Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Definición e implantación de un proceso QA para desarrollo de software

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Fco Javier Sanchis Milla

Tutor: Patricio Letelier Torres

2015/2016

Resumen

Con la definición e implantación de un proceso Quality Assurance en una empresa, en este nuestro caso, de desarrollo de software, evita en gran medida que el cliente detecte fallos y pierda la confianza en nuestro producto. Siempre sale más rentable económicamente y en cuestiones temporales, detectar defectos en fases tempranas de desarrollo y corregirlos. En esta memoria o TFG, definimos e implantamos un proceso de QA dentro de un entorno real, en una empresa en la cual he colaborado como becario durante este año lectivo, centrándonos en las pruebas de aceptación como núcleo de todo el proceso.

Palabras clave: calidad del software, aseguramiento calidad, metodología ágil.

Abstract

The definition and implementation of a QA process in a company, in this our case, software development , largely avoids the client detects failures and lose confidence in our product. Always comes more profitable economically and in temporal matters , defects in early stages of development and fix them. In this report , or TFG , define and implement a process of QA in a real environment , in a company where I worked as an intern during this school year , focusing on the acceptance tests as the core of the whole process.

Keywords : software quality, quality assurance, agile methodology.

Tabla de contenidos

Tabla de figuras

1. Introducción

1.1 Motivación y punto de partida

La motivación para llevar a cabo este trabajo es definir y depurar un proceso de aseguramiento de calidad o "Quality Assurance" (QA) robusto que permita una vez implantado en una empresa de desarrollo de software, asegurar la calidad final del producto. Importante destacar que este proceso no solo es beneficioso para el usuario final que recibe dicho producto, sino que también lo es para la empresa, ya que se establece una metodología de trabajo en la cual existe un control permanente sobre el proceso, evitando los altos costes que pueden suponer corregir fallos en etapas avanzadas de un proyecto concreto.

Actualmente muchas empresas tienden a aplicar estos procesos para llevar a cabo el desarrollo de sus productos software aunque bien es cierto que para que el proceso sea un éxito es necesaria la colaboración de cada uno de los departamentos de la organización y es por esta razón que es necesaria una definición rigurosa del proceso QA que se va a llevar a cabo.

Este proyecto se ha desarrollado con la colaboración de una empresa dedicada el desarrollo de un ERP en continuo mantenimiento donde se utiliza una metodología de Desarrollo Dirigido por Pruebas de Aceptación donde cada una de las entregas al cliente están definidas por una serie de unidades de trabajo que contienen un conjunto de Pruebas de Aceptación (PA) que deben ser satisfechas en su totalidad en cada iteración del producto. Cada unidad de trabajo, sigue un flujo de trabajo, o Workflow (WF) donde intervienen distintos roles para realizar las distintas actividades que conforman dicho WF.

Estos roles y actividades, así como la gestión de las PA que conforman cada unidad de trabajo, hacen necesario definir de una manera exhaustiva un proceso que asegure la calidad del producto entregado al cliente.

Por tanto, el punto de partida no es de cero ya que en dicha empresa ya se sigue una metodología donde tiene gran peso el aseguramiento de calidad.

1.2 Objetivos

Como hemos comentado en el anterior apartado, el punto del que partimos no es de cero ya que existen mecanismos con los cuales se controla el aseguramiento de la calidad del producto software desarrollado.

El objetivo de este trabajo es el siguiente:

- Definir el proceso QA como tal de una manera formal.

- Depurar el proceso y añadir mejoras que nos permitan, valga la redundancia, mejorar los resultados. Para ello a lo largo de las prácticas realizadas en la empresa colaboradora se ha trabajado en mejorar el proceso que ya existía, revisando cada una de las actividades que conforman el proceso QA.

- Documentar el uso las distintas herramientas sobre las cuales nos hemos apoyado para llevar a cabo los objetivos anteriores:

    - SAPI, para gestionar el desarrollo del producto.

    - Un Diseñador de Pruebas de Aceptación.

    - Rational Functional Tester (RFT) para la automatización de las Pruebas de Aceptación diseñadas.

    - La herramienta ATUN para gestionar las pruebas automatizadas, tanto para su lanzamiento como para su posterior clasificación según su resultado.

    - Un Gestor de Fallos que nos permitirá clasificar los distintos fallos (confirmados o no) que vengan por parte de los Testers Manuales o bien del resultado de las pruebas automatizadas.

Cada uno de estas herramientas nos permite obtener datos del proceso por separado para llegar a conclusiones globales sobre la eficacia del proceso de aseguramiento de calidad que estamos llevando a cabo. Esto es de vital importancia pues algo que no se puede medir, no se puede controlar.

1.3 Estructura del trabajo

La organización de la memoria va a ser la siguiente:

- **Conceptos previos**: En este capítulo se introducen algunos conceptos relacionados con la Calidad del Software. También se explica el proceso de desarrollo en el caso concreto de la empresa con la que se ha colaborado.

- **Proceso QA**: Aquí se pasa a detallar de una manera extensa la definición e implantación del proceso QA así como las herramientas con las que hemos apoyado dicho proceso, incluyendo la gestión de fallos.

- **Métricas del proceso QA**: Lo que no se puede medir, no se puede controlar y es por ello que en este apartado mostramos las métricas que han sido utilizadas para medir el proceso QA y cuantificar su impacto en el desarrollo de software.

- **Conclusiones**: Valoración final del trabajo realizado y otras cuestiones relacionadas con éste.

- **Referencias:** Fuentes de información en las que hemos apoyado esta memoria.

2. Conceptos previos

En primer lugar, antes de comenzar a explicar el proceso QA llevado a cabo para el caso concreto que nos ocupará esta memoria vamos a definir algunos conceptos previos así como el proceso de desarrollo el cual ira de la mano.

2.1 Calidad Software

Podemos encontrar numerosas definiciones sobre que es la calidad del software. Vamos a ver dos de ellas:

> *"Es la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente".* (Pressman, 1998)

Pressman (1998) considera que el aseguramiento de la calidad del software comprende una gran variedad de tareas asociadas:

- Preparar un plan para asegurar la calidad del producto software que compone un proyecto.
- Participar activamente en el desarrollo del proceso de descripción del proyecto.
- Revisar continuamente las actividades de ingeniería del software para verificar su consistencia con el proceso definido.
- Auditar el producto software para verificar el cumplimiento del proceso definido.
- Asegurar que las divergencias se documentan de acuerdo a los estándares definidos.
- Almacenar cualquier inconformidad y reportarla a la gerencia media.

"La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas." (ISO/IEC 25010)

El modelo de calidad del producto software definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características que se muestran en la Figura 1.

Figura 1: Características modelo ISO/IEC 25010

Como vemos en estos dos ejemplos de definición de la calidad del software, hay muchos aspectos involucrados como la ausencia de defectos, usabilidad, seguridad, fiabilidad y reunión de especificaciones. Sin embargo, hay algo importante que se debe tener presente: la calidad del software debe estar integrada en el proceso de desarrollo, no es algo que puede ser añadido después.

A partir de un modelo de calidad se establece el sistema para evaluar la calidad de un producto determinando que características se tienen que cumplir y se van a tener en cuenta para evaluar sus propiedades. Por tanto, para controlar la calidad del software es necesario definir parámetros, indicadores, u otros criterios de medición.

Producir con éxito software y realizarlo con una demostrada calidad, solo es posible con la implantación de un proceso de aseguramiento de calidad (QA) y con el apoyo de herramientas, tanto manuales como automatizadas, para aplicar las técnicas y procedimientos necesarios para ello. **Para que el producto final sea de calidad, el proceso por medio del cual éste es elaborado debe ser también de calidad.**

2.2 Proceso de desarrollo

Para definir el proceso QA es necesario que lo contextualicemos dentro del proceso de desarrollo del producto software. En el caso que nos atañe, la Figura 2 ilustra cómo un producto software evoluciona desde una versión a otra. Para ello, definimos una Incidencia (ID) como un cambio en el producto, que puede ser: un nuevo requisito, una mejora de un requisito ya implementado, o corrección de un fallo detectado y que ha sido introducido en versiones anteriores del producto.

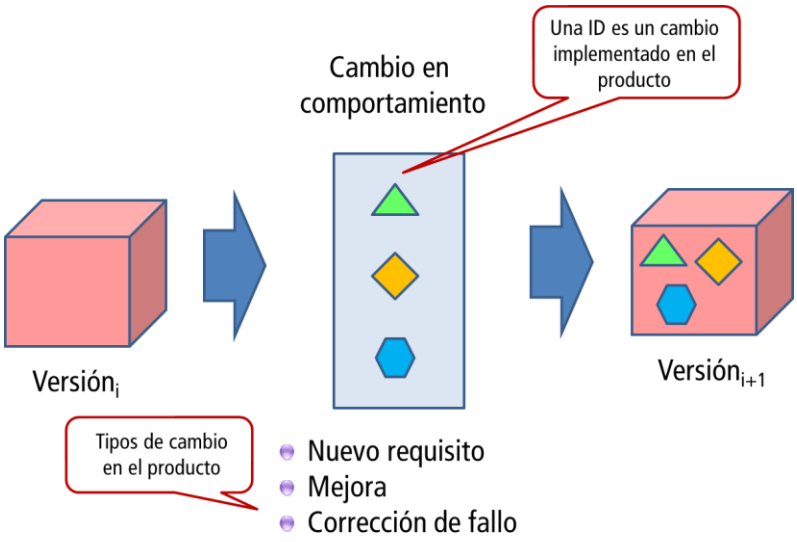

Figura 2: Evolución producto software [1]

La empresa en la cual se ha colaborado a lo largo de la elaboración de esta memoria hace uso de una metodología ágil para el desarrollo de sus productos software, es por ello que vamos a apoyarnos en esta. Siguiendo esta metodología, el tiempo empleado para generar una nueva versión del producto principal es de alrededor de un mes. A cada uno de estos períodos de trabajo los denominaremos Sprint.

Así pues, un Sprint contiene el conjunto de IDs correspondientes a los cambios que incluirá la nueva versión del producto. La relación entre Sprint, Versiones e ID se ilustra en la Figura 3.

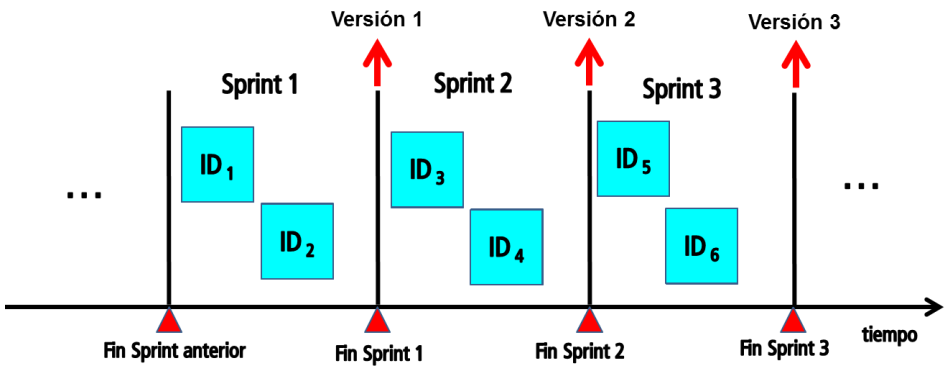

Figura 3: Relación Sprint-Versiones-ID [1]

2.3.1 Estructura del producto y del equipo de trabajo

Cuando se crea una ID se identifican las partes del producto que van a ser afectadas por el cambio. Esta tarea de análisis de impacto, la realizan los analistas. Teniendo esto en cuenta es evidente por qué la estructuración del producto es fundamental para mantener la trazabilidad entre el comportamiento actual del producto y las distintas ID que han ido incrementando (en cuanto a funcionalidad se refiere) dicho comportamiento. Para representar la estructura del producto se utiliza un grafo acíclico dirigido en el cual los nodos corresponden a requisitos y sub-requisitos.

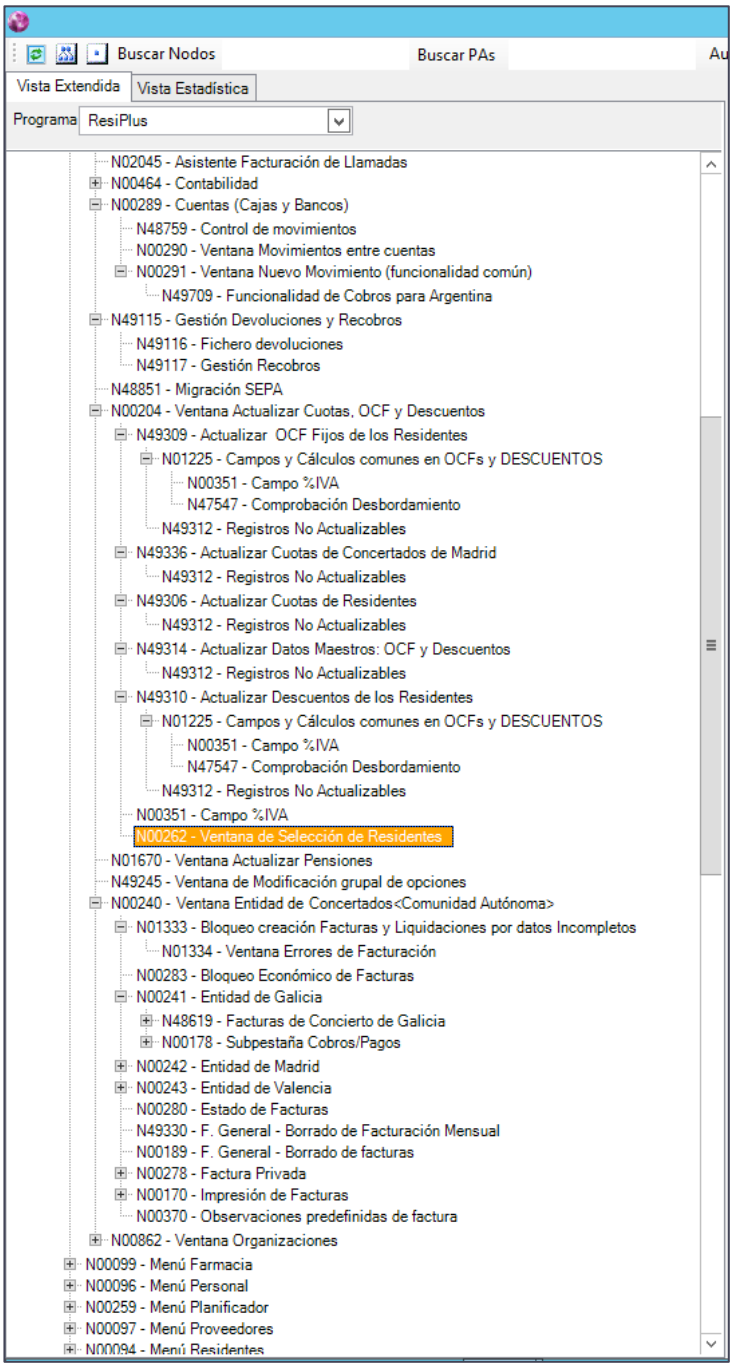

Figura 4: Estructura del producto

Dentro de cada uno de los nodos encontramos los requisitos propiamente dichos a los que en nuestra metodología llamamos Pruebas de Aceptación (PA)

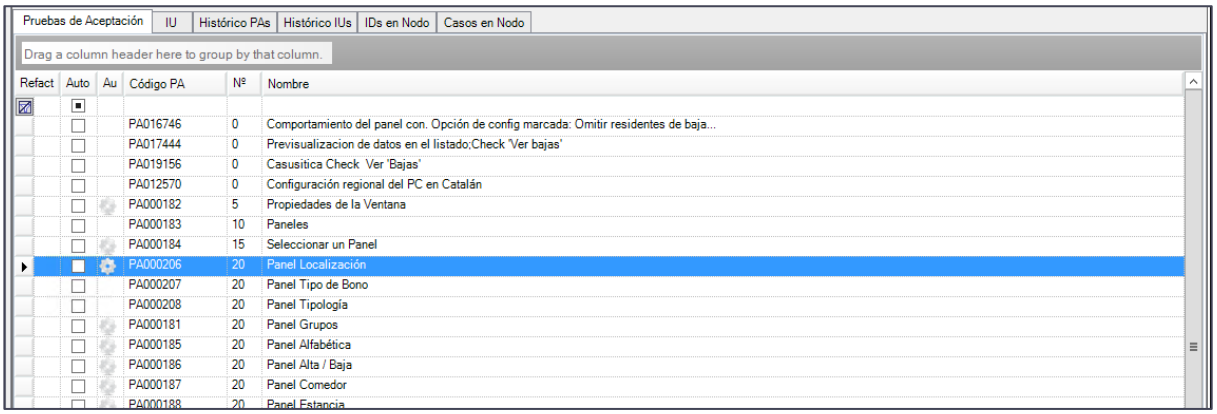

Figura 5: Pruebas de Aceptación

Por otra parte, el equipo de desarrollo está organizado en tres equipos, cada uno de ellos enfocado a la implementación de ID en diferentes áreas funcionales del producto: Equipo Económico, Equipo Asistencial y Equipo Misceláneo. La Figura 6 muestra un esquema de lo que sería la estructura del producto y cómo es mantenida por los equipos de desarrollo.

