Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Gestión de Pruebas de Aceptación integrada en una metodología y herramienta para la gestión ágil de proyectos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Marc Sebastià Pérez-Serrano Martínez

Tutor: Patricio Letelier Torres

Curso 2019-2020

Resumen

Especificar los requisitos como Pruebas de Aceptación (PA) es una estrategia que ha demostrado ser altamente eficaz para la validación e implementación de requisitos del software. Las PA y *mockups* (como complemento cuando corresponda) permiten al cliente y al equipo de desarrollo especificar y validar el comportamiento que se espera del producto software. Posteriormente, las PA establecen el alcance del trabajo de programación y, finalmente, guían el trabajo de las pruebas. Las PA son claves en toda metodología de desarrollo de software, pero lo son más en las metodologías ágiles, pues dentro del minimalismo de artefactos utilizados en una metodología ágil las PA deben estar presentes.

La metodología ágil TUNE-UP Process y su herramienta Worki siguen esta estrategia de especificación de requisitos basada en PA. En la versión desktop de Worki se realizaba una gestión bastante elaborada respecto de PA, sin embargo, en la nueva versión Web aún no se implementan gran parte de esas funcionalidades y, además, será necesario replantearlas según el nuevo diseño funcional de la aplicación. En este TFG se realizará dicho trabajo de migración y mejora de funcionalidad asociada a la gestión de PA.

El objetivo de este TFG es dotar a Worki de una gestión sencilla y potente de PA. El TFG se centrará en el desarrollo de la parte front-end de la aplicación utilizando Angular y varias librerías específicas. La validación de las funcionalidades implementadas se hará en una asignatura de la rama de Ingeniería del Software durante el presente curso académico.

Palabras clave: gestión ágil de proyectos, pruebas de aceptación.

Abstract

Acceptance Tests (AT) have proved to be a highly efficient strategy to guarantee the validity and safe implementation of software requisites. AT as well as mockups (employed as a complement whenever necessary) allow specifying and validating software's behavior by both the client and the development team. Not only that, but AT also establish a strong foundation upon which the programming and testing tasks are carried; therefore, AT should be considered a key artifact in every software development methodology, and its value exponentially increases in agile methodologies considering the minimalism of the artifacts they use.

TUNE-UP Process agile methodology and Worki —its tool— follow the "requisite specification based in AT" strategy. Worki provided a well-designed AT management in its Desktop application; however, the newer web application did not implement any of those features yet and, besides, it was necessary reconsidering them entirely in favor of the emerging functional design model that the web application had adopted. This thesis consists of migrating and improving the AT management features available in the Desktop version.

The final objective is to provide Worki with a simple yet powerful AT management system and focuses on the front-end development using the Angular framework alongside other UI libraries. Features to be implemented have proven viable during the current academic course in one of the subjects of the Software Engineering branch.

Keywords: agile project management, acceptance testing.

Dedicatoria

Dedicado a mis padres por haberme dado no una, sino dos vidas.

Agradecimientos

Estos últimos meses han sido especialmente confusos para mí. Por ello, me gustaría dar las gracias a mi tutor, Patricio Letelier Torres, por haberme infundido la determinación necesaria para terminar este trabajo y por haberme dado la oportunidad de trabajar con él y su equipo, descubriéndome la pasión que hoy en día siento por las tecnologías web.

Gracias a mis padres y a mis hermanos por veinticinco años de amor incondicional.

Gracias a mi novia por mudarse a mi lado durante este verano, cerciorándose de que comiese bien y durmiese bien cuando más lo necesitaba y a sus perritas, Gala y Lupita, por quedarse junto a mí en el sofá cada fin de semana, hasta altas horas de la madrugada, mientras estudiaba para los exámenes del grado y trabajaba en esta memoria.

Índice de contenidos

Contenido

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación

En el contexto del desarrollo de software, la ingeniería de requisitos es una actividad compleja, principalmente por falta de entendimiento entre el cliente y el equipo de desarrollo. Este problema se ve especialmente agravado en metodologías tradicionales, en las cuales los requisitos se suelen especificar en detalle muy tempranamente respecto del momento de su implementación.

La ingeniería de requisitos comprende múltiples técnicas y artefactos para establecer el contrato cliente-desarrollador [1]:

- Listados de requisitos.
- Modelos de casos de uso, diagramas de secuencia, modelos de domino, etc.
- Historias de usuario.
- Prototipos/Bocetos de IU.
- Descripción narrativa.
- Lenguaje formal.
- Otros.

Algunos de los artefactos citados son difíciles de gestionar cuando los requisitos son muy numerosos y se producen cambios (listados de requisitos, modelos de casos de uso, diagramas de secuencia, etc.), otros establecen un contrato demasiado subjetivo o abierto a interpretaciones (descripción narrativa) y otros dificultan la participación del cliente en la negociación (lenguaje formal).

Las Pruebas de Aceptación (PA) son el hilo conductor a la hora de validar si un requisito del producto se cumple o no. Las PA se aplican sobre una funcionalidad del producto para validar que ésta hace lo que el usuario final espera que haga.

Existen técnicas y procesos asentados que incluyen artefactos de pruebas en etapas anteriores a la etapa de pruebas, como lo son el Test-Driven Development [2] (TDD) y el Behavior-Driven Development [3] (BDD).

TUNE-UP Process, una metodología de desarrollo ágil, y Worki, su herramienta de apoyo, van un paso más allá y proponen el uso de PA como guía en la especificación de requisitos. TUNE-UP Process y Worki han sido desarrolladas en el Departamento de Sistemas Informáticos y Computación (DSIC) de la Universidad Politécnica de Valencia y son utilizadas en asignaturas de la Rama de Ingeniería del Software en el Grado en Ingeniería Informática impartido en la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politécnica de Valencia (UPV) y por algunas empresas. Anteriormente, Worki se distribuía como una aplicación para escritorio y es hoy en día distribuida como una aplicación web. En el momento de inicio de este trabajo, Worki no cuenta con un sistema de gestión de PA.

Dado que una PA o un conjunto de PA nos sirven para validar un requisito, también pueden servirnos para especificar un requisito. Esta aproximación recibe el nombre de Test-Driven Requirements Engineering (TDRE) [4] y convierte a las PA en no sólo el hilo conductor a la hora de validar los requisitos, sino también a la hora de especificarlos y, por tanto, en el hilo conductor del desarrollo.

El uso de las PA como artefactos durante la ingeniería de requisitos es interesante porque [1]:

- Se expresan en lenguaje natural, lo que facilita el entendimiento del cliente.
- Permiten ser descritas como un conjunto de:
  - → Condiciones.
  - → Pasos.
  - → Resultado esperado.
  - → Observaciones.

  Estas descripciones facilitan el entendimiento y cumplimiento por parte del desarrollador.
- Agrupadas correctamente de acuerdo con la estructura del producto, son fácilmente navegables y escalables.
- Una PA es independiente y reutilizable.

Como ejemplo, extraído de [1], supongamos el requisito “Retirar dinero” en el contexto de un cajero automático. El conjunto de PA que especificaría y más tarde validaría dicho requisito podría ser:

- Reintegro usando cantidades predefinidas habilitadas.
- Reintegro con cantidad introducida por el cliente.
- Reintegro saldo < cantidad.
- Cancelación de operación.
- No disponibilidad de billetes.
- No disponibilidad de papel para recibo.
- Reintegro saldo < cantidad.
- Excedido tiempo de comunicación con sistema central.
- Aviso de operaciones internas del cajero.
- Excedido tiempo de espera para introducción de acción.

Además, TUNE-UP Process incita a la definición de una PA a través de:

- Un grupo de condiciones iniciales.
- Una secuencia de pasos.
- El resultado esperado.
- Un grupo de observaciones, si las hubiese.

Considerando que las PA de una aplicación pueden ser muchas y estar sometidas a cambio y evolución constante, es necesario contar con herramientas que faciliten su gestión.

1.2. Objetivos

El objetivo de este trabajo es añadir funcionalidad para la gestión de PA en la herramienta de gestión ágil de proyectos Worki.

Las funcionalidades principales que deben desarrollarse son:

- Agrupar las PA en función de los requisitos que especifican y validan.
- Consultar los cambios que sufren las PA a lo largo del tiempo.
- Reutilizar las PA.
- Gestionar el estado de ejecución de las PA.

Estas características se integrarán en la herramienta Worki y se validarán a lo largo del curso académico 2019-2020 en dos de las asignaturas de la rama de Ingeniería del Software del Grado en Ingeniería Informática impartido en la ETSINF de la UPV: PSW (Proceso del Software) y PIN (Proyecto de Ingeniería del Software).

1.3. Estructura del trabajo

El primer capítulo justifica la necesidad de este trabajo y marca las metas finales del mismo.

En el segundo capítulo se estudia el estado del arte: en él, se analizan y comparan las soluciones propuestas para la gestión de PA por otras herramientas para gestión ágil de proyectos y se exploran posibles defectos o insuficiencias que puedan aprovecharse para mejorar nuestra propuesta.

El tercer capítulo recoge información sobre las tecnologías utilizadas para entender el contexto en el que se ha desarrollado la solución.

El cuarto capítulo expone la solución implementada y profundiza en el diseño (a través de modelos, *mockups* y PA) y en el desarrollo.

El quinto capítulo recoge las conclusiones extraídas de la aplicación de la solución y su validación a lo largo del curso académico 2019-2020 en dos asignaturas del grado de Ingeniería Informática.

Posteriormente se indican las referencias y, finalmente, el anexo contiene un manual de usuario que expone cómo hacer uso de la solución desarrollada para Worki.

2. Gestión de PA en herramientas para la gestión ágil de proyectos

Las características principales que buscamos en una herramienta de gestión ágil de proyectos, como adelantábamos en el capítulo anterior, son:

- Gestionar PA.
- Acceder a un historial de cambios por los que ha pasado la PA.
- Agrupar las PA de acuerdo con una estructura del producto en la cual cada nodo represente los requisitos del producto y podamos visualizar qué UT[1] y qué PA afectan a dicho nodo.
- Marcar OK y KO y visualizar el estado de aplicación (o ejecución) de las PA.
- Pasar PA en regresión (reutilizarlas).
- Visualizar las PA y sus ejecuciones en el contexto del producto completo y/o de un Sprint.
- Obtener gráficas que nos permitan ver el estado de ejecución de las PA respecto al producto/Sprint (una buena forma de controlar el estado de completitud del producto/Sprint).

De entre las opciones disponibles en el mercado hemos elegido tres herramientas de gestión ágil de proyectos que analizaremos durante las siguientes secciones.

[1] Una Unidad de Trabajo (UT) es un elemento de trabajo que corresponde a un cambio en el producto u otra tarea asociada al contexto del trabajo.

2.1. Jira

Jira[2] es un software desarrollado por Atalassian[3]. Cuenta con variedad de opciones de configuración y un catálogo de *plug-ins* (gratuitos y de pago) desarrollados por la comunidad o por empresas externas. Ofrece dos configuraciones iniciales: la clásica y la de nueva generación, para cada una de los cuales se puede optar por una metodología *Kanban* o *Scrum*. Es un servicio de pago, pero cuenta con una fase de pruebas de 30 días. Por defecto, no ofrece ninguna solución para gestión de PA.

En esta sección, veremos hasta dónde podemos llegar con las opciones de configuración que nos ofrece. Para empezar, partimos de un proyecto clásico que hemos llenado con un total de 10 tareas. El resultado de la creación de dichas tareas se observa en la Figura 1.

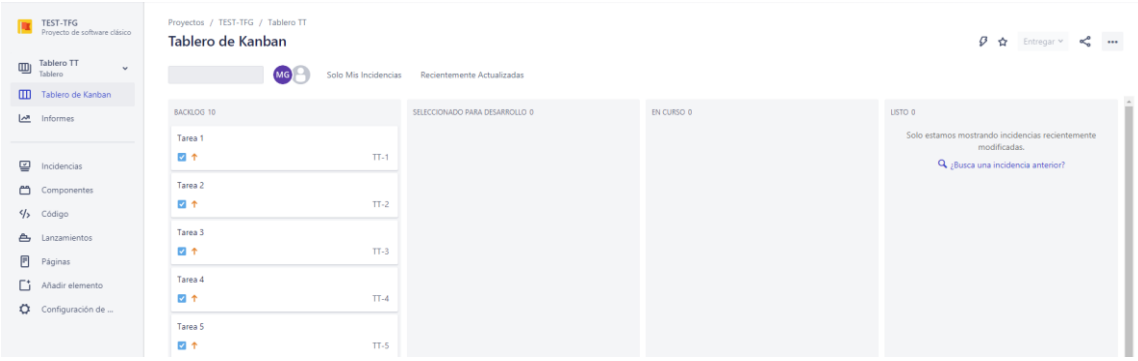

Figura 1. Kanban inicial

Para modelar la gestión de las PA a través de la configuración de Jira, tenemos dos opciones:

- Modelarlas como un tipo de incidencia.
- Modelarlas como un tipo de subtareas.

Parece más conveniente la segunda opción, pues nos interesa de algún modo contener las PA en el contexto de una incidencia. Para ello, debemos acceder a la gestión de incidencias y crear un nuevo tipo de subtarea, como se ilustra en la Figura 2.

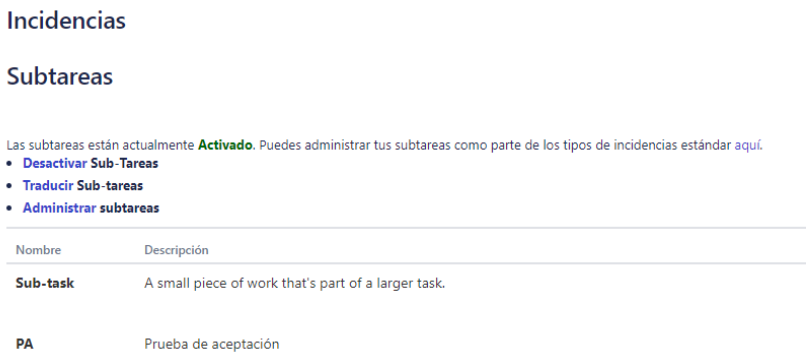

Figura 2. Configuración de incidencias (subtareas)

[2] https://www.atlassian.com/software/jira
[3] https://www.atlassian.com/es

Una vez creado el nuevo tipo de incidencia de subtarea, accederemos al esquema de tipos de incidencias del proyecto creado para añadir la nueva subtarea a la lista de tipos de incidencias disponibles, como se muestra en la Figura 3.

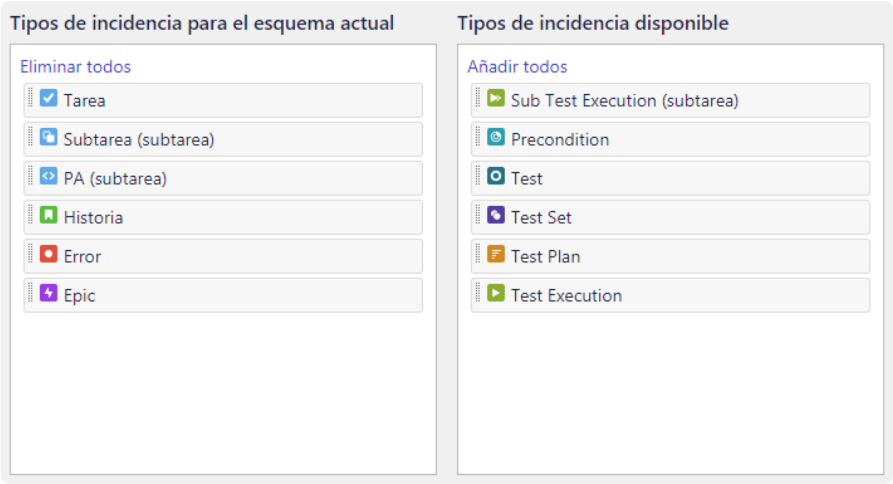

Figura 3. Gestor de incidencias del proyecto

Tras ello, estamos listos para acceder a una de las tareas anteriormente creadas y añadir nuevas PA, como se ilustra en Figura 4.

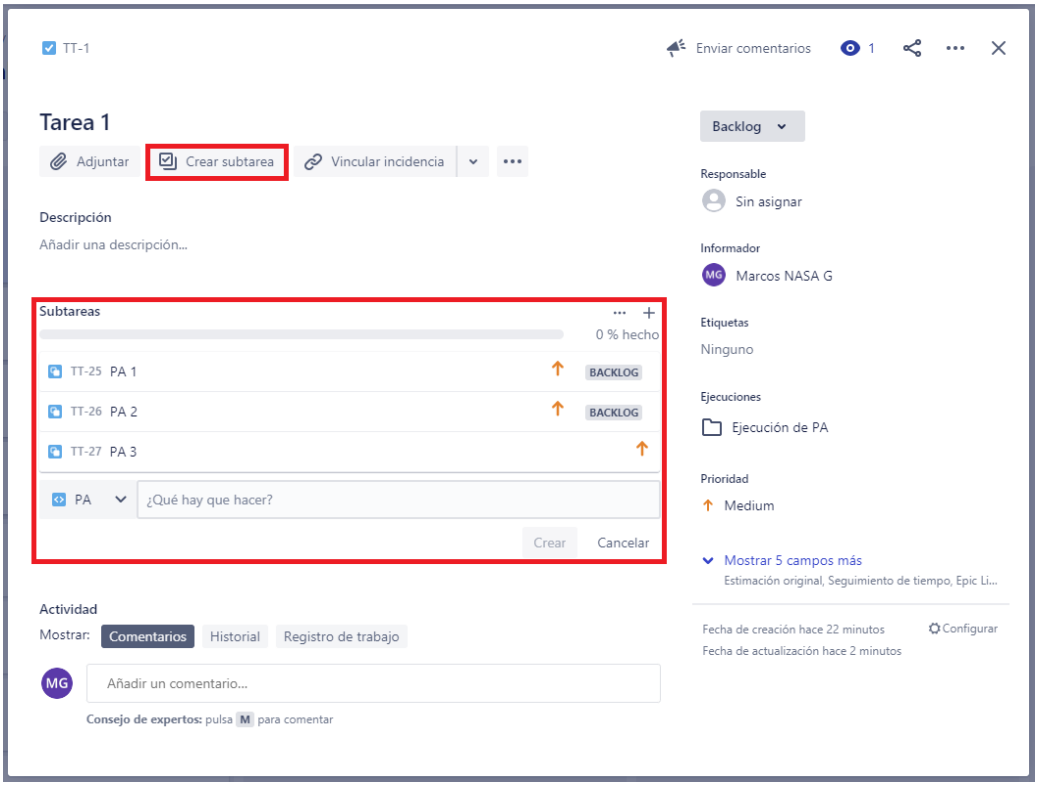

Figura 4. Añadir subtarea PA a incidencia

La personalización de la PA es lo suficientemente completa como para permitirnos añadir una nueva "pestaña" con un campo para marcar el estado de ejecución. El acceso a dicha "pestaña" se ilustra en la Figura 5.

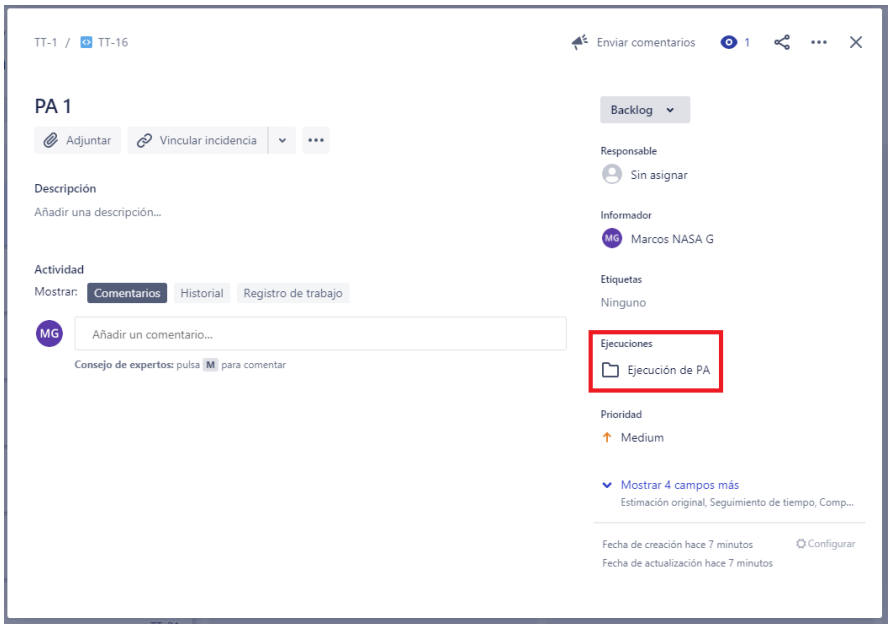

Figura 5. Subtarea PA

El marcado del estado la ejecución de PA como OK o KO se realiza a través de un menú desplegable, como se muestra en la Figura 6.

