UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Revisión de ejecuciones de pruebas automatizadas apoyada por grabaciones de vídeo

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Romero Molina, Ángel

Tutor/a: Letelier Torres, Patricio Orlando

CURSO ACADÉMICO: 2021/2022

A mi madre, por darme la oportunidad de poder estudiar en la universidad, apoyarme anímicamente y ofrecerme todos los recursos necesarios para finalizar la carrera.

Agradecimientos

Me gustaría agradecer a mi tutor Patricio por ofrecerme la posibilidad de poder entrar en el departamento de testing automatizado de la empresa donde he realizado mis prácticas y aconsejarme para realizar correctamente este trabajo. También a Miguel, mi compañero de departamento, por introducirme las herramientas de pruebas automatizadas de la empresa y dedicar su tiempo en ayudarme con la mejora del sistema de pruebas.

Por último, me gustaría mencionar a los compañeros de la carrera con los que he convivido durante estos cuatro años, los cuales han hecho mi experiencia mucho más amena y me han permitido ampliar mis conocimientos y experiencias en el campo del desarrollo de software, así como los buenos momentos que he pasado con ellos.

Resumen

La implantación de pruebas automatizadas en el área de testing del desarrollo de *software* ha significado una revolución dentro de la industria, reduciendo sus costes y aumentando la fiabilidad y rapidez de la ejecución de las pruebas. La retroalimentación que proporcionan estas pruebas es importante para asegurar el correcto funcionamiento del programa que ha sido testeado. El objetivo de este trabajo es combinar grabaciones de vídeo con un sistema de lanzamiento de pruebas automatizadas para mejorar los resultados proporcionados en sus ejecuciones. Este trabajo se ha desarrollado en el marco de una práctica de empresa, en el departamento de desarrollo donde se realiza el mantenimiento de su producto, un ERP para el sector sociosanitario.

Palabras clave: pruebas automatizadas, grabación de vídeo, mejora de sistema, testing, FFmpeg, .NET.

Abstract

The implementation of automated tests in the software development testing has meant a revolution within the industry and has meant a revolution within the industry, reducing costs and increasing the reliability and speed of test execution. The feedback provided by these tests is important to ensure the proper functioning of the program that has been tested. The objective of this work is to combine video recordings with an automated test launch system to improve the results provided in their executions. This work has been developed within the company internship, in the development department where the maintenance of its product is carried out, an ERP for the social and health sector.

Keywords: automated testing, video recording, system upgrade, testing, FFmpeg, .NET.

Tabla de contenidos

Tabla de figuras

1. Introducción

1.1 Motivación

Este trabajo tiene como motivación dotar a una infraestructura de pruebas automatizadas ya desarrollada de unos resultados de sus ejecuciones de mayor calidad, añadiendo en sus informes un archivo de vídeo mostrando todas las acciones realizadas por el *script* de la prueba en el programa testeado.

El uso de pruebas automatizadas para la validación y verificación de sistemas *software* en la industria ha adquirido protagonismo en los últimos años. Beneficios como la rapidez, la formalidad en las pruebas y la reducción coste en comparación con las pruebas manuales han incentivado más su implementación.

Como observamos en la Figura 1, la implementación de pruebas automatizadas supone una optimización considerable en el tiempo dedicado a las labores de *testing* en forma global. La inversión para su implementación es mayor en comparación con las pruebas manuales y se requiere de una ardua formación para los mantenedores de las pruebas. En cambio, se puede apreciar como el tiempo de ejecución de dichas pruebas y la generación de informes con los fallos encontrados es mucho menor de forma automatizada.

Test Steps	Manual Testing (hours)	Automated Testing (hours)	Percentage Improvement with Tools
Test plan development	32	40	–25%
Test procedure development	262	117	55%
Test execution	466	23	95%
Test result analysis	117	58	50%
Error status/correction monitoring	117	23	80%
Report creation	96	16	83%
Total duration	1090	277	75%

Figura 1. Esfuerzo estimado en horas por fase de testing con pruebas manuales vs pruebas automatizadas. [3]

La aplicación de pruebas de regresión ha facilitado la actividad de *testing* cuando se aplican cambios en un programa, asegurando que la nueva versión sigue funcionando conforme a lo esperado con los cambios introducidos. Automatizándolas, ha permitido que, en cuestión de poco esfuerzo y tiempo, se obtengan fallos no detectados en la aplicación debido a los cambios en el código fuente que se hayan producido.

La empresa donde se ha realizado este trabajo comercializa un ERP para el sector sociosanitario. De forma mensual se despliegan nuevas versiones del programa, las cuales contienen nuevos requisitos y correcciones sobre versiones anteriores. Antes del lanzamiento final a los clientes del *software*, tras pasar por los controles de calidad manuales de *testing*, se aplica una batería de pruebas automatizadas, donde verifican la

correcta funcionalidad del programa conforme a sus requisitos e informa de fallos no detectados durante las fases anteriores del desarrollo de la nueva versión.

La batería de pruebas se lanza sobre un conjunto de máquinas virtuales que, de forma paralela, ejecutan diferentes pruebas a la vez y, cuando acaba una prueba, muestra el resultado en un fichero HTML. Muchas veces, una ejecución no es suficiente para determinar el origen del problema. Por ello, es necesario realizar diversas ejecuciones completas de la prueba, sin posibilidad de retroceder en ellas y, en muchas ocasiones, dedicándole una atención completa a la ejecución de la prueba para comprender la naturaleza de los fallos encontrados.

Durante el proceso de revisión de los resultados de las pruebas automatizadas surgen muchos inconvenientes que ralentizan su proceso, como pueden ser falsos fallos, problemas en la infraestructura y diversas pruebas que fallan por un mismo defecto de la aplicación. A esto hay que sumarle que el sistema actual de la empresa cuenta con más de 6000 pruebas automatizadas, las cuales no solo hay que programar y poner en marcha en el registro de pruebas, sino mantenerlas y actualizarlas conforme a los cambios que se presentan en los requisitos de la aplicación y en los elementos que conforman sus formularios.

Debido a los problemas que estas pruebas pueden ocasionar, se requiere en diversas ocasiones a volver a ejecutar las pruebas, sin posibilidad de acceder directamente al momento del fallo. Una forma de evitar tener que relanzar una prueba automatizada y se pueda acceder de forma directa al momento en el que se detecta la anomalía es mediante la grabación de su ejecución en vídeo y mostrarlo en los ficheros HTML del informe. Esto permitiría reducir el esfuerzo de revisión y facilitar una mayor información a los departamentos implicados al utilizar el vídeo como adjunto en los sistemas de incidencias de la empresa con los fallos encontrados en el programa.

1.2 Objetivos

Este trabajo tiene por objetivo integrar grabaciones de vídeo en los resultados de ejecuciones de pruebas automatizadas. Las finalidades de implantar esta nueva funcionalidad en la infraestructura de lanzamiento de pruebas son:

- **Dotar de información más precisa** al término de las ejecuciones de las pruebas automatizadas. Al finalizar las pruebas se proporciona un archivo con el informe del resultado de la ejecución de la prueba. Con la inclusión del vídeo de su ejecución dará una información del resultado de mayor calidad y permitirá localizar de forma más clara la fuente del fallo encontrados.
- **Ahorrar recursos computacionales y tiempo** fuente de ejecuciones repetitivas de las pruebas. Cuando un usuario lanza las pruebas y necesita volver a ejecutar la prueba para comprender mejor su resultado mostrado, son recursos extra que se destinan para su ejecución. En cambio, al usar grabaciones de vídeo, el usuario puede consultar de forma más sencilla e inmediata su ejecución sin tener que lanzar la prueba de nuevo.
- **Mejorar la comunicación entre los departamentos de la empresa** con las grabaciones generadas por las pruebas. Al detectar un fallo en la aplicación durante la fase de *testing* automatizado se debe notificar al departamento de

desarrollo con la información recogida por la prueba automatizada. Adjuntando el vídeo que genera la ejecución aportaría mayor calidad al informe notificando la anomalía detectada durante el uso del programa.

1.3 Estructura del trabajo

La organización de este documento está compuesta por los siguientes capítulos:

- **Pruebas de aceptación automatizadas**: está formado por una introducción al mundo de las pruebas en el desarrollo de *software*, a las pruebas automatizadas y un glosario con sus términos más comunes que se hacen referencia a lo largo del trabajo.
- **Infraestructura de pruebas de automatización automatizadas actual**: lista en profundidad las fases por las que transcurren las pruebas automatizadas codificadas en el proyecto y detalla todos los sistemas *software* y tecnologías que posee la empresa antes de su modificación durante este trabajo, los cuales son encargados de ejecutar en local pruebas automatizadas y las máquinas virtuales disponibles para albergar ejecuciones remotas de pruebas.
- **Estado del arte**: muestra herramientas de ejecuciones de pruebas similares presentes en el mercado, funciones más características y una conclusión sobre ellas, así como la alternativa seleccionada para el desarrollo del trabajo.
- **Tecnologías utilizadas**: desglosa los programas destinados a la grabación de vídeo, *frameworks* y grupos tecnológicos que se harán uso en el transcurso del trabajo dividido en subsistemas y funcionalidades, así como sus características y las razones de su implementación.
- **Desarrollo de la solución:** expone el transcurso las modificaciones en los requisitos de las herramientas y proyectos de pruebas automatizadas de la empresa. Se muestra aplicando Ingeniería de Software y bajo el uso de metodologías ágiles su evolución y como son incluidas las nuevas funcionalidades de grabación de vídeo junto a la ejecución de pruebas.
- **Conclusiones y trabajo futuro**: valoraciones al finalizar el proyecto, retrospectiva obtenida durante el progreso de la tesis y consideraciones e ideas de trabajo futuro obtenidas tareas alrededor de los nuevos requisitos aplicados y las herramientas de la empresa mejoradas.
- **Referencias**: registro de las fuentes de información consultadas para la redacción de este trabajo.

2. Pruebas de aceptación automatizadas

La actividad de pruebas sobre un proyecto *software* es clave para la validación del producto en desarrollo. Esta etapa es vital para determinar si un programa funciona conforme a los requisitos solicitados por el cliente y, en caso de encontrarse algún fallo, corregirlo para que funcione conforme a lo estipulado en las necesidades.

Esta labor puede enfocarse a muchos ámbitos del programa y, en consecuencia, aplicar un tipo u otro de pruebas. Al ser un proceso costoso de realizar de forma manual, en los últimos años ha surgido especial interés en su automatización.

A continuación, desglosaremos los conceptos más importantes en las pruebas de *software*, sus tipos y una exposición de las pruebas automatizadas.

2.1 Pruebas en el desarrollo de un producto *software*

Las pruebas de *software* se tratan de un control de calidad en el desarrollo de una tecnología, herramienta o servicio, en la cual verifica de forma objetiva e independiente la correcta funcionalidad en función de lo que esperan los interesados o *stakeholders* del producto.

Existen varios tipos de prueba en la industria, la cual es determinada en función de la dimensión de la aplicación probada y en qué contexto es ejecutado. Estas pueden abarcar desde unas pocas líneas de código hasta el producto desarrollado en su totalidad. Las más comunes en el desarrollo de software son [22]:

- **Pruebas unitarias** (*unit testing*): nivel más bajo de prueba, donde verifica el funcionamiento de pequeños fragmentos de código o métodos de forma aislada.
- **Pruebas de integración** (*integration testing*): comprueba la ejecución de distintas piezas de código escritos en distintas áreas del programa o subsistema.
- **Pruebas de sistema** (*system testing*): se encarga de poner a prueba todo el sistema *software* en su conjunto una vez compilado con todos los componentes y subsistemas del programa.
- **Pruebas de aceptación** (*acceptance testing*): desde el entorno del *stakeholder*, se valida si el producto final desarrollado satisface las necesidades y sus funcionalidades son las esperadas de acuerdo con sus requisitos.

De forma adicional, existen las llamadas **pruebas de regresión**. Estas pruebas pueden formar parte de cualquiera de las categorías mostradas anteriormente. Su función consiste en ser aplicadas de nuevo ante cualquier cambio producido en el código y en sus funcionalidades una vez ya pasaron estas pruebas con éxito anteriormente. De esta forma, se asegura que el programa testeado siga cumpliendo con los requisitos a lo largo de sus distintas versiones.

[12] Una prueba puede no solo comprobar si una acción en el programa funciona, sino también medir propiedades que sean clave para que cumpla con estándares de calidad. Entre las pruebas de *software* encontramos dos tipos:

- **Pruebas funcionales:** destinado a la ejecución del programa y los servicios que ofrece. Desde el punto de vista del usuario, pone a prueba el correcto funcionamiento del *software*, introduciendo unos valores de entrada y esperando unos de salida, ofreciendo retroalimentación de lo ocurrido y sin tener en cuenta el algoritmo interno de la aplicación. También son conocidas como **pruebas de caja negra**.
- **Pruebas no funcionales:** asignadas a la medición y verificación de restricciones de diseño. Por ejemplo: seguridad, mantenibilidad, rendimiento, tiempos de ejecución, etc.

Entre las variedades de pruebas mostradas en este apartado, en el departamento de desarrollo de nuestra empresa se enfoca principalmente en pruebas de aceptación funcionales. Los requisitos definidos en la fase de análisis del ERP se sustentan a partir de estas pruebas definidas antes de su programación, que sirven como herramienta de ejecución para comprobar que la aplicación cumple con sus requerimientos una vez desarrollada.

Para la aplicación y automatización de estas pruebas de aceptación, se desgranan en una o más pruebas de sistema. Básicamente son instanciaciones de estas pruebas de aceptación aplicadas en el ERP una vez desarrollado por el propio equipo de desarrollo. Una vez pasen todas las pruebas de sistema de una prueba de aceptación, se da por validado el requisito establecido que representa la prueba diseñada durante la fase de análisis.

2.2 Pruebas automatizadas

Las pruebas automatizadas tienen el mismo objetivo que una prueba manual: verificar el correcto funcionamiento del programa de acuerdo con lo esperado. La principal motivación que ha llevado a la automatización del proceso es la creciente exigencia y demanda en el sector tecnológico, donde existen programas con muchísimas líneas de código y funcionalidades donde un error humano o falta de disciplina puede pasar por alto fallos de la aplicación. Debido a esto y con la presencia de aplicaciones muy críticas en el sector, se requiere del uso de metodologías más formales y automatizadas para llevar a cabo este proceso.

A diferencia de las pruebas manuales, son ejecutadas por máquinas cuya infraestructura está programada para simular las acciones del usuario sobre el programa. En comparación con la aplicación de pruebas manuales, las pruebas automatizadas ofrecen una serie de características y ventajas en el ámbito de *testing* en una aplicación (asumiendo que las pruebas automatizadas son de calidad), de las cuales destacamos las siguientes [25]:

- **Rapidez**: la ejecución del algoritmo y las acciones realizadas en la aplicación requieren de menor tiempo que el de un usuario aplicando pruebas de *software* manual.

- **Fiabilidad**: devuelven un resultado más objetivo de la prueba, tanto si ha sido positiva como negativa. En el caso de no haber pasado la prueba, devuelve en el resultado de la ejecución aquello que se esperaba de la aplicación.
- **Reusabilidad**: existen el caso en nuestra empresa en los que varias pruebas de sistemas varían en pocos pasos o cambian algunos valores muy concretos. La programación en diversos lenguajes permite que el algoritmo de la prueba pueda heredar código, favoreciendo la mantenibilidad del proyecto.
- **Mayor precisión**: la prueba hace en todo momento lo que le indicamos en el algoritmo, evitando interpretaciones o alteraciones en los pasos de la ejecución de esta.
- **Ahorro en tiempo y costes:** como observamos en la Figura 2, conforme crecen las funcionalidades y las pruebas necesarias para abordar las validaciones del usuario, el coste de realizar este proceso es cada vez mayor en un proceso manual. En cambio, el impacto en los costes de añadir nuevos casos de prueba en un proceso automatizado será mucho menor en comparación. Además, al ser ejecutadas por máquinas y no por usuarios, el requerimiento de personal humano se limita al mantenimiento de las pruebas (reparación, revisión de resultados…) y a la automatización de las nuevas.
- **Ausencia de fallos en la prueba por la acción humana:** al ser programadas las pruebas con mayor formalidad, asegura la ejecución de todos los pasos e introducción de valores de entrada establecidos.
- **Posibilidad de ejecutar varias pruebas al momento:** en la mayoría de las infraestructuras de pruebas automatizadas, permite la ejecución de más de una prueba de forma concurrente, ahorrando tiempo en comparación con una ejecución secuencial.

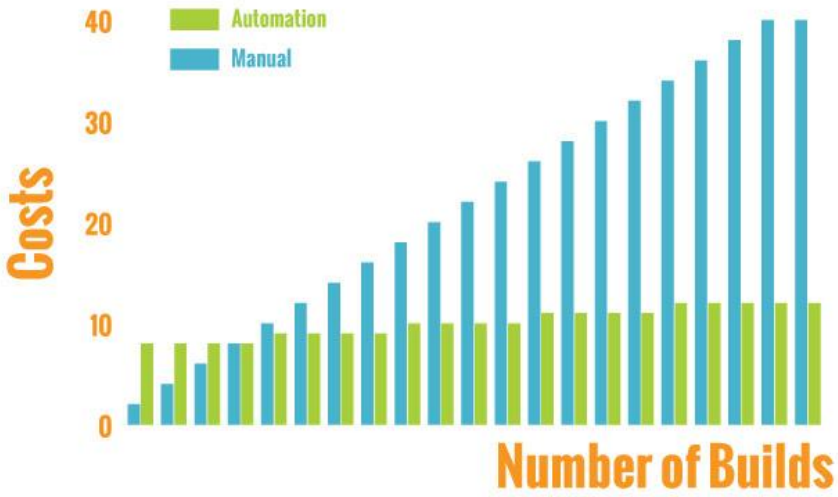


Figura 2. Comparación de costes en un proceso manual y automático de pruebas en función del tamaño de la aplicación. [25]

Si observamos la Figura 3, podemos ver de forma gráfica las diferencias de recursos requeridos entre aplicar *testing* manual y automatizado. Para establecer una infraestructura de *testing* automatizado de calidad requiere de mayor esfuerzo de implementación y configuración de herramientas. El personal encargado de su

mantenimiento y automatización requieren además de una mayor formación para realizar sus funciones. Sin embargo, el tiempo de ejecución de las pruebas y el número de personal requerido son mucho menores en *testing* automatizado.

A corto plazo es necesario un gran esfuerzo para la construcción de la infraestructura y el aprendizaje en el uso de las herramientas. En cambio, a largo plazo esa inversión de recursos se retribuye en una mayor optimización en el proceso y en costes para lanzar las pruebas en entornos automatizados.

Figura 3. Requerimientos en un proceso manual de pruebas de software en comparación con uno automático. [25]

Conceptos

Antes de avanzar con el trabajo, donde abordaremos en más detalle todo el proyecto y la situación actual de la infraestructura de *testing* automatizado de nuestra empresa, expondremos una serie de términos frecuentes en referencia al contexto de pruebas automatizadas.

[4] Los vocablos más comunes que haremos uso durante todo el trabajo son los siguientes:

- **Script de prueba**: instrucciones y datos escritos en un lenguaje formal o semiformal de programación que conforma el algoritmo de una prueba automatizada. Se almacena cada prueba en un archivo independiente y son ejecutados por las herramientas y tecnologías de automatización para llevar a cabo el lanzamiento de la prueba sobre el *software* testeado.
- **Batería o *suite* de pruebas**: colección de pruebas automatizadas reunidas para el cumplimiento de objetivos concretos de la empresa. Un ejemplo puede ser el lanzamiento de *suites* reducidas para la detección temprana de fallos sobre un módulo en específico.
- **Automatización**: diseño y programación de nuevas pruebas automatizadas. Genera una prueba automatizada a partir de una prueba manual escrita en los requisitos del programa.
- **Resultado de la prueba**: también denominado *log* de prueba. Hace referencia al registro generado en un archivo HTML por la ejecución de una prueba automatizada. Esta muestra un informe con todas las comprobaciones de datos, objetos en el formulario y acciones que ha realizado. Se incluyen los datos extraídos de la aplicación, los esperados y, en el caso de detectar una

excepción o error durante el transcurso de la prueba, la traza de la ejecución del código donde se ha producido.

3. Infraestructura de pruebas de aceptación automatizadas actual

La mayoría de las infraestructuras y proyectos de pruebas automatizadas en el mercado tienen similitudes y características en común como las mostradas en las páginas previas. Sin embargo, cada tecnología y caso contiene funcionalidades y detalles que marcan la diferencia entre ellas.

En nuestra empresa existe una clara jerarquía de proyectos y elementos relacionados que conforman toda la infraestructura de pruebas automatizadas actual, desde el elemento más mínimo, como son las pruebas de sistemas, hasta las herramientas que permiten lanzar de forma remota pruebas automatizadas.

En este capítulo, mostraremos en detalle todos los componentes y subsistemas que existen en la empresa, así como las diversas funcionalidades con las que cuentan, herramientas utilizadas y el proceso de revisión de pruebas, así como un análisis a los inconvenientes que se presentan en toda la infraestructura de pruebas automatizadas.

3.1 Pruebas automatizadas actuales

La unidad mínima de pruebas con la que trabajamos en nuestra empresa son pruebas de sistemas, las cuales en su conjunto forman cada una de las pruebas de aceptación de usuario que validan el correcto funcionamiento del ERP.

Previamente, antes de realizarse las automatizaciones de nuevas pruebas, estas son diseñadas [1] por los analistas programadores de la empresa mediante el lenguaje informal. Un ejemplo de estas pruebas se muestra en la Figura 4. Están ideadas para que dichas pruebas sean ejecutadas por los testers manuales de la aplicación y se especifiquen, paso por paso, los datos a introducir, configuraciones del programa y los resultados esperados al finalizar su ejecución.

[1] ***Diseñar**: instanciar las pruebas con datos en la aplicación.*

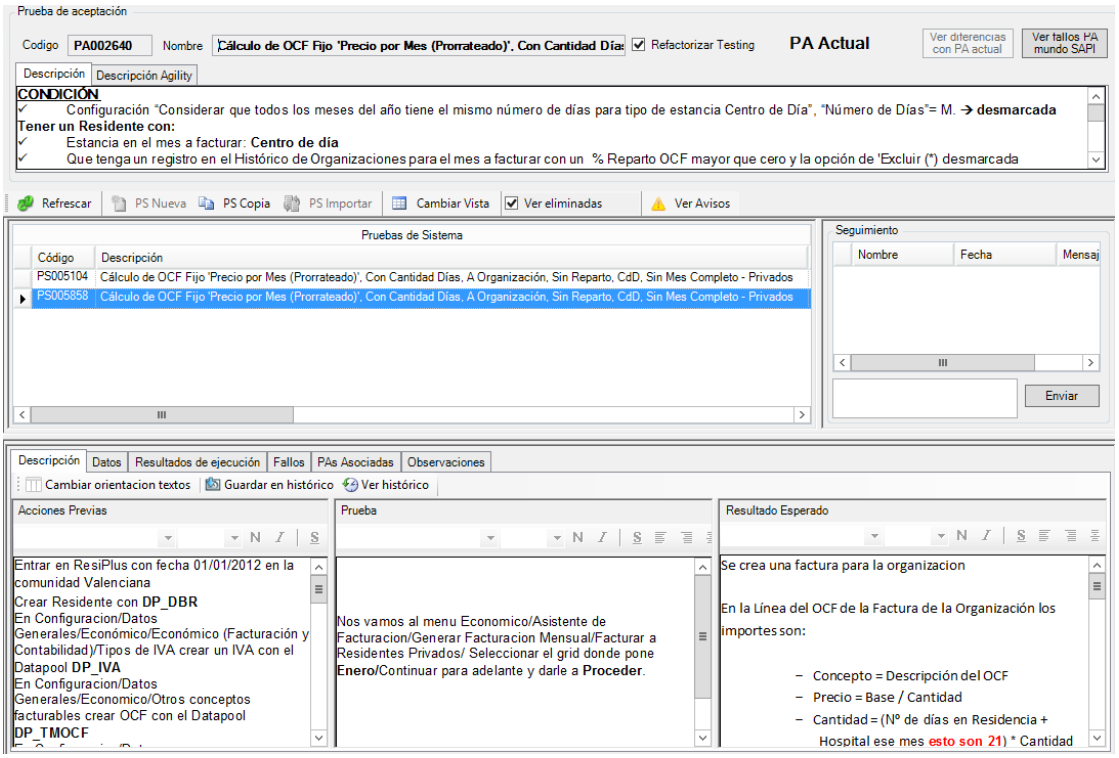


Figura 4. Prueba de aceptación manual del ERP diseñada en lenguaje natural.

Cuando realizamos el proceso de automatización, para cada una de estas pruebas de sistema se les asigna un script de prueba, el cual determinará las acciones a realizar por cada prueba por parte de la máquina que la ejecuta y sus comprobaciones en los formularios de la aplicación testeada.

Los componentes tecnológicos y herramientas que contienen nuestras pruebas están formados por:

- **IBM *Rational Functional Tester*[2] (RFT):** constituye el núcleo principal de las pruebas automatizadas y el puente operativo entre las pruebas y la aplicación testeada. Es un producto *software* diseñado por la compañía IBM que permite lanzar scripts de pruebas sobre aplicaciones con interfaces de usuario de forma automática. Simula las acciones del usuario mediante el reconocimiento y mapeado de formas, el cual por cada formulario reconoce y asigna un identificador a cada uno de los elementos que lo constituye (botones, campos de texto, listados, etc). En la Figura 5 se muestra como una prueba con ayuda de esta herramienta realiza acciones sobre la aplicación aplicando reconocimiento de formas.

