UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Dashboard para testeo de aceptación en un contexto industrial

Autor: Miguel Peiró García de la Reina

Director: Dr. Patricio Letelier Torres

Septiembre del 2012

Trabajo Final de Máster presentado para cumplir con los requisitos finales para la obtención del título de Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información, de la Universidad Politécnica de Valencia, 2012.

Agradecimientos

En estos dos últimos años he podido conocer a personas que me han hecho crecer tanto profesional como personalmente y que me han apoyado en los momentos críticos.

En primer lugar, agradecer de forma muy especial, a mi director de tesis y amigo, Patricio Letelier, gracias por la confianza que has depositado en mí, los consejos, el esfuerzo y tiempo que me has dedicado, y, sobretodo, por haberme dado esta gran oportunidad y haber conocido a gente tan grande como mis compañeros de laboratorio. Gracias Patricio.

También agradecer a cada uno los que han sido compañeros de laboratorio por haberme tratado como uno más desde el primer momento en el que entré en el laboratorio. Gracias a todos.

No me quiero olvidar de los compañeros de ADD Informática, sobretodo del grupo de testeo, con los que he compartido momentos de risa y me han dado a conocer una disciplina que tenía muy borrosa y han conseguido que en estos momentos adore y quiera seguir aprendiendo. Gracias compañeros.

Terminando, quiero acordarme de todos mis amig@s con los que comparto mi vida, con los que he crecido y compartido momentos inolvidables. Moltes gràcies.

Y, en último lugar y más importante, a mi familia y Aina por tener tanta paciencia conmigo, por haberme apoyado y ayudado en todo momento y por estar ahí siempre que os he necesitado. Muchas gracias.

Tabla de Contenido

Tabla de Figuras

Capítulo 1. Introducción

1.1. Motivación

El Desarrollo Dirigido por las Pruebas (Test-Driven Development) está introduciéndose con fuerza en el ámbito industrial de desarrollo de software, pero concentrándose particularmente en el ámbito de Pruebas Unitarias y de su automatización. Requisitos y pruebas suelen tratarse como dos tipos de artefactos gestionados por actividades y diferentes roles. Esta tesina se enmarca en TUNE-UP, una metodología que mezcla aspectos ágiles y tradicionales ofreciendo una solución pragmática y muy detallada en aspectos que otras metodologías dejan que sean resueltos por los propios desarrolladores. TUNE-UP es una metodología y herramienta para la gestión ágil de proyectos de desarrollo y mantenimiento de software. TUNE-UP incluye un enfoque innovador para la gestión de requisitos denominado TDRE [2] (Test-Driven Requirements Engineering), el cual se fundamenta en el hecho de especificar los requisitos mediante Pruebas de Aceptación (PAs). El analista, dentro del ciclo de desarrollo, es el encargado de definir las Pruebas de Aceptación, el programador implementa código para satisfacerlas y el tester realiza el testeo manual y cuando se crea oportuno realizará el diseño de las pruebas para su automatización.

Dentro de este paradigma, donde las pruebas de aceptación tienen el papel principal y el éxito de la iteración depende de la correcta implementación de las PAs y la verificación del comportamiento correspondiente es importante disponer de información sobre el estado del desarrollo en general de las PAs.

Esta tesis se ha desarrollado en el contexto de una colaboración con una PYME dedicada al desarrollo software. Esta empresa cuenta con unos 60 empleados que se encargan de desarrollar un ERP para el sector socio-sanitario, el cual está en mantenimiento continuo y el cual se acompaña de otros productos software complementarios desarrollados por la misma empresa.

Durante la realización del trabajo, se ha mantenido una estrecha colaboración con el equipo de testeo. El ritmo de desarrollo es bastante intenso, publicándose mensualmente

nuevas versiones de los productos, lo cual exige una adecuada gestión del testeo. En cuanto al proceso de testeo, éste se enmarca en la metodología y herramienta TUNE-UP.

El grupo de testeo está formado por 5 personas con más de 5 años de experiencia en el campo, que ha alcanzado un buen nivel de madurez tanto en respecto de su proceso de testeo como de las herramientas de soporte. En este punto el equipo de testeo necesitaba disponer de datos cuantitativos con los que poder evaluar, supervisar y optimizar el proceso de testeo.

En los últimos años han ido emergiendo diferentes soluciones que pretenden dar apoyo al proceso de testeo en el desarrollo software; cada una adaptada a la metodología y a los conceptos que utilizan. Un componente extra en estas soluciones son los conocidos *Dashboards*; los Dashboards están formados por diferentes informes, gráficas y tablas, que ofrecen una vista del estado en el que se encuentra el proceso de testeo.

En el ciclo de desarrollo de un producto software, se genera grandes cantidades de datos que son almacenados en bases de datos; un Dashboard tiene como propósito recopilar toda esa información de forma automática y mostrarla de tal forma que se pueda analizar y dependiendo de la situación y necesidades, poder tomar las decisiones correctas en el momento correcto.

1.2. Objetivos

El objetivo principal de este trabajo es realizar el diseño de métricas necesarias y de un entorno donde poder elaborar Dashboards a partir de las métricas diseñadas. Se va a realizar un diseño tras conocer algunas de las características de las diferentes soluciones que se pueden encontrar en la actualidad, adaptándolas a la metodología, TUNE-UP. Identificando las métricas interesantes, desarrollaremos la infraestructura necesaria para dar soporte a las mismas.

Una vez tengamos la infraestructura necesaria, se continuará con el diseño de las métricas; la elaboración pasará por un proceso definido en el que se detallarán las características de las mismas y, posteriormente, se realizará el diseño del entorno donde podremos elaborar Dashboards.

1.3. Estructura del trabajo

La tesina está divida en ocho capítulos que son los siguientes:

En el Capítulo 1 se incluye una introducción a la motivación que nos ha llevado al desarrollo de nuestro trabajo y presenta los objetivos de nuestra propuesta.

En el Capítulo 2 se presenta el enfoque Test Driven Requirements Engineering. En esta propuesta las pruebas de aceptación pasan a ser el elemento principal del proceso de desarrollo.

En el Capítulo 3 se explican los aspectos básicos de la metodología TUNE-UP y de su herramienta de apoyo TUNE-UP Software Procces. A continuación se explica cómo se realiza el seguimiento de la iteración con TUNE-UP.

En el Capítulo 4 se explica detalladamente el proceso dirigido por Pruebas de Aceptación. Después se hace una explicación de cómo lleva a cabo el proceso de testeo con TUNE-UP.

En el Capítulo 5 se presenta un breve resumen del estudio realizado sobre otras soluciones actuales que presentan un Dashboard, dando a conocer algunos de los informes y métricas más importantes de cada uno. Además, se presentarán las conclusiones a las que se llegaron tras el estudio.

En el Capítulo 6 se presenta la infraestructura desarrollada e implantada en el proceso de testeo, describiendo cada una de las herramientas que la componen.

En el Capítulo 7 se presenta un análisis detallado sobre las métricas que se van a presentar en nuestra propuesta para Dashboards. En las métricas se darán guías con las que tendremos una idea de las métricas que se complementan y pueden formar un Dashboard. Además, se presenta el prototipo del entorno de Dashboard con las características que ofrece.

En el Capítulo 8 se presentan las conclusiones a las que se han llegado con la realización de esta tesis y el trabajo futuro que ha habilitado la mismo.

Capítulo 2. Test Driven Requirements Engineering

El resumen de Test Driven Requirements Engineering presentado en este capítulo se ha elaborado a partir del contenido de la Tesina de María Company Bria)[1]

El enfoque **Test-Driven Development (TDD)** [8] se basa en que las pruebas deben dirigir el desarrollo del producto software. El TDD se ha aplicado esencialmente en el ámbito de la implementación, particularmente siguiendo el planteamiento "*no escribir código hasta disponer de las pruebas que debe satisfacer dicho código*". El TDD ha tenido un fuerte impulso con las metodologías ágiles, las cuales lo incorporan entre sus prácticas esenciales.

En productos software industriales, cuando se utilizan prácticas de ingeniería de requisitos, éstas mayoritariamente están soportadas por lenguaje natural, lo cual conlleva los ya conocidos inconvenientes relativos a ambigüedad. Sin embargo, la necesidad de validación puede más que las ventajas que puede ofrecer una especificación más formal y rigurosa de los requisitos. El cliente debe poder leer y comprender los requisitos para así poder dar su conformidad respecto de ellos. Las técnicas más populares para especificación de requisitos se resumen en **Casos de Uso** (utilizados en metodologías tradicionales, como RUP [10]) e **Historias de Usuario** (fichas de XP [7] o representaciones similares en otras metodologías ágiles como Scrum [11]). Si bien los elementos Actor (o tipo de usuario) y Requisitos (Caso de Uso o Historia de Usuario) pueden visualizarse y manipularse gráficamente o en fichas, la descripción de cada requisito se elabora esencialmente de forma textual en lenguaje natural.

Para definir los requisitos es clave involucrar al cliente. El acuerdo/contrato con el cliente y la planificación del desarrollo del producto deberían estar basados en los requisitos que se pretenden incorporar en el producto. Los requisitos son el objetivo a conseguir, es decir, es lo que espera el cliente que el producto software satisfaga.

El **modelo en V** considera las pruebas como un proceso que corre en paralelo con el análisis y el desarrollo, en lugar de constituir una fase aislada al final del proceso. En la representación gráfica clásica del Modelo en V (véase Figura 1), las fases de desarrollo de software aparecen a la izquierda (desde requisitos hasta la implementación de

unidades de código) y los correspondientes niveles de pruebas a la derecha (desde Pruebas Unitarias hasta Pruebas de Aceptación).

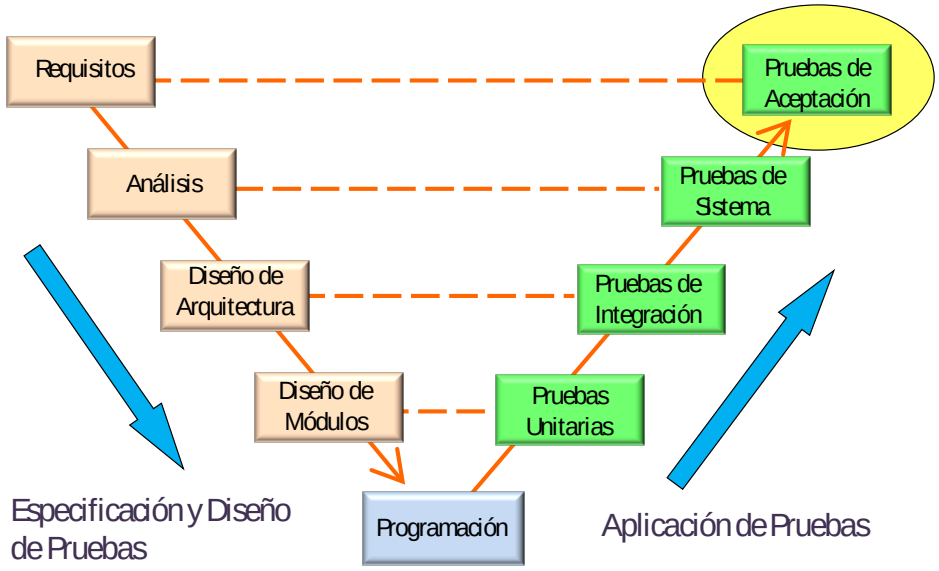


Figura 1.El Modelo en V

Partiendo de los requisitos o del diseño (a la izquierda) se deben planificar y preparar los niveles de pruebas correspondientes (a la derecha). En general, cada actividad de pruebas a la derecha valida la actividad enfrentada de la izquierda. También en general, los niveles superiores de pruebas en el diagrama están basados en **caja negra** (pruebas basadas en las especificaciones) y los niveles inferiores están basados en **caja blanca** (basadas en la estructura interna de los componentes del sistema).

La mayoría de las metodologías se basan en este modelo de pruebas, sin embargo, aquellas que apuestan decididamente por modelos de procesos iterativos e incrementales (RUP, entre las metodologías tradicionales, y todas las metodologías ágiles) utilizan de forma más explícita las **pruebas de sistema** y **pruebas de aceptación**. La única diferencia entre estos dos tipos de pruebas es el agente que las crea, las de aceptación son creadas por el cliente y las de sistema son creadas por el desarrollador. Aunque la mayoría de metodologías conocen e utilizan las **Pruebas de Aceptación**, en muy pocas las utilizan como hilo conductor del proceso de desarrollo software. En estas metodologías utilizan las Pruebas de Aceptación como **criterio de éxito de cada iteración**. Esto nos llevaría a clasificar estas metodologías como Test-Driven, en un contexto más de planificación y desarrollo global. Sin embargo, en ellas no existe una analogía entre el Test-Driven del ámbito de implementación y lo que sería **Test-Driven**

en el ámbito de los requisitos y de la planificación. El planteamiento en este último ámbito sería "*no incorporar un requisito a una iteración (o al plan en general) sin haber antes establecido sus Pruebas de Aceptación*".

En las metodologías populares (tradicionales o ágiles), los **artefactos Requisito y Prueba de Aceptación** son tratados de forma separada y abordados en diferentes actividades y momentos del desarrollo. Una vez establecidos los requisitos, se definen las Pruebas de Aceptación para ellos. Además, usualmente están involucrados dos roles también diferentes, Ingeniero de Requisitos (Analista o Analista Funcional) y Tester (es el tester especializado en Pruebas de Aceptación, no en Pruebas Unitarias y de Integración, las cuales normalmente son responsabilidad del programador).

Nuestra propuesta se denomina **Test-Driven Requirements Engineering (TDRE)** por referirnos a TDD pero en el ámbito de los requisitos y la planificación. TDRE integra los artefactos, actividades y roles asociados a la especificación y validación de los Requisitos y de las Pruebas de Aceptación. El concepto de Requisito se convierte en un contenedor de Pruebas de Aceptación, y son éstas las que adquieren protagonismo como especificación de cada requisito. Una de las ocho "buenas características" que recomienda la **IEEE 830** [9] para una especificación de requisitos se refiere a que cada requisito debe ser "**Verificable**". En nuestra propuesta los requisitos son verificables pues están especificados como Pruebas de Aceptación.

Si bien, como especificación de requisitos, las Pruebas de Aceptación pueden sufrir los mismos inconvenientes que los requisitos tradicionales (al usar lenguaje natural para especificarlas), una Prueba de Aceptación por su **granularidad** y **verificabilidad** es una unidad que determina de forma precisa una parte del comportamiento del sistema. Así, otras características recomendadas por el estándar IEEE 830 (Corrección, No ambigüedad, Completitud, Consistencia, Orden por importancia o estabilidad, Modificabilidad y Trazabilidad) también se ven favorecidas.

Aunque los requisitos y las actividades asociadas han ido haciéndose espacio entre los aspectos esenciales de un proyecto de desarrollo de software, para las pruebas no ha ocurrido lo mismo y sólo equipos de desarrollo con cierta madurez las integran como pieza imprescindible en sus procesos de desarrollo. Existen dificultades para la introducción de una "cultura", disciplina y prácticas de pruebas en un equipo de desarrollo. Entre los obstáculos o malas estrategias podemos destacar: carencia de un

proceso de desarrollo que integre las actividades de pruebas con el resto de actividades, sobrevaloración de la automatización de las pruebas (y particularmente las Pruebas Unitarias) como objetivo inmediato o único, y el poco énfasis en "rentabilizar" el esfuerzo invertido en pruebas. Lo anterior hace que la iniciativa de incorporación de pruebas además de ralentizarse en cuanto a resultados, vaya perdiendo fuerza en su proceso de implantación en la organización.

En TDRE la estrategia es implantar una "cultura" de pruebas basada en el aprovechamiento de éstas como hilo conductor del proceso de desarrollo. Nuestro enfoque TDRE está incorporado en la metodología **TUNE-UP** [2,3,4,5,6] y en su herramienta de apoyo **TUNE-UP Software Process**.

Para comprender cómo los requisitos pueden ser especificados mediante Pruebas de Aceptación comenzaremos con un ejemplo. Consideremos el requisito "Retirar dinero" en el contexto de un cajero automático. Una típica especificación narrativa podría ser la siguiente:

"El cliente debe poder retirar dinero del cajero en cantidades seleccionables. Siempre recibe un comprobante, a menos que el cajero se quede sin papel. Cuando se trata de un cliente preferencial puede retirar más dinero del que tiene en su cuenta, pero se le debe advertir que se le cobrarán intereses. El cliente debería poder cancelar en cualquier momento antes de confirmar el retiro de dinero. Las cantidades deberían poder servirse con los billetes que en ese momento tenga el cajero y no se deberían aceptar otros montos. Sería conveniente avisar cuando el cajero esté realizando operaciones internas mostrando un mensaje: El dinero retirado de la cuenta debe poder comprobarse en los registros de movimientos de la cuenta…".

La Figura 2 ilustra algunas alternativas de especificación para este requisito (incluyendo la anterior descripción narrativa como opción por defecto). Los iconos reflejan la conveniencia de cada alternativa de especificación.

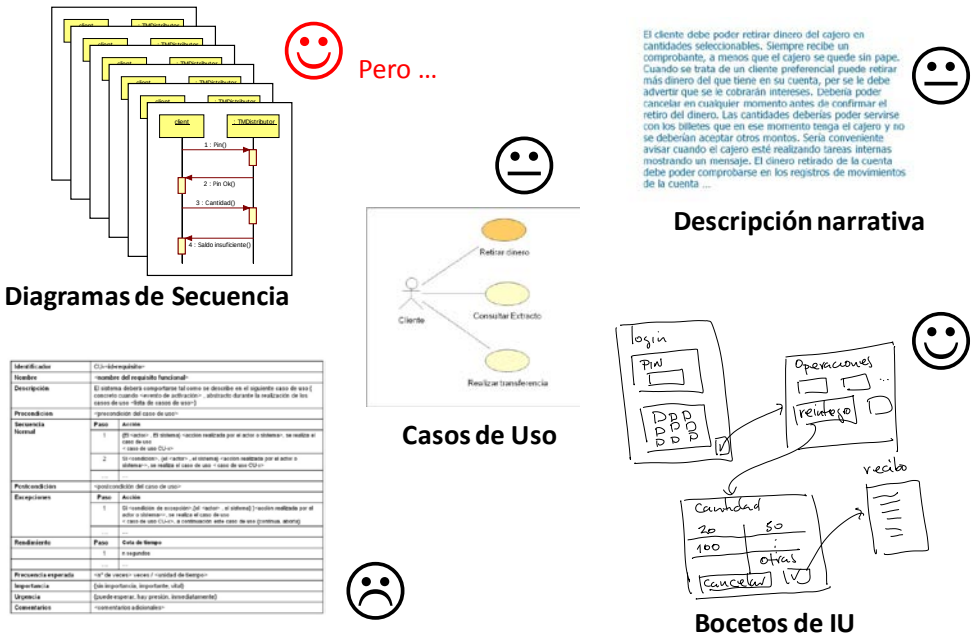


Figura 2. Alternativas populares para la especificación de requisitos

Elaborar un ***Diagrama de Secuencia*** para definir cada escenario de ejecución del requisito puede parecer interesante, sin embargo, en general no resulta apropiado por la gran cantidad de diagramas generados. Resulta más interesante la identificación de los escenarios que la ilustración de cada uno de ellos en un diagrama.

La ***descripción narrativa*** no es descartable, al menos para dar una breve definición del requisito centrándose en definir los conceptos involucrados (con la idea de un glosario o de un sencillo Modelo de Dominio).

Un ***Modelo de Casos de Uso*** no es mala idea, especialmente para organizar y visualizar los requisitos de un sistema nuevo. Sin embargo, un Modelo de Casos de Uso no es apropiado para ilustrar la estructura de requisitos detallada de un producto software en situaciones de mantenimiento a más largo plazo, ya que un producto software de tamaño medio puede tener miles de requisitos. La visualización y gestión de gran cantidad de requisitos necesita de mecanismos más apropiados.

Los ***bocetos*** (visualizaciones muy preliminares) de la Interfaz de Usuario (IU) son siempre bienvenidos pues son una herramienta efectiva de comunicación y validación con el cliente, el cual puede hacerse una idea del producto. En este contexto de requisitos no debe pretenderse realizar el diseño final de las IUs sino más bien paneles

con cierto ámbito de datos, sin profundizar en tipos de controles de interfaz o cuestiones de estética de formularios/páginas.

Las ***plantillas*** son una de las alternativas de especificación más usadas para Casos de Uso. Las plantillas son elegantes y proporcionan una sensación de orden en la especificación. Sin embargo, en general resultan contraproducentes ya que tienden a dar un tratamiento uniforme en cuanto al nivel de detalle para todos los requisitos. En aquellos muy simples se tiende a incluir cosas obvias o irrelevantes sólo para poder cubrir todos los apartados de la plantilla. Cuando un requisito incluye varios (o muchos) escenarios, el intento por sintetizar todos los escenarios en una plantilla (que sólo ofrece pasos y excepciones) lleva normalmente a especificaciones enrevesadas.

Nuestro enfoque TDRE apuesta por especificar los requisitos usando los elementos que se muestran en laFigura3: una **breve descripción narrativa** que establece los conceptos y motivación del requisito**, bocetos de la IU** (si procede) y una **lista de Pruebas de Aceptación** (PAs). Estas PAs se refinarán desde simples frases que dan nombre a los escenarios, hasta pruebas diseñadas, listas para ser aplicadas o para automatizarse y aplicarse en regresión. Así, los requisitos actúan como contenedores para las PAs.

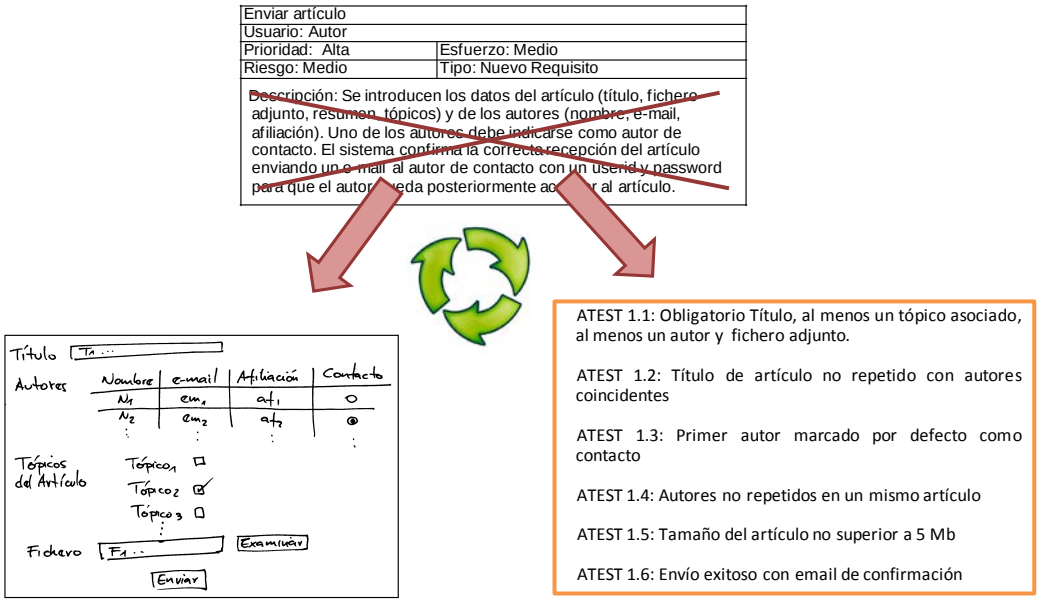


Figura 3. Especificación de requisitos en TDRE

Dependiendo del requisito, podría ser útil utilizar otras formas de especificación con carácter complementario, por ejemplo un **Diagrama de Actividad** si el comportamiento asociado al requisito es de carácter algorítmico o un **Diagrama de Estados** si el

comportamiento incluye habilitación o deshabilitación de acciones de acuerdo con el estado del sistema. La premisa esencial es pragmatismo respecto de la especificación, con lo cual no se descarta el uso combinado de alternativas de especificación, pero el criterio primordial debe ser el rentabilizar el esfuerzo en especificación y facilitar el mantenimiento de dicha especificación. Por otra parte, desde el punto de vista de esfuerzo de mantenimiento, especialmente en cuanto a consistencia, es importante no abusar de solapes o duplicaciones de especificaciones en diferentes medios de representación.

Las miles de PAs que fácilmente puede llegar a tener un producto software deben organizarse adecuadamente para poder ser gestionadas de forma eficiente. Un **grafo dirigido** (Figura 4) es una representación adecuada para realizar un refinamiento por niveles. Dicho grafo permite tanto la visualización de relaciones de descomposición como de dependencia entre requisitos. Así, cada nodo es un requisito funcional o no funcional. Los arcos entre nodos establecen relaciones padres-hijos (mediante las cuales se hace una descomposición de los requisitos) o relaciones de dependencia del tipo "nodos que afectan nodos" (estas relaciones son clave para realizar análisis de impacto). Los nodos de la parte más alta podrían, por ejemplo, seguir la clasificación de características ofrecida por la **ISO/IEC 9126** como punto de partida para la definición de los requisitos del sistema.

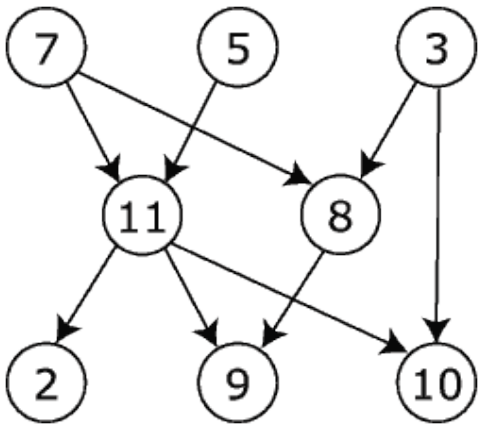
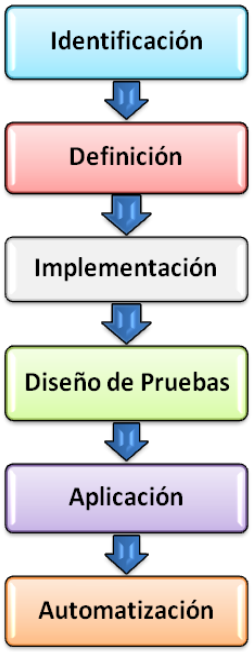


Figura 4. Representación de un grafo acíclico dirigido

La PA cuando va pasando por el proceso de desarrollo software va cambiando de estados, dependiendo de los recursos que disponga la empresa tendremos más o menos agentes interactuando con las PAs a lo largo del proceso. La situación ideal sería tener al menos tres roles diferentes: el **Ingeniero de Requisitos** (a nivel de empresa conocido

como Analista funcional o Analista de Negocio), el **Programador** y el **Tester**. Si por falta de recursos no se tuvieran los tres roles, el mismo agente que analiza podría programar o el mismo que analiza podría testear, lo conveniente es que no testee el mismo agente que ha implementado.

Los estados por los que pasa una PA se explican a continuación:



- **Identificación:** Se capturan los requisitos negociando con el cliente y se identifica el nombre de las PAs. Cuando se tienen las PAs identificadas ya se pueden tomar decisiones de planificación.

- **Definición:** Se especifica el contenido de la PA y se hace el análisis de impacto (a que otras partes del producto podría afectar el cambio). Una vez evaluamos el análisis de impacto y se valida con el cliente podemos decidir si el cambio que se define en la prueba es aceptado o no.

- **Implementación:** Durante la implementación el programador gracias a las PAs (se podrían considerar como un contrato Analista-Programador) tiene un criterio de éxito medible ya que escribe código con la idea de satisfacer la PA. Una vez ya ha pasado todas las PAs ha terminado de implementar.

- **Diseño de Pruebas:** El tester es el encargado de buscar las diferentes combinaciones de datos adecuadas para las diferentes instanciaciones de la PA.

- **Aplicación:** Se aplica la PA y se valida la implementación.

- **Automatización:** Hasta este nivel todo es manual, a partir de aquí si la empresa tiene suficientes recursos y se va a incluir la automatización de la prueba se podrían aplicar pruebas de regresión automatizadas

Siguiendo con el ejemplo anterior, el requisito "**Retirar dinero**" podría ser un nodo de la estructura de requisitos. Los nombres que identifican sus PAs podrían ser:

1. *Reintegro usando cantidades predefinidas habilitadas*
2. *Reintegro con cantidad introducida por cliente*

3. *Reintegro saldo < cantidad*
4. *Cancelación de operación*
5. *No disponibilidad de billetes*
6. *No disponibilidad de papel para recibo*
7. *Reintegro saldo < cantidad con cliente preferencial*
8. *Excedido tiempo de comunicación con sistema central*
9. *Aviso de operaciones internas del cajero*
10. *Excedido tiempo de espera para introducción de acción*

Una PA tiene como propósito demostrar al cliente el cumplimiento parcial o total de un requisito del software. Las características de una PA son:

- Una PA describe un escenario (secuencia de pasos) de ejecución o un uso del sistema desde la perspectiva del cliente. Las PAs cubren desde escenarios típicos/frecuentes hasta los más excepcionales
- Puede estar asociada a un requisito funcional o requisito no funcional. Un requisito tiene una o más PAs asociadas
- Una PA puede tener infinitas instanciaciones (ejecuciones con valores concretos)

La definición de una PA se separa en cuatro apartados: Condición, Pasos, Resultado Esperado y Observaciones.

- **Condición.** Es opcional, y se utiliza para establecer condiciones previas antes de aplicar los pasos de la prueba. Normalmente se refieren al estado de la BD antes de ejecutar la prueba y/o la navegación necesaria en la IU para localizarse en el punto adecuado para realizar la prueba (cuando no sea obvio)
- **Pasos.** Son las acciones de interacción del actor con el sistema. Cuando son varias acciones, éstas pueden ponerse en una lista numerada. Deben ser simples, del estilo “seleccionar…”, “introducir...”, evitando hacer referencia explícita a controles de interfaz o cualquier tecnicismo que dificulte su validación con el usuario
- **Resultado esperado**. Es el efecto de las acciones de interacción del actor. Cada acción puede provocar uno o más resultados. Es importante que cuando se trate

de mensajes al usuario se incluya el texto como parte del resultado esperado, así el programador dispone de dicha información ya validada con el cliente

- **Observaciones**. Es un apartado opcional, son recomendaciones que el analista estima conveniente hacerle al tester y/o programador

Las condiciones y/o resultados esperados pueden establecerse en el ámbito del requisito contenedor de la PA o de otros requisitos. Esto, como indicaremos más adelante, permitirá establecer relaciones de dependencia entre requisitos.

A continuación se presenta como ejemplo la definición de la PA "*Reintegro saldo < cantidad*".

CONDICIÓN
- ➢ Debe existir un cliente normal (esta característica se establece en el apartado datos básicos del cliente)

PASOS
- → Intentar reintegro con cliente normal y con cantidad solicitada mayor al saldo

RESULTADO ESPERADO
- ✓ Se muestra el mensaje "La cantidad solicitada supera su saldo actual, vuelva a introducir la cantidad" y se retorna a la ventana para introducir la cantidad

Es importante que el analista establezca un orden en las pruebas, desde las correspondientes a escenarios más típicos/frecuentes hasta aquellos más excepcionales. Éste orden servirá de guía de implementación a los programadores y al trabajo del tester. El programador debería escribir el código con la idea de satisfacer las PAs de forma incremental, hasta implementar toda la funcionalidad y superar todas las PAs. Por su parte, y adicionalmente, el programador podría realizar testeo unitario y de integración para las piezas de código que escriba. Dichas Pruebas Unitarias y de Integración, no siempre están relacionadas directamente con los requisitos puesto que su propósito se orienta más a verificación del software que a su validación (las PAs validan la implementación respecto de los requisitos del cliente). Sin embargo, en casos en los cuales las Pruebas Unitarias y de Integración se derivan o tienen una relación más directa con los requisitos, éstas se ven favorecidas porque en TDRE los requisitos están especificados como pruebas. Por otra parte, el tester diseñará, aplicará, y documentará una o más ejecuciones instanciadas (con valores concretos) para cada PA.

Las PAs se rentabilizan durante el proceso de desarrollo de software porque permiten:

- Especificar y validar los requisitos.
- Valorar adecuadamente el esfuerzo asociado a la incorporación de un requisito. Es más sencillo valorar el esfuerzo que requiere la satisfacción de cada PA.
- Negociar con el cliente el alcance del sistema en cada iteración de desarrollo. Las PAs introducen un nivel de granularidad útil para negociar el alcance con el cliente. Un requisito no necesita implementarse “todo o nada”, puede hacerse de forma incremental postergando la satisfacción de ciertas PAs.
- Guiar a los desarrolladores en la implementación ordenada del comportamiento asociado a un requisito. Si bien las PAs no son tan exhaustivas como las Pruebas Unitarias generan gran parte de la cobertura de código. Esto no significa necesariamente prescindir del resto de niveles de pruebas del Modelo V, sino que en un contexto de recursos limitados, las PAs deberían ser las pruebas esenciales o mínimas que deberían definirse y aplicarse al sistema.
- Identificar oportunidades de reutilización. El detalle y precisión proporcionado por las PAs permite identificar en la especificación de requisitos posibles solapes de comportamiento que den origen a oportunidades de su reutilización (lógica de la aplicación).

Capítulo 3. TUNE-UP

El resumen de TUNE-UP presentado en este capítulo se ha elaborado a partir del contenido de la Tesina de María Company Bria) [1]

3.1. Metodología TUNE-UP

TUNE-UP es una metodología desarrollada a partir del trabajo día a día en una PYME de desarrollo de software con vocación de mejora continua del proceso. TUNE-UP se ha definido bajo el marco de varios proyectos universidad-empresa. TUNE-UP ha conseguido su madurez en más de seis años de aplicación y refinamiento aplicándose en proyectos industriales, en el ámbito académico y también como objeto de publicaciones de investigación.

TUNE-UP se inspira en la esencia de **PSP** (Personal Software Process) [12], donde se destaca que la base del éxito radica en una disciplina de trabajo y productividad individual centrada en la gestión de los compromisos. En cuanto a gestión del tiempo y de priorización del trabajo personal TUNE-UP está alineada con los principios de **GTD** (Getting Things Done) [13] y **The Seven Habits of Highly Effective People** [14]. Para la organización y fácil acceso al trabajo de un agente TUNE-UP aporta una innovadora variación de **sistema Kanban**, el cual se integra con un simple pero potente mecanismo de workflows flexibles, permitiendo orquestar de forma automática gran parte de la colaboración necesaria entre actividades. TUNE-UP ayuda en cada momento del proyecto a responder a la pregunta: *¿conseguiré cumplir con los plazos de entrega de mis tareas? o ¿seremos capaces de cumplir con los plazos de entrega al cliente?* Estas simples preguntas inquietan a cualquier gestor o participante de un proyecto. No contar con una respuesta acertada, y sobre todo oportuna, conlleva en la mayoría de los casos graves complicaciones en el proyecto.

¿Por qué otra metodología y otra herramienta existiendo ya alternativas disponibles?

Después de trabajar tanto con metodologías tradicionales **(RUP y Métrica)** como ágiles **(XP y SCRUM)**, y utilizado muchas herramientas de apoyo para diferentes actividades en el proceso de desarrollo (herramientas para modelado o para pruebas, IDEs para programación, etc.), la principal lección aprendida es que los lenguajes de programación y notaciones, así como sus herramientas de apoyo, aunque contribuyen al éxito de un

proyecto no son tan decisivos como lo es el proceso de desarrollo en sí. Sin embargo, siendo las metodologías las protagonistas al respecto, las propuestas actuales continúan lejanas de la realidad cotidiana de un equipo de desarrollo. Por un lado, las metodologías tradicionales de entrada tienen el impedimento de ser demasiado genéricas y amplias, lo cual requiere un gran esfuerzo de configuración e implantación en el equipo, el cual puede ser inviable de abordar. Por otra parte, las metodologías ágiles parten de una visión demasiado simplificada de lo que es un proceso de desarrollo y particularmente del mantenimiento. Un producto exitoso requerirá mantenimiento. Con una metodología ágil el proceso de generación inicial de un producto puede resultar acelerado pero planteándose una perspectiva seria de mantenimiento no constituye un enfoque adecuado, puesto que cada vez se va haciendo más imprescindible el llevar un registro y control de los cambios y del comportamiento implementado en el producto. Esta información en las metodologías ágiles no es gestionada (ni siquiera almacenada) después de la implementación. Normalmente las metodologías tradicionales proponen una planificación también tradicional, es decir, basada en las técnicas genéricas utilizadas en todo tipo de proyectos. Dichas técnicas están siendo duramente criticadas y quedan en evidencia sus carencias para realizar un seguimiento del proyecto cuando el modelo de proceso es iterativo e incremental. *¿Quién ha probado con éxito, en este contexto, gestionar adecuadamente un proyecto software usando un diagrama Gantt?* Las metodologías ágiles han sabido abordar este aspecto y uno de sus puntos más destacables es precisamente la planificación, utilizando técnicas tales como los **diagramas Burn-down**.

Un proyecto de desarrollo o mantenimiento de software tiene por objetivo el conseguir una entrega exitosa del producto, es decir, una versión operacional que pueda ser explotada y rentabilizada por el cliente. Para conseguir este objetivo TUNE-UP incluye los siguientes elementos:

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

- **Proceso de desarrollo dirigido por las pruebas** (Test-Driven). La definición de una WU es básicamente la especificación de sus pruebas de aceptación

acordadas con el cliente. A partir de ahí, todo el proceso gira en torno a ellas, se estima el esfuerzo de implementar el comportamiento asociado y de aplicar las pruebas, se diseñan las pruebas e implementa dicho comportamiento, y finalmente, se aplican las pruebas sobre el producto para garantizar su correcta implementación.

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

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

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

TUNE-UP es una metodología que incorpora aspectos de metodologías ágiles y de metodologías tradicionales. Las dos primeras características (proceso iterativo e incremental, y proceso centrado en las pruebas de aceptación) clasifican a TUNE-UP como metodología ágil, sin embargo, las otras características están más próximas de lo que sería una metodología tradicional.

