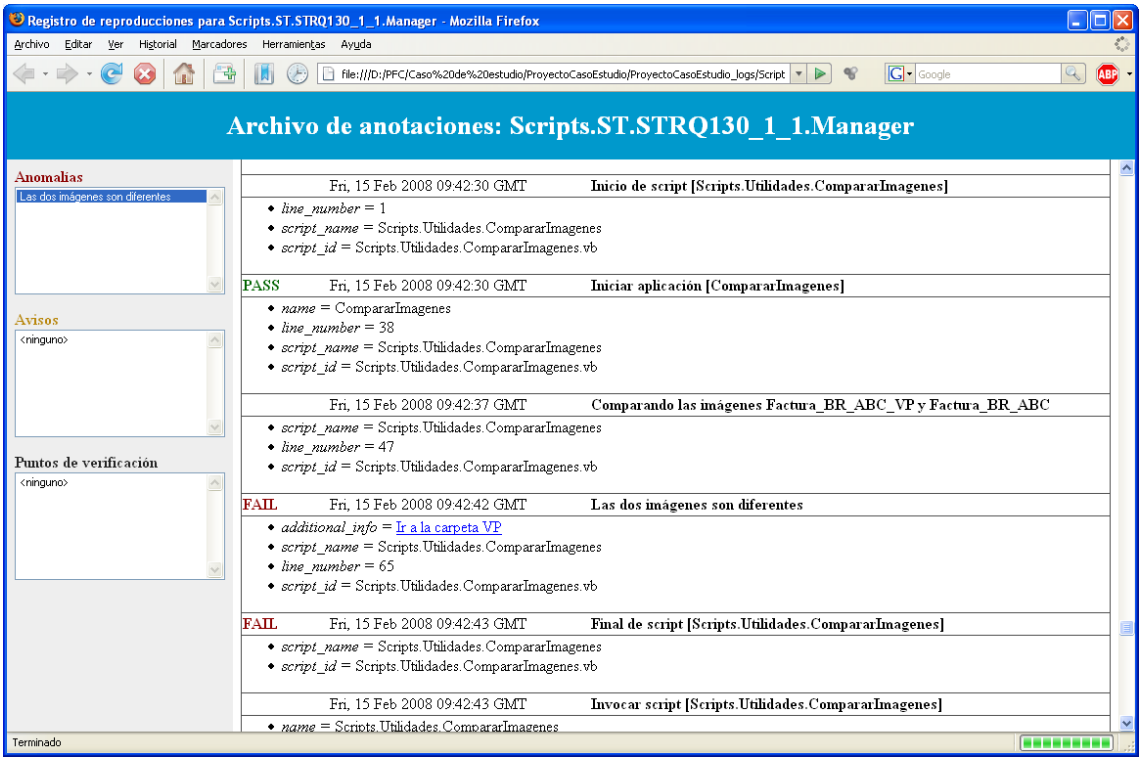
Cuando se haya acabado de implementar el script Manager de la prueba y de verificar que funciona correctamente, la prueba ya puede ser ejecutada posteriormente para pruebas de regresión.

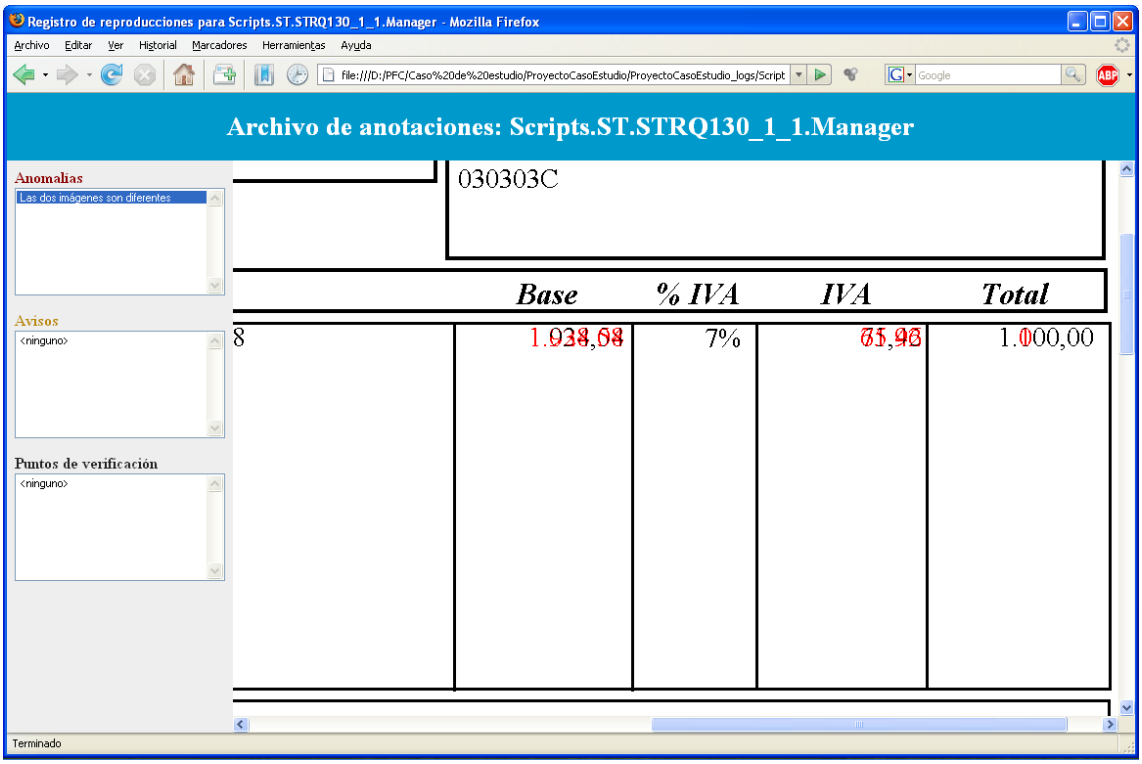


Pero, ¿qué pasaría si en pruebas de regresión, se detecta que la generación de las facturas para un tipo de bono fuera errónea? ¿Cómo se reflejaría este error en el registro de la prueba? A continuación, se va a mostrar un ejemplo del registro de la prueba cuando falla, debido a la incorrecta facturación de un residente con bono Residencia cuando es beneficiario.