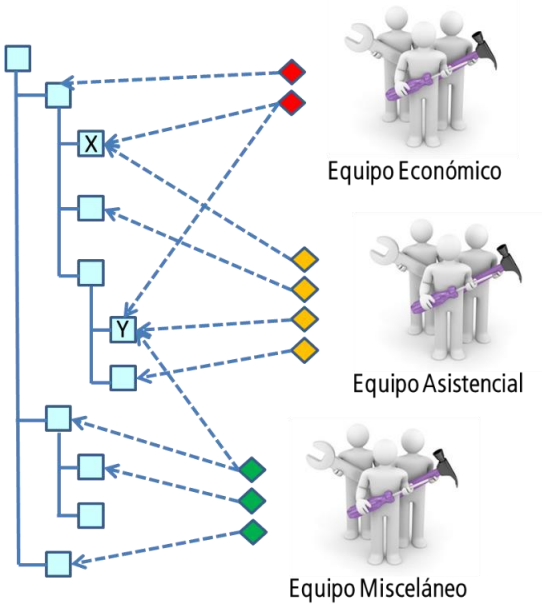

Figura 6: Equipos de desarrollo

Para gestionar el conjunto de cambios propuestos para un producto necesitamos organizarlos en un Backlog, entendiendo como tal a una lista de todas esas propuestas tendrán que ser tomadas en cuenta, o no, para incluir en el siguiente Sprint. La adecuada priorización de ID es clave para la satisfacción de los clientes y los criterios utilizados para priorizar son múltiples y deben considerarse en conjunto.

El criterio protagonista es el valor que aporta el cambio para los usuarios, bien sea por la corrección de un error especialmente importante o la adición de una funcionalidad atractiva para el cliente. Sin embargo, también debe considerarse la dificultad o esfuerzo que conlleva el cambio, los tipos de usuarios a los cuales afectará, si el cambio pertenece o no a un conjunto de cambios que tiene ciertos plazos de entrega, qué partes del producto se pueden ver afectadas por la ID, las relaciones con otras ID, etc.

Así pues, de acuerdo con los nodos del producto que se vean afectados por una ID, será un equipo u otro el que se encargue de su implementación tal y como representa la Figura 7:

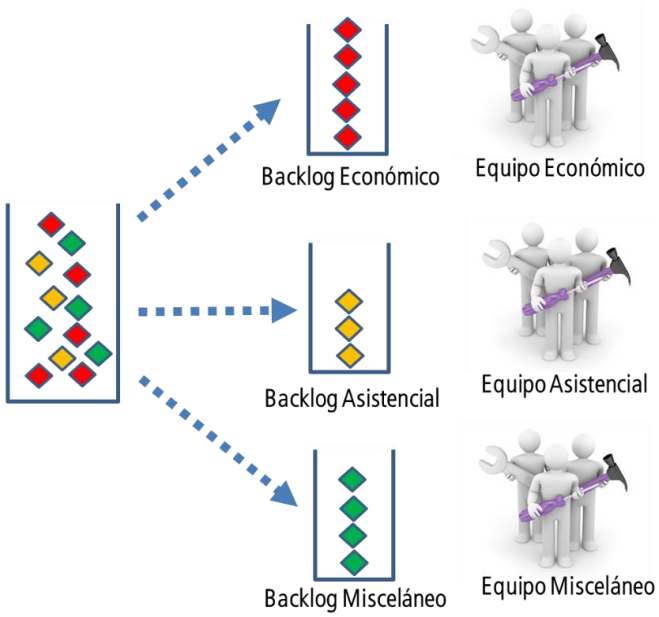

Figura 7: Backlogs específicos

Conforme los equipos de desarrollo van implementando ID para el siguiente Sprint, el Backlog General se va consumiendo. La siguiente figura complementa a la figura mostrada anteriormente, ahora mostrando también cómo se van implementando las ID del Backlog.

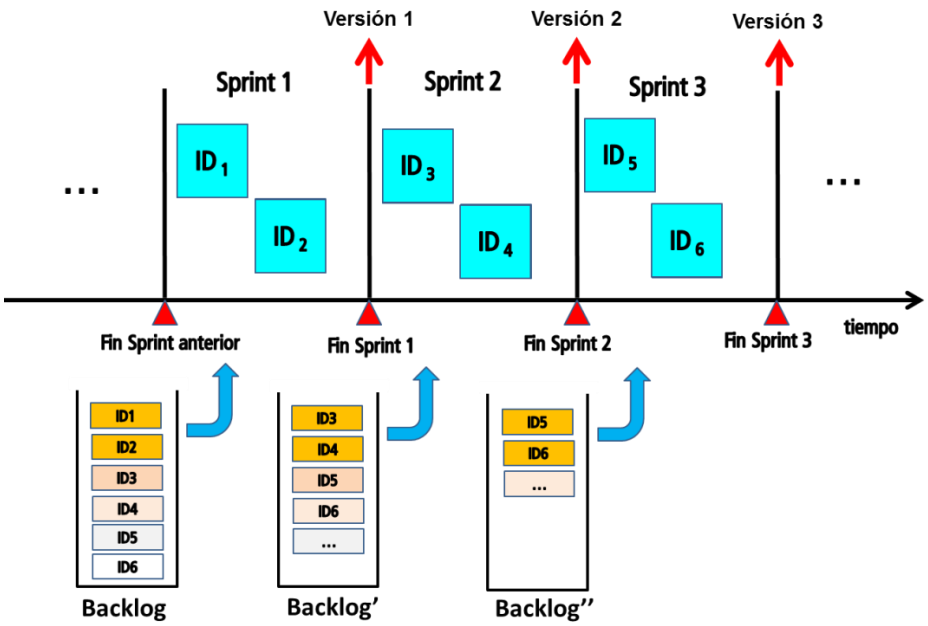

Figura 8: Evolución del Backlog y del producto software [1]

2.3.2 Workflow de desarrollo

Desde que se crea una ID hasta que se finaliza esta pasa por distintas etapas o actividades. Estas actividades son realizadas por determinados roles dentro del equipo de desarrollo y viene preestablecidas en un flujo de trabajo o Workflow (WF).

El WF de una ID constituye el camino "normal" que seguirá una ID, sin descartar que se produzcan saltos hacia adelante o hacia atrás entre las actividades. Es común por ejemplo, que entre las actividades de Aplicar Pruebas Manuales y Diseño e Implementación suelan existir vueltas atrás cuando el Tester detecta fallos en la implementación de la ID.

Los roles dentro del equipo de desarrollo pueden ser los siguientes:

- **Product Manager**. El encargado de establecer las prioridades y el contenido de cada Sprint.
- **Analista**. Quien especifica en detalle el cambio que conlleva una ID.
- **Programador.** Encargado del diseño e implementación del cambio en el código del programa.
- **Tester.** Encargado de comprobar que el cambio definido en la ID se ha implementado correctamente.

- **Traductor**. El producto software en este caso se traduce simultáneamente a varios idiomas, en cada uno de ellos existen traductores que se encargan de traducir las cadenas de texto asociadas (cuando la ID incluye cambios en cadenas de texto en la interfaz o en informes generados por la aplicación).

Las personas participantes en los equipos pueden desempeñar más de un rol, aunque la mayoría son especialistas en un solo rol.

En la Figura 9 se muestra el WF de desarrollo al completo aunque a lo largo de este trabajo nos vamos a centrar en las actividades Analizar Incidéncia, Diseño e Implementación y Aplicación de Pruebas Manuales.

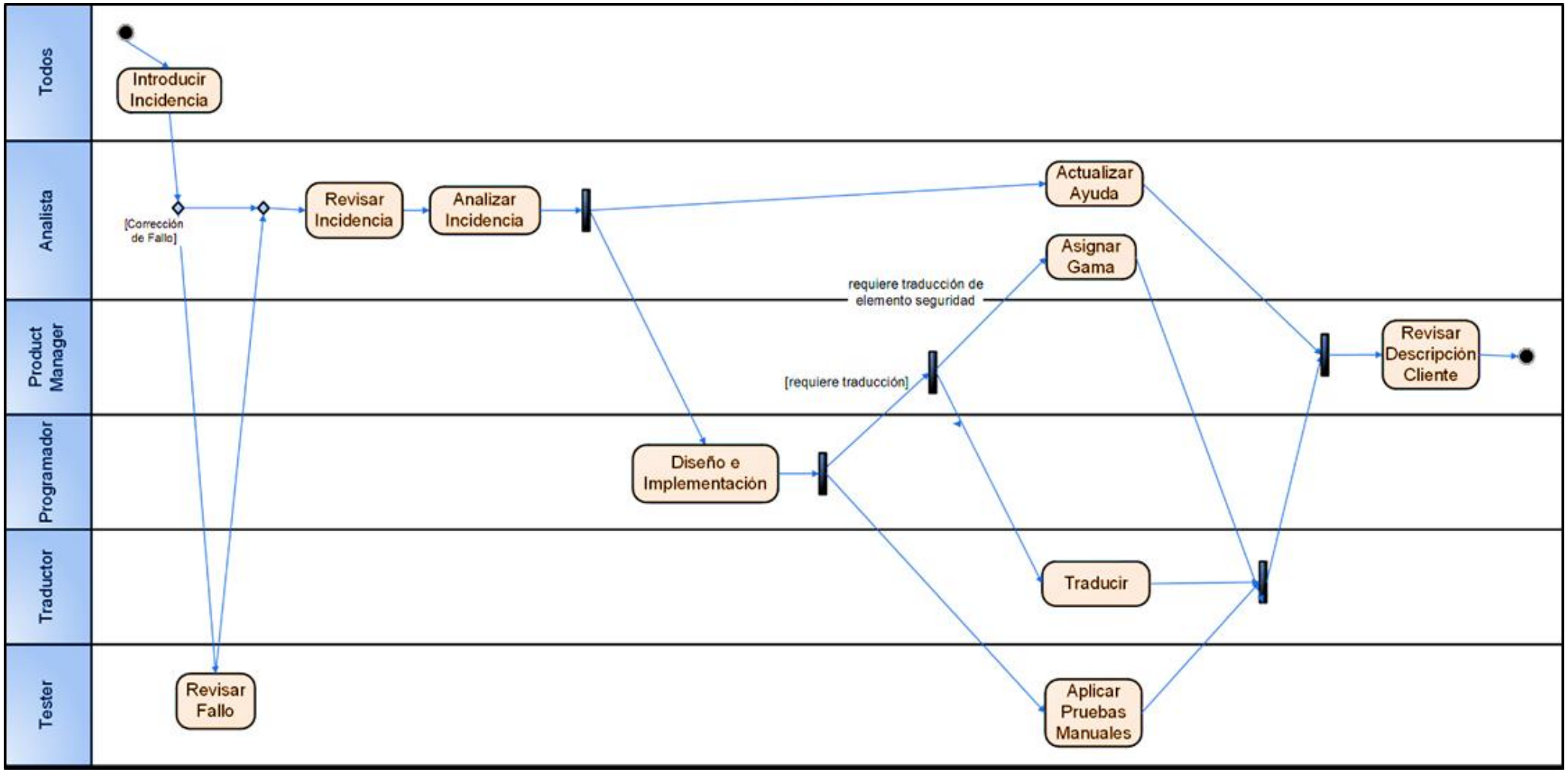

Figura 9: Workflow de Desarrollo

2.3.3 SAPI – Sistema de Apoyo al Proceso de ID

Para coordinar el trabajo colaborativo y llevar a cabo el desarrollo de las distintas ID siguiendo un WF asociado a ellas, surgió la necesidad de hacer uso de una herramienta desarrollada internamente. SAPI, cuyas siglas hacen referencia a Sistema de Apoyo al Proceso de Incidencias, nos permite y facilita la gestión y mantenimiento del producto, tanto el Backlog, Sprint como las Incidencias a nivel individual. SAPI también permite la comunicación entre los distintos usuarios o agentes y controla la participación de cada uno de ellos para cada una de las actividades del WF.

Esta herramienta esta en continuo mantenimiento ajustándose a las peticiones del equipo de desarrollo, bien sea por parte de los Analistas, Desarrolladores o Testers. Estas peticiones la mayor parte de las veces corresponden con mejoras para tener un mayor control del proceso de desarrollo que implica en mejorar el proceso de calidad.

De manera resumida y como introducción a la herramienta para posteriormente desarrollar el proceso QA, comentar qué SAPI dispone de X módulos principales que nos establecen la base para desarrollar el proceso QA de manera satisfactoria. Como se explicará más adelante el SAPI ofrece apoyo para el proceso de desarrollo centrado en las Pruebas de Aceptación, alrededor de las cuales girará todo el proceso de desarrollo de los distintos productos.

El primero de los módulos, el Planificador Personal o PP, lo encontramos nada más iniciar la aplicación. En el panel de la izquierda, a modo de tablero Kanban, se muestran las contabilizaciones de ID en cada actividad. En el panel de la derecha se muestran detalles de las ID contenidas en alguna actividad (o en todas). Mediante filtros se puede restringir la información presentada. Se ofrecen diversos filtros: producto, versión, participantes involucrados, proyecto, tipo de ID, etc.

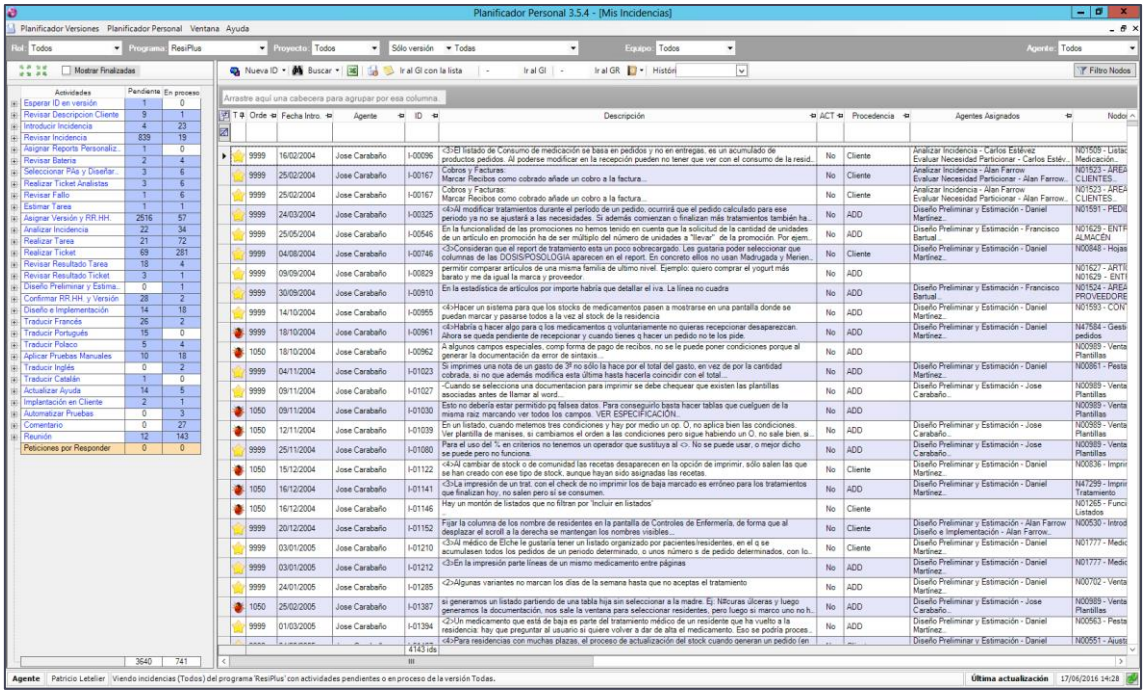

Figura 10: Pantalla principal de SAPI

Otro de los módulos importantes es el que se corresponde con la composición de los Sprint con las distintas ID de nuestro Backlog.