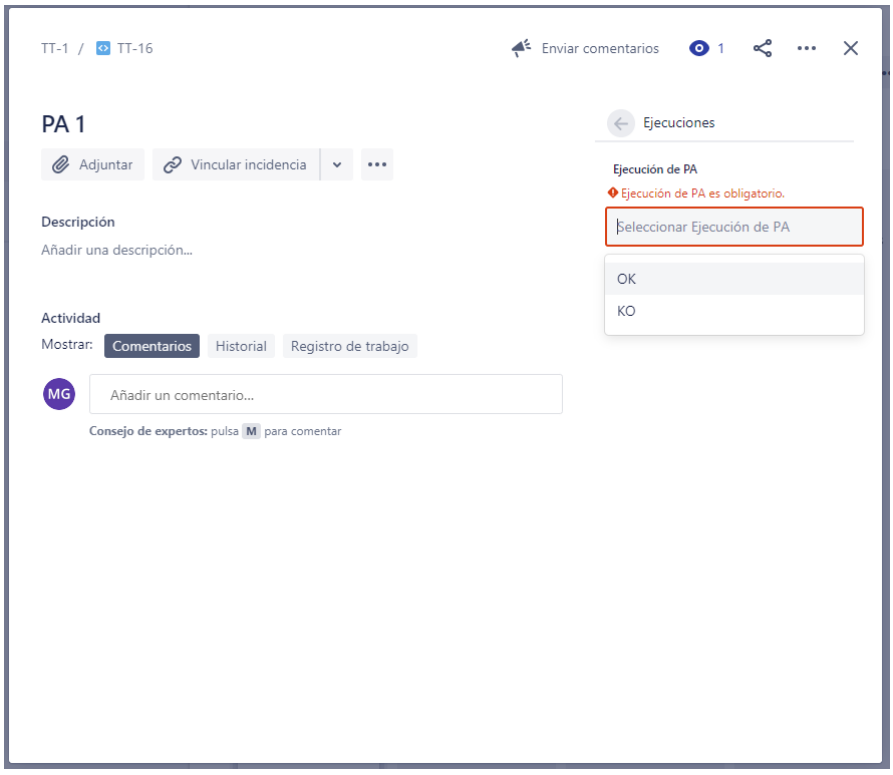

Figura 6. Ejecución PA

Toda incidencia de Jira cuenta con un historial de cambios que nos permite seguir el curso de los cambios que ha sufrido. En el caso de una PA, nos permite visualizar los cambios en su título y su descripción y, aunque no de forma idónea ya que se encuentran diluido entre el resto de cambios, los OK y los KO. La Figura 7 muestra el historial de actividad de la PA 1.

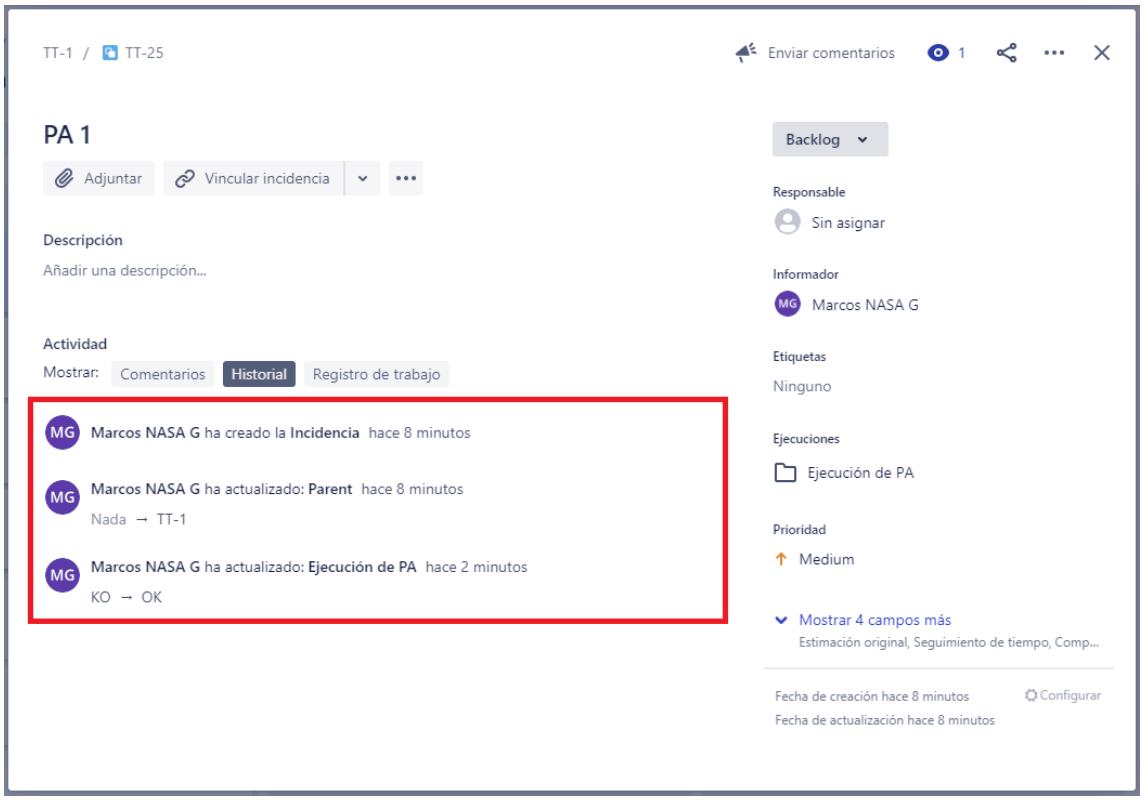

Figura 7. Historial de cambios PA

Un efecto secundario no deseado del modelado de PA como subtareas es que llena nuestro tablero Kanban de nuevas fichas correspondientes a las subtareas. Esto hace que el tablero sea más complejo de gestionar conforme aumenta el número de PA, como se observa en la Figura 8.

Figura 8. Tablero Kanban con PA

Además, la lista de subtareas asociadas a una incidencia no se desplazan junto a su tarea contenedora al avanzar ésta en el flujo de trabajo, como se muestra en la Figura 9.

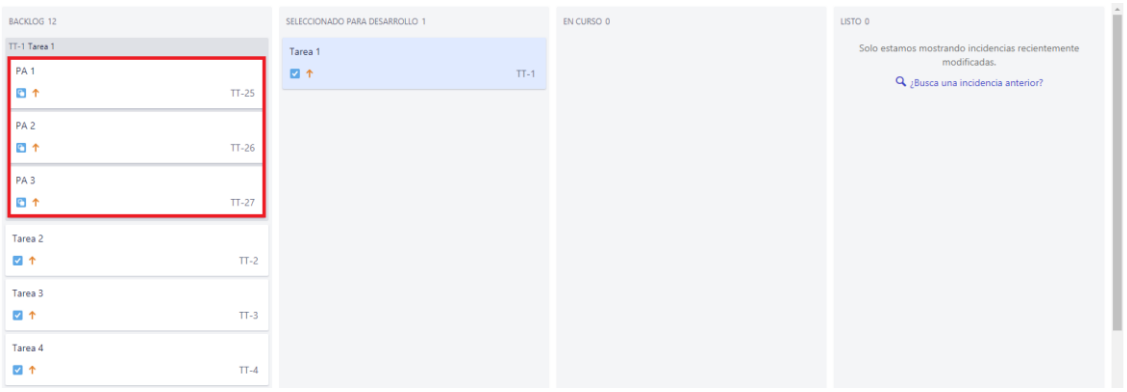

Figura 9. Las subtareas no se desplazan por el flujo de trabajo con la incidencia a la que están asociadas

Por otra parte, ¿qué opciones tenemos para explotar los datos relativos a nuestras PA?

Nos ha sido posible crear un gráfico que lleve la cuenta del número de PA activas, como se muestra en la Figura 10. Sin embargo, no hemos conseguido nada parecido a agrupar las PA según la estructura del producto.

Figura 10. Gráfica de incidencias de tipo PA activas

Pese a que las opciones de configuración prometen sobre el papel, esta solución para la gestión de PA presenta varios inconvenientes vistos durante la sección que son suficiente como para no considerarla viable.

En la siguiente sección veremos qué puede ofrecernos Jira a través de sus *plug-ins*.

2.2. XRAY Test Management

XRAY Test Management[4] es un *plug-in* para Jira desarrollado por Xpand IT[5]. Nos ofrece un catálogo de nuevas incidencias y un nuevo conjunto de vistas agrupadas bajo una nueva opción de menú: el Tablero de Testing. De dicho catálogo nos interesan dos: la prueba (de nombre Test) y la ejecución de prueba (de nombre Test Execution).

La elección entre instalar la aplicación en un proyecto clásico o de nueva generación es trivial; pero no porque uno de ellos se ajuste mejor que el otro a nuestras necesidades, sino porque ninguno de ellos parece ideal:

- Para los proyectos clásicos, ofrece un catálogo de nuevas incidencias cuya *naturaleza* (historia, tarea, error, subtarea…) no es configurable.
  - → La incidencia Test es de tipo tarea. Sería conveniente para nosotros modelarlo como subtareas.
  - → La incidencia Test Execution es una tarea, pero se ofrece también en forma de subtarea bajo el nombre Sub Test Execution. Esto sí se ajusta a lo que buscamos.
- Para los proyectos de nueva generación, permite asociar las nuevas incidencias al tipo de incidencia que nos interese. Pero los proyectos de nueva generación no pueden configurarse para incorporar nuevos tipos de subtarea, por lo que terminamos con un escenario idéntico al anterior:
  - → La incidencia Test sería una tarea.
  - → La incidencia Test Execution sería una subtarea.

Ya que no vamos a ser capaces de modelar un escenario ideal, por consistencia, elegiremos un Kanban clásico y lo iniciaremos con 10 tareas, igual que habíamos hecho en la sección anterior.

Tras añadir XRAY al proyecto, como se nos había prometido, tenemos una nueva opción disponible en el menú: el Tablero de Testing. Hablaremos brevemente de él más adelante.

[4] https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira
[5] https://www.xpand-it.com/

La única configuración que nos ha resultado necesario hacer una vez se ha añadido XRAY al proyecto se muestra en la Figura 11, donde hemos incluido los tipos de incidencias que nos interesan del catálogo que ofrece:

— Test, una nueva tarea.
— Sub Test Execution, una nueva subtarea.

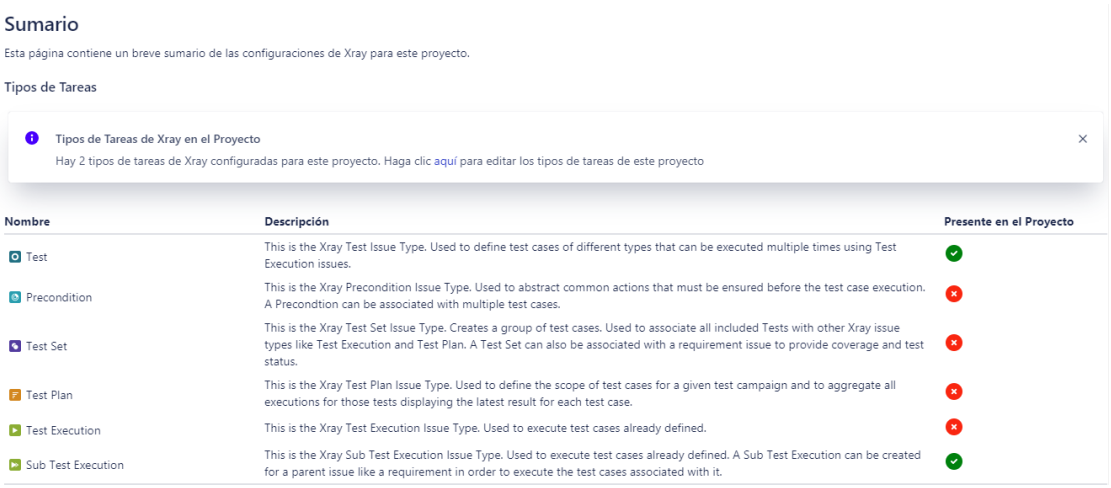

Figura 11. Incidencias de XRAY habilitadas

Añadiremos algunas incidencias de tipo Test al proyecto.

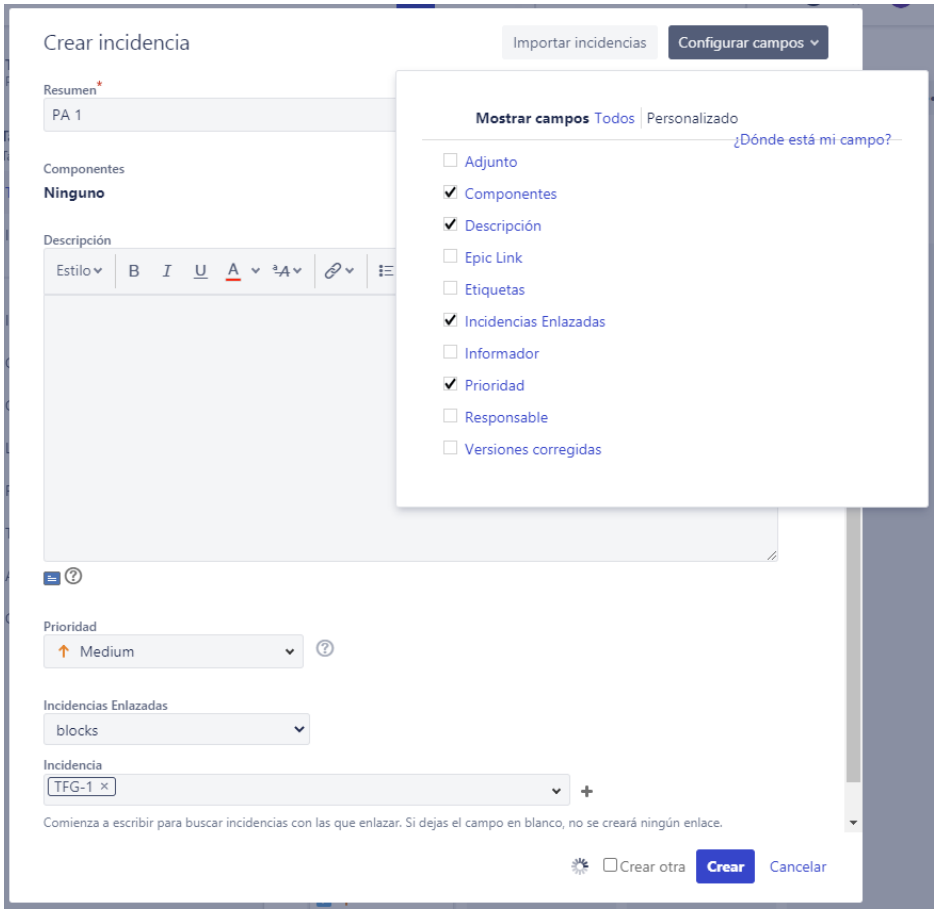

Figura 12. Creación de incidencias de tipo Test

La Figura 12 muestra la creación de una incidencia de tipo Test que bloquea a la incidencia de código TFG-1. La incidencia de tipo Test, por defecto, cuenta con múltiples campos adicionales que no se muestran para reducir y simplificar el tamaño de las imágenes. En total han sido creadas 5 incidencias de tipo Test:

— PA 1: bloquea a TFG-1.
— PA 2: bloquea a TFG-1.
— PA 3: bloquea a TFG-2.
— PA 4: bloquea a TFG-2.
— PA 5: bloquea a TFG-3.

La ficha de una de dichas incidencias se muestra en la Figura 13.

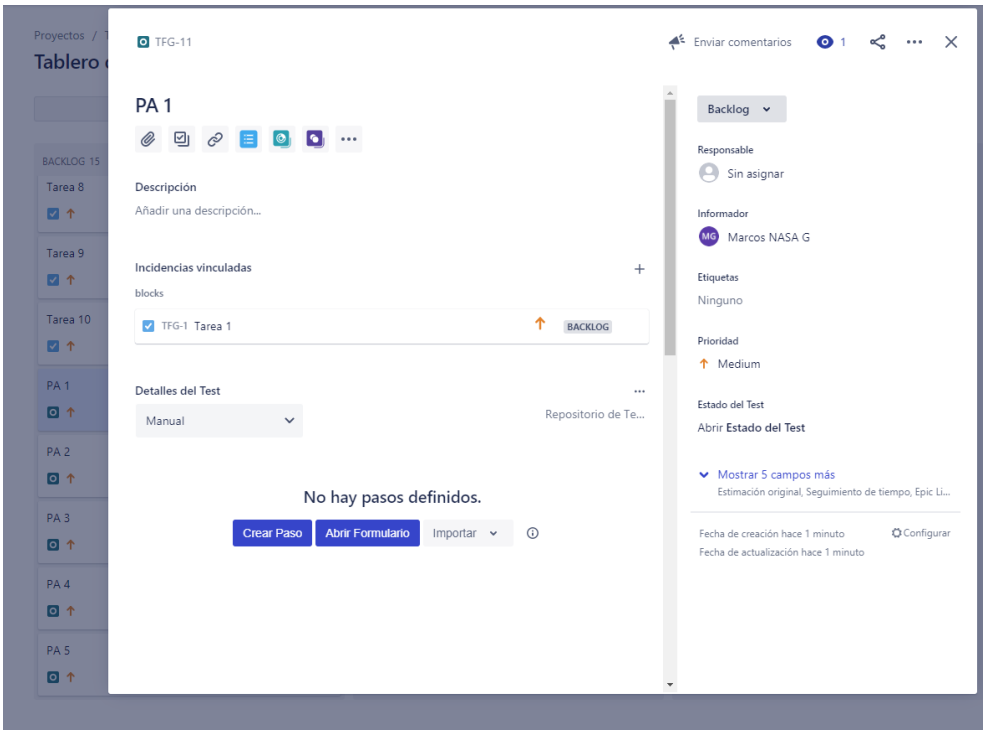

Figura 13. Ficha de "PA 1", una incidencia de tipo Test

El siguiente paso será ejecutar la PA 1 que se muestra en la Figura 14. Para ello, todo lo necesario es crear una subtarea de tipo Sub Test Execution desde la propia ficha de PA 1.

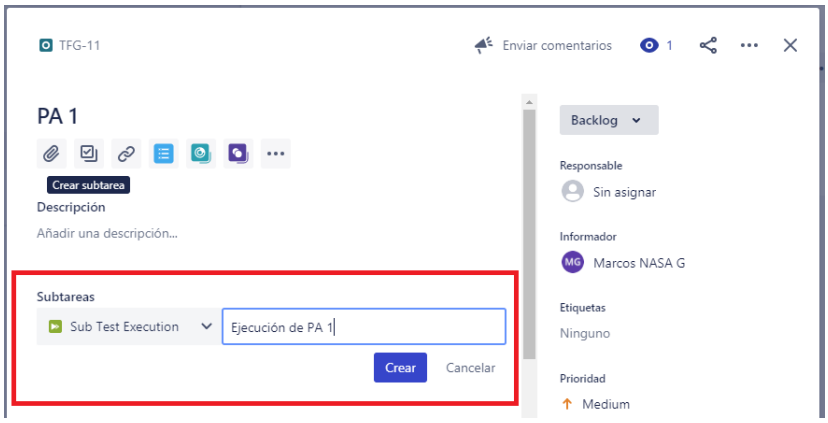

Figura 14. Crear ejecución de PA, subtarea de tipo Sub Test Execution

La aproximación que XRAY nos ofrece es la de crear subtareas de tipo ejecución (Sub Test Execution) para, desde su ficha, crear ejecuciones comprendidas en ella. Parece contraintuitivo, pero así es: una vez creada una ejecución de PA, debemos añadir ejecuciones a través del botón "Crear Test" que se muestra en la Figura 15.

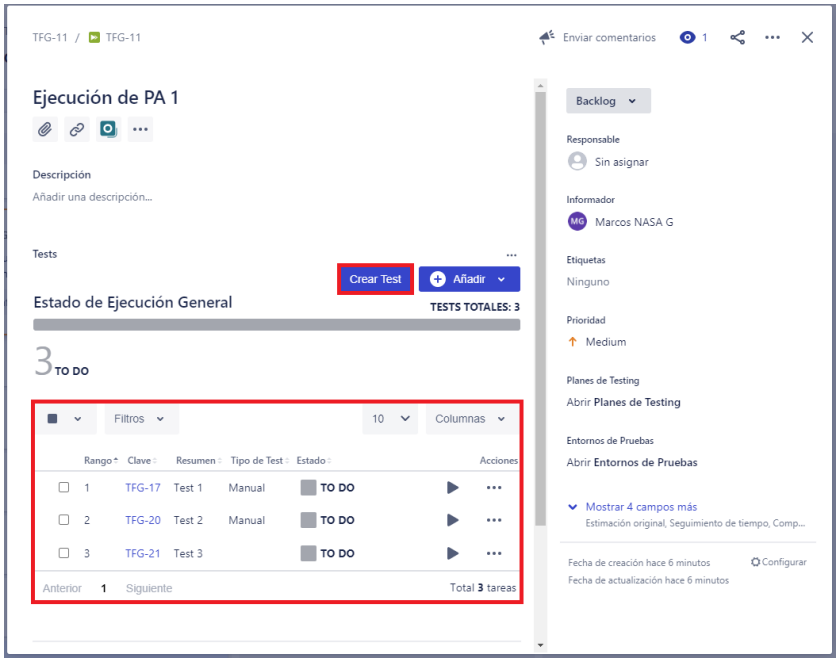

Figura 15. Tests de ejecución de PA

Teniendo en cuenta lo mencionado en el párrafo anterior, nos cuestionamos si hubiese sido mejor opción saltarse la creación de incidencias de tipo Test (PA) en favor de enlazar subtareas de tipo Sub Execution Test (Ejecuciones de PA) directamente a las incidencias de tipo tarea. Pero saltarse el paso de crear una PA hace que la solución pierda contenido semántico.

Tras ya varios pasos, hemos logrado crear tres instancias (Tests) de una ejecución de PA, como se muestra en la Figura 15. Al acceder a cada una de dichas instancias, podemos por fin brindar detalles sobre el estado de ejecución. Desde esta vista, esta vez sí, es posible marcar una PA con un OK (verde), KO (rojo) o en ejecución (amarillo), como se ilustra en la Figura 16.