El resultado de la reproducción de la prueba es que no ha pasado. En la lista de anomalías se puede ver que es lo que ha fallado. Haciendo doble clic en la anomalía, nos posiciona donde está el fallo. Haciendo clic en el enlace “Ir a la carpeta VP” se muestra el contenido de la carpeta VP de la prueba, pudiendo ver la imagen resultado de la comparación.





4. Proceso de desarrollo basado en PAs

El ERP de nuestro contexto de trabajo se lleva desarrollando desde 1997 y se está adaptando continuamente a las necesidades del sector. Debido a esos continuos cambios, se plantearon la idea de hacer pruebas automatizadas para saber si dichos cambios podrían afectar la funcionalidad de los requisitos ya implementados, y si ocurre, poderlos arreglar antes de que la versión sea entregada al cliente, el proceso implantado se caracteriza por los siguientes aspectos:

- **Modelo de proceso iterativo e incremental** para el desarrollo y mantenimiento del software. El trabajo se divide en unidades de trabajo (WUs) que son asignadas a versiones. Las versiones son frecuentes y de corta duración, entre 3 y 6 semanas dependiendo del producto.

- **Proceso de desarrollo dirigido por las pruebas** (Test-Driven). La definición de una WU es básicamente la especificación de sus pruebas de aceptación acordadas con el cliente. A partir de ahí, todo el proceso gira en torno a ellas, se estima el esfuerzo de implementar el comportamiento asociado y de aplicar las pruebas, se diseñan las pruebas e implementa dicho comportamiento, y finalmente, se aplican las pruebas sobre el producto para garantizar su correcta implementación.

- **Workflows flexibles para la coordinación del trabajo asociado a cada unidad de trabajo**. Los productos, según sus características, tienen asociados un conjunto de workflows que son utilizados para realizar WUs. Cada WU sigue el flujo de actividades del workflow. Bajo ciertas condiciones se permite saltar hacia adelante o hacia atrás en el workflow, así como cambios de agentes asignados e incluso cambio de workflow. Por ejemplo, las típicas situaciones de re-trabajo en desarrollo de software ocasionadas por detección de defectos pueden originar saltos atrás no explícitos en el workflow. Otras facilidades en cuanto a flexibilidad es permitir trabajo en paralelo o incluso añadir actividades no contempladas en la definición del workflow. Esta flexibilidad a su vez evita que la especificación del workflow se complique añadiendo explícitamente todas las situaciones posibles. Los workflows en SAPI actúan como guías esenciales del proceso pero evitando las rigideces usuales de la tecnología y herramientas específicas en el ámbito de workflows. En los workflows y mediante su refinamiento se va plasmando la mejora continua de proceso.

- **Planificación y seguimiento continuo**. En todo momento debe estar actualizado el estado de las versiones, de las WU, y del trabajo asignado a los agentes. El jefe del proyecto puede actuar oportunamente con dicha información, tomando decisiones tales como: negociar el alcance de la versión con el cliente, conseguir más recursos, redistribuir carga de trabajo entre agentes, cambiar los plazos de la versión, mover WUs entre versiones, etc.

- **Control de tiempos**. Los agentes registran el tiempo que dedican a la realización de las actividades, el cual se compara con los tiempos estimados en cada una de ellas, detectando oportunamente desviaciones significativas. Esto permite a los agentes gestionar más efectivamente su tiempo, mejorar sus estimaciones y ofrecer al jefe del proyecto información actualizada del estado de la versión.

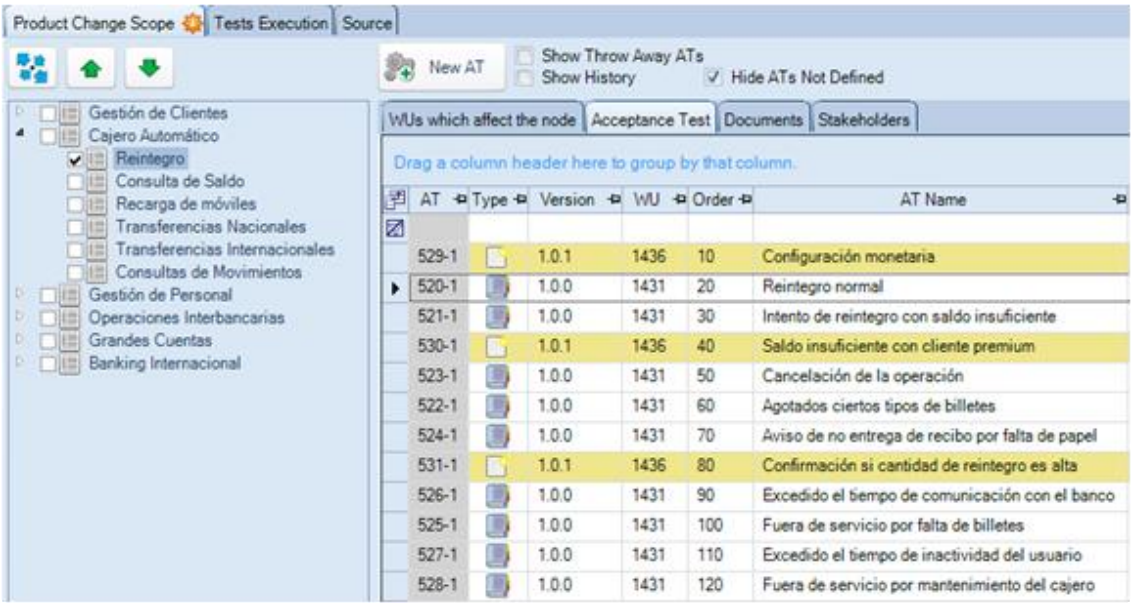
5. Diseño de PAs y pruebas automatizadas

SAPI es un sistema de apoyo al proceso de incidencias que incorpora aspectos de metodologías ágiles y de metodologías tradicionales. Las dos primeras características del proceso entre las antes mencionadas (proceso iterativo e incremental, y proceso centrado en las pruebas de aceptación) clasifican a SAPI como metodología ágil, sin embargo, las otras características están más próximas de lo que sería una metodología tradicional.