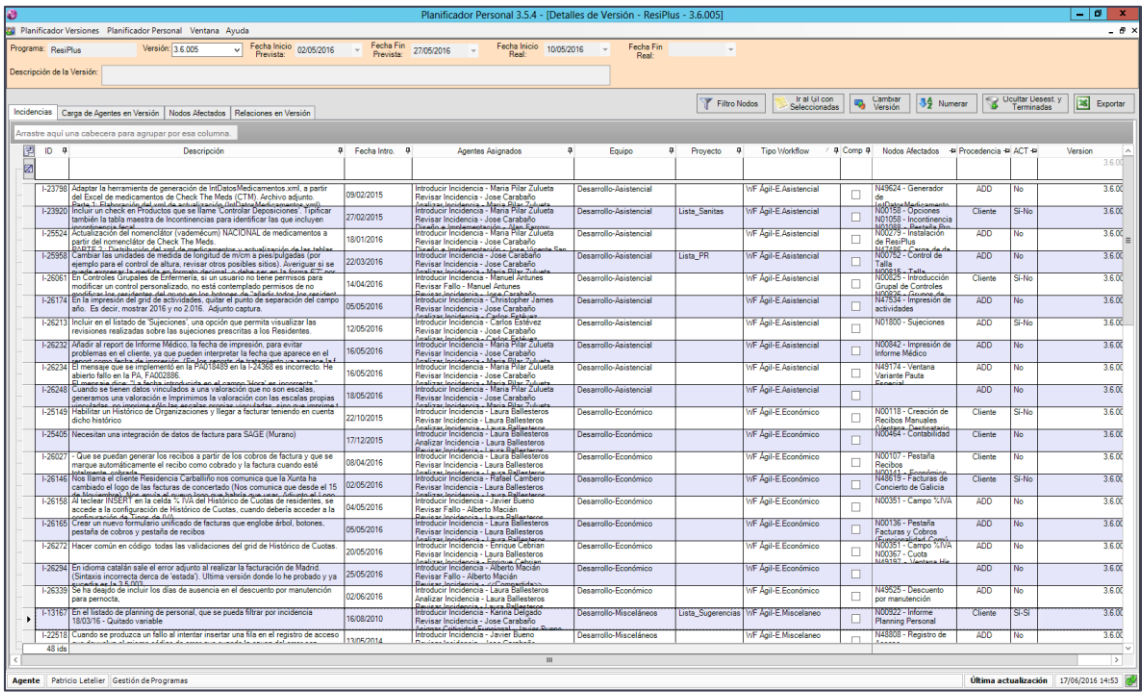

Figura 11: Creación de un Spint en SAPI

Desde esta pantalla podemos mover las ID que consideremos oportunas, siguiendo los criterios anteriormente comentados, al Sprint actual el cual conformará una nueva versión del producto. Esta función la lleva a cabo el jefe de desarrollo o Product Manager en consenso con el equipo de desarrollo mediante reuniones, que dicho sea de paso también pueden ser convocadas por SAPI.

Por último, y sin duda hablamos del módulo más importante de SAPI, tenemos el Gestor de Requisitos, desde donde podemos marcar los nodos o partes a los que afecta una ID en el producto así como crear nodos nuevos, modificar y eliminarlos. Además, dentro de cada nodo tenemos un conjunto de Pruebas de Aceptación que se corresponden con requisitos o funcionalidades del producto, las cuales marcamos también en cada ID para indicar que dicha PA se va a ver afectada, y pasa a ser una prueba de regresión, modificarlas o incluso marcarlas para eliminación.

Este módulo es tan fundamental que podríamos decir que es la base de todo el proceso QA ya que a partir de las Pruebas de Aceptación gira todo el desarrollo. Hay que ser especialmente cuidadoso con el marcado de dichas PA así como con el marcado de a que nodos o partes del programa afecta una ID. Esta función la realizan los Analistas que son grandes conocedores del producto y de las distintas funcionalidades y partes de las que está compuesto.

Para que nos hagamos una idea de la magnitud y de la complejidad de este árbol, en el producto principal, estamos hablando de un total de más de 15000 Pruebas de Aceptación repartidas en 1500 nodos.

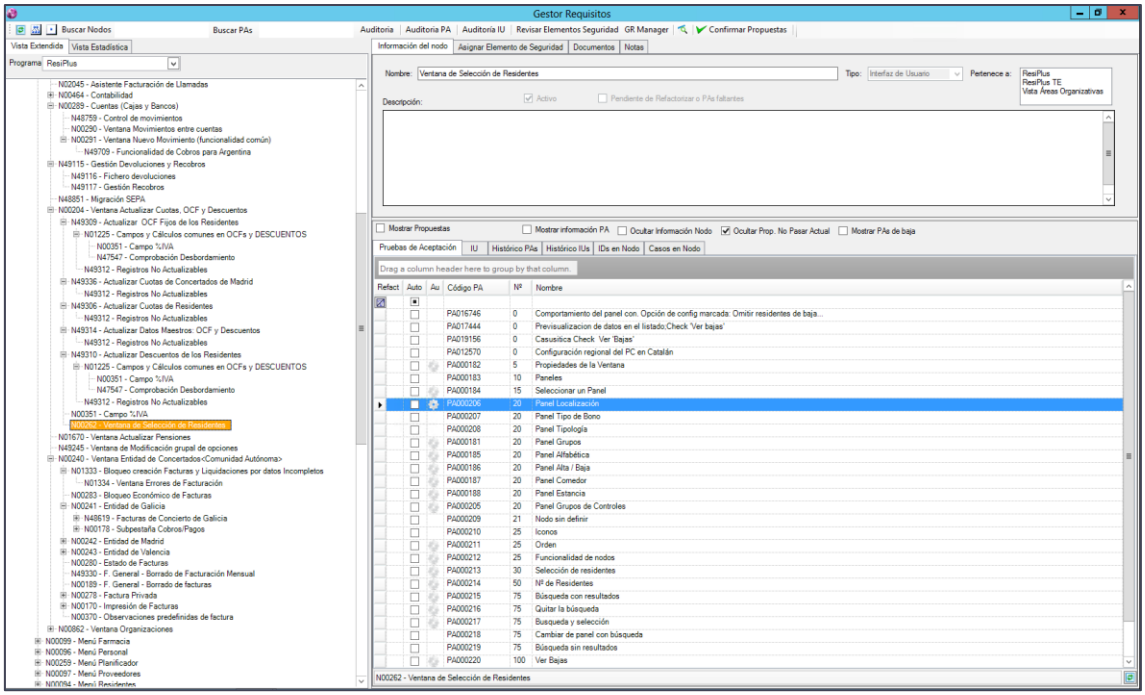

Figura 12: Gestor de Requisitos en SAPI

En posteriores apartados, explicaremos más detalladamente el proceso de selección de Pruebas de Aceptación así como su vital importancia para el desarrollo del producto software.

3. Proceso de QA

Como hemos dicho en más de una ocasión anteriormente, el proceso de desarrollo y mantenimiento de los productos software están dirigidos en este caso en concreto por las Pruebas de Aceptación. Evidentemente no hay una sola manera de gestionar y de definir un proceso de QA y es por eso que nos vamos a centrar en este caso concreto, en el caso llevado a cabo en la empresa/producto con la que el autor de esta memoria ha estado colaborando durante el último año lectivo.

Gran parte del contenido de este capítulo ha sido resumido desde [1].

3.1 Conceptos de QA

Antes de entrar con el proceso de QA propiamente dicho, vamos a definir unos conceptos de QA que nos contextualizarán dicho proceso.

- **Fallo:** comportamiento no deseado del producto, visible desde el punto de vista externo del producto.
- **Defecto:** la imperfección que causa el fallo. Un defecto puede existir en el producto y no manifestarse aún como fallo.
- **Error:** lo que causó la introducción del defecto en el producto. El error siempre es cometido directamente o indirectamente por una acción (o no acción) humana.

Diremos que un Error introduce algún Defecto, que en algún momento puede manifestarse como un Fallo. Tanto los defectos como los fallos deben ser "detectados". Los desarrolladores deben evitar cometer errores. Cuando se detecta un fallo en el producto, los desarrolladores deben detectar el defecto que los produce.

Por tanto, y teniendo en cuenta lo anterior, los objetivos del proceso QA son:

- **Prevenir errores**: estableciendo mecanismos de defensa que eviten la introducción de defectos. Por ejemplo, checklists, guías, actividades de validación y verificación, etc.
- **Detectar fallos en el producto**: disponer de mecanismos que nos garanticen una alta probabilidad de detectar fallos antes de poner una versión en producción.

- **Gestionar fallos**: asegurar que los fallos sean abordados oportunamente según su severidad y minimizando el efecto negativo en los usuarios. Estudiar los fallos para establecer medidas preventivas.

3.2 Desarrollo basado en Pruebas de Aceptación

En este caso concreto, la empresa colaboradora para desarrollar sus productos software, aplica un enfoque desarrollado por el grupo ISSI de la UPV. En este enfoque, llamado Test-Driven Requeriments Engineering (TDRE) los requisitos son especificados esencialmente mediante Pruebas de Aceptación.

Estas Pruebas de Aceptación que definimos apoyándonos en herramientas como SAPI, nos permiten organizar el trabajo en base a nuestra metodología de desarrollo y además nos permite dotar a los proyectos de una estructura basada en un grafol donde las funcionalidades están representadas como nodos y dentro de estas encontramos las distintas PA que especifican su correcto/deseado funcionamiento.

Una PA tiene como propósito demostrar al cliente el cumplimiento de un requisito del software. Precisando más, una PA:

- Describe un escenario, compuesto por una situación del sistema (condición de ejecución) una secuencia de pasos de uso y el resultado alcanzado, todo ello desde la perspectiva del usuario.
- Puede estar asociada a requisitos funcionales o no funcionales.
- Un requisito tendrá una o más PA asociadas.
- Las PA cubren desde escenarios típicos/frecuentes hasta los más excepcionales.

En la siguiente figura se muestra el contexto de TDRE usando como marco el Modelo V (solo para ilustrar los niveles de testeo puesto que a diferencia del Modelo V el proceso presentado a lo largo de esta memoria es iterativo e incremental).

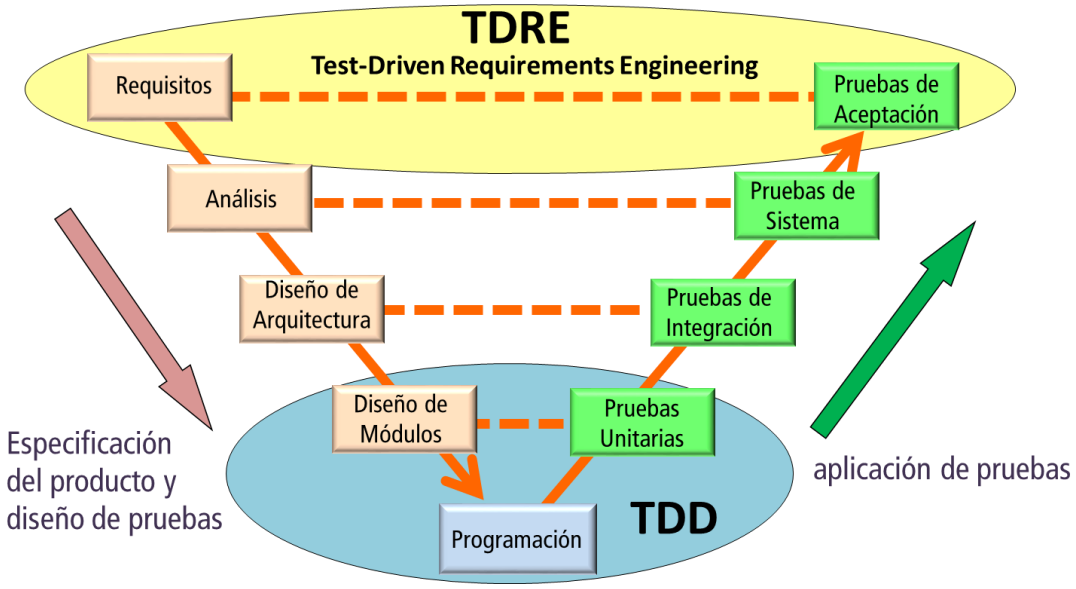

Figura 13: Modelo TDRE

Como se observa en la figura, TDRE es un enfoque que se sitúa en el nivel de la especificación de los requisitos y de las PA. La clave de TDRE es que los requisitos se especifican esencialmente como PA con lo cual éstas quedan establecidas cuando se analiza el cambio, siendo el encargado de analizar la ID (el Analista) quien las define. Posteriormente, el encargado de probar la ID realizará el diseño de las PA estableciendo instanciaciones de datos en términos de Pruebas de Sistema (PS) asociadas a cada PA, y las aplicará para verificar que las PA se satisfacen.

Las pruebas de los niveles de Pruebas de Integración y Pruebas Unitarias son realizadas por los programadores al implementar cada ID, es decir, son tareas incluidas en la actividad de Diseño e Implementación del Workflow de Desarrollo (Figura x).

El ciclo de desarrollo con TDRE se ilustra en la figura a continuación. En dicha figura se observan los tres roles técnicos principales Analista, Programador y Tester. El encargado de analizar la ID define el cambio como PA. En caso de ID solicitadas por clientes o cuando el cambio requiere una validación, el encargado de análisis contactará con clientes para validar las PA, normalmente complementadas con prototipos de IU y/o modelos.

Posteriormente, ya en el Sprint donde se implementa la ID, el programador debe implementar el comportamiento asociado a las PA de la ID, y finalmente el Tester debe instanciar las PA (establecer datos concretos para cada PA) y aplicarlas para comprobar que la implementación satisface las PA definidas por el analista.

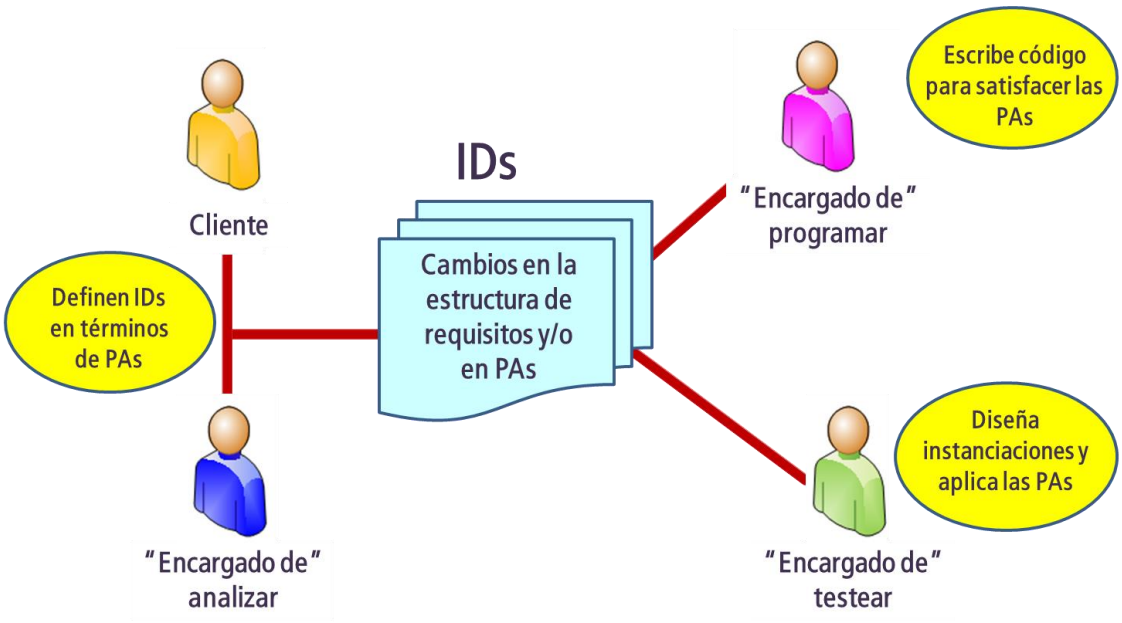

Figura 14: Roles y tareas en TDRE

Al crear las ID y sus PA el analista deberá establecer su relación con la estructura del producto, indicando los nodos afectados y poniendo en ellos las PA correspondientes. En caso de que se trate de un nuevo requisito o una mejora importante de un requisito ya implementado puede extenderse la estructura del producto con los correspondientes nuevos nodos. Así, el mantenimiento del producto se plasma añadiendo, modificando o eliminando PA en los nodos de la estructura, tal como se muestra en la siguiente figura.