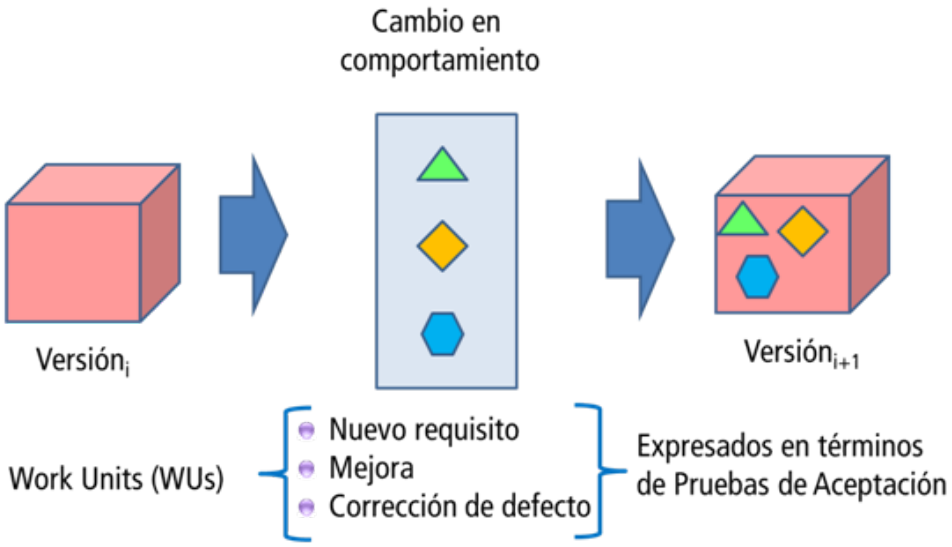


Figura 5. Estrategia Global de TUNE-UP

La Figura 5 ilustra la estrategia global de desarrollo propuesta por TUNE-UP. Una nueva versión del producto incluirá cambios de comportamiento con respecto de la versión anterior. Dichos cambios de comportamiento pueden ser en general **Nuevos Requisitos**, **Mejoras de requisitos** existentes o **Correcciones de defectos**. En TUNE-UP todos estos tipos de cambios de comportamiento se especifican de forma homogénea como PAs. Así, por un lado desde el punto de organización del trabajo todo tipo de cambio que está en una versión se aborda de forma integrada en cuanto a planificación y asignación de recursos. Por otra parte, desde la perspectiva de la especificación del cambio y su posterior verificación, en lugar de tener artefactos diferentes (requisitos/cambios/correcciones y PAs), se tiene sólo un artefacto llamado **Unidad de Trabajo** (WorkUnit, WU) expresado en términos de PAs.

TUNE-UP está dirigido por las PAs. Es un enfoque TDD pero en el ámbito más general de la gestión del proyecto y especialmente de los requisitos del producto. Tal como se ilustra en la Figura 6, y a modo de ejemplo (pues los roles podrían variar según sean los recursos disponibles y las exigencias del proyecto), el Analista define con el Cliente los cambios en el producto en términos de WUs y sus correspondientes PAs. El

Programador escribirá código para satisfacer dichas PAs y finalmente el Tester aplicará pruebas para asegurar que el comportamiento del sistema satisface las PAs.

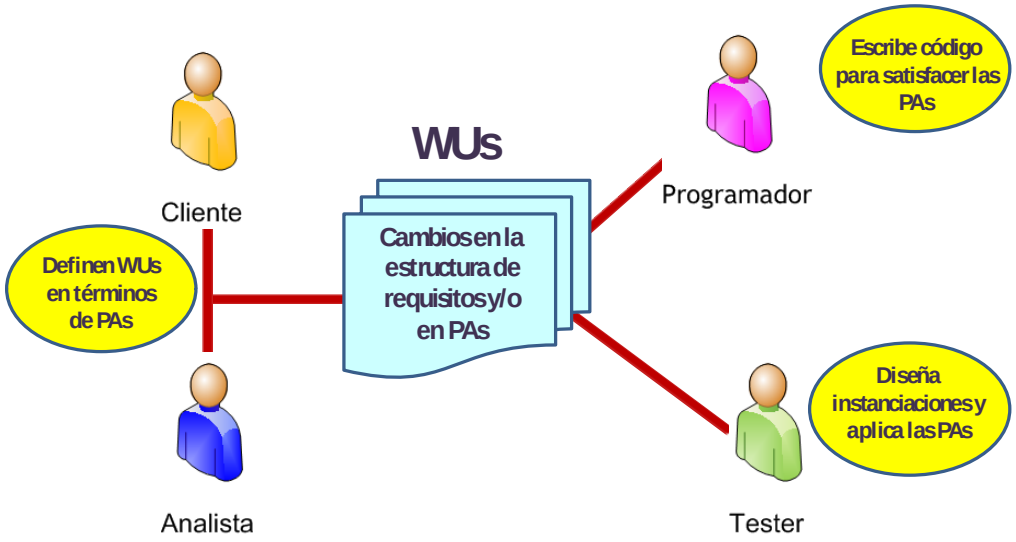


Figura 6. Proceso dirigido por las PAs

Dependiendo del tipo de WU y del proyecto en el cual debe llevarse a cabo el cambio de comportamiento de un producto, puede ser necesario realizar diferentes actividades y en diferente ordenamiento temporal. En TUNE-UP cuando se crea una WU se le asigna el workflow más apropiado. La Figura 7 muestra un **workflow** básico para el desarrollo de una WU, en el cual se han definido cuatro roles y diez actividades. TUNE-UP permite crear y modificar workflows según se requiera. En TUNE-UP se pueden gestionar diferentes productos con diversos equipos de desarrolladores. Cada producto tiene asociado ciertos workflows los cuales pueden compartirse entre diferentes productos, así como las actividades puedes ser utilizadas en diferentes workflows. Las actividades de cada workflow pueden variar significativamente dependiendo de factores tales como: cantidad y especialización de agentes participantes, validaciones o negociaciones predeterminadas con el cliente, características del producto (necesidad de migración, traducción, etc.), niveles y actividades de pruebas (unitarias, de integración, de aceptación, pruebas de regresión, automatización de pruebas), etc. El proceso de mejora continua se plasma en el refinamiento de los workflows.

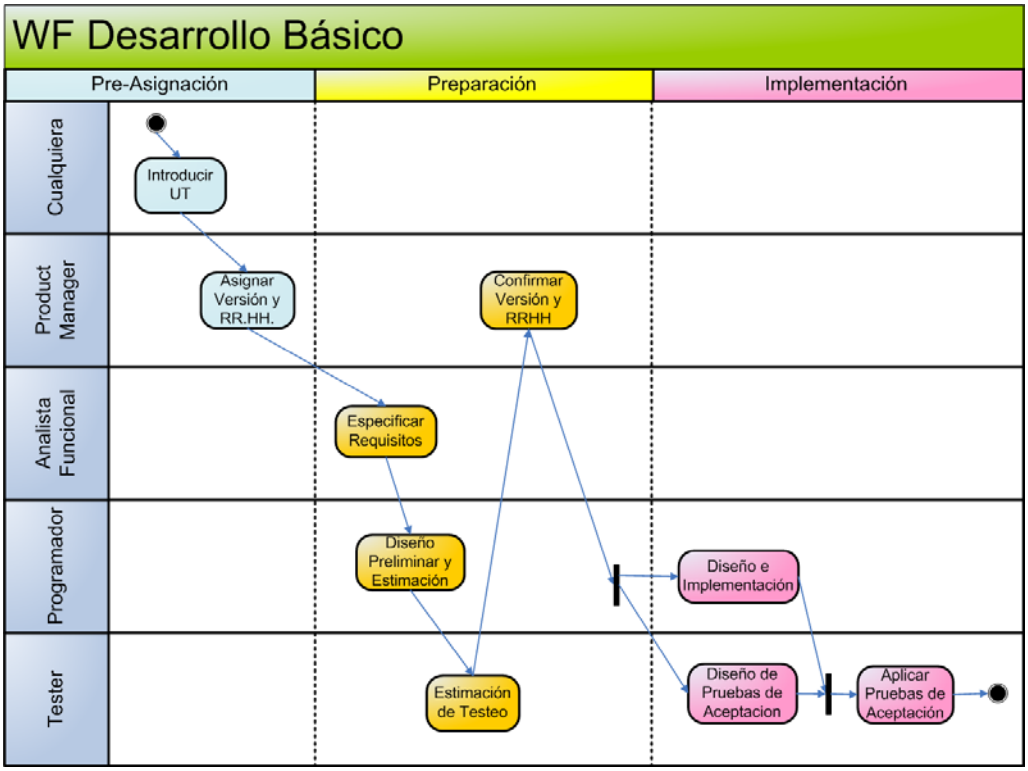


Figura 7. Un ejemplo de Workflow simple

En TUNE-UP se utiliza la siguiente terminología para referirse a comportamiento no deseado del producto: Fallo, Defecto y Error. El **fallo** es el síntoma de comportamiento no deseado que manifiesta el producto en su utilización. El **defecto** es lo que produce el fallo. Los defectos pueden estar en el código del producto o en cualquiera de sus especificaciones, por ejemplo puede que el código esté correcto respecto de la prueba, pero la prueba puede no ser correcta. Finalmente, el **error** es la acción humana (realizada o no realizada por los desarrolladores), es lo que originó el defecto. En la Figura 8 se ilustra cómo en TUNE-UP se lleva a cabo un tratamiento integral de los fallos. Todo fallo se especifica como una PA con un KO contenida en una WU. En caso que el fallo se resuelva en una misma WU (sea esta de cualquier tipo), se llevará a cabo el correspondiente re-trabajo (volviendo a realizar el análisis o programación requerida). Si el fallo no es abordado en la WU donde se detecta simplemente si la detección del fallo se hace fuera del contexto de una WU, entonces se creará una nueva WU de tipo fallo que se llevará a cabo en la versión actual o en alguna versión futura.

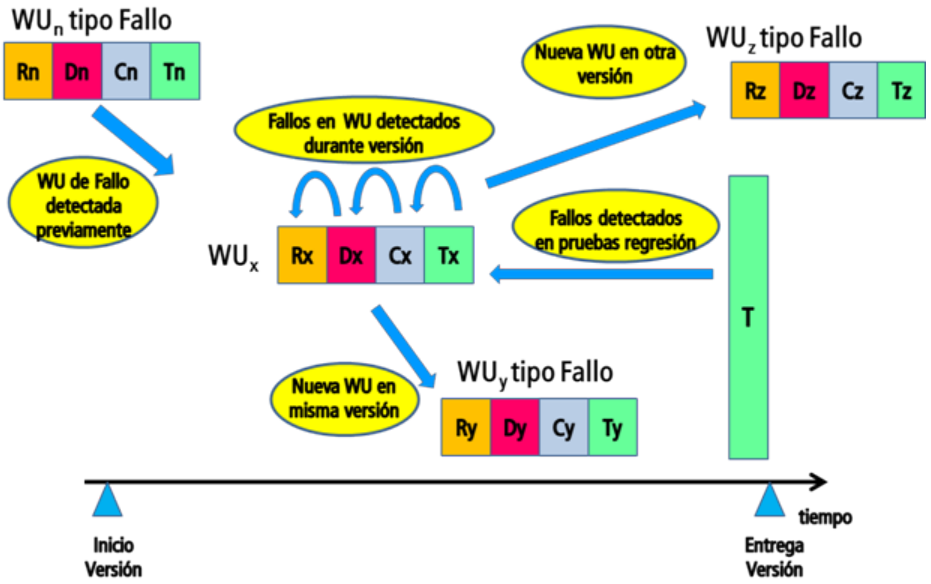


Figura 8. Gestión de Fallos en TUNE-UP

3.2. Herramienta TUNE-UP Software Process

TUNE-UP Software Process es el nombre de la herramienta de apoyo para la aplicación efectiva de la metodología TUNE-UP. La herramienta está formada por cuatro módulos principales: **Personal Planner (PEP)**, **Work Unit Manager (WUM)**, **Version Contents & Tracking (VCT)** y **Requirements Manager (REM)**. A continuación se describe brevemente cada uno de estos módulos.

Personal Planner (PEP)

Es el punto de entrada al espacio de trabajo del agente. Cuando un agente inicia su jornada laboral, accede al PEP (Figura 9) para ver el trabajo en el cual participa. Este trabajo corresponde a las actividades que tenga asignadas en las WUs no terminadas. En el **Kanban** (grid del extremo superior izquierdo de la Figura 9) el agente puede determinar fácilmente en el PEP qué actividades tiene pendientes o en progreso (activas o pausadas). Además puede distinguir aquellas actividades que tienen pendientes o en proceso otros miembros del equipo y que, o bien estarían por llegarle, o ya ha finalizado, y han continuado en actividades posteriores en el workflow. El agente tiene que seleccionar la WU en la qué va a trabajar. Un aspecto clave para el proyecto es que los agentes se dediquen a las actividades acertadas de acuerdo a sus prioridades. El PEP ofrece una variedad de facilidades de filtrado y ordenamiento de información para que el agente pueda determinar las prioridades de su trabajo y realizar una correcta elección.

El Kanban resume las contabilizaciones de las unidades de trabajo según la actividad y estado en el que se encuentran, y en el grid de la derecha de la Figura 9, muestra información de las actividades de dichas WUs incluyendo: producto, versión, descripción de la unidad de trabajo, estado de la actividad actual dentro del workflow, etc.

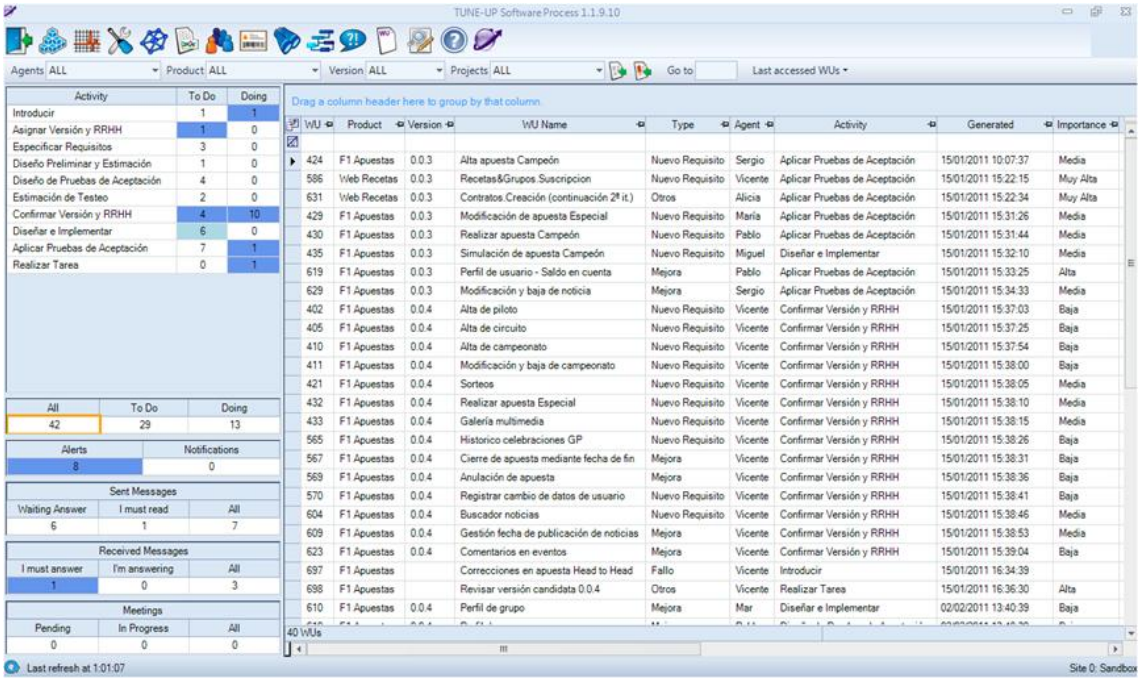


Figura 9. Personal Planner (PEP)

Work Unit Manager (WUM)

Cuando el agente decide la WU y la actividad en la que desea trabajar, accede con ella al **WUM** (Figura 10). En la ficha de la parte superior del WUM se muestran los datos generales de la WU, y en la parte inferior, un conjunto de pestañas con información más específica. En la **pestaña Tracking** se muestra la lista de actividades del workflow por las que ha pasado la unidad de trabajo. Cada actividad está asociada a un registro de seguimiento que incluye: la actividad, el agente que la desarrolla, el estado en el que se encuentra, etc. Además, en esta pestaña, con los **botones Record, Pause y Finish**, el agente puede controlar el registro de tiempos, activando, pausando o finalizando la actividad. Al finalizar una actividad, la WU continúa su workflow o puede a petición del usuario dar un salto a otra actividad del worflow. Otras facilidades para gestionar una WU y que se encuentran en otras pestañas del WUM son: **documentación** (para la WU, para apartados específicos del producto, para mensajes como adjuntos y para ejecuciones de PAs por ejemplo para indicar pasos de reproducción de un fallo),

mensajes para comunicarse con otros agentes en el contexto de la WU, **reuniones** para registrar tiempo y acuerdos alcanzados, **relaciones** con otras WUs, detalle de los **tiempos registrados** y **estimados** para cada actividad, e información de las **partes del producto que son afectadas por la WU**.

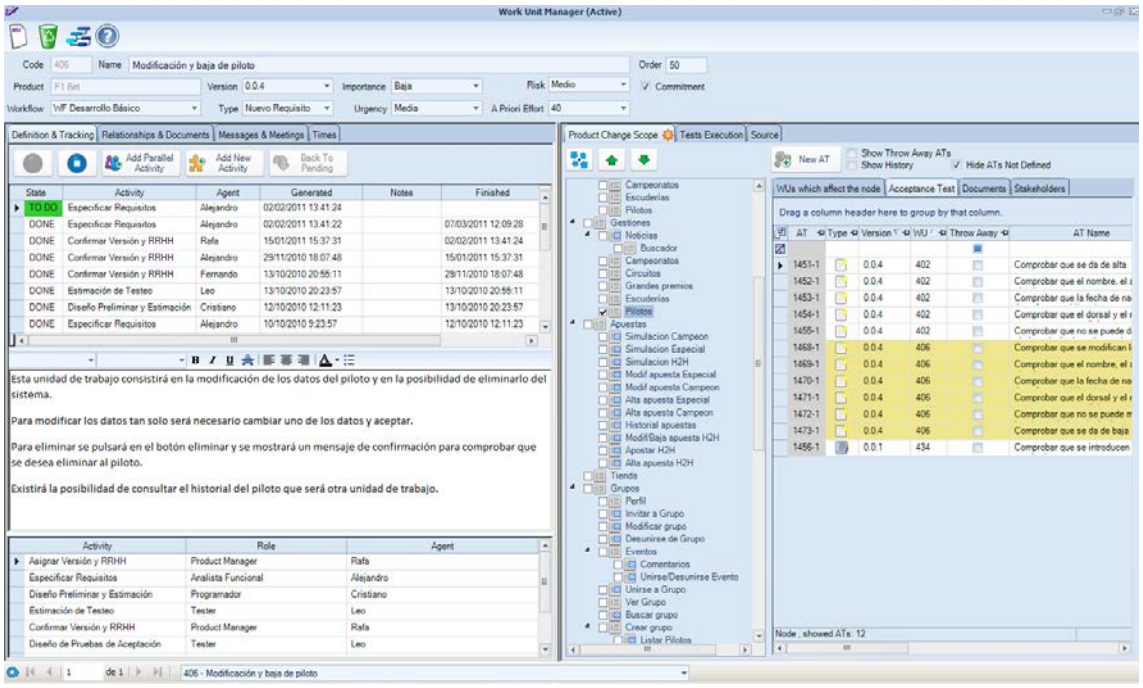


Figura 10. Work Unit Manager (WUM)

Requirements Manager (REM)

El **REM** es una de las innovaciones que ofrece TUNE-UP en cuanto a gestión del producto y sus requisitos. En TUNE-UP, el desarrollo y mantenimiento del producto está dirigido por las PAs que constituyen la especificación del comportamiento del producto. La estructura del producto es un **grafo acíclico dirigido** mostrado como un tree-view, como se muestra en la parte izquierda de la Figura 11. Cada nodo de esta estructura es un contenedor de comportamiento expresado como PAs. Así, al seleccionar un nodo, en la **pestaña 'WUs which affect the node'**, podemos conocer toda su historia en términos de WUs que lo han afectado, lo están afectando o lo afectarán en futuras versiones. De manera similar, en la **pestaña 'Acceptance Test'** se puede consultar las PAs que han sido definidas en dicho nodo y cómo han cambiado. Además, cada nodo en la **pestaña 'Documentación'** puede tener asociada documentación adicional en términos de ficheros de cualquier tipo. Finalmente cada nodo en la **pestaña 'Stakeholders'** puede tener especificados los stakeholders para los

cuales dicho nodo es relevante. A lo largo de este trabajo se explicará en más detalle este módulo ya que es esencial en la propuesta que se presenta en este trabajo.

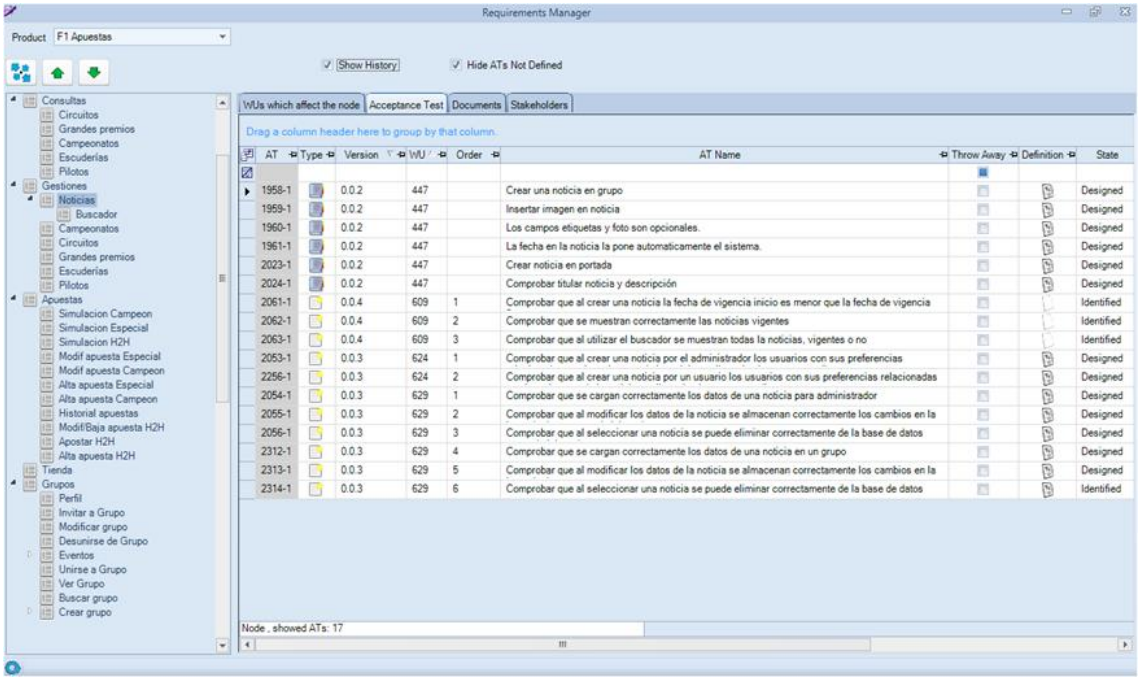


Figura 11. Requirements Manager (REM)

Version Contents & Tracking (VCT)

En este módulo encontramos facilidades para la planificación y seguimiento de las versiones. En la Figura 12 se muestra el contenido de **la pestaña 'Agent Workload'**, en ella se puede conocer en cualquier momento la holgura simple de los agentes respecto de sus actividades en una versión del producto. En esta interfaz se ofrecen potentes mecanismos de filtros y agrupaciones por columnas. Cuando una versión tiene problemas de holgura, en esta misma interfaz el **Product Manager** puede cambiar el agente asignado a la actividad (para balancear la carga de un agente) o cambiar de versión alguna unidad de trabajo. Otras alternativas son modificar la fecha de término de la versión, asignar más recursos humanos a la versión o dividir unidades de trabajo para realizarlas incrementalmente en varias versiones. Además disponemos de otras pestañas que tienen gran utilidad a la hora de planificar una versión como **'Relationships'**, **'WUs in Version'** y **'Affected Requirements'**.

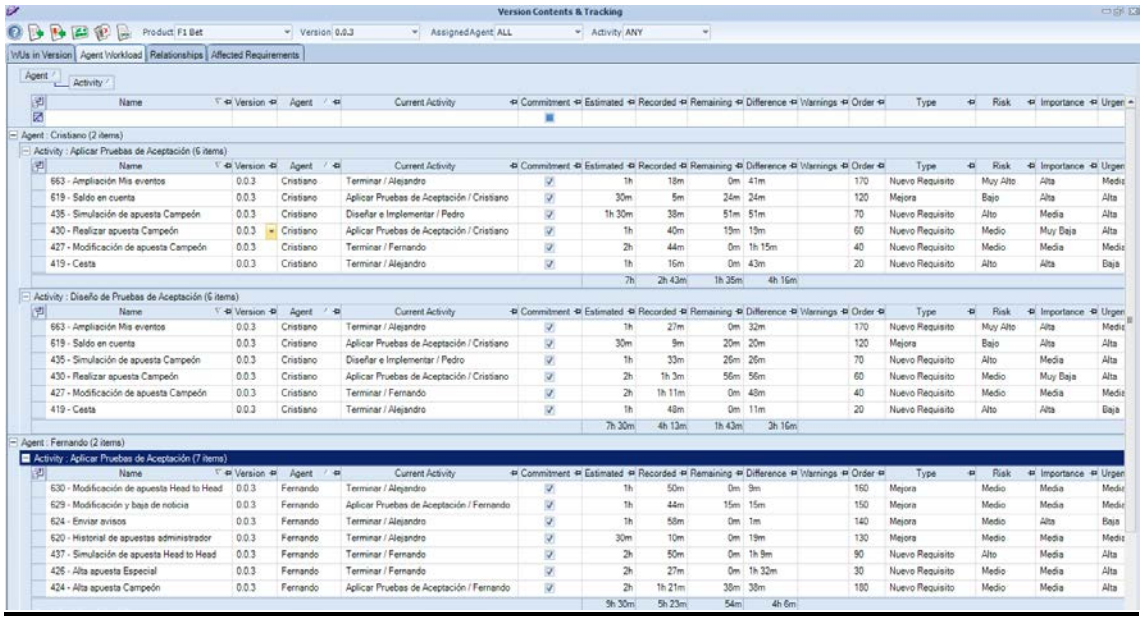


Figura 12. Versión Contents & Tracking (VCT)

3.3. Seguimiento de una iteración con TUNE-UP

En la actualidad, en TUNE-UP el seguimiento de una iteración incluye los siguientes mecanismos de apoyo:

- **Panel Kanban** (parte de la Figura 9) en el cual se sintetizan todas las actividades de los workflows en las cuales cada miembro del equipo tiene trabajo asignado. En el Kanban se puede visualizar en qué actividades se encuentran las WUs de una iteración.

- **Módulo *Version Contents and Tracking* (VCT)** (Figura 12). En este módulo se ofrecen varias vistas del contenido y estado de las WUs en una iteración.

- **Alertas y notificaciones a los miembros del equipo** (lateral inferior izquierdo figura 9). Automáticamente se generan alertas y notificaciones ante ciertos eventos, por ejemplo, cuando el esfuerzo invertido sobrepasa el esfuerzo estimado en una actividad de una WU, cuando se sobrepasa el tiempo de postergación definido para una actividad, cuando se cambia de versión una WU, etc.

- **La Gráfica Burn Down** es un mecanismo protagonista en el seguimiento de la iteración en TUNE-UP, el cual explicaremos en detalle a continuación.

- **La Gráfica WUs Finished/Unfinished** (lateral inferior izquierdo de la figura 13) muestra por día las WUs que tenemos finalizadas y las que no están finalizadas que cumplen los filtros que se encuentran en la parte superior de la figura 13.
- **La gráfica de estado de PAs** (lateral inferior derecho de la figura 13) ilustra el estado de las PAs para cada nivel de testeo de las WUs que cumplen los filtros que se encuentran en la parte superior de la Figura 13.

TUNE-UP ofrece una **Gráfica Burn Down** para cada iteración de un producto, junto con una tabla llamada **Daily Events** con información complementaria para la correcta interpretación de la gráfica (esta tabla se explica en detalle más adelante).

Figura 13. Dashboard asociado a un producto

La **Gráfica Burn Down de TUNE-UP** (lateral superior izquierdo de la Figura 13) incluye dos gráficas en una; una gráfica básica **Burn Down** (asociada a la línea serpenteante descendente) y una gráfica **Burn Up** (asociada a la línea ascendente representando el esfuerzo invertido). Además, se incluye una línea que representa el

esfuerzo estimado y una línea diagonal que representa el **esfuerzo restante** de referencia. Gracias a disponer en una misma gráfica de los esfuerzos estimados, invertidos y restantes, se facilita la interpretación del estado de la iteración y cómo se ha ido desarrollando. Situaciones tales como una bajada o subida pronunciada del esfuerzo restante pueden visualmente explicarse por una correspondiente bajada o subida en la línea de esfuerzo estimado, o bien en una subida o bajada en la línea de esfuerzo invertido. Sin embargo, la confirmación de estas interpretaciones, como veremos a continuación, exige contar con la información detallada de los eventos que pueden haber ocurrido entre dos puntos consecutivos de la gráfica. El esfuerzo restante de referencia corresponde a la línea que se traza desde el punto de mayor esfuerzo restante hacia el punto de esfuerzo restante 0 en el día de fin de la iteración. Además, asociada a esta línea se muestra en la parte inferior de la gráfica la velocidad requerida para conseguir la tendencia ilustrada por ese esfuerzo restante de referencia. Toda la información de la Gráfica Burn Down puede ser filtrada por Actividad, WU y/o Miembro del equipo.

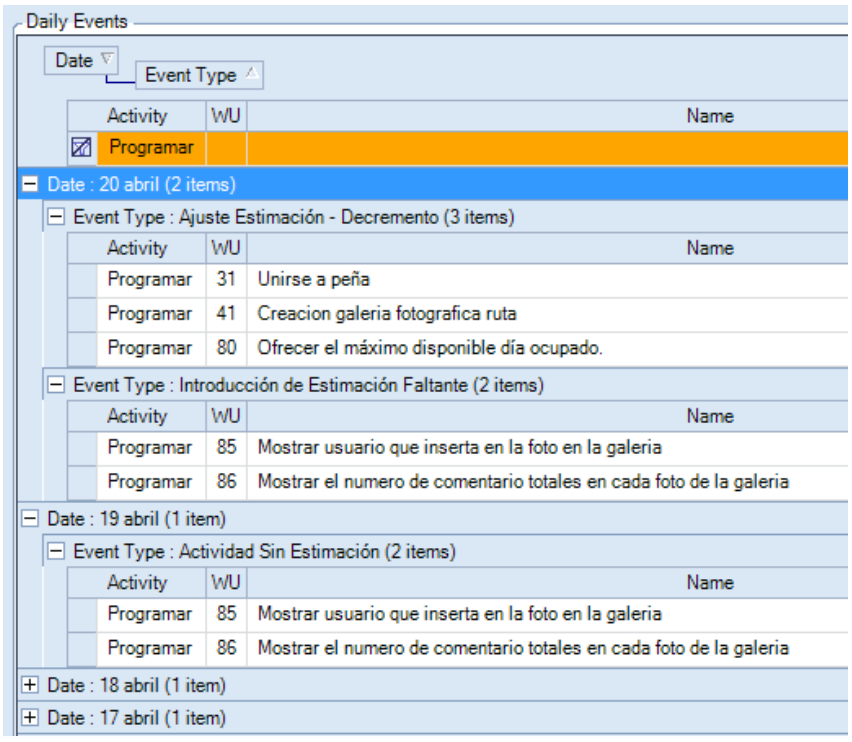


Figura 14. Tabla Daily Events

La **Tabla Daily Events** que muestra la Figura 14, contiene los eventos diarios que permiten interpretar correctamente la Gráfica Burn Down. Haciendo clic en un punto de la gráfica de la Figura 13 se despliega en la tabla de la Figura 14 la lista de eventos

ocurridos entre el día previo y el día seleccionado. Para cada evento se indica el miembro del equipo, la actividad e información de la WU donde se produce. En TUNE-UP se supervisan todos los eventos que pueden influir en la correcta interpretación del esfuerzo restante, dichos eventos se describen a continuación:

Eventos que invalidan la lectura del esfuerzo restante:

- **Actividad con estimación sobrepasada**: el esfuerzo invertido por el miembro del equipo en la actividad sobrepasa el estimado (lo cual llevaría a un esfuerzo restante negativo). El miembro del equipo asignado debería re-estimar
- **Actividad sin estimación**: la actividad no está estimada o su valor es 0. El miembro del equipo asignado debería estimar

Eventos que provocan una variación en el esfuerzo restante observado:

- **Cambios del esfuerzo invertido**: el esfuerzo invertido se ha modificado. Por ejemplo, se había registrado 10 horas de trabajo cuando realmente debían de ser 5 horas. También en casos en los cuales no se registra el esfuerzo en el momento, posteriormente es posible registrarlo
- **Incremento en la estimación**
- **Decremento en la estimación**
- **Introducción de estimación faltante**: indica que se ha estimado una actividad que el día anterior no tenía estimación
- **Actividad asignada a un miembro específico del equipo**: una nueva actividad de una WU ha sido asignada a un miembro. Esto es sólo relevante cuando se trata de la Gráfica Burn Down filtrada con un miembro del equipo específico
- **Actividad desasignada de un miembro específico del equipo**: una actividad de una WU ha sido desasignada de un miembro. Esto es sólo relevante cuando se trata de la Gráfica Burn Down filtrada con un miembro específico
- **WU nueva**: WU creada y añadida a la iteración
- **WU eliminada**

- **WU desestimada**. Su esfuerzo restante se considera igual a 0
- **WU añadida**: WU que se ha añadido a la iteración (ya existía sin iteración asignada o en otra iteración)
- **WU quitada**: WU que estaba el día anterior en la iteración y se ha cambiado de ésta

Para ilustrar el uso de estos eventos en la interpretación de la gráfica, a continuación comentamos un ejemplo. La Figura 13 muestra la Gráfica Burn Down correspondiente a la actividad Programación de la versión 0.2 de un determinado producto. En la gráfica, el día 20 de Abril se observa un descenso del esfuerzo restante. Este descenso, a priori lo podemos asociar al descenso del esfuerzo estimado. Pero cuestiones como: *¿por qué ha descendido el esfuerzo estimado?, ¿se ha movido, eliminado o desestimado trabajo?, ¿algún miembro del equipo ha ajustado alguna estimación?*, no se pueden responder a simple vista. Para responder tales cuestiones es esencial la información de **la tabla *Daily Events***. En la Figura 14 vemos los eventos asociados a la actividad Programación que ocurrieron el día 20 de Abril; hubo un decremento en la estimación de la actividad Programación en 3 WUs y se han introducido 2 nuevas estimaciones que faltaban (en la Figura 14 vemos que el día 19 de Abril faltaba por estimar la actividad Programación en dichas WUs). De esta forma sabemos que el descenso del esfuerzo restante se debe a un decremento en la estimación de 3 WUs, aunque además se hayan incluido 2 nuevas estimaciones antes no consideradas.

Finalmente, el disponer de información detallada de lo realizado durante cada iteración, aporta información útil para **reuniones de revisión de la iteración** o **reuniones de retrospectiva**, pudiendo llegar a evaluar acciones de mejora en el proceso mediante la comparación de datos de diferentes iteraciones y su tendencia.

Capítulo 4. TDRE en TUNE-UP

El resumen de TDRE en Tune-Up presentado en este capítulo se ha elaborado a partir del contenido de la Tesina de María Company Bria) [1]

Tune-Up presta atención tanto al desarrollo inicial del producto como a su mantenimiento (**"Todo producto exitoso necesitará de mantenimiento**"). Los requisitos en TUNE-UP son especificados mediante PAs. En TUNE-UP el cliente con la ayuda del analista es el encargado de definir las WUs en términos de PAs. Posteriormente el programador escribe código para satisfacer las PAs, y por último, el Tester establece combinaciones de datos para cada una de las PAs y las aplica.

En TUNE-UP la estructura de requisitos se representa como un grafo acíclico dirigido cuyos nodos son contenedores de PAs. Como se muestra en la Figura 15 un cambio en el comportamiento del producto viene dado por una WU, la cual afecta a uno o más nodos de la estructura de requisitos del producto, añadiendo, modificando o eliminando PAs.

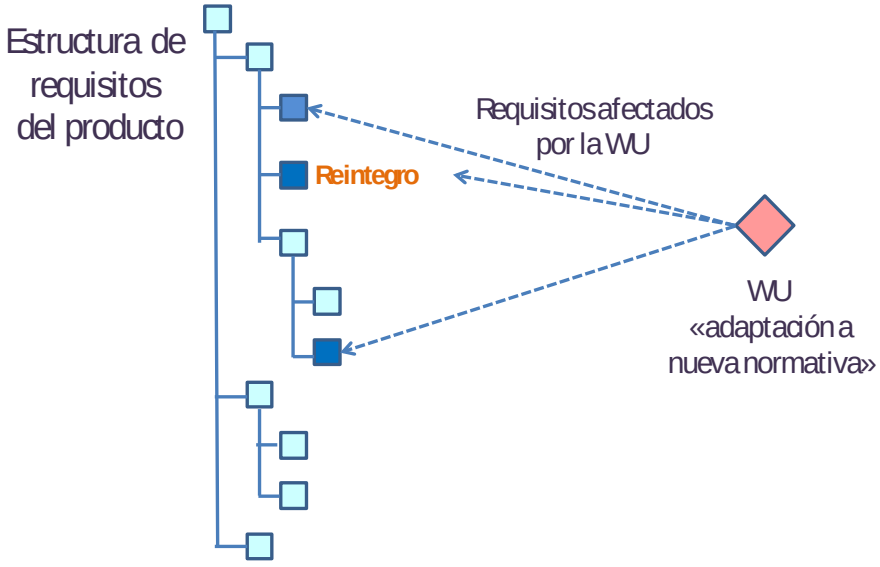


Figura 15. Estructura de Requisitos del producto

A continuación ilustraremos cómo TDRE se ha integrado en TUNE-UP Software Process mediante el módulo específico denominado REM.

4.1. Proceso dirigido por las Pruebas de Aceptación

El primer paso a realizar cuando el equipo de desarrollo o el cliente quieren hacer algún cambio de comportamiento dentro del producto es crear la WU correspondiente. Las

WUs se pueden crear desde diferentes sitios en TUNE-UP (PEP, WUM y REM). El contenido del REM es muy similar al de la **pestaña Product Change Scope** dentro del WUM, la única diferencia es que el REM está orientado a consultar el comportamiento pasado, actual y previsto del producto sin estar en el contexto de una unidad de trabajo determinada. Esta información es muy útil en el momento de proponer un nuevo cambio en el producto para por ejemplo, evitar solapes e inconsistencias entre cambios o programar de forma racional futuros cambios del producto prestando atención al conjunto de cambios pendientes de implementar.

El REM tiene dos modos de trabajo: **Standard Mode**, el cual es el modo por defecto al acceder al REM, y el **New WU Preparation Mode**, el modo en el cual se permite crear una nueva WU directamente desde el REM. En este último modo se puede hacer todo lo ofrecido en el modo estándar, pero adicionalmente, se muestran checks en los nodos del grafo para marcar los nodos afectados por la WU que posteriormente se creará. Así, resulta integrada la tarea de verificar el comportamiento ya definido o por definir en los nodos en los cuales se pretende llevar a cabo un cambio, con respecto del hecho de indicar los nodos que se verán afectados.