[2] ***IBM Rational Functional Tester*** *– Página oficial: https://www.ibm.com/products/rational-functional-tester*

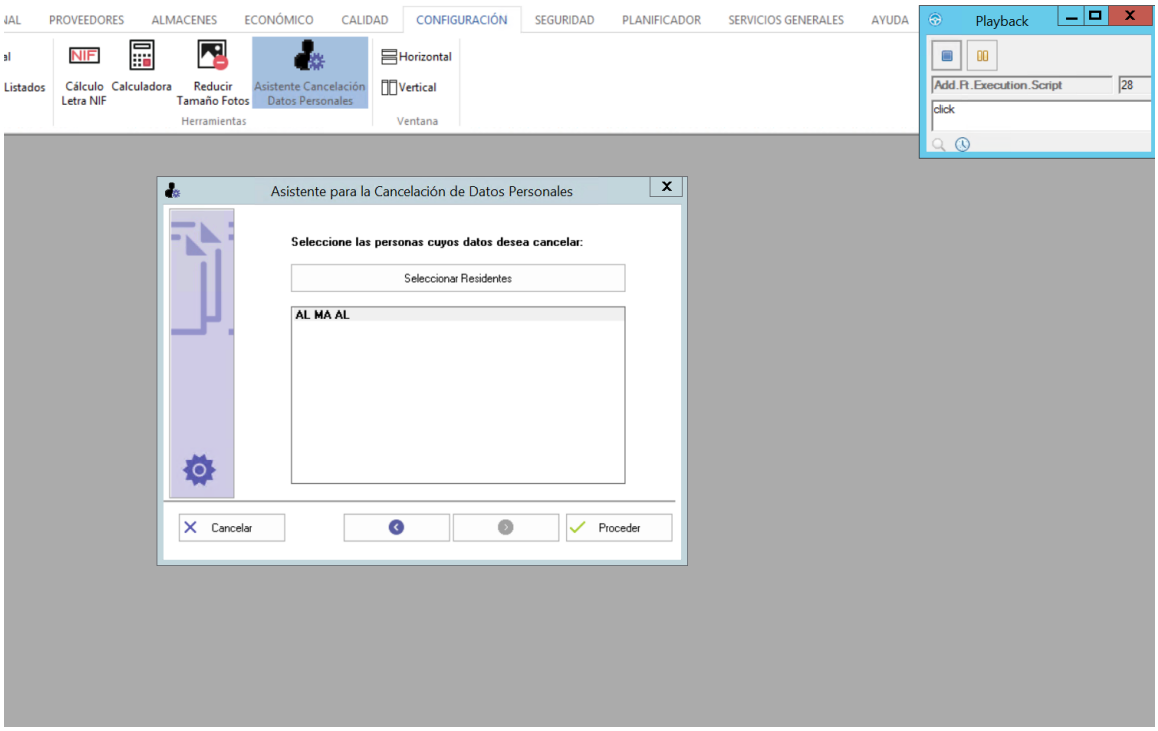


Figura 5. Ejecución de una prueba automatizada haciendo uso de IBM Rational Functional Tester.

- **Framework .NET:** es la tecnología encargada de ejecutar los procesos requeridos para el lanzamiento de las pruebas y las instrucciones de los scripts en la máquina. Complementa las funciones de *Rational Functional Tester* manejando las excepciones e introduciendo los datos necesarios en la base de datos de la aplicación.
- **Lenguajes Visual Basic/C#:** codifica las instrucciones y el uso de las diversas bibliotecas que forman parte de la infraestructura de pruebas automatizadas. Los componentes y pruebas más antiguas fueron programadas en *Visual Basic*. En cambio, las pruebas más recientes se han escrito en el lenguaje C#. Este cambio de lenguaje ha permitido conseguir mejoras en el mantenimiento y la programación de pruebas al contar con una sintaxis más sencilla de comprender en comparación con su predecesora. Ambos lenguajes comparten la librería .NET, permitiendo que el código antiguo de las pruebas en *Visual Basic* pueda funcionar con código en C# sin modificaciones.

A la hora de ejecutar las pruebas, estas transitan por varias fases desde que comienzan hasta que termina la prueba. Dichas etapas se muestran en la Figura 6. De forma detallada, las fases por las que recorren las pruebas automatizadas son las siguientes:

- **Inicio de *Rational Functional Testing*:** lanza todos los procesos que ejecutarán el código de la prueba automatizada, así como iniciar la aplicación testeada.
- ***Prepare Data* o preparación de datos:** al comienzo de la prueba, de forma opcional, introduce los datos previos y establece las opciones en la base de datos del programa para configurar los parámetros y funcionalidades esperadas de este. La forma de insertar la información y ajustes requeridos puede realizarse mediante la ejecución de un script SQL que introduzca los datos directamente

en las tablas del programa, o, en cambio, realizarlos de forma manual en los formularios de la aplicación desde la prueba automatizada.

- **Ejecución de la prueba:** fase por la que realiza sobre la aplicación testeada las acciones programadas en el script de la prueba. Durante esta etapa cabe la posibilidad que surjan fallos de aplicación o que *RFT* no sea capaz de ejecutar acciones del algoritmo. En ese caso, la prueba finalizaría de forma inmediata notificando de dicho fallo en el resultado de la prueba.
- **Comprobación de los resultados obtenidos:** al finalizar el script de la prueba, si la ejecución de la aplicación ha sido correcta en todo momento, de forma opcional puede comparar aquello que se ha guardado o generado por la aplicación y lo esperado por la prueba. Los registros generados pueden contener información confidencial de la aplicación que no se debería publicar, lo cuales son importantes de comprobar que al realizar los pasos ciertos datos no sean publicados cuando no deberían.
- **Final de la prueba:** sea cual sea el resultado final de la prueba, se finalizan de forma segura los procesos lanzados y genera correctamente el archivo con el informe del resultado de la prueba, como se observa en la Figura 7. Esto asegura que, al terminar una prueba, esta no tenga procesos en segundo plano ni queden aplicaciones abiertas, consumiendo recursos de forma innecesaria e impidiendo la ejecución de nuevas pruebas en un futuro.

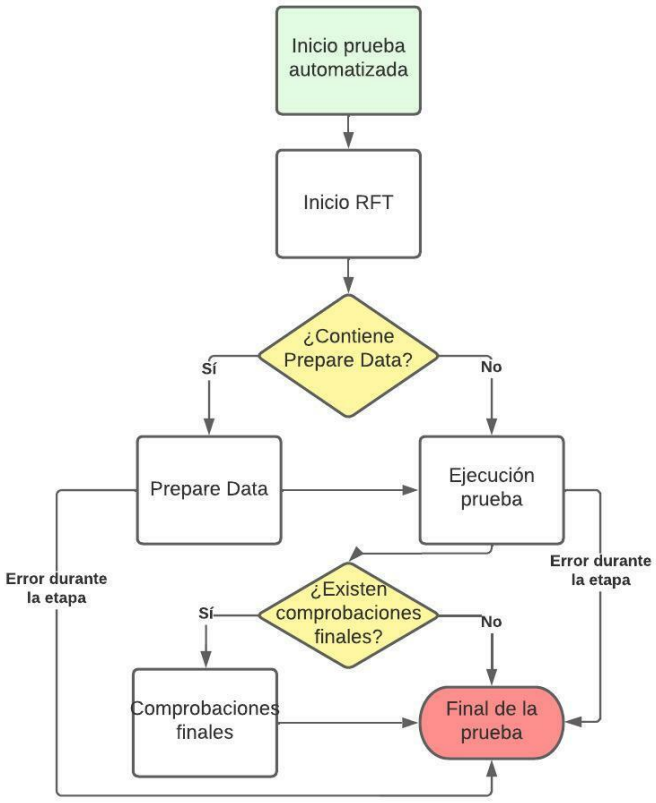


Figura 6. Diagrama de flujo de las fases de una prueba automatizada.

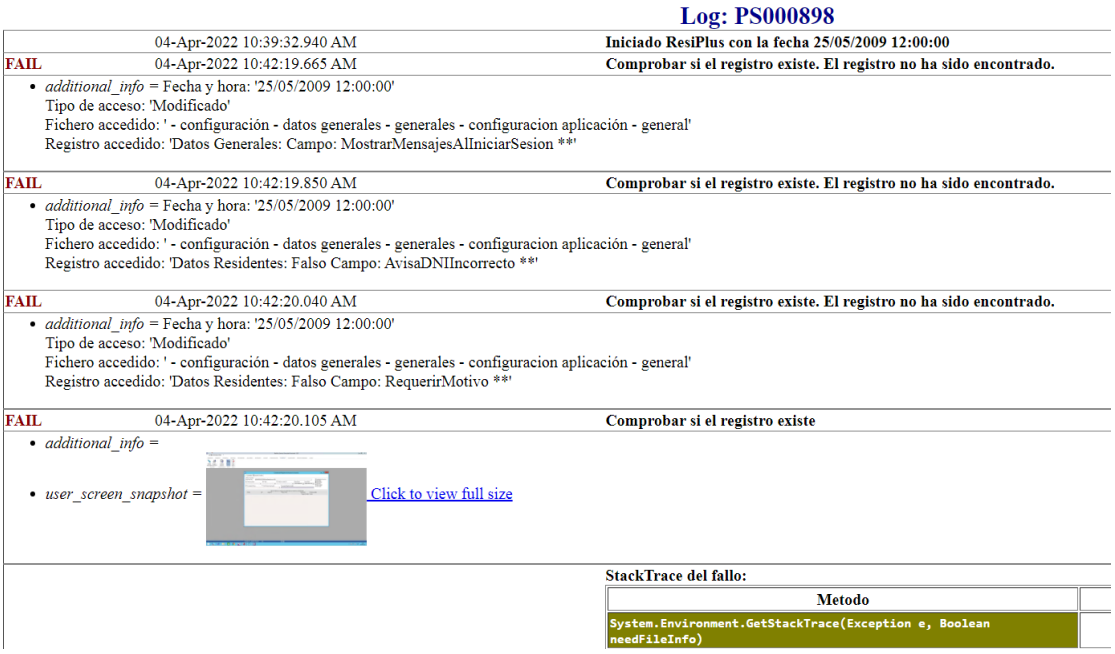


Figura 7. Informe generado por la ejecución de una prueba automatizada.

3.2 Proyecto de pruebas automatizadas. *AddFt*

Todas las pruebas automatizadas están programadas en el proyecto ***AddFt***, una solución del IDE *Visual Studio* que cuenta con todas las librerías y tecnologías necesarias para que sean puedan ser lanzadas en el entorno. El desarrollo y mantenimiento del proyecto está apoyado por la plataforma de colaboración ***Visual Studio Team Foundation Server***, [16] el cual ofrece un control de versiones centralizado, donde los cambios son almacenados en un servidor y son aplicados en las soluciones locales de cada uno de los mantenedores de las pruebas. Todas las modificaciones son registradas, tanto de archivos de código como documentos (archivos de mapeado, *scripts* de bases de datos, etc) y sus versiones previas pueden ser consultadas y comparadas con las actuales, además de ofrecer la posibilidad de deshacer cualquier cambio subido en el servidor.

Dentro del proyecto, además de albergar todos los componentes mencionados en la subsección anterior, ofrece un método para lanzar las pruebas en el entorno donde se localiza el proyecto, como se muestra un ejemplo en la Figura 8. Existen varios argumentos para personalizar el lanzamiento de una prueba automatizada, entre los cuales los más comunes se encuentran:

- **Nombre de la prueba**: se introduce una cadena de texto la cual hace referencia al identificador de la prueba y el proyecto busca la instancia de la prueba automatizada enlazada a dicho código. En caso de no encontrar ninguna prueba con ese identificador, lanzaría una excepción y mostraría del error en el resultado de la prueba.
- **Modo de ejecución**: existen una serie de opciones preestablecidas para personalizar el perfil de lanzamiento de la prueba. Entre dichas configuraciones de ejecución, podemos encontrar un lanzamiento completo, de regresión (la etapa de Prepare Data se omite, lo cual agiliza la ejecución de la prueba si esta

es relanzada de forma seguida) o modo grabación, el cual genera un script SQL con todos los datos contenidos en la aplicación al finalizar la prueba.

```
Public Class Program
    Implements IAction
    – references
    Public Shared Sub Main(ByVal args As String())
        Launcher.LaunchRft(New Program())
    End Sub

    – references
    Public Sub Run() Implements IAction.Run
        Manager.RunManagers("PS006275", Manager.RunMode.All)
    End Sub
End Class
```

Figura 8. Método en AddFt para el lanzamiento de una prueba automatizada con los parámetros de identificador de prueba y modo de ejecución.

3.3 Lanzador de pruebas automatizadas remotas ATUN

En la subsección anterior, hacemos referencia a un proyecto cuyo contexto es el de lanzar pruebas en un entorno local. Resulta algo limitado ya que, para lanzar una colección de pruebas muy extensa, resulta un proceso largo, laborioso y costoso al requerir de forma manual editar el método encargado de lanzarlo en el proyecto y no existir algún sistema automatizado para realizar dicha tarea.

Para solucionar este problema se ha desarrollado **ATUN**, un programa que permite lanzar de forma remota y bajo baterías las pruebas automatizadas en máquinas virtuales externas. Esta herramienta trabaja de forma conjunta con el proyecto *AddFt*, el cual ofrece todas sus funcionalidades y el sistema de ATUN las complementa, notificando mediante colas asíncronas a las máquinas virtuales para que, desde sus proyectos de *AddFt* instalados, ejecuten las pruebas solicitadas y se obtengan los resultados de su ejecución.

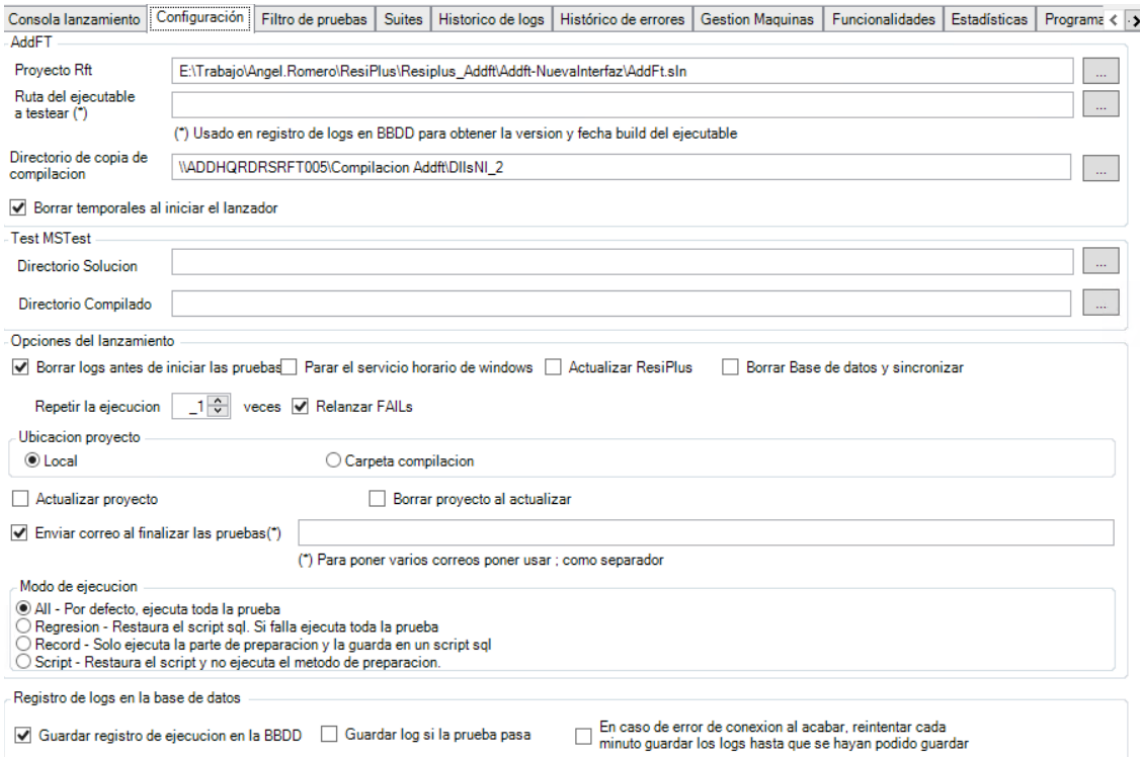


Figura 9. Formulario de configuración de ATUN para la ejecución de baterías.

ATUN dispone de un formulario donde podemos seleccionar las pruebas que queremos ejecutar realizando una búsqueda manual de sus identificadores (si son varias pruebas, se separa cada identificador con un ";") y el resultado de su búsqueda encolarlo en las máquinas virtuales, como se muestra en la Figura 10. Además, podemos establecer suites de pruebas de forma automática en función de los resultados de su última ejecución en nuestro sistema. Por ejemplo: la *suite* de pruebas de nivel 0 se compone únicamente de las pruebas cuyo resultado haya sido exitoso en su última ejecución.

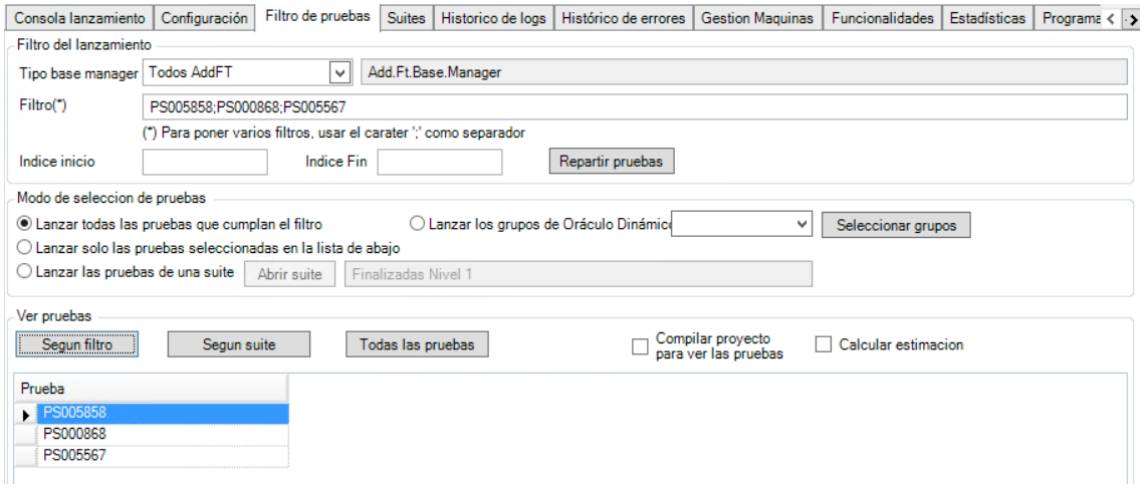


Figura 10. Formulario de búsqueda de pruebas automatizadas en ATUN.

Podemos revisar en tiempo real el estado actual de la ejecución de una batería de pruebas y el historial de baterías ejecutadas a lo largo del tiempo y sus resultados. Además, si una batería contiene pruebas que han fallado, al momento de finalizar la batería se vuelven a ejecutar de nuevo, generándose otra batería adicional con una

descripción haciendo referencia a la que formaba parte. Al seleccionar una prueba del listado de las baterías, podemos desplegar el informe del resultado de la prueba y obtener más datos de su ejecución.

Cada fila dentro de la batería corresponde a una prueba automatizada y está marcada por colores, como se muestra en la Figura 11. Dicho color hace referencia a su estado actualizado, los cuales pueden ser:

- **Verde**: indica que la prueba se ha ejecutado al completo con éxito.
- **Amarillo**: la prueba ha finalizado y muestra un aviso respecto a alguna mejora en el código u objeto detectado en el mapeado.
- **Rojo**: ha existido un error durante la ejecución de la prueba y no ha finalizado de forma correcta.
- **Blanco**: la prueba está siendo ejecutada en una máquina virtual.
- **Gris**: la prueba se encoló con éxito en la batería, pero no está siendo ejecutada en ninguna máquina virtual.

IdBateria	Filtro seleccionado	Observaciones	Sel. Maq.	Programa	Fecha inicio	Fecha fin	EnEjecucion
17418	Lista de pruebas			Todos AddFT	07/05/2022 16:3...	07/05/2022 22:31:11	☑

Codigo	CodigoPA	PARefactorizar	Último OK	ProgramaManager	Versión	NodoDescripcion	Resultado
PS000094	PA000495	☐	07/05/2022 16:4...	ResiPlus	4.2.003	N49343 - Bloqueos	WARN
PS000128	PA000761	☑	07/05/2022 16:5...	ResiPlus	4.0.001	N00421 - Historia...	PASS
PS000203	PA000477	☐	10/05/2022 17:0...	ResiPlus	4.2.003	N00420 - Histórico...	FAIL
PS000207	PA000477	☐	16/05/2022 11:2...	ResiPlus	4.2.003	N00420 - Histórico...	FAIL
PS000209	PA000477	☐	10/05/2022 16:4...	ResiPlus	4.2.003	N00420 - Histórico...	FAIL
PS000210	PA000477	☐	07/05/2022 18:2...	ResiPlus	4.2.003	N00420 - Histórico...	PASS
PS000250	PA000143	☐	07/05/2022 16:5...	ResiPlus	4.2.003	N49721 - Pestaña...	WARN
PS000257	PA000143	☐	07/05/2022 16:5...	ResiPlus	4.2.003	N49721 - Pestaña...	WARN
PS000258	PA000143	☐	07/05/2022 16:5...	ResiPlus	4.2.003	N49721 - Pestaña...	WARN
PS000260	PA001041	☐	07/05/2022 19:3...	ResiPlus	4.2.003	N00920 - Funciona...	PASS
PS000266	PA001041	☐	07/05/2022 19:3...	ResiPlus	4.2.003	N00920 - Funciona...	PASS
PS000272	PA000143	☐	07/05/2022 16:5...	ResiPlus	4.2.003	N49721 - Pestaña...	WARN
PS000273	PA000143	☐	07/05/2022 17:1...	ResiPlus	4.0.001	N49721 - Pestaña...	WARN
PS000275	PA000091	☑	07/05/2022 16:4...	ResiPlus	4.0.001	N00108 - Comprob...	WARN
PS000289	PA000091	☑	07/05/2022 16:4...	ResiPlus	4.2.003	N00108 - Comprob...	WARN
PS000366	PA000156	☑	07/05/2022 20:1...	ResiPlus	4.2.003	N49376 - Caracterí...	WARN
PS000389	PA000086	☑	07/05/2022 17:1...	ResiPlus	4.2.003	N00108 - Comprob...	WARN

Figura 11. Listado de pruebas de una batería ejecutadas en remoto en ATUN.

Además, cuando requerimos ejecutar baterías con varias pruebas, son repartidas por las distintas máquinas seleccionadas antes de lanzar la batería, como observamos en el formulario de la Figura 12. La ventaja que esto nos ofrece en el proceso de lanzamiento masivo de pruebas es la reducción entre **N** máquinas virtuales seleccionadas el tiempo **T** del lanzamiento de cada una de las **P** pruebas automatizadas de la batería.

$$Tiempo\ total\ ejecución\ batería = \frac{\sum_{i=1}^{p} T_i}{N}$$

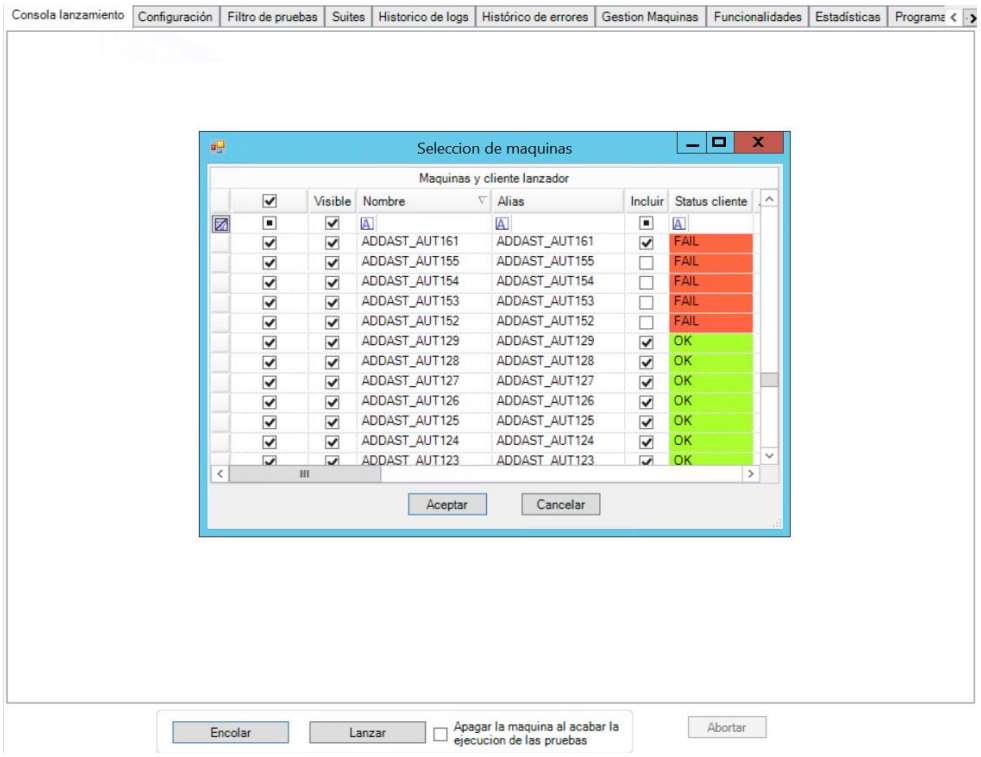


Figura 12. Formulario de selección de máquinas virtuales previo a la ejecución de una batería en ATUN.