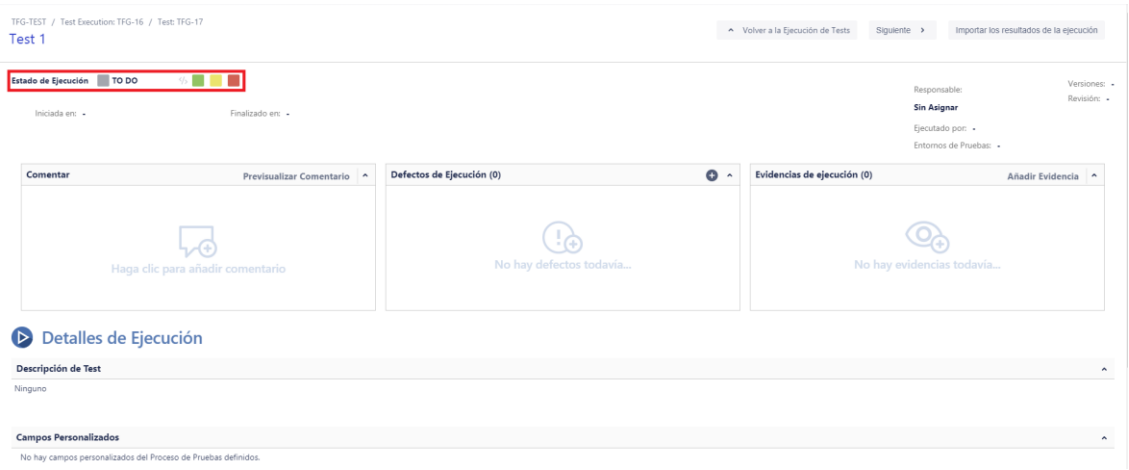

Figura 16. Ficha de un test (ejecución) de ejecución de PA

De vuelta a la ficha de la ejecución de PA tras haber dado por válida la prueba, podemos consultar el estado general de la ejecución de la PA, como se muestra en la Figura 17.

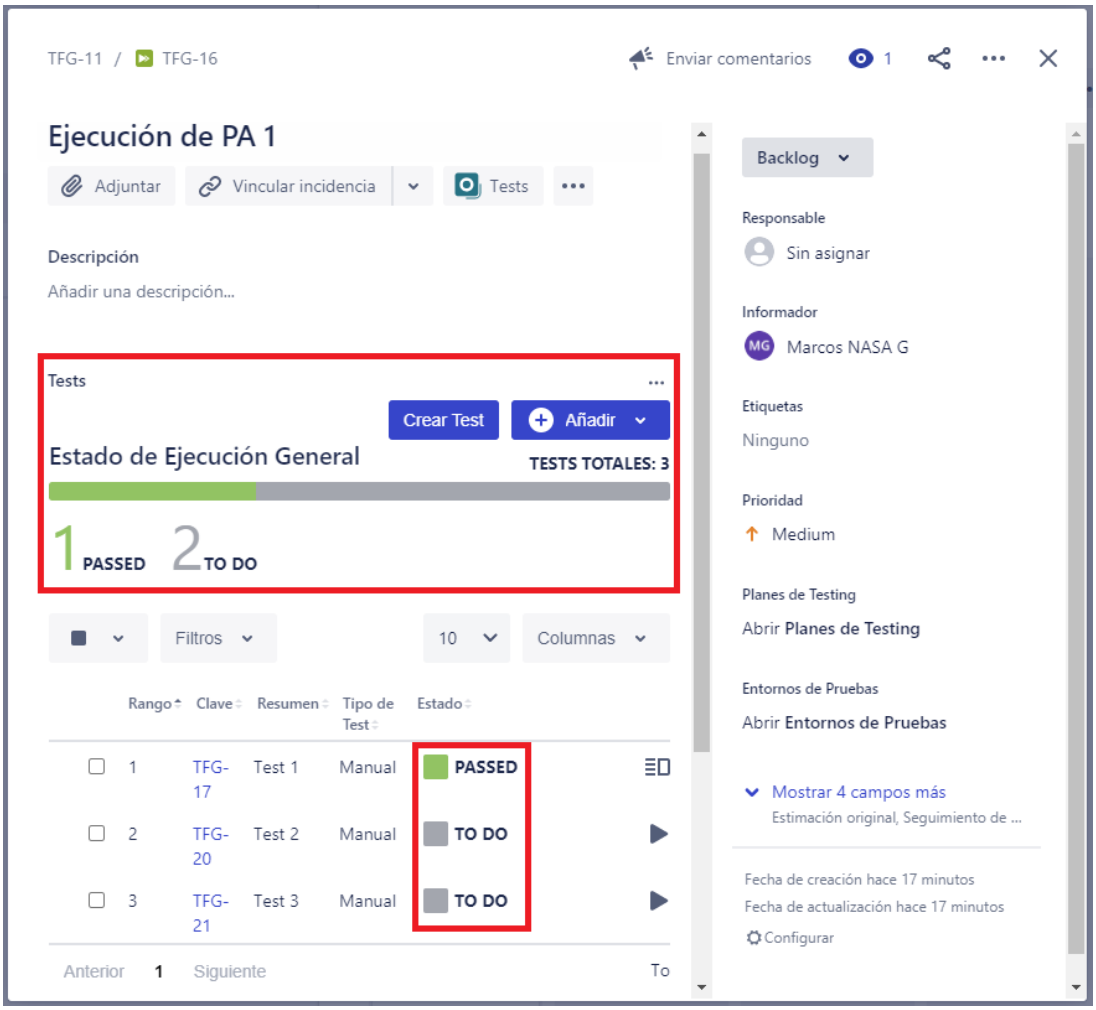

Figura 17. Estado de ejecución general de la ejecución de la PA

En cuanto a opciones de visualización, el Tablero de Testing con el que XRAY dota a Jira contiene algo similar a lo que buscamos: es posible crear una estructura en árbol para agrupar tests que convenientemente podemos modelar en función de los requisitos del producto. Hemos creado una estructura como ejemplo, mostrada en la Figura 18.

Figura 18. Vista en árbol representando la estructura de un producto con PA asociadas

Por último, deseamos poder ver todas las PA y todas las ejecuciones de PA del proyecto. Desde el Tablero de Testing tenemos acceso a estas dos vistas, las cuales ofrecen la posibilidad de generar informes como una lista de todas las PA y de todas las ejecuciones, respectivamente. Dicho menú se ilustra en la Figura 19.

Figura 19. Opciones del Tablero de Testing

2.3. Pivotal Tracker

Pivotal Tracker[6] es, a simple vista, la menos completa y configurable de las tres herramientas analizadas. Está orientado a *Scrum*. Ofrece una versión gratuita para hasta 5 colaboradores y precios adaptados al tamaño y tipología de la empresa a partir de dicha cifra.

Esta herramienta se queda atrás en cuanto a configuración respecto al resto de herramientas. Sin embargo, cuenta con un concepto interesante destinado a la definición de criterios de aceptación: las *reviews*. Las *reviews* permiten a un equipo colaborar simultáneamente en la aceptación de una historia, representando el progreso y las contribuciones de cada uno de los colaboradores en la historia [5].

Durante esta sección, modelaremos las PA como un nuevo tipo de *review* y valoraremos cuán bien se ajusta a lo que buscamos. Para empezar, crearemos un nuevo proyecto. El diálogo de creación se muestra en la Figura 20.

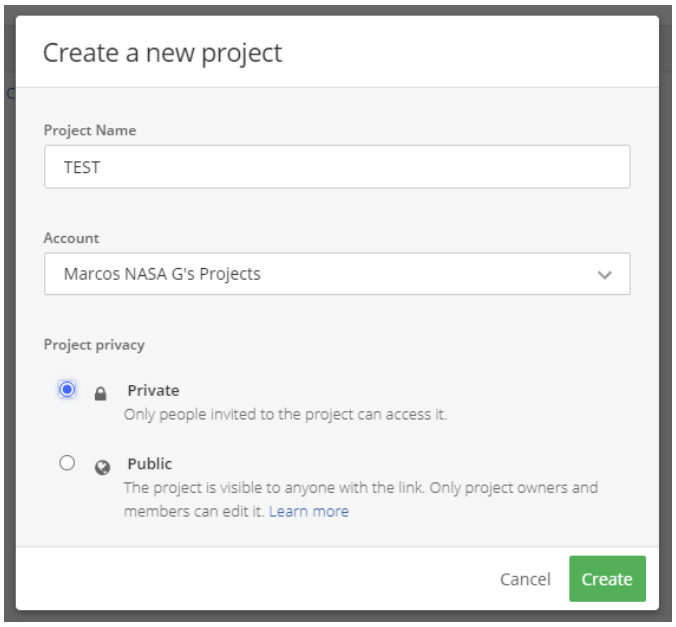

Figura 20. Creación de un nuevo proyecto

[6] https://www.pivotaltracker.com/dashboard

Como primer y único paso de configuración del proyecto, accederemos al panel de gestión del proyecto para crear un nuevo tipo de *review* llamado “PA”, ilustrado en la Figura 21.

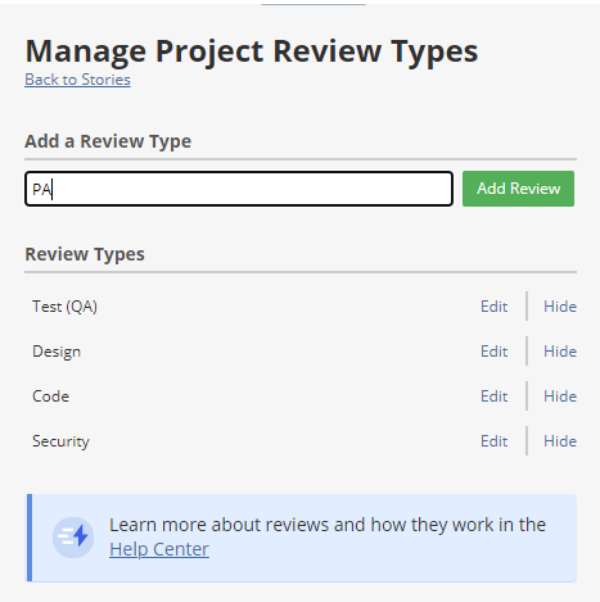

Figura 21. Creación de un nuevo tipo de review llamado "PA"

Tal y como hicimos en las secciones anteriores, añadiremos un total de 10 tareas. Dado que la herramienta se refiere a ellas como historias (*stories*), emplearemos dicha terminología de aquí en adelante durante esta sección. La Figura 22 muestra el estado del tablero tras la creación de las 10 tareas.

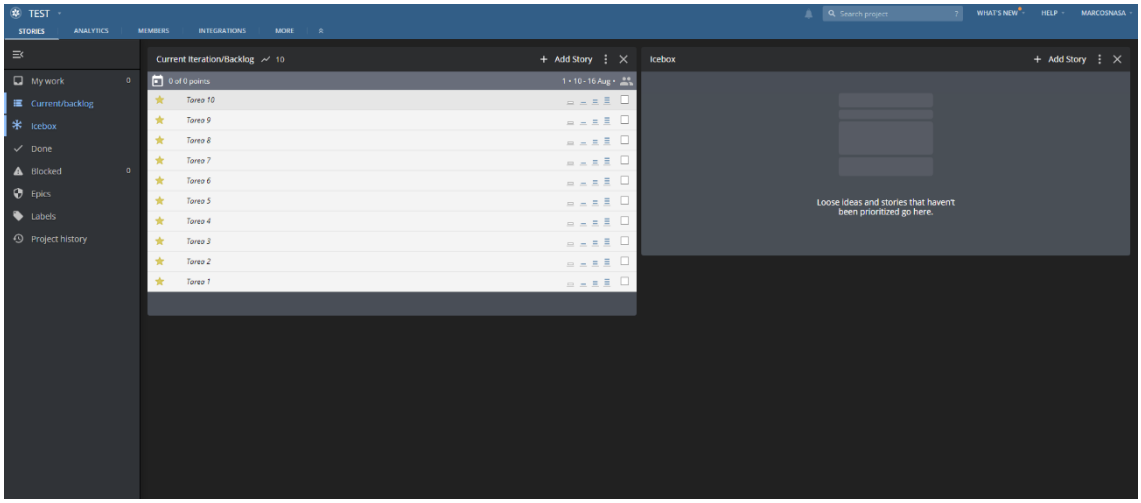

Figura 22. Proyecto de Pivotal Tracker con 10 historias

La gestión de *reviews* se realiza desde el interior de la ficha de una historia, como se muestra en la Figura 23.

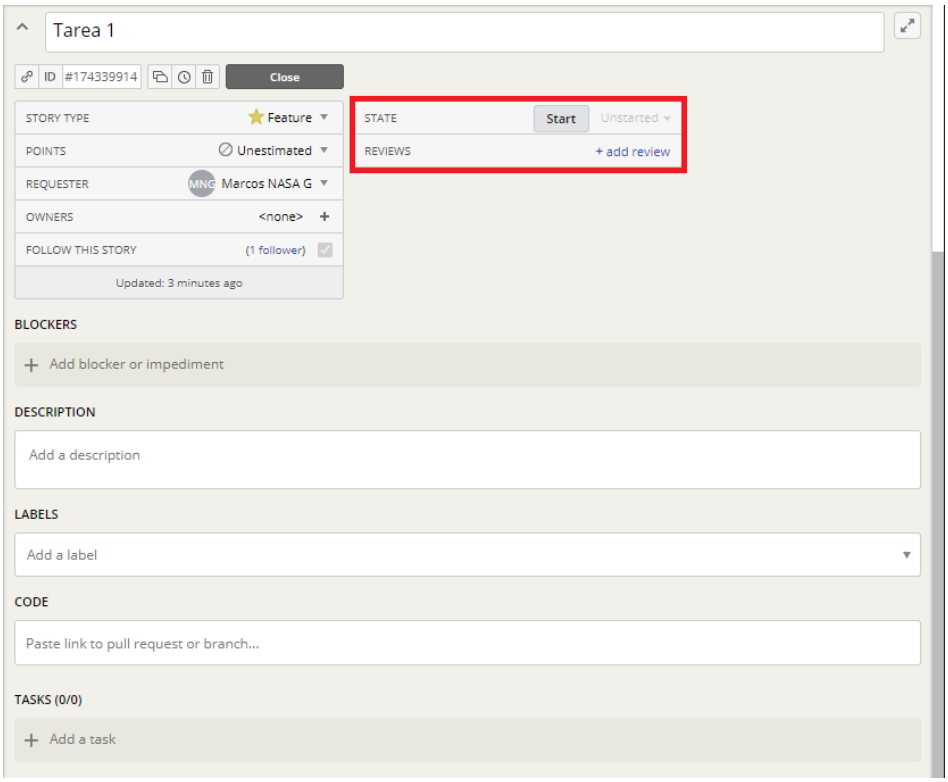

Figura 23. Ficha de la historia "Tarea 1". Se ha marcado en rojo la sección de gestión de reviews de la historia

Haciendo clic sobre "add review" (ilustrado en la Figura 23), se nos permiten añadir nuevas *reviews*. Al hacerlo (Figura 24), lo único que podemos seleccionar es su tipo. No hay forma de darle un nombre ni una descripción. La única opción para tener una gestión más significativa de las PA sería generar cada una de ellas como tipos de *reviews*, lo cual no haremos durante esta investigación.

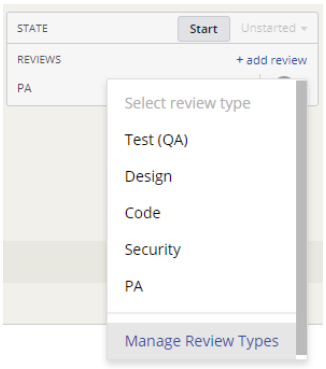

Figura 24. Añadir una review de tipo PA a una historia

Una vez añadidas, sólo es posible asignar un encargado o un resultado a la *review* de tipo PA. Para ello, Pivotal Tracker ofrece una lista desplegable con los valores "pass" (aprobado), "revise" (por revisar) y "in review" (en revisión), como se observar en la Figura 25.

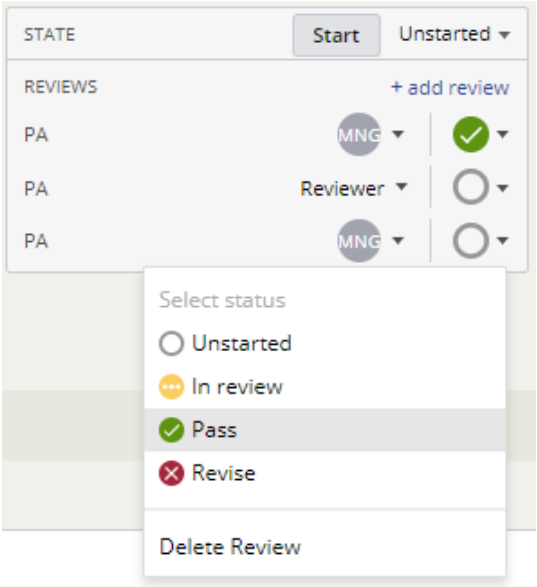

Figura 25. Cambiar el estado a la review de tipo PA

Pivotal Tracker nos ofrece la posibilidad de ver el historial del proyecto, en el cual podemos observar cómo, diluidos entre el resto de acciones, aparecen también los cambios en el estado de las *review*. La Figura 26 ilustra dicho historial.

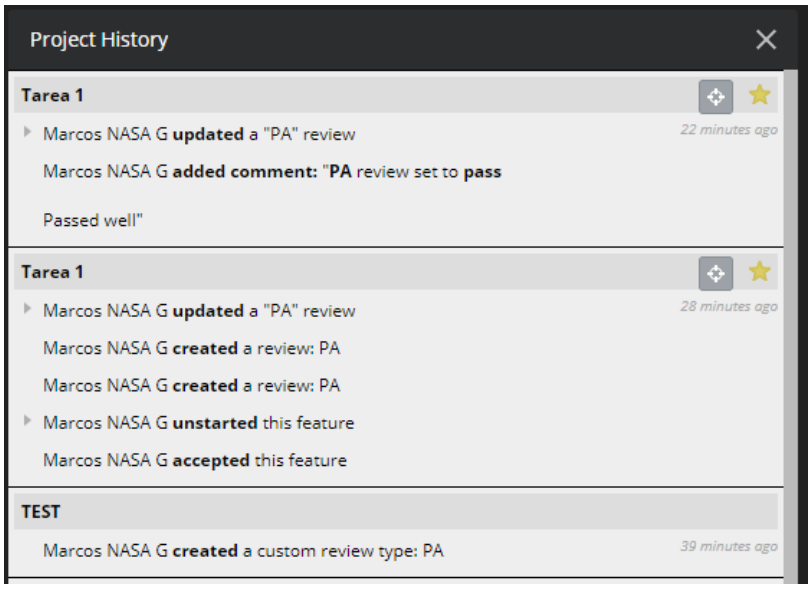

Figura 26. Historial de cambios del proyecto

Y hasta aquí llega el cumplimiento (en mayor o menor medida) de los requisitos que buscamos, ya que la herramienta no cuenta con ninguna opción adicional para agrupar o visualizar las *reviews*.

2.4. Conclusiones

Para cerrar este capítulo en el que se ha estudiado el estado de la gestión de PA en distintas herramientas para la gestión ágil de proyectos, compararemos cada una de las herramientas de las que hemos hablado marcando, según lo observado y documentado a lo largo del capítulo, el grado de cumplimiento de los requisitos inicialmente establecidos.

Los resultados de dicha comparación se presentan en la Tabla 1.

Tabla 1. Comparación de herramientas

	Jira	**Jira y XRAY**	**Pivotal Tracker**
Gestión de pruebas	Cuenta con opciones de configuración para modelar algo similar	Sí	Sí (a través de *reviews*)
Calidad de la gestión de pruebas	Mediocre	Aceptable	Insuficiente
Historial de cambios	Diluido en el historial de la incidencia	Diluido en el historial de la incidencia	Diluido en el historial del proyecto
Agrupación de PA conforme a la estructura del producto	No	Sí	No
Marcar OK y KO	Sí	Sí	Sí
Pasar en regresión	No	No	No
Listado de las PA en el contexto de un producto o un Sprint	No	Sí	No
Gráficas del estado de ejecución de las pruebas	Limitado o muy complicado	Limitado o muy complicado	No

No consideramos que el soporte para gestión de PA en estas herramientas sea satisfactorio. Las deficiencias encontradas en las herramientas estudiadas y las funcionalidades ya implementadas en la versión previa para Desktop de la herramienta Worki sientan una base a partir de la cual diseñar e implementar una gestión de PA mejorada y renovada para Worki en su versión web.

3. Tecnologías utilizadas

3.1. Angular 

Desde su lanzamiento en 2006, jQuery había sido la librería por excelencia para desarrollo web, ofreciendo facilidades para el manejo del DOM (Document Object Model).

Sin embargo, conforme los estándares web incluían las funcionalidades que ofrecía jQuery, otras librerías y marcos de trabajo para JavaScript ofrecían alternativas enfocadas en el diseño y la arquitectura. Los últimos 10 años de desarrollo web han estado marcados por una sólida entrada en escena del desarrollo basado en componentes.

Dicha arquitectura favorece la creación de SPA (*Single Page Applications*), las cuales elevan la experiencia de usuario al nivel de aplicaciones nativas de escritorio o móvil gracias a que se evitan los refrescos al navegar entre páginas.

Si bien la solución para desarrollo basado en componentes propuesta por los estándares (la Web Components API) no ha sido especialmente popular y algunos de los intentos por parte de librerías o marcos de trabajo (como Backbone.js, Knockout.js o Ember.js) no tuvieron la aceptación deseada, otros han calado hondo en la comunidad. En este grupo de afortunados es donde encontramos a las tres herramientas líderes en el desarrollo de aplicaciones web actual: Vue, Angular y React.

Angular[7] es una solución de código abierto cuyo desarrollo es liderado por Google. La primera versión fue publicada en el año 2009. Por aquel entonces ya incluía algunas de las funcionalidades clave que se han conservado hasta la actualidad. Sin embargo, presentaba problemas tan profundos que, para la segunda versión, el marco de trabajo fue totalmente reescrito [6].

