UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación

Autor: Maria Company Bria

Director: Dr. Patricio Letelier Torres

Septiembre del 2011

Trabajo Final de Máster presentado para cumplir con los requisitos finales para la obtención del título de Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información, de la Universidad Politécnica de Valencia, 2011.

Agradecimientos

Este Trabajo Final de Máster es un esfuerzo en el cual directa o indirectamente han participado diferentes personas leyendo, dándome su opinión, corrigiendo, teniéndome paciencia, dándome ánimo, apoyándome en los momentos de crisis y en los momentos de felicidad.

Mi mayor agradecimiento es para mí director de tesis, Patricio, por tantas cosas que no podría enumerar. Entre esas cosas por todo el tiempo que me ha dedicado en estos dos años, por sus sugerencias e ideas de las que tanto provecho he sacado, por el respaldo y la amistad. Le agradezco también el haberme facilitado siempre los medios suficientes para llevar a cabo todas las actividades propuestas durante el desarrollo de este trabajo. Gracias Patricio por preocuparte de que en todo momento sacara de lo bueno, lo mejor.

No me puedo olvidar de mis compañeros y amigos con los cuales he compartido casa, laboratorio y horas de trabajo. Gracias por los buenos y malos momentos, por aguantarme y por escucharme.

También me gustaría agradecer a ADD Informática por haber confiado en mí para llevar a cabo el desarrollo de este trabajo.

Aunque la entrega de este trabajo representa el final de una de las etapas más importantes de mi vida, también significa el inicio de otra que espero sea aún más enriquecedora.

Gracias a todos.

Dedicatoria

A mis padres, los mejores ingenieros de la vida y de quienes no me canso de aprender. Ellos junto con mi hermana son los que siempre se llevan la peor parte de mi trabajo; el cansancio, el mal humor, los problemas, etc… Gracias por vuestra paciencia y porque a pesar de todo, siempre me apoyáis para que pueda realizar con éxito todos mis sueños. Este título también es vuestro logro.

Tabla de Contenidos

Tabla de Figuras

Capítulo 1. Introducción

1.1. Motivación

El Análisis de Impacto de cambios en los requisitos tiene como propósito prever los efectos que tendrá un cambio propuesto, identificando qué elementos se verán afectados y de qué forma. Por ejemplo, cambiar el tipo de dato de un campo en una IU puede conllevar un cambio en un campo de una tabla y esto a su vez puede afectar a todas las otras IUs que presenten dicho campo. Así, la información necesaria para realizar análisis de impacto son las relaciones entre los artefactos del producto, incluyendo cualquier tipo de especificación y en particular las piezas de código fuente y/o elementos de la base de datos.

Conseguir un ratio positivo entre el esfuerzo invertido en trazabilidad y la rentabilización en explotación de dicha información es muy difícil. Por un lado, la recolección y mantenimiento manual de relaciones de trazabilidad entre ellos constituye un gran obstáculo, y por otro, la explotación que usualmente se consigue es muy rudimentaria. Una de las principales motivaciones para mantener trazabilidad en un producto software es explotarla para realizar Análisis de Impacto de cambios en el producto. El estado del arte en Análisis de Impacto en las últimas dos décadas prácticamente no ha cambiado. Si bien existen propuestas del ámbito de investigación, hasta dónde sabemos, la aplicación industrial de Análisis de Impacto es escasa y poco satisfactoria. En este trabajo se presenta un enfoque innovador que gracias a la detección automática de relaciones de interdependencia entre artefactos salva el mayor desafío que presenta la trazabilidad: la recolección y mantenimiento de las relaciones entre artefactos. Aunque nuestro enfoque se restringe sólo al ámbito de artefactos de tipo requisitos y pruebas de aceptación, resulta muy útil en la práctica pues resuelve situaciones usuales en las cuales se necesita conocer el impacto de cambios en un producto software de envergadura. Este enfoque está incorporado en la metodología y herramienta TUNE-UP. Las relaciones entre requisitos son automáticamente registradas y mantenidas actualizadas por TUNE-UP, y se explotan de forma semi-automática para hacer Análisis de Impacto, indicando al equipo de desarrollo aquellos elementos que pudiesen verse afectados por un cambio que se prevé realizar en el producto.