Con el botón **New WU** se abrirá el formulario de creación de una WU (formulario que está sobrepuesto en la figura 16). La WU creada desde el REM mantendrá las marcas que se hagan en el REM, pudiendo posteriormente el analista refinarlas según se requiera. El trabajo de definir detalladamente el cambio mediante PAs se hará posteriormente por el Analista.

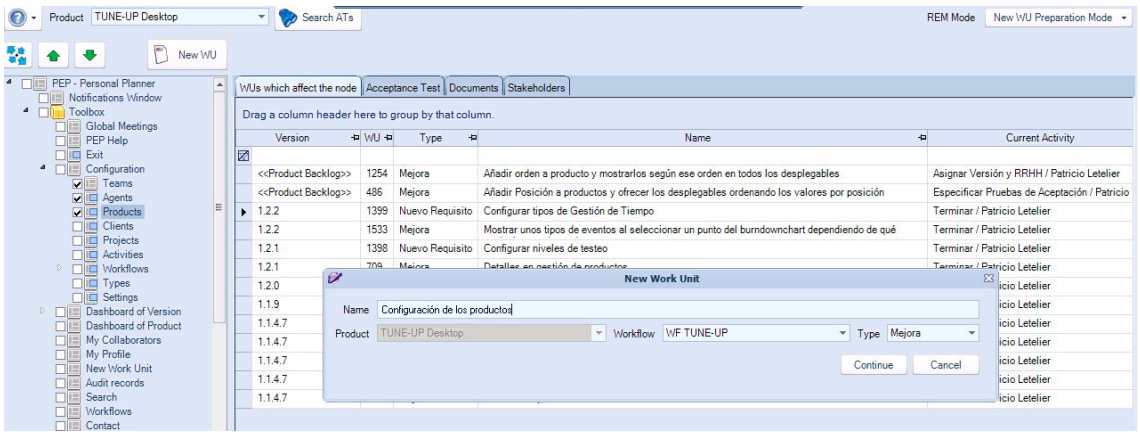


Figura 16. Creación de una WU desde el REM

En TUNE-UP el módulo WUM apoya a los agentes en sus actividades sobre una determinada unidad de trabajo. Desde este módulo y para cada unidad de trabajo, se

puede acceder a la **pestaña Product Change Scope** (ver figura 16), donde el analista revisaría las marcas de los nodos que se verán afectados y si es necesario tendría que modificarlas (podría darse el caso que el agente que ha creado la WU no conociera suficientemente la estructura de requisitos y se haya dejado nodos por marcar o haya puesto las marcas en niveles superiores). Una vez las marcas ya sean las correctas el analista se encargaría de definir los cambios en las PAs de dichos nodos.

Para el equipo de desarrollo es muy importante conocer qué nodos afecta cada unidad de trabajo. Esta información permite detectar posibles conflictos o solapes entre las unidades de trabajo contenidas en la misma versión o entre unidades de trabajo de diferentes versiones. Por ejemplo, si un analista está definiendo un cambio de comportamiento en un nodo, le interesará conocer cómo ha evolucionado su comportamiento (conocer las unidades de trabajo realizadas en el nodo con sus correspondientes cambios en PAs), el comportamiento actual del nodo (las PAs vigentes en el nodo) y los cambios pendientes en el nodo (unidades de trabajo pendientes que afectarán al nodo, con sus correspondiente cambios propuestos para sus PAs).

En el lateral izquierdo de la Figura 17 se observa la estructura de requisitos del producto (desplegada de forma parcial), donde se ven marcados los nodos afectados en la WU. Los nodos de la estructura de requisitos pueden ser de diferente tipo y se diferencian por el icono que tienen delante: (Nodo que representa un formulario o una página), (Nodo que representa un elemento) y (Nodo abstracto).

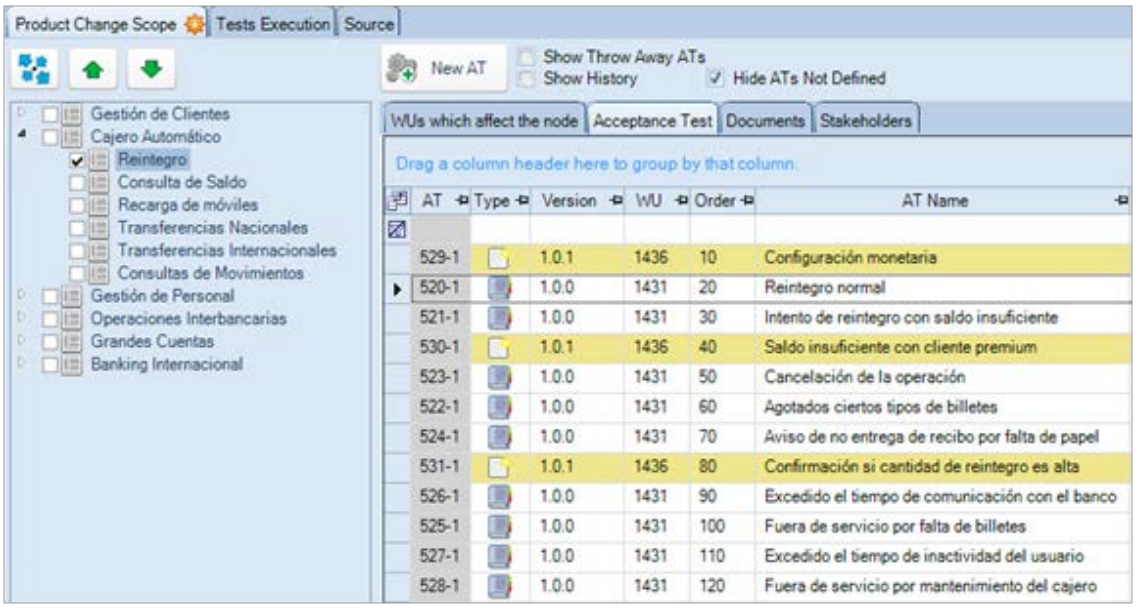


Figura 17. Pestaña Product Change Scope

En la **pestaña Acceptance Test** de la Figura 17 se observa la lista de PAs del nodo seleccionado (Nodo Usuario) en la estructura de requisitos. En este listado se muestran las PAs definidas o modificadas en esta WUM (marcadas en amarillo), aquellas que se implementaran en otras WUS y las que representan el comportamiento actual del nodo. También en cualquier momento si marcamos el check ***"Show History"*** se mostrarán todas las PAs del nodo seleccionado, incluyendo aquellas versiones anteriores de una PA (pruebas que representen comportamiento pasado y están representadas con el icono).

Las consecuencias de la implantación de una unidad de trabajo se refleja claramente por los cambios en las pruebas del nodo: **PA nueva** (), **PA eliminada** () y **PA modificada** (). Además, las PAs que tienen el icono son **pruebas actuales** del nodo e indican comportamiento ya existente que se mantiene. Las pruebas existentes pueden marcarse por el analista o por otros agentes que participan en el proceso para que sean aplicadas como pruebas de regresión, de esta forma se asegura que el cambio no afecta dicho comportamiento. Esto último es útil para poder acotar el esfuerzo o tiempo requerido para aplicar las pruebas de regresión, pues incluso aunque dichas pruebas estuviesen automatizadas, no siempre es factible aplicarlas todas en el momento específico que lo requiere (por ejemplo, cada vez que el programador publica sus cambios de código, termina la implementación de una unidad de trabajo, cuando se termina una versión, etc.).

Los checks ***"Hide Ats Not Defined"*** y "***Show Throw Away ATs"*** (parte superior de la figura 17) permiten filtrar las PAs mostradas en la pestaña Acceptance Tests. La **casilla Hide ATs Not Defined** oculta las **PAs ficticias**, las cuales se muestran para destacar que en ciertas WUs no se han definido PAs, con lo cual el cambio asociado no está explícito ni completo en las PAs existentes del nodo seleccionado. Serán WUs que al crearse han marcado los nodos que se van a ver afectados, pero aún no ha llegado a la actividad Analizar Incidencia y no se ha especificado el cambio mediante PAs. Las pruebas Ficticias se diferencian por el icono y porque tienen como nombre "ATs NOT DEFINED". La **casilla Show Throw Away ATs** muestra en el grid las **PAs Throw Away,** las cuales expresan comportamiento que no debe presentarse, están asociadas a correcciones de fallos y no representan el comportamiento actual del nodo (Ejemplo Figura 18).

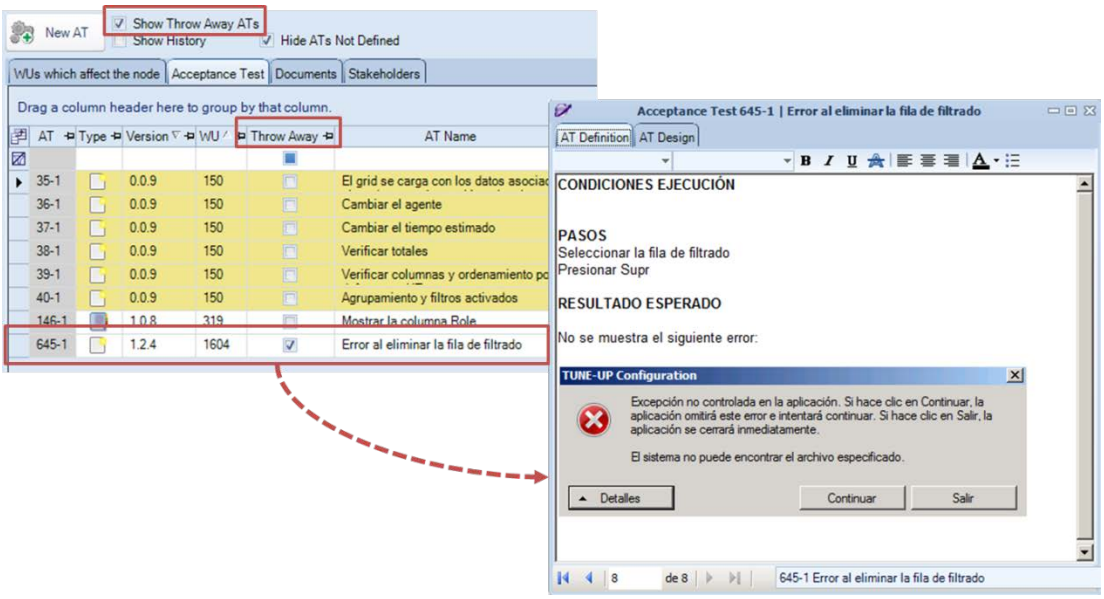


Figura 18. Ejemplo Prueba Throw Away

Para ver los detalles de una prueba al hacer doble click sobre el listado de PAs se accede al formulario de la prueba que se muestra en la figura 19.

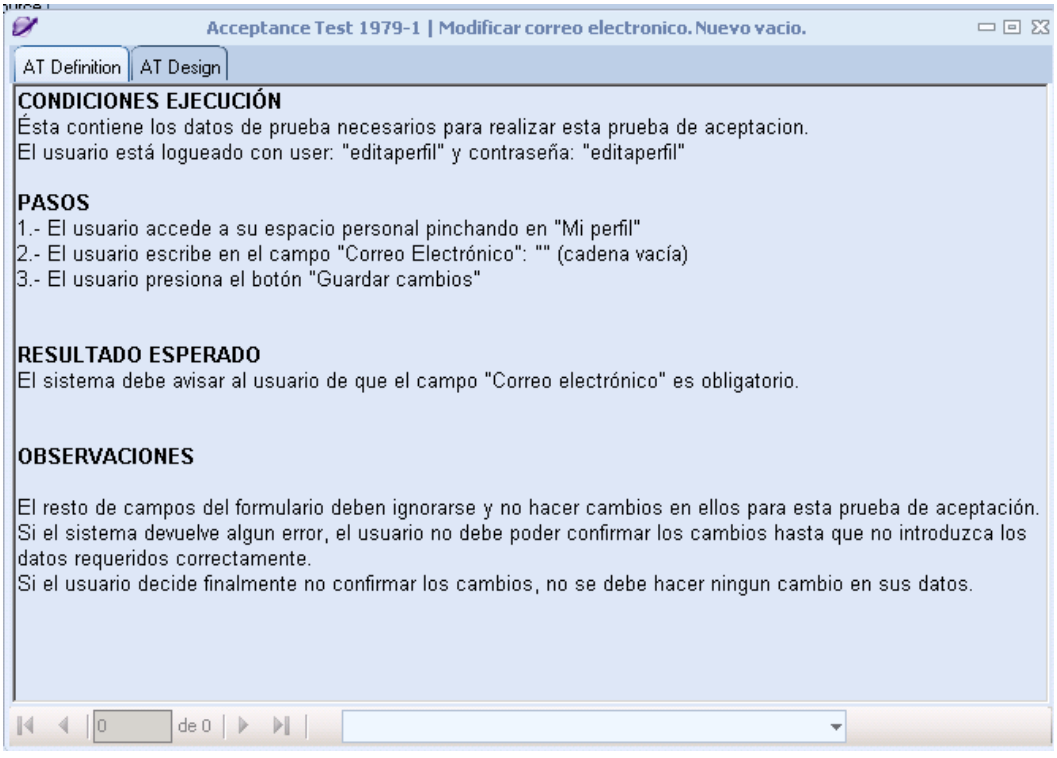


Figura 19. Formulario de Acceptance Test

En la pestaña **WUs which affect the node** de la figura 17 se pueden observar las WUs que han afectado, están afectando o afectarán al nodo seleccionado en la estructura del producto. Para cada WU se muestra la versión en la que se incluyó la WU, código, tipo,

nombre, actividad actual en la que se encuentra dentro del workflow y los agentes asignados.

También se dispone de otra pestaña **Documents** como espacio de documentos asociados al nodo, por ejemplo, modelos u otro tipo de especificación complementaria que se desee almacenar. Finalmente, es posible en la pestaña **Stakeholders** indicar los tipos de usuario u otros interesados en los servicios que ofrecerá dicha parte de la aplicación.

TUNE-UP permite además explotar la información de las PAs en el contexto de una unidad de trabajo desde la perspectiva del resto de los agentes que colaboran en su realización. TUNE-UP permite configurar los niveles de testeo que van aplicar las PAs, esto normalmente dependerá de los recursos de la empresa, en los siguientes ejemplos vamos a suponer que sólo disponemos de dos niveles de testeo: Programación y Testeo. Cada nivel tiene asociada información de seguimiento de las PAs, así cada participante (programador, tester) puede registrar sus ejecuciones de cada PA.

En la **pestaña Test Execution del WUM** (Figura 20) se muestra el listado de PAs definidas, modificadas o marcadas de regresión que el analista ha definido. En el grid se destaca el estado de aplicación de la PA, registrado por los agentes en los diferentes niveles de testeo.

Dichos **estados de aplicación de la PA** son:

- ***Vacío*:** la prueba no se ha aplicado
- : la última aplicación de la prueba ha sido exitosa
- : la última aplicación de la prueba ha detectado defectos
- **:** la prueba ha sido aplicada con éxito pero está marcada como pendiente de volver a aplicar, por ejemplo porque se ha modificado la PA o porque se sospecha que otra prueba posteriormente implementada podría haberla afectado
- **:** la última aplicación de la prueba ha sido exitosa pero existe una aplicación en un nivel de testeo posterior (aplicación realizada por otro agente) en la cual se han detectado defectos, con lo cual una vez resueltos, debería volverse a aplicar la PA

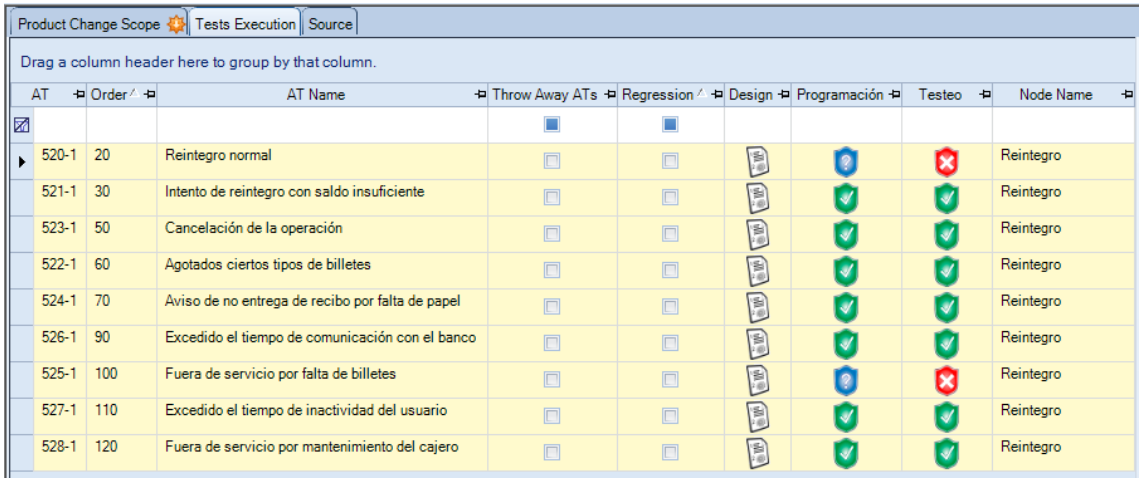


Figura 20. Pestaña Test Execution

Cada nivel de testeo crea sus propios registros de aplicación de pruebas constituyendo un histórico de las ejecuciones de una PA (parte inferior de la figura 21). Por ejemplo, cuando un programador implementa y aplica una PA, registra un seguimiento OK para indicar que se ha superado con éxito dicha PA. Un seguimiento está formado por la WU, Fecha de ejecución, Agente, Nivel de testeo, el Resultado (OK, KO y OK!), un Fichero explicativo sobre el resultado (conteniendo instrucciones para la reproducción de los defectos detectados) y un apartado para añadir comentarios.

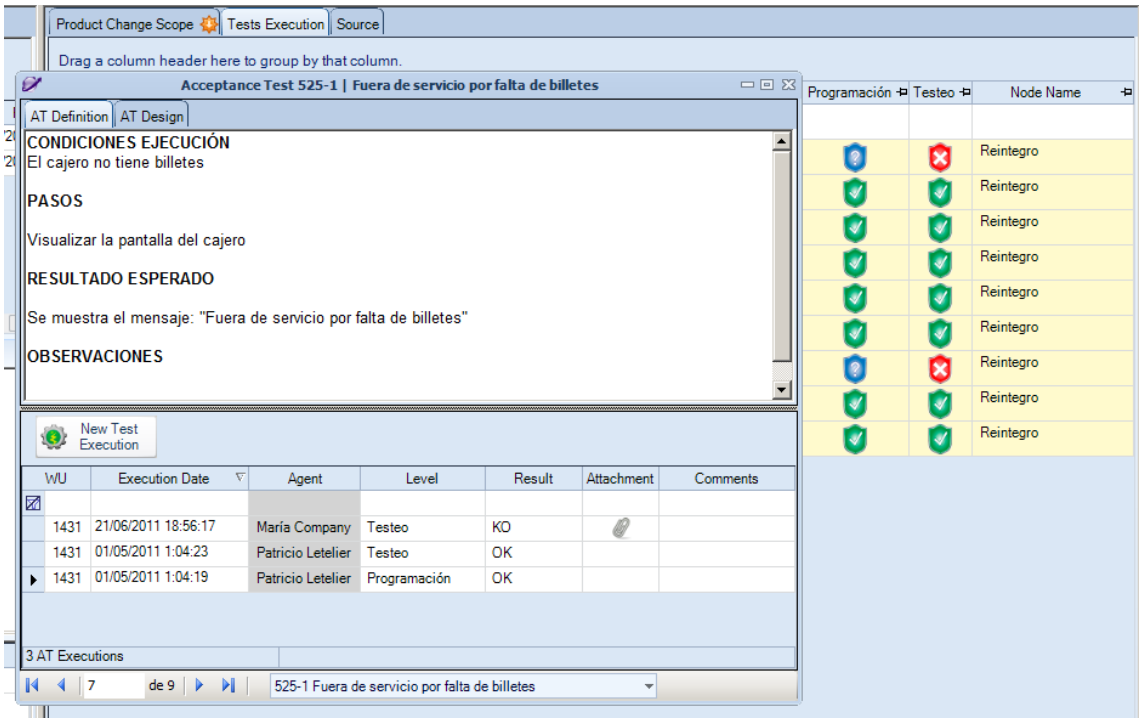


Figura 21. Ejecución de la PA

4.2. Pruebas de Aceptación y referencias a entidades

Una PA tiene como propósito demostrar al cliente el cumplimiento parcial o total de un requisito del software. Una PA describe un escenario de ejecución o de uso del sistema desde la perspectiva del cliente. Un requisito puede contener una o más PAs y éstas pueden estar asociadas tanto a requisitos funcionales como a no funcionales. A continuación se presenta como ejemplo la definición de la PA "Intento de reintegro con saldo insuficiente" (se trata del contexto de la funcionalidad Reintegro en un cajero automático).

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

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

4.3. Seguimiento de una iteración con TUNE-UP mediante PAs

En TUNE-UP, para realizar la planificación y seguimiento de las iteraciones, el jefe de proyecto utiliza el **módulo Version Content & Tracking**. La Figura 22 muestra parte de la **interfaz llamada "Version Content",** la cual contiene la lista de unidades de trabajo de la versión de un producto. El jefe de proyecto puede consultar los datos resumidos de cada unidad de trabajo en la versión, en particular, puede conocer la actividad actual en que se encuentra dentro del workflow, su orden dentro de la versión, el esfuerzo que implica elaborar la unidad de trabajo, el agente asignado a las actividades principales del workflow, etc. Además, por cada nivel de testeo existirá una columna que refleja el estado de aplicación de las PAs de cada unidad de trabajo, mostrando el porcentaje obtenido a partir del número de pruebas con estado de aplicación OK dividido por el total de PAs. Con toda esta información el jefe de proyecto tiene una visión más detallada del estado de avance de la unidad de trabajo. Esta vista es muy reveladora, lo normal es que los porcentajes de cada nivel se vayan

completando de forma secuencial, puesto que la unidad de trabajo va pasando por diferentes actividades realizadas por agentes diferentes. Sin embargo, cuando aparecen las inevitables situaciones de re-trabajo, éstas quedan en evidencia por la forma incompleta que presentan los porcentajes de pruebas OK en cada nivel de testeo. La línea subrayada en la figura 22 es un caso frecuente de retrabajo generado por saltos atrás en el proceso, en este caso en concreto el tester ha empezado a aplicar las PAs de la WU pero ha detectado fallos (ha marcado una prueba con KO) y por lo tanto le ha devuelto al programador la WU a la actividad Diseñar e Implementar para que solucione la prueba que tiene marcadas con interrogante.

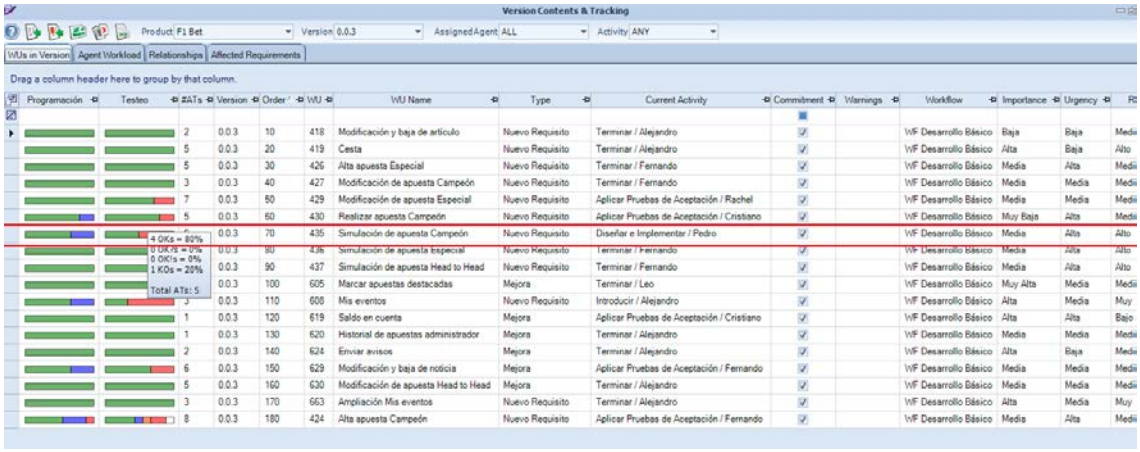


Figura 22. Pestaña WUs in Version del VCT

Para el jefe de proyecto no basta con conocer la actividad actual de la unidad de trabajo, puesto que además se puede estar realizando trabajo en paralelo respecto de la corrección de defectos detectados. En la práctica, sólo cuando los defectos son muy graves se devuelve a una actividad previa la unidad de trabajo. Por lo general los defectos se informan a los responsables de su corrección mediante mensajes, pero se continúa con el trabajo de la actividad donde ha llegado la unidad de trabajo.

El jefe de proyecto también puede disponer del estado de avance de las PAs de una versión con la **gráfica de estado de PAs** que se encuentra en el Dashboard. En la gráfica de la Figura 23 la línea vertical representa el número de PAs de una WU y la línea horizontal tiene una barra para cada nivel de testeo de una WU. En el ejemplo de la figura sólo tenemos dos niveles de testeo, por lo tanto cada WU tiene representado el estado de las PAs en los niveles de programación y testeo. Esta vista ofrece una visión en forma de gráfica y por lo tanto más fácil de interpretar que las columnas Programación y Testeo de la figura 22. La única diferencia es que en VCT sólo se puede

ver el estado de PAs de WUs de una versión determinada y en la gráfica de estado de PAs se pueden configurar, a petición del usuario a través de filtros, las WUs que se quieren mostrar en la gráfica.

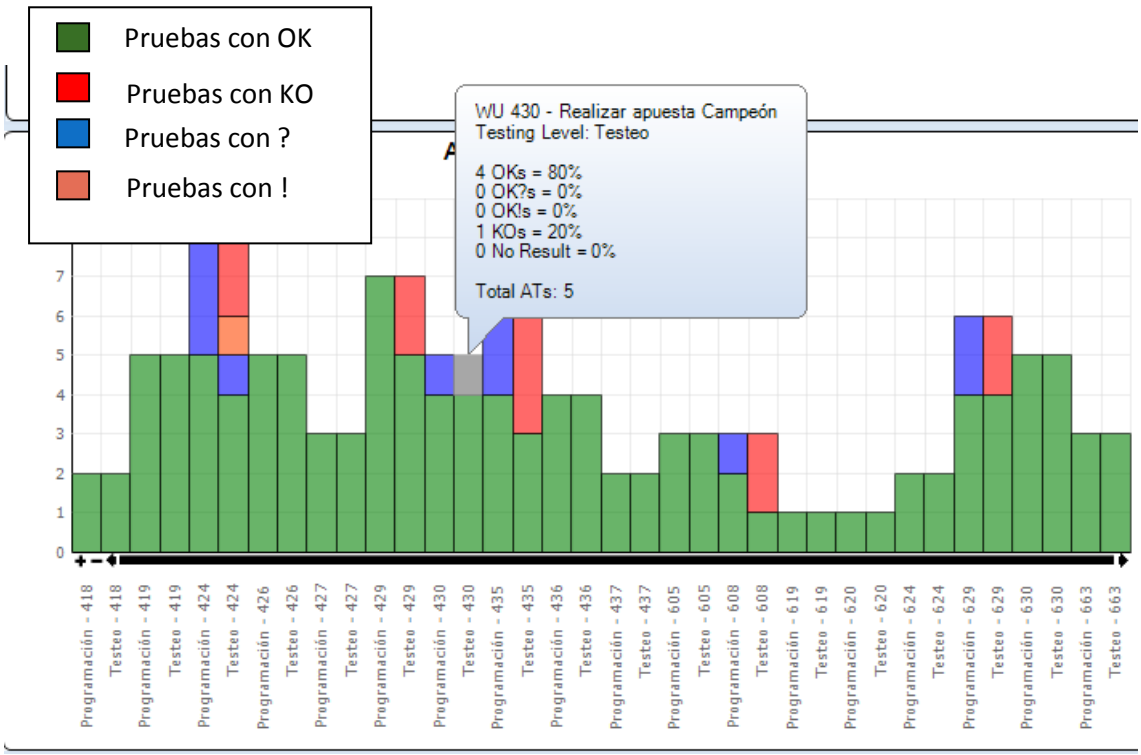


Figura 23. Gráfica de Estado de PAs

La pestaña **Affected Requirements** (Figura 24) es de gran utilidad para el jefe de proyecto ya que permite detectar posibles conflictos o solapes entre las WUs contenidas en la misma iteración o entre WUs de diferentes iteraciones. En la estructura del producto aparecen seleccionados los nodos que se ven afectados en incidencias de la versión y entre paréntesis delante del nombre del nodo el número de WUs de la versión que modifican su comportamiento. Al activar un nodo de la estructura de requisitos (panel de la izquierda), en el panel de la derecha se cargan las WUs que afectan a dicho nodo en la versión (WUs con fondo oscuro) y en otras versiones (WUs con fondo blanco). Esta información ofrece las siguientes ventajas:

- **Detectar conflictos entre WUs.** Hay otra WU que afecta a la misma parte del producto que la WU de la versión y sería conveniente que se hiciera en la misma versión o incluso antes.

- **Detectar solapes entre WUs.** Nos ofrece una ayuda para evitar que hayan WUs duplicadas.

- **Asignar los mismos agentes.** Asignar a las WUs los mismos agentes que están asignados a las otras WUs que afectan a la misma parte del producto (columna asignados), ya que conocen mejor esa parte del producto y, además si están en la misma versión las WUs se evitan problemas de protección de código.

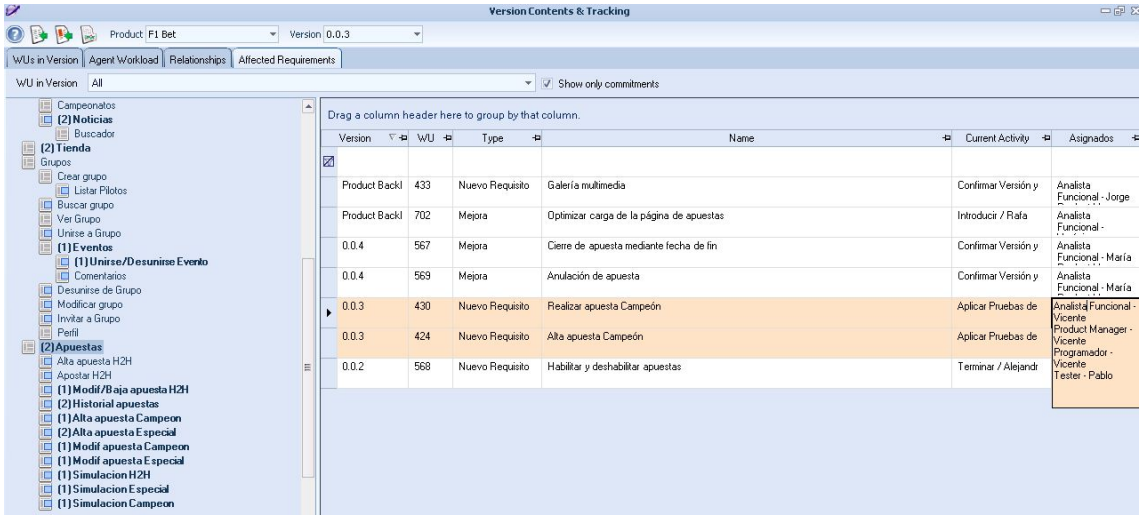


Figura 24. Pestaña Affected Requirements de VCT

4.4. Proceso de testeo en TUNE-UP

El papel de un **Tester** es asegurar que el nuevo comportamiento que se ha desarrollado o modificado en el producto se corresponde con el definido previamente por los analistas en las Pruebas de Aceptación correspondientes.

Así pues, en una iteración en el ciclo de desarrollo nos presentamos con un número determinado de WUs de las cuales los analistas deberán definir el nuevo comportamiento que conlleva esta nueva unidad de trabajo. En el momento en que el *Programador* le dé la señal al *Tester*, éste deberá preparar el entorno y realizar las pruebas oportunas al producto generado. Sin embargo, cuando el producto tiene una envergadura importante y tiene una vida que se contabiliza en años, es inevitable imaginarse la de cantidad de cambios que puede haber sufrido y que sufrirá. Es por eso, que con cada cambio de funcionalidad o nueva funcionalidad, el *Tester* deberá probar el comportamiento de esa funcionalidad, al mismo tiempo que verificará que la funcionalidad que era correcta antes, lo sigue siendo ahora.

Ante esta perspectiva es necesario buscar una solución para facilitar el testeo de regresión que se deberá repetir por cada iteración del ciclo de desarrollo del producto. Se optó por la automatización de pruebas de aceptación, donde se decidiría con anterioridad cuáles PA son interesantes para automatizar y crear tantas pruebas de sistema como sean necesarias para cubrir el comportamiento de la PA. Una prueba de sistema es una instanciación de la PA con datos concretos que automatizaremos y ejecutaremos sobre el producto a testear. Aparece un nuevo rol distinto en el ciclo de vida, el **Automation Tester**. El *Automation Tester* es una especialización del *Tester* y tiene que llevar a cabo diferentes tareas:

- Realizar la automatización.
- Aplicar las pruebas automatizadas.
- Mantenimiento de las pruebas automatizadas.

A continuación se va a pasar a explicar el trabajo que realiza el *Tester* durante una nueva iteración en el ciclo de desarrollo, en qué momento puede iniciar su trabajo y con qué herramientas se ayuda para llevar a cabo su cometido.

En TUNE-UP hay definido un Workflow General que se sigue en todas aquellas WUs creadas para su posterior desarrollo. En el Workflow General se distinguen tres fases:

- La primera transcurre en unas iteraciones previas, se introduce la WU, se realiza el análisis de la misma y se hace una estimación.
- La intermedia se basa en la revisión del análisis y transcurre la versión anterior a la asignada para desarrollar.
- La fase de desarrollo donde se implementan y se confirma el comportamiento descrito en las PAs.

Nos vamos a centrar en la parte de desarrollo, con especial atención en las actividades relacionadas con el testeo. Como se muestra en la Figura 25, el *Tester* deberá elaborar los diseños de las pruebas automatizadas (prueba de sistema), los diseños de las pruebas manuales, la preparación del entorno o entornos necesarios y realizar el testeo manual. Por otra parte, el *Automation Tester* debe automatizar las pruebas diseñadas por el *Tester* y aplicar las pruebas automatizadas.

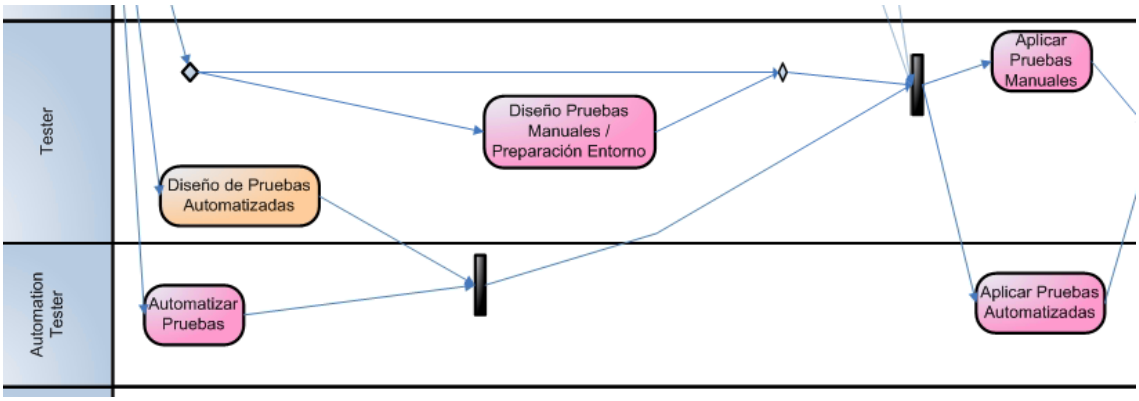


Figura 25. Vista testeo del Workflow General

Cuando se llega a la fase de desarrollo los analistas ya han definido los requisitos y las PAs que componen la WU, es en este punto donde los programadores comienza con el desarrollo de la nueva funcionalidad en paralelo con el trabajo de testeo. En los próximos puntos se va a describir en qué consiste cada actividad.

- **Diseño de Pruebas Automatizadas:**

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

  Para crear las nuevas PS el agente encargado deberá abrir el formulario de la PA y en la pestaña Pruebas de Sistema (Figura 26) podrá empezar su tarea.

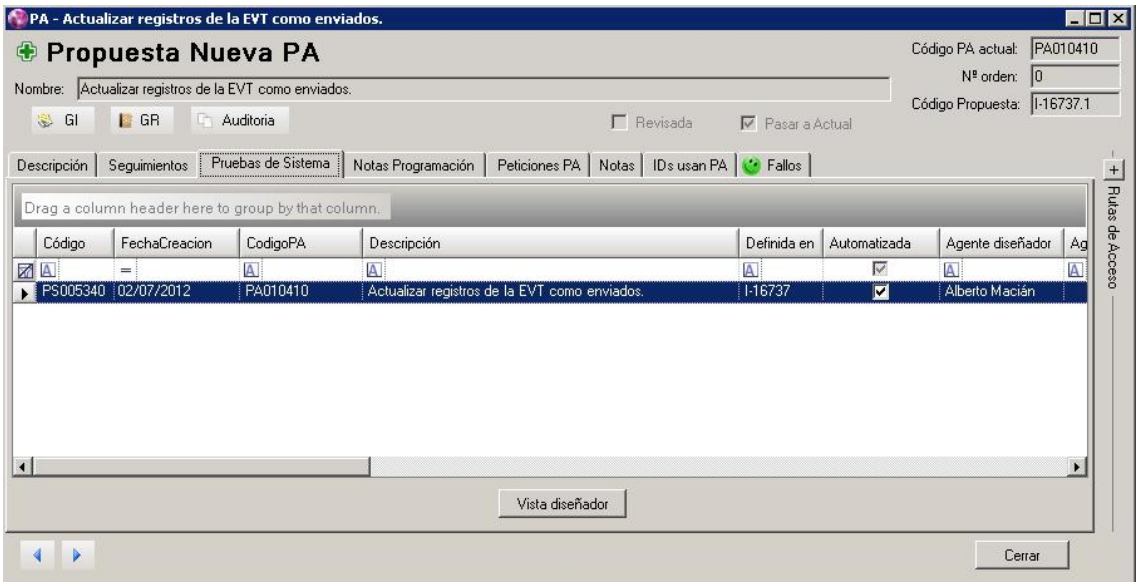


Figura 26. Pestaña Pruebas de Sistema en el formulario de la Prueba de Aceptación

Para esta actividad contamos con una herramienta de apoyo, denominada *Diseñador de Pruebas de Sistema* (Diseñador de PS), de la que hablaré más adelante describiendo cómo trabajar con él.