A partir de la segunda versión, diferenciamos entre AngularJS (versiones de Angular inferiores a la 2.0) y Angular (versiones iguales o posteriores a Angular 2.0). La primera versión de Angular (Angular 2.0) fue lanzada en 2016. A partir de la versión 4.0, apenas ha sufrido cambios notables.

Podemos definir Angular como un marco de trabajo para el desarrollo de aplicaciones web basado en componentes. Los conceptos clave para entender cómo funciona se expondrán en las siguientes subsecciones.

[7] https://angular.io/

3.1.1. Arquitectura basada en componentes y módulos

Un componente representa una vista o una sección de una vista: así pues, los componentes de Angular pueden anidarse para componer las vistas de nuestra aplicación. Vamos a emplear una sencilla aplicación desarrollada específicamente con el fin de ejemplificar, durante esta sección, el funcionamiento de Angular: se trata de un Kanban que permite añadir, editar, eliminar y mover columnas y fichas. Para facilitar referirnos a esta aplicación, la hemos llamado "Chipzset". Su interfaz se muestra en la Figura 27.

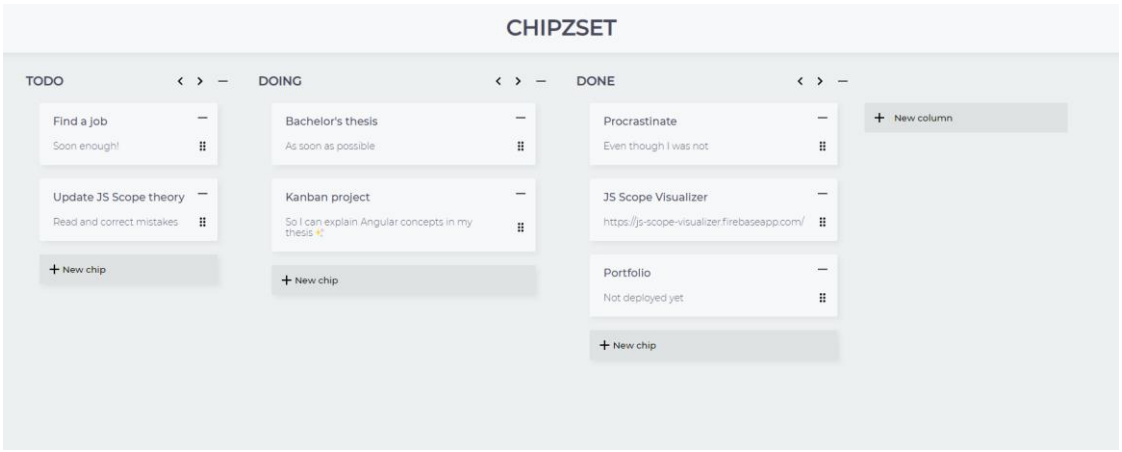

Figura 27. Interfaz de usuario de Chipzset

Su árbol de componentes se muestra en la Figura 28.

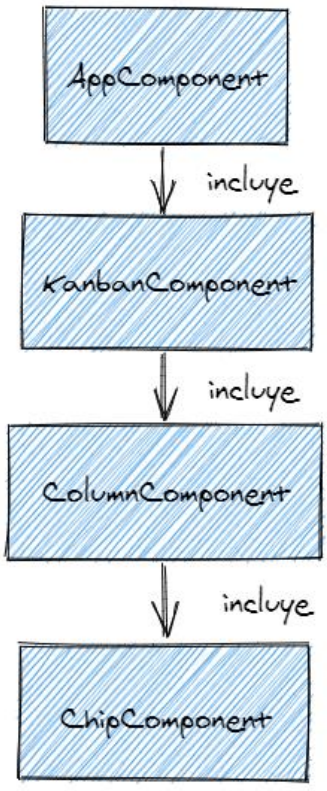

Figura 28. Árbol de componentes de Chipzset

La relación de "inclusión" mostrada en la Figura 28 (Figura anterior) implica que el componente situado en el origen de la flecha "hace uso" (incluye en su plantilla) del componente situado en el destino de la flecha.

A nivel de desarrollo, para el programador un componente no es más que una clase de TypeScript decorada mediante el decorador @Component, el cual contiene configuración crucial:

- Una propiedad "selector" de tipo string que permite dotar al componente de un identificador con el que podrá ser incluido en los templates de otros componentes.
- Una propiedad "templateUrl" que nos permite enlazar el fichero HTML que hará de plantilla para el componente.
- Una propiedad "styleUrls" que nos permite enlazar una o varias hojas de estilo.
- Otras tantas más.

La Figura 29 muestra el decorador @Component con dichas propiedades aplicadas a la clase AppComponent.

Figura 29. Parte del AppComponent de Chipzset

Por otro lado, los NgModules son unidades de compilación que se encargan de decirle a Angular cómo debe compilar los componentes. Para ello, entre otras propiedas, cuentan con declaraciones, importaciones y exportaciones:

- Declaraciones: todo componente debe estar declarado en un NgModule.
- Importaciones y exportaciones: un módulo A importa un módulo B para permitir el uso de los componentes exportados por B en los componentes declarados por A.

Los módulos se relacionan entre sí mediante una estructura en árbol no necesariamente idéntica a la de componentes, ya que un módulo puede ser el encargado de declarar varios componentes. La Figura 30 representa el árbol de módulos de Chipzset (en rojo) y su relación con el árbol de componentes (en azul).

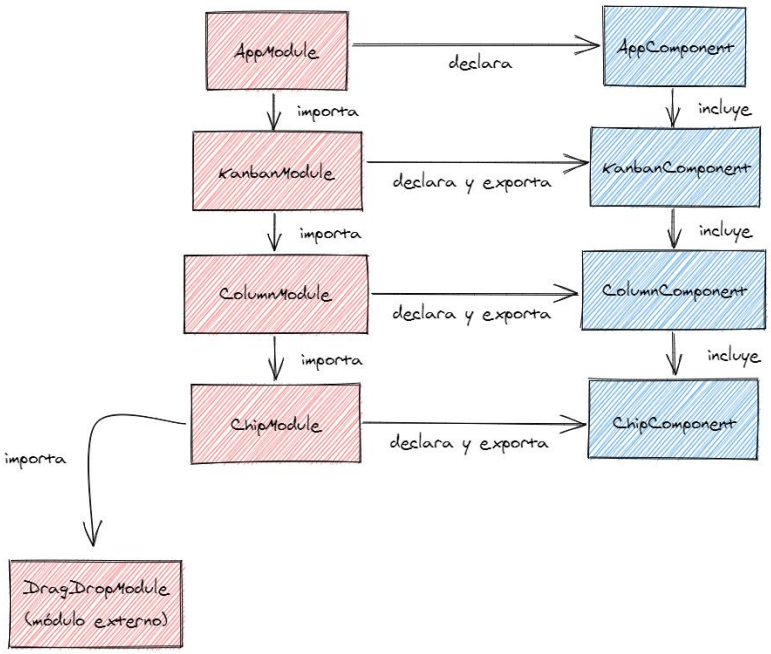

Figura 30. Árbol de módulos de Chipzset y su relación con el árbol de componentes

La relación de "importación" entre un módulo A y un módulo B, siendo A quien importa a B, permite hacer uso de los componentes exportados por B en los componentes declarados por A.

La estructura en árbol de los módulos tiene un único módulo raíz (generalmente el AppModule, que declara el AppComponent) con una propiedad "bootstrap" (arranque) que se encarga, como su nombre indica, de arrancar la aplicación, inicializando los componentes "declarados" en un arreglo como valor de la propiedad "declarations". El AppModule de Chipzset se ilustra en la Figura 31.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { KanbanModule } from './kanban/kanban.module';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, KanbanModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Figura 31. AppModule de Chipzset

Los NgModules también se encargan de configurar la inyección de dependencias (de la que hablaremos más adelante) y de configurar la carga perezosa de componentes, entre otros [7].

A nivel de desarrollo, para el programador un NgModule no es más que una clase de TypeScript vacía y decorada mediante el decorador @NgModule, el cual contiene la configuración correspondiente.

3.1.2. HTML Templates (plantillas HTML)

Cada componente cuenta con un template o plantilla HTML que mediante los conceptos de interpolación (representada en angular por dobles llaves) y diferentes directivas permite insertar código "tipo JavaScript" (*JavaScript-like*) para dotar a Angular de la capacidad de renderizar atributos del componente dinámicamente. La Figura 32 ilustra una parte de una de las plantillas de Chipzset donde está presente el uso de interpolación para renderizar dinámicamente el valor de los atributos *name* y *description*.

```
<!-- Parte del template del ChipComponent -->
<h3 class="mng-chip__title"
    contenteditable="true"
    aria-label="Update chip name"
    (blur)="updateChipName($event)">
  {{ name }} <!-- Interpolación -->
</h3>
<p class="mng-chip__description"
   contenteditable="true"
   aria-label="Update chip description"
   (blur)="updateChipDescription($event)">
  {{ description }} <!-- Interpolación -->
</p>
```

Figura 32. Ejemplo de interpolación

La Figura 33 ilustra una plantilla distinta donde está presente el uso de la directiva *ngFor*.

```
<!-- Parte del template del ColumnComponent -->
<!-- NgFor es una directiva que, aplicada sobre una
etiqueta HTML, declara una variable para cada elemento
de una lista y renderiza la etiqueta HTML una vez por
cada elemento -->
<mng-chip *ngFor="let chip of chips"
          [cdkDragData]="chip"
          [id]="chip.id"
          [(name)]="chip.name"
          [(description)]="chip.description"
          (onDeleteChip)="deleteChip($event)"
          (onUpdateState)="updateState()"
          cdkDrag>
  <i class="mng-chip__drag"
     aria-label="Drag chip"
     cdkDragHandle>
    <svg xmlns="http://www.w3.org/2000/svg"
         height="24"
         viewBox="0 0 24 24"
         width="24">
    </svg>
  </i>
</mng-chip>
```

Figura 33. Ejemplo de directivas (directiva ngFor)

Pero sin duda, en la categoría de características más potentes de comunicación *plantilla-componente*, la corona es para el *two-way data binding*. En la Figura 33 (Figura anterior) se pueden ver dos ejemplos de *two-way data binding*. A continuación, trataremos de desgranar el código mostrado en dicha Figura:

- Se muestra la parte del template del ColumnComponent responsable de renderizar la lista de fichas de la columna.
- El componente de las fichas tiene asignado el selector 'mng-chip'.
- A través de la directiva *ngFor* en la etiqueta '<mng-chip>', renderizamos cada una de las fichas de la lista de fichas que pertenecen a la columna.
- El resto de contenido de la etiqueta '<mng-chip>' son atributos y manejadores de eventos. Los atributos y manejadores de eventos son pares '<clave>=<valor>':
  - → Los atributos envían información del componente padre al componente hijo. La información en el hijo se actualiza automáticamente cuando cambia en el componente padre. Se reconocen por su forma '[clave]="valor"'.
  - → Los manejadores de eventos dicen qué función ejecutar en el componente padre cuando el hijo emite un cierto evento. Se reconocen por su forma '(clave)="valor()"'.

  Pero hay un tercer tipo presente, mucho más interesante y responsable del *two-way data binding*, que es algo así como atributo y manejador de eventos al mismo tiempo. Se pueden reconocer fácilmente por su forma [(clave)]="valor". La sintaxis "[(clave)]" se conoce popularmente por su forma como "plátano en la caja". El *two-way data binding* envía información del componente padre al componente hijo y del componete hijo al componente padre: si cambia en el componente padre, el hijo se actualiza automáticamente y si cambia en el componente hijo, el padre se actualiza automáticamente.
  Esto se traduce en que el manejo del estado de la aplicación sea mucho más sencillo e intuitivo.

3.1.3. TypeScript

TypeScript[8] es un superconjunto de JavaScript, pero de tipado fuerte. Esta característica lo convierte en la elección preferida por aquellos desarrolladores que provienen de lenguajes sin tipado dinámico.

Apoyado en TypeScript, Angular aboga mayoritariamente por el paradigma de Programación Orientada a Objetos.

[8] https://www.typescriptlang.org/

3.1.4. Lifecycle Hooks

Angular ofrece al desarrollador un conjunto de interfaces para reaccionar a los eventos del ciclo de vida de un componente. Dichas interfaces reciben el nombre de *ganchos del ciclo de vida*.

Haciendo alusión a la traducción literal, los *ganchos* no son más que eventos que se disparan en diferentes partes *del ciclo de vida* del componente: antes o después del renderizado, una vez el componente se inicia, cuando un cambio es detectado por el detector de cambios, antes o después de la destrucción del componente…

Dichos eventos nos permiten realizar operaciones en los momentos más oportunos, como hacer una petición HTTP al inicio del componente, anular la suscripción a un evento al destruir el componente…

La Figura 34 muestra un ejemplo de uso del *lifecycle hook* "*ngOnInit*", el cual efectúa operaciones antes del renderizado del componente.

```
// Ciclo de vida OnInit del componente AppComponent
export class AppComponent implements OnInit {
  // Más código

  constructor() {}

  ngOnInit(): void { // Se ejecuta cuando el componente se inicializa
    if (!localStorage.getItem('mng-kanban')) {
      this.saveColumns(this.defaultKanbanData);
    }

    this.kanbanData =
      (JSON.parse(localStorage.getItem('mng-kanban')) as Column[]) ||
      this.defaultKanbanData;
  }

  // Más código //
}
```

Figura 34. Uso de ganchos del ciclo de vida

3.1.5. Inyección de dependencias

El concepto de inyección de dependencias en Angular está profundamente relacionado con el concepto de servicios. Los servicios son clases de TypeScript decoradas mediante el decorador @Injectable.

La inyección de dependencias nos permite incluir un servicio en el constructor de un componente sin necesidad de instanciarlo nosotros mismos. La inyección de dependencias se encarga de la instanciación de forma eficiente conforme sea necesario. Para ello, el servicio debe ser declarado como *provider* en el componente, en su módulo (NgModule) o haber sido declarado como *provider* en algún componente o módulo padre. Esto conduce a código más desacoplado, facilitando el testing de los componentes.

Del mismo modo que ocurre con los componentes o los módulos, Angular mantiene un árbol de inyectores de dependencias. Cuando un servicio es requerido en el constructor de un componente, un algoritmo de resolución de dependencias se encarga de hacer una búsqueda en dicho árbol, comenzando por el componente que lo requiere o su módulo y, si no lo encuentra allí, tratando de resolver dicha dependencia a través de los inyectores ancestros [8].

3.1.6. Detección de cambios

El mecanismo de detección de cambios (CD) de Angular permite a la aplicación reaccionar a los cambios de estado.

Angular divide la detección de cambios en dos fases:

— Fase de actualización del modelo que representa el estado.
— Fase de reflejar los dichos cambios de estado en la vista.

La responsabilidad de la primera fase recae en el programador, mientras que la segunda es manejada por el marco de trabajo a través de diferentes técnicas en cada turno de la máquina virtual de JavaScript [9].

El modelo que representará el estado de Chipzset es una lista de columnas (Column[]). Los modelos de las fichas y las columnas pueden verse en las Figuras 35 y 36, respectivamente.

```
import { Chip } from './Chip';

export class Column {
  id: string | number;
  name: string;
  chips: Chip[];

  constructor() {}
}
```

Figura 35. Clase Columna

```
export class Chip {
  id: string | number;
  name: string;
  description: string;

  constructor() {}
}
```

Figura 36. Clase Ficha

Como hemos mencionado anteriormente, las aplicaciones de Angular consisten en componentes anidados formando una estructura en árbol. Cuando se produce un cambio en el estado de la aplicación, dicho cambio se propaga a través del árbol de componentes actualizando las vistas de aquellos componentes que dependiesen de las propiedades del estado que hayan cambiado.

Cada componente se compila en una serie de instrucciones que construyen un árbol DOM propio, el cual muta cuando el CD detecta un cambio; si cambia alguna de las propiedades de las que el componente depende, el detector de cambios programará un nuevo renderizado para el siguiente turno de la máquina virtual [9]. Este mecanismo de compilación de componentes es conocido como Incremental DOM y es especialmente eficiente en el contexto de las librerías más populares para JavaScript dado que:

— No construye un nuevo DOM virtual (muy caro en términos de memoria) que refleje los cambios del estado para comparar con el DOM virtual anterior (técnica empleada por React).
— Un nuevo renderizado del componente sólo es programado cuando cambian las propiedades del estado que emplee en su plantilla.
— Es *tree-shakable*: las instrucciones que un componente requiere para construir su DOM son conocidas en tiempo de compilación, por lo que instrucciones no usadas no son incluidas en el paquete final de la aplicación.

3.1.7. Zone.js

Zone.js[9] es una librería desarrollada por el equipo de Angular para el proyecto de Angular, pero puede ser y es empleada fuera del marco de trabajo. La librería proporciona un mecanismo para interceptar y hacer un seguimiento de tareas asíncronas. Para cada operación asíncrona, Zone.js crea una tarea; las tareas se ejecutan en una "zona" (un contexto de ejecución).

Dicho de otro modo, el propósito de Zone.js es proporcionar una capa intermedia entre las operaciones asíncronas y el navegador; dicha capa actúa como contexto de ejecución común para las operaciones asíncronas primitivas (*callbacks*, *Promises*...) propias de JavaScript [10]. Este comportamiento se ilustra en la Figura 37.

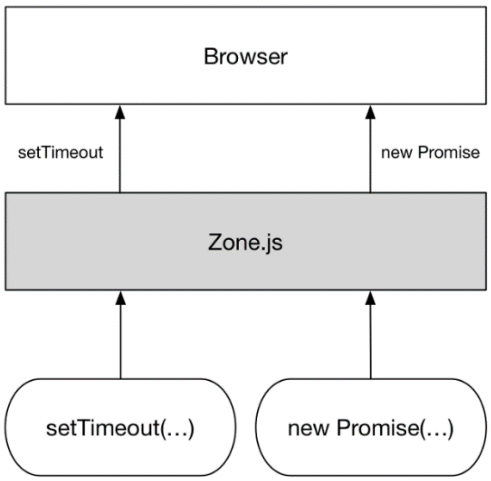

Figura 37. Zone.js

Las aplicaciones de Angular ejecutan dichas operaciones asíncronas en una única "zona" común a la que se conoce como NgZone.

La detección de cambios de la que hemos hablado anteriormente, cuando se trata de operaciones asíncronas, sólo se dispara para aquéllas ejecutadas en la NgZone. Sin embargo, es posible ejecutar operaciones asíncronas *fuera de la zona* comunicándoselo a Angular explícitamente: esto habilita la posibilidad de controlar la detección de cambios por medios propios.

[9] https://github.com/angular/angular/tree/master/packages/zone.js

3.2. Librerías de componentes de interfaz de usuario

DevExtreme y Syncfusion son dos librerías de componentes de interfaz de usuario genéricos y configurables desarrolladas por las compañías DevExpress y Syncfusion, respectivamente.

Ambas librerías son populares en diferentes comunidades de desarrollo de software, puesto que están disponibles para distintos entornos y lenguajes, como C# (.NET) o JavaScript (React, Angular, Vue y jQuery).

La calidad de los componentes y su facilidad de instalación y uso las convierten en dos opciones muy interesantes para cualquier proyecto en compañías de todas las escalas. La calidad del soporte técnico, en ambos casos, es excelente.

Además de las dos librerías mencionadas, hay una tercera que entra en juego en el contexto de Worki, NgPrime[10]. Sin embargo, apenas he interactuado de forma directa con esta librería, sino a través de servicios propios de Worki que la utilizan *por debajo* para mostrar diálogos con mensajes o advertencias al realizar ciertas acciones y para contener los formularios emergentes que permiten la ejecución de parte de la funcionalidad implementada.

3.2.1. DevExtreme 

DevExtreme[11] está disponible para JavaScript, jQuery, Angular, Vue, React, AngularJS, ASP.NET Core y ASP.NET MVC.

A nivel de licencia, se ofrece a través de un pago único que incluye todas las actualizaciones futuras. Cuenta con la posibilidad de solicitar una prueba gratuita de 30 días (incluyendo soporte técnico) y con una garantía de 60 días a partir de su compra. La documentación de esta herramienta es, por calidad y cantidad de contenido, la que sale ganando en la comparativa.

Los componentes de DevExpress utilizados para el desarrollo de la solución han sido:

- DataGrid: una tabla de datos genérica con potentes opciones de configuración, filtrado y mejora de rendimiento.
- Varios componentes *de formulario*: áreas de texto, inputs, selectores, editores de texto enriquecido, botones…
- Tooltip: una cómoda forma de incluir mensajes emergentes bajo ciertas condiciones.
- Tabs: un componente de pestañas genérico. Fue usado para ampliar las vistas de la estructura del producto con la lista de PA y para desarrollar el gestor de PA, como veremos más adelante.

[10] https://www.primefaces.org/primeng/
[11] https://www.devexpress.com/

3.2.2. Syncfusion 

Syncfusion[12] está disponible para JavaScript, jQuery, Angular, Vue, React, Flutter, Xamarin, ASP.NET Core, ASP.NET MVC y ASP.NET Web Forms.

A nivel de licencia, es extremadamente similar a DevExtreme: se ofrece a través de un pago único que incluye todas las actualizaciones futuras y, del mismo modo, cuenta con la posibilidad de solicitar una prueba gratuita de 30 días. Como se ha insinuado en la sección de DevExtreme, la documentación de Syncfusion es algo menos completa (o tal vez no tan bien estructurada) en comparación. Pese a ello, todos los componentes están correctamente documentos.

Sólo hay un componente de Syncfusion que haya sido empleado para la inclusión de gestión de PA en Worki:

— TreeView: una vista de estructura en árbol con múltiples opciones de configuración que permite (entre muchas otras opciones) marcar, mover, renombrar, desplegar o contraer nodos. Fue usado para marcar un nodo asociado a la estructura en árbol de un producto y permitir mover o copiar una prueba de aceptación a dicho nodo.