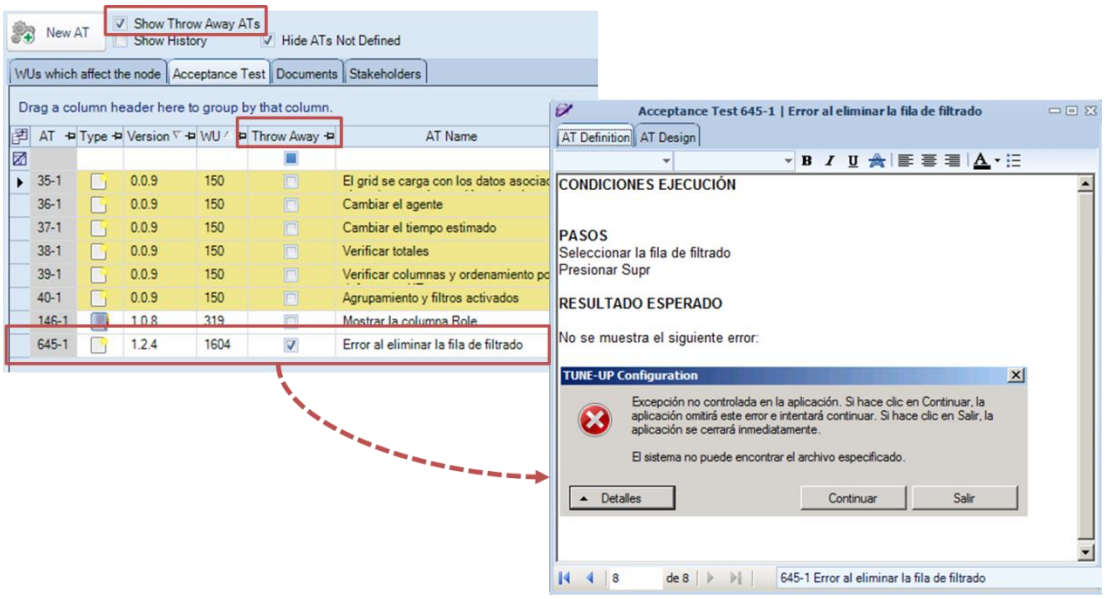
En SAPI todos los tipos de cambios de comportamiento se especifican de forma homogénea como PAs. Así, por un lado desde el punto de organización del trabajo todo tipo de cambio que está en una versión se aborda de forma integrada en cuanto a planificación y asignación de recursos. Por otra parte, desde la perspectiva de la especificación del cambio y su posterior verificación, en lugar de tener artefactos diferentes (requisitos/cambios/correcciones y PAs), se tiene sólo un artefacto llamado **Incidencia** o WU expresado en términos de PAs. SAPI está dirigido por las PAs. Es un sistema para la gestión del proyecto y especialmente de los requisitos del producto. El Analista define con el Cliente los cambios en el producto en términos de WUs y sus correspondientes PAs. El Programador escribirá código para satisfacer dichas PAs y finalmente el Tester aplicará pruebas para asegurar que el comportamiento del sistema satisface las PAs.

Dependiendo del tipo de WU y del proyecto en el cual debe llevarse a cabo el cambio de comportamiento de un producto, puede ser necesario realizar diferentes actividades y en diferente ordenamiento temporal.

El primer paso a realizar cuando el equipo de desarrollo o el cliente quieren hacer algún cambio de comportamiento dentro del producto es crear la WU correspondiente. Esta información es muy útil en el momento de proponer un nuevo cambio en el producto para por ejemplo, evitar solapes e inconsistencias entre cambios o programar de forma racional futuros cambios del producto prestando atención al conjunto de cambios pendientes de implementar.



En la siguiente imagen se puede ver un ejemplo:



Para cada WU se muestra la versión en la que se incluyó la WU, código, tipo, nombre, actividad actual en la que se encuentra dentro del workflow y los agentes asignados. SAPI permite además explotar la información de las PAs en el contexto de una unidad de trabajo desde la perspectiva del resto de los agentes que colaboran en su realización. El listado de PAs definidas, modificadas o marcadas de regresión que el analista ha definido. En el grid se destaca el estado de aplicación de la PA, registrado por los agentes en los diferentes niveles de testeo. Dichos **estados de aplicación de la PA** son:

- Diseñada
- Programada
- Testeada

Por ejemplo, cuando un programador implementa y aplica una PA, registra un seguimiento OK para indicar que se ha superado con éxito dicha PA. Un seguimiento está formado por la WU, Fecha de ejecución, Agente, Nivel de testeo, el Resultado (OK, KO y OK!), un Fichero explicativo sobre el resultado (conteniendo instrucciones para la reproducción de los defectos detectados) y un apartado para añadir comentarios. Una PA tiene como propósito demostrar al cliente el cumplimiento parcial o total de un requisito del software. Una PA describe un escenario de ejecución o de uso del sistema desde la perspectiva del cliente. Un requisito puede contener una o más PAs y éstas pueden estar asociadas tanto a requisitos funcionales como a no funcionales.