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

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

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

- **Aplicar Pruebas Manuales:**

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

Al mismo tiempo que vaya aplicando las pruebas al producto deberá marcar el resultado obtenido en la pestaña de *Test Execution* (figura 20), reportando, en aquellos casos que en la prueba se obtuvo un resultado incorrecto, el comportamiento incorrecto de la aplicación a los agentes correspondientes, además

de marcar el KO, se creará un fallo con la herramienta que se describirá en el próximo apartado, el *Gestor de fallos*.

- **Automatizar Pruebas:**

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

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

- **Aplicar Pruebas Automatizadas:**

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

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

A continuación, se va a explicar la funcionalidad de las herramientas de apoyo que tenemos para llevar a cabo el testeo.

Capítulo 5. Estado del arte

En este capítulo se van a presentar algunas de las soluciones que podemos encontrar en la actualidad que dan soporte al proceso de testeo. A continuación se va a presentar el estudio realizada sobre las herramientas *SilkCentral Test Manager* de Borland, *Rational Quality Manager* de IBM y *Visual Studio Team Foundation Server* de Microsoft. Después de realizar el estudio tendremos una idea del contenido que ofrecen los Dashboards de las herramientas y las distintas funciones de los mismos.

5.1. SilkCentral Test Manager

SilkCentral Test Manager de Borland es una solución de gestión de pruebas que aborda el proceso de testeo mediante la alineación de objetivos y requisitos de las pruebas, la tecnología y los procesos. Proporciona un marco integrado para la mejora de la productividad, la trazabilidad y visibilidad para todos los tipos de pruebas de software, y garantiza el control sobre la preparación de la aplicación.

En *SilkCentral Test Manager* las necesidades o peticiones del cliente vienen representadas por **Requisitos** que se insertarán en el sistema. Una vez se ha insertado el listado de requisitos, se elabora el **Plan de Pruebas** para verificar cómo de bien el proyecto cumple con los requisitos acordados. Un *Plan de Pruebas* se compone de tres elementos **Contenedor de Pruebas**, **Carpeta de Pruebas** y **Definición de Prueba**; el *Contenedor de Pruebas* se utiliza para identificar el Plan de Pruebas, cada Contenedor de Pruebas tiene varias *Carpetas de Pruebas*, que hace referencia a conceptos que se quieren satisfacer, como usabilidad, correctitud o eficiencia, en estas Carpetas de Pruebas se pueden asociar *Definiciones de Pruebas*, que pueden ser manuales o scripts desarrollados que se encargarán de verificar el comportamiento de los requisitos. Cuando ya se han definido los Requisitos y el Plan de Pruebas, se enlazan las *Definiciones de Pruebas* con los *Requisitos* disponibles. A parte, es posible crear **Ciclos de Pruebas**, que representan un periodo de tiempo en el que se asocian un conjunto de Testers y unas Definiciones de Pruebas que se deben pasar. Ahora, ya se pueden pasar las pruebas y revisar los resultados, si se encuentra un error podremos asociar **Problemas** a las Definiciones de Pruebas.

SilkCentral Test Manager ofrece un panel donde poder añadir los diferentes elementos disponibles. A continuación se van a presentar los diferentes elementos que podemos encontrar para elaborar el *Dashboard*.

Ciclo de vida de los problemas

Esta gráfica muestra los Problemas definidos en un producto; muestra la evolución de los Problemas junto con los estados que se muestran en la leyenda. El único filtro disponible en la gráfica es el *Producto*.

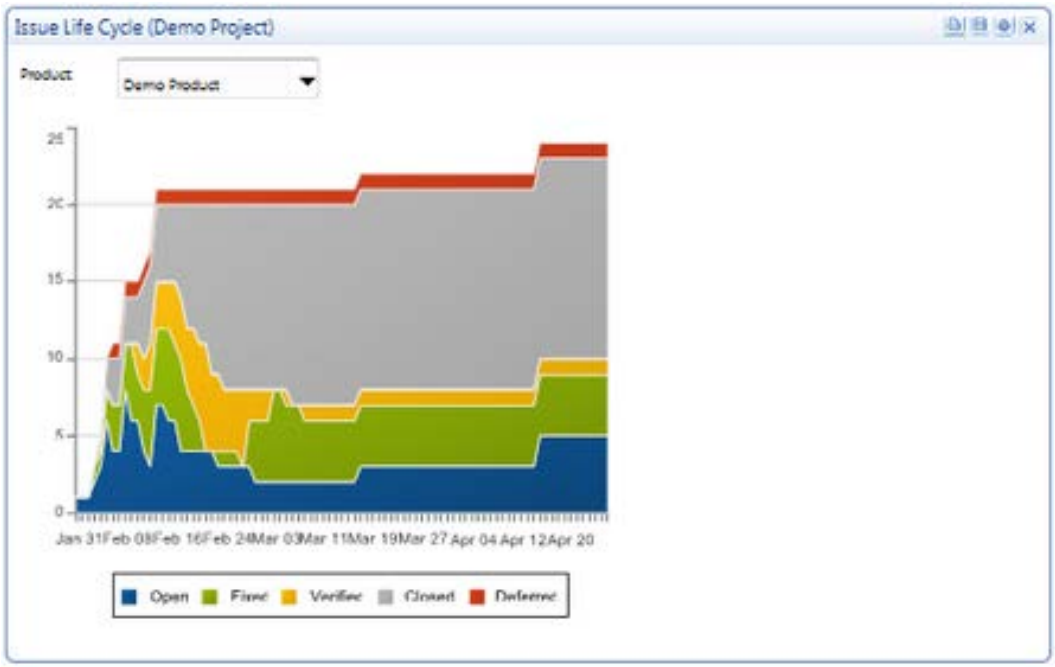


Figura 27. Ciclo de vida de los problemas

Se trata de un tipo de gráfica de áreas apiladas, donde el área de color azul son los problemas *Abiertos*, en verde los *Arreglados*, en amarillo los *Verificados*, en gris los *Cerrados* y en rojo los que se han *Desestimado*.

Con esta grafica podemos conocer la siguiente información:

- Cantidad de Problemas que se han introducido en un periodo de tiempo
- Rapidez con la que se resuelven los Problemas
- Frecuencia con la que se introducen

Pruebas manuales asignada a mí

Este grid muestra toda la información referente a las pruebas asignadas al agente para un proyecto en concreto.

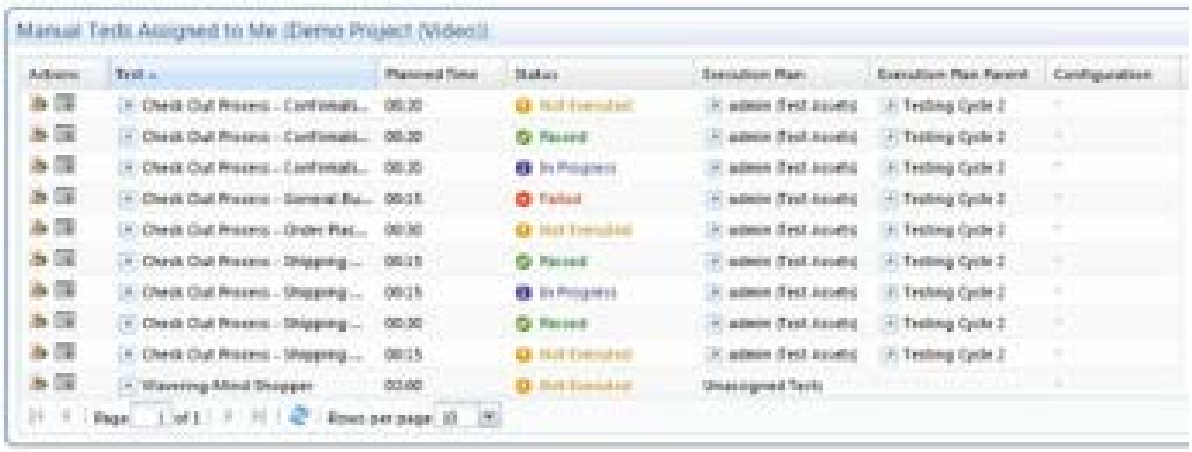


Figura 28. Pruebas manuales asignada a mí

Este grid permite visualizar el estado de las pruebas y la posibilidad de abrir las pruebas con la información detallada donde poder actualizar el estado de la prueba.

Con este grid podemos conocer la siguiente información:

- Cantidad de Pruebas Manuales que tiene asignados el agente
- En qué estado se encuentran las Pruebas
- A qué ciclo de pruebas pertenecen

Progreso de objetivos de calidad

En SilkCentral Test Manager existen unos Objetivos de calidad; cada uno tendrá asignado unas Pruebas con las que se tendrá conocimiento de que se está cumpliendo con esos objetivos.

Esta gráfica muestra el número de pruebas que se deben ejecutar para satisfacer unos objetivos de calidad junto con el número de pruebas que se han ejecutado hasta el momento. En este ejemplo se estudia el riesgo que presenta el estado actual de las pruebas; hay cuatro niveles de riesgo, Bajo, Medio, Alto y Crítico, para cada uno se asigna un número mínimo de pruebas que deben pasarse correctamente para disminuir los riesgos. En este ejemplo, existe un elevado riesgo ya que no se han cubierto los el porcentaje de pruebas pasadas en ninguno de los niveles.

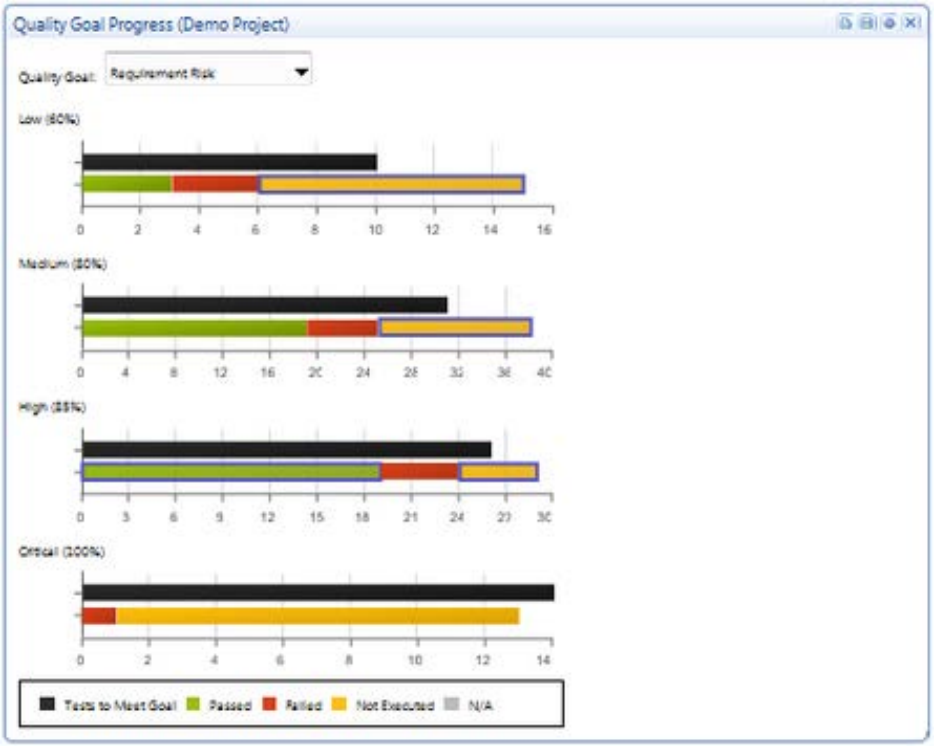


Figura 29. Progreso de objetivos de calidad

En negro tenemos el número de pruebas que se deben ejecutar para satisfacer el objetivo, en verde son las pruebas que se han pasado correctamente, en rojo las que han fallado, en amarillos las que aun no se han ejecutado y en gris aquellas que no se tiene información para clasificarlas en ninguna de las otras.

Con esta gráfica podemos conocer la siguiente información:

- Qué cantidad de pruebas se deben pasar para disminuir el riesgo de introducir Problemas que puedan ser importantes
- El estado de las pruebas que nos muestran si se cumple con el objetivo

Estado cobertura de requisitos

Esta grafica muestra si los requisitos son cubiertos por las pruebas manuales o automatizadas. Las pruebas se enlazan con los requisitos; de esta manera se obtiene el resultado de la gráfica.

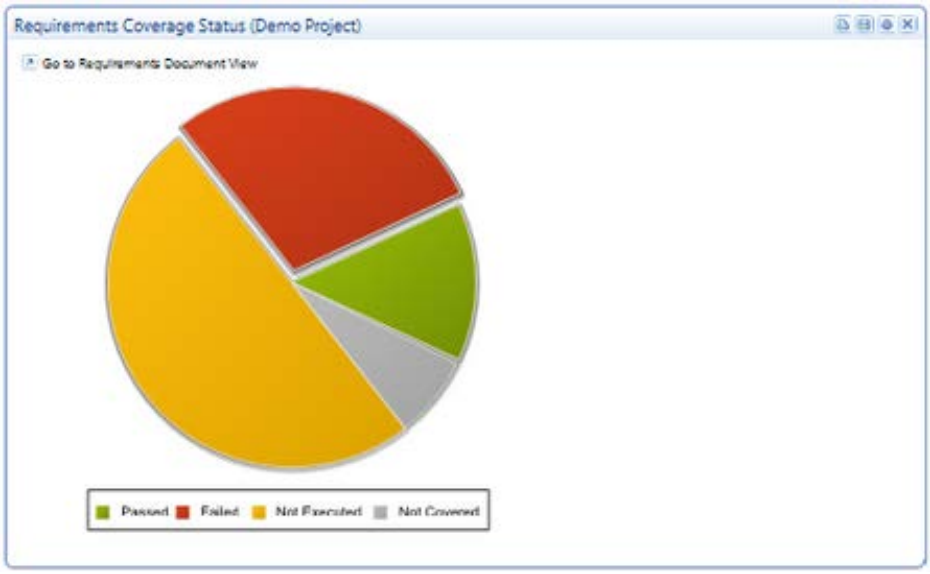


Figura 30. Estado cobertura de requisitos

Se trata de un tipo de gráfico circular donde las áreas diferenciadas son las siguientes, en color gris se muestran los requisitos que no son cubiertos por ninguna prueba, el resto sí que están cubiertos y se muestran por el resultado de las pruebas, donde el área verde son las pruebas que han pasado correctamente, el área roja son las pruebas que han fallado y en amarillo las que no se han ejecutado.

Con esta gráfica podemos conocer la siguiente información:

- Cantidad de requisitos cubiertos y no cubiertos
- Resultado de las pruebas que están enlazadas con los requisitos

Progreso de ciclo de pruebas

Esta grafica muestra la evolución que se sigue en un ciclo de pruebas definido; en este ciclo de pruebas hay asignados unos testers con unas pruebas que deberán pasar. Los Testers responsables son los que cambiarán el estado a las Pruebas, como ya se vio en la gráfica *Pruebas manuales asignadas a mí*, y son esos cambios los que se representarán aquí.

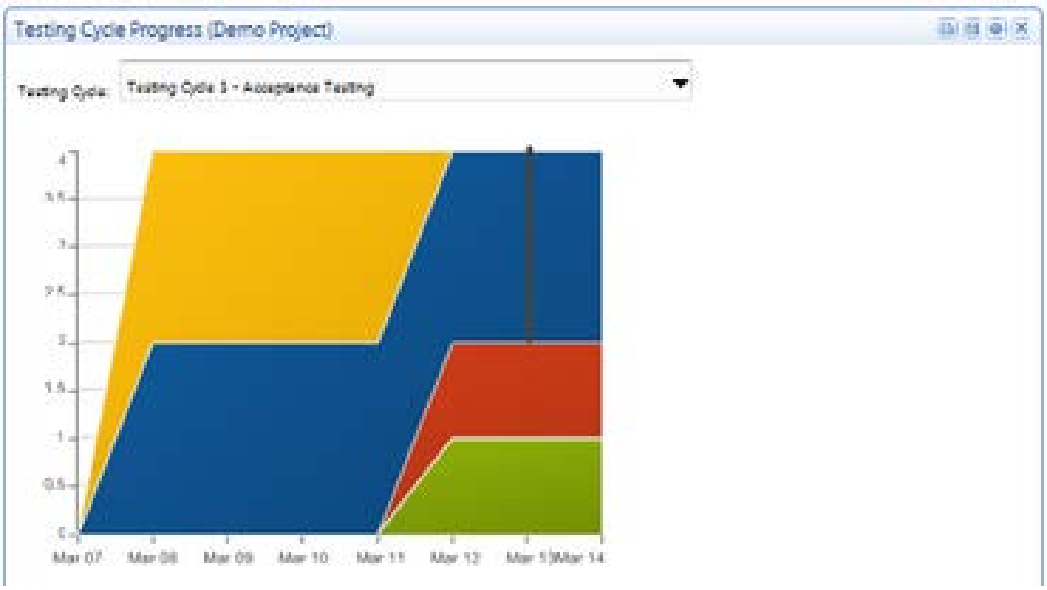


Figura 31. Progreso de ciclo de pruebas

Se trata de un tipo de gráfico de áreas apiladas donde el área amarilla son las pruebas que aun no se han ejecutado, el área azul son las pruebas que están en progreso de ejecución, el área rojo las que ya se han ejecutado y han fallado y el área verde aquellas pruebas que han pasado correctamente.

Con esta gráfica se puede conocer la siguiente información:

- En qué estado se encuentra el ciclo de pruebas
- La rapidez con la que se pasan las pruebas
- Cantidad de pruebas pendientes a pasar y en progreso
- Cantidad de pruebas que han fallado y han pasado

Resultados de ciclo de pruebas en resumen

Este panel muestra información más detallada sobre el ciclo de testeo seleccionado. En la parte superior hay una visión global donde se muestra el estado de las pruebas y en la parte inferior se divide entre los agentes que participan en ese ciclo de testeo.

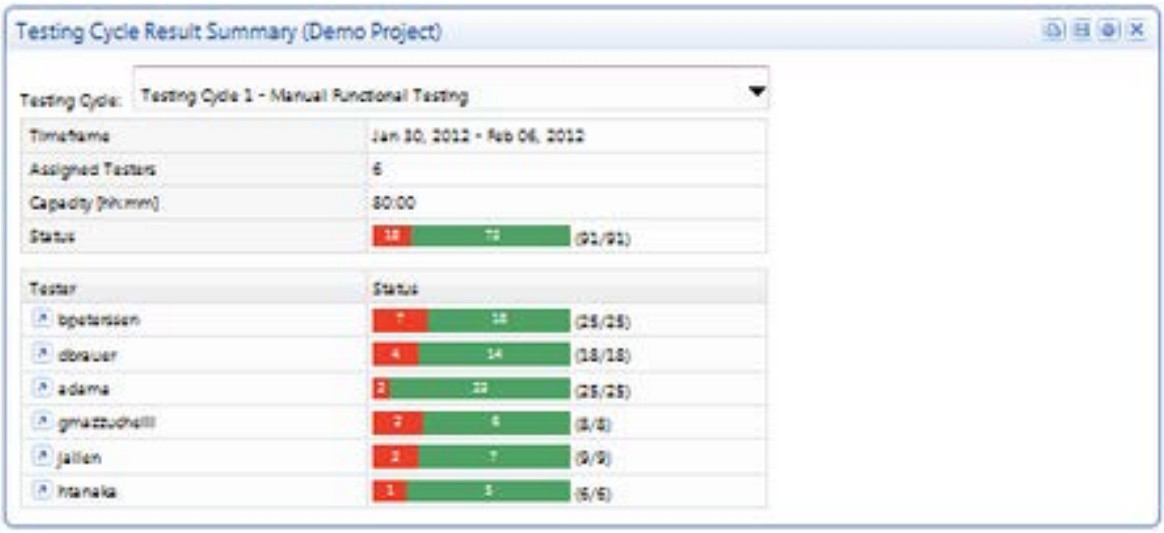


Figura 32. Resultados de ciclo de pruebas en resumen

Con este panel podemos conocer la siguiente información:

- Estado, en general y distribuido en testers, del resultado de las pruebas
- Cuántos testers están participando
- El periodo de días asignado y la cantidad de tiempo estimada

Progreso de testeo respecto de los ciclos de pruebas

La grafica siguiente muestra la planificación en horas necesarias para finalizar el proceso de testeo. Esta grafica ayuda a predecir si el equipo de testeo podrá finalizar la aplicación de pruebas a tiempo.

Figura 33. Progreso de testeo respecto de los ciclos de pruebas

Los ciclos de vida se definen dentro de un intervalo de días y con una cantidad de horas que corresponde con el sumatorio del tiempo que se le asignó a cada prueba. Así cada escalón que se diferencia en la línea se corresponde con un ciclo de prueba y, por otra parte, las áreas en azul apiladas se corresponden con las pruebas que están en progreso, y en gris las que ya se han completado. En el ejemplo se puede observar como en el penúltimo ciclo de pruebas no se ha conseguido pasar todas las pruebas en el intervalo de días planificado.

Con esta gráfica podemos conocer la siguiente información:

- Si se va a completar el proceso de testeo en la fecha indicada
- Si los ciclos de prueba se completan correctamente, con todas las pruebas pasadas.

5.2. IBM Rational Quality Manager

IBM Rational Quality Manager es un sistema de gestión de calidad colaborativo y basado en la Web, que ofrece la planificación completa de testeo y para el ciclo de vida de desarrollo del software. *Rational Quality Manager* trabaja principalmente con **Planes de Pruebas**, en estos planes de pruebas se definen los objetivos de negocio y testeo que debe cumplir, para ello se asignan al *Plan de Pruebas* los **Requisitos**, acordados con el cliente; a estos requisitos se les podrá relacionar con unos **Casos de Prueba** nuevos o ya existentes. En la ejecución de los casos de prueba se pueden detectar **Defectos** que se relacionarán con el caso de prueba.

Esta solución permite elaborar un *Dashboard personalizable*, pudiendo añadir los diferentes elementos disponibles que se van a ver a continuación.

Estado de ejecución por Tester

En esta gráfica se muestra el estado de los casos de prueba agrupados por los agentes responsables. Se pueden seleccionar más de un plan de pruebas para ver el estado de ejecución de los casos de pruebas.

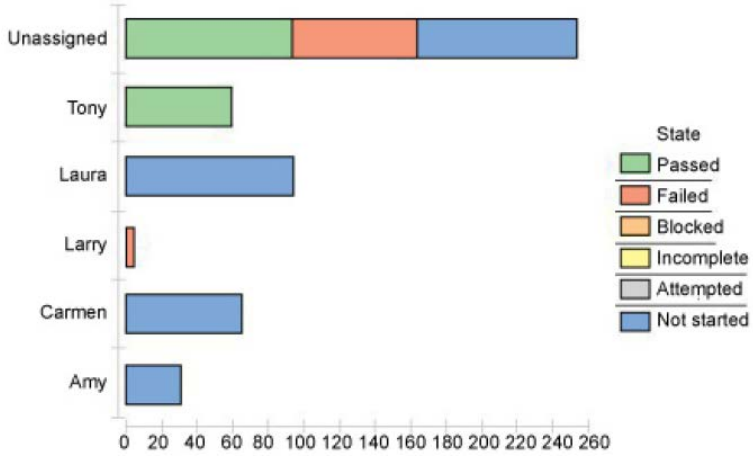


Figura 34. Estado de ejecución por Tester

Con esta gráfica se puede conocer la información siguiente:

- Testers que están participando en los planes de pruebas seleccionados
- El estado de la ejecución de los casos de pruebas
- La cantidad de casos de pruebas asignados a los testers

Tendencia de ejecución

Esta gráfica puede ser usada para comparar el progreso real de ejecución de pruebas para el progreso previsto. En ella se compara lo que se ha hecho con lo que pensaba hacer. También muestra la cantidad de trabajo que queda y lo que necesita para cambiar la velocidad si se desea permanecer dentro del objetivo marcado.

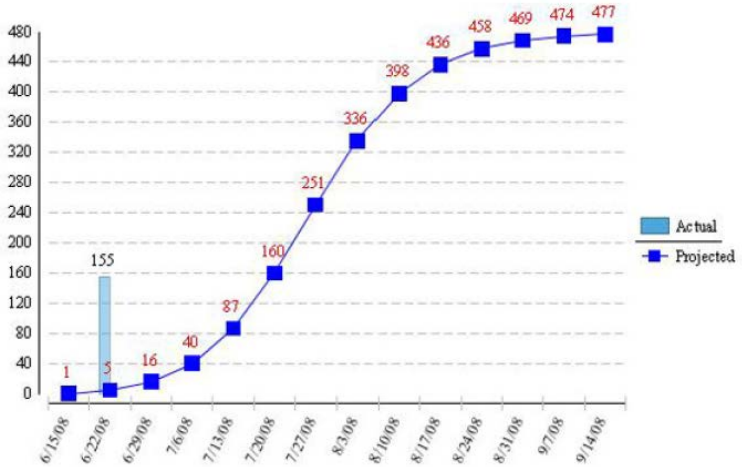


Figura 35. Tendencia de ejecución

La línea azul corresponde a la evolución que se ha planificado para poder lograr el objetivo final, por otra parte están las barras que muestra lo que se ha hecho diariamente.

Con esta gráfica podemos conocer la siguiente información:

- El progreso actual de testeo respecto del planificado
- Si se necesita acelerar la actividad de testeo para llegar a lo planificado o, por el contrario, se tiene un buen ritmo

Estado actual de ejecución

Esta gráfica es configurable, pero por defecto se muestra el estado de ejecución de la prueba por el plan de prueba dentro de su proyecto. Es posible mostrar varios planes de pruebas. De cada Plan de pruebas se contabilizan los TERs (Registros de ejecución de pruebas) que se corresponden con el estado de los casos de prueba.

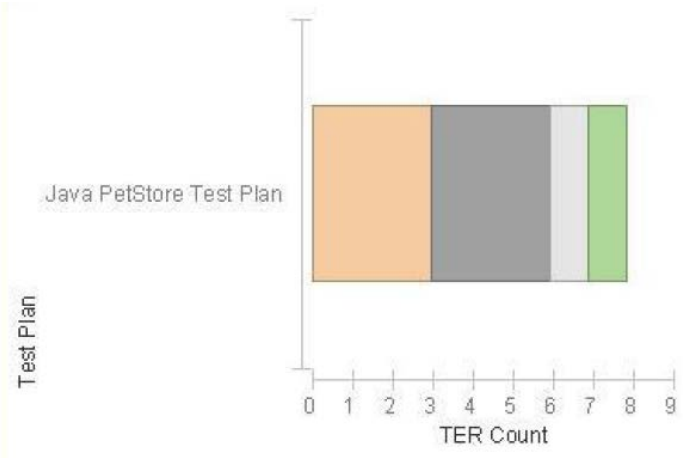


Figura 36. Estado actual de ejecución

Hay varios estados disponibles por defecto; en este caso, los estados que se muestran corresponden a los casos de prueba *Bloqueados*, en naranja; los casos de prueba *Incompletos*, en color gris oscuro; casos de prueba *No ejecutados*, en color gris claro y, en color verde, los casos de pruebas *Pasados*.

Con esta gráfica podemos conocer la siguiente información:

- Numero de pruebas que están listas para ejecutar
- Numero de pruebas que faltan completar
- Numero de pruebas que contiene el plan de pruebas

Cobertura de requisitos

La presente gráfica muestra la cobertura global de los requisitos. El gráfico se divide en dos secciones: cubiertos y no cubiertos. Al pinchar sobre cualquiera de las secciones se puede obtener los detalles de los requisitos y casos de prueba correspondientes. La zona verde se corresponde con los requisitos cubiertos y la zona roja con los requisitos no cubiertos.

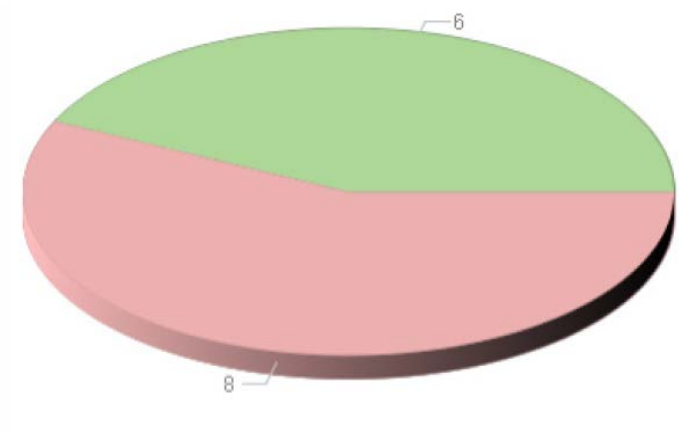


Figura 37. Cobertura de requisitos

Con esta grafica se puede conocer la siguiente información:

- Requisitos que están cubiertos por casos de pruebas
- Requisitos que no están cubiertos por ningún caso de prueba

5.3. Visual Studio Team Foundation Server

Microsoft proporciona una serie de *Dashboards* diferentes en los que se agrupan distintos gráficos para tener una visión conjunta del *Dashboard* escogido. El proyecto debe tener un portal habilitado y asociado a un sitio SharePoint. El elemento clave en el proceso de desarrollo que se sigue son las **Historias de Usuario**, en ellas se plasma la funcionalidad que se ha acordado con el cliente; se debe evitar detallar cómo se va a implementar esta funcionalidad, se debe mostrar la descripción de la funcionalidad desde la perspectiva del usuario final. Estas Historias de Usuario se implementan y se les asignan **Casos de Prueba** que deberán validar el comportamiento de las mismas, que pueden ser pruebas manuales o automatizadas. Existen los **Planes de Pruebas** en los que se planificará la actividad de testeo y contendrán casos de pruebas específicos.

A medida que se vayan ejecutando los casos de prueba se pueden encontrar **Defectos**, en lo que se detalla la información referente al defecto encontrado para arreglarlo.

Existen varios Dashboards creados por defecto, *Project Dashboard*, *Progress Dashboard*, *Quality Dashboard*, *Test Dashboard*, *Bugs Dashboard* y *Build Dashboard*, entre otros. A continuación se va a proceder a analizar aquellos gráficos que están relacionados con el objetivo de esta tesina.

Progreso del Plan de Pruebas

Esta gráfica está representada a través de un gráfico de áreas. Muestra el progreso del *Plan de Pruebas* a lo largo del tiempo. En el eje de las abscisas se muestra el rango de días seleccionado, mientras que en el eje de las ordenadas se muestra en número de casos de prueba que componen el *Plan de Pruebas*.

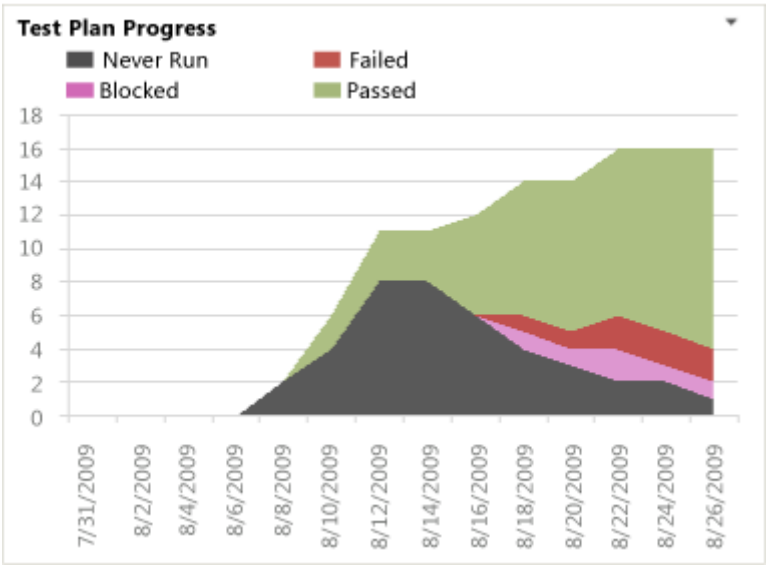


Figura 38. Progreso del Plan de Pruebas

A partir del gráfico se pueden realizar, entre otras, las diferentes lecturas:

- Número de casos de pruebas que pasan correctamente.
- Número de casos de pruebas que no pasan correctamente.
- Número de casos de pruebas restantes para finalizar.
- Número de casos de pruebas que no se han ejecutado.
- Si se va a cumplir los tiempos estimados.

Además, tenemos opciones de filtrado sobre el gráfico. Es posible cambiar el rango de días para mostrar el progreso entre ellas, seleccionar de entre todas las áreas del

producto disponible, filtrar por cualquier iteración deseada y, por último, filtrar por un plan de pruebas creado.

Disponibilidad de los casos de pruebas

Esta gráfica está representada a través de un gráfico de áreas. Muestra el estado de los casos de prueba a lo largo del tiempo. Se distinguen dos estados, **Design** y **Ready**, un caso de prueba está en estado *Design* desde que se crea hasta que el equipo, después de haber realizado y revisado el diseño, decide que ya está preparado para ejecutarse. En el eje de las abscisas se muestra el rango de días seleccionado, mientras que en el eje de las ordenadas se muestra en número de casos de prueba.

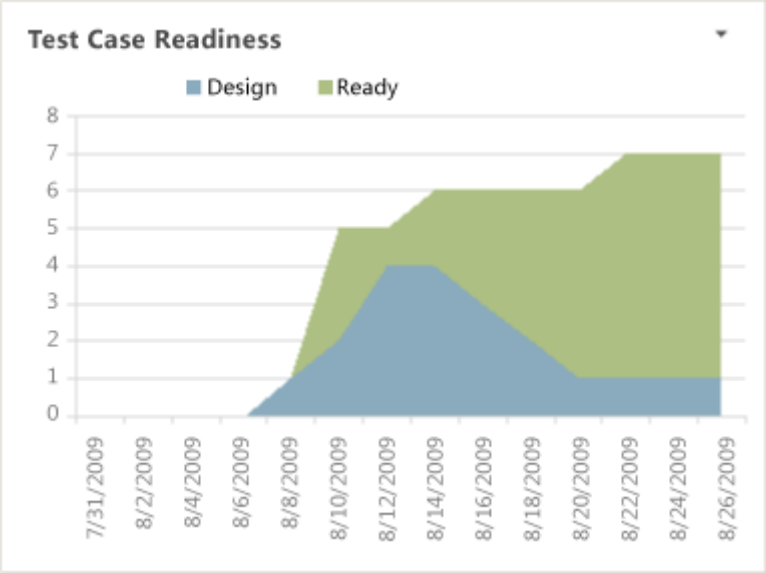


Figura 39. Disponibilidad de los casos de pruebas

A partir del gráfico se pueden realizar, entre otras, las diferentes lecturas:

- Número de casos de pruebas disponibles para ejecutar.
- Número de casos de pruebas pendientes de diseñar o revisar el diseño.
- Si se va a cumplir los tiempos estimados.

Además, tenemos opciones de filtrado sobre el gráfico. Es posible cambiar el rango de días para mostrar el progreso entre ellas, seleccionar de entre todas las áreas del producto disponible, la prioridad y estado de los casos de prueba.

Estado de testeo de las historias de usuario

Esta gráfica está representada a través de un gráfico de de barras horizontales. Para cada *historia de usuario* muestra el número de casos de prueba y los últimos resultados obtenidos para cada caso de prueba. En el eje de las abscisas se muestra el número de casos de prueba que contiene la historia de usuario, mientras que en el eje de las ordenadas se muestra las historias de usuario.

Figura 40. Estado de testeo de las historias de usuario

A partir del gráfico se pueden realizar, entre otras, las diferentes lecturas:

- Número de casos de pruebas por cada historia de usuario.
- Qué historias de usuario tienen un mayor número de fallos.
- Si el equipo está ejecutando todos los casos de prueba para cada historia de usuario.
- Qué historias de usuario tienen un mayor número de casos de pruebas bloqueadas o sin ejecutar.

Además, tenemos opciones de filtrado sobre el gráfico. Es posible cambiar la vista del gráfico, pudiendo filtrar por iteraciones y por las diferentes áreas del producto disponibles.

Actividad de testeo

Esta gráfica está representada a través de un gráfico de líneas. Muestra la cantidad de casos pruebas que se han ido ejecutando en el tiempo. En el eje de las abscisas se muestra el rango de semanas seleccionado, mientras que en el eje de las ordenadas se muestra el número de casos de prueba.

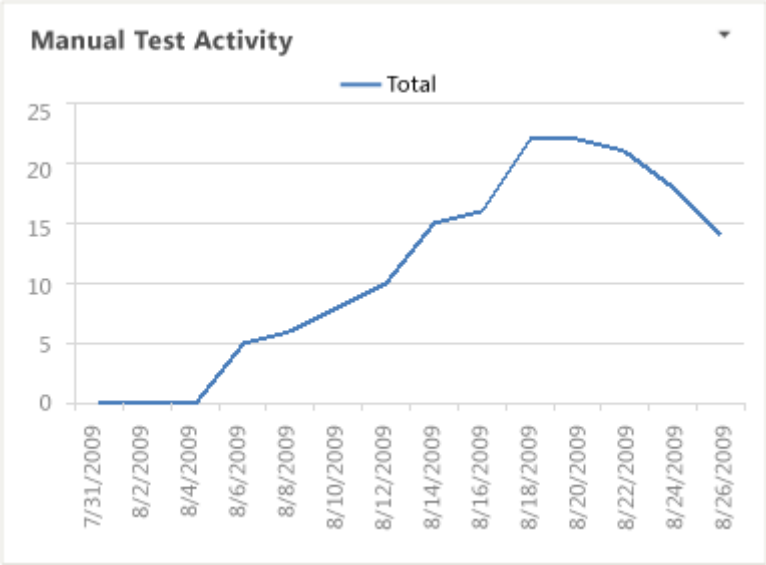


Figura 41. Actividad de testeo

A partir del gráfico se pueden realizar, entre otras, las diferentes lecturas:

- Frecuencia a la que se pasan los casos de prueba
- Cantidad de casos de prueba que se pasan diariamente

Además, tenemos opciones de filtrado sobre el gráfico. Es posible cambiar el rango de semanas, visualizar la actividad para los casos de prueba automatizados y seleccionar entre las diferentes iteraciones, áreas del producto y planes de testeo disponibles.

Análisis de pruebas fallidas

Esta gráfica está representada a través de un gráfico de áreas apiladas. Muestra los tipos de fallos que se han ocasionado en el último período de tiempo seleccionado. Los distintos elementos del gráfico son **New Issue**, **Known Issue**, **Regression** o **None**. En el eje de las abscisas se muestra el rango de semanas seleccionado, mientras que en el eje de las ordenadas se muestra el número de casos de prueba que se han ejecutado.