El Análisis de Impacto es especialmente útil durante el mantenimiento del producto, pero también puede ser de ayuda durante su desarrollo [22]. Los resultados del Análisis de Impacto apoyan decisiones en diferentes ámbitos del desarrollo de software:

- **Planificación**: ayuda a comprender el alcance de un determinado cambio, lo cual contribuye a una mejor estimación de esfuerzo y evaluación de la conveniencia u oportunidad de dicho cambio.
- **Definición del cambio**: ayuda en la especificación del cambio para que ésta sea completa y consistente, evitando sorpresas posteriores durante su implementación.
- **Programación**: Disponer tempranamente de información más abstracta del cambio, aunque sea menos detallada, resulta útil para que el programador puede hacer una estimación de su trabajo y tener una primera aproximación a los cambios que enfrentará.
- **Testeo**: conociendo las partes del producto que pueden verse afectadas por un cambio, se puede tomar mejores decisiones respecto de las pruebas de regresión que se deben aplicar después de implementar dicho cambio.

El Análisis de Impacto se basa en las relaciones de trazabilidad existentes entre los artefactos que especifican el producto. El alto esfuerzo invertido en especificación y mantenimiento de dichas relaciones es uno de los principales obstáculos para conseguir un ratio positivo entre el esfuerzo invertido y el aprovechamiento de la información resultante [35]. La técnica tradicional para establecer relaciones de trazabilidad y explotarlas mediante Análisis de Impacto son las Matrices de Trazabilidad [29, 30], en las cuales se cruzan dos tipos de artefactos marcándose las celdas que corresponden a una relación *trace to* o *trace from*. Por ejemplo, se suelen elaborar matrices entre Features y Casos de Uso, Casos de Uso y Clases, Clases y Componentes, etc. En cuanto a la explotación de información, en las Matrices de Trazabilidad se muestran celdas marcadas *Suspect* (sospechosas), cuando se modifica un artefacto del lado *trace from* de una relación. Cuando se utiliza este enfoque para trazabilidad y Análisis de Impacto los artefactos suelen ser de gránulo muy grueso (IUs, Casos de Uso, Clases, Componentes) con lo cual la información de impacto es demasiado general y se necesita un trabajo manual más detallado para conocer con mayor precisión el impacto del cambio.

TUNE-UP es una metodología y herramienta para la gestión ágil de proyectos de desarrollo y mantenimiento de software. TUNE-UP se ha ido refinando en el trabajo diario de una PYME de desarrollo de software desde hace más de seis años. TUNE-UP incluye un enfoque innovador para la gestión de requisitos denominado TDRE [1] (Test-Driven Requirements Engineering), el cual se fundamenta en el hecho de especificar los requisitos mediante Pruebas de Aceptación (PAs), primer punto que se va a explicar en este trabajo. En TDRE un requisito y sus PAs son parte del mismo artefacto, lo cual conlleva una serie de ventajas; no se tiene que hacer ninguna transformación de Requisito a PA, van a ser gestionados por el mismo rol y se van a realizar en la misma actividad. En este contexto, cuando nos planteamos ofrecer un apoyo específico para Análisis de Impacto nos dimos cuenta que podíamos ofrecer una solución novedosa, pragmática, y además con soporte automatizado para la identificación y mantenimiento de las relaciones de trazabilidad. Nuestro propósito no es determinar el impacto de un cambio a través de todos los tipos de artefactos, sino el conocer en detalle el efecto que un cambio en un requisito podía tener en el resto de los requisitos. Es decir, queremos apoyar la especificación del cambio, evitando detectar tardíamente que dicha especificación está incompleta o es inconsistente.