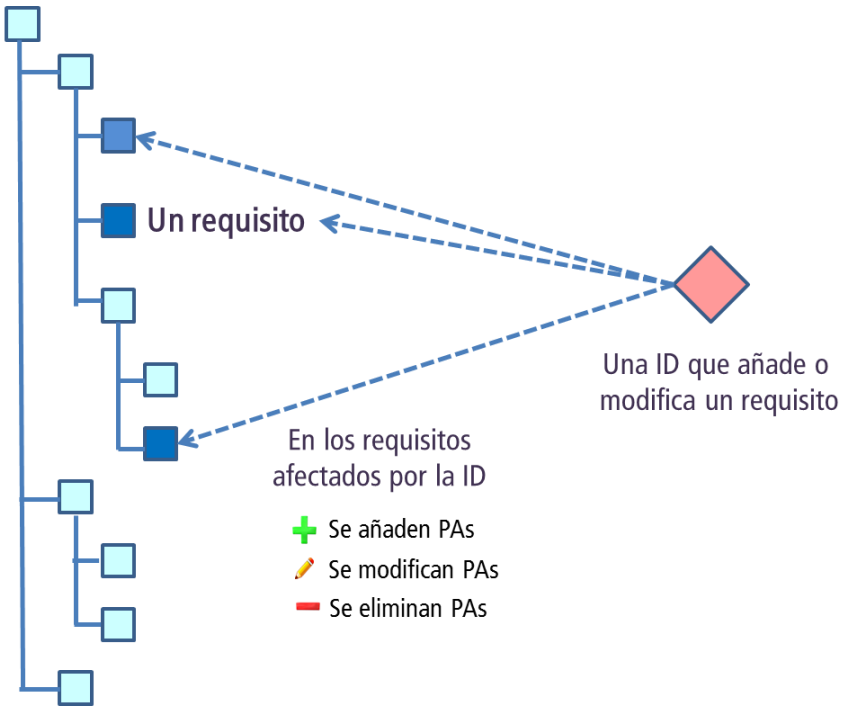

Figura 15: Relación ID y requisito (PA)

Por tanto, y teniendo en cuanta lo anterior, es claro que el número de PA de un producto en la mayoría de ocasiones es cambiante con respecto una versión anterior y pasa por distintas fases o estados:

Figura 16: Estados PA

- **Identificación:** Se capturan los requisitos negociando con el cliente y se identifica el nombre de las PA. Cuando se tienen las PA 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 PA (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 han pasado todas las PA, 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, o dicho de otro modo, se crean las pruebas de sistema.
- **Aplicación:** Se aplica la PA y se valida la implementación.
- **Automatización:** Hasta este nivel todo es manual, a partir de aquí la figura de Automation Tester procederá a la automatización de la prueba (si procede según criterio) para aplicar pruebas de regresión automatizadas.

Por tanto, ahora que hemos explicado detalladamente que es una Prueba de Aceptación y porque son el eje central del desarrollo, vamos a detallar el proceso haciendo uso de las herramientas adecuadas para ello.

Seguiremos el WF de Desarrollo en sus principales etapas y posteriormente introduciremos un nuevo Workflow referido a la automatización de Pruebas de Aceptación.

3.3 Detalle del proceso QA

La primera de las actividades es **Introducir Incidencia**. Puede ser llevada a cabo por cualquier agente, sea cual sea su rol. Los motivos para introducir una nueva incidencia son variados —**mejora, nuevo requisito, corrección de fallo** y **otros**—. En este punto, además del motivo de la introducción, explicado en la descripción, podemos señalar a que programa afecta, la severidad del fallo —si se trata de una corrección de fallo— y algunas opciones más.

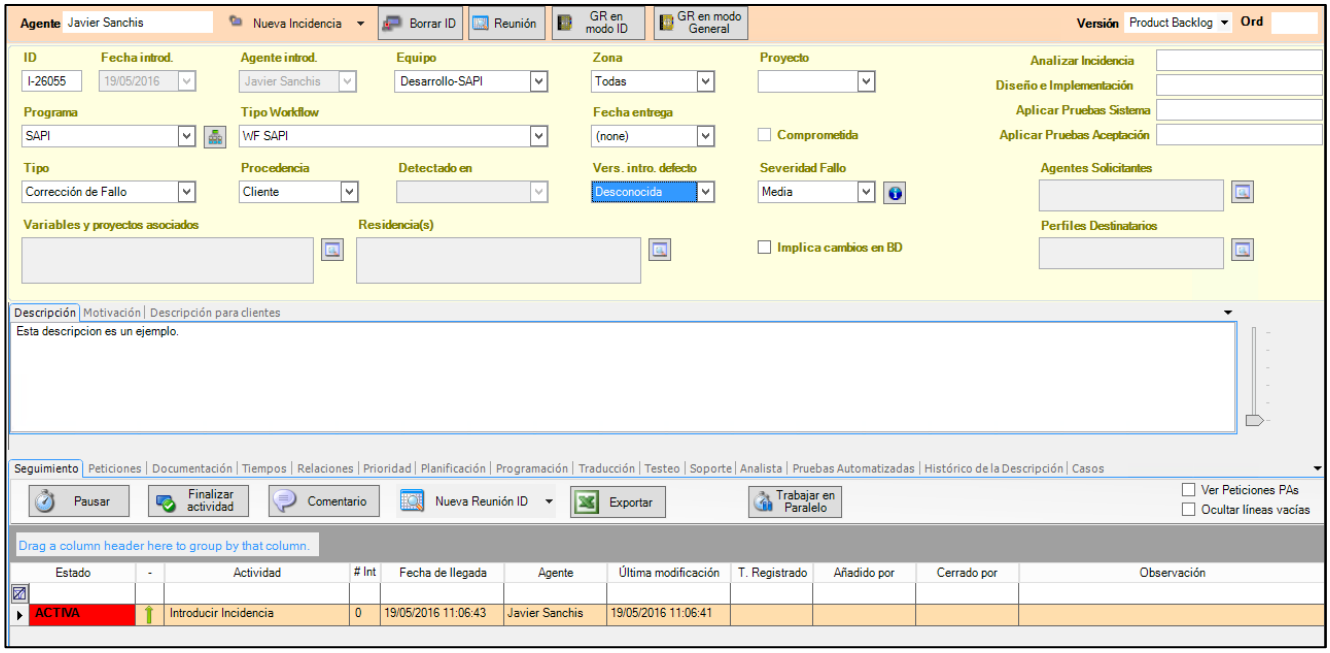

Figura 17: Introducir ID en SAPI

Una vez que se ha introducido la incidencia, si es marcada del tipo Corrección de fallo, el Tester comprueba que efectivamente se trata de un fallo de aplicación y que se produce. En caso de que el Tester compruebe que no se trata de un fallo real, la incidencia es desestimada. En cualquier otro caso, bien se trate efectivamente de un error o la incidencia haya sido creada con un tipo distinto a Corrección de fallo, pasamos a la actividad **Analizar Incidencia.**

En esta actividad fundamental para todo el proceso QA, el Analista que recoge la actividad es el encargado de marcar que Pruebas de Aceptación se van a ver afectadas por la ID así como que nodos o partes del programa pueden verse afectados por la resolución de la ID. Estas Pruebas de Aceptación pueden ser o bien nuevas, de modificación —cuando por cambio en la funcionalidad, tiene que cambiar la definición de la PA— o de regresión — pruebas cuya definición aún tiene que explicitar el comportamiento de la aplicación después de la resolución de la ID—.

Las pruebas son marcadas o creadas en los distintos nodos afectados por la incidencia. **Los analistas tienen que ser especialmente cuidadosos ya que como hemos dicho anteriormente, todo el proceso de desarrollo girara en torno de estas PA.** Estas pruebas son marcadas en el Gestor de Requisitos que hemos nombrado anteriormente, donde cada nodo del árbol que hace referencia a la estructura del programa contiene las distintas Pruebas de Aceptación.

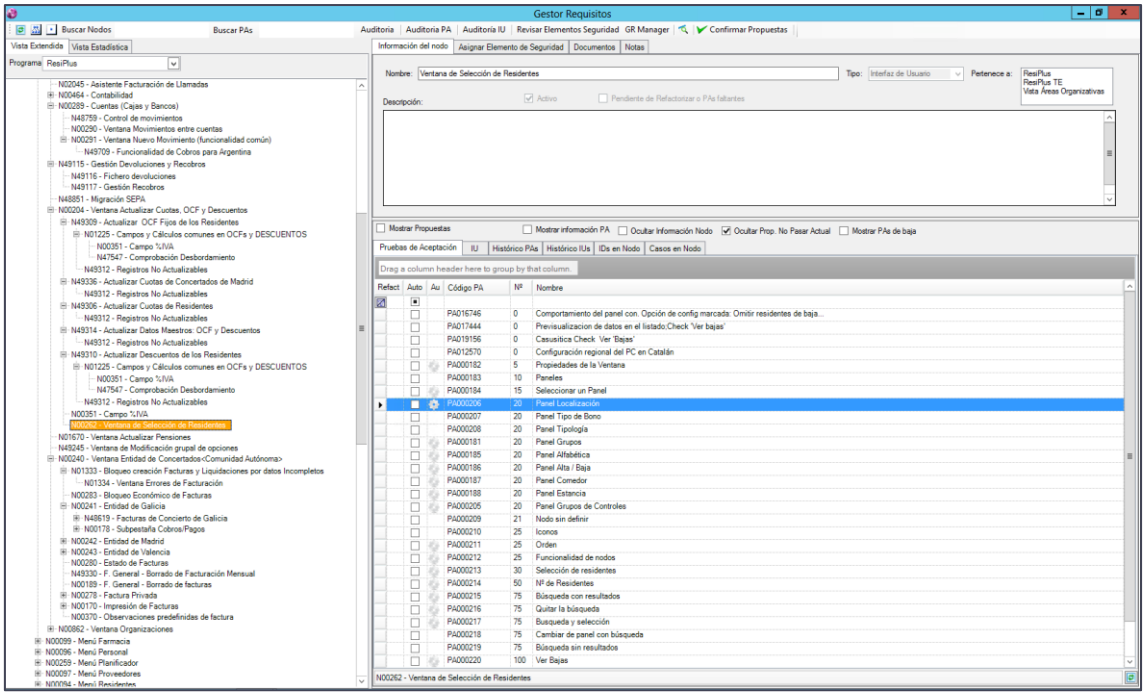

Figura 18: Marcar Nodos y PA en el Gestor de Requisitos en SAPI

Cuando los analistas terminan su trabajo, la incidencia pasa a la actividad de **Diseño e Implementación** donde los programadores implementarán los cambios o nuevas funcionalidades del producto de manera que cumplirán con las distintas definiciones de las PA marcadas por los analistas. **Esto es de vital importancia para el proceso y el programador tiene que ser muy riguroso en este aspecto**.

Conforme los programadores van cumpliendo con las pruebas de aceptación, estás son marcadas con el pertinente OK desde SAPI y **solo cuando estén todas implementadas correctamente bajo el criterio y responsabilidad del agente que se ha encargado de ello, deberán finalizar la actividad.**

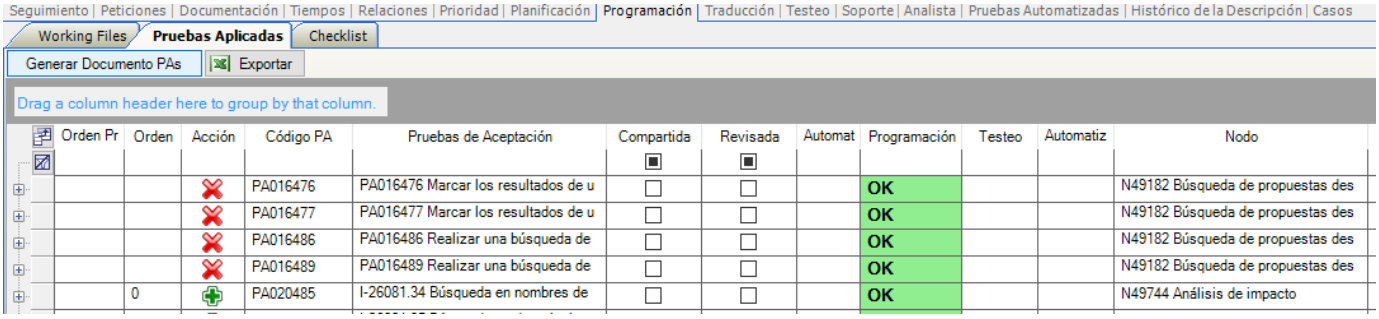

Figura 19: Implementacion OK por parte del programador siguiendo PA

La siguiente actividad es **Aplicar Pruebas Manuales**, donde los agentes encargados son los Testers Manuales. Normalmente dentro del equipo de testeo hay algunas personas más especializadas en unas zonas del programa que en otras, es por ello, que dependiendo de la zona del programa a la que hace referencia la incidencia, esta actividad será asignada a un Tester o a otro.

La función del Tester Manual es comprobar que la implementación ha sido correctamente realizada a nivel de Prueba de Aceptación, es decir, que efectivamente las PA marcadas por los analistas sobre las cuales ha girado el trabajo de los programadores, se satisfacen. Para ello se procede manualmente pues se trata de un testeo manual, aunque posteriormente veremos que muchas de las pruebas del sistema que componen las pruebas de aceptación están automatizadas.

Conforme el agente encargado de la actividad comprueba que las pruebas son satisfechas, son marcadas con OK en la pestaña "Testeo". En otro caso, donde la prueba de aceptación no se cumpla, la incidencia volverá a la actividad "Diseño e Implementación" para que sea corregida.

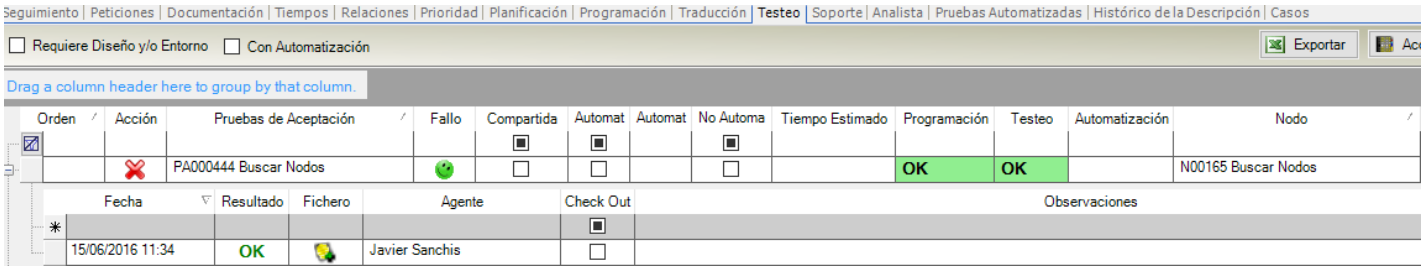

Figura 20: OK de la PA implementada por parte de los Testers Manuales

Para dar por terminada una ID ésta debe satisfacer todas sus PA. Correspondientemente, para dar por terminado un Sprint, todas las ID del Sprint deben estar terminadas. Dado que las ID se van terminando a lo largo del Sprint, puede suceder que después que una ID ha sido probada con éxito, otra ID introduzca algún cambio que invalide la ID anterior. Esto mismo puede darse entre diferentes Sprint, es decir, que un cambio en una ID de un Sprint tenga efectos secundarios no deseados en el comportamiento ya implementado y probado en Sprint anteriores.

Así pues, para detectar estos posibles fallos es importante aplicar Pruebas de Regresión. Una Prueba de Regresión es una prueba que ya se aplicó anteriormente con éxito pero que se vuelve a aplicar para verificar que el comportamiento asociado a la prueba se sigue cumpliendo. Como se muestra en la Figura 21, cada ID tiene básicamente tres actividades: Requisitos, Programación y Pruebas. Aunque en cada ID se realizan pruebas del comportamiento que modifica la ID, al final del Sprint, es importante realizar Pruebas de Regresión, tanto del comportamiento ya probado en cada ID como del comportamiento global ya implementado en el producto en Sprint anteriores.