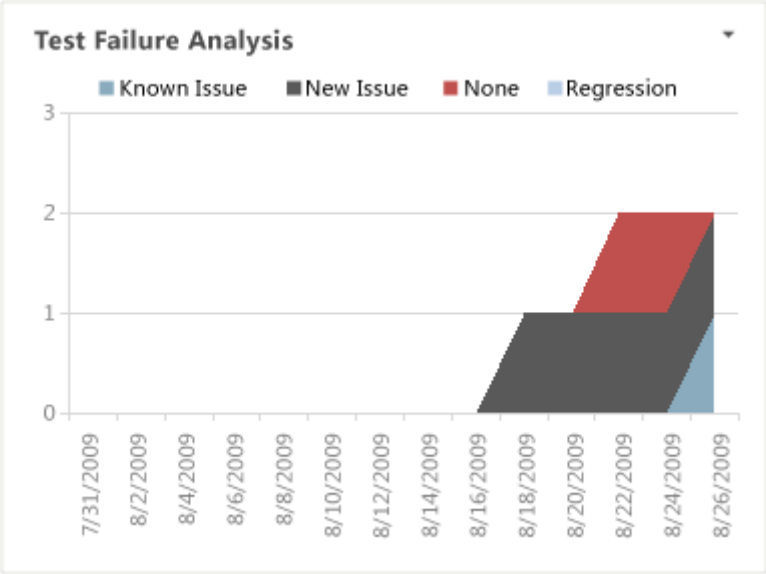


Figura 42. Análisis de pruebas fallidas

A partir del gráfico se pueden realizar, entre otras, las diferentes lecturas:

- Número de fallos que se detectan.
- Número de fallos que se solucionan.
- Tiempo que se tarda en solucionar fallos.
- Número de fallos de regresión que se detectan.

Además, tenemos opciones de filtrado sobre el gráfico. Es posible cambiar el rango de semanas, seleccionar entre las diferentes iteraciones, áreas del producto y planes de testeo disponibles.

5.4. Conclusiones

Tras realizar el estudio del estado del arte se decidió descartar la implantación de alguna de las herramientas en el proceso de testeo, ya que el entorno de testeo manual y automatizado se encontraba en un punto muy avanzado, llegando a tener aproximadamente 10.000 pruebas de aceptación, 3.000 pruebas de sistema y más del 50% de interfaces del productos grabadas.

Por otra parte, el estudio realizado sobre las herramientas nos ha ayudado a conocer qué información muestran los Dashboards, recoger ideas sobre métricas que nos pueden resultar útiles y plantearnos qué infraestructura necesitábamos desarrollar para dar soporte a las mismas.

Respecto a las características que ofrecen los Dashboards, en las tres herramientas ofrecen la posibilidad de elaborar Dashboards personalizados, pudiendo escoger las métricas que conformarán el Dashboard de un conjunto de métricas disponibles. Además, mencionar que en la solución de Microsoft, como ya se ha comentado, hay disponibles diferentes Dashboards con un fin diferente cada uno, por ejemplo *Test Dashboard*, *Defect Dashboard*, entre otros, con los que podemos tener un enfoque dirigido hacia los conceptos que se tratan.

Capítulo 6. Infraestructura desarrollada

A continuación se va a mostrar la infraestructura actual que se ha desarrollado para poder dar soporte a las métricas que se utilicen para supervisar el proceso de testeo descrito en el capitulo anterior. El mapa que presenta el conjunto de herramientas que forman la infraestructura corresponde con la imagen que se muestra a continuación.

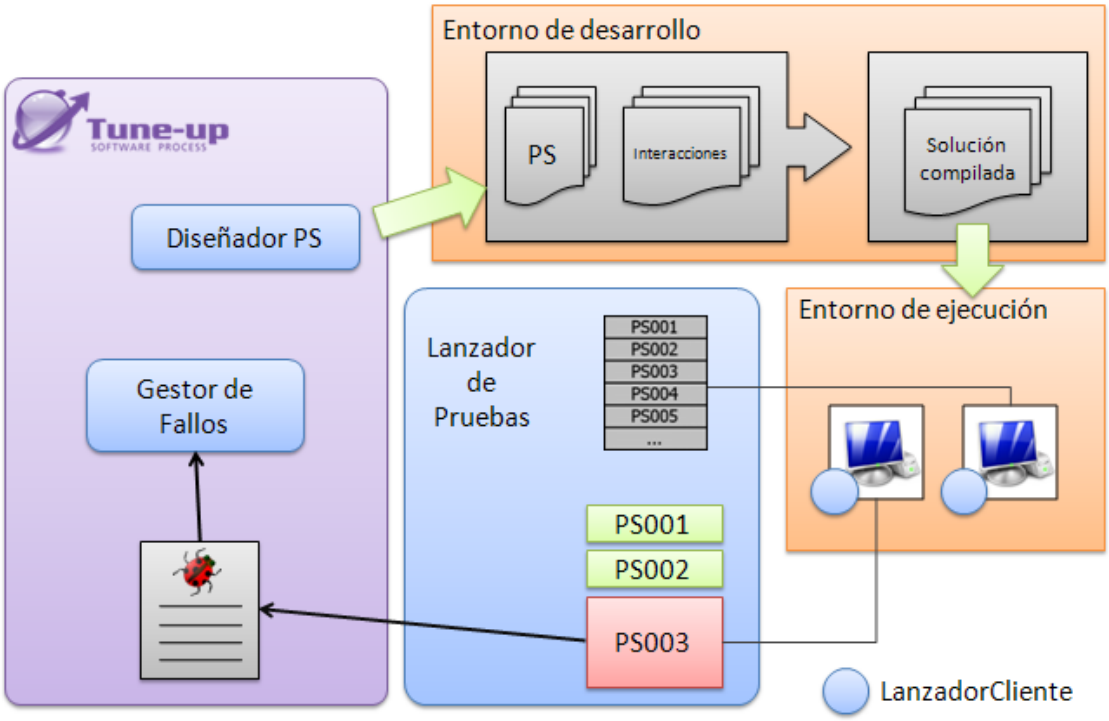


Figura 43. Esquema de la infraestructura desarrollada

Siguiendo el proceso automatización que engloba todo el conjunto de herramientas desarrolladas, tenemos como base la herramienta *TUNE-UP*. En el diseñador de PS se crean las instancias de las PAs.

Existe un entorno de desarrollo con la solución de testeo automatizado y un entorno de ejecución donde en cada máquina está instalado el *LanzadorCliente*, encargado de realizar las tareas que se le asignen. La solución de testeo automatizado se compila y el resultado se deposita en una carpeta compartida accesible desde los entornos de ejecución; el *LanzadorCliente* tendrá la función de consultar si existe una nueva compilación, y en el caso de que sí exista, actualizar los ficheros de la máquina.

La siguiente herramienta es el *Lanzador de Pruebas*, cuya función principal es iniciar el proceso de ejecución Al crear una cola de pruebas el *LanzadorCliente* de cada máquina

se encargará de coger la primera prueba de sistema sin ejecutar, ejecutarla y registrar el resultado obtenido; en caso de ser un FAIL tendremos un log con las anomalías encontradas. Dependiendo de FAIL se creará un fallo relacionado con la PS y quedará accesible con el *Gestor de Fallos*. Surgió como un sistema de apoyo en el equipo de testeo para tener constancia de los fallos que se encontraban y poder seguir con el proceso necesario para solucionarlos.

En los siguientes puntos se va a explicar en detalle cada herramienta desarrollada.

6.1. Diseñador de Pruebas de Sistema

El **Diseñador de Pruebas de Sistema** se empezó a desarrollar como un proyecto de mejora interna en la empresa. Surgió la necesidad de montar un sistema con el que poder crear diseños de una manera fácil, rápida y más intuitiva que las prácticas que se tenían hasta el momento, tales como hojas de Excel, documentos de texto, entre otros.

La interfaz que presenta el *Diseñador de PS* es el que se muestra en la figura 44; el *Diseñador de PS* se desarrolló utilizando como base de datos la de TUNE-UP. A la parte de la izquierda hay una serie de pestañas que corresponden a las PAs que se van a automatizar de una WU seleccionada, hay una opción para visualizar todo el conjunto de PAs, pero el objetivo del sistema es crear los diseños de las pruebas del sistema, por ello, por defecto, se discriminan las PAs que no se van a automatizar. En la parte superior tenemos la descripción de la PA, en la parte central se muestra el grid de las pruebas de sistemas creadas, en la parte inferior tenemos toda la información referente a las PS seleccionada, el correspondiente diseño de la PS, los datos que se van a utilizar en la PS, los resultados de ejecución que se han registrado y los fallos asociados a la PS y a la parte derecha del grid las opciones posibles para crear PS o copiar.

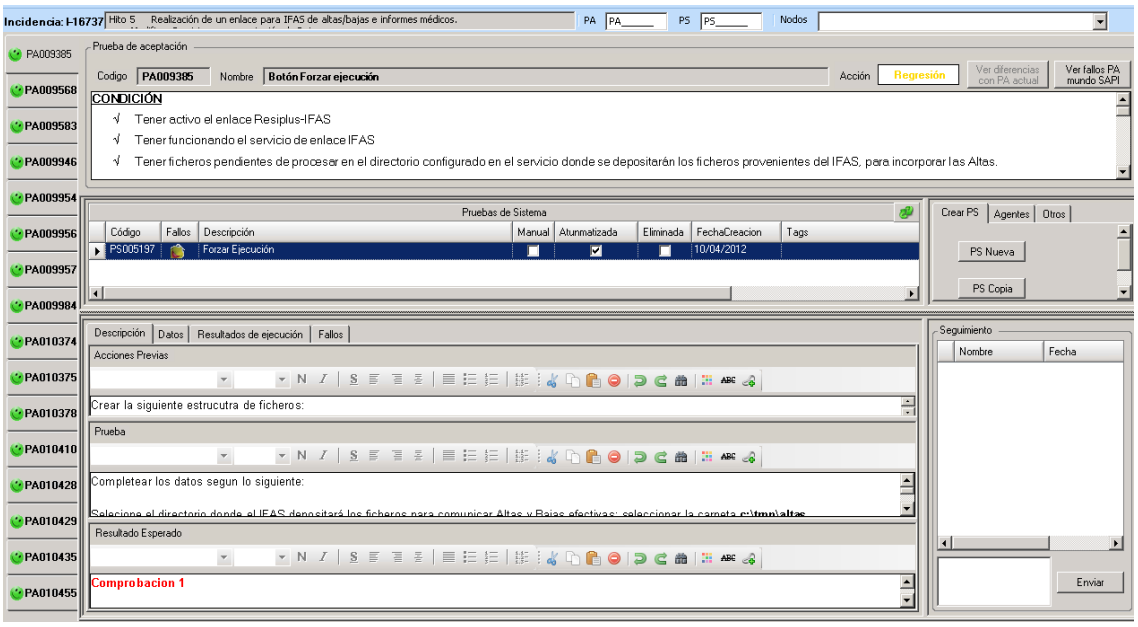


Figura 44. Formulario principal del Diseñador de PS

Una vez el *Diseñador de PS* llegó a una versión estable se decidió integrar en TUNE-UP y optimizar el proceso de diseñado desde TUNE-UP. La integración consistió en llevar a TUNE-UP el grid del pruebas de sistema, la información relacionada con la misma y las opciones disponibles; al apretar en el botón "**Vista diseñador**" que podemos ver en la figura 45 el contenido de la pestaña cambia al que se muestra en la figura siguiente.

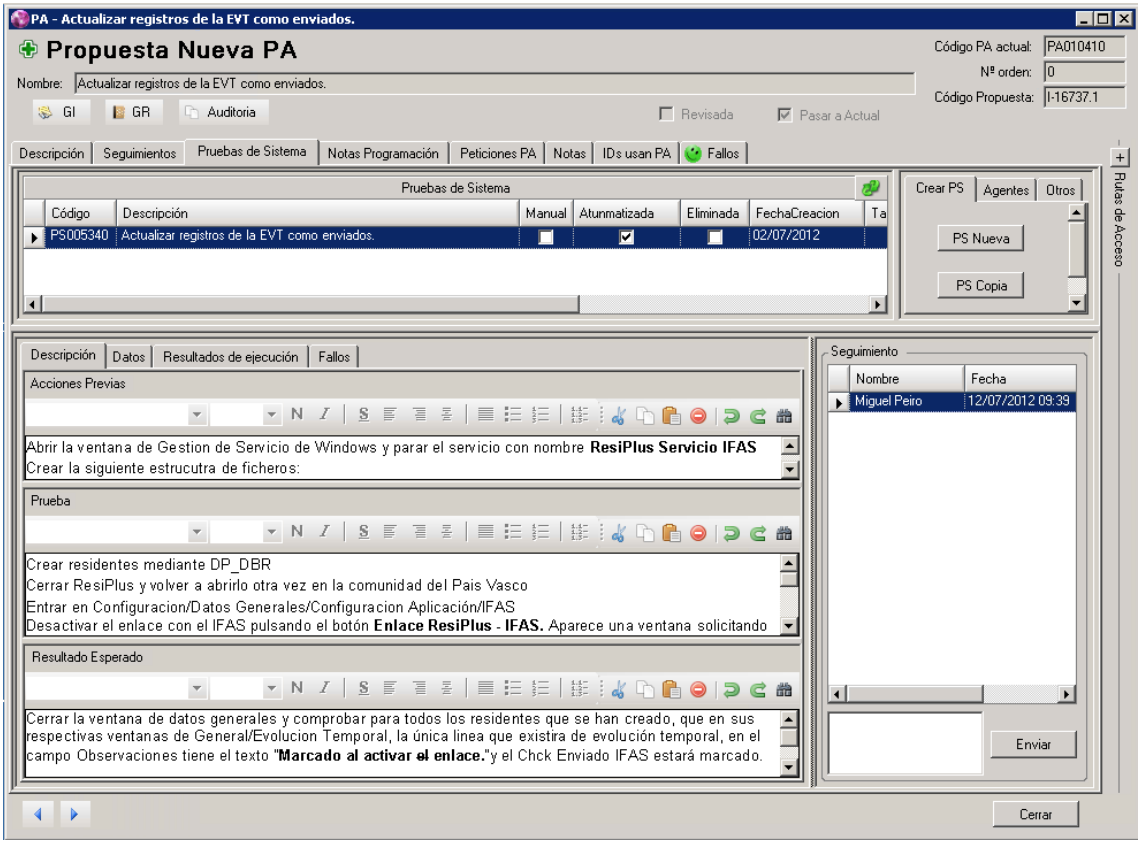


Figura 45. Formulario de PA con el Diseñador de PS integrado

A continuación se va a describir un caso práctico del proceso que se sigue para diseñar pruebas automatizadas cuando a un Tester le llega una WU con sus PAs correspondientes. El caso práctico se va a realizar sobre la siguiente PA "*Asteriscos no permitidos, DC no numérico*"

CONDICIÓN
Tener marcado el check "No permitir la introducción de ** como dígitos de control."
Cualquier configuración excepto Portugal.
PASOS
Introducir letras o caracteres, asteriscos incluidos, en el campo DC.
Intentar salir.
RESULTADO
No se escriben en el campo

El comportamiento que define esta PA es que teniendo una opción de configuración marcada, el check "*No permitir la introducción de ** como dígitos de control*", y para cualquier configuración excepto Portugal, que hace referencia a la localización donde se está utilizando el producto. Con estas condiciones se debe seguir una serie de pasos en todos los sitios posibles del producto donde se pueda insertar una cuenta bancaria, el

resultado de los pasos será que no se ha insertado nada en el campo DC de la cuenta bancaria.

Así tendremos tres casos base; uno introduciendo letras en el campo DC, otro introduciendo carácteres especiales y por último introduciendo **. Estos tres casos base se deberán repetir en tantas PSs como zonas posibles en el producto exista la opción de introducir una cuenta bancaria; en concreto, existen seis zonas diferentes en las cuales poder introducir una cuenta bancaria, por lo tanto, deberemos crear 18 PSs con las que comprobar los tres casos base para cada zona. Para ello, pincharemos sobre el botón “PS Nueva” y se nos creará una PS vacía. En este punto es donde se introduce la descripción de la PS que corresponderá a una instancia de la PA.

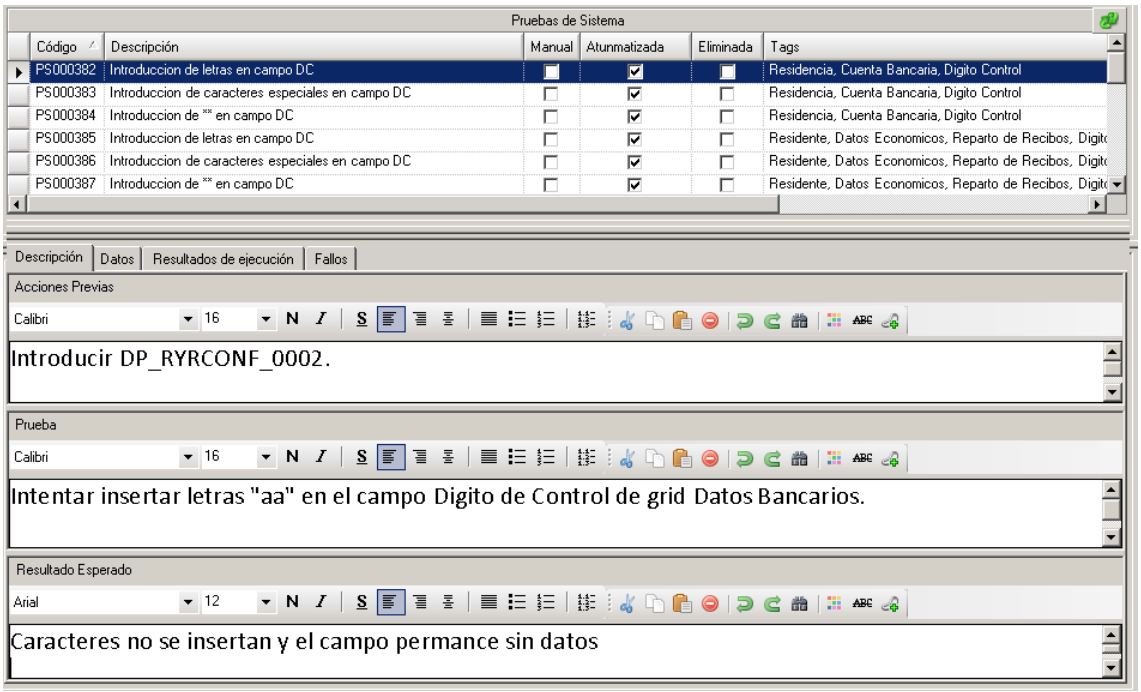
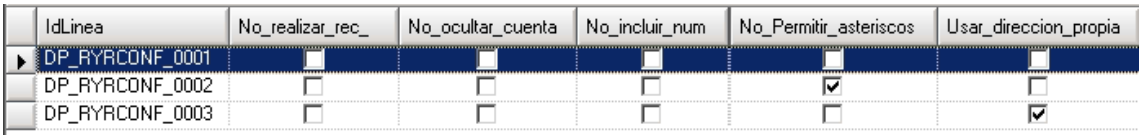


Figura 46. Ejemplo Prueba de Aceptación con Pruebas de Sistema

Como se muestra en la figura 46, en **Acciones Previas** describimos las condiciones que se pedían en la PA, en **Prueba** tendremos los pasos que hay que seguir, descritos en la PA, y en el **Resultado Esperado** lo que corresponde; para los diseños de PS se utiliza la idea de **Datapool** (DP), un datapool es un conjunto de datos con valores concretos que se utilizarán durante la ejecución. El DP_RYRCONF (Figura 47) da valor a un conjunto de checks que hay en una zona del producto, entre los cuales podemos observar el campo **No_Permitir_asteriscos**; en todas las PS utilizaremos la línea **DP_RYRCONF_0002** ya que es la única que lo tiene marcado. Las diferentes líneas

que pueden haber en un datapool corresponden a las posibles combinaciones de datos que necesitemos.



IdLinea	No_realizar_rec_	No_ocultar_cuenta	No_incluir_num	No_Permitir_asteriscos	Usar_direccion_propia
DP_RYRCONF_0001	☐	☐	☐	☐	☐
DP_RYRCONF_0002	☐	☐	☐	☑	☐
DP_RYRCONF_0003	☐	☐	☐	☐	☑

Figura 47. Ejemplo Datapool

Cuando el *Tester* crea los diseños necesarios activa un aviso para el *Automation Tester* correspondiente, este sistema de avisos está disponible en la pestaña *Agentes* de la parte derecha del grid de pruebas de sistema.

6.2. IBM Rational Functional Tester

El *Automation Tester* es el responsable del proyecto de automatización, en este proyecto se crean todas las PS que se diseñan en la iteración. Por otra parte, existe un proyecto por cada producto que se va a testear; por cada producto, según vaya siendo necesario, se va grabando el mapa de objetos que presenta. El proyecto de pruebas de sistema esta enlazado al proyecto donde se graban los mapas de objetos de las aplicaciones.

La empresa posee una licencia del sistema **IBM Rational Functional Tester**; con *IBM Rational Functional Tester* se puede grabar las interacciones con una interfaz y reproducirlas posteriormente. En la empresa se adaptó la funcionalidad del *Rational Functional Tester* para compatibilizarlo con los proyectos de testeo, de manera que desde el **Visual Studio** se puede invocar el grabador del *Rational Functional Tester* y generar en el proyecto las interacciones que se han llevado a cabo con la aplicación junto con el mapa de objetos que han intervenido en las interacciones.

Existen varios proyectos satélites que ayudan a completar y optimizar la ejecución de las pruebas de sistemas, tales como comparadores de textos, creación de logs, etc. En las pruebas de sistema, el *Automation Tester* debe insertar puntos de verificación que insertarán un registro en el log del resultado marcando si la comprobación ha sido correcta o no.

CONDICIÓN
Tener marcado el check “No permitir la introducción de ** como dígitos de control.”
Cualquier configuración excepto Portugal.
PASOS
Introducir letras o caracteres, asteriscos incluidos, en el campo DC.
Intentar salir.
RESULTADO
No se escriben en el campo

Siguiendo con el caso práctico anterior, el Automation Tester deberá grabar, si aún no lo estuvieran, los controles que van a ser utilizados en la ejecución; en este caso se grabará el check que tendremos en las opciones de configuración “*No permitir la introducción de ** como dígitos de control.*”, además de los grids o campos de texto donde tengamos que introducir el valor del DC. Una vez tenemos todos los controles grabados en el mapa de objetos correspondiente, podemos iniciar la automatización de la PS; en primer lugar, la aplicación se iniciará en la comunidad por defecto, que es Comunidad Valenciana, se abrirá el formulario de configuración y se marcará el check en cuestión, con esto ya hemos completado las condiciones de la PS; a continuación, iremos a la zona indicada en la PS e intentaremos introducir el valor del DC correspondiente. Por último, deberemos comprobar con un punto de verificación que en el campo DC no se ha insertado nada. Cuando la PS está automatizada únicamente queda ejecutarla para validar el comportamiento del producto definido en la PA.

6.3. Lanzador de Pruebas

El **Lanzador de Pruebas** es otro proyecto que se inició como un proyecto de mejora interna en la empresa. En la primera versión se podían ejecutar pruebas de sistema en local, es decir, el Lanzador permitía ejecutar pruebas solo en la misma máquina donde se iniciaba, visualizar el resultado de la ejecución de las pruebas gracias al **Histórico de Logs** y todas las opciones de configuración básicas. El *Historico de Logs* está formado por un grid maestro-detalle donde por cada batería de pruebas ejecutadas, podemos expandir y ver el resultado individual de las pruebas del sistema como se muestra en la figura 48.

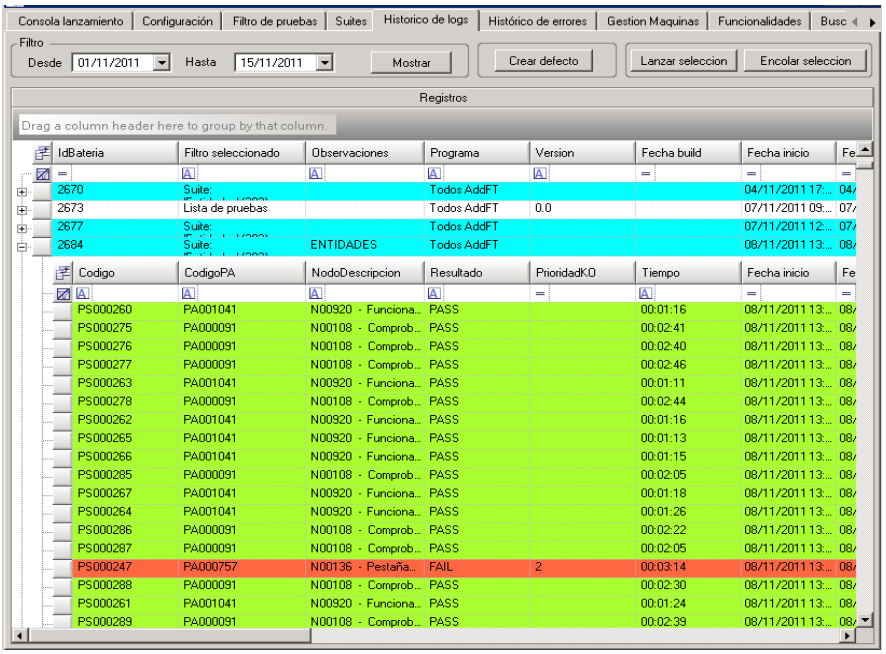


Figura 48. Pestaña Histórico de Logs del Lanzador de Pruebas

Desde entonces, el *Lanzador de Pruebas* se ha ido desarrollando incluyendo nuevas funcionalidades. Las principales mejoras han sido la posibilidad de ejecutar pruebas en remoto sin necesidad de abrir el Lanzador en la máquina. Se optó por crear un proceso cliente, el **LanzadorCliente**, que ubicaríamos en cada máquina disponible y habilitada para ejecutar pruebas, este cliente se encargaría de consultar si tiene tareas pendientes a realizar, como la actualización del proyecto de pruebas, ejecución de alguna prueba de sistema, entre otras. Así, en la actualidad, cuando se ejecuta un conjunto de pruebas en varias maquinas, se crea una cola con las pruebas seleccionadas y el cliente de las maquinas que han sido seleccionadas consultaran si quedan pruebas por ejecutar e irán ejecutando hasta que no queden más. El proceso que se sigue es sencillo y los resultados son mas óptimos, ya que con esta nueva funcionalidad el tiempo que se tarda en ejecutar todo el conjunto de pruebas es el mínimo, en concreto el tiempo total que suman todas las PS y al mismo tiempo no hay problemas cuando una máquina deja de funcionar. En una batería determinada de 150 PS, ejecutadas en 5 máquinas, siguiendo el proceso antiguo llegaban a tardar alrededor de 5 horas, mientras que ahora tarda menos de la mitad; los motivos se basan en que las PSs se asignaban a las máquinas, obligando la total ejecución de ese conjunto de PSs, y pudiendo darse el caso que las otras máquinas hayan acabado y no se aprovechen esos recursos.

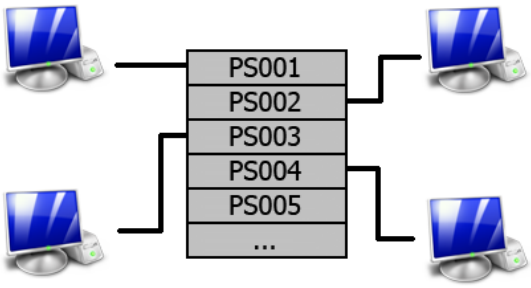


Figura 49. Ejecución de pruebas de sistema con el LanzadorCliente

Además, una vez contamos con la existencia del *LanzadorCliente* teníamos la posibilidad de controlar las máquinas de forma remota, como se ha comentado antes, pudiendo actualizar el proyecto de pruebas, actualizar el producto a testear, evitándonos la conexión a las maquinas para realizarlo de forma manual; toda esta funcionalidad se encuentra en el *Lanzador de Pruebas*, en la pestaña **Gestión Máquinas** (figura 50).

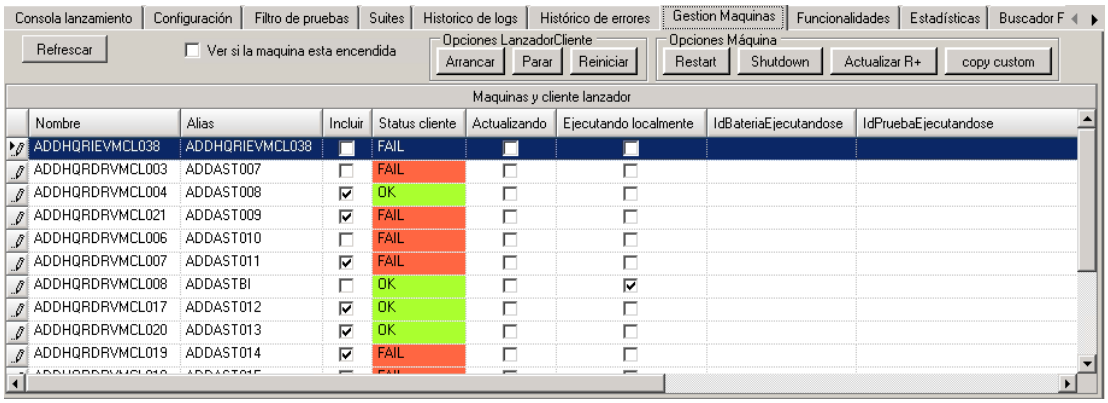


Figura 50. Pestaña Gestor Máquinas del Lanzador de Pruebas

Por otra parte, de la literatura que disponíamos [15,16,17,18] decidimos incorporar la idea de **suite de pruebas** e incorporarlo en el proceso. Una *suite* consiste en un conjunto de pruebas que cumplen con una serie de condiciones; estas suites se guardan en base de datos y quedan disponibles hasta su borrado.

Volviendo a la PA automatizada, el siguiente paso es ejecutar las PSs y revisar el resultado obtenido. Para ello, el Lanzador dispone de la pestaña Filtro de Pruebas, donde podremos filtrar por la PA000086 que se ha automatizado y en el panel inferior se mostrarán las PSs automatizadas. En el "**Modo de selección de pruebas**" deberemos indicar el conjunto de pruebas que queremos lanzar, podemos ejecutar el resultado del filtro por completo, seleccionar un subconjunto de pruebas o abrir una suite de pruebas creada anteriormente.

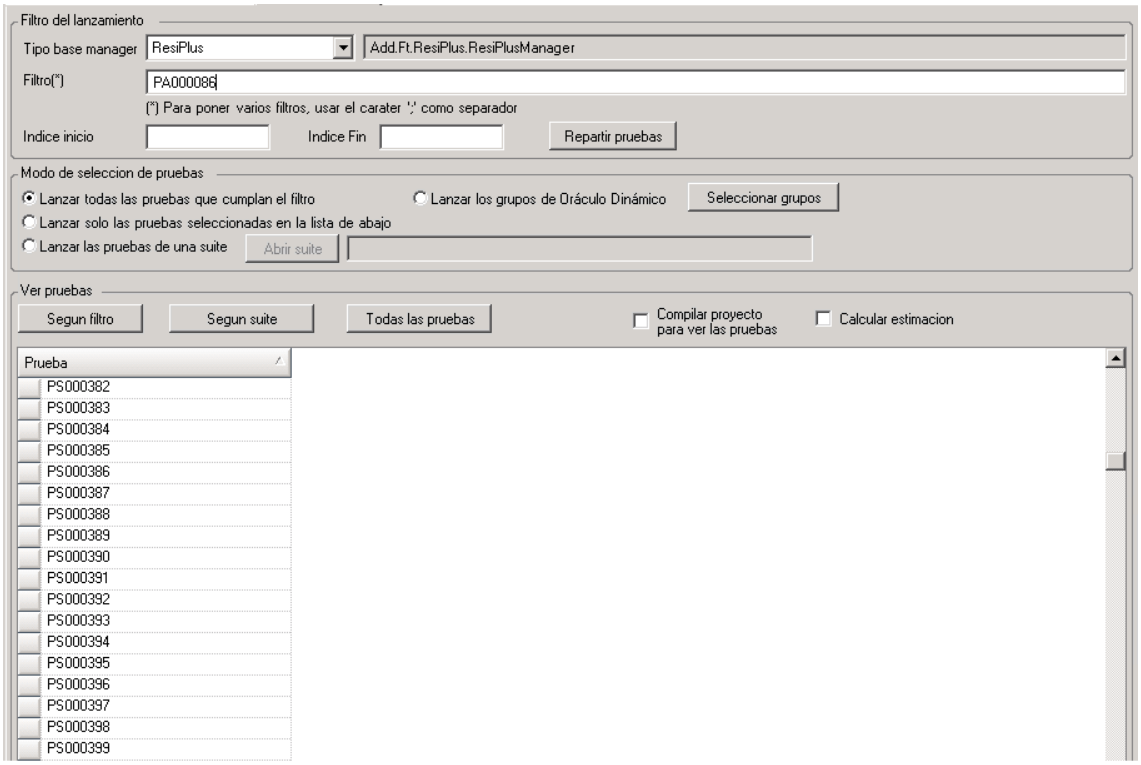


Figura 51. Pestaña Filtro Pruebas del Lanzador de Pruebas

Por cada ejecución de una PS se guarda el *Resultado*, la *FechaInicio*, la *FechaFin*, la *Máquina* en la que se ejecutó, el *Log* y el *Mensaje*. El resto de información, el *CodigoPA*, *NodoDescripcion*, que se consigue a través de TUNE-UP, el *ProgramaManager* dependerá del proyecto de automatización en el que esté creada la PS, la *PrioridadKO*, tendrá valor únicamente cuando en la ejecución se haya obtenido un FAIL y se distinguirán en:

- **Prioridad 1**: Se asignará un FAIL de P1 cuando en la última ejecución se obtuvo un PASS, es decir, es un indicador para saber qué se debe priorizar a la hora de revisar los resultados.

- **Prioridad 2**: Se asignará un FAIL de P2 cuando en alguna ejecución anterior se obtuvo un PASS.

- **Prioridad 3**: Se asignará un FAIL de P3 cuando todas las ejecuciones de la PS hayan resultado FAIL.

Codigo	CodigoPA	ProgramaManager	NodoDescripcion	Resultado	PrioridadK	Tiempo	Maquina	Mensaje
PS000382	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:08	ADDAST008	
PS000383	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:10	ADDAST014	
PS000384	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:23	ADDAST015	
PS000385	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:03:01	ADDAST007	
PS000386	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:03:14	ADDAST013	
PS000387	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:03:08	ADDAST011	
PS000388	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:21	ADDAST009	
PS000389	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:12	ADDAST008	
PS000390	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:12	ADDAST014	
PS000391	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:52	ADDAST012	
PS000392	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:03:05	ADDAST015	
PS000393	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:55	ADDAST007	
PS000394	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:25	ADDAST009	
PS000395	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:21	ADDAST013	
PS000396	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:20	ADDAST011	
PS000397	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:50	ADDAST008	
PS000398	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:03:01	ADDAST014	
PS000399	PA000086	ResiPlus	N00108 - Comprob...	PASS		00:02:46	ADDAST012	

Figura 52. Resultados de ejecución en el Lanzador de Pruebas

6.4. Gestor de Fallos

El **Gestor de fallos** se empezó a desarrollar como un proyecto de mejora interna en la empresa. El *Gestor de fallos* es un sistema que centraliza todos los fallos, permite gestionarlos y que surgió como un sistema de apoyo dentro del equipo de testeo para tener constancia de los fallos que se encontraban y poder seguir con el proceso necesario para solucionarlos.

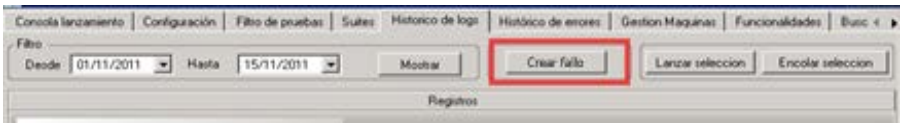


Figura 53. Formulario asociado a un fallo

La creación de fallos se puede hacer desde el propio gestor de fallos y a través del lanzador en la pestaña *Histórico de Logs*, estos fallos estarán relacionados a las PS; los fallos se crean y se asignan a un agente completando el formulario del fallo (figura 54) con toda la información necesaria y que se utilizaba para reportar a los diferentes agentes responsables a través de TUNE-UP, marcando el KO correspondiente en la PA.

Los fallos se componen de la información que se muestra en la figura 54, todo fallo tendrá una descripción que describa el comportamiento incorrecto encontrado, una severidad relacionada con el grado de importancia del fallo, quién ha detectado el fallo y en qué fase; además es posible visualizar la evolución que ha seguido el fallo, los cambios de estado hasta el actual.

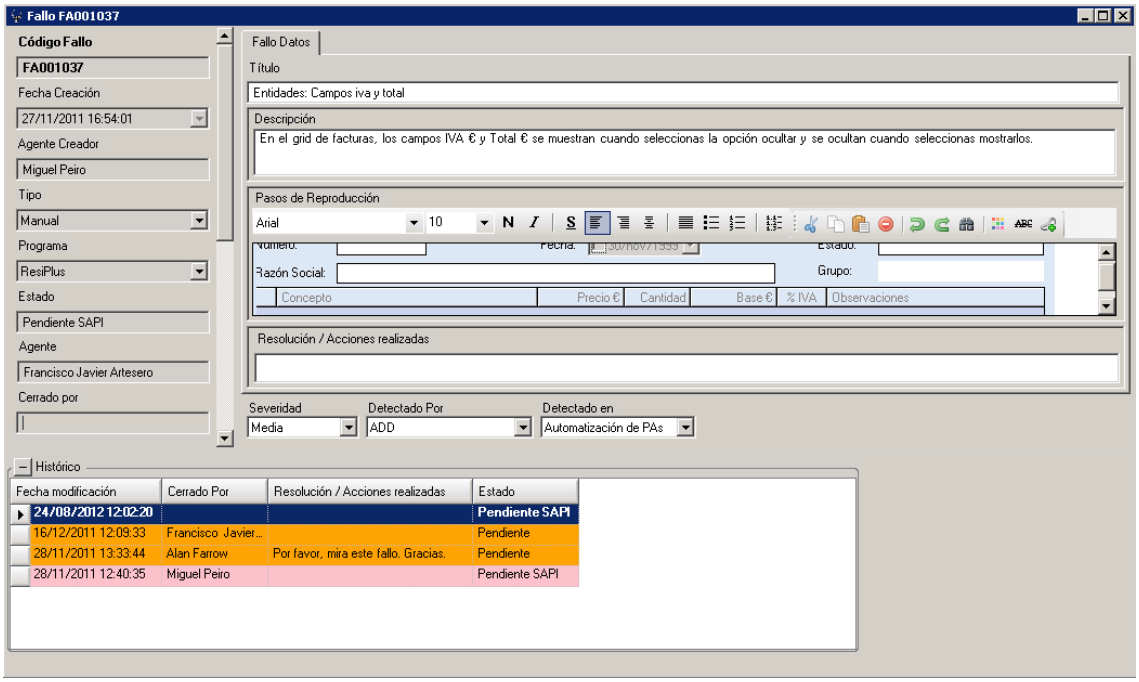


Figura 54. Formulario asociado a un fallo

Al igual que sucedió con el *Diseñador de PS*, cuando el *Gestor de fallos* llegó a una versión avanzada y estable se decidió integrar el sistema desarrollado en TUNE-UP, aún está en fase de integración y de aceptación, a diferencia del *Diseñador de PS*, que no sobrepasaba el campo de testeo ya que las pruebas de sistemas son elementos que se utilizan únicamente en testeo, la integración del *Gestor de fallos* va a afectar y va a ser utilizado por todos los roles que participan en una iteración.