Los escritorios virtuales preparados para la ejecución de pruebas automatizadas cuentan con los lanzadores de pruebas enlazados a la infraestructura de ATUN y el proyecto *AddFt* actualizado al momento. Al iniciar la máquina, se vincula con ATUN y se registra en la lista de máquinas preparadas para poner en lanzamiento pruebas de forma remota. Posteriormente, la máquina se encuentra a la espera de la cola para la ejecución de alguna prueba automatizada cuando sea incluida, si su máquina fue seleccionada en los parámetros de ejecución de la batería.

En nuestra empresa, contamos con un largo sistema de máquinas virtuales, como vemos en la Figura 13. En función de la demanda de las máquinas, para ahorrar recursos y no tener máquinas en marcha sin utilizarse, se activan o apagan, siendo el tiempo en el que más máquinas hay en marcha en el *release* de una nueva versión del ERP y se deben lanzar *suites* con una alta carga de pruebas en remoto.

VMs (31)

Name	Stat...	Virt...	Availab...	Host	Cloud	Job Status	Ow...	Use...	CPU Av...
ADDAST_AUT119	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT121	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT122	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT123	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT124	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT125	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT126	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT127	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT128	Running	Running			Self-Ser...	Completed			1 %
ADDAST_AUT129	Running	Running			Self-Ser...	Completed			0 %
ADDAST_AUT161	Stopped	Stopped			Self-Ser...	Completed			0 %
ADDAST_AUT251	Stopped	Stopped			Self-Ser...	Completed			0 %
ADDAST_O001	Running	Running			Self-Ser...	Completed			0 %
ADDAST_O365	Stopped	Stopped			Self-Ser...	Completed			0 %

Figura 13. Listado de máquinas virtuales preparadas para la ejecución de pruebas automatizadas en remoto.

3.4 Proceso de revisión de pruebas

Antes de publicar la nueva versión de la aplicación a testear, se inicia un proceso de lanzamiento masivo de todas las pruebas del proyecto para comprobar que la aplicación sigue cumpliendo con sus requisitos y encontrar fallos no detectados en producción.

El objetivo principal de esta labor es notificar fallos de la aplicación de forma temprana al equipo de desarrollo para corregirlos en la versión antes de ser enviada a los clientes y reparar las pruebas que fallaron por cambios en formularios, funciones o requisitos del programa. Este proceso se repite con el lanzamiento de cada nueva versión del ERP.

Cuando una nueva versión cumple los controles de *testing* manual, se notifica al departamento de *testing* automatizado para que lancen la suite completa de pruebas desde el ATUN sobre esta nueva versión. Una vez finalizada su ejecución, comienza la tarea de revisión de los resultados de las pruebas.

El proceso comienza inspeccionando las pruebas cuyo resultado de fallo (las pruebas marcadas en rojo en la batería de ATUN). Una vez revisadas las pruebas y habiendo recolectado todos los fallos de aplicación encontrados, se procede al envío de un informe al equipo de desarrollo donde se aportan datos sobre la ejecución de la batería, destacando los fallos y dándolos de alta en el sistema de incidencias de la empresa junto a los detalles para su corrección antes ser publicada la versión a los usuarios.

Defectos y dificultades durante el proceso

Durante la labor de revisión de resultados de pruebas falladas, surgen varias trabas que entorpecen la tarea e incluso imposibilita la comprensión de los errores mostrados en los informes. Puede darse el caso donde una prueba falla en los escritorios remotos por diversas razones (ficheros no actualizados, fallos momentáneos debidos al entorno en el que se encuentran, etc) y en el escritorio local las pruebas pasan o dan fallos

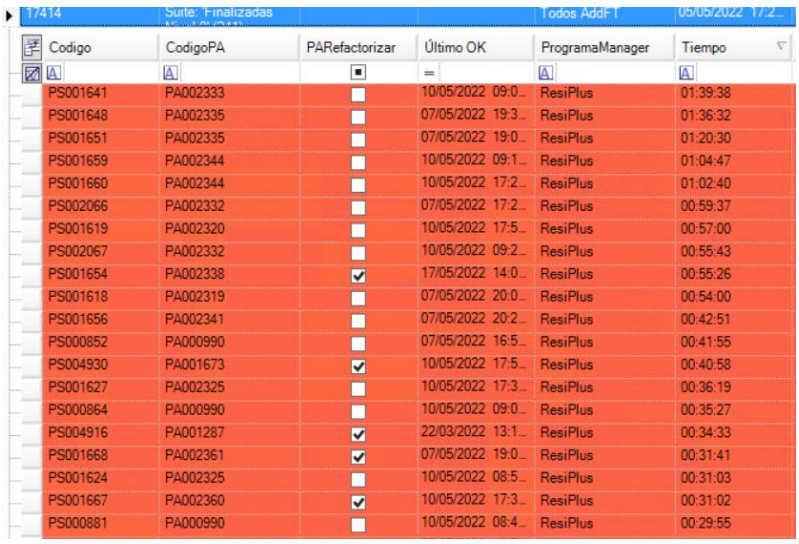
distintos. En ambos casos, requiere ejecutar la prueba de nuevo en local por parte del mantenedor, requiriendo de su atención completa durante todo su lanzamiento.

Esto resulta en una situación no deseada, debido a los siguientes inconvenientes:

- **Imposibilidad a la hora de retroceder pruebas**: si es cierto que en el proyecto *AddFt* desde *Visual Studio* se pueden añadir puntos de ruptura en el código de las pruebas y detenerla momentáneamente, si en el momento del fallo no se ha comprendido su causa y se requiere de su retroceso, la naturaleza de la aplicación y las tecnologías de las pruebas automatizadas no lo permiten. Por lo tanto, requeriría de otro lanzamiento de la prueba desde el comienzo, requiriendo de nuevo de la plena atención del mantenedor e impidiéndole realizar otras tareas del departamento.
- **Costes temporales altos**: en ocasiones, las pruebas realizan muchas acciones en la aplicación, llegando a tardar mucho tiempo en completarse. Esto, ligado al punto anterior, el coste de realizar la tarea de comprensión de fallos de pruebas es muy elevado.
- **Resultados de las pruebas con poca información**: se da el caso en el que el *log* ofrece una información poco precisa, y la causa del fallo no es posible de encontrar con una lectura al informe y se requiere lanzar la prueba en local.

Combinando todos los puntos anteriores, da una situación en la que el mantenedor de las pruebas requiere relanzar las pruebas de una batería extensa de forma reiterada en el entorno local. No es un proceso idóneo, ya que hablamos de mucho tiempo en el que el usuario debe dedicarle a visualizar la prueba sin posibilidad de retrocederla o restaurar el estado de la aplicación en un punto exacto, requiriendo detener la ejecución de la prueba y lanzarla de nuevo en local y remoto.

Para comprender la envergadura de este problema, se ha consultado el último lanzamiento de la *suite* o de pruebas y se han obtenido sus tiempos de ejecución, ordenando las pruebas de mayor a menor tiempo de ejecución.



Codigo	CodigoPA	PARefactorizar	Último OK	ProgramaManager	Tiempo
PS001641	PA002333	☐	10/05/2022 09:0...	ResiPlus	01:39:38
PS001648	PA002335	☐	07/05/2022 19:3...	ResiPlus	01:36:32
PS001651	PA002335	☐	07/05/2022 19:0...	ResiPlus	01:20:30
PS001659	PA002344	☐	10/05/2022 09:1...	ResiPlus	01:04:47
PS001660	PA002344	☐	10/05/2022 17:2...	ResiPlus	01:02:40
PS002066	PA002332	☐	07/05/2022 17:2...	ResiPlus	00:59:37
PS001619	PA002320	☐	10/05/2022 17:5...	ResiPlus	00:57:00
PS002067	PA002332	☐	10/05/2022 09:2...	ResiPlus	00:55:43
PS001654	PA002338	☑	17/05/2022 14:0...	ResiPlus	00:55:26
PS001618	PA002319	☐	07/05/2022 20:0...	ResiPlus	00:54:00
PS001656	PA002341	☐	07/05/2022 20:2...	ResiPlus	00:42:51
PS000852	PA000990	☐	07/05/2022 16:5...	ResiPlus	00:41:55
PS004930	PA001673	☑	10/05/2022 17:5...	ResiPlus	00:40:58
PS001627	PA002325	☐	10/05/2022 17:3...	ResiPlus	00:36:19
PS000864	PA000990	☐	10/05/2022 09:0...	ResiPlus	00:35:27
PS004916	PA001287	☑	22/03/2022 13:1...	ResiPlus	00:34:33
PS001668	PA002361	☑	07/05/2022 19:0...	ResiPlus	00:31:41
PS001624	PA002325	☐	10/05/2022 08:5...	ResiPlus	00:31:03
PS001667	PA002360	☑	10/05/2022 17:3...	ResiPlus	00:31:02
PS000881	PA000990	☐	10/05/2022 08:4...	ResiPlus	00:29:55

Figura 14. Pruebas falladas de la suite o con mayor tiempo de ejecución en ATUN (formato HH:MM:SS).

Como podemos observar en la Figura 14, dentro de las 400 pruebas que fallaron en la batería, las 10 pruebas que más tiempo duraron superan los 50 minutos de ejecución y 20 de ellas los 30 minutos. En el caso de requerir lanzarla de nuevo en el entorno local, es cada tiempo que el mantenedor debe estar dedicando completa atención a la ejecución de una sola prueba sin posibilidad alguna de retroceder en su ejecución, saltar pasos, etc.

Solución propuesta

Podrían aplicarse varias soluciones para mejorar el proceso de revisión de los resultados: volver a codificar las pruebas para optimizarlas, migrar de tecnología, la construcción desde cero de otra infraestructura de pruebas automatizadas, etc.

No son alternativas ideales debido al alto coste que supone su implementación, además de no ser las principales fuentes de los problemas causados. La dificultad con la que nos encontramos es la escasa información que en muchas ocasiones proporcionan los resultados de las pruebas y los largos tiempos de ejecución que puede dedicarse a la ejecución de una prueba automatizada.

Una posibilidad de subsanar los fallos sería mejorando los informes generados por las ejecuciones de las pruebas y la información que estas ofrecen al mantenedor. Cuando surge un fallo, en la actualidad se suelen realizar capturas de pantalla como medio de apoyo para el informe o comparaciones de archivos junto a las diferencias encontradas respecto al archivo esperado. Sin embargo, no suele ser suficiente para comprender los fallos de las pruebas

Es importante ofrecer la mayor información posible al mantenedor. Al igual que se muestran imágenes como apoyo en la revisión, se podría realizar la grabación en vídeo de la ejecución de la prueba. Esta solución mitigaría todos los defectos mostrados en el proceso de revisión de las pruebas, requiriendo de una sola ejecución de las pruebas y accediendo al vídeo si se necesitara consultar un punto en concreto de los pasos realizados por esta de forma inmediata.

4. Estado del arte

En los últimos años, en el campo de pruebas automatizadas sobre aplicaciones de escritorio se han desarrollado una serie de herramientas que podrían ser candidatas para implementar en nuestra empresa con el objetivo de mejorar el proceso de revisión de los resultados de las pruebas.

En esta sección expondremos las tecnologías ya presentes en el mercado, realizaremos un desglose de las funcionalidades que nos ofrecen y finalmente las conclusiones obtenidas tras la investigación realizada.

4.1 Herramientas presentes en el mercado

Haciendo una búsqueda en los repositorios de Internet, obtenemos herramientas alternativas que nos ofrecen funciones interesantes de cara a la administración de pruebas automatizadas y a su posterior revisión de resultados.

Cabe destacar que ha sido una tarea muy ardua debido a las pocas opciones que hemos encontrado que sean dedicadas al campo en el que realizamos nuestro trabajo. Si es cierto que existen muchas tecnologías que facilitan el lanzamiento de pruebas automatizadas en plataformas como las aplicaciones web, para una tarea como es la administración y ejecución de pruebas automatizadas en aplicaciones de escritorio existe muy poca variedad disponible en el sector.

Microsoft Test Manager[3]

[17] Herramienta desarrollada por Microsoft para la administración y ejecución de pruebas durante el ciclo de desarrollo de un producto. Fue creada para apoyar la evolución de nuevos productos tecnológicos en soporte de las herramientas de desarrollo *Visual Studio* y *Team Foundation Server*.

En esta plataforma en vez de usar *suites*, se utiliza como herramienta de trabajo los "*Test Plan*", los cuales cumplen con la misma función de agrupar las pruebas en distintas categorías o distribuciones.

Esta aplicación nos permite realizar funcionalidades como [1]:

- **Ejecución de suites de pruebas en cada versión nueva del producto**: en el lanzamiento de un incremento en el producto, la plataforma crea de forma automática una nueva batería con la ejecución de las pruebas automatizadas, detectando fallos en cada nueva versión.
- **Grabación de los pasos de cada prueba**: genera un *script* con las acciones del usuario realizadas durante las pruebas y permite al mantenedor de las pruebas repetir cada acción, retroceder en la ejecución, etc.

[3] ***Microsoft Test Manager*** *– Página oficial: https://visualstudio.microsoft.com/es/vs/test-professional/*

- **Resultados de las pruebas**: otorga retroalimentación al mantenedor de las pruebas ofreciendo detalles e información de su ejecución, así como del resultado final y las distintas comprobaciones y pasos realizados, como se muestra en la Figura 15.
- **Análisis y gráficas con los resultados de las pruebas**: muestra en pantalla gráficas, estadísticas e información personalizable relevante de las pruebas ejecutadas en cada "*Test Plan*". En la Figura 16 se muestra un ejemplo donde proyecta de forma estadística la ejecución de un "*Test Plan*" en la aplicación.

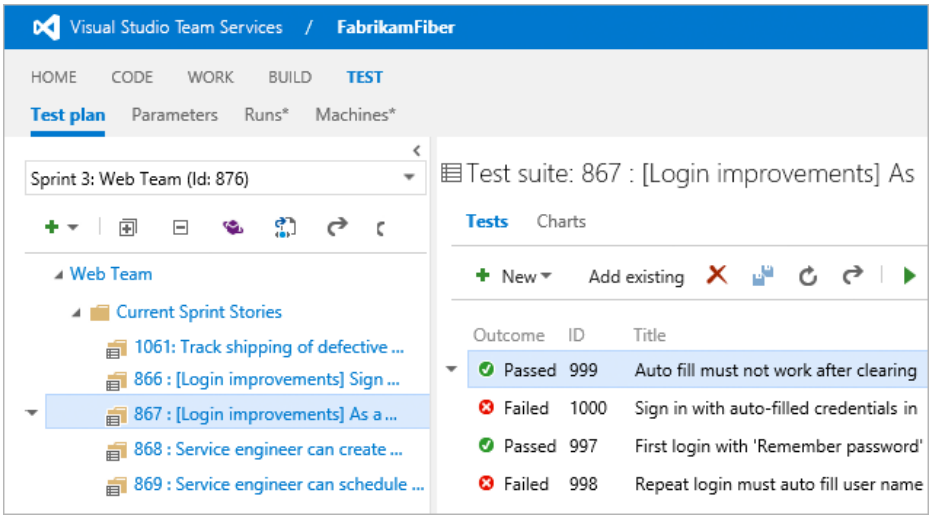


Figura 15. Ejemplo de ejecución de una batería de pruebas automatizadas en Microsoft Test Manager. [19]

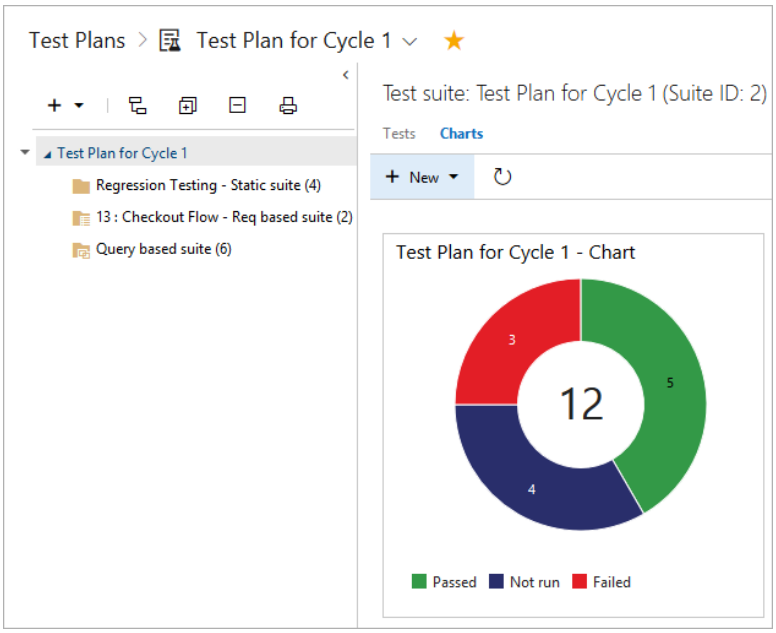


Figura 16. Formulario con gráfica de la ejecución de un "Test Plan" en Microsoft Test Manager. [24]

A pesar de contener funcionalidades y métodos que podrían facilitar la labor en la revisión de pruebas, no es una herramienta que pueda ser candidata para nuestro trabajo. Esto es debido a los siguientes inconvenientes:

- **Introducción manual de los casos de prueba**: para cada prueba automatizada programada en nuestro repertorio actual hay que crearla una por una sin posibilidad de migrar entre la plataforma de nuestra empresa y esta nueva infraestructura de forma automatizada.
- **Reproducción de los pasos de la prueba no disponible en la aplicación testeada**: si es cierto que sería una funcionalidad idónea para resolver la problemática de la revisión de pruebas automatizadas, aplicarlo en nuestra aplicación testeada sería algo muy complicado y costoso de implementar. Revertir acciones de usuario, saltarse pasos de las pruebas o recuperar estados de la aplicación distintos al actual resulta una tarea imposible por la naturaleza de la aplicación. Guardar un estado anterior de nuestra aplicación requiere de generar *scripts* de bases de datos muy extensos, además de requerir de diversos reinicios al programa para que estos cambios se apliquen.

IBM Rational Quality Manager[4]

Sistema *software* en línea creado por la compañía IBM el cual está destinado a la gestión de calidad de *software* de los proyectos. Se trabaja mediante el uso de casos de prueba en los cuales pueden ser automatizados por la herramienta y ser lanzados por varios usuarios de forma concurrente.

Ofrece funcionalidades interesantes de cara a la gestión de proyectos, de las cuales podríamos destacar [10]:

- **Planificación de lanzamiento de baterías de pruebas:** permite encolar y asignar fechas de ejecución de pruebas automatizadas. Esto es útil cuando el lanzamiento requiere de grandes recursos computacionales y no quiere interferir en la actividad de la empresa.
- **Asignación de tareas a usuarios de la empresa:** al trabajar en equipos de desarrollo, la comunicación es un elemento fundamental para el éxito del proceso. La plataforma da la posibilidad de repartir tareas y pruebas asignadas del proyecto a los distintos miembros del grupo de trabajo, dejando en constancia y de forma clara en el *dashboard* del proyecto los elementos asignados a cada usuario y acceso en detalle a la descripción de cada uno. En la Figura 17, expone al usuario las tareas de pruebas que tiene pendientes.
- **Independencia con la plataforma de trabajo:** esta plataforma facilita su uso a la empresa y permite manejar los casos de prueba de nuestro proyecto sin importar las tecnologías o librerías con las que se estén trabajando.
- **Integración directa con otras tecnologías de IBM:** su integración directa con las pruebas ya programadas de nuestra empresa es posible debido al uso que hacen estas de las librerías y tecnologías de RFT, las cuales fueron desarrolladas por la misma desarrolladora.

[4]***IBM Rational Quality Manager*** *– Página oficial: shorturl.at/bnswz*

Figura 17. Dashboard principal de IBM Rational Quality Manager. [11]

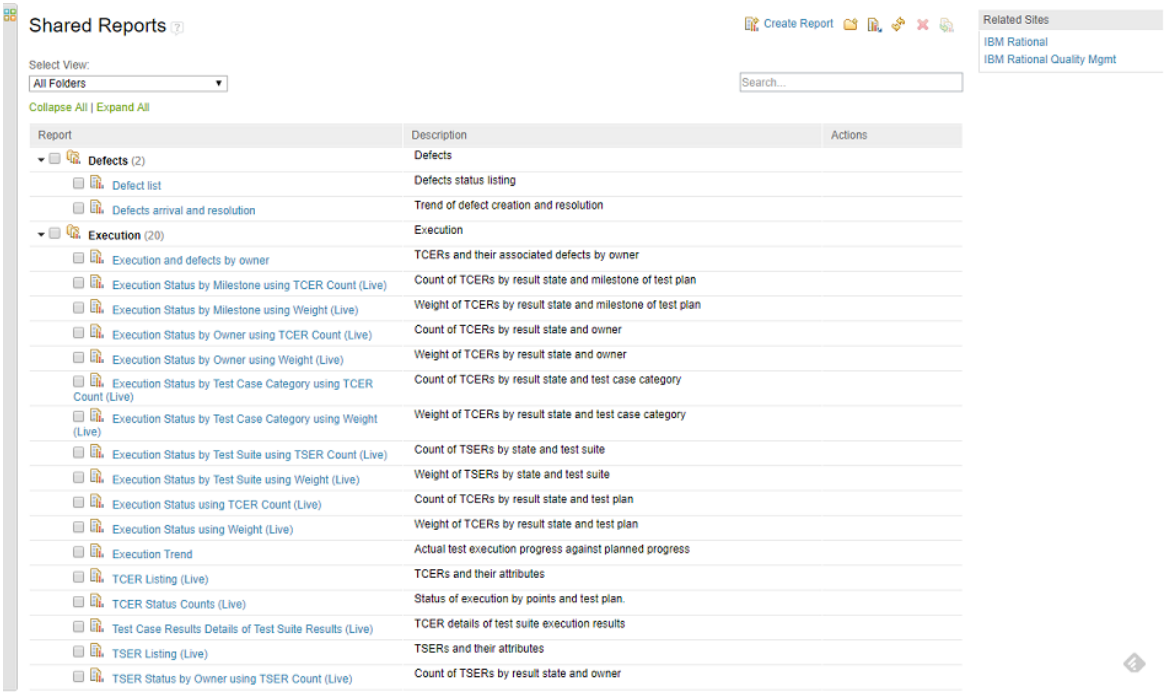


Figura 18. Lista de los registros de pruebas en un proyecto tecnológico en IBM Rational Quality Manager. [11]

Sin embargo, esta herramienta tampoco es un candidato para resolver los problemas en la revisión de pruebas automatizadas. Aunque exista la opción de una directa interoperabilidad con la tecnología de pruebas automatizadas *Rational Functional Tester*, esta plataforma sigue sin mostrar toda la retroalimentación suficiente al mantenedor de pruebas para evitar la ejecución reiterada de pruebas, como se muestra en la Figura 18. En caso de fallo, no ofrece mucha mayor información al usuario que ha lanzado las pruebas y seguirían surgiendo los mismos inconvenientes en el proceso de revisión y reparación de pruebas.

Kualitee[5]

Plataforma en la nube diseñada para el manejo y ejecución de pruebas, mejorando y optimizando la *QA*[6] de los proyectos tecnológicos. Una característica destacable en este servicio *cloud* es la posibilidad de combinar pruebas manuales y automáticas en una misma batería.

Entre los servicios ofrecidos por la plataforma en relación con la ejecución de pruebas automatizadas podemos resaltar los siguientes [13]:

- **Historial de la ejecución de baterías de pruebas:** ofrece a los mantenedores un detallado historial de su ejecución. Permite, mostrado en una línea de tiempo y en varios gráficos, una mayor perspectiva del proyecto e información respecto a su avance en sus distintas versiones.
- **Interfaz sencilla**: en comparación con otros productos de la competencia, el apartado gráfico de la aplicación es bastante más fácil de usar y de familiarizarse. Esto permite a los usuarios reducir su tiempo de formación para aprender a utilizar la herramienta, algo que en otras tecnologías no es posible debido a sus extensos detalles y dificultad de uso.
- **Posibilidad de acceder a la herramienta con dispositivos móviles:** en relación con el apartado anterior, la simplicidad que ofrece ha permitido poder adaptar de forma simple la tecnología para poder acceder a ella con dispositivos de menor tamaño, como los *smartphones*.
- **Integración directa con librerías de pruebas automáticas:** ofrece total interoperabilidad con las bibliotecas más populares del sector dedicadas a la ejecución de pruebas automatizadas. Entre ellas se encuentran *Selenium*[7], *Jira*[8] y *Jenkins*[9].
- **Lanzamiento de pruebas optimizado y sencillo**: debido a la sencillez de la plataforma que se ha comentado en uno de los puntos, ofrece la posibilidad de ejecutar de forma remota baterías de pruebas en simples pasos, independientemente de la plataforma en la que se encuentre el usuario y de su la localización. Como observamos en la Figura 19, se tiene a disposición el acceso a la aplicación desde dispositivos móviles y ordenadores de sobremesa.