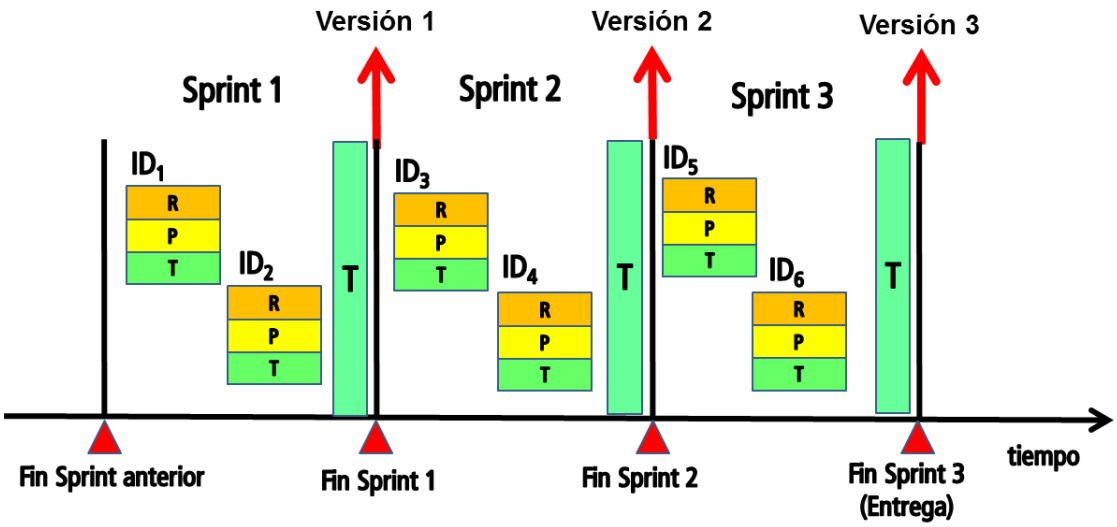

Figura 21: Resumen Sprint y tareas (R-P-T)

Evidentemente pasar manualmente más de 15000 Pruebas de Aceptación, como algunos programas desarrollados tienen, es algo inalcanzable, por lo que en paralelo al trabajo de desarrollo, el equipo de automatización de pruebas hace su acometido.

Los Analistas y Testers establecen qué PA deberían ser automatizadas. **El equipo de automatización de pruebas, además de automatizar pruebas, también está encargado de aplicar las suites de pruebas de regresión automatizadas**. Así, durante la realización de un Sprint, el equipo de automatización realiza las siguientes tareas:

- **Automatización de pruebas**. Estas pruebas automatizadas estarán disponibles para añadirse a las suites de pruebas de regresión automatizadas.

- **Refactorización y actualización de pruebas automatizadas**. Realización de mejoras de rendimiento y organización del código de pruebas automatizadas, mejoras de infraestructura para la ejecución de pruebas, o actualización del código de pruebas debido a cambios en las PA correspondientes.

- **Aplicación de suites de pruebas de regresión automatizadas a demanda**. Un programador puede solicitar aplicar una suite de pruebas para el contexto de la ID con la que está trabajando.

Como equipo 'independiente' al de desarrollo propiamente dicho, se dispone de un Workflow de Automatización de Pruebas que será el flujo de trabajo normal para automatizar Pruebas de Aceptación.

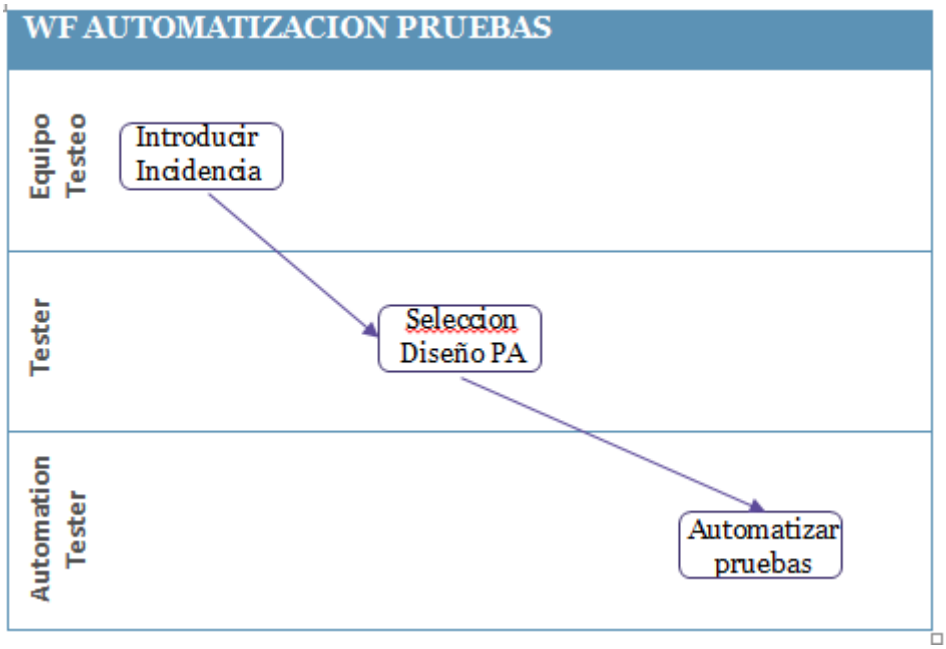

Figura 22: Workflow Automatización de Pruebas

La primera de las actividades es realizada haciendo uso de SAPI, y aunque las dos siguientes también hacen uso de este programa, nos tenemos que apoyar en otros para llevarlas a cabo. Dicho programas se explican a continuación.

En la actividad Introducir Incidencia, el equipo de testeo ya ha decidido que PAs son las elegidas para automatizar en la incidencia creada para ese efecto. El criterio seguido viene dado principalmente por criticidad y uso de la funcionalidad. El equipo de testeo realiza reuniones semanales para seleccionar las PAs a automatizar.

Una vez que creada la incidencia, en la siguiente actividad hay que diseñar las distintas instanciaciones de la PA, es decir, hay que diseñar las Pruebas del Sistema o PS asociadas. Estas PS podemos verlas como instanciaciones e la PA, casos particulares que deberán de abordar la mayor parte de caminos que pueden darse siguiendo la definición de la PA.

Esta actividad la lleva a cabo los Testers Manuales pues tiene un amplio conocimiento del producto y hay que ser muy cuidadoso pues de un buen diseño dependerá la calidad de la prueba automatizada y como hemos dicho en numerosas ocasiones a lo largo de este documento, el proceso se basa constantemente en el cumplimento de las distintas PAs que componen el producto.

Para realizar el diseño, los Testers Manuales hacen uso de una herramienta específica para ello, y creada en la propia empresa, el Diseñador de Pruebas.

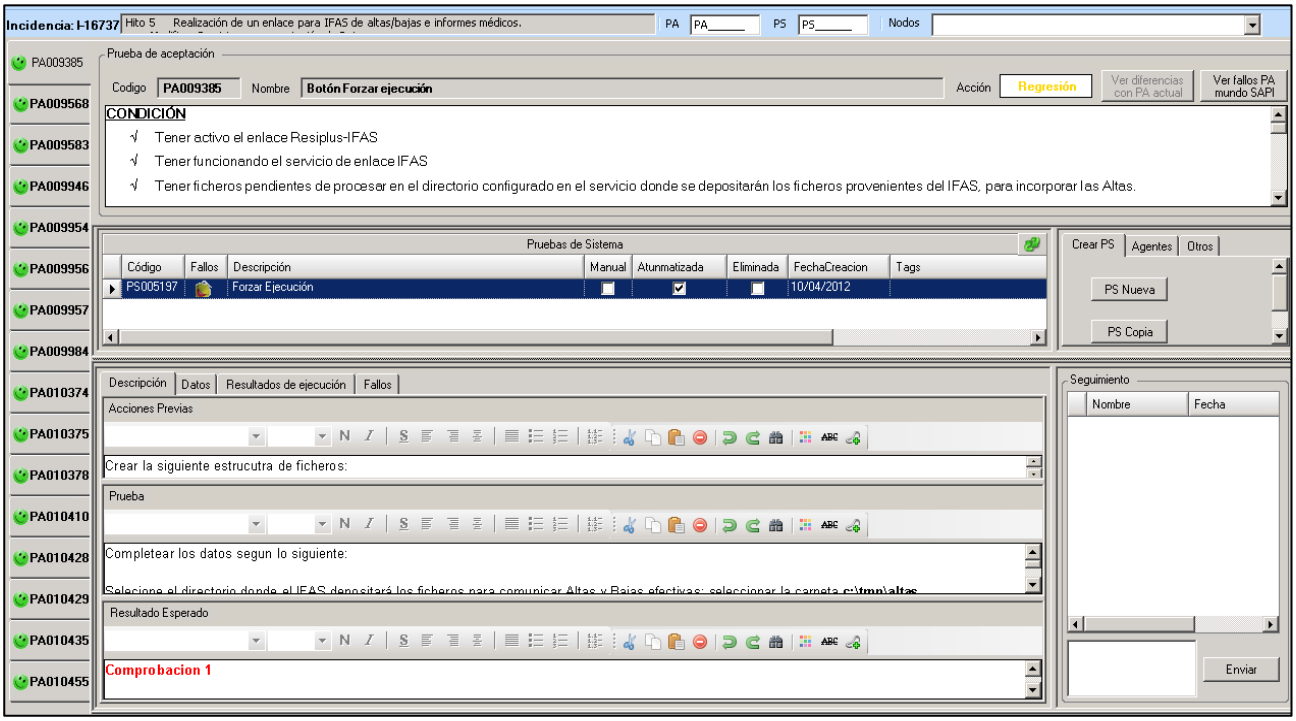

Figura 23: Diseñador de PS

Como vemos en la figura anterior, en la parte izquierda se presentan las PAs que se van a automatizar en la incidencia seleccionada, hay una opción para visualizar todo el conjunto, pero el objetivo es crear los diseños de las PS, por ello, por defecto, se discriminan las PA que no se van a automatizar.

En la parte superior tenemos la descripción de la PA seleccionada mientras que en la parte central se muestra el grid de las PS creadas para esa PA. En la parte inferior tenemos toda la información referente a la PS seleccionada, su correspondiente diseño, 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. Por ultimo a la parte derecha del grid disponemos de las opciones posibles para crear PS o copiar.

A continuación y a modo de ejemplo, vamos a describir un caso de automatización de una PA cuya definición es la siguiente:

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

Figura 24: Definición de una PA

Si analizamos la condición, pasos y resultado de la PA, podemos vislumbrar que tenemos tres casos base posibles para describir las pruebas del sistema, uno introduciendo letras en el campo DC, otro introduciendo caracteres especiales y por último introduciendo **. Estos tres casos deberían probarse en todas las partes del programa donde exista la opción de introducir una cuenta bancaria. Por tanto, es necesario disponer de cierta experiencia y conocimiento del producto para tener una alta cobertura de prueba.

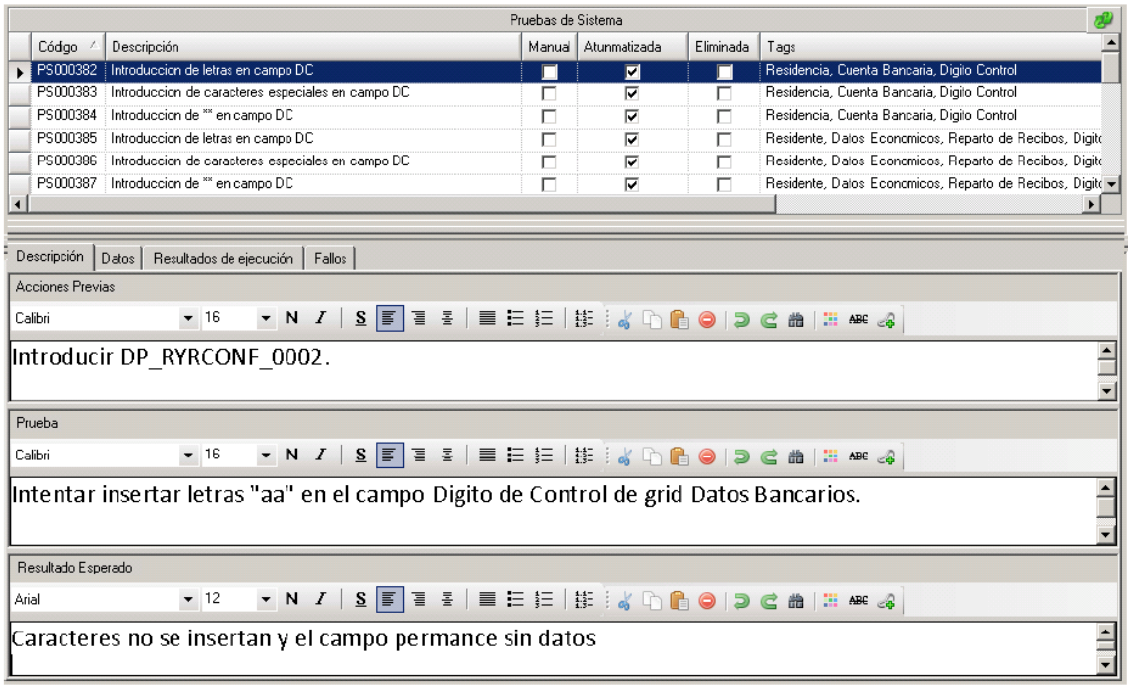

Figura 25: Definición de una Prueba de Sistema o PS

Según la figura anterior, en Acciones Previas describimos las condiciones que se pedían en la PA, en Prueba tendremos los pasos que hay que seguir, que están descritos en la PA, y en el Resultado Esperado lo que corresponda. Como vemos, aparece el concepto de datapool, entendiendo como tal un conjunto de datos con valores concretos que se utilizarán durante la ejecución.

En este caso DP_RYRCONF da valor a un conjunto de checks que hay en una zona del programa al que hace referencia esta prueba, 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 puede haber en un datapool corresponden a las posibles combinaciones de datos que necesitemos. Señalar que estos Datapools pueden ser creados y modificados desde el mismo diseñador.

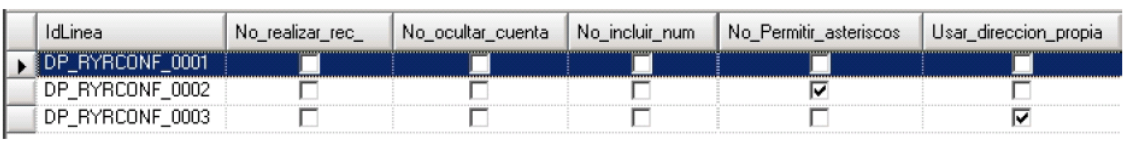

Figura 26: Datapool

Una vez el Tester ha acabado de diseñar, manda un aviso al Automation Tester y este ya sabe que tiene que comenzar a automatizar las distintas PS de la PA.

Para automatizar, el Automation Tester, hace uso de la herramienta **Rational Functional Tester**, a partir de ahora RFT. Se trata de una herramienta avanzada de testing funcional y de regresión automatizado creada para los Testers y los desarrolladores de GUI que necesitan de un control superior para probar sus aplicaciones Java, Microsoft Visual Studio .NET y Web.

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

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

Cuando el Automation Tester termina de automatizar una PA, la marca como automatizada de modo que en SAPI sale como tal y ya no es objeto de revisión manual por parte de los Testers Manuales si no que pasa a ser incluida en el conjunto de pruebas automatizadas.

Como vemos, SAPI nos dota de la consistencia y coherencia necesaria entre las distintas actividades por las que pasa el proceso QA facilitando la comunicación entre los distintos departamentos que participan en este.

Como hemos comentado anteriormente dentro de las funciones del Automation Tester, además de automatizar pruebas y de refactorizarlas si fuese necesario, también debe lanzar Suites o conjunto de pruebas automatizadas de manera rutinaria o por demanda de los Testers Manuales o de algún desarrollador que quiera probar alguna funcionalidad que ha implementado en alguna zona concreta del programa.

Las pruebas o conjunto de pruebas, llamadas Suites (lanzamiento conjuntos de pruebas que afectan a una misma parte del producto, las pruebas que han fallado en lanzamientos anteriores o simplemente todas las PS automatizadas) son lanzadas por los Automation Testers mediante el **Lanzador ATUN**, una aplicación creada expresamente para ello y que lanza las pruebas contra una serie de máquinas virtuales que contienen la versión actual del producto que estamos testeando de manera continuada. Por tanto, otra función que tiene el Automation Tester es mantener estas máquinas en un estado consistente para garantizar que el proceso se está realizando correctamente.

Este estado consistente debe que ser el siguiente:

- Las máquinas sobre las que lanzamos las suites tienen que estar disponibles y encendidas.

- Deben tener la versión del producto instalada lo más actualizada posible, concorde a la que tienen los desarrolladores y sobre la que han realizado las modificaciones e implementaciones.

- Tenemos que vigilar que el comportamiento sea el adecuado y si no lo es investigar por qué y buscar soluciones.