1.2. Objetivos

El objetivo principal de este trabajo es incorporar Análisis de Impacto en el marco de especificación de requisitos TDRE. Para ello, en primer lugar se pretende desarrollar e implantar el enfoque TDRE en el contexto de TUNE-UP y en segundo lugar, definir y desarrollar en el contexto de TUNE-UP una propuesta de Análisis de Impacto de Requisitos en un proceso de desarrollo centrado en Pruebas de Aceptación, es decir, extender TDRE con Análisis de Impacto.

Refinar el ciclo de desarrollo TDD y mejorar la gestión de PAs en el contexto de TUNE-UP son algunos de los objetivos secundarios que se pretenden abordar en este trabajo. Para refinar el ciclo de desarrollo TDD se mejorará la gestión de productos en el grafo de requisitos y para mejorar la gestión de PAs se van a registrar las ejecuciones de las PAs que van a realizar los diferentes roles que participen en el desarrollo. Posteriormente se utilizará esta información para ver más detalladamente el estado de avance de una unidad de trabajo o de una versión. Además cada vez que se incorpore un cambio en el producto se tendrá que identificar que partes del producto (nodos de la

estructura de requisitos) se van a ver afectados por el cambio introducido, esta información se podrá utilizar después cuando se vaya a planificar un versión. También para encontrar fácilmente las PAs de un producto se va a implementar un buscador de PAs que incorporará filtros para encontrar las PAs con diferentes criterios.

1.3. Estructura del trabajo

A continuación se presenta la organización de la memoria:

El Capítulo 1 incluye una introducción a la motivación de nuestro trabajo y presenta los objetivos de nuestra propuesta. Este trabajo tiene como fin presentar un enfoque innovador de análisis de impacto en un proceso de desarrollo centrado en PAs.

En el Capítulo 2 se presenta nuestro enfoque, Test Driven Requirements Engineering. Esta propuesta se basa en que las pruebas de aceptación pasan a ser el hilo conductor del proceso de desarrollo.

El Capítulo 3 explica los aspectos básicos de la metodología TUNE-UP y de su herramienta de apoyo TUNE-UP Software Procces. Después de explicar brevemente cada módulo de TUNE-UP, se presentan las facilidades que ofrece TUNE-UP para el seguimiento de una iteración.

En el Capítulo 4 se completa la metodología TUNE-UP con nuestra propuesta, detallando de forma precisa el proceso dirigido por las PAs y cómo trabajar de esa forma aporta beneficios a la hora de hacer el seguimiento de una iteración.

El Capítulo 5 resume el estudio realizado sobre los trabajos de investigación relacionados con TDRE. Principalmente nos centramos en las dos propuestas que más se acercan a TDRE: Acceptance Test Driven Development y Behaviour Driven Development. Por último finalizamos el capítulo con ejemplos de herramientas que soportan estos enfoques.

En el Capítulo 6 se describe la infraestructura para Análisis de Impacto que se ha implementado en TUNE-UP y da soporte a la especificación de PAs con referencias a entidades.

El Capítulo 7 presenta los dos tipos de Análisis de Impacto que puede ofrecer nuestra propuesta, los cuales hemos denominado Interdependencia Simple y Dependencia Causa-Efecto.

En el Capítulo 8 se resumen los diferentes trabajos de investigación que se encuentran en la literatura respecto de Análisis de Impacto y una breve explicación de algunas herramientas industriales que incluyen Análisis de Impacto entre sus características.

El Capítulo 9 presenta las conclusiones obtenidas a partir de este trabajo y los impedimentos con los que nos hemos encontrado. Además, comenta el trabajo futuro que ha surgido a raíz de este Trabajo Final de Máster.