[5] ***Kualitee*** *– Página oficial: https://www.kualitee.com/*
[6] ***QA (Quality Assurance):*** *[20] término que hace referencia a las tareas encargadas de asegurar el correcto funcionamiento de los productos software y de la ausencia de fallos en los servicios y funcionalidades.*
[7] ***Selenium*** *– Página oficial: https://www.selenium.dev/*
[8] ***Jira*** *– Página oficial: https://www.atlassian.com/es/software/jira*
[9] ***Jenkins*** *– Página oficial: https://www.jenkins.io/*

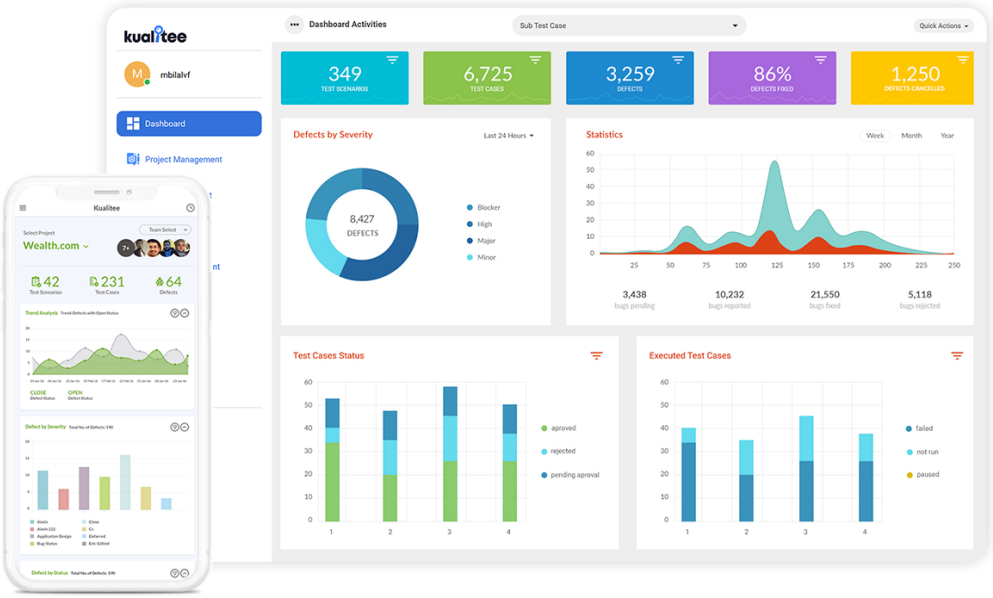


Figura 19. Dashboard principal de la plataforma Kualitee en la versión de escritorio y en la de móvil. [13]

A pesar de ser una herramienta con mucho potencial por su reciente incorporación al mercado, no resulta ser una plataforma ideal para nuestro proyecto. Esto se debe a los siguientes motivos:

- **Necesidad de migración de las pruebas a otras tecnologías:** hemos destacado anteriormente la posibilidad de la herramienta de poder integrar directamente librerías de pruebas automatizadas. En nuestro proyecto, con la tecnología aplicada actualmente, esta no sería capaz de interoperar de forma correcta. Esto significa que, para poder ser utilizada correctamente, deberíamos programar de nuevo todas las pruebas de la infraestructura actual a los nuevos *frameworks*.
- **Información poco detallada en los resultados de las ejecuciones de las baterías:** si es cierto que ofrece un apartado estadístico muy potente, en la sección de motivos de los fallos de las pruebas ejecutadas no es precisa debido a su brevedad. Por tanto, el resultado de aplicar esta plataforma al proceso de revisión de pruebas nos conduce a los mismos problemas que se presentan actualmente.

4.2 Conclusiones

Entre las herramientas comentadas ninguna de ellas es relevante para ayudar a mejorar la tarea de revisión de pruebas automatizadas. La mayoría de estas tecnologías ofrecen funcionalidades distintas a las que requiere nuestra empresa y para contextos muy distintos al lanzamiento de pruebas masivas en equipos remoto. Las aplicaciones mostradas están pensadas para un público y un mantenimiento de pruebas muy inferior al que disponemos actualmente en la infraestructura de nuestra empresa y no solucionarían los problemas que surgen en el proceso de revisión de sus resultados.

Además, al querer implementar una funcionalidad única y no previamente diseñada en ninguna herramienta del mercado, debemos barajar entre qué opción sería la más

adecuada para el desarrollo de este trabajo: implementar una aplicación e infraestructura desde cero implementando los nuevos requisitos de grabación de vídeo o adaptar la tecnología que disponemos en estos momentos añadiendo las nuevas funciones para mejorar las tareas de revisión.

Se ha considerado que la opción más acertada y con mayor garantía de éxito era la aplicar una mejora en la infraestructura de pruebas automatizadas actual añadiendo los nuevos requisitos. Las únicas necesidades que requerimos es la de soportar la grabación de vídeo de nuestras pruebas y poder visualizar los archivos multimedia desde sus resultados. Desarrollar una herramienta completamente nueva para esta funcionalidad requeriría, no solo de una costosa implementación al reprogramar todo el sistema, sino de un esfuerzo que no lograría mayores resultados o satisfacción al mantenedor de las pruebas a la hora de realizar las revisiones de *suites*.

En definitiva, para el desarrollo de este trabajo, añadiremos la funcionalidad de grabación de vídeo en el proyecto *AddFt*, soporte de grabación de pruebas en el lanzador de pruebas ATUN y modificaciones en los informes generados por las pruebas para la visualización de los archivos de vídeo desde los propios resultados de las pruebas.

5. Tecnologías utilizadas

Para el desarrollo de la solución nos apoyaremos en una serie de tecnologías y herramientas presentes en el mercado. En las nuevas funcionalidades de grabación de vídeo en el proyecto de *AddFt*, tendremos mayor libertad a la hora de seleccionar la aplicación para implementar. En cambio, si queremos adaptar el proyecto actual para que soporten las funcionalidades y el manejo de las grabaciones de vídeo, estaremos limitados a utilizar los mismos lenguajes y tecnologías establecidos en el lanzador de pruebas automatizadas. De lo contrario, deberíamos migrar todo el proyecto a otra tecnología, cuya opción que sería muy costosa de realizar.

En los siguientes subcapítulos, expondremos las distintas aplicaciones y tecnologías que tenemos a nuestra disposición en el mercado para cubrir los nuevos requisitos y su comparativa, las razones de la opción seleccionada y las tecnologías presentes en la infraestructura actual de las pruebas automatizadas.

5.1 Grabadora del escritorio del ordenador

En el mercado existe una gran variedad de aplicaciones destinadas a la captura de vídeo de las pantallas de ordenadores. La amplitud de opciones e interés en este tipo de servicios y productos ha crecido notoriamente en los últimos años. Este fenómeno se debe a la creciente demanda de las plataformas multimedia en línea, como *YouTube* y *Twitch*, y a la necesidad de digitalización de muchos sectores debido a la pandemia del COVID-19, como es la educación e instituciones dedicadas a la formación. En la Figura 20, se proyecta como la gráfica de interés crece notoriamente a partir del primer trimestre del 2020.

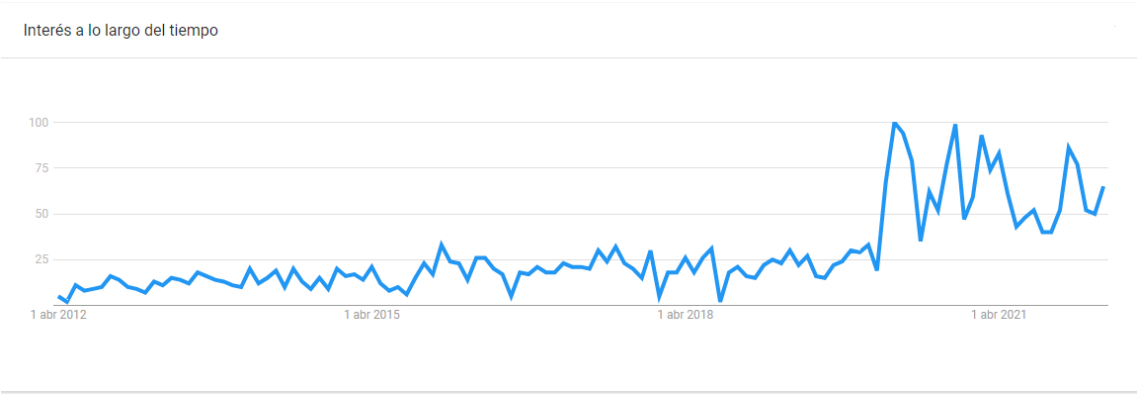


Figura 20. Porcentaje de interés a lo largo del tiempo en España en la búsqueda del término "grabadora de pantalla" en Google desde 2012 hasta 2022. [8]

En consecuencia, el rango de aplicaciones que se nos presentan para el desarrollo de este trabajo es muy amplia. Habiendo realizado una búsqueda de las distintas alternativas disponibles, la que más se ajusta a nuestras necesidades es *FFmpeg*.

Tras un análisis de las distintas opciones que se nos ofrecían, nos encontramos también con otras herramientas interesantes que podrían haber sido candidatas para utilizarlas

en el desarrollo de esta solución, como son *Camtasia Studio*[10], *Icecream Screen Recorder*[11] y *Bandicam*[12].

Sin embargo, surgían varios inconvenientes a la hora de implementar dichas alternativas con la infraestructura de pruebas automatizadas actual en nuestra empresa, como son los siguientes:

- **Altos costes del *software***. Cuando hablamos de costes nos referimos tanto monetarios, a la hora de adquirir licencias *premium* y versiones completas de los productos, como a la demanda de recursos computacionales en el momento de ejecutar la tecnología en nuestros equipos (memoria, procesador, etc).
- **Limitaciones de uso de los programas y de las licencias de usuario**. En las condiciones de uso y servicios de varios programas, no se permite reutilizar las mismas membresías entre varios equipos, lo que multiplica los costes de ellas a la hora de equipar todos los equipos y las máquinas virtuales de la empresa con el *software* de grabación de vídeo.
- **Escasa flexibilidad en los servicios**. La mayoría de estos productos están enfocados a un público y usuarios sin mucha experiencia en la informática o con pocos conocimientos técnicos. Por lo que, al requerir de grabaciones personalizadas en este trabajo, muchos de estos sistemas ofrecen muy pocas alternativas tanto a la hora de grabar en vídeo como al generar los archivos en formatos y calidades muy concretos.
- **Versiones gratuitas con funcionalidades muy restringidas y de una calidad inferior**. Es bastante frecuente que las compañías publiquen una alternativa sin coste del *software* que comercializan. En cambio, dichas alternativas suelen contar con licencias de usuario limitadas a un tiempo muy acotado de uso, servicios a disposición muy reducidos respecto a la versión de pago del programa o *banners* de publicidad y ventanas emergentes que entorpezcan la experiencia del usuario. La meta de estas versiones de *software* es, en muchas ocasiones, para que el usuario pueda probar la aplicación y posteriormente comprar la versión completa del producto de forma opcional.
- **Problemas con la privacidad y filtrado de información.** Este punto tiene relación con el apartado anterior, el cual se justifica la aparición de estas alternativas sin costo en el mercado del *software*. Existen casos en la competencia donde, para poder realizar un uso gratuito de su *software*, se ceden datos e información de su uso para otros fines, como son comerciales, seguimiento de uso, análisis de datos, etc. Dicha propagación puede contener información sensible de la empresa, en cuyo caso puede suponer un gran riesgo para su seguridad y la de su infraestructura.
- **Ardua implementación con el proyecto de pruebas automatizadas.** Al tratar con programas cuya interacción con el usuario se realiza mediante interfaces gráficas, supone mayores dificultades de integración respecto a herramientas que sean utilizadas mediante líneas de comandos. Además, requeriría del uso de una tecnología adicional para simular las acciones del

[10] ***Camtasia Studio*** *– Página oficial: https://www.techsmith.es/store/camtasia*
[11] ***Icecream Screen Recorder*** *– Página oficial: https://icecreamapps.com/Screen-Recorder/*
[12] ***Bandicam*** *– Página oficial: https://www.bandicam.com/es/*

usuario en la interfaz gráfica, lo que aumentaría el uso de recursos y de tiempo de respuesta con la aplicación.

Con las problemáticas expuestas de las otras herramientas, consideramos que *FFmpeg*[13] posee grandes ventajas en comparación con la competencia del sector y mitiga las principales desventajas comentadas anteriormente.

En la siguiente subsección haremos una exposición de la herramienta de grabación de vídeo seleccionada, sus funcionalidades, características y un breve análisis de sus virtudes y defectos encontrados a la hora de implementarlo en nuestro trabajo.

FFmpeg

[5] La aplicación *FFmpeg* se trata de un *software* libre y gratuito desarrollado en el lenguaje C dedicado a la grabación, renderizado, manejo y retransmisión en vivo de audio y vídeo, cuya interacción con el usuario se realiza mediante línea de comandos, como muestra la Figura 21.

Esta herramienta reúne una gran cantidad de funciones y parámetros que nos permite crear y personalizar nuestras grabaciones de vídeo con las características que deseemos y generarlas en archivos de vídeo con las calidades y formatos deseados. Entre las opciones que podemos establecer en las instrucciones se encuentran: resolución de vídeo, fotogramas por segundo[14], formato de salida del archivo de vídeo, *presets*[15], etc.

Por ejemplo: si quisiéramos grabar el escritorio del ordenador para la demostración en vídeo de la defensa de nuestro trabajo en una resolución 1980x1080 píxeles y 30 fotogramas por segundo, lo realizaríamos lanzando el siguiente comando en la terminal del sistema operativo:

Ffmpeg -f gdigrab -framerate 30 -i desktop -video_size 1920x1080 nombre_grabación.mp4

Para detener la grabación, nos bastaría con cerrar el programa manualmente o presionando la tecla ‘q’ en la ventana del proceso para detener la grabación. En el caso de que detengamos forzosamente el proceso (mandando una señal “*kill*” al proceso) el vídeo no terminaría de procesarse correctamente y el archivo quedaría dañado, una situación que deberíamos evitar usando la aplicación.

---

[13] ***FFmpeg*** *– Página oficial: https://ffmpeg.org/*
[14] ***Fotogramas por segundo*** *(o tasa de fotogramas): [6] medida de frecuencia en la cual se muestran o graban distintas imágenes en un vídeo por cada segundo.*
[15] ***Preset****: [2] ajustes para el revelado de fotogramas, aplicado tanto para la fotografía como la edición de vídeo.*

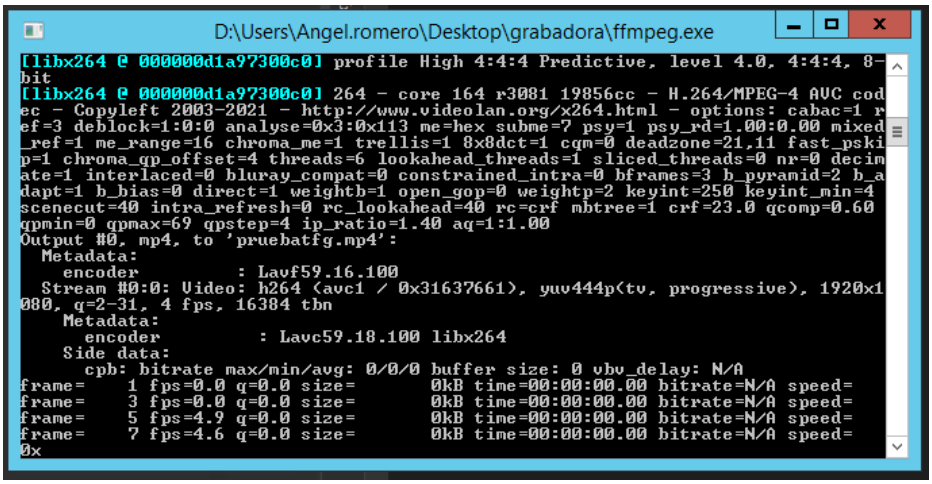


Figura 21. FFmpeg durante la grabación de pantalla de una máquina virtual.

Las **ventajas** con las que nos encontramos al integrar la aplicación *FFmpeg* con *AddFt* respecto a otros programas son:

- **Simplicidad y rapidez en la ejecución de tareas**: mediante un comando ejecutado en la terminal, el programa es capaz de lanzar los servicios solicitados por el usuario de forma inmediata. Además, incluso tras largas sesiones de grabación de vídeo, la herramienta es capaz de renderizar el archivo de vídeo de forma optimizada.
- **Funciones personalizadas**: los servicios del programa nos ofrecen la posibilidad de parametrizarlos con un gran número de argumentos en las instrucciones. Como hemos visto en el ejemplo anterior, permite ajustar los parámetros de calidad deseados en la salida de vídeo y el formato deseados.
- **Uso de recursos optimizado**: al no contar con una interfaz gráfica y hacer uso únicamente del núcleo operacional del programa, permite centrarse todos los recursos dedicados en la funcionalidad de la herramienta. Hay que añadir que, al ser programado en C y tener el algoritmo público, ha permitido aplicar optimizaciones en el código desarrollados por la comunidad de programadores.
- **Código abierto**: [5] el proyecto posee una licencia GNU LGPL 2.1 (GNU *Lesser General Public License*), la cual permite al usuario usar, distribuir y modificar el código fuente del programa de forma libre. Para nuestro trabajo, al permitirnos visualizar el algoritmo de la aplicación, nos otorga más seguridad en la integridad de la infraestructura que lanza las pruebas automatizadas y los datos confidenciales de la empresa.

Por lo contrario, hay que resaltar un **inconveniente** a la hora de hacer uso de este programa. Si es verdad que cuenta con mucha personalización a la hora de hacer uso de los servicios que ofrece, estos son difíciles de lanzar por los usuarios convencionales y la aplicación no destaca por ser fácil de utilizar. Como recordamos en el ejemplo anterior de grabación de la pantalla, requiere de varios parámetros a definir por el usuario para lanzar la funcionalidad requerida. Sin embargo, este problema puede solventarse debido a la gran comunidad que sustenta la tecnología y a los amplios recursos y documentaciones ofrecidas en la página oficial del producto y la comunidad de desarrolladores en Internet.

En conclusión, *FFmpeg* ofrece de forma gratuita y libre todo el núcleo de las funciones necesarias para la grabación en vídeo de las pruebas automatizadas. Nos encontramos ante un programa que, con pocos recursos computacionales, su algoritmo publicado en los repositorios de su página oficial, personalización absoluta de las funcionalidades y rapidez a la hora de lanzar los procesos.

5.2 Lanzador de pruebas automatizadas

El proyecto *AddFt* y ATUN requieren de cambios distintos en el código para la implementación de los nuevos requisitos. Desglosaremos por cada proyecto las modificaciones previstas y las tecnologías que haremos uso durante el trabajo.

AddFt

Dentro del proyecto local de pruebas automatizadas, debemos añadir la posibilidad al mantenedor de pruebas si desea habilitar las grabaciones de pantalla durante su ejecución.

Para conseguir dicha finalidad, se requiere de una modificación en los parámetros del método encargado de ejecutar las pruebas automatizadas. Como recordamos en el capítulo tres, el usuario puede establecer diversos parámetros que variarán la ejecución de la prueba, como se visualiza en la Figura 22. Estos atributos son: el identificador de la prueba, modo de ejecución (completa, regresión…), modo de búsqueda de la prueba y la base de datos a usar para el almacenamiento de los datos en la aplicación durante la ejecución de la prueba.

```
Public Shared Sub RunManagers(ByVal filter As String, Optional ByVal mode As RunMode = RunMode.All,
                              Optional ByVal searchMode As SearchMode = SearchMode.FullNameContains,
                              Optional ByVal bd As String = "DatosResidencia_001")
    RunManagers(AddressOf FilterType, New Object() {GetType(Manager), filter, searchMode}, mode, bd)
End Sub
```

Figura 22. Método utilizado por los usuarios para lanzar una prueba automatizada y sus parámetros de ejecución.

Por tanto, realizaremos una modificación en el algoritmo encargado de su ejecución para la inclusión de los nuevos requisitos. Se seguirá haciendo uso del lenguaje de programación Visual Basic junto al *framework* .NET y las bibliotecas diseñadas anteriormente por la empresa para la ejecución de pruebas automatizadas.

Lanzador de pruebas ATUN

Para implementar la herramienta de grabación de vídeo en remoto, debemos adaptar toda la infraestructura de máquinas virtuales y el sistema *software* de lanzamiento de pruebas automatizadas en los equipos.

Al seleccionar las pruebas automatizadas que deseamos lanzar y pulsar el botón “Encolar” de la ventana principal de la aplicación, se iniciará el proceso de lanzamiento de las pruebas. En el momento en que seleccionamos las máquinas virtuales disponibles que realizarán la ejecución de la batería de pruebas en la interfaz de ATUN, el lanzador se encarga de registrar la petición en las bases de datos compartidas con la

infraestructura de máquinas virtuales junto a los parámetros establecidos en la configuración del programa. Dentro de dichas variables, debemos incluir la posibilidad al lanzador de querer habilitar la funcionalidad de grabación en vídeo en remoto.

Además, para establecer información al lanzador de pruebas y poder modificar las preferencias en la ejecución de las pruebas, existe una ventana "Configuración" en la cual el usuario puede establecer los parámetros necesarios para su ejecución, como se muestra en la Figura 9.

Por ello, debemos realizar las modificaciones en los aspectos mostrados en los párrafos anteriores. Haremos uso del lenguaje C# y *framework* .NET, los cuales ya fueron utilizados para programar el algoritmo del lanzador de pruebas ATUN para modificar el código requerido para crear la instancia en la base de datos de ejecución de pruebas a las máquinas virtuales e incorporar el parámetro de grabación de la prueba.

Para modificar la ventana de configuración de proyecto, nos apoyaremos en la tecnología *Windows Form*, encargada de mostrar la interfaz gráfica de la aplicación y la retroalimentación de la aplicación y la infraestructura. Será necesario para agregar las opciones y parámetros para personalizar el lanzamiento de baterías con los nuevos requisitos incorporados.

5.3 Generación de resultados de las pruebas automatizadas

Al finalizar la ejecución de una prueba automatizada, desde *AddFt* se generan los ficheros HTML de los informes con el resultado detallado de su ejecución y la traza del código en el caso de detectar alguna excepción en RFT.

La función de generación de estos ficheros está implementada previamente en el proyecto *Base* bajo el *framework* .NET (haciendo uso del lenguaje de programación Visual Basic) y una biblioteca de generación de resultados de las ejecuciones procedente del *framework* de *Rational Funcional Tester*. Dicha librería establece las cabeceras de los métodos necesarios para la generación de los registros de las pruebas, y son utilizades por las clases *Log* y *LogManager* de *AddFt* para extender la lógica en dichas funciones. En la Figura 23 se muestra como en el método encargado de escribir fallos en la ejecución de la prueba llama a distintos métodos de la librería de RFT para escribir en el informe.

```
Public Shared Sub Failure(e As Exception, Optional includeSnapshot As Boolean = True, Optional includeStackTrace As Boolean = True)
    Try
        Dim message As String = "<b>Exception: " + e.GetType().FullName + "</b><br/>" + e.Message
        ApplyManagerResultInfo("Exception: " & e.GetType().FullName & " Message: " + e.Message, ActionResult.Failure)
        If includeStackTrace Then
            message = message + FormatStackTrace(e.StackTrace)
        End If

        Dim aux As Exception = e.InnerException
        While Not aux Is Nothing
            message = message + "<br/>INNER EXCEPTION: " + aux.GetType().FullName + "</b><br/>" + aux.Message
            If includeStackTrace Then
                message = message + FormatStackTrace(aux.StackTrace)
            End If
            aux = aux.InnerException
        End While

        Failure(message, includeSnapshot, False)
    Catch ex As Exception
        LogError(ex)
    End Try
End Sub
```

Figura 23. Método de la clase Log.vb encargado de escribir en el registro de la prueba un fallo detectado durante su ejecución.

Dentro del proyecto Base, debemos editar el algoritmo para poder visualizar directamente el vídeo desde los informes de las pruebas. Para ello, modificaremos el proyecto Base, añadiendo los métodos y la lógica necesaria para que en los ficheros de informes de las pruebas se permita acceder al vídeo.

Haremos uso de las bibliotecas y métodos ya creados para la escritura de los nuevos elementos dentro de los resultados. Requeriremos de conocimiento en HTML para la introducción del reproductor del vídeo nativo dentro de la página.

Sin embargo, no siempre estará presente el vídeo en los sistemas de almacenamiento. El papel que tiene estos archivos es más bien temporal y, al ejecutar de nuevo la misma prueba, la utilidad de la antigua grabación de vídeo habrá finalizado, lo cual sería ideal eliminarlo para permitir espacio de almacenamiento para nuevas grabaciones. Por ello, realizar *hard-coding* [16] en el HTML haciendo referencia al archivo de vídeo directamente sin comprobar que este sigue existiendo no sería una buena práctica.