Todos los días se realizan compilaciones automáticas del producto software, tanto para el equipo de desarrollo como para el equipo de automatización de pruebas. Tienen el mismo núcleo, pero difieren en algunos parámetros como interfaz y depuración. Más adelante comentaremos algunos problemas relacionados con estos puntos y donde se han ido realizando mejoras paulatinamente.

La herramienta Lanzador ATUN, como hemos dicho, nos permite lanzar Suites contra máquinas virtuales con la versión del producto a testear. Podemos crear Suites con las PS que nos interese, pero hay 3 que permanecen de manera constante y qué son revisadas a lo largo de cada Sprint.

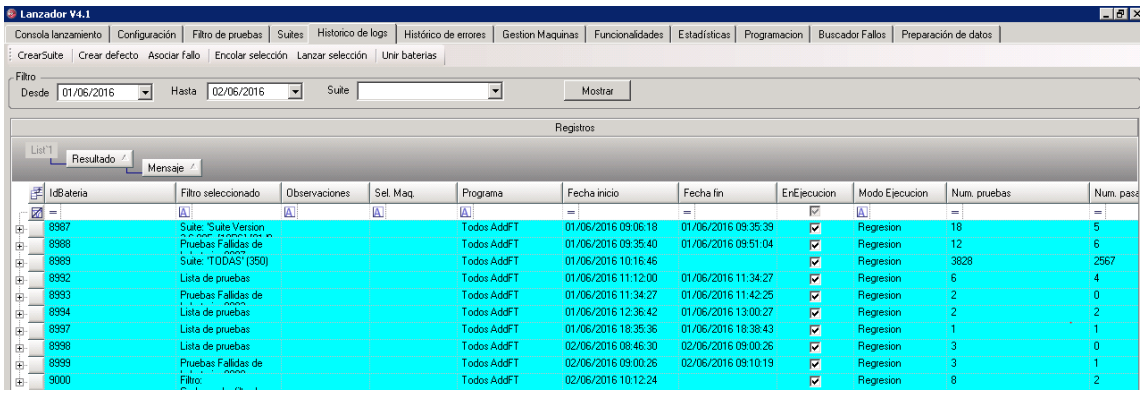

Figura 27: Lanzador de Pruebas Automatizadas ATUN

- Suite Total: en esta suite encontramos la totalidad de las PS automatizadas.

- Suite 0: las PS que en la última ejecución pasaron satisfactoriamente.

- Suite 1: las PS que en la última ejecución no pasaron satisfactoriamente.

Además de estas Suites, también tenemos creadas baterías que revisan alguna parte concreta del producto, como pueden ser la económica, la asistencial, o la miscelánea. De esta manera cuando un desarrollador tiene dudas de si su trabajo está afectando negativamente otras funcionalidades del programa, nos puede pedir que lancemos a demanda una de estas Suites para comprobarlo.

Es importante destacar que el **desarrollo de cada versión del producto pasa por dos fases** distintas y que dependiendo de ellas el lanzamiento de pruebas tiene que ser más o menos intensivo aunque siempre continuo.

Figura 28: Fases en el proceso de desarrollo

La primera de las fases, denominada fase de desarrollo, el equipo de desarrollo está trabajando sobre incidencias que solucionaran o agregaran funcionalidad a la nueva versión. En esta fase el Automation Tester tiene que lanzar de manera constante la Suite de Versión que contiene todas las pruebas sobre las que está girando el desarrollo de la versión. De esta manera un error es localizado tempranamente ahorrando costes/tiempo al detectarlo en una fase temprana de desarrollo. Es importante mantener las maquinas sobre las que lanzamos las pruebas actualizadas ya que cada día hay una compilación y esta compilación puede agregar funcionalidad o corregir errores en el producto desarrollado.

La segunda fase, preproducción, dura en torno a una semana y la versión ya está cerrada, por lo que los desarrolladores ya no están implementando nada nuevo para la dicha versión. Los Automation Testers deben actualizar las máquinas virtuales y lanzar la Suite Total, que contiene todas las pruebas, para comprobar que ninguna de las modificaciones ha alterado la funcionalidad general del producto software.

3.3.1 Gestión de fallos

Los resultados obtenidos del lanzamiento de las distintas pruebas o Suites pueden ser los siguientes y disponemos de un histórico para acceder a cada lanzamiento:

- **PASS**: la prueba ha pasado correctamente.
- **Fallos ocasionales:** simplemente tenemos que volver a realizar otra ejecución de la prueba ya que se trata de un fallo ajeno a la aplicación o a la prueba, siendo un fallo de la máquina, ya sea por recursos o por aspectos ajenos a la ejecución. En los fallos de la prueba se detecta la zona del código de la prueba donde no se está reconociendo algún control de manera adecuada o a cambiado

el funcionamiento de la prueba, ya que se trata de un sistema (RFT) que realiza un reconocimiento gráfico. **En estos casos es función del Automation Tester refactorizar la prueba.**

- **Fallos de programación:** se realizar un reporte mediante el **Gestor de Fallos**, un programa interno y creado expresamente para ello. Desde este podemos crear un fallo proporcionando información sobre cómo se reproduce el error, en que está fallando y que PA/PS están siendo afectadas. Una vez creado el error, se manda un aviso a los Testers Manuales para que confirme el error y comenzar nuevamente el ciclo.

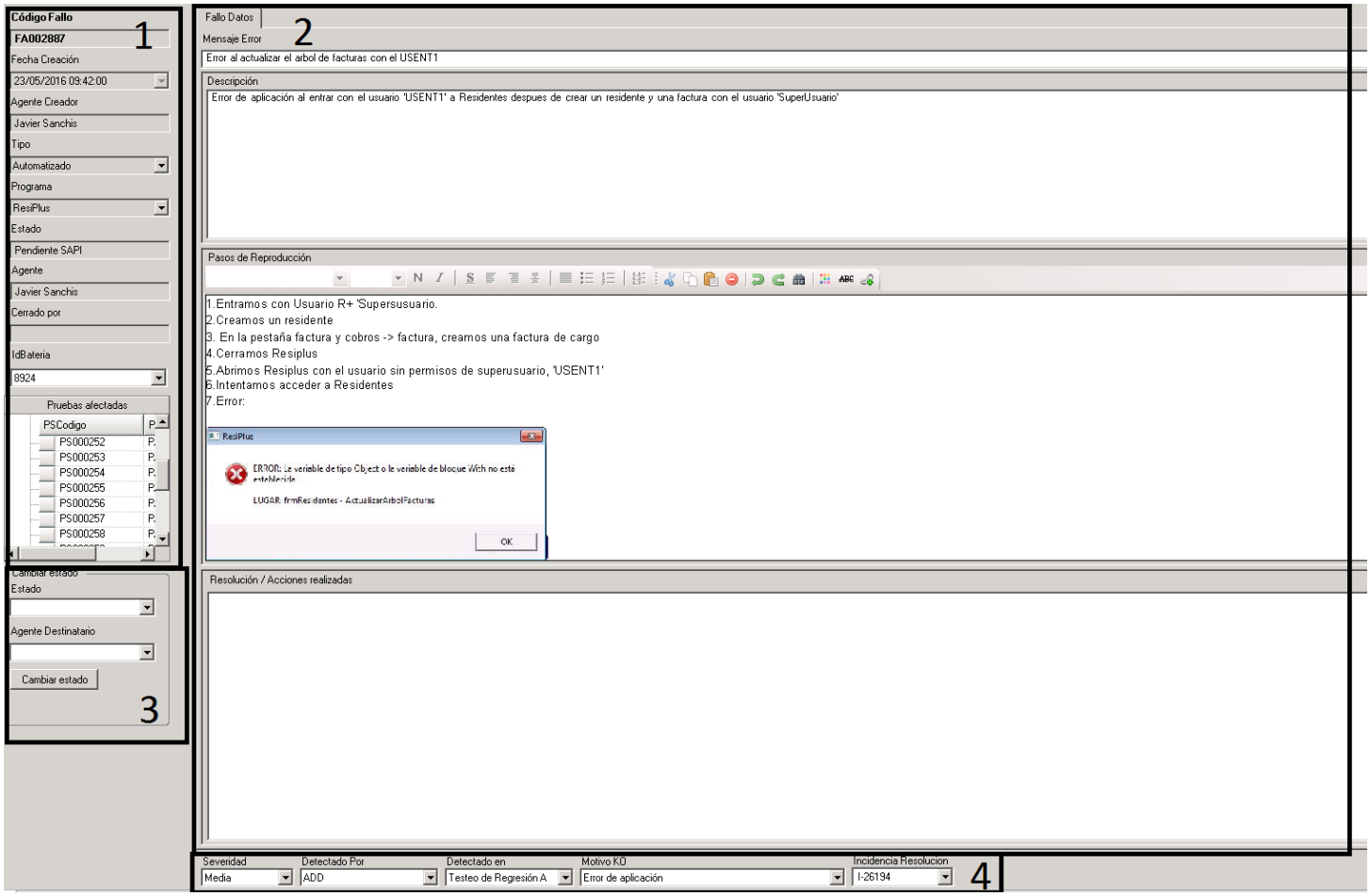

Figura 29: Gestor de fallos

La ventana de creación de fallo la podemos dividir en 4 zonas. En la primera de ellas (1) tenemos información no editable del fallo como puede ser su código, el agente introductor, fecha, el programa al que afecta el fallo y las Pruebas de Sistema afectadas. En la segunda zona (2), tenemos distintos cuadros de texto donde debemos describir el fallo y realizar una minuciosa redacción para la correcta reproducción de este por parte del Tester Manual. En la tercera zona (3) disponemos de un par de desplegables para indicar en qué estado se encuentra el fallo y que agente va a ser avisado. Actualmente el aviso es manual pero estamos trabajando para automatizarlo y que le llegue al Tester Manual como un aviso en SAPI. Por ultimo en la cuarta zona (4) podemos indicar la

severidad del fallo, quien lo ha detectado (si el cliente o algún componente del equipo), en que actividad, el motivo por el que se produce y si el Tester Manual confirma que efectivamente es un error y crea la incidencia para resolverla, podemos asociarla a está cerrando el ciclo.

Como era de esperar y es deseable, cuando se crea un fallo se puede acceder a este desde SAPI, desde un buscador de fallos que incorpora.

Por último, si el fallo es desestimado, o si ha sido resuelto el Tester Manual debe acceder de nuevo al fallo creado y cambiar su estado (zona 3 de la Figura 10) a Desestimado o Cerrado respectivamente.

La gestión de fallos es fundamental para que éstos se resuelvan de forma satisfactoria para el cliente. Los fallos pueden ser detectados también por los desarrolladores durante el proceso de desarrollo y mantenimiento o por los usuarios al utilizar la aplicación, obviamente esto último es lo menos deseable. La siguiente figura resume la detección de fallos en el contexto de trabajo de un Sprint y las posibles alternativas para gestionarlos.

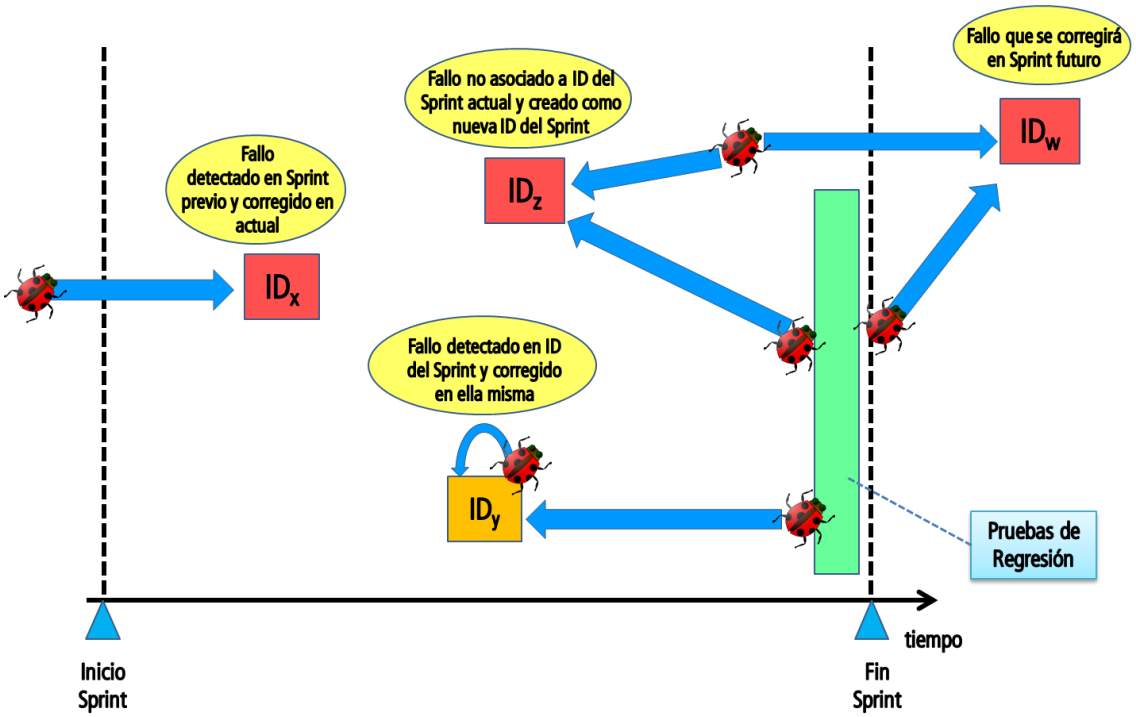

Figura 30: Gestión de los fallos [1]

Cuando el fallo detectado está asociado a una ID del Sprint actual, su corrección es abordada en dicha ID. Sin embargo, si el fallo NO esté asociado a la implementación de una ID en el Sprint actual, se crea una nueva ID del tipo “Corrección de fallo”. En este

último caso, dependiendo de la severidad del fallo dicha ID de corrección del fallo se abordará en el Sprint actual o en un próximo Sprint.

La siguiente tabla, tomando como referencia [2], muestra los niveles de severidad según los cuales se clasifica cada fallo. De acuerdo al nivel de severidad se determina la urgencia con la cual debe corregirse el fallo. Un fallo crítico puede conllevar que se resuelva inmediatamente y se publique una actualización directamente a producción. Un fallo severo podría incluirse como ID en el Sprint actual. Fallos moderados o leves podrían incluso esperar a ser resueltos en nuevos Sprint. De todas formas, los fallos son en general priorizados para que se resuelvan lo más pronto posible.

Severidad del Fallo	**Indicadores** (puede que NO se presenten todos a la vez)
Crítico	Nivel de detección del fallo por el usuario: **Muy Alto** Frecuencia con la que se presenta: **Muy Alta** Número de clientes/usuarios afectados: **Muy Alto** Requiere soporte en línea: **SI** Pérdida de funcionalidad esencial: **SI** Pérdida de datos: **SI** Problema de seguridad: **SI** El fallo tiene workaround aceptable para el usuario: **NO**
Severo/Importante	Nivel de detección del fallo por el usuario: **Alto** Frecuencia con la que se presenta: **Alta** Número de clientes/usuarios afectados: **Alto** Requiere soporte en línea: **Posiblemente SI** Pérdida de funcionalidad: **NO** Pérdida de datos: **NO** Problema de seguridad: **NO** El fallo tiene workaround aceptable para el usuario: **SI**
Moderado	Nivel de detección del fallo por el usuario: **Medio** Frecuencia con la que se presenta: **Media** Número de clientes/usuarios afectados: **Pocos** Requiere soporte en línea: **Posiblemente NO** Pérdida de funcionalidad: **NO** Pérdida de datos: **NO** Problema de seguridad: **NO** El fallo tiene workaround aceptable para el usuario: ---
Leve	Nivel de detección del fallo por el usuario: **Bajo** Frecuencia con la que se presenta: **Baja** Número de clientes/usuarios afectados: **Muy Pocos** Requiere soporte en línea: **NO** Pérdida de funcionalidad: **NO** Pérdida de datos: **NO** Problema de seguridad: **NO** El fallo tiene workaround aceptable para el usuario: ---

Figura 31: Severidad del fallo [2]

Como resumen al proceso QA explicado podemos presentar la Figura 32:

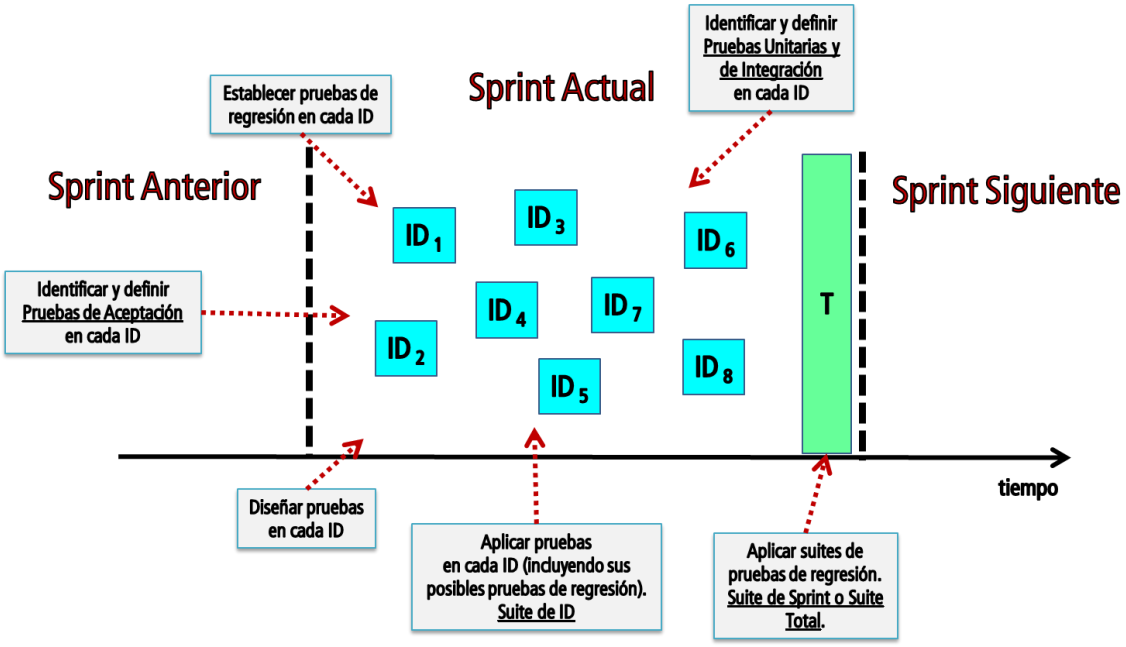

Figura 32: Resumen proceso QA

La identificación y definición de PA asociadas a una ID, así como la decisión de aplicar ciertas pruebas de regresión, se realiza durante el análisis de la ID.