Capítulo 2. Test Driven Requirements Engineering

El enfoque **Test-Driven Development (TDD)** [12] se basa en que las pruebas deben dirigir el desarrollo del producto software. El TDD se ha aplicado esencialmente en el ámbito de la implementación, particularmente siguiendo el planteamiento "*no escribir código hasta disponer de las pruebas que debe satisfacer dicho código*". El TDD ha tenido un fuerte impulso con las metodologías ágiles, las cuales lo incorporan entre sus prácticas esenciales.

En productos software industriales, cuando se utilizan prácticas de ingeniería de requisitos, éstas mayoritariamente están soportadas por lenguaje natural, lo cual conlleva los ya conocidos inconvenientes relativos a ambigüedad. Sin embargo, la necesidad de validación puede más que las ventajas que puede ofrecer una especificación más formal y rigurosa de los requisitos. El cliente debe poder leer y comprender los requisitos para así poder dar su conformidad respecto de ellos. Las técnicas más populares para especificación de requisitos se resumen en **Casos de Uso** (utilizados en metodologías tradicionales, como RUP [15]) e **Historias de Usuario** (fichas de XP [8] o representaciones similares en otras metodologías ágiles como Scrum [20]). Si bien los elementos Actor (o tipo de usuario) y Requisitos (Caso de Uso o Historia de Usuario) pueden visualizarse y manipularse gráficamente o en fichas, la descripción de cada requisito se elabora esencialmente de forma textual en lenguaje natural.

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

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

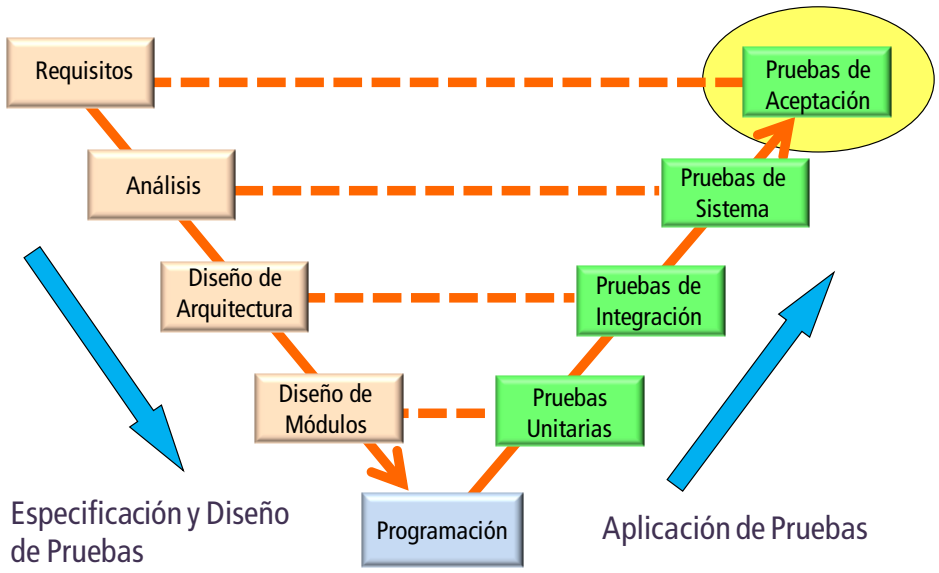


Figura 1. El Modelo en V

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

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

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

Nuestra propuesta se denomina **Test-Driven Requirements Engineering (TDRE)** por referirnos a TDD pero en el ámbito de los requisitos y la planificación. TDRE integra los artefactos, actividades y roles asociados a la especificación y validación de los Requisitos y de las Pruebas de Aceptación. El concepto de Requisito se convierte en un contenedor de Pruebas de Aceptación, y son éstas las que adquieren protagonismo como especificación de cada requisito. Una de las ocho "buenas características" que recomienda la **IEEE 830** [14] para una especificación de requisitos se refiere a que cada requisito debe ser "**Verificable**". En nuestra propuesta los requisitos son verificables pues están especificados como Pruebas de Aceptación.