Para solucionar el inconveniente expuesto en el párrafo anterior, deberemos escribir en el código HTML del fichero de los resultados una etiqueta "*script*" posterior al título, como se muestra en el código resaltado de la Figura 24. Dicha alternativa nos permitirá añadir piezas código en lenguaje JavaScript y ser ejecutados al abrir los archivos HTML de los informes de las ejecuciones. Es decir, de forma automática y sin realizar cambios manuales en el código fuente de los archivos, podemos comprobar al abrir el fichero si aún existe el archivo de vídeo y, en caso de existir, mostrará el reproductor de vídeo asociado a la grabación. De lo contario, el reproductor no se generará en el fichero, puesto que el vídeo de la ejecución de la prueba fue eliminado o ya no está disponible.

[16] ***Hard-coding:*** *[9] práctica en la programación que hace referencia a la obtención de datos directamente desde el código fuente de un programa en vez de hacerlo mediante fuentes externas de información.*

Figura 24. Porción del código fuente HTML del resultado generado por una prueba automatizada.

El **motivo principal** para el uso de JavaScript como scripting en el fichero del resultado es por el soporte nativo que tiene con HTML con la etiqueta "*<script>*" y por su funcionamiento *client-side*[17]. Al abrir el fichero con el código HTML, será el navegador el encargado de ejecutar todo el algoritmo escrito dentro de la etiqueta.

De esta forma, dotaremos de mayor calidad a los resultados generados por las ejecuciones de las pruebas automatizadas. De lo contrario, podría generar inconsistencias y falta de retroalimentación a los mantenedores de las pruebas si siempre mostrase el reproductor de vídeo en los informes sin importar si sigue existiendo el archivo o fue posteriormente eliminado.

[17] ***Client-side**: [14] en el contexto de una aplicación, se refiere a las operaciones y funciones que se ejecutan en el lado del cliente.*

6. Desarrollo de la solución

Al instalar el programa de grabación de vídeo en la infraestructura, debemos automatizar su uso en la ejecución de pruebas automatizadas. Por ello, realizaremos una mejora en el proyecto *AddFt* para poder grabar en vídeo las ejecuciones de las pruebas.

Para ello, debemos modificar los dos pilares del lanzamiento de pruebas automatizadas de nuestra empresa: el proyecto de pruebas ***AddFt*** y el lanzador de pruebas automatizadas **ATUN**.

Durante el presente capítulo realizaremos, bajo el uso de una metodología ágil, las modificaciones de los programas de la empresa para combinar las grabaciones de vídeo con las ejecuciones de pruebas automatizadas en toda la infraestructura.

6.1 Requisitos

Antes de entrar en aspectos y detalles tecnológicos específicos, debemos entender realmente el producto que debemos desarrollar y las modificaciones en la infraestructura. Antes de realizar cualquier prototipado o diseño específico, hay que generar los paquetes de trabajo del proyecto y documentar todos sus requisitos.

Durante la etapa previa a la inicialización del trabajo, se aplicaron dos planes de elicitación de requisitos para obtener los nuevos requisitos por parte de los *stakeholders*. En primer lugar, se hicieron entrevistas cerradas a los miembros del departamento de *testing* automatizado sobre las posibles funciones novedosas para incluirlas al sistema de grabación de vídeo. Finalmente, teniendo en cuenta los datos recogidos y las opiniones de la técnica anterior, se realizó una sesión individualizada de *brainstorming* donde, a partir de los puntos obtenidos en las entrevistas con los compañeros de la empresa, se obtuvieron los paquetes de trabajo que formarían parte del *backlog* del proyecto.

Durante una sesión de *brainstorming*, no existe mayor limitación que dejar constancia de todas las ideas del nuevo producto que puedan surgirnos sin restricciones ni prejuicios sobre estas. Al finalizar esta etapa, debemos recapitular todos los elementos generados y darles una prioridad en su implementación. Es importante asignarle un orden de desarrollo ya que, si consideramos que todos los elementos a implementar en nuestro proyecto son prioritarios, realmente ninguno lo es.

La técnica que se ha aplicado para la asignación de prioridades a cada uno de los elementos es **MoSCoW** [18], categorizando cada uno de los paquetes de trabajo generados y así tener una visión global del proyecto, poniendo en énfasis aquellas funcionalidades que son más importantes.

[18] *Priorización **MoSCoW** en detalle: shorturl.at/bdnRX*

Tras aplicar todos los pasos y técnicas mencionadas en la elicitación de requisitos y darles una prioridad, el *backlog* de nuestro proyecto con los paquetes de trabajo queda mostrado en la Figura 25.

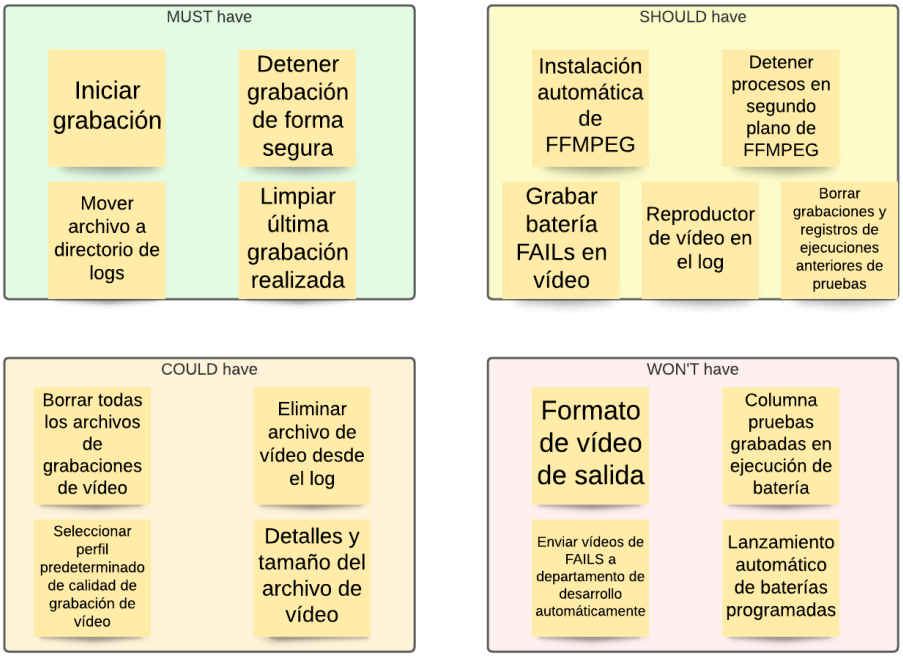


Figura 25. Elementos del backlog priorizados tras aplicar MoSCoW en su elicitación de requisitos.

Como recordamos, el proyecto tiene una duración limitada y el esfuerzo el cual podemos dedicarle al desarrollo del trabajo está acotado. Debemos estimar para cada elemento el esfuerzo que convendría completar su paquete de trabajo durante todas sus fases.

Para este proyecto hemos estimado mediante puntos de esfuerzo tanto la capacidad que tenemos en cada *sprint* [19]del proyecto como el trabajo necesario para completar cada uno de los elementos generados en el *backlog*. En promedio, se ha estimado que la cantidad total a poder destinar en cada *sprint* es de 10 puntos de esfuerzo y en total se realizan dos *sprints* de cuatro semanas cada uno (en total, 20 puntos).

Teniendo en cuenta la priorización realizada anteriormente y tras haber estimado en puntos cada uno de los paquetes de trabajo, los elementos del *backlog* que formarán parte de nuestro proyecto para los dos *sprints* son los siguientes (ordenados de mayor a menor prioridad de implementación):

- **Iniciar grabación** – 5 puntos.
- **Detener grabación de forma segura** – 3 puntos.
- **Mover archivo de vídeo a carpeta de logs** – 1 punto.
- **Limpieza de la última grabación realizada** – 1 punto.
- **Reproductor de vídeo en el log** – 2 puntos.
- **Grabar batería de *FAILs* en vídeo** – 5 puntos.

[19] ***Sprint***: *[23] periodo de tiempo el cual tiene por objetivo final realizar un incremento del producto usable.*

- **Instalación automática de *FFmpeg*** – 2 puntos.
- **Detener procesos en segundo plano de *FFmpeg*** – 1 punto.

Antes de comenzar con un *sprint*, a cada elemento asignado se documentará sus requisitos mediante casos de uso (se muestra un ejemplo en la Figura 26) y prototipos visuales que simulen el comportamiento esperado del sistema durante su uso, realizando así una especificación de requisitos semiformal combinando modelos y el lenguaje natural. Serán utilizados como elemento imprescindible para que, al finalizar con su desarrollo, se valide si realmente es la funcionalidad con la que esperaba el usuario.

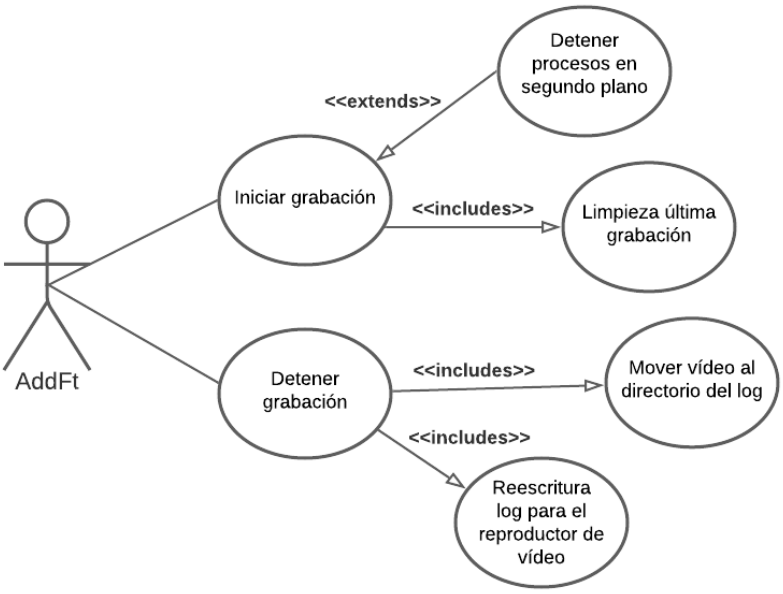


Figura 26. Caso de uso de grabación de vídeo con FFmpeg en AddFt.

6.2 Diseño

A lo largo de esta fase, se adaptarán los requisitos de usuario a las restricciones y características que la tecnología posee. Durante este proyecto, el único módulo independiente que se desarrollará de cero será el encargado de interoperar el proyecto de pruebas automatizadas *AddFt* con la grabadora de vídeo. El resto de los subsistemas mantendrán el mismo esquema y arquitectura de diseño con el que contaban previamente, añadiendo las modificaciones necesarias por los requisitos establecidos en la fase anterior.

Implementación de *FFmpeg* con el proyecto de pruebas automatizadas

Como hemos expuesto en el capítulo de tecnologías utilizadas, la herramienta seleccionada para realizar las tareas de grabación de vídeo se adapta perfectamente a nuestras necesidades. Sin embargo, al ser un programa de interacción con el usuario mediante línea de comandos, no es posible integrar directamente el código fuente del

programa con la infraestructura de *testing* automatizado al ser programados en lenguajes distintos.

Se han considerado varias alternativas para el ensamblado de ambos proyectos. La posibilidad de migrar el código fuente de *FFmpeg* a .NET fue descartada al no tener acceso a diversas librerías con las que el programa contaba y por requerir de un gran esfuerzo para llevarlo a cabo.

La solución óptima para conseguir una correcta integración con el proyecto de *AddFt* es mediante una clase ***wrapper*** que interactúe con el ejecutable oficial del programa simulando las acciones del usuario (ejecución de la aplicación junto a los parámetros deseados, manejo de excepciones, ficheros y fallos, etc).

Para programar el *wrapper*, usaremos del patrón de diseño *Facade* en las funcionalidades y servicios que creemos. Esto nos permitirá varias facilidades a la hora de integrar la aplicación con la infraestructura de pruebas automatizadas.

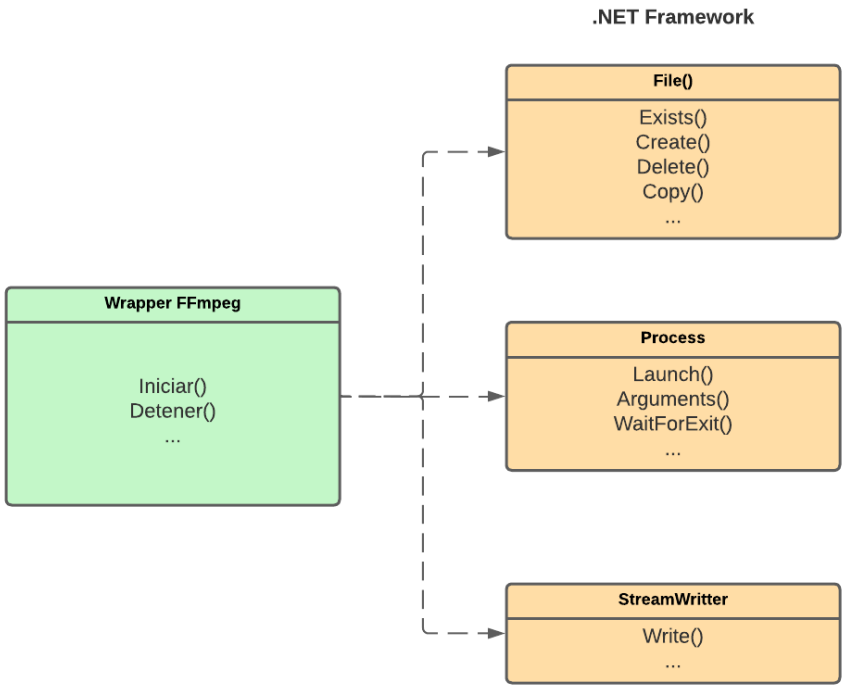


Figura 27. Esquema del funcionamiento de la clase Wrapper de integración con FFmpeg.

La principal ventaja que nos ofrece este patrón es la **simplicidad** a la hora de llamar a las acciones del programa mediante el uso de una interfaz. Esto permite al programador utilizar las funcionalidades de la aplicación haciendo uso de métodos sencillos definidos en la interfaz sin prestar atención a los detalles de implementación o un largo número de parámetros, ya que de ello se encargaría la interfaz *Facade*.

Por ejemplo, como se muestra en la Figura 27, si al finalizar el script de una prueba automatizada deseamos detener la grabación en curso, llamando a una única instrucción "*DetenerGrabación*" del wrapper desde *AddFt* ejecutaría todo el trabajo necesario, sin necesidad de llamar a más funciones dependientes de esta funcionalidad. La interfaz se encargaría de buscar el proceso, enviar la instrucción a la grabadora para

que detenga la grabación, manejar las posibles excepciones que puedan ocurrir durante la ejecución (puede ocurrir que el proceso no se encontrara), etc.

Además, de cara a la **mantenibilidad** del programa, permite que, modificando únicamente los métodos en las interfaces *Facade*, se apliquen esos cambios a todas las referencias de los métodos definidos de la interfaz al resto del código de *AddFt*, evitando tener que realizar cambios en cascada o modificar todo el código de todas las partes de los programas dependientes de dicha funcionalidad, como se proyecta en la Figura 28.

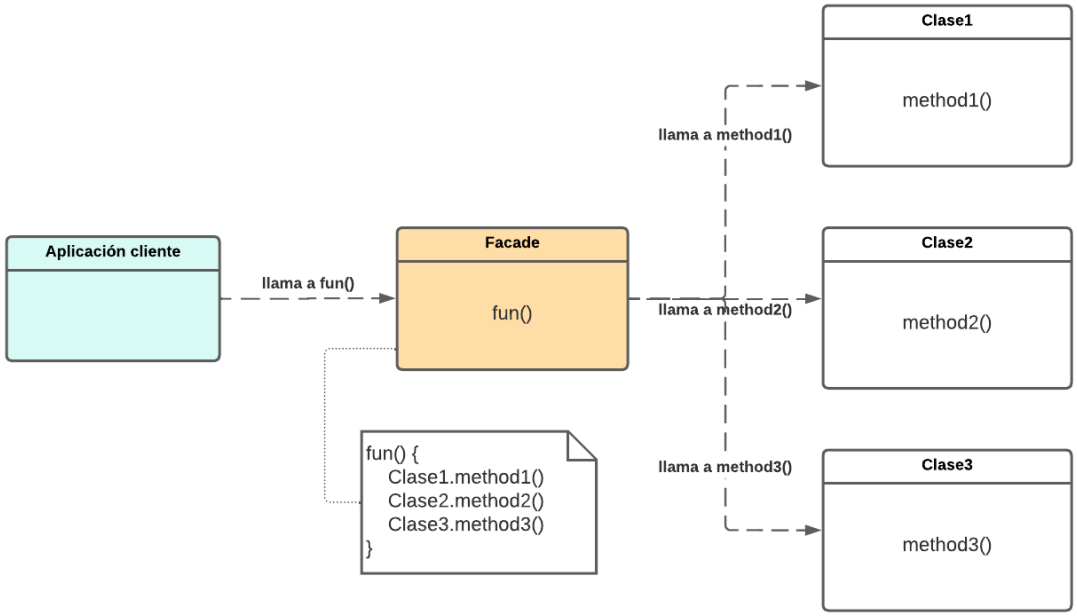


Figura 28. Esquema de ejemplo de la estructura del patrón de diseño Facade.

Sin embargo, la **desventaja** que surge al hacer uso de este patrón es, como hemos indicado anteriormente, su excesiva simplicidad. No permite al programador, por ejemplo, entrar en mucho detalle a la hora de programar un método con un número alto de parámetros. Para cumplir con ese estándar de simplicidad, se requeriría desarrollar más métodos independientes que se llamaran para personalizar la ejecución del otro método, lo que rompería por completo la esencia de este patrón y las ventajas que ofrece.

Entre las distintas opciones de lenguajes y tecnologías, la más idónea para desarrollar el *wrapper* es el *framework* .NET, mediante el lenguaje de programación C#. Las principales razones y ventajas que nos permite esta tecnología son:

- **Integración directa con el proyecto de pruebas automatizadas**: al trabajar todos los sistemas bajo .NET, no requerimos de más conectores externos para que puedan funcionar conjuntamente. Dentro del proyecto de pruebas automatizadas, las pruebas e infraestructura de lanzamiento fueron inicialmente programadas en el lenguaje Visual Basic y, posteriormente, se automatizaron nuevas pruebas bajo el lenguaje C#. Ambos lenguajes, al estar bajo el mismo *framework*, pueden funcionar conjuntamente sin realizar cambios en el código.

- **Biblioteca para la gestión, integración y ejecución de procesos en Windows**: permite lanzar ejecutables en el sistema operativo, añadiendo los argumentos y propiedades requeridos por el usuario (visibilidad de la ventana del proceso, interacción con el programa etc). Ofrece la posibilidad de interactuar desde el programa con el proceso ejecutado, automatizando el uso de programas lanzados por líneas de comandos. Este punto permitiría automatizar funcionalidades de *FFmpeg* durante la ejecución de pruebas automatizadas.
- **Operabilidad con archivos y directorios**: al generar los archivos con las grabaciones de vídeo y moverlos al directorio del informe de la prueba, pueden saltar excepciones que detenga su ejecución y no finalice correctamente el algoritmo. La biblioteca de administración de ficheros y carpetas de .NET facilita mucho estas labores, requiriendo de pocos métodos para llevar a cabo las funciones (crear directorio, eliminar archivo duplicado, etc).

Proyecto *AddFt* con opción de grabaciones de vídeo

Una vez creado el módulo encargado de utilizar de forma automática *FFmpeg*, se integrará en el algoritmo de ejecución de las pruebas automatizadas en *AddFt*. Como recordamos en el capítulo de infraestructura, cada usuario dispone de un proyecto personal en *Visual Basic* para lanzar de forma local pruebas automatizadas junto a los parámetros establecidos. En el ejemplo de la Figura 8, existían dos parámetros: identificador de la prueba y modo de ejecución. En la realidad, por cómo se diseñó originalmente el proyecto, no es necesario introducir tan solo esos argumentos para lanzar una prueba, también existe la posibilidad de introducir más variables de la ejecución bajo el mismo nombre del método.

Esto es posible gracias al uso del patrón de diseño *Command*, el cual permite que los programadores no tengan que introducir toda la información requerida para la ejecución de una prueba, como se muestra en el esquema de la Figura 29. En combinación con la sobrecarga de nombres de métodos, se dispone a los usuarios versiones reducidas de los métodos, en la cual se establecen un número reducido de parámetros y, aquellos que no se incluyeron, son introducidos por defecto dentro del método *Command*. La función de dichas funciones no es más que de lanzadera para el método general, el cual pone en marcha el algoritmo de las pruebas de una forma más sencilla y fácil de mantener que escribir siempre el método al completo en el código.

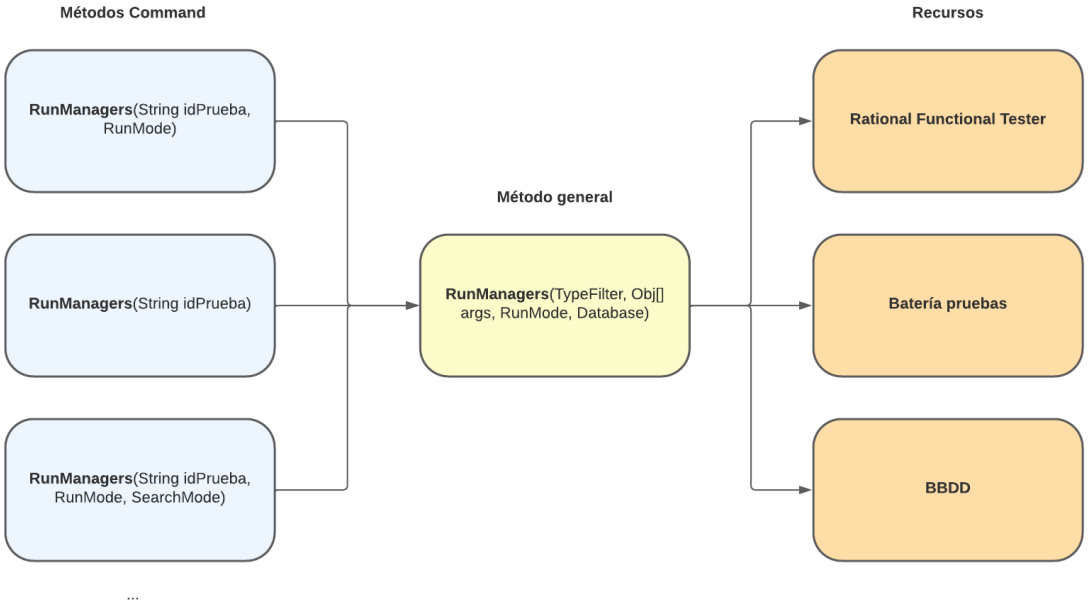


Figura 29. Esquema del patrón Command en el proyecto AddFt para lanzar pruebas automatizadas.

Al igual que un usuario quiere establecer un modo de ejecución u otros parámetros, ocurre lo mismo con las grabaciones de vídeo. Para momentos oportunos, es posible que el programador desee deshabilitar la grabación en el lanzamiento de una prueba.

Para lograrlo, se ha ideado la introducción de un nuevo argumento opcional en el método general el cual permita establecer la grabación de vídeo durante la ejecución mediante un enumerador llamado *VideoRecording*. Con ello, no requeriremos cambiar todos los métodos *Command* previamente establecidos. Además, crearemos otras dos funciones *Command*, como se observa en la Figura 30, que permita al programador habilitar la grabación de prueba en vídeo y, en caso de no introducir dicho parámetro, por defecto se encontrará deshabilitado.

Durante las prácticas de empresa, los parámetros que más veces se modificaron en el proyecto fueron el identificador de la prueba a ejecutar y, menos frecuentemente, el modo de ejecución. Por ello, se ha considerado oportuna la inclusión de un método en el que establezcan ambos parámetros junto a la grabación de vídeo y, como novedad respecto a otros métodos, la creación de una función la cual solo se establezca el nombre de la prueba y la opción de grabar la prueba, ofreciendo mayor simplicidad en el lanzamiento de pruebas dentro del proyecto *AddFt* y potenciando más el principal objetivo que tiene *Command* como patrón de diseño.

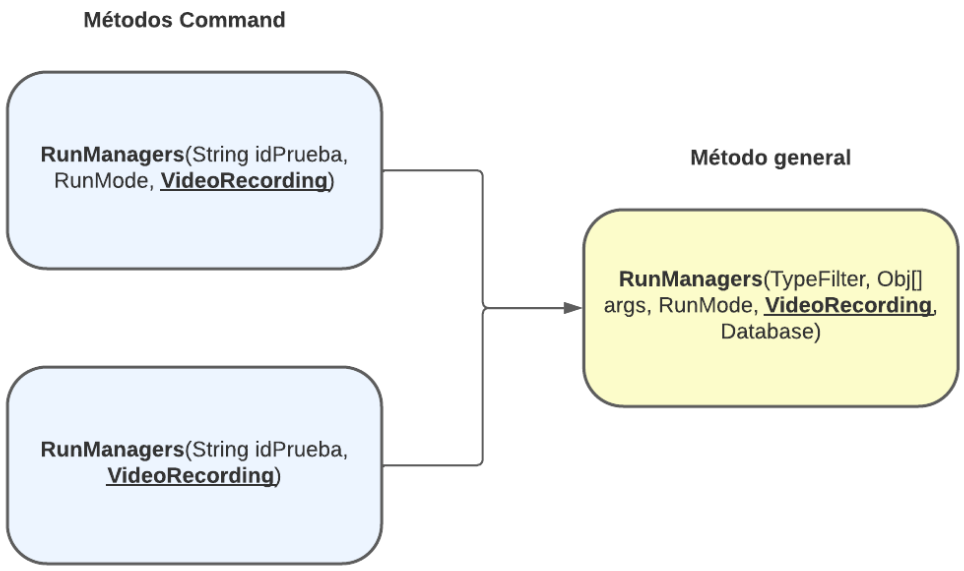


Figura 30. Esquema de los nuevos métodos Command y el método general con la inclusión de la grabación de las pruebas en vídeo.