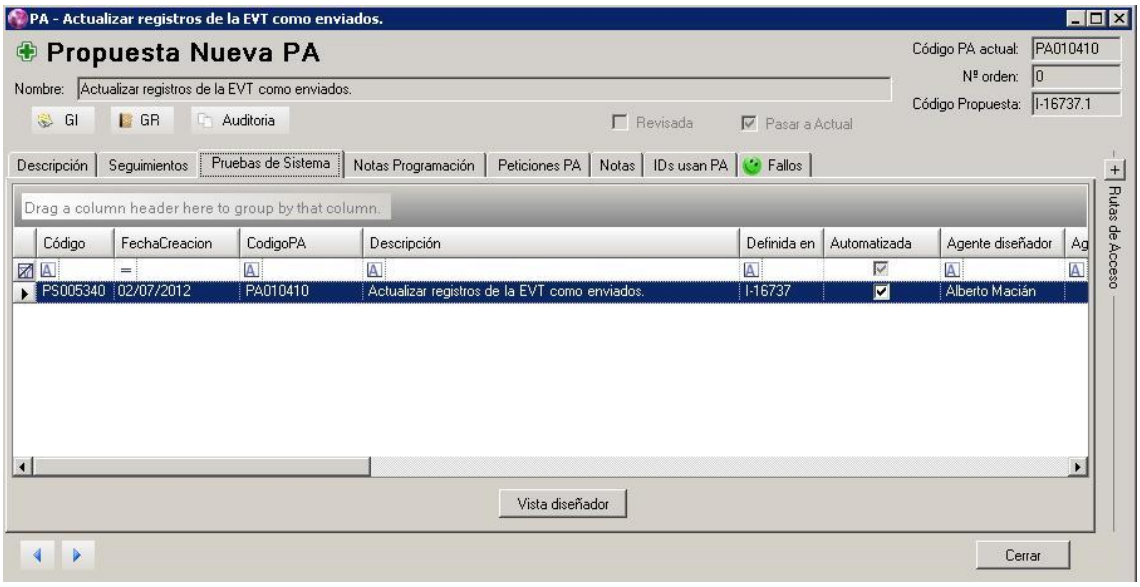
A continuación se presenta como ejemplo la definición de la PA “Intento de reintegro con saldo insuficiente” (se trata del contexto de la funcionalidad Reintegro en un cajero automático).

CONDICIÓN
Cliente del tipo normal
Cliente con saldo positivo
Acceder a ventana de reintegro
PASOS
Introducir cantidad mayor que el saldo
RESULTADO
Se muestra el mensaje “Saldo insuficiente”
Se ofrece nueva introducción

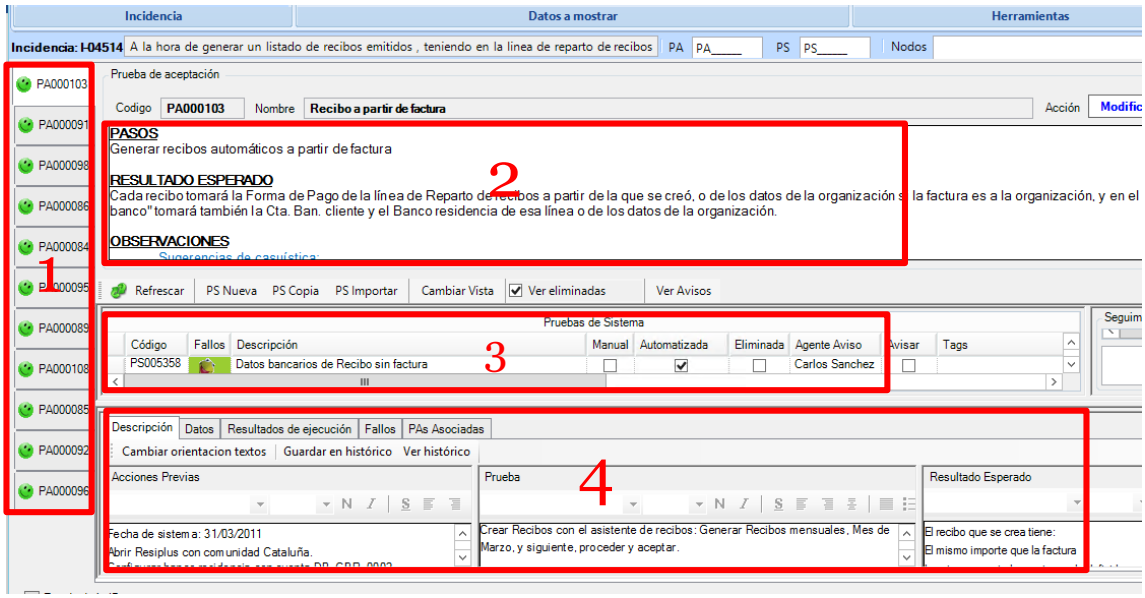
En SAPI la definición de una PA se compone de tres apartados: Condiciones, Pasos y Resultado.

Una WU tiene asignados los agentes encargados de desarrollarla. Así, el *Tester* encargado podrá comenzar a diseñar pruebas automatizadas desde el primer momento. El análisis de la WU, con lo que comprende el mismo, se realiza una versión anterior en la que se desarrollará, por ello, el *Tester* ya tendrá todas las PAs que se necesiten asociadas a la WU. Llegados a este punto, hay que tener claro qué criterios debemos seguir a la hora de decidir qué automatizar.

Para crear las nuevas PS el agente encargado deberá abrir el formulario de la PA y en la pestaña Pruebas de Sistema podrá empezar su tarea.



Para esta actividad contamos con una herramienta de apoyo, denominada *Diseñador de Pruebas de Sistema* (Diseñador de PS), de la que podemos ver a continuación una imagen:



Esta herramienta no es más que la parte diseñador de ATUN, la cual vamos a detallar a continuación. En el recuadro uno encontramos el área de las PAs asociadas a la incidencia, estas PAs indican con una carita de distinto color si están KO, OK o OK¡, el recuadro numero dos nos da una descripción de los distintos pasos, resultados y observaciones que tiene cada Ps, en el recuadro número tres aparece la Ps la cual puede esta automatizada o no y e indica en la columna Automatizada de la tabla del recuadro, con otras especificaciones, y en el cuarto y último recuadro encontramos el detalle de la prueba, con las acciones previas, en que consiste a prueba a alto nivel, y el resultado esperado.