3.3. Git, Bitbucket y Source Tree

Git[13] es un sistema de control de versiones de código libre usado alrededor de todo el mundo por grandes, medianas y pequeñas empresas y desarrolladores independientes.

Como hosting para Git, TUNE-UP Process emplea la solución Bitbucket[14] desarrollada por Atalassian, además de SourceTree[15], propiedad de la misma empresa, para facilitar el uso de Git a través de una amigable interfaz gráfica de usuario.

[12] https://www.syncfusion.com/
[13] https://git-scm.com/
[14] https://bitbucket.org/
[15] https://www.sourcetreeapp.com/

4. Desarrollo de gestión de PA en Worki

Como se ha indicado en el primer capítulo, el objetivo final de este trabajo es la implementación de un sólido sistema de gestión de PA en Worki que sirva para especificar y validar los requisitos de un producto.

Worki representa los requisitos de un producto como una estructura en árbol donde cada nodo del árbol representa un requisito. Las UT se definen como cambios en la estructura de requisitos [11], pudiendo afectar a uno o más nodos de la estructura. En relación con esto, las PA estarán siempre asociadas:

- Al nodo de la estructura de requisitos que especifican y validan.
- A una de las UT que representa un cambio en dicho requisito (última UT modificadora).
- A un agente (último agente modificador).

Además, las PA modifican su estado a través de ejecuciones de PA, las cuales están también asociadas a la UT desde la que se ejecutan y al agente que las ejecuta. A través de la última ejecución de una PA podemos inferir una segunda relación entre la PA y las entidades UT (última UT ejecutora) y agente (último agente ejecutor). Esta dupla de relaciones de PA con UT (última UT modificadora y última UT ejecutora) añaden una capa semántica más profunda a la gestión de PA en Worki, de la cual se infiere información crucial. Dada una UT A:

- Si una PA ha sido modificada por A, pero no ejecutada por A, todavía está por aplicar.
- Si una PA no ha sido modificada por A, pero sí ejecutada por A, ha sido aplicada en regresión.

Por otro lado, como ya hemos anticipado, la herramienta Worki permite el control del estado de aplicación de una PA mediante las ejecuciones de PA. En Worki, las ejecuciones de las PA tienen tres posibles resultados:

- OK: prueba pasada. No es necesario especificar observaciones.
- CHECK: no es posible determinar si la prueba ha pasado o no o no se tiene claro cómo reproducirlo. Requiere especificar observaciones (pistas o indicios útiles para determinar el resultado).
- KO: prueba no pasada. Requiere especificar observaciones (generalmente, los pasos para reproducir el error).

Las actividades del flujo de trabajo de una UT susceptibles de producir ejecuciones son, generalmente y en el contexto del desarrollo de software, aquéllas relacionadas con el desarrollo y el testing:

— El programador debería notificar a través de una ejecución que los requisitos de la UT a la que está asociada la PA se cumplen y que su trabajo en lo que respecta a dicha UT ha terminado.
— El tester debería diseñar los casos de prueba y correr las pruebas necesarias para verificar el cumplimiento de los requisitos.

El Anexo A muestra las interfaces desarrolladas y explica cómo usar en profundidad todas las funcionalidades que se describen durante este capítulo.

4.1. Especificación de Requisitos

Una vez hemos hablado de la solución, especificaremos los requisitos a partir de un diagrama UML de casos de uso y los prototipos de las interfaces de usuario desarrolladas para cada requisito, los cuales irán acompañados de la lista de PA que los especifican y validan.

La Figura 38 muestra el diagrama UML de casos de uso.

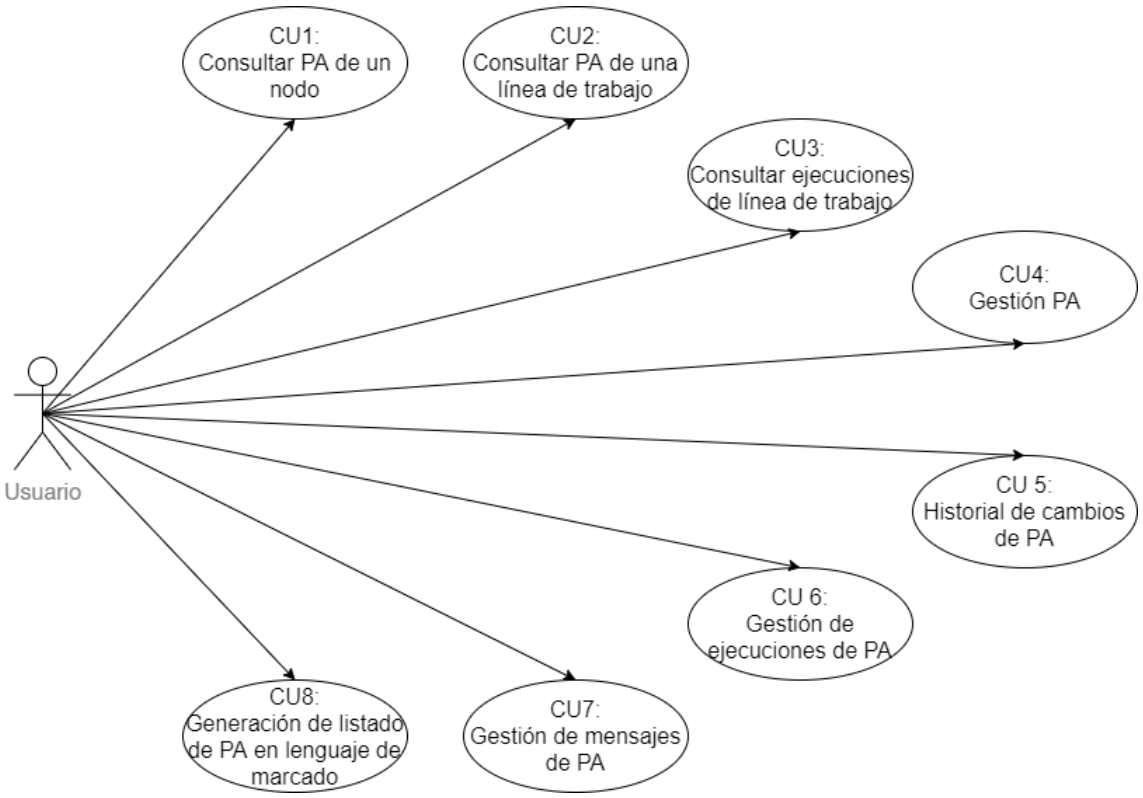

Figura 38. Diagrama de casos de uso

REQUISITO 1 (CU1). Consultar PA de un nodo.

Para la dotación de gestión de PA a Worki partimos de dos interfaces muy similares: la de Estructura-Temas y la de Estructura-Temas asociada a UT presente dentro del gestor de la UT. Las partes comunes (casi la totalidad) de dichas interfaces de partida se ilustran en la Figura 39.

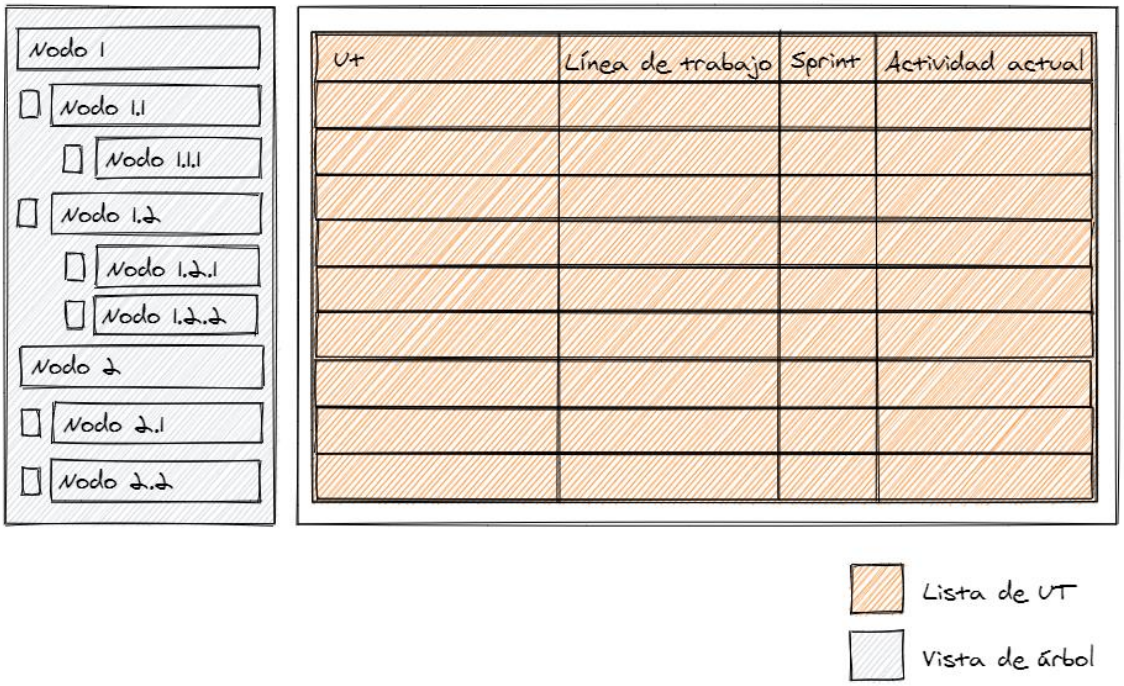

Figura 39. Estructura-Temas inicialmente.

Ambas muestran la estructura del producto mediante una vista en árbol y una tabla de datos que representa la lista de UT relacionadas al nodo de la vista en árbol seleccionado. La principal diferencia es que desde la vista de estructura del producto asociada a UT es posible marcar nodos de la vista en árbol para asociar la UT a uno o varios nodos.

Dichas interfaces serán ampliadas para incluir un componente de pestañas que permita, en función del nodo seleccionado en el árbol, mostrar la lista de UT asociadas mencionada anteriormente o —como primera novedad en el contexto de desarrollo del TFG— la lista de PA asociadas. Éste supuso el primer reto en el desarrollo de la solución y es representado en la Figura 40.

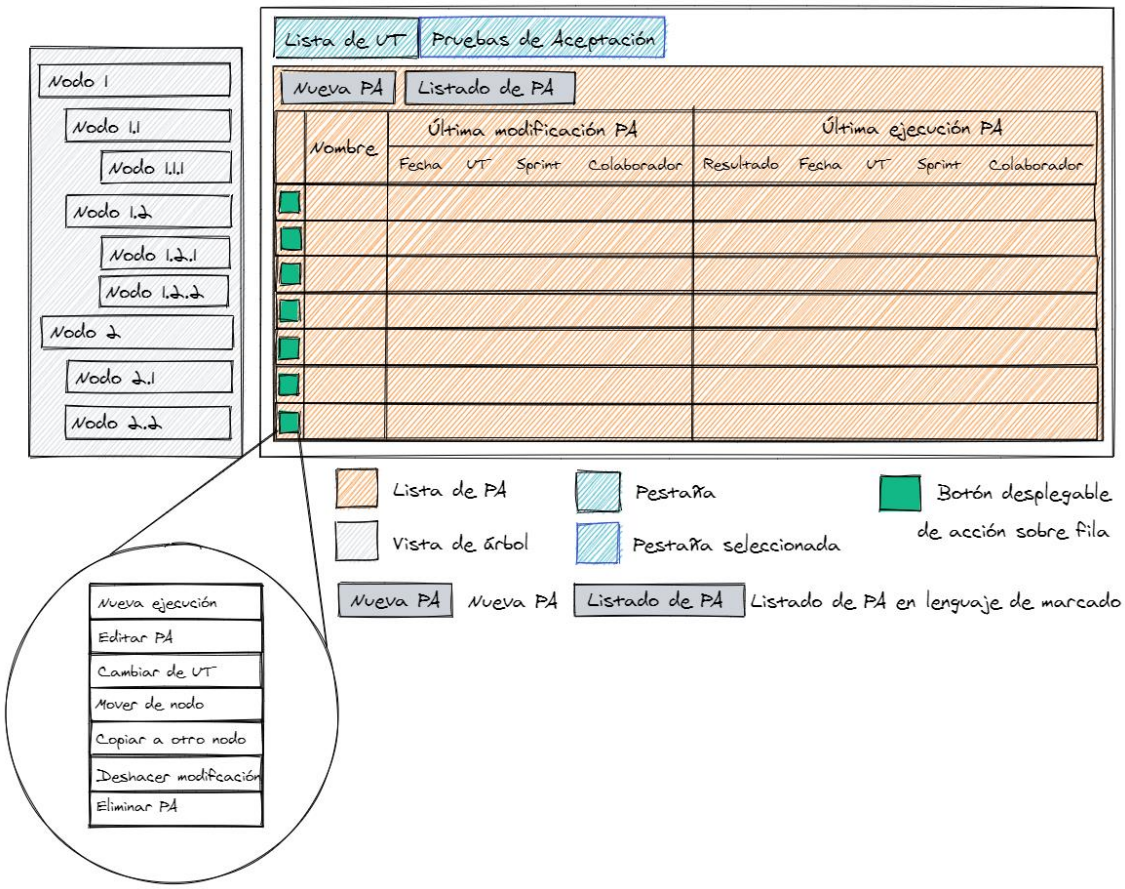

Figura 40. Prototipo Estructura-Temas ampliada

La vista de Estructura-Temas no asociada a UT no permitirá la creación de una PA ni ofrecerá opciones adicionales de gestión (aquéllas del botón desplegable de acción sobre fila).

— **PA requisito 1**.
1. Si no hay nodo seleccionado, no se muestran PA.
2. Al seleccionar un nodo sólo se muestran las PA de UT asociadas a dicho nodo.

REQUISITO 2 (CU2). Consultar PA de un producto.

La interfaz presenta un desplegable de selección de producto y una tabla de datos que muestra todas las PA asociadas a dicho producto, como se muestra en la Figura 41.

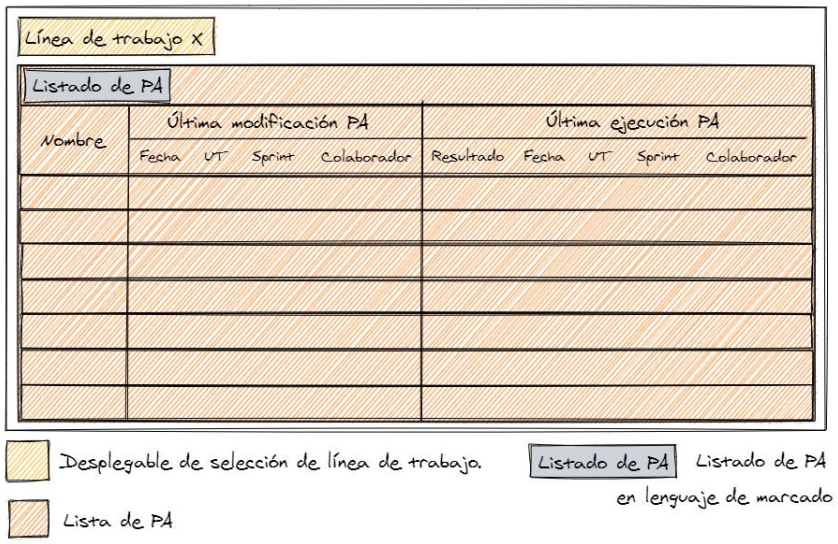

Figura 41. Prototipo Lista de PA

— **PA requisito 2**.
3. Si no hay producto seleccionado, no se muestran PA.
4. Al seleccionar un producto sólo se muestran las PA de UT asociadas al producto.

REQUISITO 3 (CU3). Consultar ejecuciones de un producto.

La interfaz presenta un desplegable de selección de producto y una tabla de datos que muestra todas las ejecuciones de PA asociadas a dicho producto, como se muestra en la Figura 42.

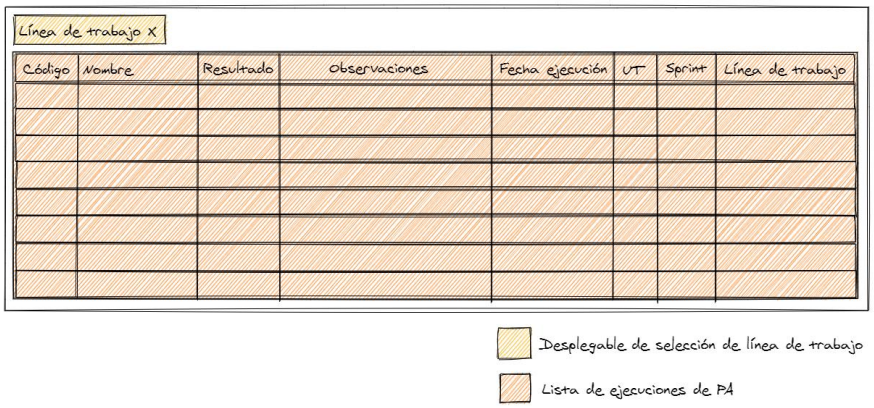

Figura 42. Prototipo Lista de ejecuciones de PA

— **PA requisito 3**.
5. Si no hay producto seleccionado, no se muestran ejecuciones.
6. Al seleccionar un producto sólo se muestran las ejecuciones de PA de UT que pertenecen al producto seleccionado.

REQUISITO 4 (CU4). Crear PA.

La creación de una PA se realiza a través de un modal localizable en Estructura-Temas del gestor de UT, ilustrado en la Figura 43.

Figura 43. Prototipo crear PA

— **PA requisito 4**.

7. No se permite añadir PA a una UT si la UT ya está terminada.
8. La definición ofrece por defecto la plantilla:
    — CONDICIONES.
    — PASOS.
    — RESULTADO.
    — CONCLUSIONES.

REQUISITO 5 (CU4). Editar PA.

Para una gestión aislada de una PA se generó una vista independiente que contuviese toda la información de una PA: el gestor de PA.

El gestor de PA permite la edición, la consulta del histórico de cambios, la gestión de ejecuciones y el intercambio de mensajes asociados a la PA. Es una vista altamente inspirada en el gestor de UT con el que Worki ya contaba.

El gestor de la PA es accesible desde varias vistas de la aplicación:

- Estructura-Temas (en el contexto de un nodo y un producto).
- Estructura-Temas asociada a UT (en el contexto de un nodo y una UT).
- Lista de PA (en el contexto de un producto).

En función del contexto desde el que accedemos al gestor, algunas de las operaciones estarán disponibles o no. Sólo el acceso en el contexto de una UT permite realizar la mayor parte de la gestión asociada. La interfaz básica del gestor de PA es ilustrada en la Figura 44.

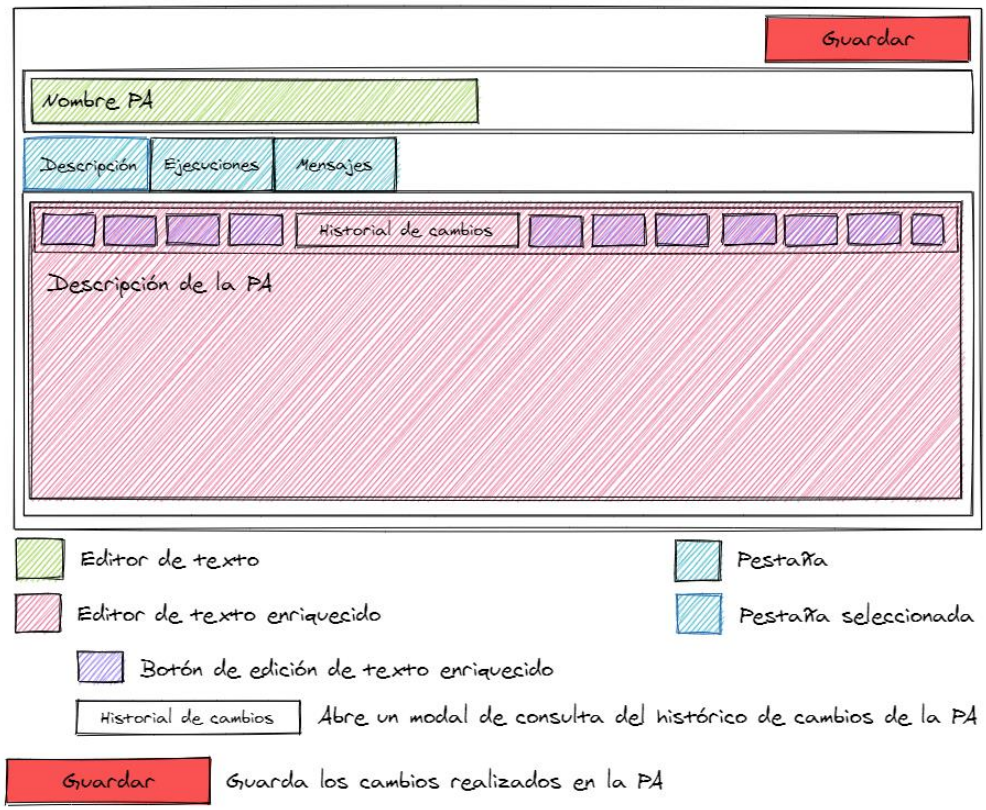

Figura 44. Prototipo gestor de PA

- **PA requisito 5**.
    9. No se permite editar una PA desde fuera del contexto de una UT.
    10. No se permite editar una PA en el contexto de una UT terminada.

REQUISITO 6 (CU4). Eliminar PA.

La eliminación de una PA se realiza desde Estructura-Temas del gestor de UT.

— **PA requisito 6**.
  11. No se puede eliminar una PA desde el contexto de una UT terminada.
  12. No se puede eliminar una PA desde el contexto de una UT distinta a la UT que realizó la última modificación a la PA.