Con el uso del enumerador en el parámetro de la grabación de vídeo durante la prueba, nos permitirá en un trabajo futuro introducir en el proyecto perfiles de grabación dentro de la clase, el cual establezcan distintas configuraciones de calidad, extensiones de salida del archivo de vídeo u otras opciones deseables.

Generación de *logs* de pruebas con reproductor de vídeo

Al terminar la ejecución de las pruebas, es interesante tener lo más accesible posible el archivo de vídeo generado. Como vimos en el capítulo de infraestructura, el *log* de la prueba inserta diversos archivos multimedia que enriquecen la información de la prueba, como son capturas de pantalla o informes de comparación de archivos. Sería muy deseable insertar el archivo de vídeo al informe generado y pudiese reproducirse directamente desde este, sin necesidad de estar accediendo a otros directorios o fuentes externas de información para su visualización.

La opción más acertada sería insertar el vídeo justo en la parte superior del informe, debajo del título de la prueba. Esto permitiría a los usuarios acceder al archivo de primera mano sin necesidad de deslizar por el archivo.

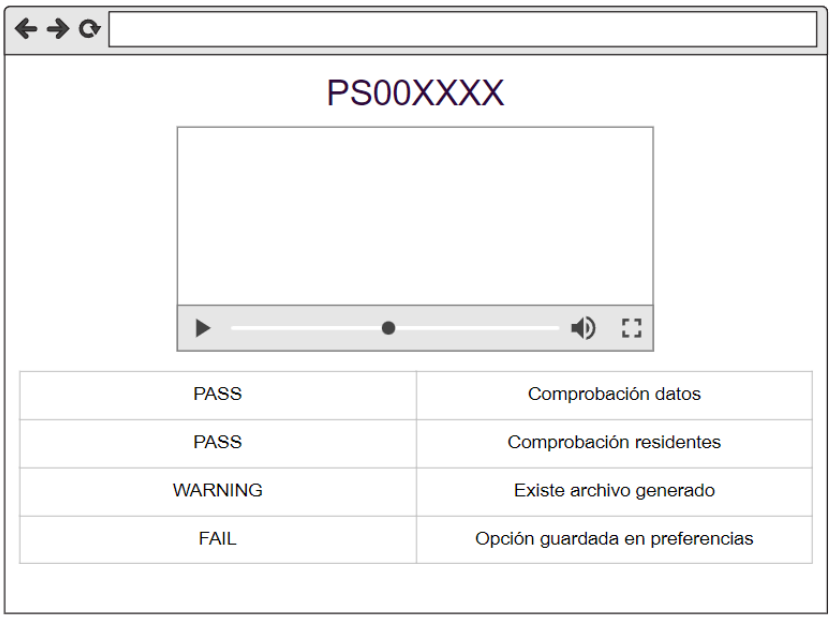


Figura 31. Prototipo de log generado en una prueba automatizada con el reproductor de vídeo con su ejecución.

La librería de generación de informes de *Rational Functional Testing* no dispone de ningún método para poder insertar vídeos o editar el formato de los logs de las pruebas automatizadas. Para ello, se ha decidido programar un algoritmo externo a RFT el cual, al finalizar la ejecución de una prueba automatizada grabada, se encargue de insertar el reproductor de vídeo en código HTML. Todo esto se realizará apoyándose de la tecnología .NET, la cual ofrece funcionalidades para poder editar el contenido de ficheros de texto e insertar las etiquetas necesarias para añadir el reproductor de vídeo dentro del informe.

Como vimos en la Figura 31, el lugar idóneo para insertar el reproductor es justo entre el título y el comienzo de la tabla con las comprobaciones realizadas por la prueba. Por ello, en el algoritmo, sería conveniente buscar de la etiqueta en la cual da comienzo la tabla y, una vez localizada como se muestra en la línea de código resaltada en la Figura 32, escribir antes de su comienzo el código del reproductor nativo de HTML vinculado al vídeo de la ejecución de la prueba.

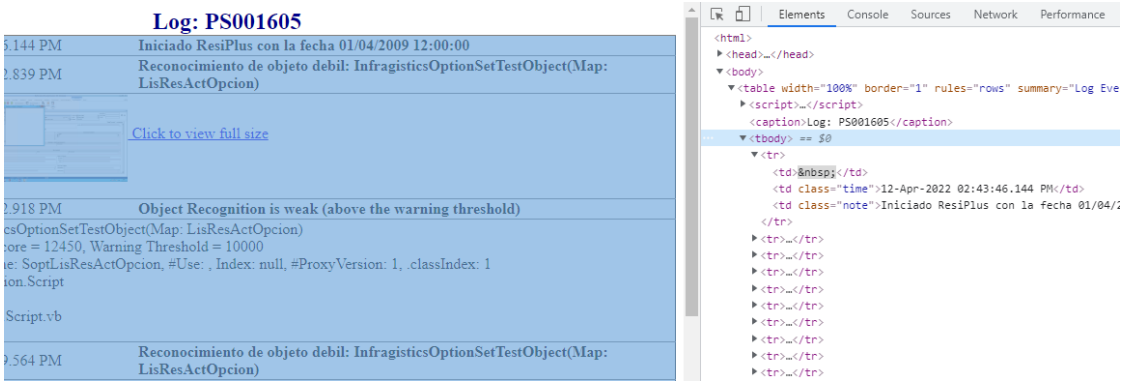


Figura 32. Código fuente de un informe generado por la ejecución de una prueba automatizada.

Lanzamiento de pruebas en ATUN con opción de grabación de vídeo

Al lanzar una batería de pruebas automatizadas de forma remota, las pruebas que fallaron se encolan de nuevo en las máquinas virtuales, creándose una nueva batería incluyendo únicamente estas pruebas. Durante la fase de análisis se estableció que la forma óptima de combinar la grabación de vídeo y la ejecución en remoto de pruebas automatizadas es habilitar *FFmpeg* únicamente en la batería de las pruebas que fallaron. Esto es debido a que debemos optimizar el almacenamiento de los archivos de vídeo ya que suponen un espacio en los sistemas de información de la empresa y debemos evitar saturarlos con grandes cantidades de archivos.

Como vimos en el capítulo de infraestructura, en el panel principal del programa ATUN existe un apartado donde podemos establecer la configuración de las baterías lanzadas y de sus parámetros, como son la localización del proyecto *AddFt* vinculado o el modo de ejecución de las pruebas en remoto. Internamente, la configuración de la aplicación es persistida en un archivo XML con los valores del formulario de Configuración. De esta forma, nunca se pierden los ajustes de la herramienta y, en el momento de encolar una batería en las bases de datos, poder consultar en el fichero la configuración de la batería que debe aplicar, como se muestra en la Figura 33.

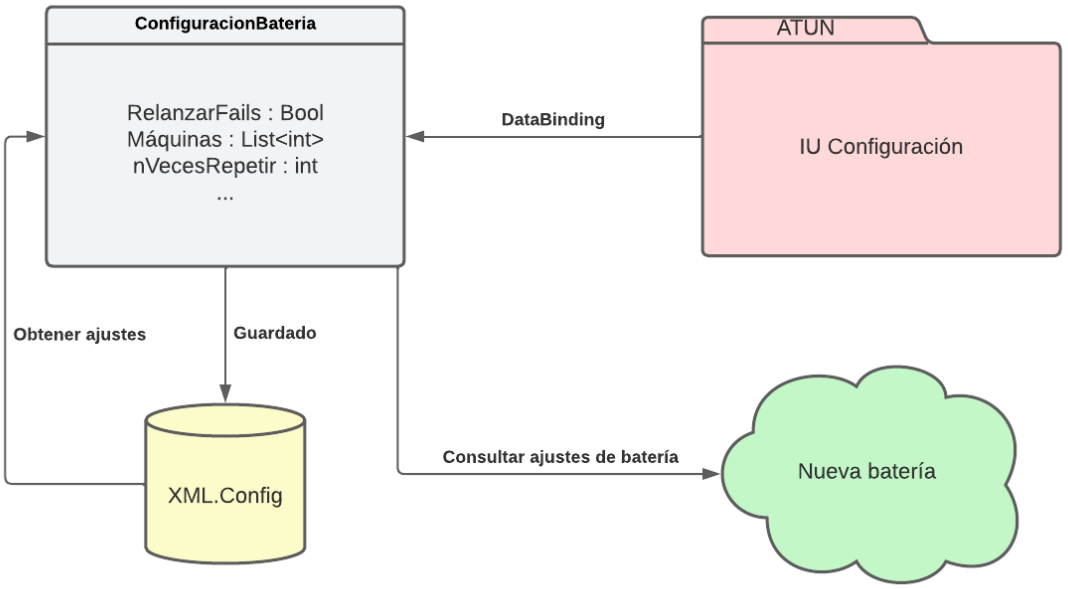


Figura 33. Esquema de funcionamiento actual de la Configuración en ATUN.

De esta forma, para añadir soporte a la grabación de vídeo, se añadiría un atributo extra que establezca si usar la grabadora durante el relanzamiento de las pruebas falladas y esté conectado mediante un *DataBinding* a un *checkbox* adicional en el formulario de configuración para poder editar los parámetros mediante la interfaz gráfica por parte del usuario y este quede persistente en el sistema, asimilándose al prototipo de la Figura 34.

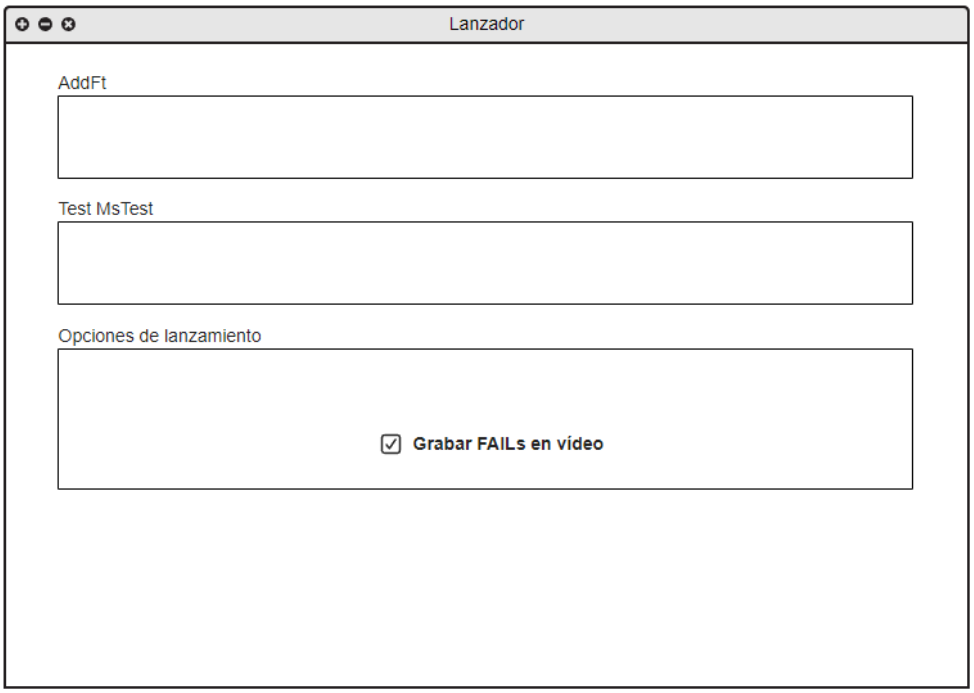


Figura 34. Mockup del checkbox adicional en la ventana de configuración de ATUN.

Al encolarse pruebas en ATUN estas se incluyen en una batería nueva junto a unos parámetros establecidos en el formulario de Configuración de ATUN. Entre dichos ajustes, puede establecerse si las pruebas que fallaron deben volver a ejecutarse, el número de veces y las máquinas destinadas a su ejecución.

Cuando se vuelven a encolar las pruebas que fallaron en ATUN, estas son incluidas en una instancia de batería en la base de datos con unos parámetros distintos (por ejemplo, pruebas incluidas y la batería padre de la que proviene el relanzamiento de las pruebas). Por ello, sería ideal añadir un nuevo atributo que establezca si las pruebas a las que pertenecen a la batería deben ser grabadas en vídeo.

Al crear una batería con las pruebas que fallaron, se usa un método distinto para lanzar la nueva batería, pero siempre haciendo uso de la función general que crea la batería con todos los parámetros, aplicando una funcionalidad similar al patrón de diseño *Command*. Por ello, al crear una batería desde cero, en el caso de relanzarse las pruebas fallidas, el campo de grabación de vídeo estará desactivado. En el momento de encolar la nueva batería de FAILs, se llamaría de nuevo al método que crea la batería, estableciendo esta vez el valor que le corresponda según la configuración de ATUN la habilitación de la grabación de vídeo de la batería, como muestra el esquema de la Figura 35.

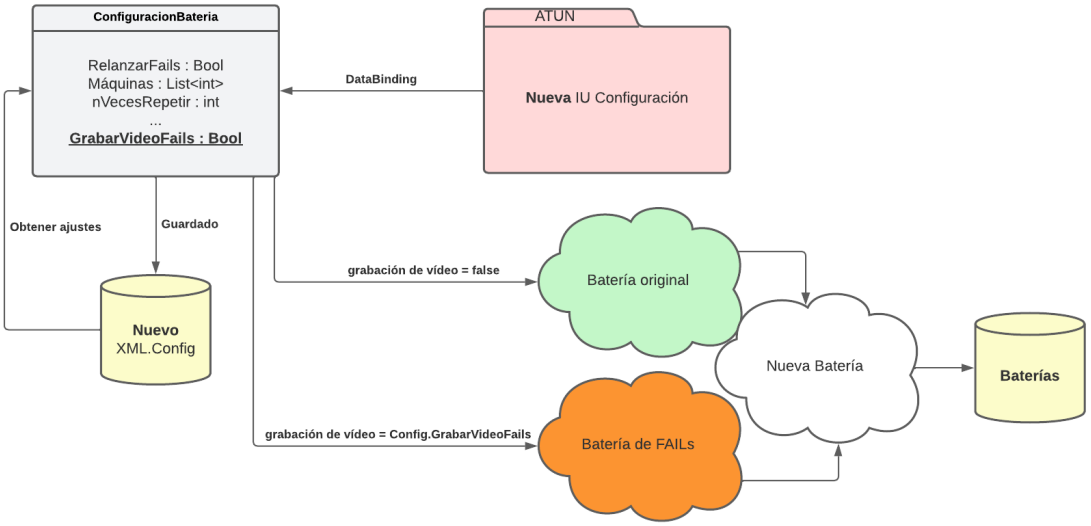


Figura 35. Esquema del nuevo lanzamiento de baterías en ATUN.

Por último, en las máquinas virtuales de la infraestructura de ATUN ejecutan una herramienta llamada *LanzadorCliente*. Dicho programa está en segundo plano consultando en la base de datos de ATUN si llegan nuevas pruebas disponibles de baterías para ser ejecutadas en el entorno y mandar los informes generados al sistema de información compartido. Para ejecutar la prueba, el lanzador lee los atributos de la batería en la que forma parte la prueba y lanza el proceso de *AddFt* con los parámetros establecidos en su configuración.

Haciendo uso de los cambios aplicados en los atributos de las baterías, para determinar si una prueba debe grabarse desde la máquina virtual, el algoritmo puede obtener el valor de grabación de vídeo de la batería de la prueba y, al momento de ejecutar el proceso de *AddFt* de la máquina virtual, se llamará al método lanzador de pruebas incluyendo el parámetro obtenido de grabación de vídeo. Para ello, se editará el proyecto *AddFt* para que, desde *LanzadorCliente*, pueda establecer también las grabaciones de vídeo en las ejecuciones de las pruebas, como muestra la Figura 36.

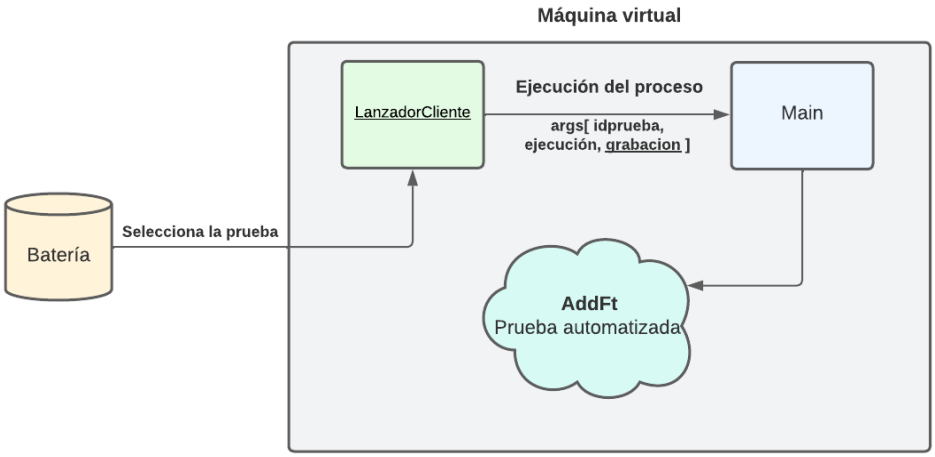


Figura 36. Esquema del nuevo funcionamiento del LanzadorCliente en la ejecución de pruebas junto al proyecto AddFt.

6.3 Programación

Muchos desarrolladores tienden a menospreciar la labor de diseño y saltan directamente a escribir código de forma descontrolada sin realizar una previa documentación o visión general por el producto programado. A esta práctica se le conoce en la Ingeniería de Software como *Code and Fix*. Para realizar scripts o programas de líneas de código muy escasas podría ser una metodología para tener en cuenta, debido a que la documentación en estos casos podría omitirse. Sin embargo, en un proyecto de esta magnitud es importante tener una disciplina de trabajo y realizar paso por paso una documentación y tener una estructura clara de lo que se quiere desarrollar con un cierto control y retrospectiva del estado del programa durante la escritura del código.

El objetivo de esta fase del proyecto es programar las nuevas funcionalidades especificadas durante la fase de análisis, manteniendo la arquitectura y características del código en la etapa de diseño y teniendo en cuenta las limitaciones del lenguaje y las tecnologías utilizadas.

Proyecto *AddFtScreenRecorder* (grabadora de pantalla)

Durante la etapa de diseño mencionamos la necesidad de conectar y automatizar *FFmpeg* con la ejecución de las pruebas en *AddFt*. El proyecto encargado de realizar estas funciones y administrar las grabaciones generadas durante la ejecución de una prueba automatizada ha sido nombrado como *AddFtScreenRecorder*.

Aplicando el patrón *Facade*, se han programado métodos estáticos públicos que permiten el uso de las funciones desde cualquier proyecto externo, como muestran las cabeceras en la Figura 37. Entre las funciones disponibles que se encuentran son:

- **Comenzar grabación**: antes de lanzar *FFmpeg*, borra la última grabación realizada y busca si el programa *FFmpeg* está instalado en la máquina. En caso contrario, se descarga de forma automática desde el servidor de la intranet empresarial. Una vez realizadas estas comprobaciones, pone en marcha el proceso de *FFmpeg* con los argumentos necesarios para grabar el escritorio en vídeo y guarda la referencia del proceso en el atributo estático privado *FFMPEGprocess* dentro de la clase del proyecto.
- **Cerrar procesos de la grabadora en segundo plano**: se encarga de buscar los procesos abiertos de *FFmpeg* en la máquina y abortarlos. De esta forma, en la máquina donde se aloja el proyecto, libera recursos de memoria en el caso de que un proceso se quedase abierto tras la ejecución de alguna prueba.
- **Terminar grabación**: obtiene el proceso de *FFmpeg* desde *FFMPEGprocess* y detiene su grabación de forma segura, guardando el archivo de vídeo en la ruta introducida en su parámetro. Antes de ser ejecutado, comprueba que el atributo estático del proceso *FFmpeg* no es nulo para evitar excepciones y que los programadores solicitan finalizar grabaciones antes de iniciarlas. Al finalizar con el proceso, dicho atributo volverá a nulo para evitar inconsistencias en el sistema. A diferencia del método anterior, el proceso es cerrado de forma segura interaccionando con el programa solicitando que detenga la grabación y se

genere el vídeo de forma correcta, en lugar de cerrarlo forzosamente y corromper el archivo con la grabación.

```
public class FFmpegWrapper
{
    public static Process FFMPEGprocess;
    private static string RecordingFileName = "playback.mp4";
    private static string FFmpegDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + "\\ffmpeg.exe";
    private static string DownloadFFmpegPath = "\\\\ADDHQRDRSRFT005\\EjecucionPruebas\\FFmpeg\\ffmpeg.exe";
    2 references
    public static void StartRecording()...

    1 reference
    public static void EndRecording(string savePath)...

    1 reference
    public static void KillPreviousBackgroundProcess()...
```

Figura 37. Métodos estáticos públicos de AddFtScreenRecorder para la grabación en las pruebas automatizadas.

Ejecución de pruebas en *AddFt* con grabación de vídeo

Como mencionamos en la subsección anterior, cada usuario dispone de un proyecto personal desde el cual llamar a un método *Command* lanzadera de pruebas automatizadas y establecer mediante sus parámetros la prueba a ejecutar, su base de datos, etc.

Para el método lanzador general, hemos añadido un nuevo parámetro opcional, el cual un usuario puede establecer mediante una clase enumerador *VideoRecording* el modo de grabación de la prueba, y hemos añadido los dos métodos *Command* para simplificar el lanzamiento de pruebas, como muestra la Figura 38. Por defecto, si dicho argumento no está especificado en el método, la grabación de vídeo estará desactivada.

En estos momentos solo disponemos de dos perfiles en la clase enumerador: grabación activada y desactivada. Una alternativa a la utilización de este método habría sido mediante un valor booleano dentro del método dicho parámetro. Sin embargo, de cara a un mantenimiento futuro, el uso de la clase enumerador permitirá crear nuevos perfiles de grabación y dar configuraciones más flexibles al usuario, como especificamos en la fase de diseño.

```
Public Class Program
    Implements IAction
    0 references
    Public Shared Sub Main(ByVal args As String())
        Launcher.LaunchRft(New Program())
    End Sub

    5 references
    Public Sub Run() Implements IAction.Run
        Manager.RunManagers("PS006423", Manager.RunMode.All, Manager.VideoRecording.Enabled)
    End Sub
End Class
```

Figura 38. Método de AddFt para el lanzamiento de pruebas automatizadas con la grabación en vídeo habilitada.

Dentro del algoritmo interno de lanzamiento de pruebas hemos añadido nuevas ramas de ejecución en función del parámetro de la grabación de vídeo. Al comenzar con el lanzamiento de una nueva prueba, siempre se cerrarán procesos en segundo plano que quedaron de *FFmpeg* de una ejecución anterior. Si la grabación de vídeo está habilitada, se llamará al método del *wrapper* que comience con la grabación y

ejecutará un nuevo proceso de la grabadora, como se observa en el código de la Figura 39.

```
Public Sub Run(ByVal mode As RunMode, Optional ByVal videoRecording As VideoRecording = VideoRecording.Disabled,

    Dim initialized As Boolean = False
    Dim overridesPrepareData As Boolean = False
    _runmode = mode

    FFmpegWrapper.KillPreviousBackgroundProcess()
    If videoRecording = VideoRecording.Enabled Then
        FFmpegWrapper.StartRecording()
    End If
```

Figura 39. Algoritmo de lanzamiento de una prueba automatizada con grabaciones de vídeo en AddFt.

Al finalizar la ejecución de una prueba, como muestra la Figura 40, si la grabación en vídeo estaba habilitada, usa el método del *wrapper* para finalizar la grabación indicando la ruta donde se aloja el informe de la prueba en el parámetro con la información obtenida de RFT. El *wrapper* se encargaría de copiar el archivo de vídeo en dicho directorio junto al resto de los ficheros generados. Además, como explicaremos en la siguiente subsección, enviará la información del directorio del log a la clase *HTMLEditor* del proyecto *AddFtScreenRecorder*, el cual reescribirá el código HTML del resultado de la prueba.

```
        FreeOwnedObjects()

        _watch.Stop()

        If videoRecording = VideoRecording.Enabled Then
            FFmpegWrapper.EndRecording(LogManager.Instance.GetLogDirectory())
            HTMLEditor.setLogPath(LogManager.Instance.GetLogDirectory() + "\\rational_ft_logframe.html")
        End If

        Log.Information("Tiempo de ejecucion de la prueba: " + Duration.ToString())
        Log.ManagerEnd(Me) ' Throws Nothing
        _current = Nothing

        Profiler.GlobalProfiler.SumUpResults(Me)

    End Try
End Sub
```

Figura 40. Final de la ejecución de una prueba automatizada con la detención de la grabación en vídeo.