El programador, al implementar la ID, tiene como objetivo satisfacer las PA establecidas. Además, el programador debería realizar pruebas unitarias y de integración del código generado.

Los testers se encargan del diseño de las pruebas de sistema (PS) asociadas a las PA. El diseño de las pruebas debe prepararse antes de aplicar manualmente las PA de una ID. Estos diseños pueden ser utilizados posteriormente para automatizar la PA, o para volver a aplicarla manualmente.

Al terminar cada ID los testers aplican manualmente las PS para verificar la satisfacción de las correspondientes PA. También, al final de Sprint los testers vuelven a revisar el comportamiento de cada ID implementada.

El equipo de automatización aplica suites de pruebas automatizadas durante el Sprint y en particular al final del Sprint.

4. Métricas del proceso QA

Para aumentar el valor y la madurez del proceso QA en el desarrollo de software, es necesario desarrollar y crear métricas para rastrear su calidad en el estado actual y comparar la mejora con las versiones previas. Actualmente disponemos de varios mecanismos e indicadores para controlar el proceso QA y ver su evolución:

- Tiempo destinado a cada tipo de ID por versión
- Número y criticidad de las incidencias del tipo Corrección de Fallo por versión
- Resultados históricos de Suites lanzadas en el Lanzador ATUN
- PA involucradas en cada versión
- Numero de fallos creados en el Gestor de Fallos.
- Histórico de PS creadas/automatizadas

Todos estos datos son accesibles desde distintas herramientas y surgió la necesidad de aunar dicha información en un Dashboard dinámico.

Esta información esta extraída del TFG presentado por Alejandro del Rincón López, titulado "Dashboard de testo para entornos con pruebas automatizadas"

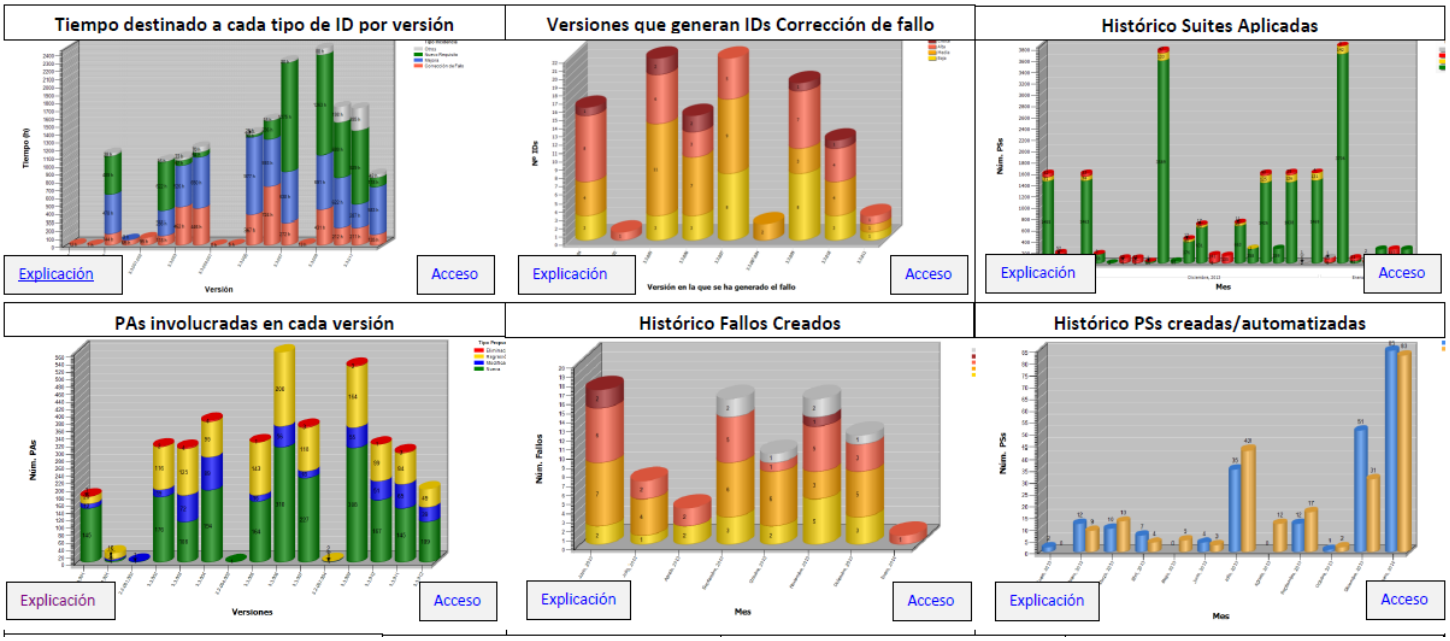

Figura 33: Resumen de los Dashboard

4.1 Tiempo destinado a cada tipo de ID por versión

Report en el cual aparece el volumen de incidencias y el tiempo invertido en cada una de ellas en una versión de un programa. Cada barra corresponde a una versión y aparece coloreado el esfuerzo invertido en cada tipo de incidencia.

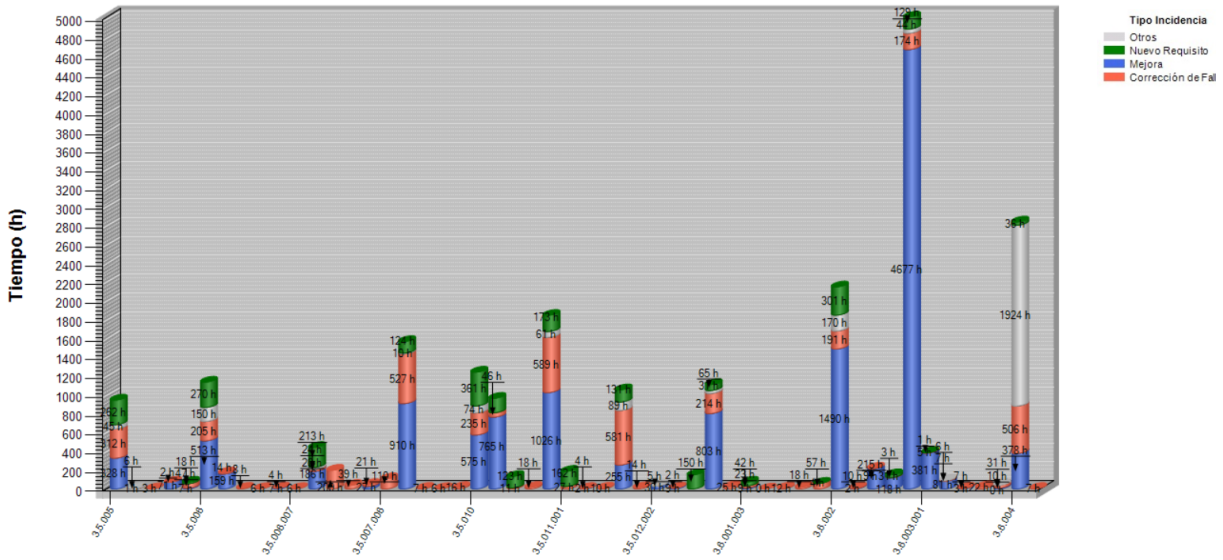

Figura 34: Dashbloard 'Tiempo destinada a cada ID por versión'

Debe tenerse en cuenta que el esfuerzo invertido en las incidencias de una versión no se concentra normalmente solo en el período de la versión, gran parte del trabajo puede haber sido realizado en versiones anteriores, especialmente el trabajo de análisis.

Además, como complemento para la interpretación se tiene otra gráfica que muestra **por versión el número de incidencias de cada tipo.**

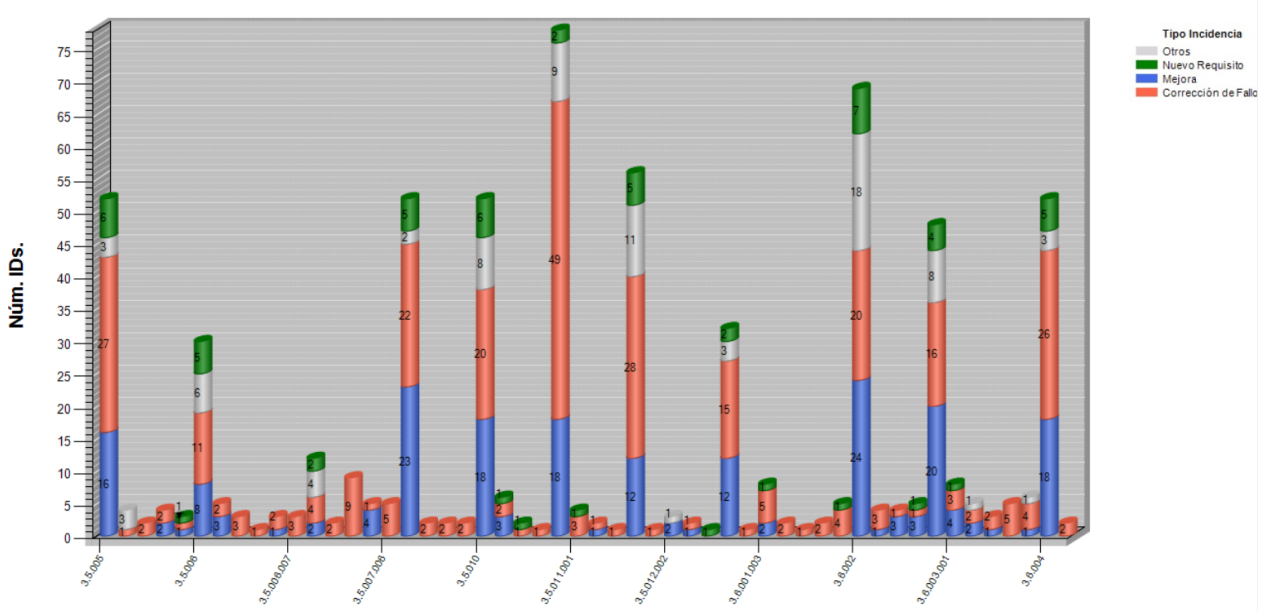

Figura 35: Dashboard 'Num y tipo de ID por versión'

4.2 Número y criticidad de las ID del tipo Corrección de Fallo por versión

Report en el cual aparece el **volumen de incidencias de tipo corrección de fallo generadas en una versión.** En este caso, las incidencias se clasifican por la criticad del fallo. Podemos filtrar los resultados por programa, versión, y por la procedencia del fallo (Cliente o equipo).

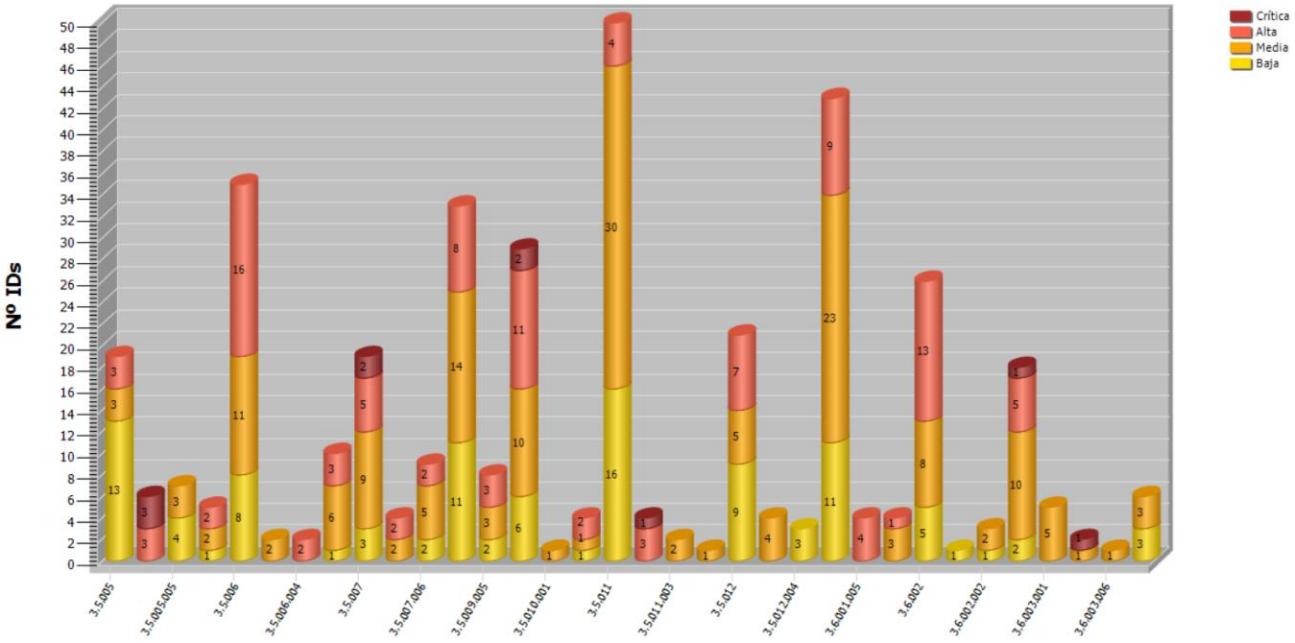

Figura 36: Dashboard 'Num y criticidad de ID de Correccion de fallo por versión'

4.3 Resultados históricos de Suites lanzadas en el Lanzador ATUN

Report en el cual aparece la información de las diferentes ejecuciones de suites de pruebas en un periodo de tiempo determinado (por defecto se muestran los dos últimos meses). En cada ejecución de suite, además, aparece el número de pruebas dependiendo del resultado.

Para interpretar correctamente la gráfica es importante conocer nuestra estrategia de aplicación de suites. Si la suite es relativamente pequeña normalmente se pasará siempre de forma completa, incluso después de obtener resultados FAIL. Sin embargo, cuando se trata de una suite que puede tardar mucho, primero se pasa la suite completa y luego mientras existan resultados FAIL, solo se aplican las PS que han dado FAIL en la última ejecución (aunque pueda quizás al final volver a pasarse la suite completa).