REQUISITO 7 (CU4). Mover PA de nodo.

El movimiento de una PA de un nodo a otro se realiza a partir de un modal, ilustrado en la Figura 45, al que se accede desde Estructura-Temas del gestor de UT.

Bien por error, bien por reusabilidad de una PA o bien por un cambio en los requisitos del producto, es conveniente poder mover PA de un nodo a otro.

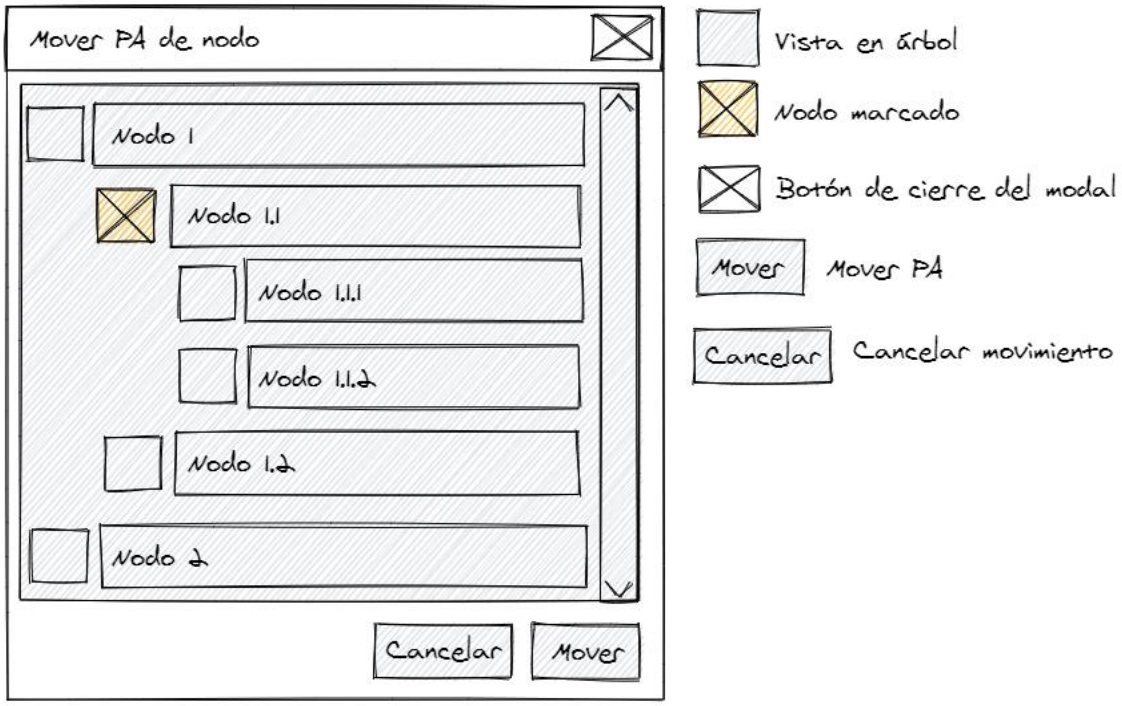

Figura 45. Prototipo mover PA de nodo

— **PA requisito 7**.
  13. Sólo es posible seleccionar un nodo.
  14. Si se mueve la PA a un nodo de la estructura no afectado por la UT desde la que se realiza el movimiento, pasará a estar afectado.

REQUISITO 8 (CU4). Copiar PA a otro nodo.

La copia de una PA a un nuevo nodo se realiza a través de un modal, ilustrado en la Figura 46, al que se accede desde Estructura-Temas del gestor de UT.

Por cuestiones de reusabilidad de una PA, es conveniente poder copiarla de un nodo a otro.

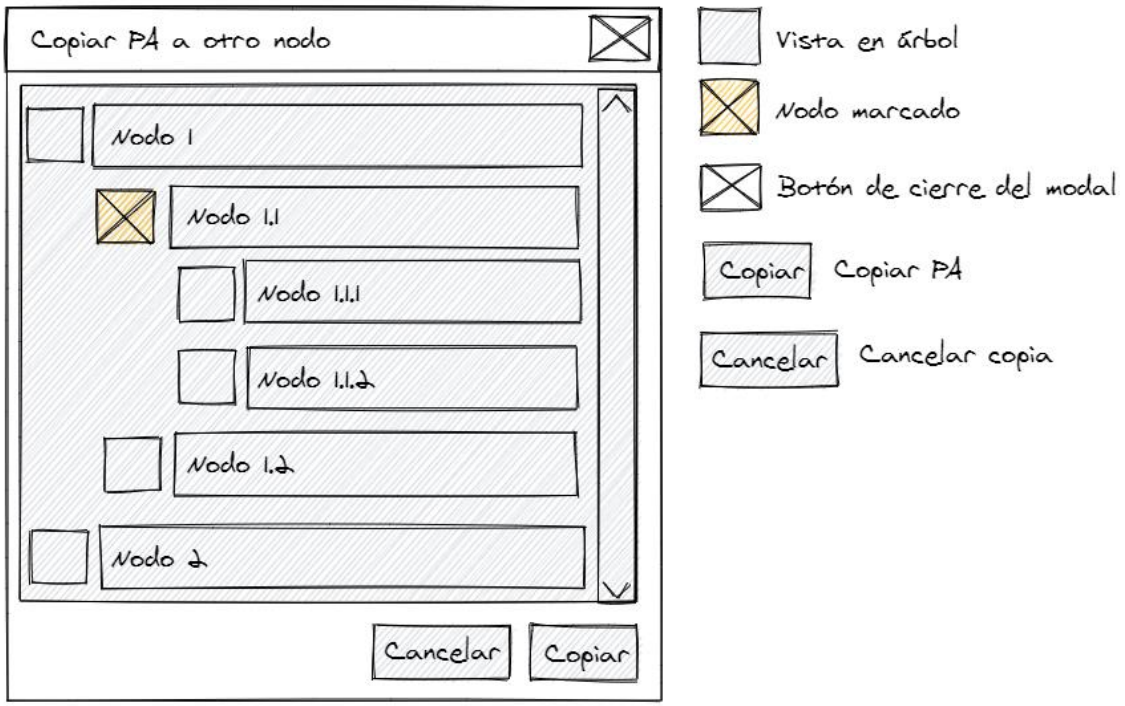

Figura 46. Copiar PA a otro nodo

— **PA requisito 8**.

15. Sólo es posible marcar un nodo.
16. Si se copia la PA a un nodo de la estructura no afectado por la UT desde la que se está realizando la copia, pasará a estar afectado.

REQUISITO 9 (CU4). Mover PA a otra UT.

El movimiento de una PA de una UT a otra se realiza a partir de un modal, ilustrado en la Figura 47, al que se accede desde Estructura-Temas del gestor de UT.

Una vez más, por error o por reusabilidad de una PA, es conveniente poder moverla a otra UT.

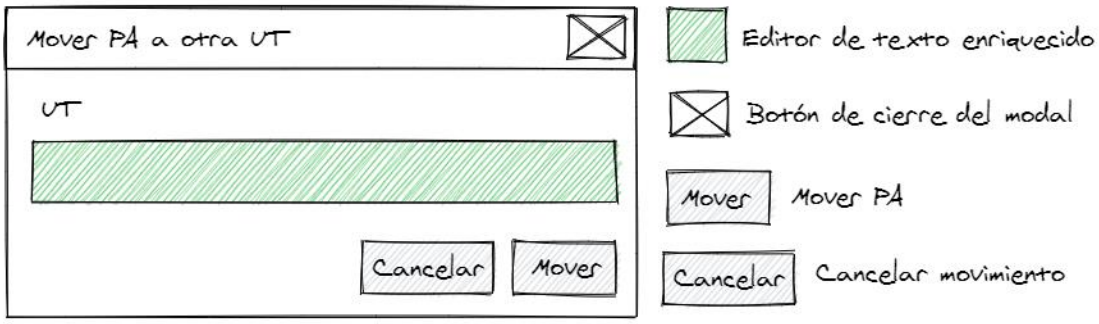

Figura 47. Prototipo mover PA a otra UT

— **PA requisito 9**.

17. No se puede mover una PA a otra UT desde el contexto de una UT terminada.
18. Si se mueve una PA a una UT que no afectase al nodo de la estructura al que está asociado la PA, el nodo pasará a estar afectado por la UT.

REQUISITO 10 (CU4). Deshacer modificación.

La acción de deshacer una modificación de una PA se realiza desde Estructura-Temas del gestor de UT.

— **PA requisito 10**.

19. No se puede deshacer una modificación de una PA desde el contexto de una UT terminada.

REQUISITO 11 (CU5). Historial de cambios de PA.

El histórico de cambios de una PA nos permite consultar las descripciones que una PA ha tenido a lo largo del tiempo, la UT desde la que se realizó el cambio y la fecha en la que se realizó, además del código de instancia que supuso dicho cambio para la PA.

Dicho histórico de cambios es un modal, ilustrado en la Figura 48, al que se puede acceder desde la pestaña "Descripción" del gestor de PA.

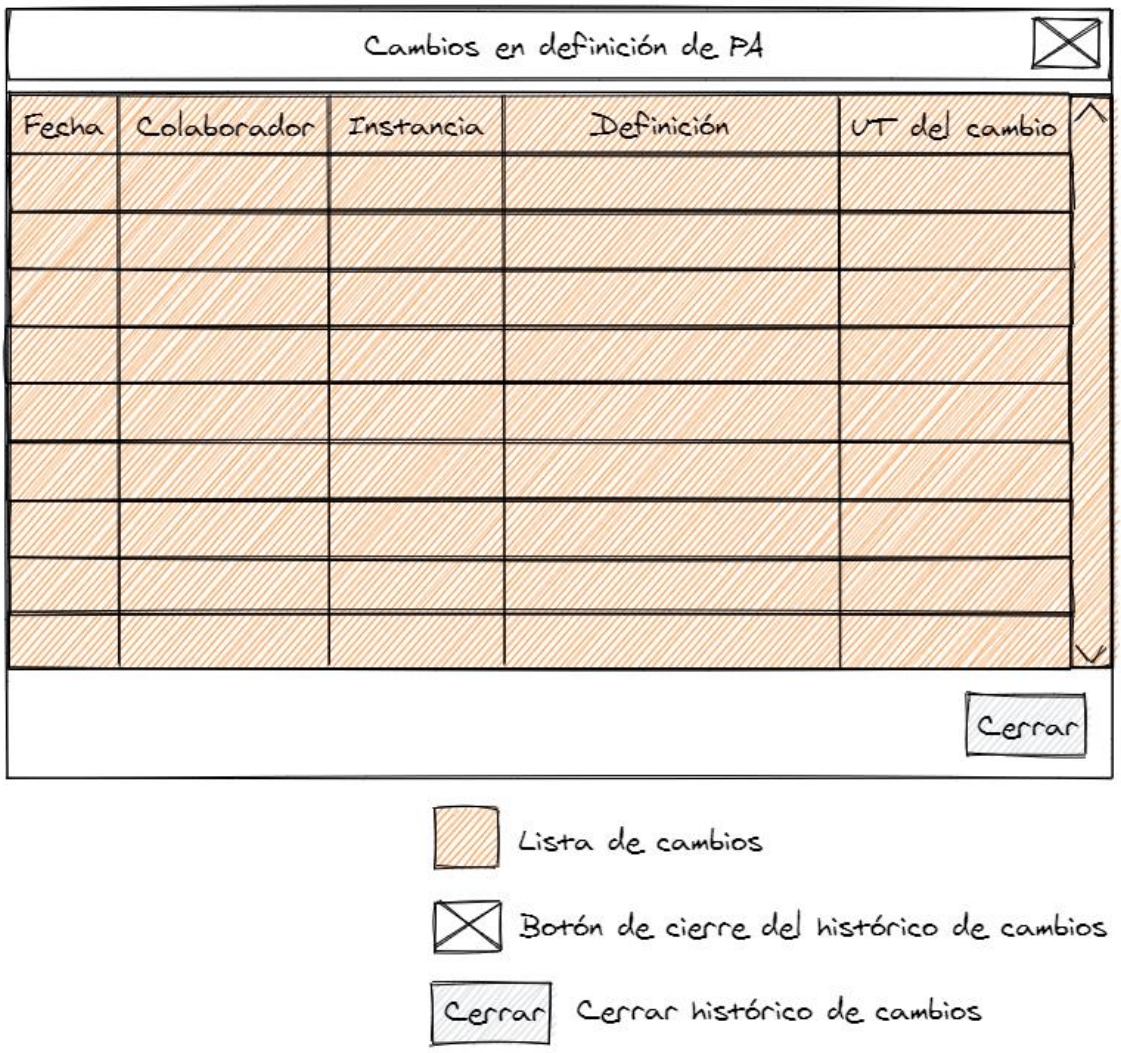

Figura 48. Prototipo histórico de cambios

REQUISITO 12 (CU6). Gestión de ejecuciones de PA.

La pestaña de ejecuciones de PA del gestor de PA nos permite consultar, eliminar, crear y editar las ejecuciones de una PA.

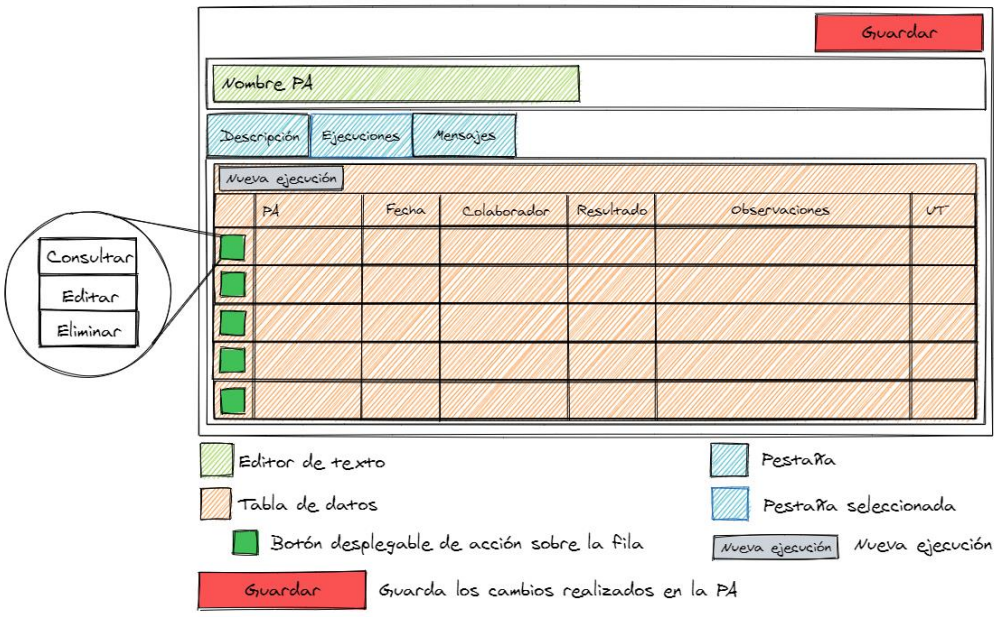

Figura 49. Prototipo ejecuciones de PA.

— **PA requisito 12**.

20. Sólo se pueden crear ejecuciones desde el contexto de una UT no terminada.
21. Sólo se puede editar la última ejecución de PA.
22. Sólo un administrador o el usuario que la creó puede editar una ejecución.
23. Sólo se puede eliminar la última ejecución de PA.
24. Sólo un administrador o el usuario que la creó puede eliminar una ejecución.

REQUISITO 13 (CU7). Gestión de mensajes de PA.

La pestaña de mensajes del gestor de PA nos permite consultar, eliminar, crear y editar los mensajes asociados a una PA para favorecer la comunicación entre miembros del equipo involucrados en el diseño o ejecución de una PA. La interfaz asociada al requisito se ilustra en la Figura 50.

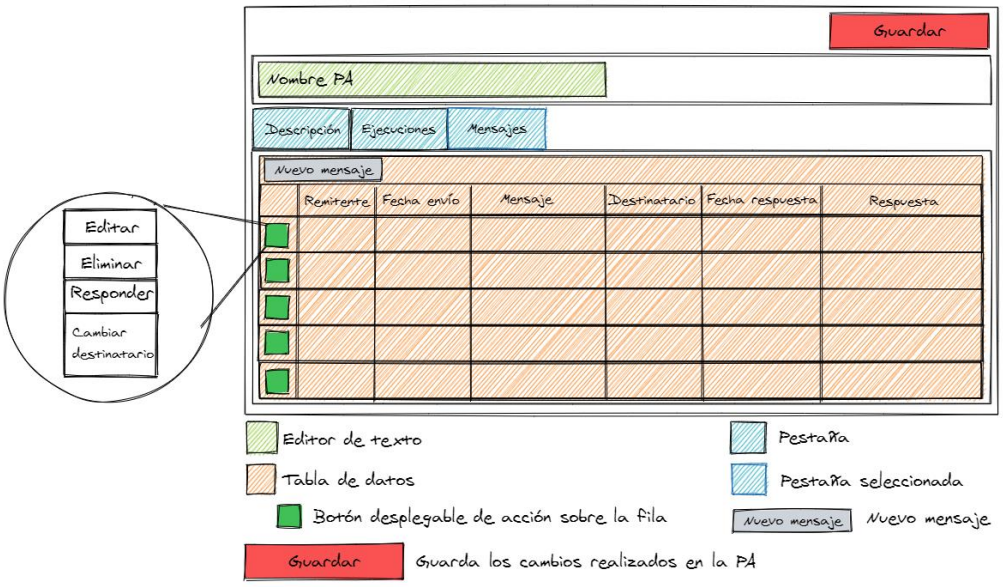

Figura 50. Prototipo mensajes de PA

— **PA requisito 13**.
25. Sólo el remitente puede editar un mensaje.
26. Sólo se puede editar un mensaje si no ha sido respondido ni leído.
27. Sólo el destinatario puede editar la respuesta a un mensaje.
28. Sólo se puede editar la respuesta a un mensaje si no ha sido marcada como leída por el remitente del mensaje.
29. Sólo el remitente puede eliminar un mensaje.
30. Sólo se puede eliminar un mensaje si no ha sido respondido.
31. Sólo se puede delegar un mensaje si no ha sido respondido.

REQUISITO 14 (CU8). Generación de listado de PA en lenguaje de marcado para facilitar su impresión.

El listado de PA en lenguaje de marcado tiene el principal propósito de ofrecer una forma sencilla de imprimir la lista de PA de un nodo o un producto.

— **PA requisito 14**.
32. Las PA mostradas en el listado respetan los filtros aplicados en la tabla de datos.

4.2. Diseño

La sección de diseño consistirá en la descripción del modelo de datos y el modelo de componentes a desarrollar.

4.2.1. Modelo de datos

Como se observa en la Figura 51, las PA son la entidad central del modelo de datos que representa el módulo desarrollado en el contexto de este trabajo.

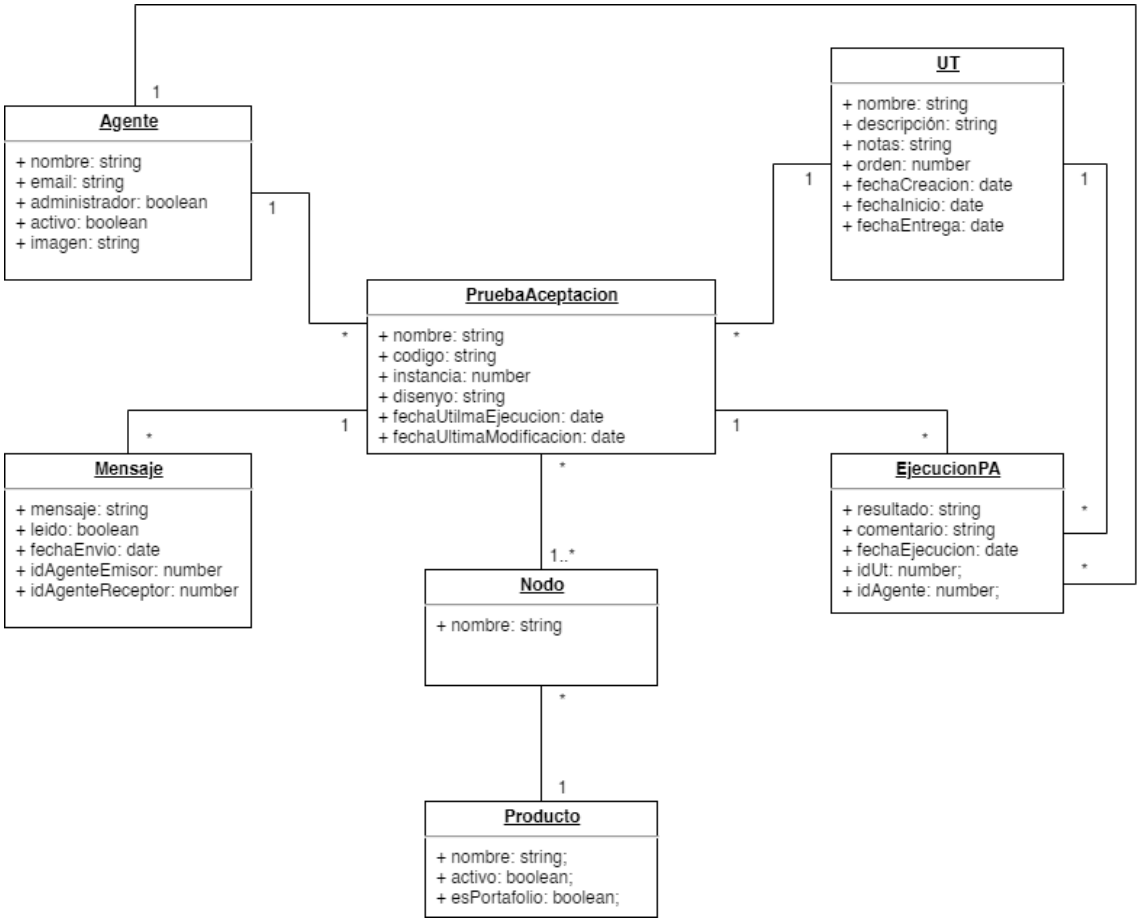

Figura 51. Modelo de datos

Una PA se relaciona con el agente que la crea o modifica y con la UT en la que se crea o a la que se mueve o desde la cual se modifica y con el nodo de la estructura del producto al que está asociado. Además, contiene las ejecuciones de PA y los mensajes asociados.