Reescritura del *log* para la inclusión del reproductor de vídeo

En HTML se da la posibilidad de incluir reproductores de vídeo de forma nativa sin tener que añadir extensiones o componentes externos. Por ello, al no tener acceso al código fuente de la tecnología *RFT* y poder modificar el comportamiento de generación de los informes de las pruebas, se procederá a realizar una reescritura del código fuente del resultado una vez haya finalizado la ejecución de la prueba.

En el proyecto *AddFtScreenRecorder* se creó una clase adicional llamada *HTMLEditor* encargada de realizar las tareas de edición en los *logs* de las pruebas. Dicha clase contiene dos métodos estáticos públicos:

- **Establecer ruta del log** de la prueba a editar: método setter que establece la ruta del fichero del log en un atributo estático privado de la clase. Se ha programado este método para agregar la información necesaria de la ruta del archivo antes de cerrar de forma definitiva la edición del *log* en RFT. Durante su

escritura, RFT contiene la información de la ruta del archivo y, al finalizar la ejecución de la prueba automatizada, esta información ya no es accesible y no es posible editar el fichero mientras el editor de RFT siga en marcha. Por ello, primero se guarda la información en el atributo estático y posteriormente se accede a dicha información en el otro método para saber qué fichero editar.

- **Edición del archivo** (*SetVideoPlayerInHTMLLog*): función encargada de reescribir el *log* de la prueba HTML para incluir en el lado inferior del título un reproductor de vídeo del archivo de la grabación. Al comienzo del algoritmo, como se muestra en la Figura 41 para evitar excepciones o inconsistencias del sistema, se comprueba desde el atributo estático que contiene la ruta del informe que realmente existe. Si se cumple procede a buscar la línea en la cual se encuentra la primera fila de la tabla HTML por su etiqueta *"<tr>"*. Al encontrarla, se inyecta en la línea anterior del archivo el código de la grabadora de vídeo y guarda los cambios realizados.

```
public static void SetVideoPlayerInHTMLLog()
{
    if(File.Exists(LOG_PATH))
    {
        string[] fileLines = File.ReadAllLines(LOG_PATH);
        int lineToEdit = -1;

        for(int i = 0; i < fileLines.Length; i++)
        {
            if (fileLines[i] == "<TR>" || fileLines[i] == "<tr>")
                lineToEdit = i - 1;
        }

        if (lineToEdit != -1)
            fileLines[lineToEdit] += SCRIPT_VIDEO;

        File.Delete(LOG_PATH);
        File.WriteAllLines(LOG_PATH, fileLines);
    }
}
```

Figura 41. Algoritmo de reescritura del archivo del log de la prueba automatizada en la clase HTMLEditor de AddFtScreenRecorder.

Tras ejecutar una prueba automatizada con la grabación en vídeo habilitada, estará a disposición del usuario el archivo con la ejecución completa de la prueba en el directorio de su informe y, desde el propio *log*, podrá ser reproducido desde reproductor nativo de HTML, como muestra la Figura 42. Esto permite una mejor usabilidad y dará mayor accesibilidad al vídeo, evitando así tener que acceder a distintas fuentes de datos para poder reproducirlo.

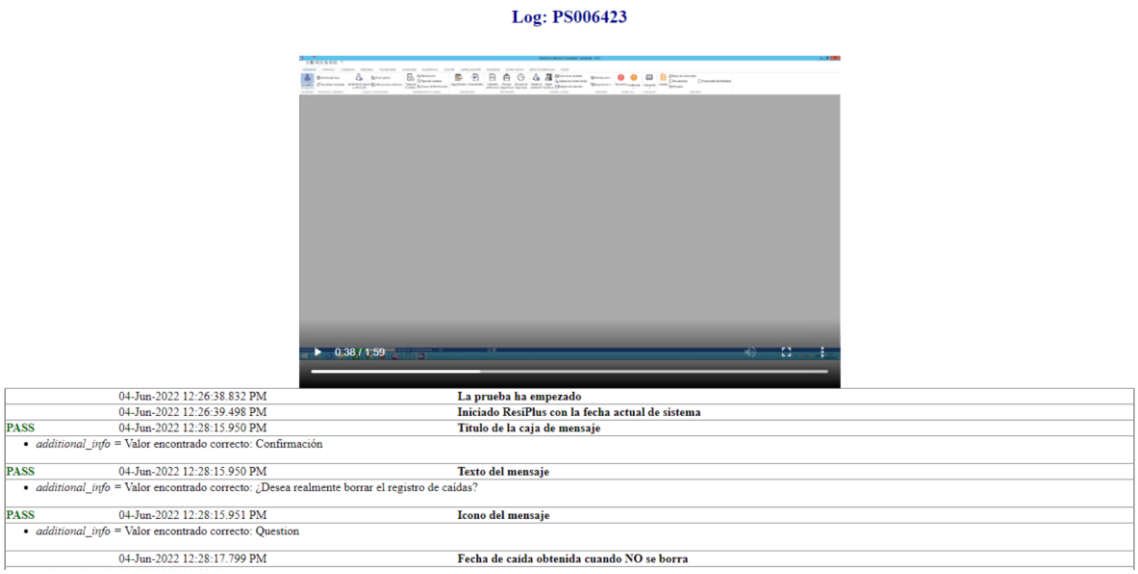


Figura 42. Nuevo log generado por una prueba automatizada con la grabación de vídeo habilitada.

Lanzamiento de pruebas automatizadas grabadas en vídeo en remoto mediante ATUN

Para la inclusión de baterías con grabaciones de vídeo en el programa ATUN se han realizado cambios tanto en el *front-end* de la aplicación como en su lógica de negocio en el *back-end*.

En el componente ***front-end*** del sistema es donde menos cambios se han producido, como podemos comparar entre las Figuras 9 y 43. Aplicando el prototipo realizado durante la fase de diseño, se ha añadido un *checkbox* más en la ventana de opciones. En la región de opciones de lanzamiento se ha puesto a disposición del usuario la posibilidad de elegir si quiere que se graben las pruebas que fallaron en vídeo o se deshabilitan de forma completa. Para el nuevo *checkbox* creado, se ha vinculado mediante un *DataBinding* a un nuevo atributo del archivo XML de configuración de la aplicación.

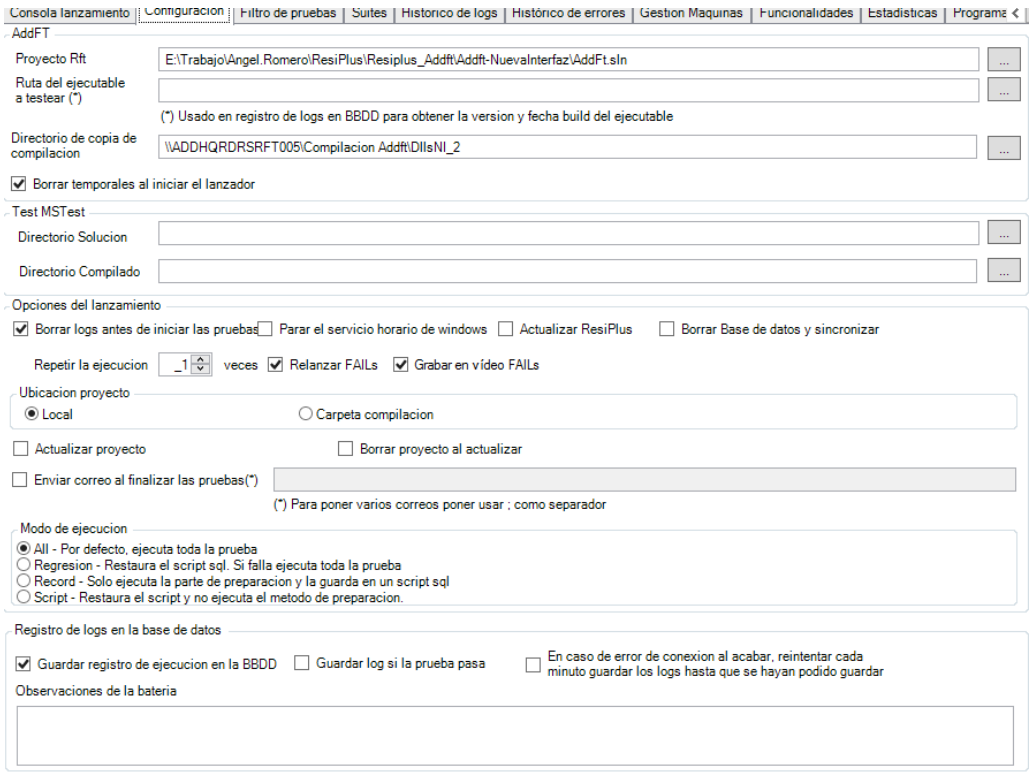


Figura 43. Nuevo formulario de configuración de ATUN con el checkbox de grabación de baterías.

En el lado del ***back-end*** es el componente donde más modificaciones se han realizado. Comenzando por los ajustes, se ha añadido una nueva clave para almacenar en el XML el nuevo *checkbox* introducido en la Figura 43. Como recordamos, en dichos valores se almacena la configuración de la herramienta y ser accedida y persistida por todos los módulos del proyecto. Dicha nueva clave permitirá a los componentes de ATUN consultar si debemos habilitar la grabación en la batería lanzada según su valor GrabarVideoFails en la Figura 44.

```
    <add key="ModoEjecucionScripts" value="0" />
    <add key="ModoSeleccionPruebas" value="" />
    <add key="IdSuiteSeleccionada" value="" />
    <add key="BorrarTemporalesAlAbrir" value="True" />
    <add key="DirectorioSolucionMSTest" value="" />
    <add key="DirectorioDllsMSTest" value="" />
    <add key="RelanzarFAILs" value="True" />
    <add key="GrabarVideoFAILs" value="True"/>
    <add key="ClientSettingsProvider.ServiceUri" value="" />
</appSettings>
```

Figura 44. Nueva clave de valor de la configuración de ATUN para la grabación de baterías.

En el lanzamiento de nuevas baterías de pruebas se utilizan dos clases en ATUN: ***ConfiguracionBateria*** y ***Batería***. La clase ***ConfiguraciónBateria*** es la encargada de establecer qué ajustes contienen cada una de las baterías creadas. Dichos parámetros están enlazados con las claves del fichero XML de configuración previamente mostrados. Al lanzarse una nueva batería se consulta con *ConfiguraciónBatería* todos los parámetros solicitados para su ejecución, que a su vez consulta con el fichero XML de los valores establecidos en la aplicación. Se ha añadido el atributo “grabarVideo” a la clase para que almacene el valor de la grabación de vídeo en la configuración de ATUN, como se muestra en la Figura 45.

```
public class ConfiguracionBateria : AtunUtils.XmlObject
{
    private bool _actualizarResiPlus;
    private bool _enviarCorreo;
    private string _correos;
    private bool _guardarLogSiPruebaPasa;
    private AtunEntities.ModoEjecucionPrueba _modoEjecucion;
    private bool _pararServicioHorario;
    private bool _apagarMaquina;
    private bool _inicializarBDsResiplus;
    private bool _relanzarFails;
    private int _nVecesRepetir;
    private bool _grabarVideo;
    private List<int> _maquinas;
```

Figura 45. Clase ConfiguracionBateria con el nuevo atributo de grabación de vídeo enlazado a los ajustes de ATUN.

Las instancias de la clase ***Batería*** son almacenados en la base de datos y consultados por *LanzadorCliente* para ejecutar las pruebas en las máquinas virtuales. Se le ha añadido otro atributo booleano que establezca si la batería debe ser grabada. De esta forma podemos establecer que la primera ejecución no se grabe poniendo su atributo en falso y, en las ejecuciones de las pruebas que fallaron, establecer el valor en función de *ConfiguraciónBatería*.

Además del propio constructor de la clase Batería, existen unos métodos estáticos públicos *Command*, como se observa en la Figura 46, en los cuales pueden crear una instancia de Batería y guardarlos en la base de datos con pocos o ningún argumento. En esos métodos constructores, se ha añadido el parámetro y atributo booleano de grabación de vídeo. De esta forma, se establece en la batería original que no se graben las pruebas y dicho valor cambie cuando se cree el objeto de la Batería con las pruebas que fallaron. Además, con el nuevo atributo en la clase Batería, se ha modificado también la relación *Batería* en la base de datos y ser leído por *LanzadorCliente* al lanzar el proceso de *AddFt* en la máquina virtual.

```
public Bateria CrearBateriaFAILs()
{
    using (DataClassesLogsDataContext logs = new DataClassesLogsDataContext())
    {
        List<string> pruebasFAIL = this.GetPruebasFallidas();
        Bateria bNueva;
        using (DataClassesLogsDataContext db = new DataClassesLogsDataContext())
        {
            ConfiguracionBateria c = this.GetConfiguracion();
            bool relanzar = false;
            if (c.NVecesRepetir - 1 > 0)
                relanzar = true;
            ConfiguracionBateria cNueva = new ConfiguracionBateria(false, c.EnviarCorreo, c.Correos, c.GuardarLogSiPrue
            bNueva = CrearBateria(pruebasFAIL.Count, this.Version, this.IDMaquina, this.IDPrograma, this.FechaBuild,
                "Pruebas Fallidas de la bateria: " + this.IDBateria, this.Observaciones, this.EnEjecucion, cNueva,
                cNueva.ModoEjecucion, _IdSuite, this.IDBateria, this.EjecutarTodasMaquinas, this.GrabarVideoBateria);
            db.Baterias.InsertOnSubmit(bNueva);
            db.SubmitChanges();
        }
```

Figura 46. Método Command constructor de la batería de FAILs con grabación de vídeo.

En último lugar, se ha procedido a modificar el módulo de *LanzadorCliente* y poder lanzar el proceso *AddFt* con el parámetro de grabación de vídeo. Al asignarse una prueba a la máquina virtual, obtiene los datos de la configuración de su batería almacenados en la base de datos. Una vez leídos, lanza el proyecto *Main* del proyecto *AddFt* alojado en la máquina y se le asignan los argumentos de su lanzamiento en función de la configuración de la batería. Los parámetros del proyecto *Main* son idénticos a los proyectos personales de los usuarios de *AddFt,* adaptados al lanzamiento

de pruebas desde LanzadorCliente. Desde LanzadorCliente puede establecerse la prueba a ejecutar, el modo de ejecución y, de forma novedosa, se ha añadido el parámetro de grabación de vídeo de la ejecución, como muestra la Figura 47.

```
public void EjercutarScript(string codigo, AtunEntities.ModoEjecucionPrueba modo, bool grabarVideo)
{
    if (!_proyecto.Exists)
        throw new ArgumentException("No existe el proyecto " + _proyecto.FullName);

    FileInfo fiMain = new FileInfo(DirectorioSolucion + @"\Main\Main.exe");
    ProcessStartInfo psi = new ProcessStartInfo(fiMain.FullName,
        "\"" + codigo + "\"" + " " + "\"" + grabarVideo + "\"" + (modo != AtunEntities.ModoEjecucionPrueba.NoDefinido ? (" " + (int)modo) : ""));
    psi.WorkingDirectory = fiMain.Directory.FullName;
    psi.UseShellExecute = false;
    psi.WindowStyle = ProcessWindowStyle.Hidden;
    psi.RedirectStandardError = true;
    Process pr = Process.Start(psi);
    pr.WaitForExit();
    string error = pr.StandardError.ReadToEnd();
    if (!string.IsNullOrEmpty(error))
    {
        if (error.Contains(StopButtonHitException.INDICATOR))
            throw new StopButtonHitException(error);
        else
            throw new Exception(error);
    }
}
```

Figura 47. Método de LanzadorCliente para ejecutar pruebas automatizadas del proyecto AddFt en la máquina virtual desde Main.

6.4 Pruebas

Durante el transcurso del trabajo, se ha probado minuciosamente todas las piezas de *software* creadas o modificadas para añadir las nuevas funcionalidades deseadas. Para esta etapa, se ha planificado una fase de *testing* donde, de forma manual, usando la documentación generada en la fase de análisis y por etapas, se verifique y valide de forma correcta todo el producto desarrollado y se pueda utilizar en las tareas de revisión de pruebas automatizadas.

Los tramos que componen la fase de pruebas en nuestro proyecto son:

- **Pruebas controladas** durante el desarrollo: al finalizar con la fase de programación de una UT, esta es testeada de forma manual por el programador en su entorno de trabajo y consultando que se verifican y validan todas las funciones programadas en el paquete de trabajo aplicando sus **pruebas de aceptación**. Cuando pasa por todos los controles, el paquete de trabajo se da por finalizado en el tablero Kanban del proyecto.
- **Versión *Alpha*:** al finalizar un *sprint*, el equipo de trabajo pone a prueba todo el *software* en su conjunto y lanza las **pruebas de regresión** en el producto, comprobando que todas las características siguen funcionando después de los cambios aplicados al finalizar con el *sprint*. Se asegura en esta fase que todo sigue acorde a la documentación de requisitos y no existen fallos de implementación de las UTs durante el uso del sistema en el entorno del desarrollador.
- **Versión *Beta*:** al completarse la segunda fase de *testing*, se da paso a la última etapa antes de lanzar la versión completa a producción. A diferencia de las anteriores, aquí interfieren más compañeros del departamento que no pertenecen al desarrollo del proyecto. En sus propios entornos, utilizan la herramienta de forma no guiada sobre el producto. En el caso de encontrar anomalías o fallos durante su ejecución, este es notificado al equipo de desarrollo, el cual se crearía una UT para corregir el fallo detectado en un *sprint*

futuro y transcurrir de nuevo todas las etapas de pruebas hasta validarse por completo. Una vez terminada esta etapa, se publicaría la nueva versión de la infraestructura de la empresa para ser implementada en las labores de revisión y lanzamiento de pruebas automatizadas.

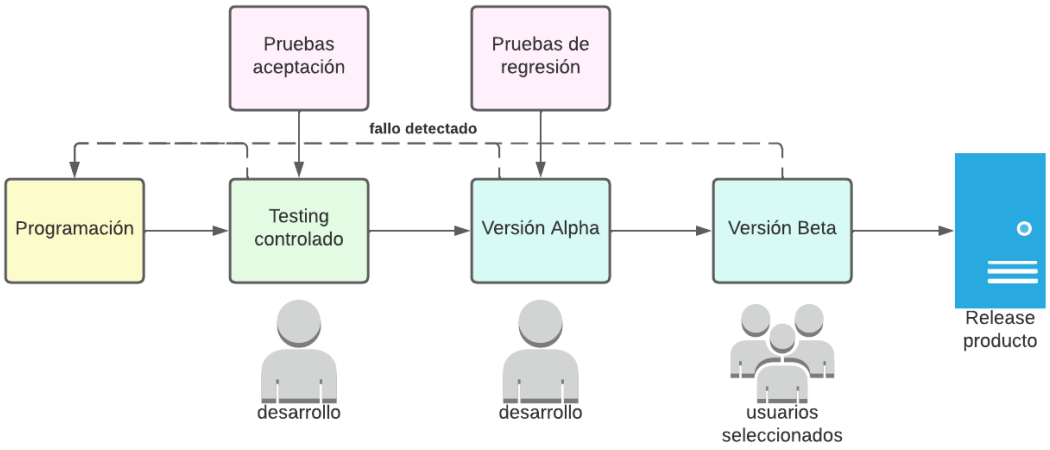


Figura 48. Esquema de las fases de pruebas durante el desarrollo del proyecto.

Dividir esta fase en varias capas de *testing* y de usuarios implicados ha permitido la detección de varios fallos tempranos en el desarrollo de nuevas funciones en las herramientas, evitando que sean propagados en las versiones publicadas del producto, como visualizamos en la Figura 48.

Por ejemplo: en el momento de lanzar el proceso de grabadora de vídeo, previamente se buscan procesos de *FFmpeg* que se hubiesen quedado en segundo plano de alguna prueba ejecutada anterior (por una mala ejecución de la prueba o una detención forzosa de la infraestructura de pruebas automatizadas). Durante la fase *Beta*, más de dos usuarios probaron a grabar de forma simultánea pruebas automatizadas desde sus entornos. Al encontrarse en una misma máquina virtual, el programa *AddFtScreenRecorder* detectaba el proceso lanzado por el otro usuario y, al no disponer de permisos para detenerlo, lanzaba una excepción y no permitía grabar a más de un usuario en una misma máquina.

Otro suceso que destaca la importancia de probar nuestras aplicaciones y, en especial, sistemas *software* mantenidos desarrollados previamente por otros programadores, es el lanzamiento de las pruebas de regresión para cualquier cambio aplicado sobre este. Cuando se cambiaron las clases Batería y Configuración en ATUN añadiendo los nuevos atributos de grabación de vídeo, al crearse una nueva batería no se guardaba correctamente la instancia en la base de datos de la infraestructura. Esto provocaba que, aunque se crease de forma adecuada el objeto en la aplicación, estos no se persistían en los sistemas de información, haciendo que nunca se llegaran a ejecutar en las máquinas virtuales de la empresa.

6.5 Metodología del proyecto

En los últimos años, se ha popularizado en el desarrollo de *software* el uso de las metodologías ágiles aplicado en proyectos tecnológicos. Se trata de una forma de

trabajo en la cual pone especial énfasis en la flexibilidad y la entrega continua de *software* al cliente, basado en el principio de MVP[20].

Esta nueva metodología se ha externalizado en los últimos años a proyectos no tecnológicos de otros ámbitos de la industria. Las principales características que conlleva el uso de estas metodologías en el desarrollo de proyectos tecnológicos son:

- **Iterativo**: se trabaja en ciclos breves de tiempo, donde se repite el proceso del desarrollo de *software* hasta alcanzar el producto final deseado.
- **Especificación de requisitos dinámica**: durante el desarrollo del producto es posible que los requerimientos del usuario cambien o se definan unos nuevos. Por ello, se evita realizar excesivos esfuerzos en documentar el producto en su totalidad y posibilita realizar las modificaciones oportunas en la etapa de requisitos al comienzo de cada sprint.
- **Entrega evolutiva**: se prioriza al final de cada iteración el envío de una versión del producto usable al cliente, cumpliendo con el concepto de MVP previamente expuesto. Nos permite recibir una retroalimentación temprana del cliente y realizar los cambios oportunos en el producto lo más temprano posible, siendo menos costosos de implementar que realizarlos sobre el producto final una vez terminado de desarrollar al completo.

Además, los principales beneficios que conllevan su utilización respecto al uso de una metodología tradicional son:

- **Amoldamiento**: ante la incertidumbre y cambios de contexto durante el desarrollo de un proyecto, la postura ágil es más abierta y está preparada para afrontar y asimilar los cambios requeridos en los requisitos del usuario en el producto. Si, por lo contrario, se aplicara una metodología secuencial (por ejemplo, *waterfall*[21]), y se requiriese de una nueva funcionalidad, la naturaleza del propio proceso no lo permitiría, produciendo un resultado de menor calidad y que no satisface las necesidades del usuario
- **Retroalimentación del proyecto**: al finalizar cada iteración del proyecto, permite al equipo de desarrollo y al cliente recibir información más precisa del estado actual del proyecto y de los resultados obtenidos. En cambio, en una metodología secuencial, hasta el final del proyecto no se puede obtener un *feedback* del resultado obtenido del cliente, lo que supondría un problema si el producto recibido no es el que se esperaba o requiriese de cambios parciales o totales en la implementación.
- **Mayor calidad y probabilidades de éxito**: se tiende a confundir que el uso de estas metodologías obtiene resultados de menor calidad. Es totalmente erróneo. La filosofía ágil en el desarrollo de proyectos permite resultados y un desarrollo mucho más optimizado y con calidades superiores al de otras metodologías.

---

[20] ***MVP**: [18] siglas cuyo significado es Producto Mínimo Viable (en inglés Minimum Viable Product). Hace referencia al desarrollo de prototipos o versiones reducidas de un producto usable sin dejar funcionalidades sin implementar y así recibir un feedback más temprano de los clientes invirtiendo menor esfuerzo.*

[21] ***Metodología en cascada** o **waterfall** – Más información: https://es.wikipedia.org/wiki/Desarrollo_en_cascada*

- **Ideal para proyectos de menor y medio tamaño**: el uso de metodologías ágiles en el mundo del desarrollo de *software* es ideal para equipos de desarrollo de menor tamaño, el cual es ideal para el desarrollo del proyecto de este trabajo.

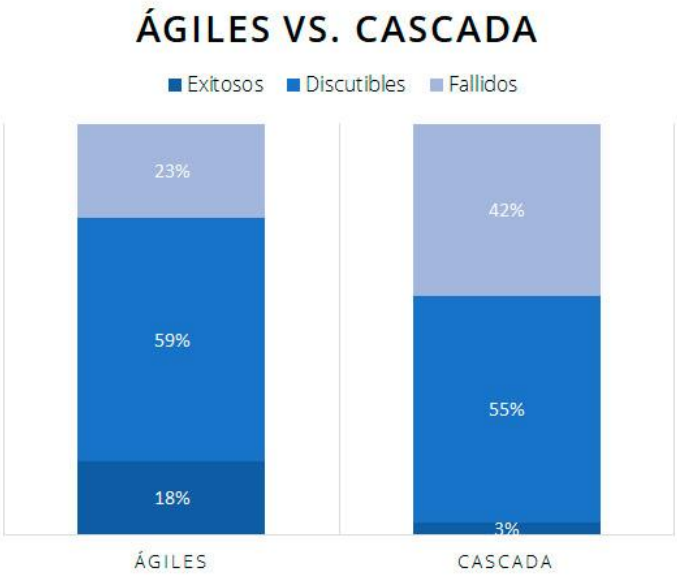


Figura 49. Porcentaje de éxito en proyectos según el uso de una metodología ágil o una tradicional [7].