Podemos ver también en este último recuadro otras pestañas que son "*Datos*", la que se encarga de almacenar ficheros ya sean de bases de datos como ficheros de resultado; en la pestaña "*Resultados de la Ejecución*", tenemos el log de esta prueba, indicando en tres colores, verde pasada, rojo prueba fallida y amarillo identificando una prueba como pasada pero con avisos (fallos pero de poca importancia).

- **Diseño de Pruebas Manuales / Preparar Entorno:**

  En esta actividad el *Tester* debe recoger las condiciones de la PA, sistema operativo, versión del motor de base de datos, versión del producto, variables con las que debe estar compilado, entre otras, y preparar el entorno de pruebas.

  Esta actividad también sirve de ayuda para la actividad de *Diseño de Pruebas Automatizadas*, sabiendo cuáles van a ser más costosas y, por lo tanto, convenientes de automatizar.

- **Aplicar Pruebas Manuales:**

  En esta actividad el *Tester* ya tiene el entorno de pruebas preparado y sabe qué pruebas debe realizar al producto; lo que debe hacer es pasar las pruebas manuales que ha diseñado.

Al mismo tiempo que vaya aplicando las pruebas al producto deberá marcar el resultado obtenido en la pestaña de *Test Execution*, reportando, en aquellos casos que en la prueba se obtuvo un resultado incorrecto, el comportamiento incorrecto de la aplicación a los agentes correspondientes, además de marcar el KO, se creará un fallo con la herramienta que se describirá en el próximo apartado, el *Gestor de fallos*.

- **Automatizar Pruebas:**

  En esta actividad, el *Automation Tester* tiene los diseños de las pruebas y se encarga de crear los scripts en el proyecto e implementar las pruebas de sistema para su posterior ejecución.

  Las pruebas de sistema contienen puntos de verificación donde la propia prueba deberá comprobar que el comportamiento del producto obtenido es el que se esperaba.

- **Aplicar Pruebas Automatizadas:**

  En esta actividad, el *Automation Tester* deberá preparar el entorno necesario para la ejecución de las pruebas de sistema automatizadas. Para llevar a cabo esta actividad, contamos con la herramienta Lanzador llamado ATUN, con la que podemos preparar un conjunto de pruebas de sistemas y ejecutarlas en las máquinas disponibles. Además, se puede consultar los resultados obtenidos de las pruebas de sistema que estamos ejecutando.

  Por otra parte, el *Automation Tester* es el encargado de revisar e interpretar los logs que se han generado en la ejecución; en la revisión puede encontrar un fallo en el producto que deberá reportar al *Programador* correspondiente, un problema en la automatización que deberá reportar al *Automation Tester* o si fuera un problema de diseño debería reportarlo al *Tester*, para llevar a cabo esta tarea contamos con una herramienta que facilita la gestión de fallos, el *Gestor de fallos*, que se encuentra integrado en la herramienta mencionada anteriormente (ATUN).

6. Infraestructura para pruebas de regresión

De esta manera encontramos una infraestructura en la cual ya tenemos todos los campos a tratar cubiertos. Para el diseño de pruebas para cada prueba de aceptación utilizaremos el Diseñador y el control de SAPI mencionado anteriormente con detalle para control de tiempos, historial de modificaciones, etc…

Para el control de fallos y el lanzamiento de pruebas utilizaremos ATUN y finalmente el sistema que hemos seleccionado para la automatización RFT (Rational Funtional Tester).

Ahora vamos a tratar más a fondo la herramienta ATUN que es lo único que nos falta por entrar en detalle, para hablar sobre los diferentes procesos que vamos a realizar en ATUN, tenemos que explicar un poco que es esta herramienta, ATUN es una herramienta de gestión de pruebas y control de las máquinas de ejecución, está compuesto de diferentes pestañas, configuración, filtro de pruebas, suite, histórico de logs, gestor de máquinas de ejecución, entre otras…

Para explicar las distintas pestañas se irán detallando según la utilidad que tienen en dos grupos, **lanzamiento de pruebas, gestión de pruebas fallidas.**

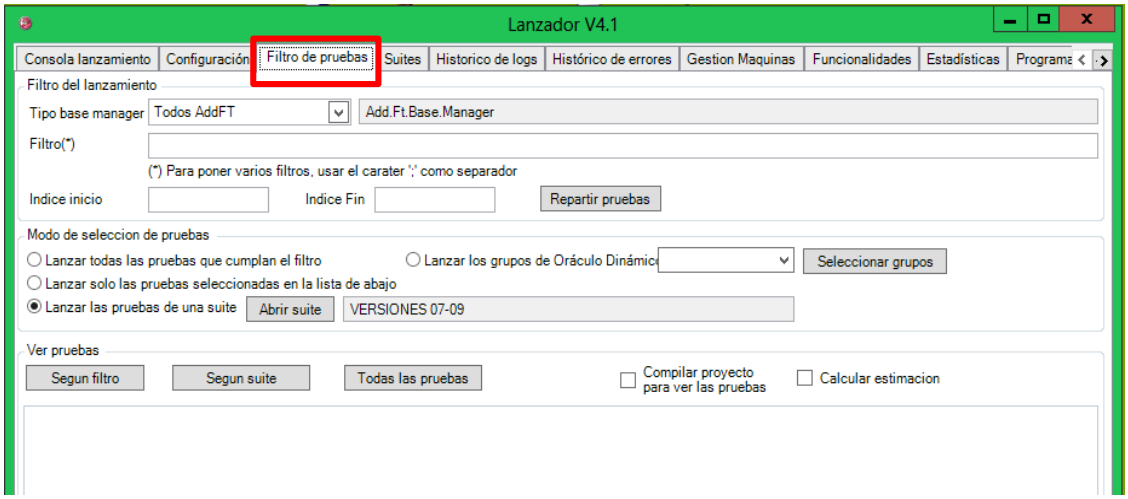
6.1. Lanzamiento de pruebas ATUN

El proceso de lanzamiento de pruebas consiste básicamente en tres pasos:

- ***Selección de pruebas***: En este apartado tenemos que tener en cuenta distintos tipos de pruebas, están las ***pruebas a demanda***, donde un tester u otro miembro del equipo de desarrollo necesita que ciertas pruebas automatizadas sean pasadas, de esta manera verificar si los cambios que ha realizado no afectan al funcionamiento de otros componentes. Para este caso se realiza una incidencia asociada al tester automatizador donde se indica en qué condiciones se deben pasar las pruebas, versión, variables de compilación activadas y precondiciones de BD que deben crearse para dichas pruebas. Las pruebas pueden agruparse ***mediante SUITE***, donde se eligen pruebas que afectan a cierto nodo de la aplicación, ECONOMICO, ENTIDADES, FACTURACIÓN, etc… o pruebas aisladas que verifiquen algo que se ha modificado.

  También tenemos las ***pruebas de regresión***, en estas se realiza un lanzamiento periódico donde se intenta acoplar unas semanas antes del paso de versión a producción, para ver si se encuentran fallos antes de que puedan llegar errores a los clientes.

  Después tenemos el ***lanzamiento global de pruebas***, donde de manera menos frecuente se realizan estadísticas y se recogen datos para ver que rendimiento se está teniendo tanto en pruebas automatizadas como número de errores detectados mediante pruebas automatizadas.

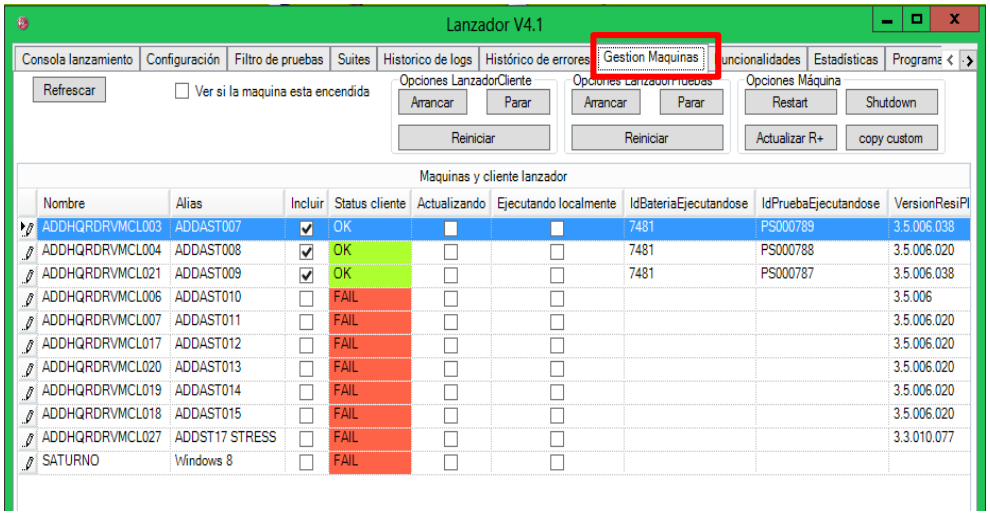


Aquí tenemos la pestaña de **filtrado de pruebas**, donde podemos elegir el tipo de filtro que queremos aplicar, ya sea pruebas individuales, por suites o todas las pruebas automatizadas como lanzamiento de regresión.

- ***Preparación de las máquinas y ejecutables***: Para la preparación de las máquinas y ejecutables se realiza un proceso bastante metódico para asegurarse de que no estamos obteniendo resultados erróneos, consiste en:

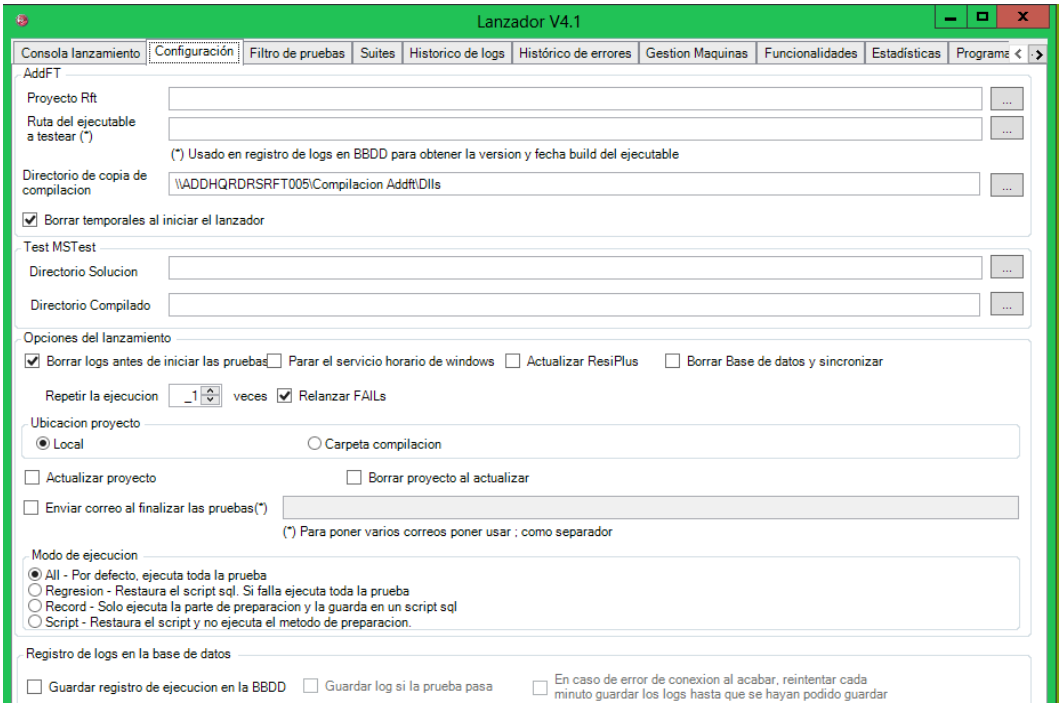
  1. Compilación de ejecutables con sus respectivas variables de compilación. Esta acción se realiza según las especificaciones que se encuentran en la incidencia o las correspondientes a la última versión del ejecutable con las variables de compilación que se están utilizando en producción junto a la variable de testing activa.