Como se muestra en la figura 55 el formulario de la PA tiene una pestaña nueva "**Fallos**", donde quedan registrados todos los fallos que se hayan creado directamente en la PA o a través de alguna PS de la PA. Una vez se ha creado el fallos se marcará el KO en la PA y el agente responsable tendrá visible todos los fallos que se han introducido y que deben ser subsanados para cumplir con la descripción de la PA.

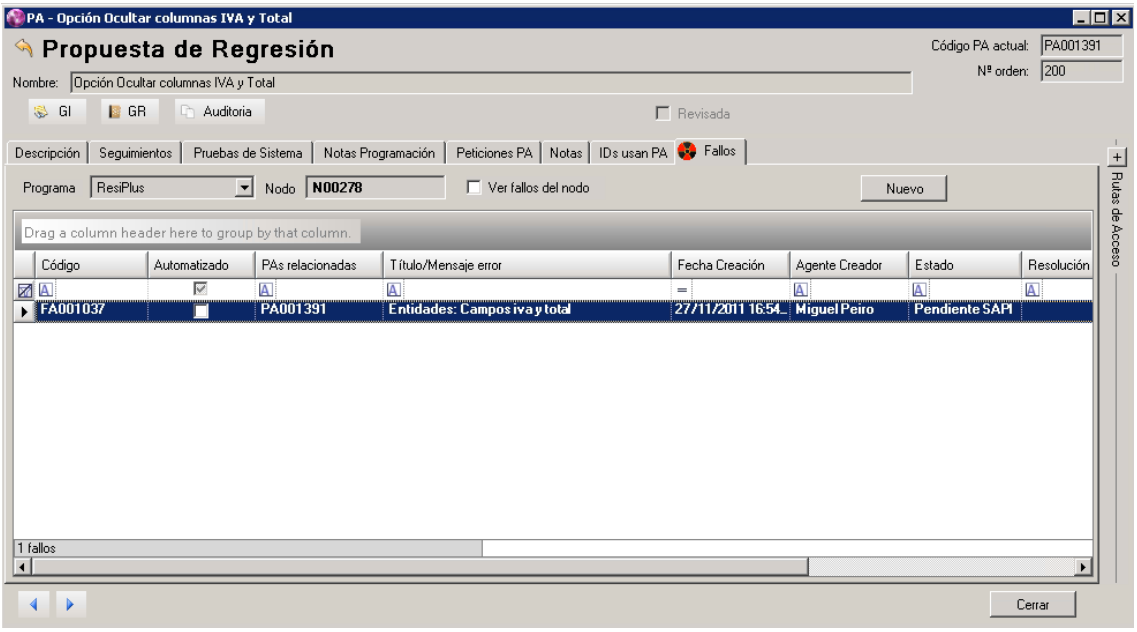


Figura 55. Pestaña Fallos en el formulario de una PA

Capítulo 7. Propuesta de Dashboard

7.1. Introducción

El proceso que se siguió es el que se muestra en la figura. Dependiendo de quién vaya a visualizar el Dashboard se deberá de mostrar los datos de una forma u otra, la interpretación de los datos será diferente para un jefe de proyecto o un cliente y por ello el punto de vista se debe adaptar a los intereses de cada uno. Por ejemplo, el cliente se interesará más por saber qué recursos se están utilizando para desarrollar una WU, el sobrecoste que pueda tener el desarrollo de una WU a lo largo de una iteración, entre otras, mientras que un jefe de proyecto pueda interesarse por qué zonas del producto son las más críticas, qué cantidad de pruebas dan fallo, etc. En nuestro caso, el Dashboard que diseñaremos tendrá un enfoque dirigido a los miembros del equipo de testeo, el *Tester* y *Automation Tester*, y el *Product Manager*.

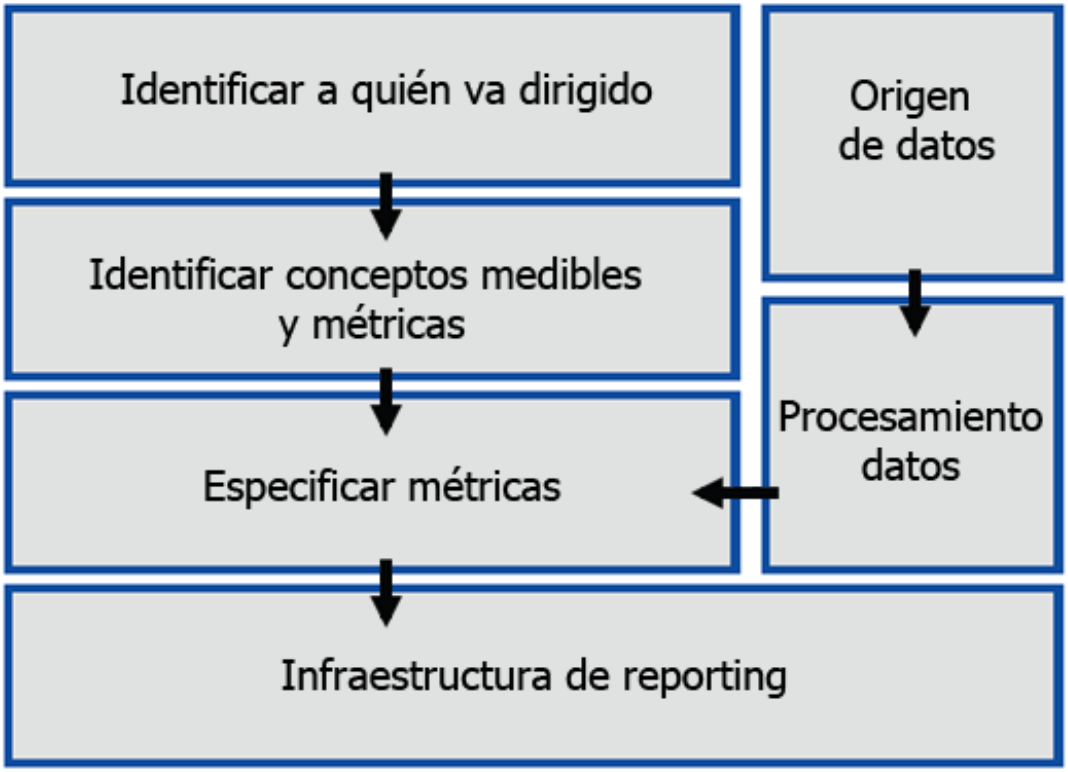


Figura 56. Proceso seguido para elaborar el Dashboard

Las métricas que se implementen dependerán del agente o agentes que vayan a utilizar el Dashboard, por ello, una vez tenemos decidido qué agente va a ser quien interpretará

las métricas, hay que conocer qué características del producto se quiere estudiar. Estas características son conceptos medibles, que según *ISO-15939 [19]* es una relación abstracta entre atributos de una entidad y la necesidad de información, donde la entidad es un producto software y un atributo puede ser el tamaño del código fuente, por ejemplo. En el próximo apartado se nombrarán algunos de los posibles conceptos medibles que se pueden tener en cuenta a la hora de elaborar las métricas. El siguiente paso es la elaboración de las métricas; se detallará la información necesaria, el origen de datos necesarios, unidades, etc. Y por último, la distribución de las métricas en el Dashboard y las distintas funciones que ofrece el Dashboard para facilitar el estudio.

Algunos de los posibles conceptos medibles que nos pueden venir a la cabeza a la hora de querer estudiar nuestro producto suelen estar dirigidos al proceso de desarrollo, la calidad del producto que se está desarrollando, costo del desarrollo del sistema, entre otros.

Una vez está decidido sobre qué se desea tener información, se debe desarrollar la infraestructura necesaria con la que se deberán apoyar las métricas para cumplir con las necesidades de supervisar el proceso de testeo que se desea.

En el próximo apartado, se van a diseñar métricas que intentarán ofrecer información sobre el proceso de desarrollo, donde tendremos métricas relacionadas con el testeo manual y automatizado; la calidad del producto, donde tendremos información detallada sobre los fallos introducidos, la tendencia en introducir fallos, etc.; el costo de desarrollo, que esta derivado al esfuerzo invertido en cada actividad de testeo.

7.2. Métricas

Esfuerzo Testeo Manual

En el *Dashboard* que TUNE-UP ofrece actualmente, disponemos del gráfico **Burn Down Chart** donde vemos la información relacionada con el esfuerzo estimado (línea representada con ), el esfuerzo invertido (línea representada con ) y el esfuerzo restante (línea representada con ). La información mostrada hará referencia a la relación de tiempos de la actividad de Testeo Manual; en TUNE-UP se recogen los tiempos computados por los agentes en las actividades correspondientes. Se ha decidido mantener esta gráfica por la información que ofrece para conocer el estado de la

iteración y como se ha ido desarrollando. Como ya se ha comentado, las situaciones de bajada o subida en el esfuerzo restante pueden explicarse por una correspondiente bajada o subida en el esfuerzo estimado, o en caso de subida, también puede deberse a una subida pronunciada en el esfuerzo invertido o bajada del esfuerzo estimado.

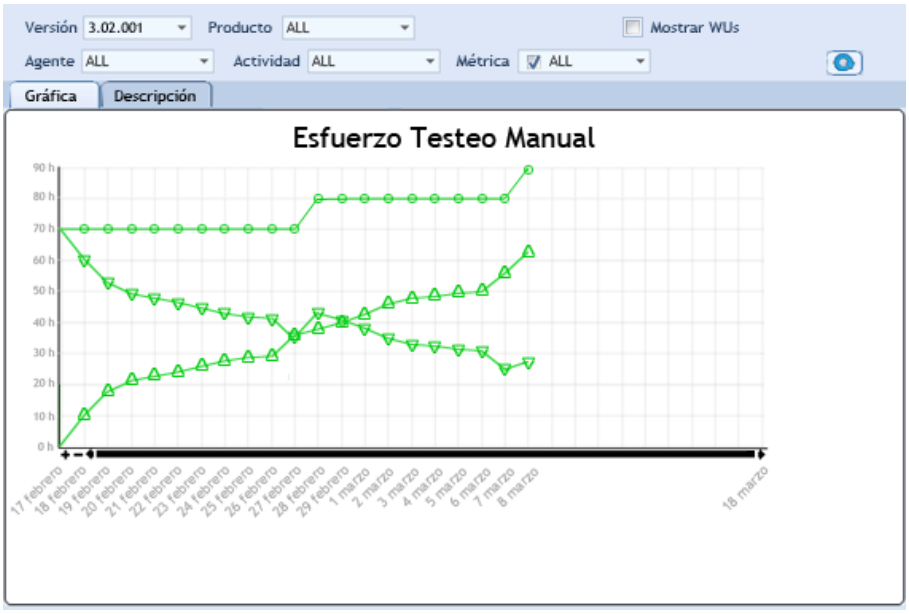


Figura 57. Vista general de Esfuerzo Testeo Manual

Esta gráfica muestra la relación de tiempos en general, cada WU en cuyo workflow haya una actividad de Testeo Manual estará incluida en esta gráfica, mostrando el sumatorio total de los tiempos de las WUs, sin hacer distinción. Ofrecer la posibilidad de identificar qué WU está ocasionando esas irregularidades nos ahorrará tiempo ya que de una manera sencilla y rápida se puede distinguir todas las WUs participantes en una iteración. Por esta razón la gráfica tendrá una opción, "**Mostrar WUs**", para que la información se muestre separada en WUs. La gráfica, como se ha comentado, muestra tres líneas con una simbología que las diferencia entre ellas; así, para distinguir entre las WUs, cada WU estará relacionado con un color, mostrando en una leyenda la correspondencia de la WU con el color. En la gráfica con la opción "*Mostrar WUs*" desmarcada, con la vista donde las WUs están separadas, tendremos la posibilidad de acceder a la gráfica de una de las WUs mostradas, ya sea pinchando en la WU en la leyenda de la derecha o en cualquier línea de la gráfica que corresponda a la WU.

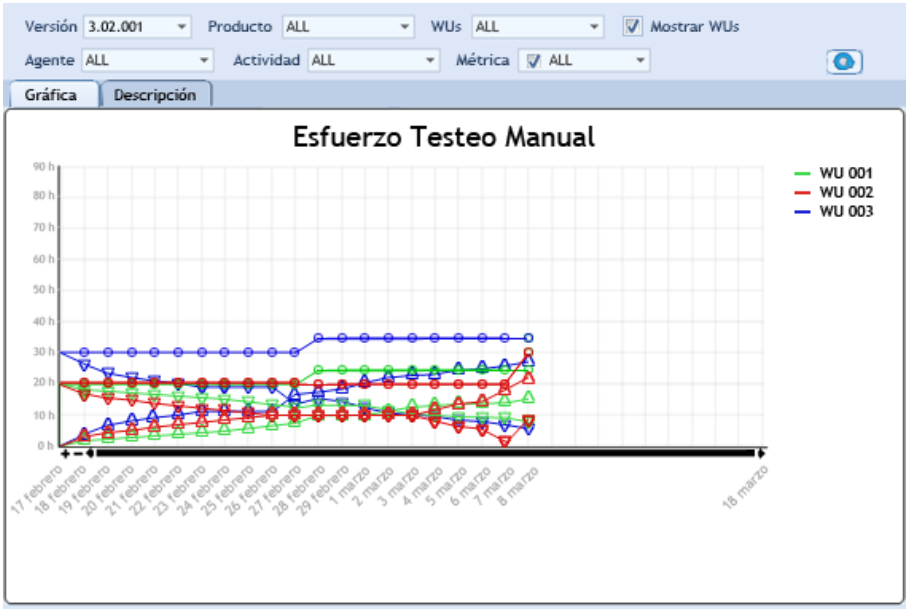


Figura 58. Vista WUs de Esfuerzo Testeo Manual

A parte de las opciones que hemos visto, la gráfica tendrá disponible el filtrado por versión, producto, métrica, agentes, por actividad y por WUs. El filtro **Métrica** se refiere a las tres líneas que se muestran "*Esfuerzo estimado*", "*Esfuerzo restante*", "*Esfuerzo inver*tido". Se dispone del filtro **Actividad** ya que a simple vista no se puede conocer si las irregularidades han sido en la actividad de *Diseño de Pruebas Manuales* o *Aplicar Pruebas Manuales*. El filtro **WUs** únicamente aparece cuando tenemos la opción "*Mostrar WUs*" activada. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando, todos los productos, todas las métricas, todos los agentes participantes en la iteración, ambas actividades mencionadas y la opción "*Mostrar WUs*" desmarcada, para visualizar el estado, en general, de la iteración.

Sin embargo es importante tener más información complementaria que nos ayude a entender la gráfica, qué hecho han causado que la gráfica tenga ese aspecto. La gráfica *Estado Ejecución Testeo Manual* puede ayudarnos a entender las posibles variaciones en la gráfica; una variación en el esfuerzo estimado puede deberse a que las pruebas fallen y se necesite volver a pasar más tarde, también puede deberse a la introducción de nuevas PAs que confirmen comportamiento que no se había tenido en cuenta.

En la pestaña descripción tendremos el siguiente texto:

“La gráfica Esfuerzo Testeo Manual permite visualizar la relación de tiempos en testeo manual; se muestra el esfuerzo estimado, el invertido hasta la fecha actual y el esfuerzo restante. Tiene una segunda vista posible en la que se distinguen las WUs que participan en la iteración seleccionada.”

Trabajo Testeo Manual

El objetivo de esta gráfica es mostrar el trabajo de testeo manual que hay para la iteración actual. En cada iteración se llevan a cabo la implementación de diferentes WUs, estas WUs tendrán asociadas unas PAs que describirán el comportamiento que debe cumplir la implementación de las WUs. Podemos tener PAs de distintos tipos, están las PAs de regresión que hacen referencia a un comportamiento implementado en alguna iteración anterior, las PAs de modificación que suelen ser PAs que ya estaban creadas pero el comportamiento que describían ha cambiado y la definición debe modificarse al actual y, por último, las PAs nuevas que son aquellas que se crean para introducir comportamiento nuevo que no se ha implementado aún.

Con estos datos podemos elaborar la gráfica circular donde podremos visualizar la cantidad total de PAs que el tester deberá pasar en la iteración seleccionada. Las distintas áreas de la gráfica corresponderán con los tipos de PAs, así sabremos el porcentaje de PAs de cada tipo que tendremos que pasar en la iteración.

En la figura 59 se muestra la gráfica circular con las áreas mencionadas anteriormente, las PAs nuevas se representarán con el quesito de color verde, las PAs de regresión serán de color amarillo y las de modificación serán de color azul.

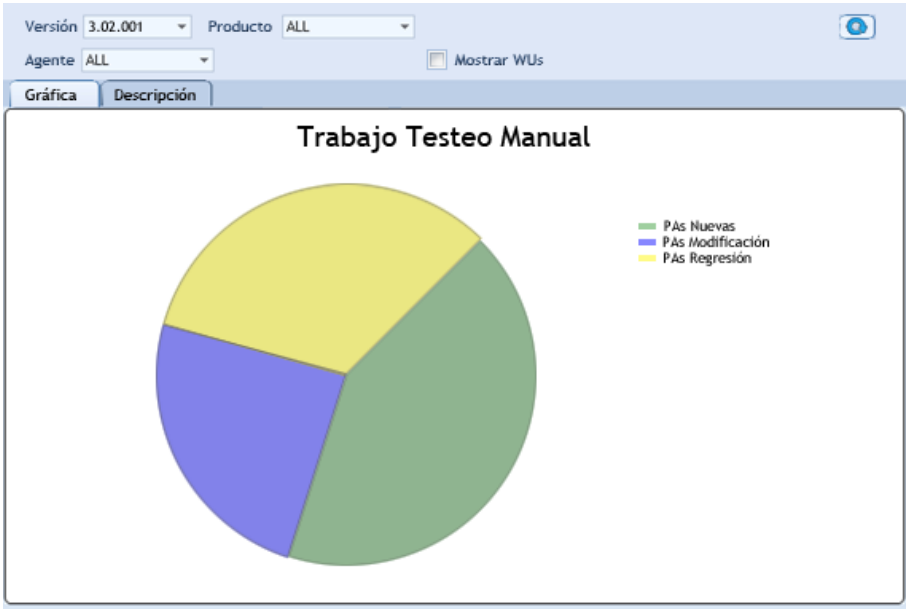


Figura 59. Vista general de Trabajo Testeo Manual

La información referente al porcentaje de los tipos de PAs nos permite saber qué cantidad de PA ya han sido implementadas y, en un caso normal, ya probadas y pasadas, que serian las PAs de regresión, y cuáles van a tener que ser revisadas al presentar nuevo comportamiento o un cambio de comportamiento. Cuanto mayor sea el porcentaje de las PAs nuevas o de modificación, mayor será el riesgo que conlleve la iteración, ya que supone nuevo código o modificación del mismo junto con las complicaciones que puede presentar.

La gráfica tendrá una opción de configuración "**Mostrar por WUs**", que cambiará la vista a una gráfica de barras donde tendremos el número de PAs de cada tipo por WU, el orden de las áreas en la barra será el que se muestra en la figura 60, en la parte más baja las pruebas que se han creado nuevas para la WU, en la parte intermedia las pruebas de modificación y en la parte más alta las pruebas de regresión.

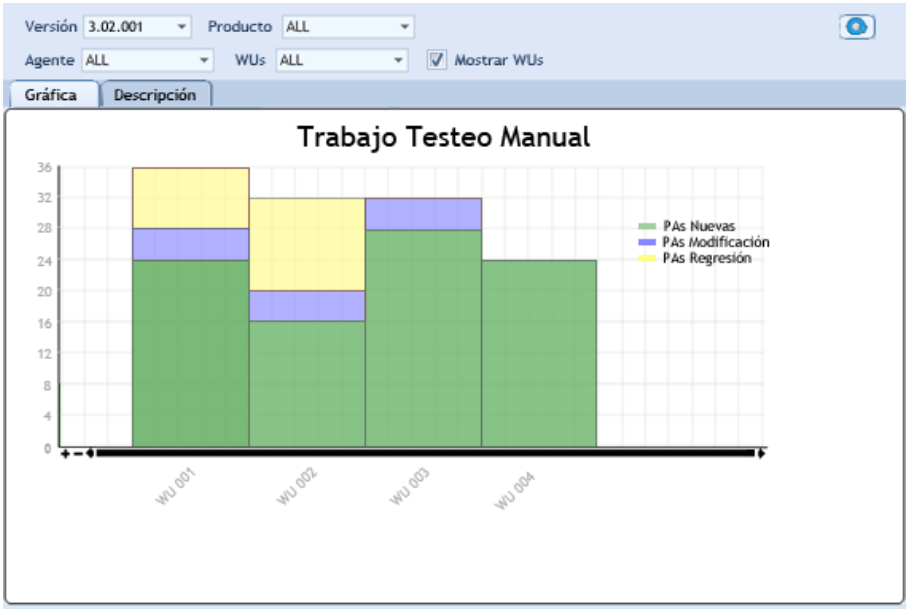


Figura 60. Vista WUs de Trabajo Testeo Manual

A parte de las opciones que hemos visto, la gráfica tendrá disponible el filtrado por versión, producto, agentes y WUs. Los filtros que tendrá la gráfica aplicados por defecto, corresponderían a la versión actual que se está desarrollando, todos los productos, todos los agentes y todas las WUs participantes de la iteración.

El filtrado por agentes mostrará una distribución de la totalidad de pruebas que tiene que pasar cada agente en la iteración seleccionada. Tenemos la posibilidad de visualizar por cada agente el número total de pruebas que deben pasar en la iteración, con la opción "*Mostrar por WUs*" desmarcado o visualizar de cada agente cuantas pruebas, separadas por las WUs, deben pasar, para ello deberemos tener la opción "*Mostrar por WUs*". Cada WU tiene asignados unos agentes; cuando se filtre por un agente se recogerá la información de las PAs que estén contenidas en las WUs del agente seleccionado, así se tiene una visión de la carga que puede tener un agente y en caso de creerlo oportuno hacer un cambio de agente responsable de la WU.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Trabajo Testeo Manual permite visualizar la cantidad de trabajo de testeo manual que se requiere en la iteración. El trabajo se contabiliza con el número de PAs que se deben pasar.
Se puede cambiar la vista de la gráfica, pudiendo tener una vista general y otra distinguiendo por WUs.

Permite el filtrado por agentes y tener conocimiento de la cantidad de trabajo que tiene cada uno.”

Estado Pruebas de Aceptación

En el *Dashboard* que TUNE-UP ofrece actualmente, disponemos de la gráfica que muestra el estado de las PAs para cada WU “**Acceptance Tests State**”. Esta gráfica se va a aprovechar ya que ofrece una vista del estado actual de las PAs. Sin embargo, es importante saber qué proceso ha seguido hasta llegar al estado que muestra la gráfica ya desarrollada. Por ello habrá una opción de configuración disponible en la gráfica, que llamaremos “**Mostrar seguimiento**”, donde podremos visualizar todos los estados por los que ha pasado el testeo manual hasta llegar al estado actual (Figura 61).

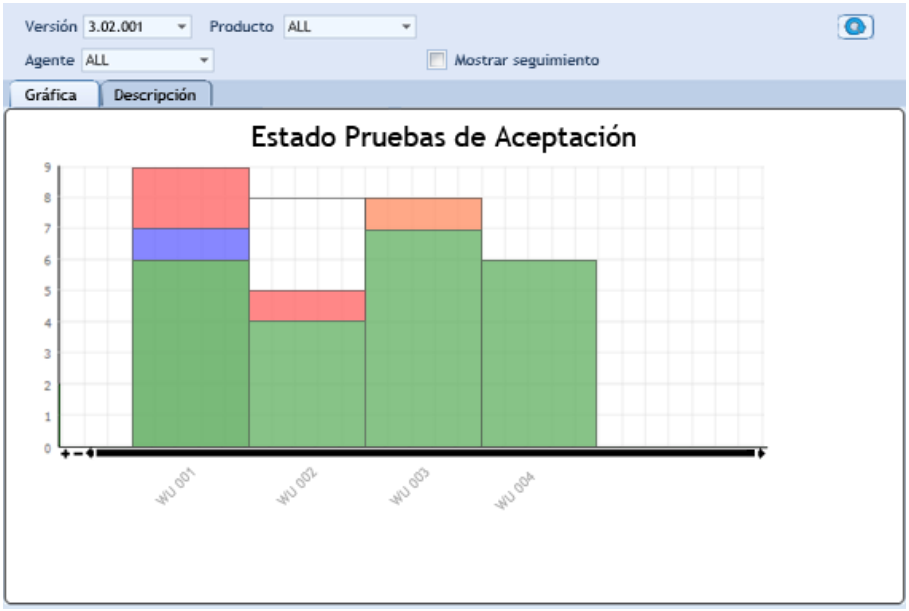


Figura 61. Vista general de Estado Pruebas de Aceptación

Esta nueva gráfica va a mostrar, a través de aéreas apiladas, la evolución de las PAs a lo largo de un tiempo especificado. En la gráfica se visualizarán cinco áreas distintas, que corresponden con los estados que se comentaron en el capitulo anterior, con los colores del mismo modo, en las que encontramos con el área en color blanco las PAs que aún están pendientes de aplicar, con el área en color rojo (KO) aquellas PAs que se han pasado y se ha observado un comportamiento incorrecto durante su aplicación, en naranja (OK!), aquellas pruebas aplicadas y marcadas como pendiente de volver a aplicar, el área en color azul (OK?) que son las pruebas que pasaron correctamente y

otro agente detectó un fallo posterior y, por último, en color verde (OK), aquellas pruebas que se han pasado correctamente.

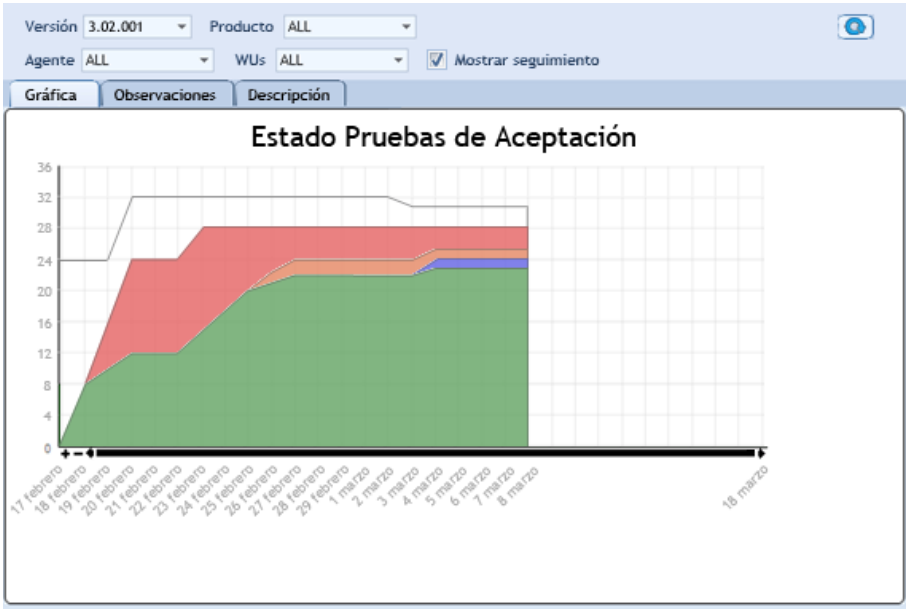


Figura 62. Vista seguimiento de Estado Pruebas de Aceptación

En esta gráfica se muestra para cada unidad de tiempo el número total de PAs que deben ser aplicadas distribuidas en los cinco estados distintos. El orden en que se van a apilar los estados será el que se muestra en la figura 62, en la zona más baja, el área de OKs, ya que la tendencia es a tener todas las PAs con un OK, en la zona superior las PAs con el estado OK?, a continuación aquellas con el estado OK!, en la siguiente aquellas PAs con estado KO y por ultimo aquellas que aun no se han aplicado.

Pero, ¿cómo se consigue plasmar el seguimiento en la gráfica? Como ya vimos en la (figura 21), cada PA tiene un histórico de ejecución donde los agentes marcan el resultado de aplicar la PA en cada fase. Consultaremos el histórico de ejecución del nivel de testeo y con ellos tendremos la información necesaria para saber qué pruebas aún no se han aplicado, ya que no tendrá ningún registro a nivel de testeo, qué pruebas tienen un KO, cuáles tienen un OK, un OK! o un OK?.

Gracias a poder visualizar la evolución de la ejecución del testeo manual, tenemos conocimiento de los fallos que se han obtenido durante la actividad de testeo, con qué rapidez se subsanan esos KO. Un dato interesante a estudiar, son los posibles aumentos y disminuciones que se puedan visualizar en el seguimiento; por norma general cuando se produzca una disminucion de alguna de las áreas mencionadas se deberá a que esa

PA se ha eliminado o desestimado; en otro caso, los cambios de un estado a otro que puedan presentar las PAs se plasman con la disminución del área del primer estado y un aumento del área del segundo estado. En el caso que se produzca un aumento en el área de las PAs pendientes de testear puede deberse a que se han añadido nuevas PAs a la WU y por lo tanto quedan pendientes, por el contrario, si lo que se muestra en una disminución sea porque alguna PA se ha desestimado o eliminado. En el caso en que estas irregularidades se correspondan con el área de los KO, en ocasiones los cambios suelen afectar o estar afectados por cambios en el área de los OK, en el caso de un aumento en el área de KO, se puede deber a un aumento sustancial de PAs que han resultado KO que puede afectar a una disminución en el área de los OK, OK! u OK?; en el caso de una disminución en el área de KO, suele estar en relación con la "reparación" del KO de alguna PA.

Con el objetivo de facilitar la lectura de la gráfica se introduce una nueva pestaña que llamaremos "**Observaciones**"; en esta pestaña tendremos detallados los cambios que se realizan cada día que se muestran en la gráfica, nuevas PAs añadidas, eliminadas. Para conseguir esta funcionalidad, se realizará un procesamiento de los cambios que han afectado a las PAs y se informará al usuario sobre las causas que han originado la gráfica.

A parte de las opciones que hemos visto, la gráfica tendrá disponible el filtrado por versión, producto, agentes y por WUs. El filtro **WUs** únicamente aparece cuando tenemos la opción "*Mostrar seguimiento*" activada. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando, todos los productos, todos los agentes participantes en la iteración y la opción "*Mostrar seguimiento*" desmarcada, para visualizar el estado, en general, de la iteración.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Estado Ejecución Testeo Manual permite visualizar el estado de los resultados actualizados de testeo manual. Está disponible una vista donde se muestra la evolución que se ha obtenido al pasar las pruebas."

Esfuerzo Testeo Automatizado

Esta gráfica es similar a la de *Esfuerzo Testeo Manual* mostrando la información sobre el proceso de automatización. Al igual que en la gráfica anterior, disponemos de la

información relacionada con el esfuerzo estimado (línea representada con), el esfuerzo invertido (línea representada con) y el esfuerzo restante (línea representada con).

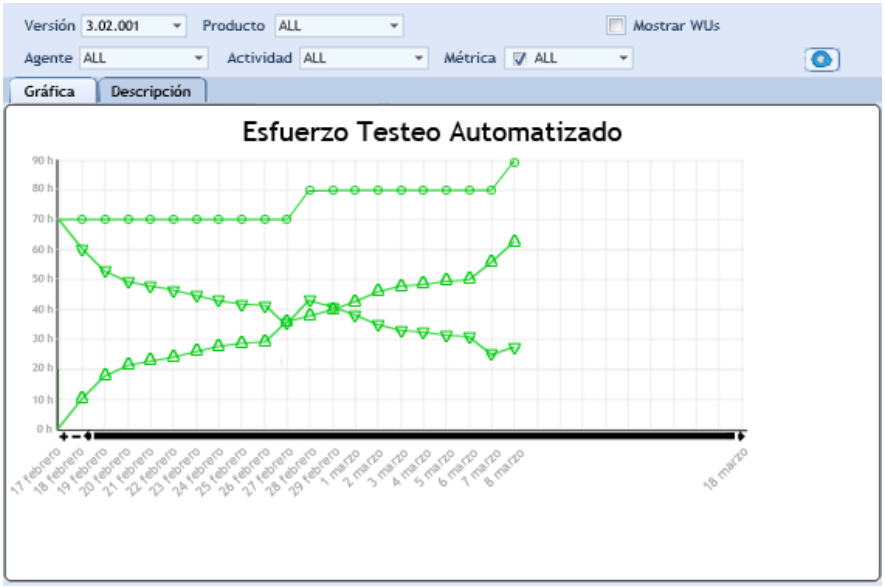


Figura 63. Vista general de Esfuerzo Testeo Automatizado

Tanbien contará con la opción "**Mostrar WUs**" con el fin de distinguir las WUs que se desarrollan en la iteración seleccionada. Tendremos la posibilidad de acceder a la gráfica de una de las WUs mostradas, ya sea pinchando en la WU en la leyenda de la derecha o en cualquier línea de la gráfica que corresponda a la WU. En la siguiente imagen se puede ver el resultado de filtrar por la *WU 003*.

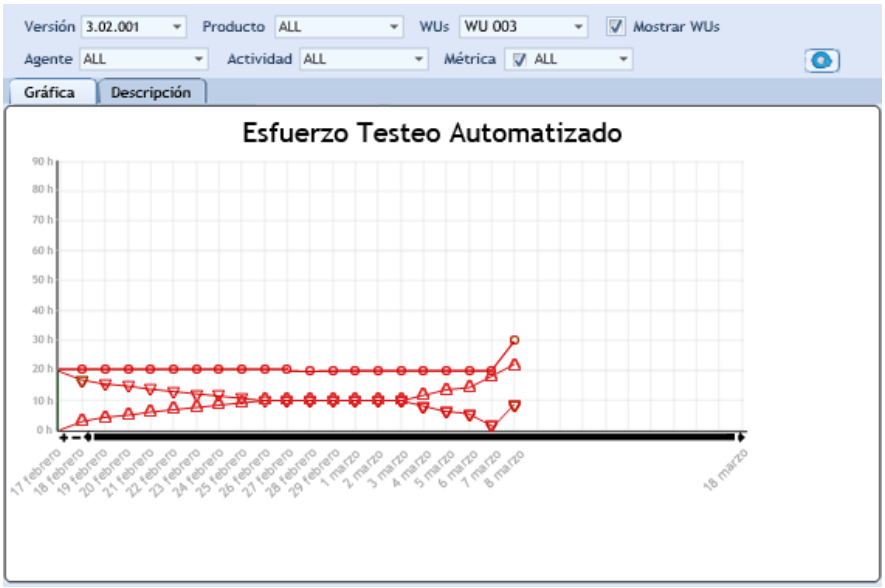


Figura 64. Vista WU de Esfuerzo Testeo Automatizado

A parte de las opciones que hemos visto, la gráfica tendrá disponible el filtrado por versión, producto, métrica, agentes, por actividad y por WUs. El filtro **Métrica** se refiere a las tres líneas que se muestran "*Esfuerzo estimado*", "*Esfuerzo restante*", "*Esfuerzo invertido*". Se dispone del filtro **Actividad** ya que a simple vista no se puede conocer si las irregularidades han sido en la actividad de *Diseño de Pruebas Automatizadas*, *Automatización de Pruebas* o *Aplicar Pruebas Automatizadas*. El filtro **WUs** únicamente aparece cuando tenemos la opción "*Mostrar WUs*" activada. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando, todos los productos, todas las métricas, todos los agentes participantes en la iteración, las tres actividades mencionadas y la opción "*Mostrar WUs*" desmarcada, para visualizar el estado, en general, de la iteración.

Sin embargo es importante tener más información complementaria que nos ayude a entender la gráfica, qué hecho han causado que la gráfica tenga ese aspecto. La gráfica *Estado Testeo Automatizado* puede ayudarnos a entender las posibles variaciones en la gráfica; una variación en el esfuerzo estimado puede deberse a que se hayan creado más pruebas de sistema y por lo tanto se haya tenido que invertir más esfuerzo en la automatización de las mismas.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Esfuerzo Testeo Automatizado permite visualizar la relación de tiempos en testeo automatizado; se muestra el esfuerzo estimado, el invertido hasta la fecha actual y el esfuerzo restante. Tiene una segunda vista posible en la que se distinguen las WUs que participan en la iteración seleccionada."

Disponibilidad de Pruebas de Sistema

Esta gráfica muestra la información referente al estado de la automatización. En una iteración, se desarrollan una cierta cantidad de WUs, las WUs expresan el comportamiento que debe cumplir el producto definiendo PAs. Al inicio de la iteración se decide que PAs se van a automatizar en la iteración y se marcan como tal.

Al igual que la gráfica *Estado Ejecución Testeo Manual* da la posibilidad de tener una visión del estado actual de la automatización y otra visión sobre la evolución que ha tenido la automatización a lo largo del tiempo.

La vista del estado actual (Figura 65) estará compuesta por una gráfica de barras en la que se contabilizará el número de PSs de cada WUs incluida en la iteración. En la leyenda se observa los estados que pueden tener las PSs; por orden de aparición, en primer lugar aparecen las PSs que están automatizadas, en segundo lugar las que tienen el diseño elaborado a falta de automatizar y, por último, las que aún están pendientes de realizar el diseño. Sin embargo, cuando se está realizando el diseño de las PAs que se van a automatizar, no se sabe con exactitud cuántas PSs se vayan a diseñar, por ello las PAs que no tengan ninguna PS creada se contabilizará como una más, ya que al estar marcada para automatizar tendrá como mínimo una PS creada.

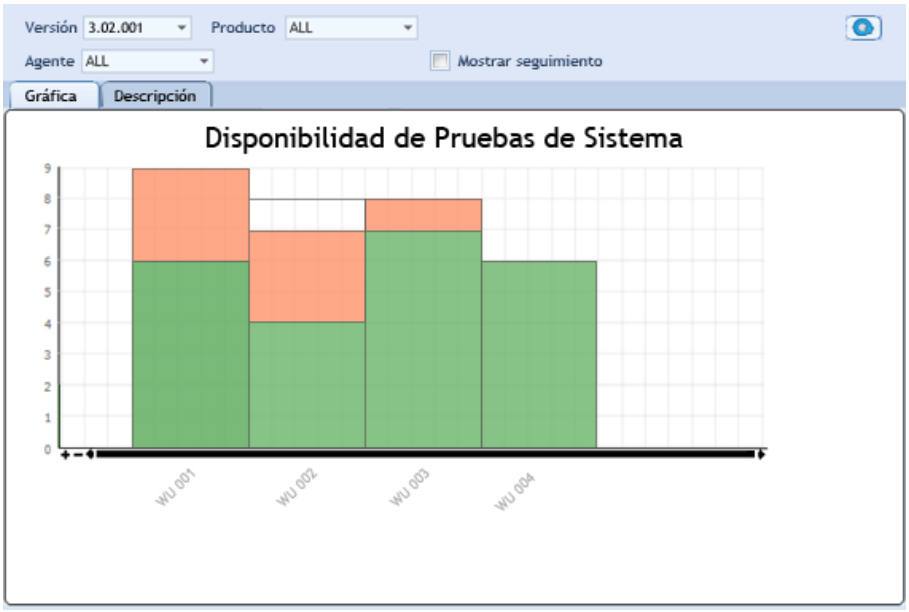


Figura 65. Vista general de Disponibilidad de Pruebas de Sistema

En la gráfica tenemos la misma opción que en la gráfica *Estado Ejecución Testeo Manual*, la opción "**Mostrar seguimiento**", que al marcarla se mostrará la evolución del estado de la automatización a lo largo de la iteración. Esta vista estará compuesta por una gráfica de áreas apiladas. Las áreas que formarán la gráfica se corresponden con los estados anteriormente mencionados, que son los que se pueden ver en la leyenda.