Del mismo modo, las ejecuciones de PA están relacionadas con el agente que las crea y con la UT desde la cual es creada.

Las implicaciones e importancia semántica de la relación de una PA con una UT en conjunto con la segunda relación que se infiere entre PA y UT a partir de su última ejecución de PA ha sido descrita al comienzo de este capítulo.

4.2.2. Modelo de componentes de Angular y Web API

La Figura 52 muestra el modelo de componentes de la aplicación.

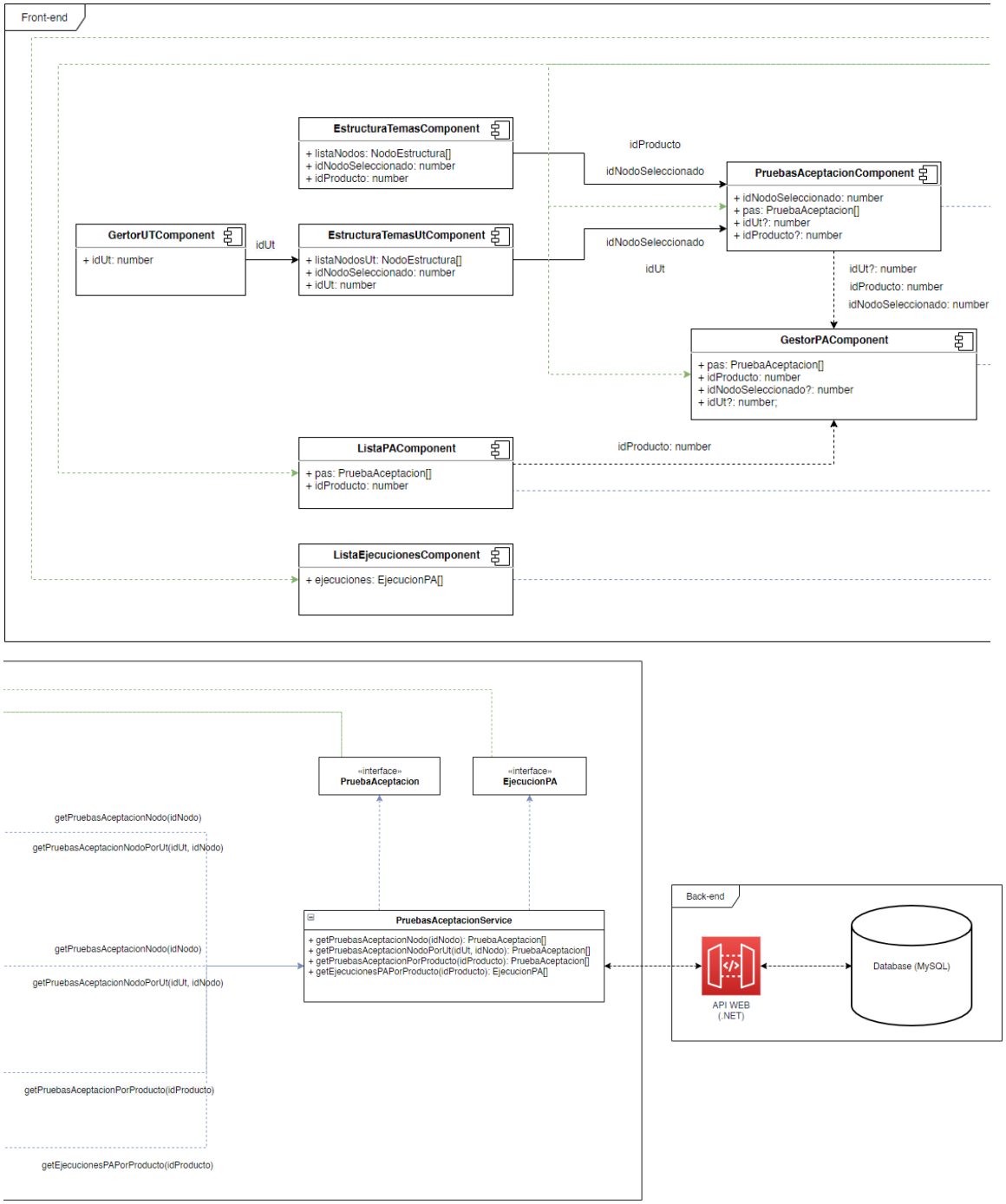

Figura 52. Modelo de componentes

A continuación, se describen los componentes desarrollados y algunos de los servicios de la API Web empleados.

Componentes

Los componentes desarrollados en el contexto del TFG son:

- PruebasAceptacionComponent.
- ListaPAComponent.
- ListaEjecucionesComponent.
- GestorPAComponent.

PruebasAceptacionComponent.

PruebasAceptacionComponent puede ser accedido desde EstructuraTemasComponent y EstructuraTemasUtComponent. Esta abstracción nos permite reutilizar el componente, facilitando su mantenimiento y reduciendo el tamaño del paquete de la aplicación una vez construida.

Dependiendo desde dónde se acceda, se hará en un contexto u otro:

- Desde EstructuraTemasComponent se hace en el contexto de un nodo (idNodoSeleccionado) y un producto (idProducto). En este contexto, muestra la lista de PA correspondiente a dicho nodo y producto.
- Desde EstructuraTemasUtComponent se hace en el contexto de un nodo (idNodoSeleccionado) y una UT (idUt). Como en Worki los nodos y las UT siempre están asociados a un producto, indirectamente también se accede en el contexto de dicho producto. En este contexto, se muestra la lista de PA correspondiente a dicho nodo y UT (más específico que el caso anterior).

En el contexto de EstructuraTemasUtComponent se permite realizar acciones que no se permiten en el contexto de EstructuraTemasComponent: sólo en el contexto de EstructurasTemasUtComponent el botón de Crear PA y los botones desplegables de acción sobre fila (consultar Figura 40) están presentes en la interfaz.

ListaPAComponent.

ListaPAComponent muestra todas las PA de las UT de un producto.

ListaEjecucionesPAComponent.

EjecucionesPAComponent muestra todas las ejecuciones de las PA de las UT de un producto.

GestorPAComponent.

El componente puede ser accedido desde PruebasAceptacionComponent y ListaPAComponent. Esta abstracción nos proporciona las mismas ventajas que en el caso de PruebasAceptacionComponent.

Desde ListaPAComponent, se accede a GestorPAComponent en el contexto de un producto.

Desde PruebasAceptacionComponent, es más complicado:

- Si PruebasAceptacionComponent ha sido accedido desde EstructuraTemasComponent, el acceso se realiza en el contexto de un producto y un nodo.
- Si PruebasAceptacionComponent ha sido accedido desde EstructuraTemasComponent, el acceso se realiza en el contexto de un nodo y una UT (e indirectamente, del mismo modo que hemos descrito anteriormente, de un producto).

El hecho de que se acceda o no en el contexto de un nodo es hoy en día irrelevante, pero nos puede ser útil en el futuro. Sin embargo, acceder en el contexto de una UT o no vuelve a marcar la diferencia a la hora de permitir realizar acciones: la edición de una PA o las ejecuciones de la PA sólo pueden ser gestionadas en caso de haber accedido en el contexto de una UT.

Servicios de la API Web

Para el desarrollo de la solución han sido necesarios un total de veintiún servicios de la API Web. La Figura 53 muestra los principales servicios empleados para recuperar las PA y las ejecuciones de la base de datos.

```
getPruebasAceptacionNodo(idNodo: number): Observable<PruebaAceptacion[]> {
  return this.httpService.get(`Nodos/${idNodo}/PruebasAceptacion`);
}

getPruebasAceptacionNodoPorUt(idUt: number, idNodo: number): Observable<PruebaAceptacion[]> {
  return this.httpService.get(
    `Uts/${idUt}/Nodos/${idNodo}/PruebasAceptacion`
  );
}

getPruebasAceptacionPorProducto(idProducto: number): Observable<PruebaAceptacion[]> {
  return this.httpService.get(`Productos/${idProducto}/PruebasAceptacion`);
}

getEjecucionesPAPorProducto(idProducto: number): Observable<EjecucionPA[]> {
  return this.httpService.get(`Productos/${idProducto}/EjecucionesPA`);
}
```

Figura 53. Servicios API Web principales

4.3. Programación

En el contexto de un marco de trabajo como Angular, el cual cuenta de serie con una cantidad de herramientas considerable, salirse de las soluciones propuestas por éste suele pasar también por salirse de las buenas prácticas. En la sección de Angular del capítulo "Tecnologías utilizadas" se han descrito varias características, tales como:

— Comunicación: atributos, emisores de eventos y *two-way data binding*.
— Encapsulación: componentes y módulos (NgModules).
— Desacoplamiento de código: inyección de dependencias.
— Ampliación de comportamiento: decoradores (@Component, @Injectable, @NgModule...).

Puesto que son inherentes a un uso apropiado del marco de trabajo, todas estas técnicas han sido aplicadas a lo largo del desarrollo de la solución.

Angular también incluye RxJS[16], una librería cercana al paradigma funcional para componer aplicaciones web asíncronas basadas en eventos a través de una implementación del patrón Observador y múltiples operadores. Es una librería cuyo uso adecuado es complejo dado que es necesario conocer conceptos de programación funcional en JavaScript para aprovechar las ventajas que ofrece dicho paradigma.

A continuación, veremos un ejemplo del uso de varias de las técnicas mencionadas junto a la librería RxJS para generar una *Guard*[17] que controle la navegación de salida de cierto componente. La Figura 54 muestra la *Guard* "PamEditCanDeactivate". En ella, se hace uso del patrón decorador (@Injectable) y se devuelve el tipo *Observable* de RxJS, además de hacer uso de los operadores de esta misma líbreria *pipe* y *take*.

```
@Injectable()
/**
 * Clase que representa la Guard que controla la salida del gestor de PA
 */
export class PamEditCanDeactivateGuard implements CanDeactivate<boolean> {
  /**
   * Método constructor de la Guard en el que se inyetan las dependencias
   * @constructor
   * @param {PamService} pamService
   */
  constructor(private readonly pamService: PamService) {}

  /**
   * Método invocado al tratar de salir del gestor de PA.
   * @return { Subject<boolean> } Observable al que se suscribe el router de Angular
para decidir si permite o no salir de la ruta actual
   */
  canDeactivate(): Observable<boolean> {
    return this.pamService.canDeactivate().pipe(take(1));
  }
}
```

Figura 54. Clase PamEditCanDeactivateGuard

[16] https://rxjs-dev.firebaseapp.com/
[17] Las Guards de Angular son clases inyectables que implementan la interfaz CanActivate/CanDeactivate y, al declararse como *providers* en un componente, invocan un método canActivate/canDeactivate encargado de controlar si se permite navegar al componente o salir de él, respectivamente.

La Figura 55 muestra el servicio "PamService" (PA Manager Service o servicio del gestor de PA) del que hace uso la *Guard*, ya que mantiene el estado de la PA original y de la PA que haya (o no) sufrido cambios. Se emplea el método *of* de la clase Observable y se crea una nueva instancia de un Observable para manejar la operación asíncrona en la que se pide al usuario tomar la decisión de salir o no del gestor de PA (emitiendo nuevos valores del Observable) en caso de que haya cambios sin guardar.

```typescript
@Injectable()
export class PamService {
  idUt: number;
  pA: PruebaAceptacion;
  pAOld: PruebaAceptacion;

  constructor(
    private pruebasAceptacionService: PruebasAceptacionService,
    private confirmDialogService: TnConfirmDialogService
  ) {}

  /**
   * Método encargado de indicar a la Guard que controla la salida del gestor de PA
   * si debe permitir o no la salida del componente en función de
   * si se han producido o no cambios en la PA y de la decisión del usuario.
   * @return { Observable<boolean> } Indica si debe salirse o no del gestor de PA.
   */
  canDeactivate(): Observable<boolean> {
    if (this.pA.equals(this.pAOld)) {
      return Observable.of(true);
    }

    return this.mostrarDialogoConfirmacion();
  }

  mostrarDialogoConfirmacion(): Observable<boolean> {
    return new Observable<boolean>((observer: Subscriber<boolean>) => {
      /**
       * Abre un diálogo con tres botones:
       * Guardar y salir, Salir sin guardar, Cancelar.
       * Cada botón llama a una función que ejecuta la acción correspondiente y
       * emite un nuevo valor (observer.next(true) o observer.next(false)).
       */
    });
  }
```

Figura 55. Clase PamService (simplificada)

4.4. Cronología del TFG

El desarrollo de las interfaces y funcionalidades descritas en las secciones anteriores han sido fruto de cerca de un año de trabajo.

El desarrollo de la solución tomó hasta seis Sprints alineados con el propio desarrollo de Worki, comenzando en el 1.7, pausando durante el 1.8 y 1.9 y reanudándose desde el 2.0 hasta el 2.4 (ambos incluidos).

La Figura 56 es una ilustración que recoge los hitos principales del desarrollo de este Trabajo Final de Grado.

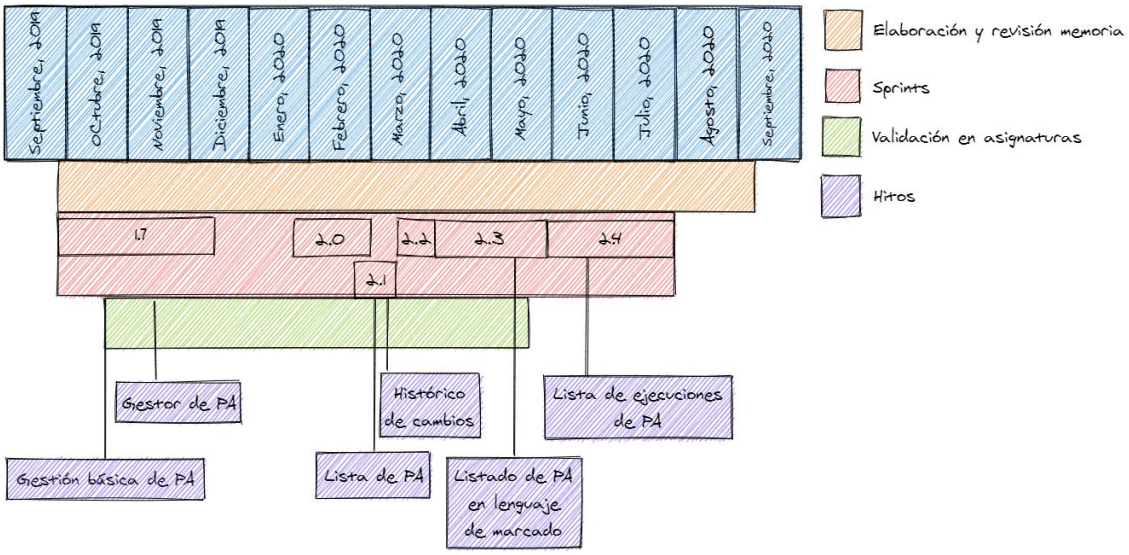

Figura 56. Cronología del TFG

Las horas aproximadas dedicadas a la elaboración del TFG han superado las trescientas:

— Cerca de cincuenta horas de **elicitación de requisitos** y **diseño** repartidas entre múltiples reuniones, elaboración de *mockups* y diagramas.
— Alrededor de ciento ochenta horas de **programación** asociadas a más de cincuenta UT en Worki.
— La **elaboración de la memoria** ha supuesto unas ochenta horas, incluyendo las reuniones de revisión con el tutor.

4.4.1. Sprint 1.7 (23/09/2019 - 05/12/2019)

Desarrollo de la gestión de PA y ejecuciones básica.

Nuevos requisitos:

1. Ampliación de Estructura-Temas y Estructura-Temas del gestor de UT para incluir la Lista de PA asociada a un nodo.
2. Creación de PA.
3. Edición de PA.
4. Eliminación de PA.
5. Desarrollo del gestor de PA (con las consecuentes mejoras en edición de PA y la inclusión de gestión de ejecuciones).
6. Cambiar PA de UT.

Fallos y mejoras:

- Fallo al intentar crear una PA.
- No funciona el filtro de texto de nodos cuando se está posicionado en la pestaña de PA.
- Cuando el Sprint de la UT modificadora o ejecutora es el Backlog, no se muestra en la tabla.
- Añadir diálogo de confirmación al Cambiar PA de UT.

4.4.2. Sprint 2.0 (14/01/2020 - 25/02/2020)

Nuevos requisitos:

7. Mover PA de nodo.
8. Copiar PA a otro nodo.

Fallos y mejoras:

- Mejora en formato y contenido de la tabla de PA.
- Cambiar el control de edición de texto enriquecido de Syncfusion en la definición de la PA por el de DevExtreme.

4.4.3. Sprint 2.1 (17/02/2020 - 04/03/2020)

Nuevos requisitos:

9. Lista de PA asociada a producto.
10. Histórico de cambios en la descripción de la PA.

Fallos y mejoras:

— Al guardar cambios en una PA, recargar la nueva instancia.

4.4.4. Sprint 2.2 (04/03/2020 - 22/03/2020)

El Sprint 2.2 fue exclusivamente dedicado a mejoras en las funcionalidades ya implementadas y a la solución de errores.

Fallos y mejoras:

— Al eliminar una PA, eliminar todas sus instancias.
— Mostrar siempre la última instancia de la PA al acceder al gestor de PA.
— Cambiar encabezado "Contexto" por "UT" en las tablas de PA.
— Al cambiar el nombre de la PA no generar una nueva instancia.
— No permitir añadir una nueva ejecución a una PA si se está en una UT terminada.
— No permitir eliminar una PA si tiene ejecuciones en otras UT.
— No permitir crear, modificar ni eliminar una PA si se está en una UT terminada.
— Advertir de que el tiempo de carga puede ser alto si no hay filtro de producto en Lista de PA.
— Añadir columna "Instancia" en el histórico de cambios de PA.

4.4.5. Sprint 2.3 (22/03/2020 - 07/05/2020)

Nuevos requisitos:

11. Ampliación del gestor de PA para dotar a las PA del sistema de mensajes asociados.
12. Listado de PA en lenguaje de marcado.

Fallos y mejoras:

- Al aplicar un filtro en la tabla de Lista de PA, el listado de PA en lenguaje de marcado debería mostrar sólo las PA filtradas.
- Corregir copiar PA a otro nodo.
- No funciona la generación del listado de PA en lenguaje de marcado si hay agrupaciones por columna en la tabla.

4.4.6. Sprint 2.4 (07/05/2020 - 24/07/2020)

Nuevos requisitos.

13. Desarrollo de la lista de ejecuciones de PA asociadas a producto
14. Deshacer cambio en PA.

Fallos y mejoras:

- Añadir contador de mensajes en la pestaña Mensajes de PA.
- Ampliar la altura de las pestañas del gestor de PA.
- Almacenar configuración de las tablas de PA en caché.
- Añadir opción “Editar” en menú de rayo de lista de PA que abra el gestor de PA.
- Añadir la columna “Observaciones” en Lista de ejecuciones de PA.
- Separar en dos columnas el código y el nombre de la PA en el historial de cambios de PA.

5. Conclusiones y trabajo futuro

Este trabajo final de grado ha sido fruto de casi un año de trabajo. Durante este periodo, se ha diseñado una solución que satisface los objetivos establecidos, se ha desarrollado e integrado en la herramienta Worki y se ha validado su funcionamiento con alrededor de ochenta alumnos del Grado en Ingeniería Informática de la UPV, en las asignaturas PSW y PIN.

A nivel personal, este trabajo ha fortalecido algunas de mis actitudes y aptitudes personales y profesionales (madurez, perseverancia, consistencia) y mis habilidades y conocimientos tecnológicos (diseño, modelado, desarrollo web, JavaScript, TypeScript, Angular…). Además, me ha permitido integrarme en un equipo de trabajo con experiencia donde se me ha permitido aprender y desarrollarme.

El trabajo dedicado a este proyecto comenzó con varias entrevistas y conversaciones con Patricio Letelier Torres, tutor de este trabajo final de grado y líder del desarrollo de Worki. Durante estas primeras conversaciones dimos forma a los requisitos a través de diferentes técnicas (listados, descripciones narrativas, modelos de casos de uso, *mockups* y, sobre todo, PA). Estas técnicas fueron introducidas en la asignatura de Ingeniería del Software de tercer curso y más tarde ampliadas en una asignatura más específica de la rama de Ingeniería de Software: Análisis y Especificación de Requisitos (AER).

Del mismo modo, el desarrollo de la solución mediante Angular requería el conocimiento y la aplicación de patrones de diseño estudiados en Desarrollo de Software (DDS). La importancia de la documentación del código se estudió en Mantenimiento y Evolución del Software (MES) a través de Javadoc (para Java) y se aplicó durante el desarrollo mediante JSDocs (para JavaScript).

La herramienta Worki es empleada en dos asignaturas de la rama de Ingeniería del Software en las que se estudian y aplican procesos y metodologías de software: Proceso del Software y Proyecto de Ingeniería del Software. Durante este trabajo se ha hecho uso de metodologías ágiles de desarrollo y técnicas como TDRE estudiadas y aplicadas en ambas asignaturas, además de haberse probado en ellas la validez de la solución desarrollada durante el curso académico 2019-2020.