  2. Actualización de las maquinas tanto a nivel de bases de datos como con los nuevos ejecutables. Esta actividad se encarga de hacer que las máquinas que van a realizar la ejecución de las pruebas estén con las configuraciones, capa de persistencia y ejecutables adecuados.



En esta imagen podemos ver la pestaña de configuración de las maquinas encargadas de ejecutar las pruebas automatizada, donde se pueden apagar las máquinas, reiniciarlas, actualizar ResiPlus y supervisar el estado de cada máquina, que prueba está ejecutando cada máquina, si está actualizándose la máquina o la versión que está utilizando de la aplicación.

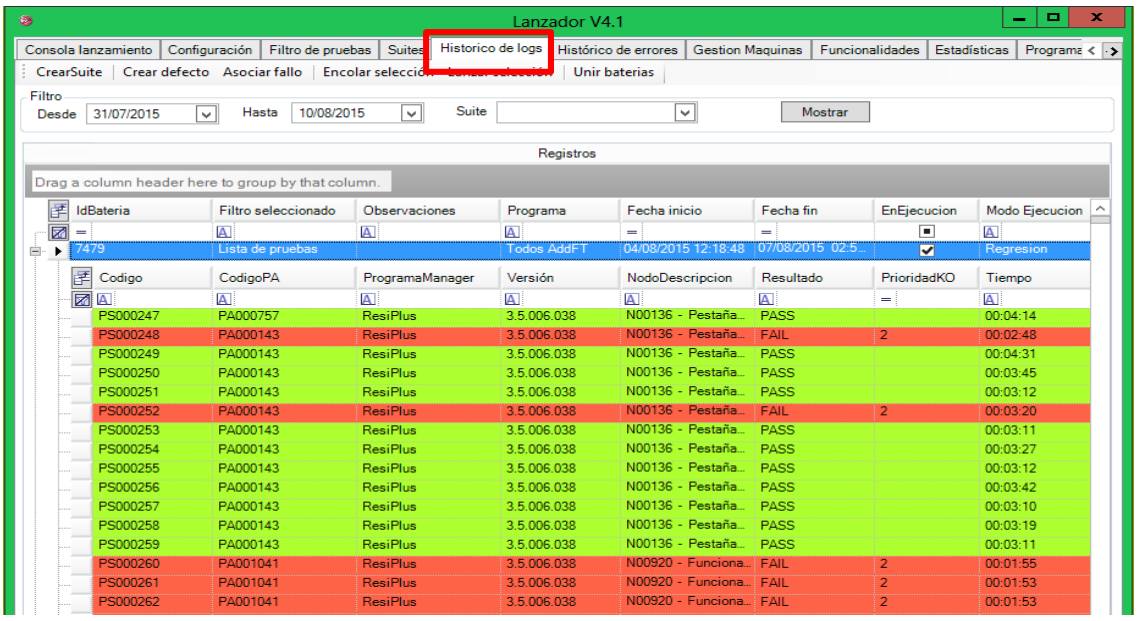
- ***Lanzamiento de pruebas***: En este último paso primero tienes que realizar las distintas configuraciones de lanzamiento, cuantas repeticiones realizara cada prueba la ejecución, si se realizara un borrado de los temporales antes del lanzamiento, la ventana de configuración la podemos ver en la siguiente imagen:



Ahora solo falta seleccionar las máquinas y realizar el lanzamiento de las pruebas elegidas mediante la aplicación en la pestaña de consola de lanzamiento.

6.2. Gestión de pruebas fallidas ATUN

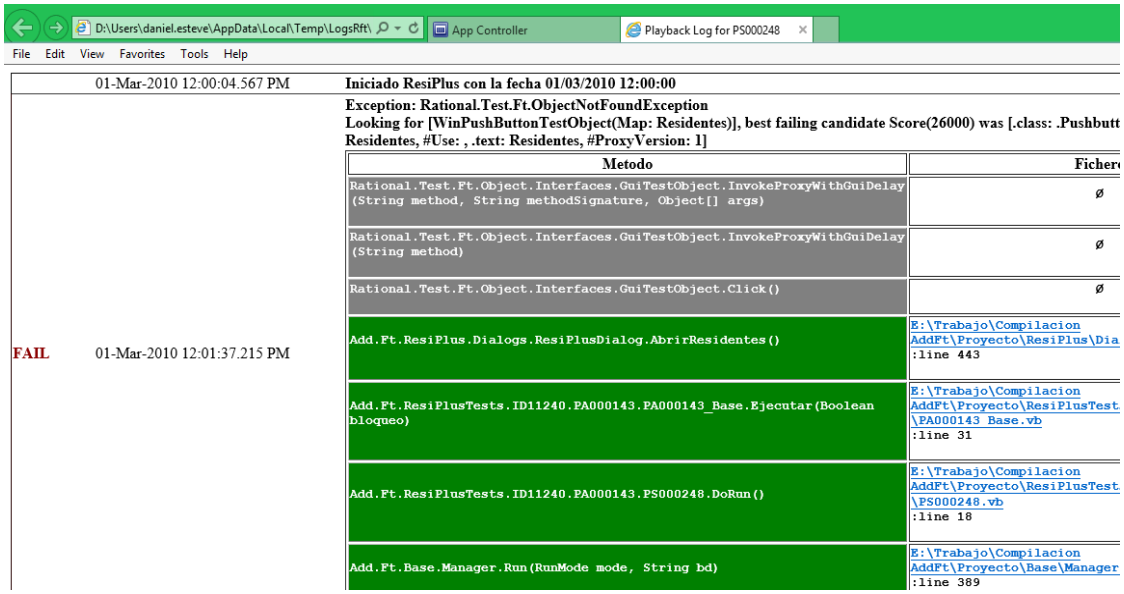
Para la gestión de fallos, en la aplicación tenemos una pestaña:



Aquí podemos filtrar las pruebas ejecutadas por fecha que hemos guardado en BD, donde encontraremos muchos detalles de cada una de las pruebas, para así tener una

mejor perspectiva y toda la información posible sobre las pruebas, como se puede apreciar en la figura anterior, errores, versión, tiempo de ejecución, etc…

Aquí el tester de automatización es el encargado de ir recogiendo estos errores y determinar cuál es la causa del fallo de la prueba, para ello solo seleccionamos alguna prueba fallida y nos aparecerá el siguiente log de fallo:



En este log encontramos tres tipos de fallos, fallos ocasionales, fallos de la prueba y fallos de la aplicación.

En los ***fallos ocasionales*** simplemente tenemos que volver a realizar otra ejecución de la prueba ya que se trata de un fallo ajeno a la aplicación o a la prueba, siendo un fallo de la máquina, ya sea por recursos o por aspectos ajenos a la ejecución. En los ***fallos de la prueba*** se detecta la zona del código de la prueba donde no se está reconociendo algún control de manera adecuada o a cambiado el funcionamiento de la prueba, ya que se trata de un sistema (RFT) que realiza un reconocimiento gráfico. Si se trata de algún ***fallo de programación***, se realizar un reporte mediante SAPI sobre cómo se reproduce el error, en que está fallando y es dirigido este reporte a un miembro del equipo de testeo para realizar una confirmación del fallo y su posterior corrección pro un programador.

7. Conclusiones

Aún existe desarrollo de productos SW sin una actividad de testeo elaborada y adecuada. En los últimos años está aumentando la buena práctica de utilizar recursos en disponer de un equipo de testeo que haga el trabajo específico y adecuado para asegurar el buen funcionamiento de los productos y entrega de software de calidad.

La actividad de testeo automatizado tiene un papel importante en el ciclo de desarrollo; Una de las maneras de entregar un producto probado y desarrollado para el cliente es teniendo un buen proceso de testeo automatizado que ayude a una necesidad cada vez más demandada en la actualizada, calidad pero en tiempos cada vez más cortos, junto con una buena infraestructura detrás donde poder apoyarse a lo largo del mismo de manera ágil y cumpliendo plazos establecidos con los clientes.

Se ha definido un proceso de testeo basado en PAs y apoyado por herramientas con las cuales se ha documentado dicho proceso e ilustrado con ejemplos para su mejor comprensión. Como experiencia personal puedo decir que tras haber realizado este TFG he aprendido a valorar lo importantes que es aplicar testeo automatizado al desarrollo software y viendo que al final no es solo teoría, sino que en la práctica funciona y da resultados.

Como ejemplo voy a contar la transición que ha tenido que realizar la empresa de la que soy becario, siendo una empresa dedicada a realizar aplicaciones para el control de pacientes en residencias, muchos de sus clientes han pedido tener más movilidad por parte del personal de las residencias, siendo un ordenador un sistema estático, por ello han empleado una buena parte del personal a realizar un sistema táctil para que personal de las residencias pueda tener un acceso de información del paciente a la hora de tratarlo sin depender de un ordenador, de una manera más práctica y totalmente actualizada. De esta manera han necesitado crear un módulo independiente para la automatización de dicho sistema, ya que no se trata de controles similares a los empleados para una aplicación de escritorio. Este es el motivo de la importancia que se tiene que tener a la evolución de aplicaciones software, siempre intentando crear sistemas generalizados y con visión de posibles cambios que puedan ocurrir, por lo menos a un corto, o medio plazo, para así poder reutilizar todo lo que se pueda en implementaciones futuras. En este caso la importancia del proceso de automatización se ha visto reflejada, ya que se han conseguido entregar versiones al cliente asegurando un porcentaje de calidad bastante alto y llegando a los plazos gracias a la utilización de dicho proceso.

8. Referencias

[1] Haugset, B., Hanssen, G., Automated Acceptance Testing: A Literature Review and an Industrial Case Study, Proc. of Agile 2008, pp. 27-38

[2] Pruebas software automatizadas para garantizar la calidad de una aplicación industrial (2008). PFC Héctor Sánchez Barba.

[3] Dashboard para testeo de aceptación en un contexto industrial (2012). TFM Miguel Peiro Garcia de la Reina.

[4] Manuales en OneNote sobre las distintas herramientas dispuestos por la empresa para el aprendizaje y correcta utilización de sus trabajadores.

[5] M. Company. Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación (2011). TFM Maria Company Bria.

[6] Beizer B. *Software testing techniques (2nd ed.)*, ISBN:0-442-20672-0, Van Nostrand Reinhold Co, 1990.

[7] Kaner C., Falk J., Nguyen H. *Testing Computer Software, 2nd Edition*, ISBN: 0471358460, Wiley, 1999.

[8] IEEE Standard Glossary of Software Engineering Terminology Institute of Electrical and Electronics Engineers,ISBN: 155937067X, 1990.

[9] Software Test Automation, Effective use of test execution tools. Mark Fewster & Dorothy Graham. ISBN 0-201-33140-3

[10] Black R. *Managing the Testing Process, 2nd Edition*. ISBN 0-471-22398-0, Editorial Wiley, 2002.

[11] International Software Testing Qualifications Board, Certified Tester Foundation Level Syllabus, Versión 2005.
http://www.istqb.org/fileadmin/media/SyllabusFoundation.pdf

[12] Dustin E., Rasca J., Paul J. *Automated Software Testing*, ISBN 0-201-43287-0, Addison Wesley, 1999.