El orden de las áreas apiladas es el mismo al que tenemos en la vista del estado actual, en la parte más baja, en color verde, se mostrará el área que corresponde con las PSs automatizadas, en la zona intermedia, en color naranja, se mostrará el área que corresponde con las PSs creadas y diseñadas pero pendiente de automatizar y, por último, en color gris y en la zona más alta el área de las que están pendientes de diseño.

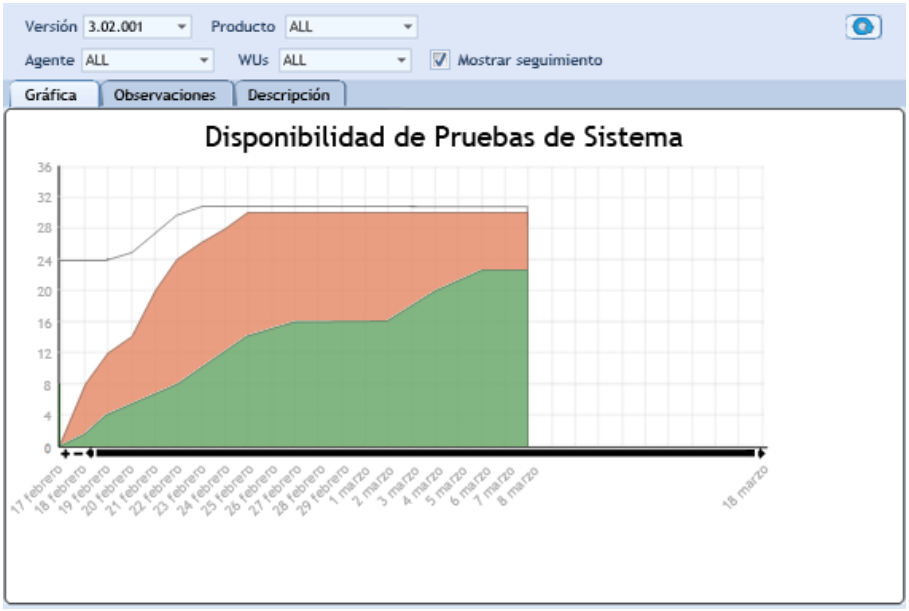


Figura 66. Vista seguimiento de Disponibilidad de Pruebas de Sistema

Los posibles aumentos y disminuciones que se puedan visualizar en el seguimiento, en el caso que se produzca un aumento en el área de PS pendiente de diseñar puede deberse a que se han añadido nuevas PAs a automatizar, por el contrario, si lo que se muestra en una disminución sea porque alguna PA, que no tenía ninguna PS creada, se ha desestimado o eliminado, en el caso en que estas irregularidades se correspondan con el área de las PSs diseñadas pendientes de automatizar, en el caso de un aumento en el área es resultado de crear nuevas PSs y tener nuevos diseños pendientes de automatizar, una disminución del área puede deberse a dos razones, una que hubiesen diseños pendientes de automatizar y la PA se eliminó o que los diseños se hayan automatizado y en ese caso se observaría un aumento de PSs automatizadas también, y por último, en el área de las pruebas automatizadas, una aumento refleja la automatización de los diseños creados y una disminución, que se hayan marcado como eliminado PSs.

En la actualidad, podemos acceder a buena parte de la información necesaria para elaborar esta gráfica. Para montar la gráfica necesitamos saber qué PAs se van a automatizar, para dar una solución aproximada si nos encontramos con el caso en que estamos en pleno proceso de diseño de pruebas y hay PAs sin diseñar, así, como ya se ha comentado, asignaremos una PS por defecto a las PAs marcadas para automatizar en las que aún no se han diseñado ninguna PS. El único dato que, en la actualidad no se puede conocer, es la fecha en la que una PS se marca como eliminada; para ello se procederá a realizar un cambio en la tabla de la base de datos, se añadirá un campo

nuevo, que llamaremos "**FechaEliminacion**", en la tabla donde se guardará la fecha en la que se eliminó la PS.

A parte de las opciones que hemos visto, la gráfica tendrá disponible el filtrado por versión, producto, agentes y por WUs. El filtro **WUs** únicamente aparece cuando tenemos la opción "*Mostrar seguimiento*" activada. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando, todos los productos, todos los agentes participantes en la iteración y la opción "*Mostrar seguimiento*" desmarcada, para visualizar el estado, en general, de la iteración.

Al igual que en la gráfica *Estado Ejecución Testeo Manual* se introduce una nueva pestaña que llamaremos "**Observaciones**"; en esta pestaña tendremos detallados los cambios que se realizan cada día que se muestran en la gráfica, nuevas PSs creadas, eliminadas, etc.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Estado Testeo Automatizado permite visualizar el estado de las pruebas automatizadas de testeo automatizado. Está disponible una vista donde se muestra la evolución que se ha seguido diseñando y automatizando las pruebas de sistema."

Estado Ejecución de Suites

En esta gráfica vamos a poder visualizar los resultados obtenidos en la ejecución de pruebas automatizadas a través de las suites creadas. Como ya se comentó en el capitulo anterior, las *suites* son un conjunto de pruebas que cumplen unos filtros concretos; en cada iteración se crean una serie de suites que contienen las pruebas automatizadas que se están implementando y la ejecución de estas suites serán las que utilizaremos en esta gráfica para recopilar información.

La gráfica se representará con un gráfico de tipo barras, donde se mostrará el resultado de la ejecución de las suites disponibles. En esta gráfica se mostrarán las últimas ejecuciones de cada suite. En una ejecución de un conjunto de pruebas se pueden dar diferentes resultados de ejecución, los que tenemos definidos son los siguientes:

- Resultado **PASS**, cuando una prueba ha pasado con éxito todos los puntos de verificación que tiene el script.

- Resultado **FAIL**, cuando una prueba ha fallado en algún punto de verificación del script, o se ha producido un comportamiento no esperado, y posiblemente incorrecto de la aplicación, que ha hecho que la ejecución se haya abortado.
- Resultado **WARN**, cuando una prueba ha pasado con éxito todos los puntos de verificación que tiene el script, pero el entorno de ejecución no es correcto, por ejemplo un mapa de objeto sin actualizar, la aplicación abierta al finalizar la ejecución de la prueba, etc., son casos que no impiden la ejecución pero que el *Automation Tester* debe corregir.
- Y, por último, en ocasiones se da la situación que el resultado de una prueba no se ha podido guardar, en estos casos en que no se conoce el resultado los trataremos como **UNKNOWN**.

En la gráfica se pueden distinguir por el color que le corresponde a cada resultado, para las ejecuciones con resultado *PASS* las identificaremos por el color verde, para las que hayan resultado *FAIL* con el color rojo, para las ejecuciones en *WARN* con el color amarillo y para las ejecuciones *UNKNOWN* con el color gris, tal y como se muestra en la figura 67.

Figura 67. Vista general de Estado Ejecución de suites

Al pasar el ratón por encima aparecerá un "bocadillo" con información exacta; en esta información encontramos el número total de pruebas de sistema que constituyen la suite, la cantidad de pruebas que han resultado *PASS*, las que ha dado un *WARN* y las que acabaron de forma incorrecta, las *FAIL*. En el capítulo anterior ya se mencionó que

en los registros *FAILs* se distinguían tres prioridades; también se añadirá esta información junto con la anterior. Además, se mostrará la cantidad de *FAILs* con un fallo relacionado; que podremos considerar que se están revisando y que se solucionarán en breve.

En cada barra correspondiente a una suite podremos ver las últimas ejecuciones con el componente disponible *Histórico de Logs*. Para ello, cada barra tendrá disponible un menú contextual que tendrá una única opción "**Mostrar últimas ejecuciones**" que se mostrarán en el componente del Lanzador que se podrá incluir.

En esta gráfica tendrá disponible entre las opciones de filtrado, el filtrado por versión y por suites. Los valores del filtro por **Versión** serán ["Todas", "Versión N", "Versión N-1",…], donde *N* es la versión actual. Por otra parte, los valores del filtro por suites serán las suites disponibles ordenadas descendientemente según el orden de creación. Los filtros que tendrá la gráfica aplicados por defecto, corresponderían a la versión actual que se está desarrollando y todas las *suites* creadas para esa versión. Sin embargo, no disponemos de este dato en la actualidad, para ello hay varias soluciones posibles, una es guardar la fecha en la que se creó la suite y calcular a qué versión pertenece y la otra asignarle la versión como característica directa; puesto que hay suites que no pertenecen a ninguna versión en concreto, como por ejemplo, el conjunto de pruebas automatizadas que ya se han automatizado y cuyas WUs se han finalizado, pero que, por seguridad, se deberán ejecutar en cada iteración; por ello, se va a guardar la versión directamente en la suite y las opciones posibles de versión al crear la suite serán las mismas que en la gráfica; este cambio se llevará a cabo añadiendo en la tabla donde se guardan las suites creadas un nuevo campo que llamaremos "**VersionAsociada**". Ahora que podemos saber a qué versión pertenece cada suite, cuando se filtre por la versión deseada, los valores del filtro por suites quedarán reducidos a aquellos que pertenezcan a la misma.

Al filtrar por una *suite* o en la gráfica de la figura 68 haciendo clic en una suite, se mostrará una gráfica circular individual con la información de los resultados de la última ejecución de la suite. Tendrá los mismos elementos que la vista anterior distinguidos por los mismos colores, teniendo el mismo comportamiento que la barra de la gráfica anterior con el menú contextual.

Figura 68. Vista Suite de Estado Ejecución de suites de pruebas automatizadas

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Ejecución de suites de pruebas automatizadas permite visualizar el resultado de las suites creadas. Es posible filtrar por versión y por suite.
Se puede combinar con el componente Histórico de Logs. Añadir."

Si el Dashboard ya presenta el componente Histórico de Logs no se mostrará el enlace *Añadir*.

Criticidad Producto

Los niveles de criticidad están relacionados con la severidad y la frecuencia del mal funcionamiento del producto software, de forma que cuanto más severos y frecuentes sean los fallos del producto más alto será el riesgo y por conclusión mayor nivel de criticidad presentará.

En TUNE-UP, como ya se ha descrito en el capitulo anterior, disponemos del *Requirements Manager*, donde cada nodo de esta estructura es un contenedor de comportamiento expresado como PAs. En el nuevo Dashboard vamos a aprovechar esta estructura de nodos para mostrar información sobre el producto.

Los niveles de criticidad que se van a establecer tendrán estos valores:

- **Nivel crítico**: Obteniendo un valor entre 81 y 100 puntos de criticidad.

- **Nivel alto**: Obteniendo un valor entre 61 y 80 puntos de criticidad.
- **Nivel medio**: Obteniendo un valor entre 41 y 60 puntos de criticidad.
- **Nivel bajo**: Obteniendo un valor entre 0 y 40 puntos de criticidad.

Utilizaremos estos criterios para calificar a los nodos del árbol con el nivel de criticidad que le corresponde. El cálculo del mismo se hará gracias a los fallos que quedan registrados en las PAs y que están contenidas en los nodos del *REM*. Al registrar un nuevo fallo, se le asigna una severidad del fallo cuyos valores (*Crítica*, *Alta*, *Media* y *Baja*) coinciden con los niveles mencionados anteriormente. Para realizar el cálculo asignaremos un valor a las opciones posibles de severidad de un fallo, los fallos con severidad crítica que estén abiertos tendrá el valor 100, para los fallos cerrados el valor 90, los fallos con severidad alta que estén abiertos tendrán el valor 80, mientras que los que estén cerrados tendrán el valor 70, los fallos con severidad media que estén abiertos tendrán el valor 60, mientras que los que estén cerrados tendrán el valor 50 y para la severidad baja los fallos que estén abiertos tendrán el valor 40 y para los que estén cerrados 20 puntos. La fórmula que se va a utilizar para calcular el nivel de criticidad será la siguiente:

Dado un nodo seleccionado, se recorrerán todos los nodos hijos no repetidos y se sumará la severidad de todos los fallos encontrados que no se hayan desestimado divididos entre el número total de fallos.

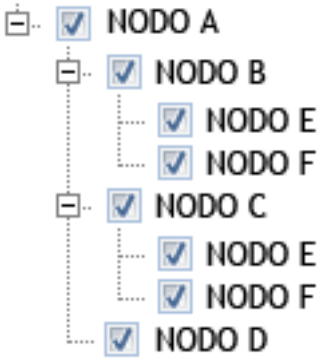


Figura 69. Ejemplo de estructura de nodos

Pongamos como ejemplo el de la figura 69, donde tenemos la siguiente situación:

Nodo A	1 fallo Alto cerrado (70 pts)
Nodo B	2 fallos Medio cerrados (50+50=100 pts)
Nodo C	0 fallos

Nodo D	1 fallo Crítico cerrad (90 pts)
Nodo E	1 fallos Bajo (20)
Nodo F	1 fallo Alto abierto (80 pts)

Para conseguir una calificación óptima, si existen nodos repetidos, se contabilizarán únicamente una vez; así pues, el cálculo del porcentaje de criticidad de los nodos de la figura 69 sería el siguiente:

Nodo A	(70+100+90+20+80)/6= 60 pts	Medio
Nodo B	(100+20+80)/4=50 pts	Medio
Nodo C	(20+80)/2= 50 pts	Medio
Nodo D	(90)=90 pts	Crítico
Nodo E	(20)=20 pts	Bajo
Nodo F	(80)= 80 pts	Alto

En el árbol habrá un indicador criticidad para cada nodo, este indicador constará de un cuadrado relleno de un color asociado al nivel de criticidad que tiene ese nodo. Los colores que se corresponden con cada nivel de criticidad serán los siguientes:

Crítico	
Alto	
Medio	
Bajo	

Cuando el cursor pase por encima del indicador de criticidad, se visualizará la información referente a la puntuación exacta que ha obtenido junto con el número de fallos que existen de cada tipo de severidad (Figura 70).

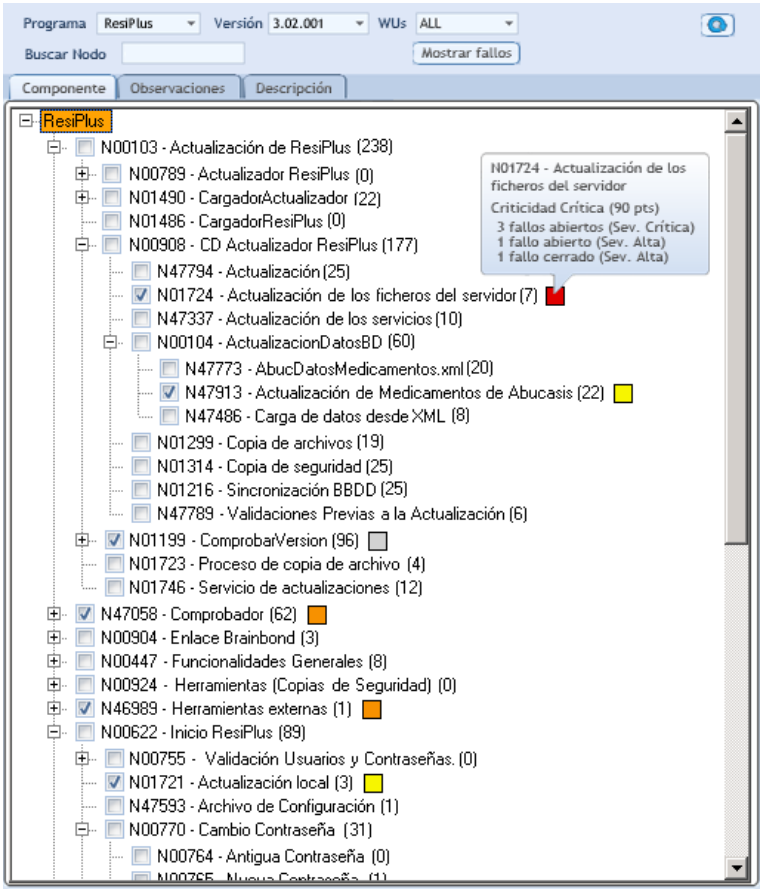


Figura 70. Vista general de Criticidad Producto

Al marcar un *checkbox* del árbol aparecerá al lado derecho un indicador que nos mostrará el nivel de criticidad; al pasar el cursor por encima del indicador se podrá visualizar la información detallada del resultado obtenido, junto con los fallos que han dado valor al nivel de criticidad del nodo. Como se puede observar, existe un botón "*Mostrar fallos*" con el que podremos visualizar la información de los fallos que contienen los nodos seleccionados en la gráfica *Análisis de fallos* y en el *Componente del Grid de fallos*, de las que se hablarán más adelante. El botón estará habilitada si hay algún nodo seleccionado y si, al mismo tiempo, hay alguna gráfica de las mencionadas en el Dashboard.

En la pestaña "**Observaciones**" tendremos el nivel de criticidad general de toda la versión seleccionada y de forma más detallada, para cada nodo se podrá visualizar el mismo contenido que se puede ver al pasar el cursor por encima de los indicadores de criticidad de los nodos.

Una característica importante de conocer la criticidad de cada nodo del *REM* reside en tener información sobre futuras versiones y poder anticiparse a posibles problemas

durante el desarrollo de la iteración y reducir el riesgo. Además del nivel de criticidad, también tenemos disponible el número de Pruebas de Sistema automatizadas en cada nodo, dando la posibilidad al equipo de testeo a localizar aquellas zonas claves que sean criticas y con poca automatización, para diseñar nuevas pruebas de sistema y automatizarlas.

El componente tendrá disponible el filtrado por programa, versión, WUs y buscador de nodos, para facilitar la localización de nodos. Los filtros que tendrá la gráfica aplicados por defecto, corresponderían a la versión actual que se está desarrollando, todas las WUs participantes de la iteración. Al cambiar la versión del filtro automáticamente se seleccionarán aquellos nodos que participen en el conjunto de WUs que participan en ella.

En la pestaña descripción tendremos el siguiente texto:

"El componente Criticidad del producto permite visualizar el nivel de criticidad que presenta el producto. Se utiliza la estructura en árbol definida en el REM. El nivel de criticidad se calcula en base al nivel de severidad de los fallos registrados en las PAs y PSs contenidos en los nodos del árbol.
Se puede combinar con la gráfica Análisis de fallos. Añadir.
Se puede combinar con el componente Grid de fallos. Añadir."

Si el Dashboard ya presenta el componente Grid de fallos o la gráfica Análisis de fallos no se mostrará el enlace *Añadir* correspondiente.

Estado de fallos

Es importante tener conocimiento sobre el estado de los fallos que presenta el producto. Esta gráfica pretende mostrar una visión actual simplificada del estado de los fallos y, por otra parte, mostrar la evolución de los fallos dentro de un intervalo de tiempo especificado.

En primer lugar, se va a describir la gráfica que mostrará la visión simplificada. Esta gráfica se representará con un gráfico circular, donde la información mostrada depende del filtro de las fechas inicio y fin. Tanto en esta visión como en la que se explicará más adelante, la de seguimiento, únicamente se mostrarán los fallos solucionados en el intervalo de tiempo marcado; decisión que se ha tomado para centrarnos en datos

importantes, ya que, en caso de no omitir esos registros, tendríamos una gráfica desfigurada debido a que, conforme van finalizando iteraciones, se habrán creado fallos y solucionado, y presentaría fallos solucionados en un tiempo pasado.

En la gráfica (Figura 71) se presentarán estos 5 elementos, los fallos que se hayan **Solucionado** o **Desestimado** en el intervalo de tiempo seleccionado, los fallos que están **Pendiente TUNE-UP** y, por último, los estados que se utilizarán solamente en el proceso de automatización, los estados **Pendiente** y **Revisar**.

Los colores asociados a cada elemento, que se mostrarán en la leyenda de la gráfica, son naranja para el estado *Pendiente*, rosa para el estado *Pendiente TUNE-UP*, verde para el estado *Revisar*, gris para el estado *Cerrado* y negro para el estado *Desestimado*.

Figura 71. Vista general de Estado de fallos

Al igual que en las gráficas mencionadas anteriormente que mostraban una evolución, tendremos esa misma opción, la opción "**Mostrar seguimiento**", que al marcarla se mostrará la evolución del estado de los fallos entre las fechas indicadas en los filtros. Esta vista estará compuesta por una gráfica de áreas apiladas. Las áreas que formarán la gráfica se corresponden a los mismos estados anteriormente mencionados, que son los que se pueden ver en la leyenda.

El orden de las áreas apiladas será: en la parte más baja, en color gris, se mostrará el área que corresponde a los fallos con el estado *Cerrado*, en la zona inmediatamente superior, en color negro, se mostrarán los fallos con el estado *Desestimado*, la siguiente

zona, en color naranja, el área de los fallos con el estado *Pendiente*, la siguiente, en color verde, el área de los fallos con el estado *Revisar* y, por último, en color rosa, se mostrará el área que corresponde a los fallos con el estado *Pendiente TUNE-UP*.

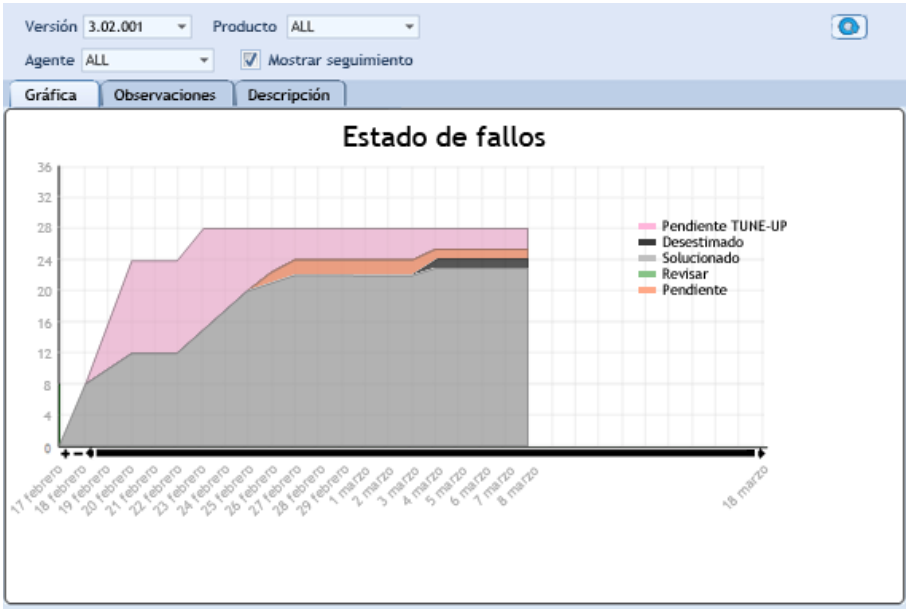


Figura 72. Vista seguimiento de Estado de fallos

Los elementos que observamos en la gráfica son los estados posibles a los que puede ir cambiando un fallo a lo largo de su ciclo de vida, por lo tanto, lo más probable cuando se observen posibles aumentos y disminuciones en algún estado provocarán el efecto invertido en otro estado. Los únicos casos particulares que se pueden encontrar en la gráfica se darán cuando se observe una disminución en algún estado y ningún aumento en otro, esto indicará que el fallo ha sido desestimado; por otro lado, cuando se observe un aumento en algún estado y ninguna disminución en otro, esto indicará que se ha creado un nuevo fallo.

En el *Gestor de Fallos*, el agente correspondiente tiene la posibilidad de cambiar de estado el fallo. Estos cambios son guardados teniendo disponible un histórico del fallo. Cada registro del fallos se guarda con la fecha del cambio de estado, el agente destinatario y la resolución que se ha obtenido; a partir del histórico de los fallos podemos elaborar esta grafica, consultando las fechas en las que se hicieron los cambios de estado del fallo.

Al igual que en otras gráficas comentadas anteriormente, con la opción "*Mostrar seguimiento*" tendremos disponible la pestaña "**Observaciones**" en la que se mostrarán

posibles explicaciones a las variaciones que pueda presentar la gráfica. En esta pestaña se observará un área de texto, de sólo lectura, donde se mostrará, cuando haya cambios y agrupado por días, la información que explique las causas de esos cambios.

A parte de las opciones que hemos visto, la árbol tendrá disponible el filtrado por versión, producto, agentes. Los filtros que tendrá la gráfica aplicados por defecto, corresponderían a la versión actual que se está desarrollando, todos los productos y todos los agentes.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Estado de fallos permite visualizar en qué estado se encuentran los fallos. Además, se puede visualizar la evolución de los fallos durante la iteración seleccionada."

Tendencia de fallos

Esta gráfica muestra la tendencia a introducir fallos a lo largo de todas las iteraciones, cuantos fallos se solucionan en ellas y los que están pendientes de solucionar. Está formada por una gráfica compuesta, por una parte se mostrará una línea cuyos valores de la línea son los números de fallos introducidos en una versión, por otra parte, para cada iteración se mostrará una barra que hará referencia a los fallos solucionados o desestimados, que serán aquellos fallos cuya fecha registrada está dentro del intervalo de tiempo de la versión, y los que aún no se han solucionado. Los fallos solucionados, que se representan con la barra, no tienen porque hacer referencia a los fallos introducidos en la iteración, puede encontrarse un caso en que un fallo se haya introducido en una iteración y se conozca, pero por falta de recursos no se solucione en la misma iteración y se decida arreglar en una iteración posterior. En estas situaciones se crean WU cuyo único objetivo es corregir fallos introducidos en iteraciones anteriores, de ahí que podamos tener iteraciones que tengan más fallos solucionados que introducidos.

En la figura 73 se muestra el aspecto de la grafica, la línea mostrará el seguimiento de fallos obtenidos a lo largo de las iteraciones y las barras pueden presentar dos áreas, los fallos solucionados o desestimados en la iteración estarán representados con el color gris, mientras que los fallos introducidos en esa versión se mostrarán de rojo.

Figura 73. Vista general de Tendencia de fallos

El valor de los fallos solucionados o desestimados se calcula directamente con los registros guardados en la base de datos, al igual que los fallos introducidos en la iteración y los fallos pendientes de solucionar, sin embargo los fallos pendientes de solucionar siguen un patrón para comprobar que el valor es consistente; el valor obtenido sobre los fallos pendientes a solucionar (F.P.) corresponde al número de fallos pendientes acumulados en la iteración anterior más el número de fallos introducidos (F.I.) en la iteración actual menos los fallos solucionados (F.S.).

$$F.P._{\text{Versión N}} = F.P._{\text{Versión N-1}} + F.I._{\text{Versión N}} - F.S._{\text{Versión N}}$$

En cada versión se podrá acceder a la información detallada de los fallos introducidos, pendientes y solucionados gracias a la gráfica *Análisis de fallos*; para ello, en cada versión, la línea referente a los fallos introducidos y las barras de los fallos solucionados y pendientes dispondrán de un menú contextual con la opción "**Mostrar en Análisis de fallos**" que refrescará el contenido de la gráfica. Esta opción estará habilitada si la gráfica está añadida al Dashboard.

Esta gráfica tendrá únicamente la opción de filtrar por versiones, habrá un filtro para la versión inicial y otro para la versión final, que constituirán el intervalo de versiones que se representarán en la gráfica, además de filtrar por producto. Los filtros que tendrá la gráfica aplicados por defecto, corresponderían a la versión actual en "**Versión Final**", cuatro versiones anteriores en "**Versión Inicio**" y todos los productos.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Tendencia de fallos permite visualizar la tendencia de introducción de fallos en las iteraciones junto con los fallos que se han solucionado y los que aún están abierto.
Se puede combinar con la gráfica Análisis de fallos. Añadir."

Si el Dashboard ya presenta la gráfica Análisis de fallos no se mostrará el enlace *Añadir*.

Análisis de fallos

Esta gráfica tiene como objetivo presentar un conjunto de gráficas que permitan poder analizar el estado de los fallos de una iteración seleccionada. Para cada iteración podremos conocer con más detalles los fallos que se han introducido. En esta nueva gráfica tendremos cuatro gráficas circulares mostrando diferente información cada una.

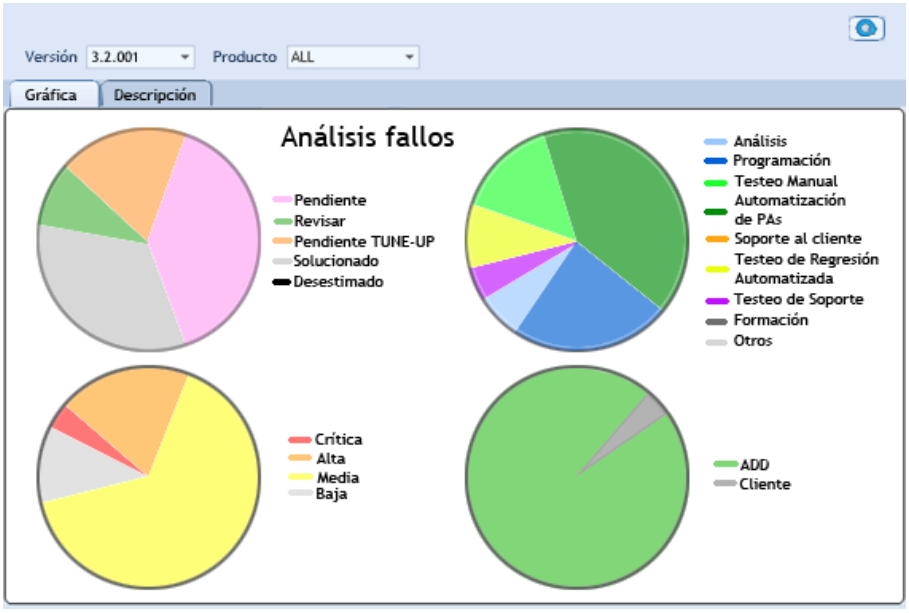


Figura 74. Vista general de Análisis de fallos

En la esquina superior izquierda, se mostrará una gráfica equivalente a la vista general que tendremos en *Estado de fallos*, detallando en porcentajes ***los estados actuales en los que se encuentran los fallos***.

En la esquina superior derecha, se mostrará en porcentajes la proporción de ***las fases en las que se introdujeron los fallos de la iteración***. Las fases en las que se pueden detectar fallos son los siguientes:

- Análisis
- Programación
- Testeo Manual
- Automatización de PAs
- Soporte al cliente
- Testeo de Regresión Automatizado
- Testeo de Soporte
- Formación
- Otros

Los colores para reconocer las fases en la gráfica serán azul claro para *Análisis*, azul oscuro para *Programación*, verde claro para *Testeo Manual*, verde oscuro para *Automatización de PAs*, naranja para *Soporte al cliente*, amarillo para *Testeo de Regresión Automatizado*, violeta para *Testeo de Soporte*, gris oscuro para *Formación* y gris claro para *Otros*. La situación que se debe evitar o que no es idónea, es tener un porcentaje moderadamente alto en *Soporte al cliente* y *Formación*, por estar en contacto directo con el cliente en el momento que se detectó el fallo. Por otra parte, cuanto más tarde se detecte el fallo, menos tiempo y menos recursos existen para reaccionar y solucionarlo antes de finalizar la iteración.

En la esquina inferior izquierda, se mostrará en porcentajes la proporción de ***la severidad que existe en el conjunto de fallos de la iteración***. Como se ha comentado anteriormente, los niveles de severidad existentes son:

- Crítica
- Alta
- Media
- Baja

Los colores para reconocer los niveles de severidad serán los mismos que se mencionaron anteriormente, rojo para la severidad *Crítica*, naranja para la *Alta*, amarillo para la *Media* y, por último, amarillo claro para la *Baja*. En esta gráfica, la situación a evitar es el alto porcentaje de fallos críticos o altos, ya que suelen conllevar a un retrabajo en programación, en análisis o testeo, dependiendo de en qué fase se han encontrado.

Por último, en la inferior derecha, se mostrará en porcentajes la proporción de ***quién ha detectado el conjunto de fallos de la iteración***. Para esta gráfica únicamente tendremos dos valores que son:

- ADD
- Cliente

Los colores para reconocer quién ha detecta el fallo serán verde para hacer referencia a ADD y gris para el Cliente. Lógicamente, en esta ocasión la situación a evitar es el alto porcentaje de fallos reportados por el cliente.

Las cuatro gráficas sirven de filtros entre ellas para profundizar en los detalles de los fallos. Tenemos cuatro gráficas mostrando información en porcentaje sobre la fase en la que se introdujeron los fallos de una iteración, en qué estado se encuentran, qué severidad tienen o quién los detectó. Así, en esta misma gráfica podremos conocer la relación entre la severidad, el estado, la fase en la que se introdujo y quién los detectó.

En cada gráfica podremos seleccionar de qué área queremos conocer la información; por ejemplo, como se puede ver en la gráfica (Figura 75), pincharemos sobre la gráfica de *Quién detectó* en el área correspondiente a *Cliente*; al pinchar sobre la zona *Cliente*, las otras tres gráficas se refrescarán actualizando la información y proporcionando información más detallada.

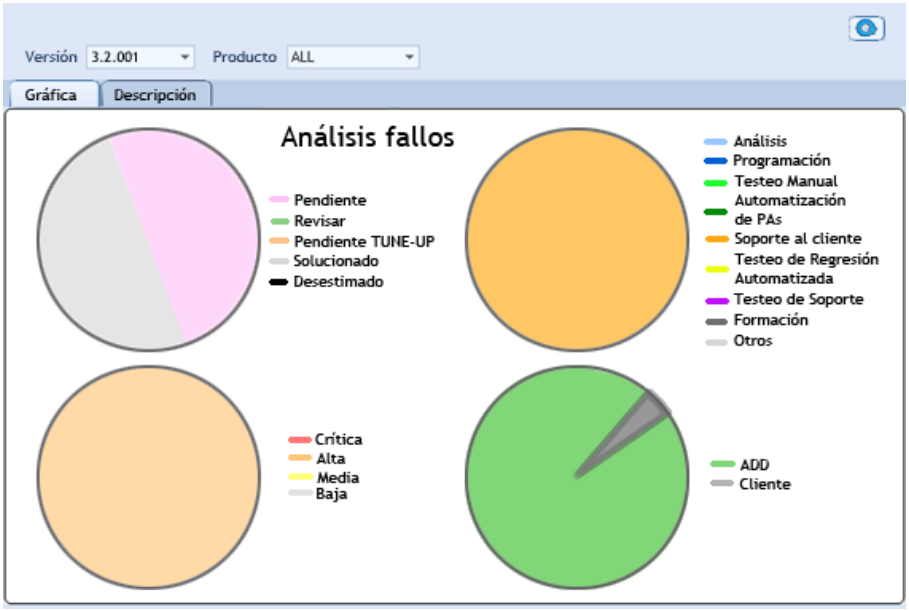


Figura 75. Vista filtrada por Cliente de Análisis de fallos

Habrá un botón actualizar con el que podremos volver a ver las cuatro gráficas con los porcentajes que hacen referencia a todos los fallos de la iteración.

Esta gráfica se podrá combinar con el componente Grid de fallos; cada área de cada una de las cuatro gráficas tendrá un menú contextual con la opción "**Mostrar en Grid de fallos**", donde podremos ver el listado de fallos.

Estas gráficas únicamente tendrán como opción de filtrado la versión y el producto. Como ya se ha comentado, entre ellas se pueden filtrar para conocer en detalle, un subconjunto de cualquier gráfica, la información relacionada con las otras gráficas.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Análisis de fallos permite visualizar cuatro gráficas que ofrecen distinta información sobre los fallos asociados a una versión:
- Los estados actuales de los fallos.
- Las fases en las que se introdujeron.
- La severidad de los fallos.
- Quién detectó los fallos.
Se puede combinar con el componente Grid de fallos. Añadir.
Se puede combinar con la gráfica Tendencia de fallos. Añadir.
Se puede combinar con el componente Criticidad del producto. Añadir."

Si el Dashboard ya presenta el componente Grid de fallos, la gráfica Tendencia de fallos o el componente Criticidad del producto no se mostrará el enlace *Añadir* correspondiente.

Tendencia esfuerzo invertido

Esta gráfica nos muestra la el esfuerzo que se ha realizado en las ultimas iteraciones para finalizar toda la actividad de testeo de las WUs que se desarrollaron. Podemos identificar que iteraciones han supuesto un esfuerzo importante. Se trata de una gráfica lineal donde por cada iteración se contabilizará el tiempo total invertido en cada una de las actividades de testeo de cada WU desarrollada en la iteración.

Figura 76. Vista general de Tendencia esfuerzo invertido

Las opciones que tendrá disponible son el filtrado por versión inicio, versión final, producto, agente y actividad. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando para *Versión Final*, cuatro versiones anteriores a la actual para *Versión Inicio* y todos los productos.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Tendencia esfuerzo invertido permite visualizar la tendencia entre iteraciones sobre el esfuerzo invertido en el proceso de testeo."

Tendencia Pruebas Aceptación

Esta gráfica muestra la tendencia que se ha tenido entre unas iteraciones sobre cuánto trabajo de testeo manual se ha realizado. En una iteración existen distintas WUs que se deben implementar; en estas WUs se definen el comportamiento en términos de PAs.

Los estados que componen las barras coinciden con los de la gráfica *Trabajo Testeo Manual*, ya que lo que se muestra es el conjunto total de PAs, diferenciando los tipos de PA, que se han aplicado en una iteración.

Figura 77. Vista general de Tendencia Pruebas Aceptación

Las opciones que tendrá disponible son el filtrado por versión inicio, versión final y producto. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando para *Versión Final*, cuatro versiones anteriores a la actual para *Versión Inicio* y todos los productos.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Tendencia Pruebas Aceptación permite visualizar la tendencia entre iteraciones sobre el trabajo de testeo manual que se realiza, contabilizándolo por el numero de Pruebas de Aceptación."

Tendencia Pruebas de Sistema

En esta grafica se muestra la tendencia en las últimas iteraciones a la hora de automatizar pruebas de sistema. Se trata de una gráfica lineal, donde para cada iteración se contabilizan el número de pruebas de sistema automatizadas.

Con esta grafica podemos saber si en las ultimas iteraciones se ha hecho un trabajo de automatización alto o bajo. Esta gráfica la podemos combinar con la gráfica Tendencia Prueba Aceptación para ver la correlacion del número de Pruebas de Sistema con el total de PAs aplicadas en la iteración; cuántas más PAs nuevas más probabilidad habrá de que el número de PSs sea alto.