Como podemos observar en las gráficas de la Figura 49, muestra la clara diferencia en el potencial de éxito que tiene el desarrollo de proyectos en la industria según el uso de una metodología ágil o una tradicional y secuencial. Analizando la gráfica, obtenemos los siguientes datos relevantes en el cómputo global de proyectos comparando el uso de metodología ágil con la tradicional:

- La probabilidad de **éxito** es 5 veces mayor haciendo uso de una metodología ágil.
- El porcentaje de **fracaso** en un 45% menor en comparación con la metodología tradicional secuencial.

Por otra parte, aplicar estas formas de trabajo pueden acarrear problemas si no hacemos uso correcto de la metodología. Al no existir un modelo único, se tiende a caer en el error de creer que ciertas actitudes durante el desarrollo del proyecto son ágiles cuando en realidad son lo contrario. Por ello, requiere de una formación, experiencia y diversas lecturas para comprender el uso de estas metodologías. Es recomendable antes de aplicarlas revisar el manifiesto ágil[22], el cual describe los puntos y la filosofía que hay detrás de esta forma de trabajo. Como indica, el manifiesto no es una lista de normas el cual haya que seguir al pie de la letra. Está enfocado a guiar y a recomendar su correcta implementación en el desarrollo de productos *software*.

En el mundo del desarrollo de *software*, no hay varias formas de aplicar la metodología ágil. Existen distintas interpretaciones y guías de desarrollo que se han adaptado al manifiesto ágil durante el paso de los años aplicando sus heurísticas. Durante el

[22] ***Manifiesto Agile:*** *https://agilemanifesto.org/*

progreso del trabajo, se combinaron dos marcos de trabajo populares en el sector: **Scrum**[23] y el tablero de **Kanban**[24].

Enfocaremos Scrum en nuestra metodología con el uso de unidades de trabajo (UTs), las cuales representan las funcionalidades a implementar en el producto. A cada UT se le asignan los requisitos del usuario, pruebas de aceptación, puntos de esfuerzo estimados y orden de desarrollo. Todas las UTs forma parte de un *backlog*, donde se recogen todas las UTs pendientes del producto y quedan pendientes de asignarse a un *sprint* del proyecto.

En nuestro trabajo, como muestra la Figura 50, se realizan un total de dos *sprints* de cuatro semanas cada uno dedicado al desarrollo de UTs de 10 puntos de esfuerzo en total en cada uno. Antes de comenzar con el proyecto, se creó un sprint adicional de dos semanas para la elicitación de requisitos, preparar el *backlog* creando las UTs y el desarrollo de proyecto. Además, antes de comenzar con un *sprint* de desarrollo, se dedica una semana de pausa entre *sprints* para establecer los requisitos del proyecto y la fase de diseño (prototipos, patrones de diseño, etc). Durante el transcurso del *sprint*, se aplican las UTs al producto aplicando las fases de programación y *testing*.

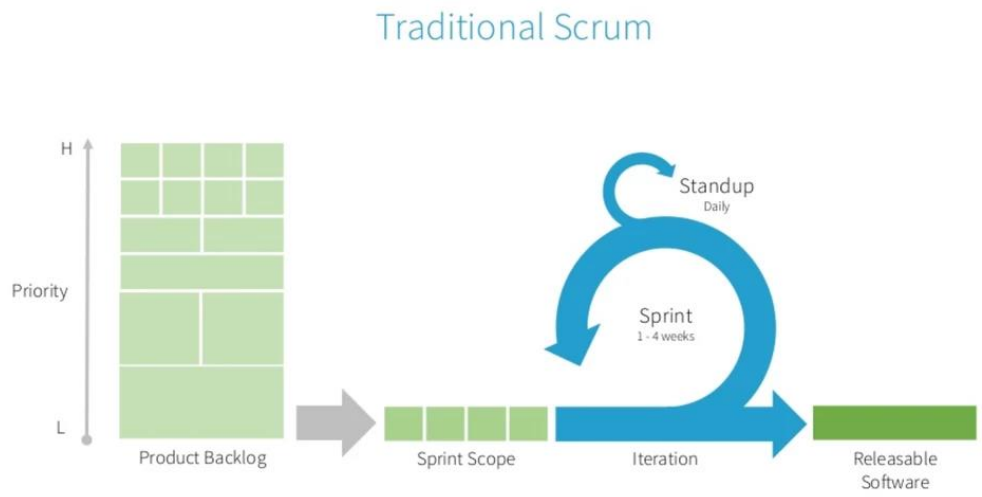


Figura 50. Esquema explicativo de las iteraciones por sprints en la metodología ágil Scrum. [21]

En colaboración con Scrum, se ha aplicado el tablero de Kanban para organizar las UTs del proyecto. Tiene por objetivo apoyarse en Scrum para mostrar de forma sencilla el estado del proyecto. Como muestra la Figura 51, las columnas muestran el backlog con las UTs pendientes, las fases de preparación de UTs, paquetes de trabajo durante un sprint y aquellas funcionalidades que se terminaron.

[23] ***Scrum*** *– Más información: https://shorturl.at/IU169*
[24] ***Kanban*** *– Más información: https://es.wikipedia.org/wiki/Kanban*

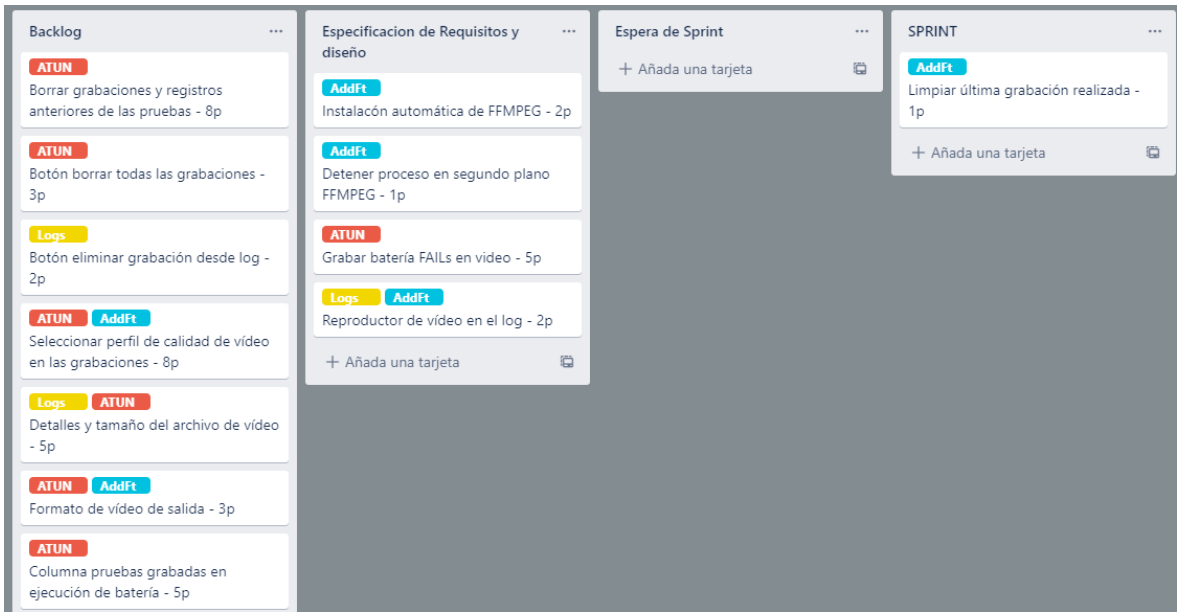


Figura 51. Tablero Kanban durante el transcurso del proyecto

Cronología

Durante el transcurso de este trabajo, se han realizado distintas tareas para conseguir finalizar con la solución requerida y esta memoria. Cronológicamente el trabajo se realizó entre febrero y julio de 2022, desde los trabajos de investigación hasta la programación del último sprint del desarrollo.

De forma secuencial, las etapas por las que transcurrió el trabajo fueron:

- **Investigación del trabajo, problema a resolver y herramientas de la solución** (7 de febrero – 2 de marzo): se identificaron los inconvenientes del proceso de revisión de pruebas automatizadas y se buscaron las formas de resolverlos, realizando una extensa búsqueda en los repositorios de Internet sobre aplicaciones de lanzamiento de pruebas automatizadas que mejorasen a la de la empresa. Durante este periodo, se investigó en el campo de las pruebas automatizadas y conceptos del campo que ayudarían a la redacción de la memoria. Se encontró que la solución óptima pasaba por grabar las pruebas automatizadas en vídeo.
- **Herramientas de la solución** (3 de marzo – 6 de abril): se buscó por los repositorios en línea grabadoras que pudieran lanzarse mediante línea de comandos. Se obtuvo que realizar las grabaciones mediante *FFmpeg* tenía un alto potencial de éxito con poco esfuerzo de implementación en comparación con otras soluciones más radicales. Se procedió a investigar sobre el uso de las herramientas y encontrar la configuración con el uso de recursos más optimizados, debido a la escasa memoria que cuentan las máquinas virtuales.
- **Preparación del proyecto** (7 de abril – 24 de abril): durante este periodo, se recopiló todo lo necesario para dejar todo listo de cara al desarrollo del proyecto. Entre las tareas realizadas, se destaca: descarga del código fuente de las herramientas de la empresa, diseño de la metodología a aplicar, aplicaciones

para apoyarnos en el proyecto (por ejemplo, Trello[25] para el tablero Kanban) y planificación de los *sprints* y esfuerzo disponible para el desarrollo de la solución. En todos los sprints se respetará el principio de MVP, evitando funciones a mitad de implementarse o falsos resultados simulados.

- ***Sprint* 0** (25 de abril – 8 de mayo): *sprint* previo a la programación del proyecto para elicitar los requisitos y diseñar los prototipos de la fase de diseño. Se definieron las UTs que se estimaron que podrían completarse en el transcurso del proyecto y se documentaron las UTs del primer *sprint*.
- ***Sprint* 1** (9 de mayo – 5 de junio): primer *sprint* de desarrollo de la solución. Se incluyeron las cuatro primeras UTs del *backlog* expuestos en el capítulo 6.1. Al finalizar el *sprint*, se completaron con éxito todos los paquetes de trabajo. Se entregó una primera versión de *AddFt* con la grabación de vídeo en las ejecuciones de pruebas automatizadas y el *wrapper* de *FFmpeg*. Al finalizar con el *sprint*, se realizaron los preparativos del último *sprint* detallando las UTs restantes del proyecto.
- ***Sprint* 2** (6 de junio – 3 de julio): último *sprint* del proyecto, en el que se incluyeron las últimas cuatro UTs del *backlog* del proyecto. Durante el transcurso del proyecto, el esfuerzo estimado de aplicar los cambios en ATUN fue mayor al esperado. Sin embargo, se pudo completar debido a que varias UTs no requirieron de tanto esfuerzo como se estimó en una primera instancia y se pudo finalizar sin problemas el *sprint*. Se completó de desarrollar la implementación de *FFmpeg* con *AddFt*, añadiendo los vídeos en los informes y dotando de mayor calidad al proyecto al automatizar acciones como detener procesos abiertos y la instalación de la herramienta de vídeo. Por último, se instauró la grabación de las pruebas en remoto modificando los componentes de ATUN y permitiendo acceder al vídeo de la prueba desde la aplicación.

[25] ***Trello**: https://trello.com/*

7. Conclusiones y trabajo futuro

Consideramos que los objetivos de este trabajo se han cumplido, integrándose de forma correcta en la infraestructura de pruebas automatizadas de todas las funcionalidades para la grabación en vídeo de las ejecuciones.

La introducción de las grabaciones de vídeo en el departamento ha supuesto un cambio significante y una mejora durante su transcurso, tanto en el desarrollo de las tareas diarias de pruebas automatizadas como en la comunicación con los desarrolladores y miembros de otros departamentos para la notificación de fallos. Antes de realizar la solución del trabajo, se hicieron pruebas manuales de la eficiencia que resultada combinar las grabaciones de vídeo junto al proyecto *AddFt* y resultó ser de gran ayuda para comprender los fallos que se producían causados por las propias pruebas y los introducidos en las nuevas versiones de la aplicación testeada.

La incorporación de grabaciones de ejecuciones de pruebas es una idea innovadora. Tal y como se comentó en el capítulo de estado del arte, no se encontró ninguna herramienta actual en el mercado que ofreciera dicha combinación o una funcionalidad similar a la deseada.

El mantenimiento de las infraestructuras de pruebas automatizadas no ha sido una tarea sencilla debido a diversas dificultades encontradas durante el transcurso del trabajo. Entre ellas, hay que destacar la escasa documentación técnica que estos proyectos poseen. Para conseguir el código fuente de ATUN fue una labor muy ardua al no encontrarse en los sistemas de control de versiones de la empresa y, tras consultar a varios compañeros del departamento, se pudo obtener de una copia de seguridad antigua. Es importante que una empresa entre sus herramientas cuente con las guías de desarrollo y un fácil acceso al código fuente de estas para poder facilitar modificaciones en su comportamiento o añadidos en sus requisitos. Por ello, habría que tener en cuenta una tarea a corto plazo en el departamento de, mediante el uso de ingeniería inversa, generar los manuales y documentación técnica de las funcionalidades que contiene y la infraestructura que forma para, en un futuro, ahorrar esfuerzos en la implementación de nuevos requisitos y una comprensión más fácil de su funcionamiento.

También es importante destacar la tarea pendiente por parte del departamento de buscar soluciones en los tiempos de ejecución de diversas pruebas automatizadas. Como vimos en la Figura 14, se da el caso de pruebas fallidas que superan incluso los 60 minutos de ejecución. En el último desarrollo de pruebas, se han realizado pruebas donde el *Prepare Data* se realiza directamente mediante un script SQL, el cual reduce el tiempo de ejecución de la prueba de hasta un 90% respecto a la introducción manual de datos requeridos. Una refactorización de pruebas sustituyendo la etapa de *Prepare Data* manual por la de un script de SQL para reducir la deuda técnica sería bastante deseable en un trabajo futuro.

Debemos tener en cuenta que la grabación de estas pistas de vídeo contiene un ciclo de vida muy corto, el cual cuando estos dejen de ser útiles, debemos deshacernos de ello para, no solo dar la posibilidad a la introducción de más grabaciones, sino por

cuestiones de privacidad. Es de vital importancia tener un gran control sobre estas pistas ya que podrían contener información sensible de la empresa, lo cual debería establecerse un control para evitar compartir dichos archivos al exterior de la Intranet de la empresa.

Podría darse la posibilidad de aprovechar el *wrapper* desarrollado en el proyecto *AddFt* junto a la grabadora *FFmpeg* y migrarlo al resto de la empresa, programando una nueva tecnología capaz de crear grabaciones independientes en los escritorios virtuales. Los empleados actualmente tienen a disposición herramientas para poder realizar capturas de pantalla, editarlas y compartirlas con otros compañeros mediante *Lync* [26]y correo electrónico. Realizar algo similar con capturas de vídeo de la pantalla podría ser un próximo reto para dotar de mayor calidad la información compartida en la empresa, respetando siempre su privacidad e integridad evitando el filtrado de datos e información confidencial que esta pueda poseer.

En lo personal, este trabajo me ha aportado mucha experiencia como desarrollador de software. En el mercado laboral nos encontramos en muchas ocasiones con la necesidad de realizar cambios en proyectos realizados por otros equipos de desarrollo y, frecuentemente, con abundante deuda técnica y escasa documentación del producto. La tendencia que hay actualmente en el mundo de la Ingeniería de Software es, cada vez mayor, el mantenimiento de programas ya desarrollados sobre la programación de otros nuevos, como observamos en la Figura 52.

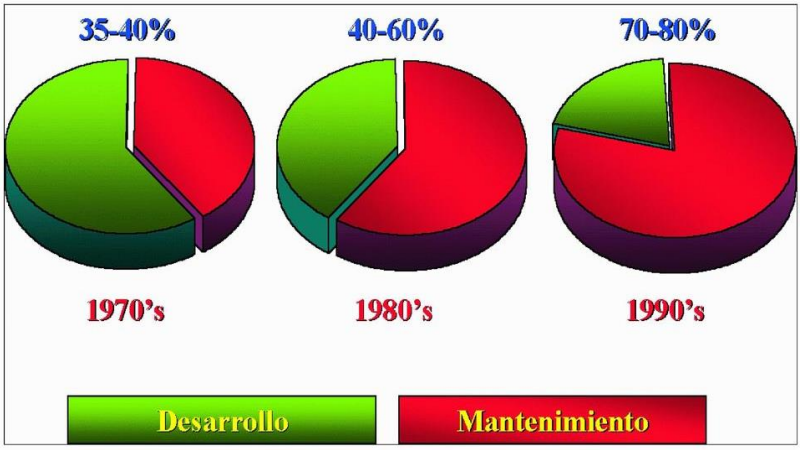


Figura 52. Gráficas de porcentaje de dedicación al desarrollo de nuevos productos frente al mantenimiento de programas software en la industria. [15]

En relación con el grado que he estudiado, cabe mencionar tres asignaturas que me han servido de gran ayuda y de apoyo para poder realizar de forma correcta este trabajo. Principalmente, la que me ha servido como punto de partida ha sido **Proceso de software**, la cual me ha aportado todos los conocimientos y actitudes necesarios a mostrar como ingeniero de software en la realización de un proyecto bajo la metodología ágil desde cero. Otras asignaturas importantes fueron **Desarrollo de software** y **Análisis y especificación de requisitos**, las cuales, durante el transcurso de las fases del proyecto, las tuve en cuenta para poder utilizar mejores patrones de diseño y escritura limpia de código, así como desarrollar una buena etapa de elicitación y documentación de requisitos, respectivamente.

[26] ***Lync*** *– Skype Empresarial: https://es.wikipedia.org/wiki/Skype_Empresarial*

8. Referencias

1. AUTOMATIZACIÓN DE CASOS DE PRUEBA CON MICROSOFT TEST MANAGER Y VISUAL STUDIO. [en línea], 2013. Disponible en: shorturl.at/gjnRV.
2. BYNAU, 2020. ¿Qué son los presets? *By NAU* [en línea]. Disponible en: https://bynau.com/que-son-los-presets/.
3. DUSTIN, E., 1999. *Automated Software Testing: Introduction, Management, and Performance*. 1st edition. Addison Wesley Professional. ISBN 978-1-282-70079-6.
4. ESTEVE AMBROSIO, D.A., 2015. *Implantación de un proceso de automatización de pruebas para una aplicación software*. 2015. S.l.: Universitat Politècnica de València.
5. FFmpeg. En: Page Version ID: 142865654, *Wikipedia, la enciclopedia libre* [en línea], 2022. Disponible en: shorturl.at/bopY6.
6. Fotogramas por segundo. En: Page Version ID: 142360495, *Wikipedia, la enciclopedia libre* [en línea], 2022. Disponible en: shorturl.at/DFHQ2.
7. GESTIÓN ÁGIL vs GESTIÓN TRADICIONAL DE PROYECTOS ¿CÓMO ELEGIR? - Escuela de Negocios FEDA. [en línea], [sin fecha]. Disponible en: shorturl.at/kqxY1.
8. Google Trends. *Google Trends* [en línea], [sin fecha]. Disponible en: https://trends.google.es/trends/.
9. Hard code. En: Page Version ID: 142971968, *Wikipedia, la enciclopedia libre* [en línea], 2022. Disponible en: shorturl.at/cvyDJ.
10. IBM Docs. [en línea], 2022. Disponible en: shorturl.at/CLSWX.
11. IBM Rational Quality Manager | reqpro.com. [en línea], [sin fecha]. Disponible en: http://reqpro.com/products/ibm-rational-quality-manager.
12. Introducción a las Pruebas de Software (Testing software). *Java desde 0* [en línea], 2019. Disponible en: https://javadesde0.com/introduccion-a-las-pruebas-de-software-testing-software/.
13. Kualitee. *Kualitee* [en línea], 2021. Disponible en: https://www.kualitee.com/.
14. Lado del cliente. En: Page Version ID: 121062804, *Wikipedia, la enciclopedia libre* [en línea], 2019. Disponible en: shorturl.at/jmOU4.
15. Mantenimiento del Software. [en línea], [sin fecha]. Disponible en: https://docplayer.es/8334529-Mantenimiento-del-software.html.
16. MAXIM MALYGIN, [sin fecha]. ¿Qué es Control de versiones de Team Foundation? - Azure Repos. [en línea]. Disponible en: shorturl.at/npRV9.
17. Microsoft Test Manager. [en línea], [sin fecha]. Disponible en: https://desarrolloweb.com/articulos/microsoft-test-manager-dotnet.html.
18. Producto viable mínimo. En: Page Version ID: 145523393, *Wikipedia, la enciclopedia libre* [en línea], 2022. Disponible en: shorturl.at/dfNQS.
19. Pruebas de software para profesionales. *Visual Studio* [en línea], [sin fecha]. Disponible en: https://visualstudio.microsoft.com/es/vs/test-professional/.
20. Quality assurance. En: Page Version ID: 1084704702, *Wikipedia* [en línea], 2022. Disponible en: shorturl.at/BMS48.
21. Qué es un Sprint de Scrum. *OpenWebinars.net* [en línea], 2018. Disponible en: https://openwebinars.net/blog/que-es-un-sprint-scrum/.
22. RAMOS, J., [sin fecha]. Los diferentes tipos de Pruebas de software. [en línea].

Disponible en: https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software.
23. Scrum sprint. En: Page Version ID: 1106804776, *Wikipedia* [en línea], 2022. Disponible en: shorturl.at/IJKQS.
24. Ver informe de progreso - Azure Test Plans. [en línea]. Disponible en: https://docs.microsoft.com/es-es/azure/devops/test/track-test-status.
25. Why to Do Automated Testing? *Bitbar* [en línea], 2015. Disponible en: https://bitbar.com/blog/why-to-do-automated-testing/.

9. Anexo ODS

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**		x		
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**			x	
ODS 10. **Reducción de las desigualdades**.				x
ODS 11. **Ciudades y comunidades sostenibles**.				x
ODS 12. **Producción y consumo responsables.**		x		
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El trabajo realizado tiene un impacto positivo a nivel social. La infraestructura de pruebas automatizadas realiza las labores de *testing* sobre las nuevas versiones de un ERP sociosanitario instaurado principalmente en residencias de tercera edad y grupos de atención especial.

El objetivo de las ODS más destacable durante el transcurso del proyecto es **salud y bienestar**. El ERP ayuda a las labores diarias de los empleados de los centros en los que se encuentra instalado el programa. Al dotar de mayor calidad a la infraestructura de pruebas automatizadas, aseguramos que las nuevas versiones del producto cumplan con sus requisitos y se encuentren libres de fallos en sus funcionalidades. De esta

forma, de forma indirecta favorecerá el cuidado y tratamiento de los grupos vulnerables en las residencias y centros de día que cuenten con el ERP.

Otro punto que se podría acreditar a este trabajo es **producción y consumo responsables**. Antes de programar los cambios en las herramientas de pruebas automatizadas, el relanzamiento de pruebas para comprender los fallos detectados por ellas era muy común y no se contemplaba el impacto que eso suponía en el consumo de energía. Con las grabaciones de vídeo, los mantenedores de las pruebas no requieren de un número reiterado de ejecuciones para comprender el fallo detectado y notificarlo al equipo de desarrollo si lo requiere.

Desde la empresa también se fomenta el ahorro energético, encendiendo tan solo aquellos equipos que se utilicen y apagarlos una vez se termina con la ejecución de una batería y no se les den más uso. Se esta forma, evitamos desperdiciar el consumo de electricidad en máquinas virtuales que no ejecuten pruebas.

El tercer y último objetivo que podemos asignar al trabajo es **industria, innovación e infraestructura**. Durante el proceso de investigación del trabajo, no se encontró ninguna herramienta de *testing* automatizado que contase con grabaciones de vídeo integradas de las ejecuciones de las pruebas. Si es cierto que es una funcionalidad que no se había contemplado antes en el sector, aún no se tiene en conocimiento si realmente supondrá una revolución en los procesos de pruebas de software automatizados. Por el momento en la organización ha facilitado el desarrollo de las labores de testeo y ha favorecido la comunicación entre el equipo de desarrollo y el de *testing*. Es por este motivo que, hoy en día, el impacto que este punto tiene en el trabajo por el momento es leve en comparación con los otros objetivos.

A pesar de ser un trabajo que aporta muchos beneficios en la sociedad, resulta complicado relacionarlo con más objetivos de desarrollo sostenible. En los proyectos tecnológicos, no es común encontrarse con casos que favorezcan a la conservación de los ecosistemas o la vida marina de forma indirecta, salvo aquellos que se realicen para su debido cometido.

Sin embargo, objetivos como **acción por el clima** podrían relacionarse con el punto de consumo responsable si la electricidad suministrada a las infraestructuras de la empresa viniese de combustibles fósiles y contaminantes como el carbón o el gas. Al no tener conocimiento de ello, el punto no podría relacionarse con el trabajo ya que podría provenir en su totalidad de fuentes renovables.