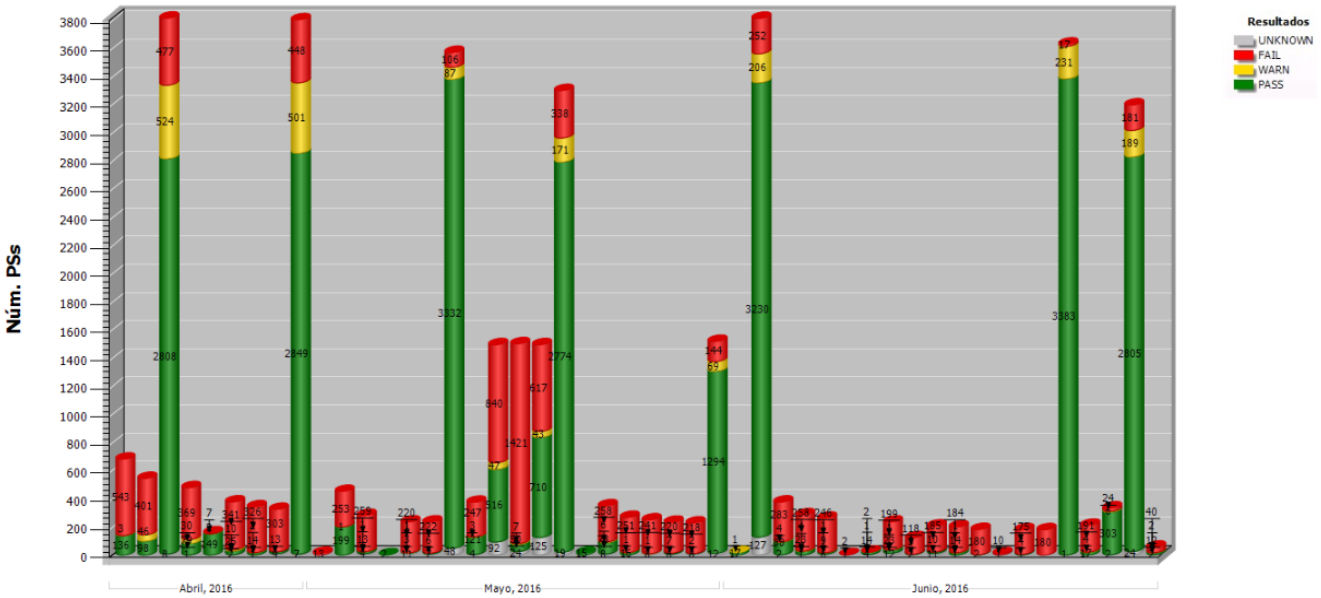

Figura 37: Dashboard 'Resultado del lanzamiento de las SUITES'

4.4 PA involucradas en cada versión

Report en el cual podemos ver el volumen de Pruebas de Aceptación involucradas en las incidencias de una versión de uno de los productos.

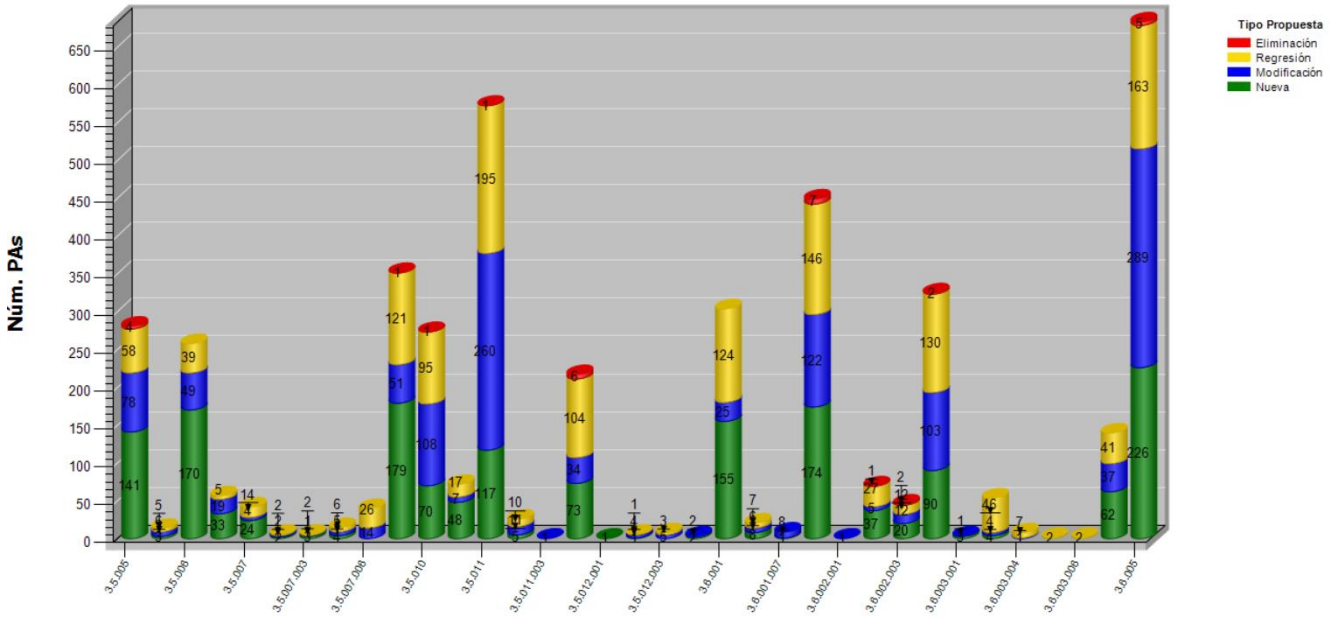

Figura 38: Dashboard 'PA involucradas en cada versión'

4.5 Numero de fallos creados en el Gestor de Fallos.

Report destinado a visualizar la información de los fallos creados en cada mes. En este caso no se trata solo de Incidencias de Corrección de Fallo, sino de fallos registrados en el Gestor de Fallos, es decir, cualquier defecto detectado, tanto durante la implementación de una ID como después de terminada.

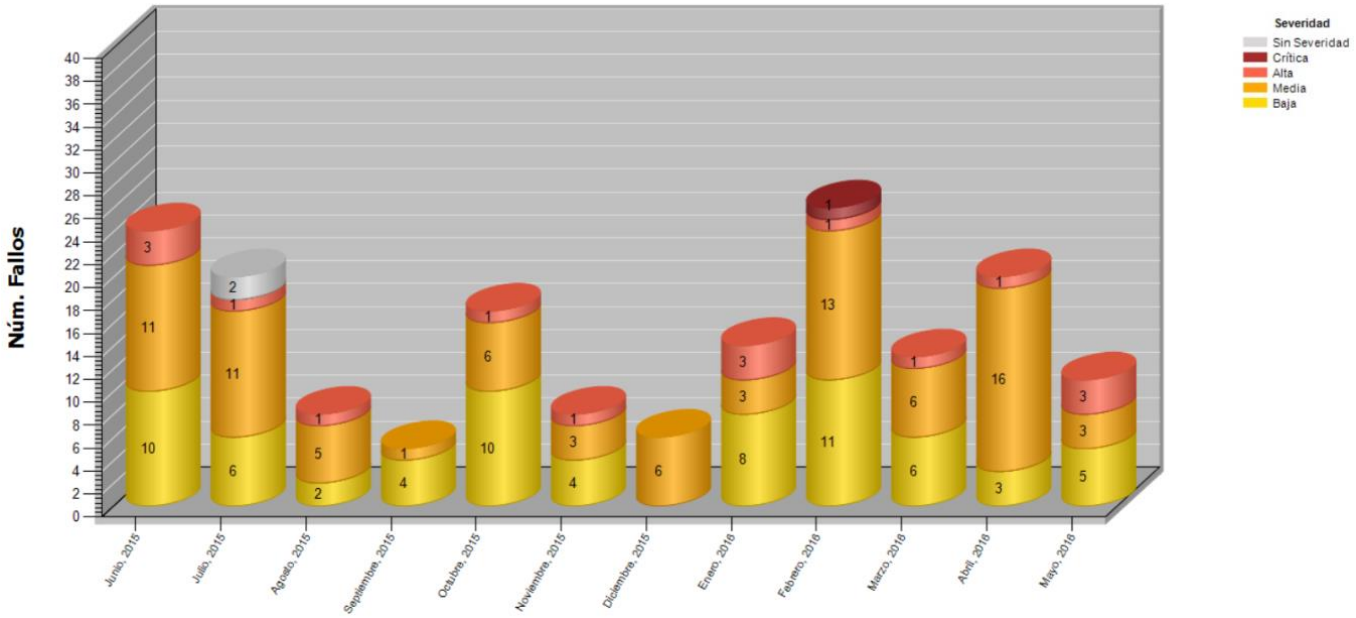

Figura 39: Dashboard 'Num fallos creados en el Gestor de Fallos'

4.6 Histórico de PS diseñadas/automatizadas

Report en el cual aparece la información de las pruebas de sistema que se han diseñado y automatizado por el equipo de testeo en un periodo de tiempo determinado, en el que por defecto el filtro de tiempo es de un año.

Las barras representan datos independientes, en azul los diseños que se han automatizado ese mes y en naranja los diseños que se han creado. Puede darse el caso que en un mes se haya creado un diseño y al mes siguiente se haya automatizado.

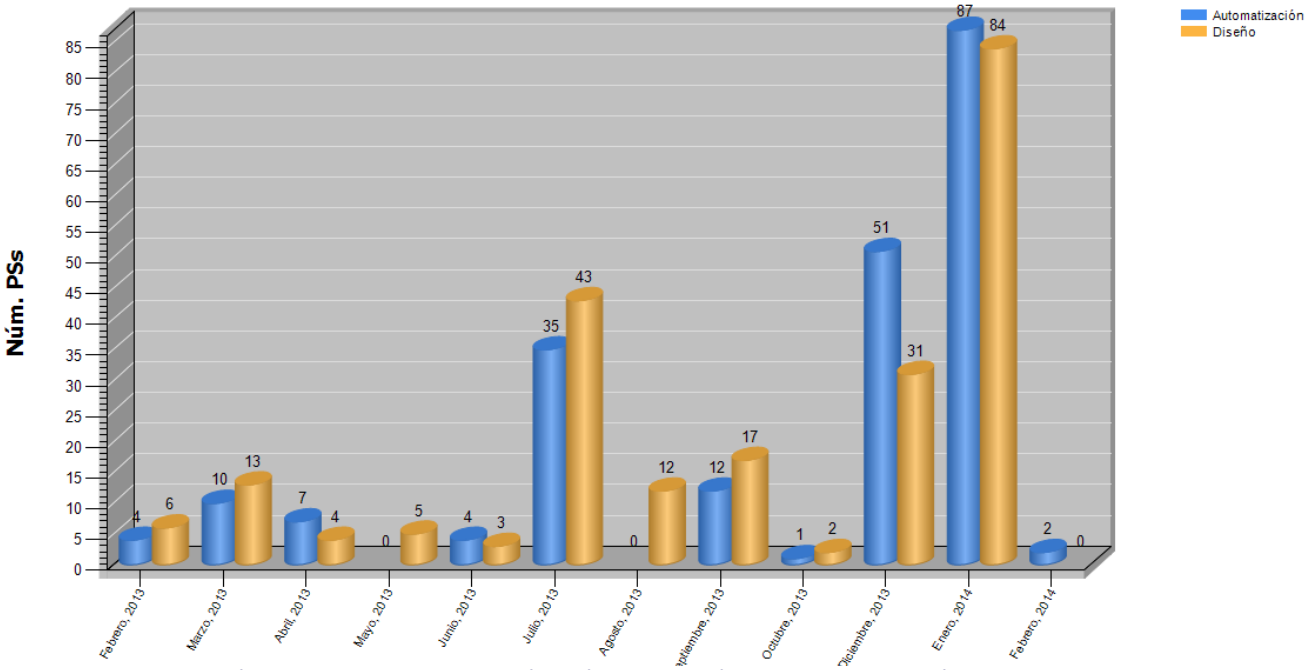

Figura 40: Dashboard 'Historico de PS diseñadas/automatizadas'

4.7 Otros mecanismos para controlar el proceso QA

Además de estos Dashboards que podemos revisar a final de cada Sprint para obtener información de cómo se está desarrollando el proceso QA, el responsable del equipo de automatización manda al finalizar cada Sprint un informe al jefe de desarrollo, con la siguiente información:

- Numero de PS automatizadas
  - Numero de PS automatizadas que han pasado satisfactoriamente
  - Numero de PS automatizadas que no han pasado satisfactoriamente
  - Numero de PS automatizadas que en el anterior Sprint no pasaban satisfactoriamente

- Numero de fallos creados
  - Numero de fallos creados con una ID asociada
  - Numero de fallos creados con una ID asociada y resuelta

Solo cuando el Product Manager recibe este informe y lo considera adecuado, la versión del producto sale a producción. En ocasiones, si queda alguna ID asociada a algún fallo y el jefe de desarrollo considera que tiene que resolverse antes de sacar versión a producción, puede retrasarse el sprint lo necesario para que dicha ID sea resuelta.

5. Conclusiones

Cada vez más las empresas de desarrollo de productos software apuestan por tener bien definido un proceso QA. Dicho proceso es muy variado dependiendo de la metodología de programación utilizada, del producto, o simplemente del tamaño de la empresa por lo que no es algo sencillo y estandarizado. Es por ello que definir un proceso QA es una tarea larga que requiere de una predisposición colaborativa por parte de cada uno de los equipos que conforman una empresa.

Si bien en la empresa donde se realizó este trabajo estaba bastante avanzada en aspectos de QA, no existía una definición global y explicita del todo el proceso ni del apoyo de herramientas. Así pues, este objetivo se ha cumplido con la realización de este trabajo. Además se ha contribuido a impulsar un trabajo continuo de refinamiento y mejora del proceso QA, cuyas principales mejoras han sido las siguientes:

- Se han establecido reuniones mensuales con los Testers Manuales para concretar que Pruebas de Aceptación automatizar.

- Mejora en el rendimiento del lanzador de SUITES.

    - En estos últimos meses hemos duplicado el número de máquinas virtuales para realizar las pruebas de regresión automatizadas lo cual nos permite un número mayor de lanzamientos de las distintas suites para comprobar la existencia de errores en la aplicación antes de que lleguen al cliente.

    - Estabilización de las máquinas virtuales. Para ello se estudió el comportamiento de algunas pruebas y por qué fallaban cuando realmente no se trataba de un error. Se pudo observar que se trataba por el alto consumo de memoria RAM del antivirus, por lo que se tomaron medidas al respecto.

- Adición de una columna en el lanzador de SUITES que nos informaba de cuándo fue la última vez que la prueba automatizada pasaba satisfactoriamente. Esto nos ayudaba muchísimo para localizar aquellas pruebas que de repente comenzaban a fallar, y nos permitía centrarnos en estudiar como afectaban los últimos cambios introducidos por los programadores y si implicaban un fallo en la aplicación. En muchos casos se trataban de fallos ocasionales de la prueba automatizada por lo que bastaba con refactorizarla.

- Depuración del informe enviado al jefe de proyecto al final de cada Sprint.

- Definición de todo el proceso QA de una manera clara y ordenada a través de la realización del trabajo aquí presentado.

- Se ha establecido una reunión semanal en la cual exponíamos como había transcurrido la semana en cuanto a resultados de las pruebas automatizadas se refiere, controlando que no existieran desviaciones muy llamativas en cuanto a resultados satisfactorios, y en caso de ser así, encontrar el motivo.

En cuanto a la experiencia personal después de la realización de este trabajo colaborando con una empresa real ha sido muy satisfactoria y me ha permitido comprender como se conforma un equipo global centrado en que el proceso QA se cumpla, empezando por los analistas-desarrolladores, siguiendo por los Testers Manuales y acabando con el equipo de Automation Tester. Esto unido al uso de metodologías agiles para el desarrollo de software ha contribuido a aplicar conocimientos adquiridos a lo largo de la rama especifica cursada por lo que considero que ha sido una gran experiencia a nivel formativo y profesional.

6. Referencias

[1] Letelier, P. Apuntes de la asignatura Proceso del Software, ETSINF, Universitat Politécnica de València (2015)

[2] Letelier, P. Fallos - Defectos - Errores y su gestión en un contexto ágil, http://agilismoatwork.blogspot.com.es/search/label/Gesti%C3%B3n%20de%20Fallos (2012)

[3] Pressman (1998), "Ingeniería del software, un enfoque práctico"

[4] IBM, Rational Functional Tester, http://www03.ibm.com/software/products/es/functional (2016)

[5] 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).

[6] Company, M. Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación, Tesis del Master, DSIC – Universitat Politécnica de València (2011).

[6] Del Rincon Lopez, A. Dashboard de testo para entornos con pruebas automatizadas, Trabajo Fin de Grado, ETSINF - Universitat Politécnica de València (2015)

[7] Portal ISO 25000, http://iso25000.com/ (2016)

[8] Manuales en OneNote sobre las distintas herramientas de la empresa en la cual se realizó este TFG.