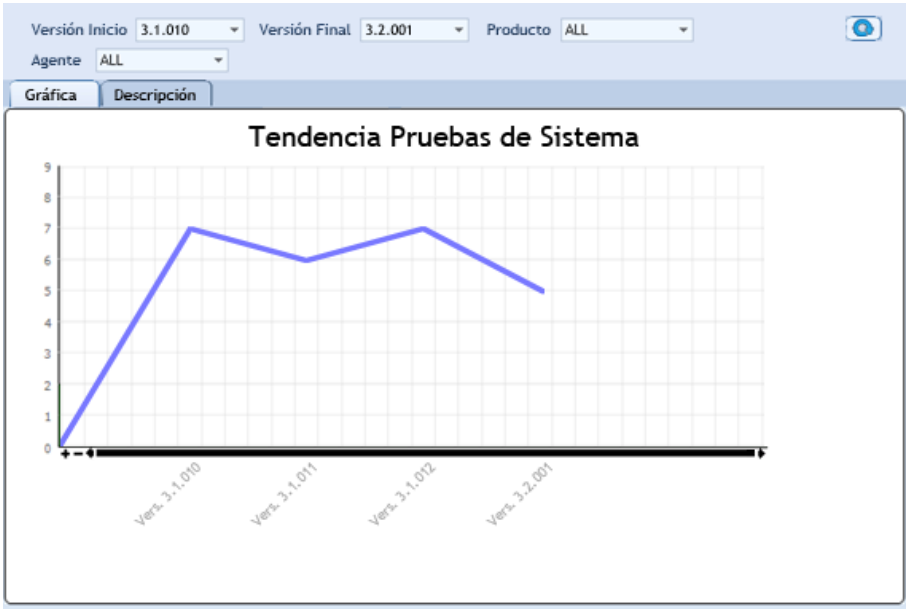


Figura 78. Vista general de Tendencia Pruebas de Sistema

Las opciones que tendrá disponible son el filtrado por versión inicio, versión final y producto. Los filtros que tendrá la gráfica por defecto corresponderían a la versión actual que se está desarrollando para *Versión Final*, cuatro versiones anteriores a la actual para *Versión Inicio* y todos los productos.

En la pestaña descripción tendremos el siguiente texto:

"La gráfica Tendencias Pruebas de Sistema permite visualizar la tendencia entre iteraciones a la hora de automatizar Pruebas de Sistema. En cada iteración se mostrará el número Pruebas de Sistema que se han automatizado."

Componente Grid de fallos

El *Gestor de fallos* ya se comentó en el capítulo anterior, es una herramienta de apoyo en la que podemos visualizar todos los fallos registrados, crear nuevos y modificar los existentes.

Se va a integrar en el Dashboard una parte del *Gestor de fallos*, en concreto, tendremos la posibilidad de visualizar el grid de fallos pudiendo abrirlos en el formulario de fallos para ver la descripción del mismo. No es apropiado dar la posibilidad de crear nuevos fallos ni modificarlos, ya que lo único que se pretende con esta integración es recoger y visualizar información.

El grid presentará opciones de filtrado para modificar el contenido del mismo. Sin embargo, dado que el grid ya ofrece la función de filtrar el contenido por columna es innecesario añadir una opción de filtrado por *Programa*, *Agente*, por ejemplo, se podría hacer directamente filtrando por el *Programa* que se desee desde la propia columna. Las únicas opciones de filtrados útiles serán las fechas de inicio y fin, que filtrarán los fallos mostrando aquellos que se hayan creado dentro del intervalo y aquellos fallos que no se hayan solucionado antes de la fecha de inicio.

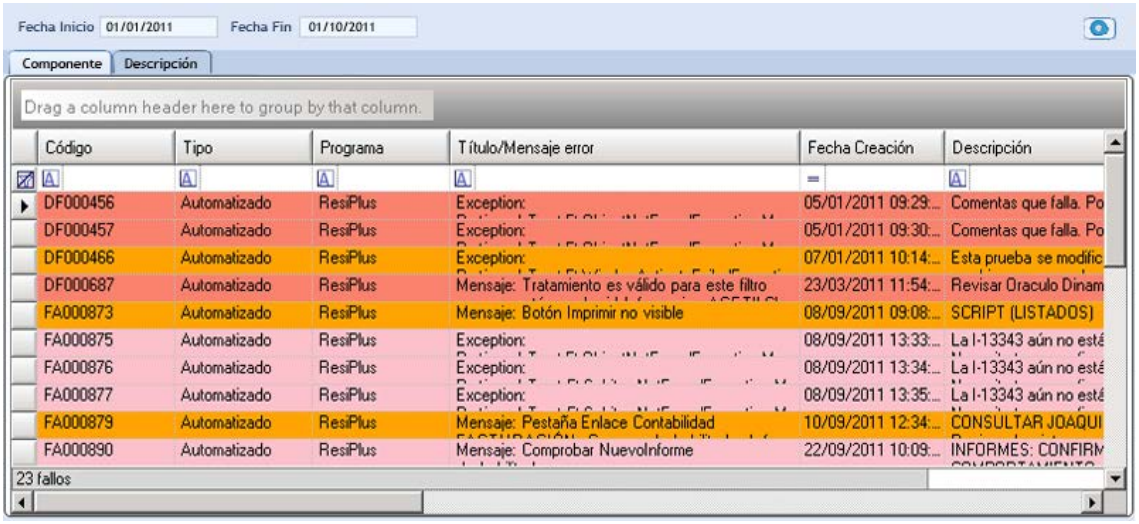


Figura 79. Vista general de Componente Grid de fallos

En la pestaña descripción tendremos el siguiente texto:

"El componente Grid de fallos permite visualizar el listado de fallos que se introdujeron entre las fechas dadas. Se puede abrir cualquier fallo del grid para ver toda su información en el formulario correspondiente.
Se puede combinar con el componente Criticidad del producto. Añadir.
Se puede combinar con la gráfica Análisis de fallos. Añadir."

Si el Dashboard ya presenta el componente Criticidad del producto o la gráfica Análisis de fallos no se mostrará el enlace *Añadir* correspondiente.

Componente Histórico de logs

El *Lazandor de Pruebas* engloba muchas funciones, para el Dashboard utilizaremos el grid de *Histórico de Logs*; este grid nos permite visualizar los resultados de la ejecución de las pruebas automatizadas.

Como se muestra en la figura 80 se va a importar el grid junto con las opciones de filtrado; este componente ofrecerá la posibilidad de filtrar por fechas. Cuando se filtra por fechas, el resultado contiene todas aquellas baterías de pruebas que se ejecutaron dentro de ese intervalo de tiempo.

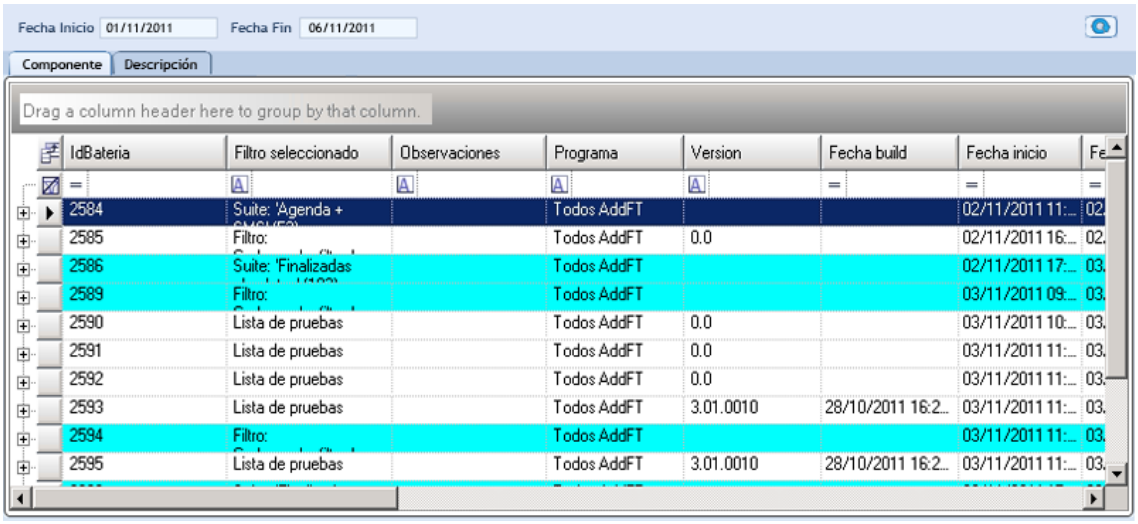


Figura 80. Vista general de Componente Histórico de logs

Sin embargo, como ya se mencionó anteriormente en la grafica *Estado Ejecución de Suites*, es posible la conexión entre el componente del Lanzador y la gráfica. Cuando en la gráfica se quiera visualizar las últimas ejecuciones de una suite, en el grid se mostrarán las diez últimas ejecuciones, si las tiene, ordenados por fecha de ejecución.

En la pestaña descripción tendremos el siguiente texto:

"El componente del Histórico de Logs permite visualizar el resultado de las ejecuciones de las baterías dentro de un intervalo de tiempo seleccionado.
Se puede combinar con la gráfica Ejecución de suites de pruebas automatizadas. Añadir."

Si el Dashboard ya presenta la gráfica Ejecución de suites de pruebas automatizadas no se mostrará el enlace *Añadir*.

7.3. Entorno de Dashboards

En el apartado anterior se han descrito los elementos disponibles para el Dashboard que queremos diseñar. En este apartado se va a presentar el prototipo diseñado del entorno

donde podremos elaborar Dashboards, las diferentes funciones que ofrecerá y cómo trabajar con él.

En primer lugar, habrá que modificar la interfaz principal de TUNE-UP para poder acceder al Dashboard. En el menú principal se añadirá un nuevo ítem en el menú de "**Planificador de Versiones**" al que nombraremos "**Entorno Dashboards**".

Al abrir el Entorno de Dashboards se mostrará el formulario que vemos en la figura 81; en la parte superior estarán las opciones disponibles que tendremos, en la parte de la izquierda habrá un panel oculto con los componentes presentados en el apartado anterior y, en la parte central, se presentarán los Dashboards que podemos confeccionar separados por pestañas.

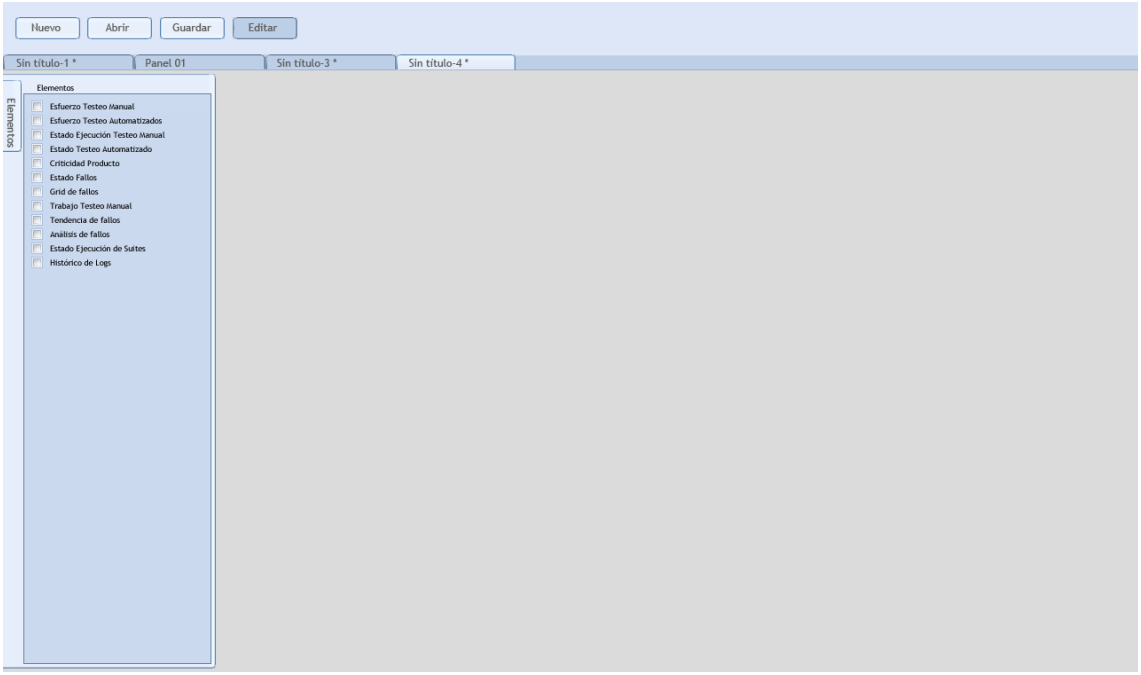


Figura 81. Formulario principal de la propuesta de Dashboard

Entre las opciones disponibles tendremos la posibilidad de crear nuevos Dashboards, editar o abrir algún Dashboard creado anteriormente y guardar cambios que se hayan hecho en el Dashboard. Para poder ofrecer esta funcionalidad, necesitaremos tener los datos guardados para poder trabajar con ellos.

En la base de datos existente para TUNE-UP se creará una nueva tabla, que llamaremos "*TUNEUPDashboards*", con tres campos, por ahora, que serán un campo identificador de tipo autonumérico al que llamaremos "*id*", un campo de texto al que llamaremos "*nombre*" y otro campo de texto que llamaremos "*descripcion*". El campo "*id*" se

rellenará automáticamente al guardar un nuevo Dashboard, el campo "*nombre*" será el valor que se le asigne por interfaz y el campo "*descripcion*" contendrá un texto adicional que se crea conveniente para identificar el contenido del Dashboard.

En la parte superior tendremos cuatro botones con los que podremos realizar las acciones comentadas, el primer botón será el que nos permita crear un Dashboard y tendrá de texto "**Nuevo**", el segundo botón nos permitirá abrir Dashboards guardados y tendrá de texto "**Abrir**", el tercero nos permitirá guardar los cambios que se hayan podido producir en un Dashboard y tendrá de texto "**Guardar**" y el ultimo nos permitirá editar las opciones del Dashboard y tendrá de texto "**Editar**". Los botones "*Nuevo*" y "*Abrir*" estarán siempre habilitados; el botón "*Guardar*" estará habilitado únicamente cuando tengamos cambios pendientes, como con un cuadro de mando recién creado, una gráfica añadida o se hayan cambiado las opciones del Dashboard, entre otros; el botón "*Editar*" estará habilitado cuando tengamos activo una cuadro de mando guardado, esto excluye los recién creados sin haberlos guardado.

Al apretar a "*Nuevo*" se creará una nueva pestaña en el panel central con el nombre "*Sin título-i **" donde "*i*" será el siguiente valor numérico al que pueda existir en el conjunto de pestañas, es decir, si tenemos tres pestañas, la primera con el nombre "*Sin título-1 **", la segunda con el nombre "*Panel 01*" y la tercera con el nombre "*Sin título-3 **", al apretar sobre "*Nuevo*" se creará el Dashboard con el nombre "*Sin título-4 **" quedando la pestaña activa.

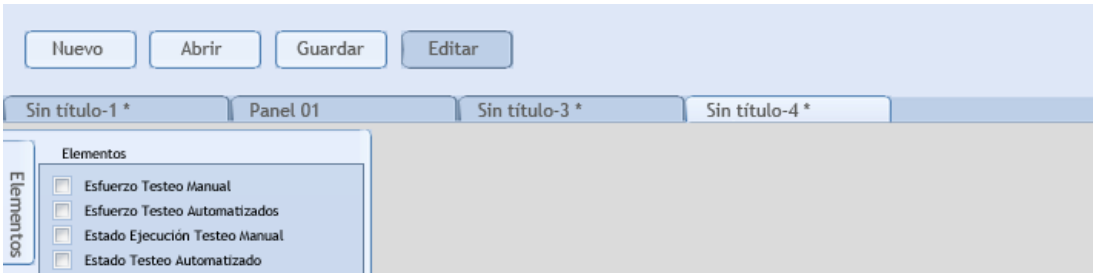


Figura 82. Vista simplificada de las pestañas del Dashboard

En la parte de la izquierda tendremos un panel que se oculta con el listado de componentes y gráficos disponibles para los Dashboards. Cada elemento del listado tendrá su correspondiente *checkbox* con el que podremos añadir o eliminar componentes o gráficos del Dashboard.

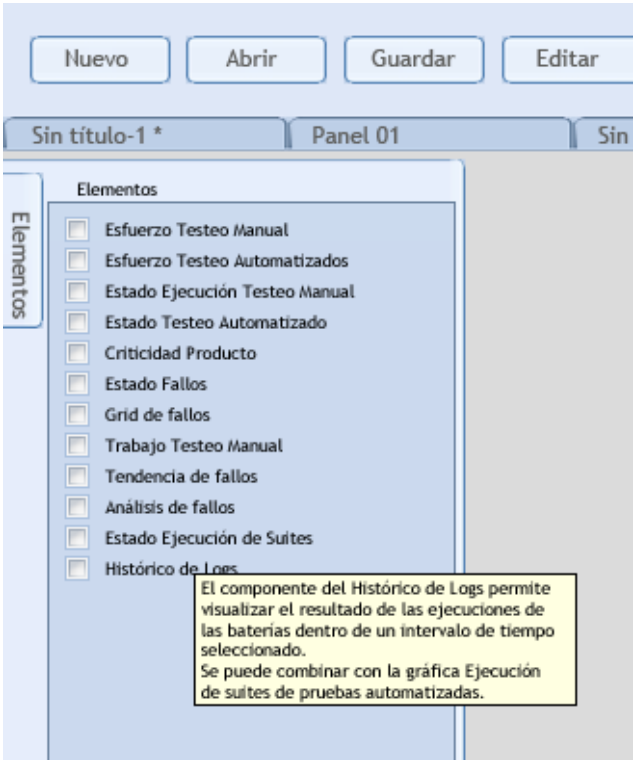


Figura 83. Panel de elementos disponibles para el Dashboard

Como se muestra en la figura, los elementos del listado se identificarán por el nombre propio del componente o gráfica y cuando el ratón se encuentre encima del elemento saldrá un *tooltip* con el texto que corresponde a la misma descripción ya definida en el apartado anterior.

Una vez tenemos el Dashboard creado podremos añadir elementos del listado y distribuirlos por la zona disponible según deseemos. La zona central visible estará dividida en siete áreas no marcadas, una en la parte superior donde estarán ubicadas las opciones generales de filtrado y otras seis áreas que corresponderán con el tamaño estándar de los gráficos (Figura 84); conforme se vayan agregando más gráficos se colocarán en la zona no visible y se habilitará un *scroll* en el Dashboard. Los únicos casos especiales son los componentes, la estructura en árbol del producto, el grid del *Gestor de fallos* y el grid que muestra el *Histórico de logs* del *Lanzador de Pruebas*. En el caso del árbol ocupará dos áreas en dirección vertical, mientras que el grid del *Gestor de fallos* y el grid del *Histórico de logs* del *Lanzador* ocuparán dos áreas en dirección horizontal; el resto de gráficos ocuparán un área.

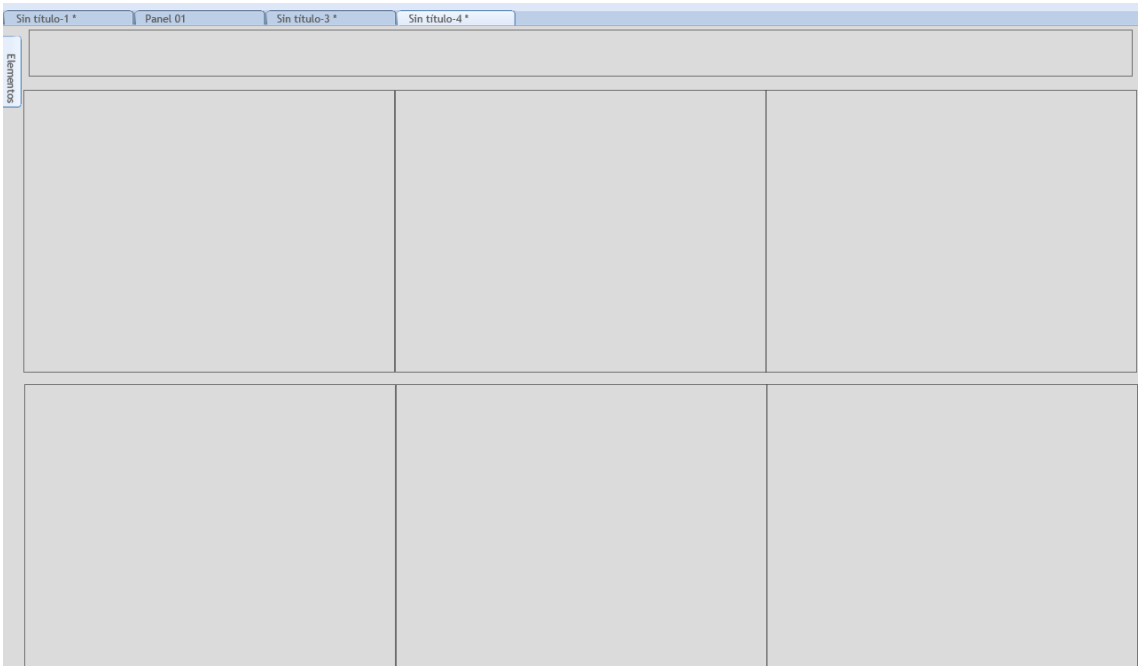


Figura 84. Parte central del Dashboard dividida en áreas

En este punto, la distribución y el contenido del cuadro de mando depende del usuario y lo que se quiera medir. El motivo de trabajar de esta forma, creando varios Dashboards, es el poder juntar gráficas que ofrezcan información sobre algo relacionado complementándose entre ellas, como puedan ser los fallos, el seguimiento de una iteración, y centrar el estudio en algo concreto.

Una vez se ha elaborado el contenido del Dashboard deberemos guardarlo para consultarlo en cualquier momento. Al apretar sobre el botón "*Guardar*" tendremos dos posibles comportamientos, el primero cuando el cuadro de mando no se haya acabado de crear, que guardará la información modificada, y el segundo, cuando tengamos un nuevo cuadro de mando que no haya sido guardado con anterioridad, en el que nos aparecerá el formulario que se muestra en la figura 85; este formulario recoge la información necesaria para el nuevo cuadro de mando, en primer lugar el nombre y la descripción y en la parte de abajo tendremos el conjunto de filtros que presentan todas las gráficas y componentes que contenga el Dashboard; la gran mayoría de gráficas y componentes presentan muchos filtros en común, por ello podremos escoger que filtros queremos poder cambiar a nivel de Dashboard en general, que serán las opciones que escojamos al guardar. En el conjunto de opciones de filtrado, únicamente aparecerán habilitados aquellos filtros que se encuentren en las graficas del Dashboard. Ese mismo formulario aparecerá cuando apretemos al botón "*Editar*".

Figura 85. Formulario de edición y guardado por primera vez

Necesitaremos un mecanismo para guardar la configuración del Dashboard junto con todos los elementos que contiene, para ello vamos a utilizar la **tecnología XML** donde tendremos definidas todas las características del Dashboard. Se ha decidido llevar a cabo esta función con XML porque nos permite expresar información de una forma estructurada, sencilla de entender, además de existir muchos procesadores de ficheros XML y, puesto que es un texto con una estructura especifica, podemos guardar el contenido en base de datos sin problemas. Así, a la tabla que diseñamos al principio se le añadirá un nuevo campo, al que llamaremos "**estructuraXML**", donde guardaremos la estructura del XML que se genere.

La generación de la estructura XML del Dashboard será cuando apretemos al botón "*Guardar*" que como se ha comentado anteriormente, el proceso que se seguirá al presionar el botón "*Guardar*" depende de si el Dashboard es nuevo o no. En el primer caso, la estructura se creará al aceptar el formulario de la figura 85, y se guardará en el campo de la tabla; en el segundo caso la estructura se creará directamente y se guardará en base de datos.

El *XML* generado tendrá la siguiente estructura; el nodo raíz tendrá el nombre "**cm**", los nodos hijos serán "**ops-filtrado**" y "**elementos**". El nodo "*ops-filtrado*" hace referencia a las opciones de filtrado que estarán a nivel de Dashboard; no tendrá atributos, en el valor del nodo tendremos las opciones seleccionadas separadas por ";". El nodo "*elementos*" tendrá tantos nodos hijos "**elemento**" como gráficas y componentes

contenga el Dashboard. El nodo "*elemento*" tendrá dos atributos, el primero será el nombre y el segundo el área donde está colocado; además, tendrá tantos nodos hijos como opciones de configuración tenga la gráfica o componente, estos nodos se identificaran por la opción de configuración y tendrán el valor seleccionado.

```
<cm>
	<ops-filtrado>{filtro1};{filtro2};{filtro3}</ops-filtrado>
	<elementos>
		<elemento nombre="{nombre-elemento}" area="1">
			<{op-conf1}>True</{op-conf1}>
			<{op-conf2}>False</{op-conf2}>
		</elemento>
		…
	</elementos>
</cm>
```

Hay que definir unos términos para las opciones de filtrados, los nombres de los elementos disponibles y las posibles opciones de configuración que tenga cada uno. En la primera tabla se muestran los términos correspondientes a los nombres de los elementos disponibles, en la segunda los términos correspondientes a las opciones de filtrado y en la última tabla los términos correspondientes a las opciones de configuración que puedan presentar.

Esfuerzo Testeo Manual	el-esfuerzo-testeo-manual
Trabajo Testeo Manual	el-trabajo-testeo-manual
Estado Pruebas de Aceptación	el-estado-pruebas-aceptacion
Esfuerzo Testeo Automatizado	el-esfuerzo-testeo-automatizado
Disponibilidad de Pruebas de Sistema	el-disponibilidad-pruebas-sistema
Estado Ejecución de Suites	el-estado-ejecucion-suites
Criticidad Producto	el-criticidad-producto
Estado Fallos	el-estado-fallos
Tendencia de fallos	el-tendencia-fallos
Análisis de fallos	el-analisis-fallos
Tendencia esfuerzo invertido	el-tendencia-esfuerzo-invertido
Tendencia Pruebas Aceptación	el-tendencia-pruebas-aceptacion
Tendencia Pruebas de Sistema	el-tendencia-pruebas-sistema

Estado Ejecución de Suites	el-estado-ejecucion-suites
Histórico de Logs	el-historico-logs

Versión	fil-version
Fecha Inicio	fil-fecha-inicio
Fecha Fin	fil-fecha-fin
Agente	fil-agente
Actividad	fil-actividad
WUs	fil-wus
Métrica	fil-metrica
Suites	fil-suites
Versión Inicio	fil-version-inicio
Versión Fin	fil-version-fin
Producto	fil-producto

Mostrar WUs	conf-mostrar-wus
Mostrar seguimiento	conf-mostrar-seg

Ahora que ya tenemos la relación de elementos, filtros y opciones en términos únicos se va a mostrar un ejemplo de Dashboard junto con la estructura XML que le corresponde.

```
<cm>
	<ops-filtrado>fil-version;fil-fecha-inicio;fil-fecha-fin</ops-filtrado>
	<elementos>
		<elemento nombre=”el-criticidad-producto” area=”1”></elemento>
		<elemento nombre=”el-grid-fallos” area=”2”></elemento>
		<elemento nombre=”el-analisis-fallos” area=”5”></elemento>
		<elemento nombre=”el-estado-fallos” area=”6”>
			<conf-mostrar-seg>False</conf-mostrar-seg>
		</elemento>
	</elementos>
</cm>
```

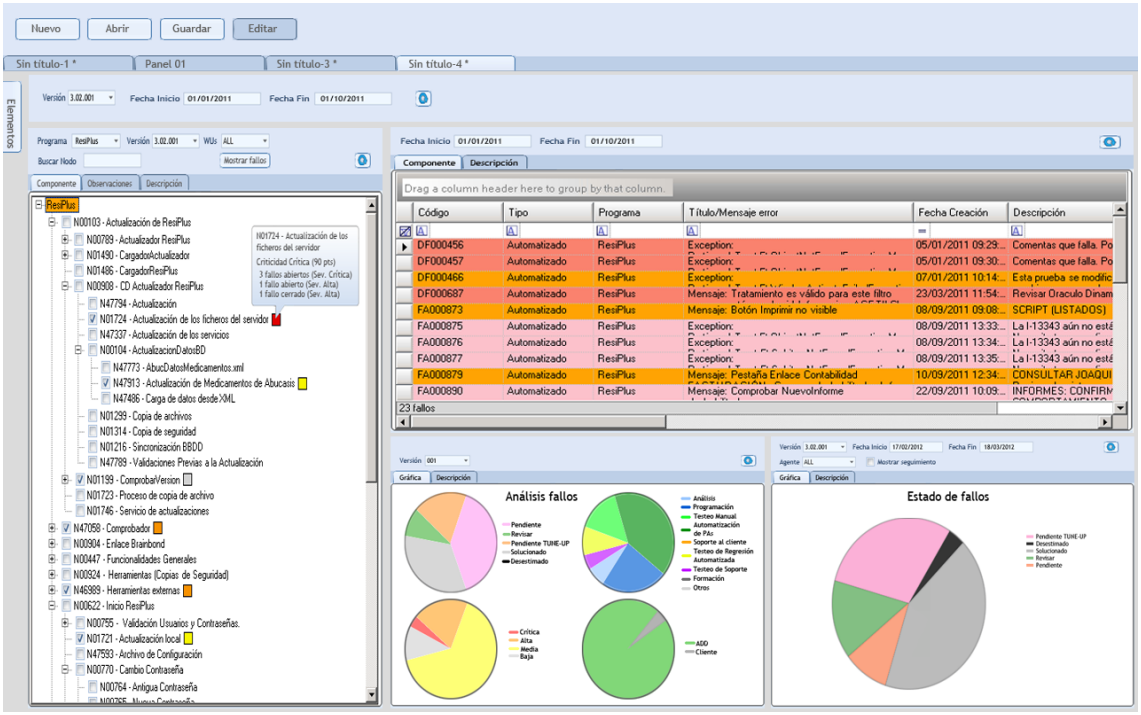


Figura 86. Ejemplo de Dashboard

Por último, podremos abrir aquellos Dashboards que hayamos guardado con anterioridad; para abrir un Dashboard deberemos apretar sobre el botón "*Abrir*" y aparecerá el formulario de la figura 87; en este formulario tendremos un grid con todos los Dashboards disponibles junto con la descripción del mismo. Seleccionando un dahsboard y aceptando tendremos nuestro Dashboard abierto.

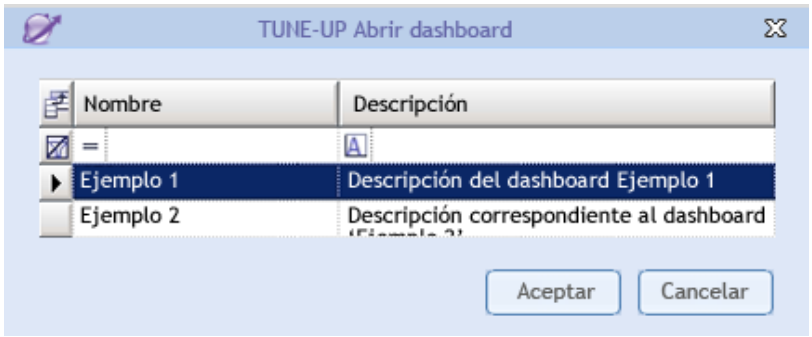


Figura 87. Formulario Abrir Dashboard

Capítulo 8. Conclusiones y trabajo futuro

Aunque aún existen desarrollos de productos sin una actividad de testeo elaborada y adecuada, en los últimos años está aumentando la buena práctica de utilizar recursos en disponer de un equipo de testeo que haga el trabajo especifico y adecuado para asegurar el buen funcionamiento de los productos y entregar calidad.

La actividad de testeo tiene un papel importante en el ciclo de desarrollo; es en el equipo de testeo donde se recopila la información de las pruebas pasadas, los fallos que se han producido; son los que tienen la respuesta a una pregunta importante que es *"¿se puede entregar el producto desarrollado?"*. En un buen proceso de testeo se necesita una buena infraestructura detrás con el que poder apoyarse a lo largo del mismo.

Esta tesina se ha desarrollado en el contexto de una colaboración con una pyme de unos 60 empleados que se encargan de desarrollar y vender un ERP para el sector socio-sanitario. Actualmente la empresa usa la metodología TUNE-UP para la gestión de más de 15000 pruebas de aceptación.

En un equipo de desarrollo suelen haber tres subgrupos diferenciados, por un lado están los *Analistas*, los *Programadores* y, por último, los *Testers*, además de existir un jefe de proyecto, encargado de coordinar en todo momento los departamentos para finalizar con éxito la iteración de desarrollo. Durante la realización de la tesis he colaborado con el equipo de testeo, siendo el responsable del testeo automatizado con el rol de *Automation Tester*.

Un motivo por el que surgió la necesidad de un Dashboard fue por parte del jefe de proyecto, que necesitaba poder tener datos que se generaban en el proceso de testeo ordenados de tal forma que mostrase información clara y útil y le permitiese supervisar el estado actual del proceso de testeo.

Así pues, el trabajo final se ha concentrado en realizar un diseño de métricas necesarias y de un entorno donde poder elaborar Dashboards a partir de las métricas diseñadas. Para llevar a cabo este cometido, primero se estudiaron las soluciones comerciales que ofreciesen soporte al proceso de testeo; se hizo un estudio de las herramientas SilkCentral Test Manager de Borland, Rational Quality Manager de IBM y Visual Studio Team Foundation Server de Microsoft, sin embargo, se descartó implantar

alguna de las soluciones puesto que el entorno de testeo manual y automatizado se encontraba en un punto muy avanzado.

El estudio del estado del arte nos sirvió, por un lado, para concluir que no existe un Dashboard estándar definido y que la propia infraestructura es el cuello de botella a la hora de elaborar un Dashboard; y por otro lado, identificar aquellas métricas que nos parecieron interesantes. Una vez tuvimos claro qué información queríamos obtener de las métricas, identificamos qué infraestructura necesitábamos para poder dar soporte a las mismas.

Este proceso ha finalizado con la optimización del Lanzador de Pruebas y el desarrollo íntegro del *Diseñador de PS* y el *Gestor de Fallos*. El Diseñador de PS también se utiliza como base para las pruebas manuales y se integró con TUNE-UP para optimizar la creación de pruebas de sistema y conseguir unos diseños más intuitivos para el *Automation Tester*; el *Gestor de Fallos* nos permite una absoluta gestión de los fallos hasta conseguir solucionarlos, al igual que con el *Diseñador de PS*, el *Gestor de Fallos* se ha integrado en TUNE-UP, esperando a obtener resultados de su implantación en el proceso.

Una vez tuvimos la infraestructura necesaria, se inició el trabajo de diseño de las métricas y del entorno donde podremos elaborar los Dashboards deseados a partir de las métricas.

Tras la realización de esta tesina, el trabajo futuro que supone este trabajo es la implementación e integración de la propuesta en la herramienta TUNE-UP. Una vez tenemos el Dashboard implementado y activo, deberemos seguir un proceso de análisis de utilidad de las métricas diseñadas, buscando soluciones óptimas, si es necesario, para poder tener un control y conocer el estado del producto que se está desarrollando. Además, una vez se haya rodado durante un tiempo sería interesante integrar indicadores estableciendo umbrales que nos muestren el estado de los objetivos definidos en el proceso.

Capítulo 9. Referencias

1. M. Company. Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación (2011).
2. Marante, M. Company, M. Letelier, P. Suarez, F. Gestión de requisitos basada en pruebas de aceptación: Test-Driven en su máxima expresión. XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010).
3. Company, M., Letelier, P., Marante, M., Suarez, F. Análisis de impacto en requisitos soportado de forma semi-automática y en un marco de desarrollo TDD basado en pruebas de aceptación. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)
4. Marante, M., Company, M., Letelier, P. Seguimiento ágil de proyectos de desarrollo de software utilizando Gráficas Burn Down. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)
5. Marante, M., Letelier, P., Suarez, F. *TUNE-UP: Seguimiento de proyectos software dirigido por la gestión de tiempos.* XIV Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2009). ISBN 978-84-692-4211-7 (pág. 57-68), Septiembre 2009.
6. Marante, M., Letelier, P., Suarez, F. *TUNE-UP: Un enfoque pragmático para la planificación y seguimiento de proyectos de desarrollo y mantenimiento de software*. I Congreso Iberoamericano SOCOTE – SOporte al COnocimiento con la TEcnología (SOCOTE 2009). ISBN 978-84-613-8585-0, Noviembre 2009.
7. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley. 2000.
8. Haugset, B., Hanssen, G., Automated Acceptance Testing: A Literature Review and an Industrial Case Study, Proc. of Agile 2008, pp. 27-38
9. IEEE 830-1993, IEEE Recommended Practice for Software Requirements Specifications
10. Kroll P., Kruchten P., Booch G. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley Professional. 2003.
11. Schwaber K., Beedle *M. Agile Software Development with Scrum.* Prentice Hall, Series in Agile Software Development. 2001

12. Humphrey, Watts S. The Personal Software Process. Software Process Newsletter, Technical Council on Software Engineering, IEEE Computer Society, Volume 13, No. 1, Sept. 1994, pp SPN 1-3.
13. Allen, D. Getting Things Done: The Art of Stress-Free Productivity. Published in Penguin Books 2003. ISBN 0 14 20.0028 0
14. Covey, S. The Seven Habits of Highly Effective People. Published Simon & Schuster, Limited. Edition Softcover. ISBN 0743269519
15. Crispin, L., Gregory, J. Agile Testing. A Practical Guide for Testers and Agile Teams. ISBN 0321534468
16. Riley, T., Goucher, A. Beautiful Testing: Leading Professionals Reveal How They Improve Software. ISBN 0596159811
17. Whittaker, J.A. How To Break Software: A Practical Guide to Testing. ISBN 0201796198
18. Page, A., Johnston, K., Rollison, B. How We Test Software at Microsoft. ISBN 0735624259
19. ISO/IEC 15939, Systems and software – Measurement process
20. Ebert, C., Reiner, D., Bundschuh, M., Schmietendorf, A. Best Practices in Software Measurement. ISBN 3540208674
21. MSDN Library. Microsoft Dashboards. *http://msdn.microsoft.com/en-us/library/dd420548*
22. IBM Rational Quality Manager. *http://www-01.ibm.com/software/rational/products/rqm/*
23. SilkCentral Test Manager. *http://www.borland.com/products/SilkCentral/*
24. Certified Tester. Foundation Level Syllabus. *http://www.istqb.org/downloads/finish/16/15.html*
25. Certified Tester. Advance Level Syllabus. *http://www.istqb.org/downloads/finish/3/2.html*
26. Glosario Estándar de Términos Utilizados en Pruebas de Software. Versión 1.3 *http://www.sstqb.es/ficheros/sstqb_file94-5205ea.pdf*