La explotación de datos relativos a PA en forma de gráficas fue planteada inicialmente como un posible requisito, pero quedó excluida del alcance de este trabajo debido a la envergadura que el resto de la solución suponía de por sí. No por ello se menosprecia su importancia ni se descarta su inclusión en Worki en versiones posteriores.

Otra funcionalidad muy interesante es la trazabilidad de requisitos a partir de PA. Worki plantea un análisis de impacto en el que las UT y PA afectan a nodos de la estructura de requisitos del producto. Sería interesante para Worki disponer de una herramienta de visualización de relaciones de mayor profundidad entre PA, UT y nodos en forma de grafo dirigido.

6. Referencias

1. Company Bria, M. (2011). Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación. Valencia: Tesis Final de Máster, DSIC, UPV.

2. Koskela, L. (2007). *Test Driven: Practical TDD and Acceptance TDD for Java Developers*. Manning.

3. North, D. (Marzo de 2006). *Introducing BDD*. Obtenido de Dan North & Associates: https://dannorth.net/introducing-bdd/

4. Muñoz Pérez, Á. (2011). *Gestión de Requisitos dirigida por Pruebas de Aceptación*. Valencia: Tesis Final de Máster, DSIC, UPV.

5. Pivotal Tracker. (s.f.). *Pivotal Tracker Help*. Obtenido de https://www.pivotaltracker.com/help/articles/reviews/

6. Huszárik, M. (13 de Marzo de 2018). *Angular JS to Angular*. Obtenido de RisingStack Engineering: https://blog.risingstack.com/angularjs-to-angular-history-and-tips-to-get-started/

7. Savkin, V. (20 de Diciembre de 2016). *Essential Angular - NgModules*. Obtenido de Nrwl Blog: https://blog.nrwl.io/essential-angular-ngmodules-16474ea99713

8. Savkin, V. (27 de Diciembre de 2016). *Essential Angular - Dependency Injection*. Obtenido de Nrwl Blog: https://blog.nrwl.io/essential-angular-dependency-injection-a6b9dcca1761

9. Savkin, V. (3 de Enero de 2017). *Essential Angular - Change Detection*. Obtenido de Nrwl Blog: https://blog.nrwl.io/essential-angular-change-detection-fe0e868dcc00

10. Junker, M. (8 de Octubre de 2019). *Medium*. Obtenido de Medium: https://medium.com/better-programming/zone-js-for-angular-devs-573d89bbb890

11. Letelier Torres, P. (14 de Diciembre de 2011). *Agility at Work*. Obtenido de Agility at Work: http://agilismoatwork.blogspot.com/2011/12/gestion-de-requisitos-agil.html

7. Anexos

ANEXO A: Manual de usuario

Como guía para el manual partiremos de un producto desarrollado durante el curso académico 2019-2020 para la asignatura PIN. La estructura de requisitos de dicho producto, parcialmente colapsada, se muestra en la Figura 57.

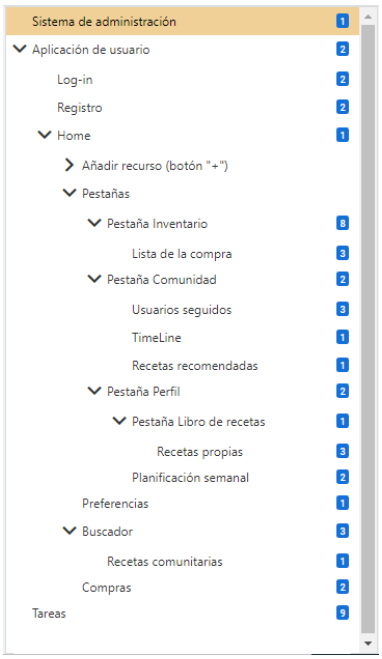

Figura 57. Estructura del producto

En cada nodo de la estructura se identificarán las UT que han afectado al nodo. Cada UT podrá añadir, modificar o eliminar PA de los nodos a los que esté asociada. La Figura 58 muestra la Lista de UT asociadas al nodo “Pestaña Inventario”.

Lista de UT | Pruebas de Aceptación

Buscar...

UT	Línea de trabajo	Sprint	Actividad actual
2248 - Configuración categorías	*RECIPTORY	Backlog	Esperar Prioridad / Sergio Rico Alfonso
2731 - Restricciones al eliminar productos	*RECIPTORY	Backlog	Esperar Prioridad /
2726 - Añadir foto para los productos	*RECIPTORY	Backlog	Esperar Prioridad /
2233 - Gestión fechas caducidad	*RECIPTORY	Sprint 3	Terminada
2786 - Cocinar receta	*RECIPTORY	Sprint 3	Terminada
2729 - Los productos no pueden estar repetidos	*RECIPTORY	Sprint 2	Terminada
2727 - Poder elegir el sitio del inventario al añadir producto	*RECIPTORY	Sprint 2	Terminada
2232 - Gestión Inventario	*RECIPTORY	Sprint 1	Terminada

8 UT

Figura 58. Lista de UT asociada al nodo "Pestaña Inventario"

De la lista mostrada en la Figura 58 (Figura anterior) nos centraremos en la UT "Configuración categorías". El identificador asociado para esta UT es el 2248. La Figura 59 muestra la ficha de dicha UT.

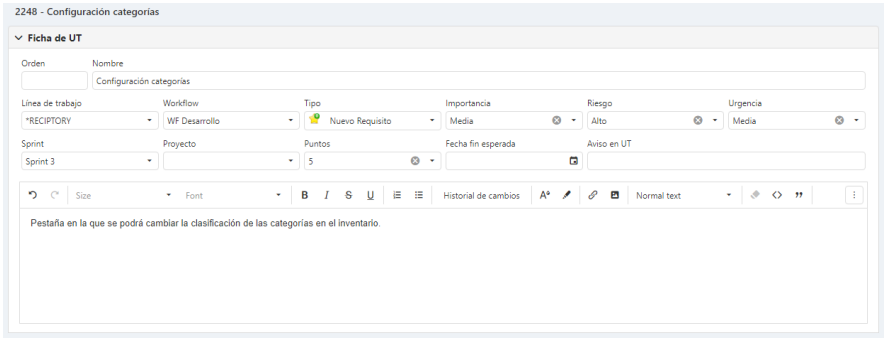

Figura 59. Ficha de de la UT 2248: "Configuración categorías"

Desde la pestaña Estructura-Temas del gestor de UT podemos acceder a la lista de PA del nodo correspondiente al que está asignada la UT, que contiene cinco PA como se muestra en la Figura 60.

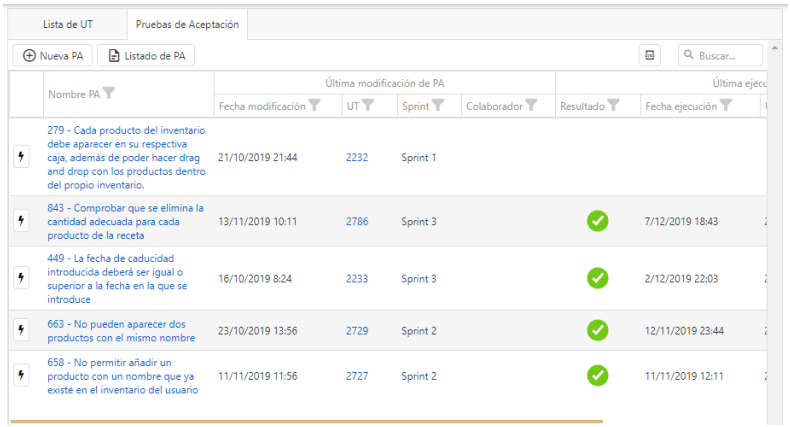

Figura 60. Lista de PA del nodo "Pestaña Inventario"

Desde esta vista, tenemos la opción de crear una nueva PA, como se muestra en la Figura 61.

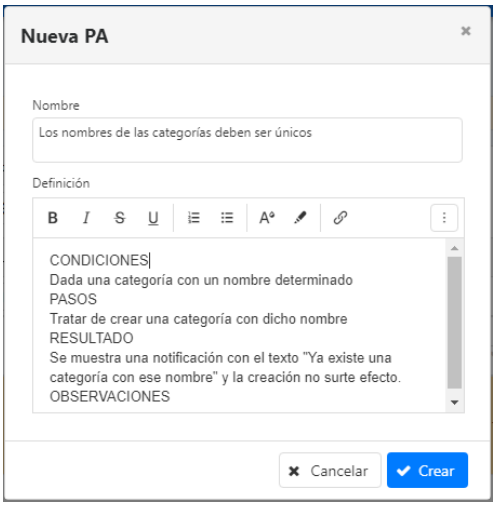

Figura 61. Nueva PA

Como se observa en la Figura 62, al acceder a Estructura-Temas del gestor de UT desde una UT, las PA que dicha UT haya creado o modificado presentarán una medalla azul en la columna de UT del conjunto de columnas “última modificación de PA”. Sin embargo, las PA creadas o modificadas por otra UT no presentan dicha medalla. Tras la creación de la PA introducida en la Figura 61, observamos que se muestra dicha medalla.

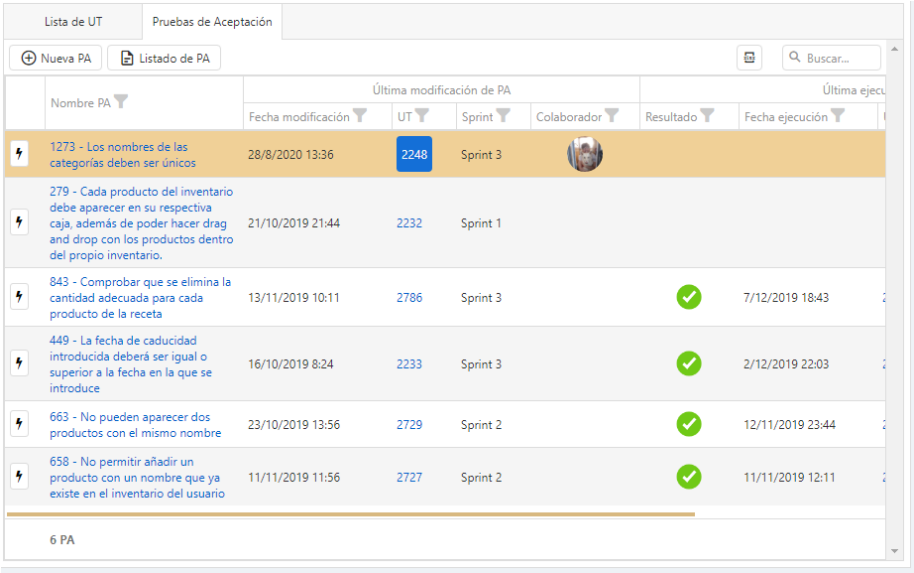

Figura 62. Lista de PA con la nueva PA recién creada

Este mismo comportamiento respecto a la presencia de medalla o no ocurre con el identificador de la UT en el conjunto de columnas “última ejecución de PA”. De esta forma, es sencillo ver qué PA han sido modificadas o ejecutadas por la UT que nos aporta el contexto y extraer información crucial de ello:

— Si una PA ha sido modificada por la UT que nos aporta el contexto, pero no ejecutada, todavía está por aplicar.
— Si una PA no ha sido modificada por la UT que nos aporta el contexto, pero sí ejecutada, ha sido aplicada en regresión.

Desde esta vista (pestaña Lista de PA de Estructura-Temas del gestor de UT) tenemos acceso a múltiples acciones sobre la PA al hacer clic sobre el botón con el icono en forma de rayo, como se muestra en la Figura 63.

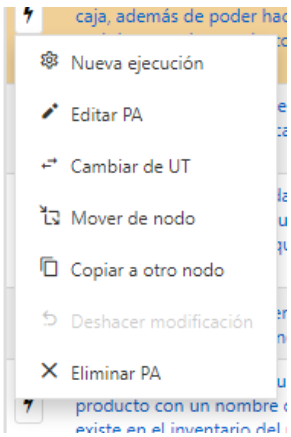

Figura 63. Funcionalidades PA

Haciendo clic sobre el nombre de la PA o sobre el botón Editar PA de la lista de botones de acción sobre la fila, accederemos al gestor de la PA. Dicha vista se representa en la Figura 64.

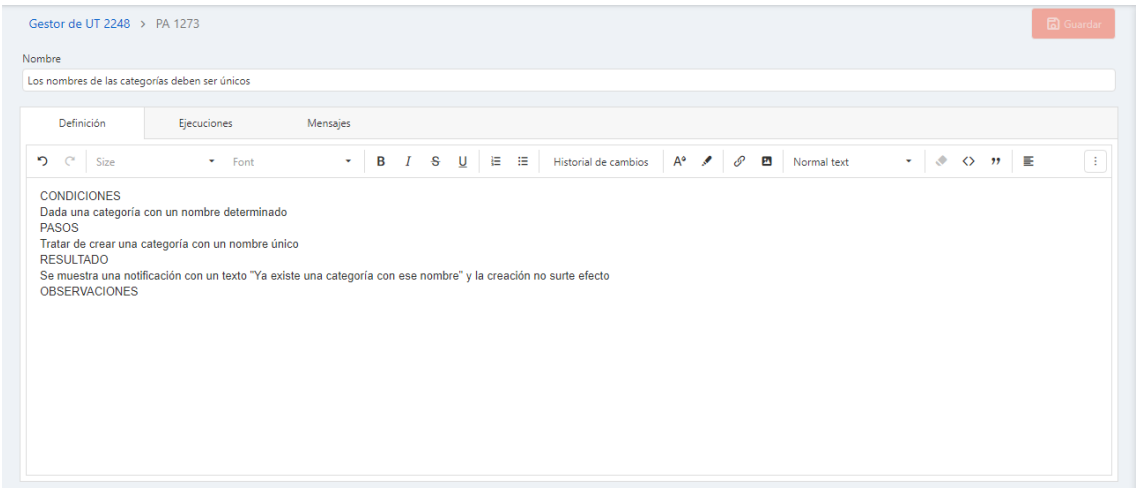

Figura 64. Gestor de PA

Al acceder al gestor de PA desde el gestor de una UT, accederemos en el contexto de dicha UT. Esto garantiza que las modificaciones y ejecuciones registradas en el gestor de la PA se asocien a la UT que aporta el contexto.

Desde el gestor de la PA podremos modificar su nombre. Desde la pestaña de definición del gestor de PA, podremos modificar su definición; modificar la definición de una PA produce una nueva instancia de la PA. El histórico de cambios e instancias de una PA puede ser consultado desde el historial de cambios en la pestaña de definición y se ilustra en la Figura 65.

Figura 65. Historial de cambios

Desde la pestaña ejecuciones de PA podemos crear nuevas ejecuciones de PA, como se muestra en la Figura 66.

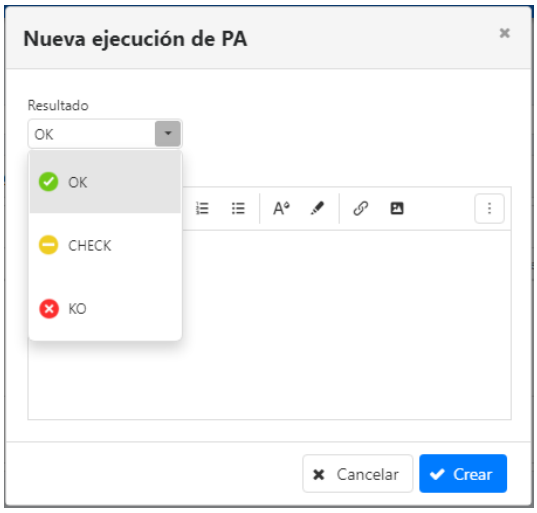

Figura 66. Creación de una ejecución de PA

Además, será posible consultar el listado de ejecuciones de la PA y editar o eliminar la última ejecución realizada (Figura 67).

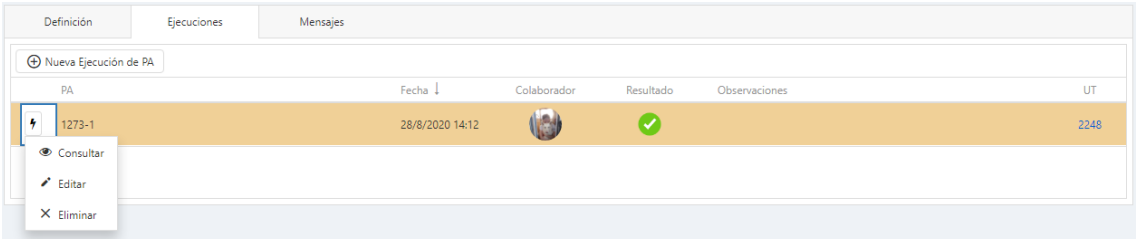

Figura 67. Pestaña "Ejecuciones" del gestor de PA

La pestaña de mensajes, la última pestaña del gestor de PA, nos permite enviar mensajes a otros colaboradores (que participen en el producto) en el contexto de la PA actual (Figura 68).

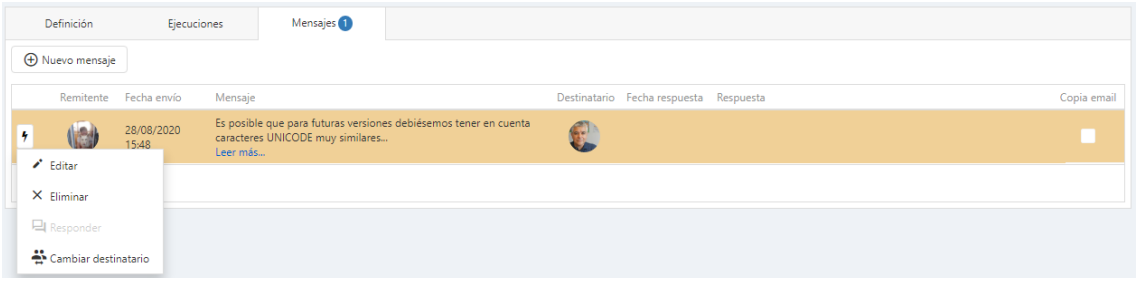

Figura 68. Pestaña "Mensajes" del gestor de PA

Desde el menú de Worki es posible acceder a dos vistas adicionales: Lista de PA y Ejecuciones de PA. Las dos últimas entradas del menú de Worki, ilustrado en la Figura 69, hacen referencia a dichas vistas.

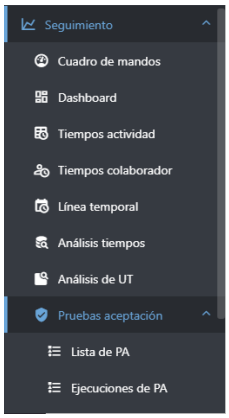

Figura 69. Menú de Worki

Para consultar todas las PA asociadas a un producto, accederemos a la vista Lista de PA, ilustrada en la Figura 70.

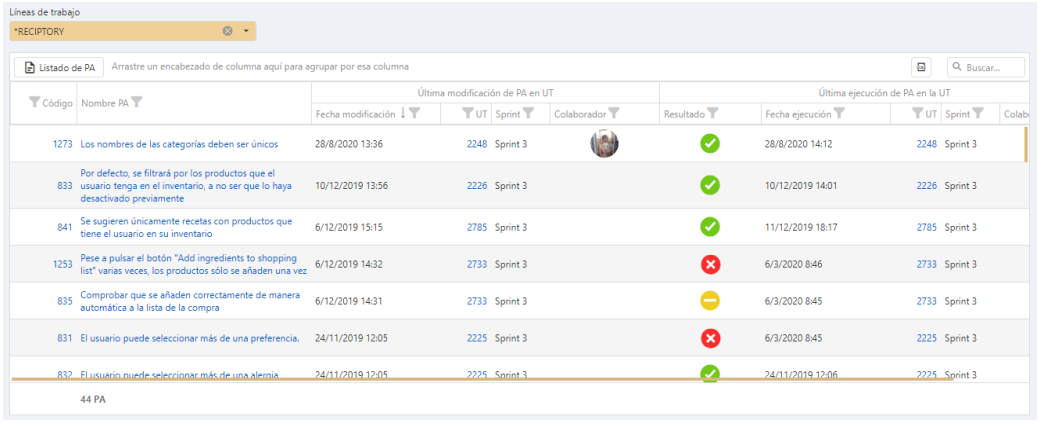

Figura 70. Lista de PA por producto

Para consultar todas las ejecuciones de las PA asociadas a un producto, accederemos a la vista Ejecuciones de PA, ilustrada en la Figura 71.

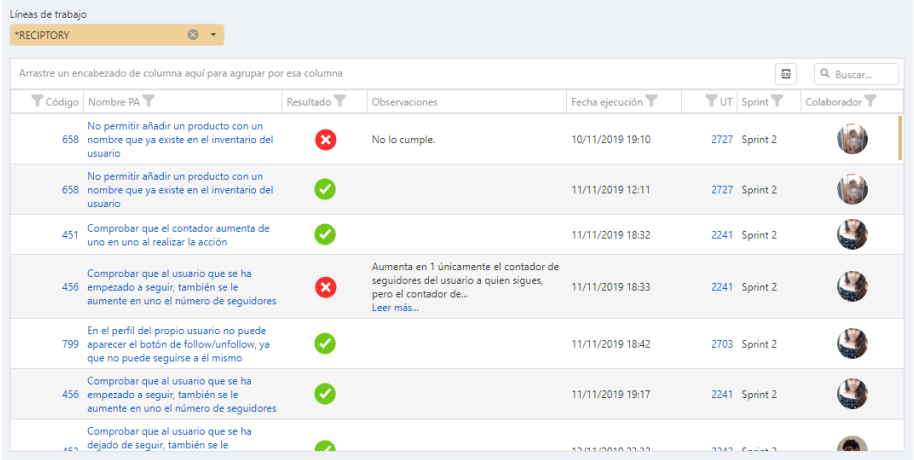

Figura 71. Lista de Ejecuciones de PA por producto