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

Aunque los requisitos y las actividades asociadas han ido haciéndose espacio entre los aspectos esenciales de un proyecto de desarrollo de software, para las pruebas no ha ocurrido lo mismo y sólo equipos de desarrollo con cierta madurez las integran como pieza imprescindible en sus procesos de desarrollo. Existen dificultades para la introducción de una "cultura", disciplina y prácticas de pruebas en un equipo de desarrollo. Entre los obstáculos o malas estrategias podemos destacar: carencia de un proceso de desarrollo que integre las actividades de pruebas con el resto de actividades, sobrevaloración de la automatización de las pruebas (y particularmente las Pruebas Unitarias) como objetivo inmediato o único, y el poco énfasis en "rentabilizar" el

esfuerzo invertido en pruebas. Lo anterior hace que la iniciativa de incorporación de pruebas además de ralentizarse en cuanto a resultados, vaya perdiendo fuerza en su proceso de implantación en la organización.

En TDRE la estrategia es implantar una "cultura" de pruebas basada en el aprovechamiento de éstas como hilo conductor del proceso de desarrollo. Nuestro enfoque TDRE está incorporado en la metodología **TUNE-UP** [1,2,3,4,5] y en su herramienta de apoyo **TUNE-UP Software Process**.

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

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

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

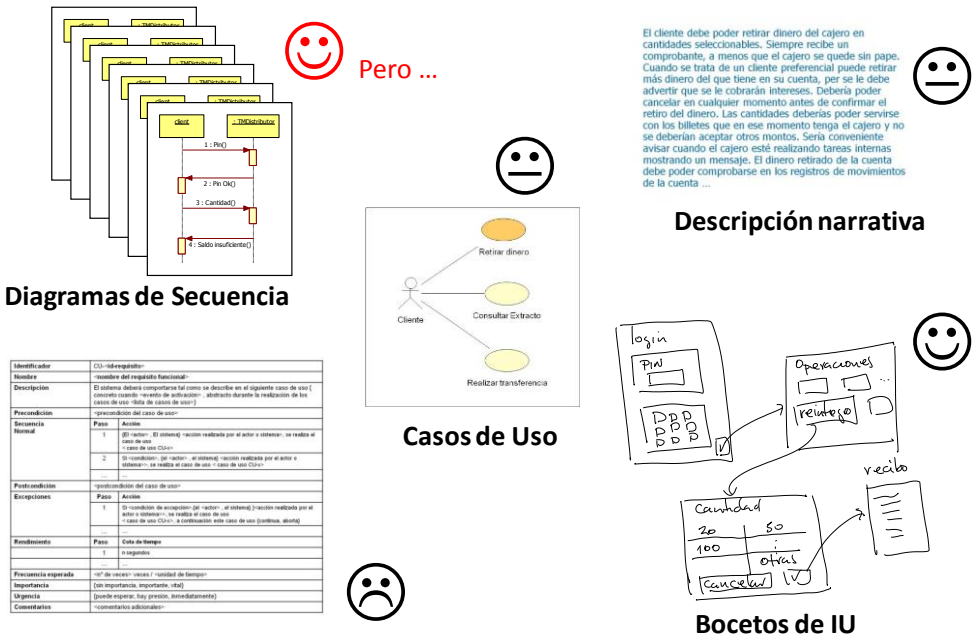


Figura 2. Alternativas populares para la especificación de requisitos

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

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

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

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

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

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

Nuestro enfoque TDRE apuesta por especificar los requisitos usando los elementos que se muestran en la Figura 3: una **breve descripción narrativa** que establece los conceptos y motivación del requisito**, bocetos de la IU** (si procede) y una **lista de Pruebas de Aceptación** (PAs). Estas PAs se refinarán desde simples frases que dan nombre a los escenarios, hasta pruebas diseñadas, listas para ser aplicadas o para automatizarse y aplicarse en regresión. Así, los requisitos actúan como contenedores para las PAs.

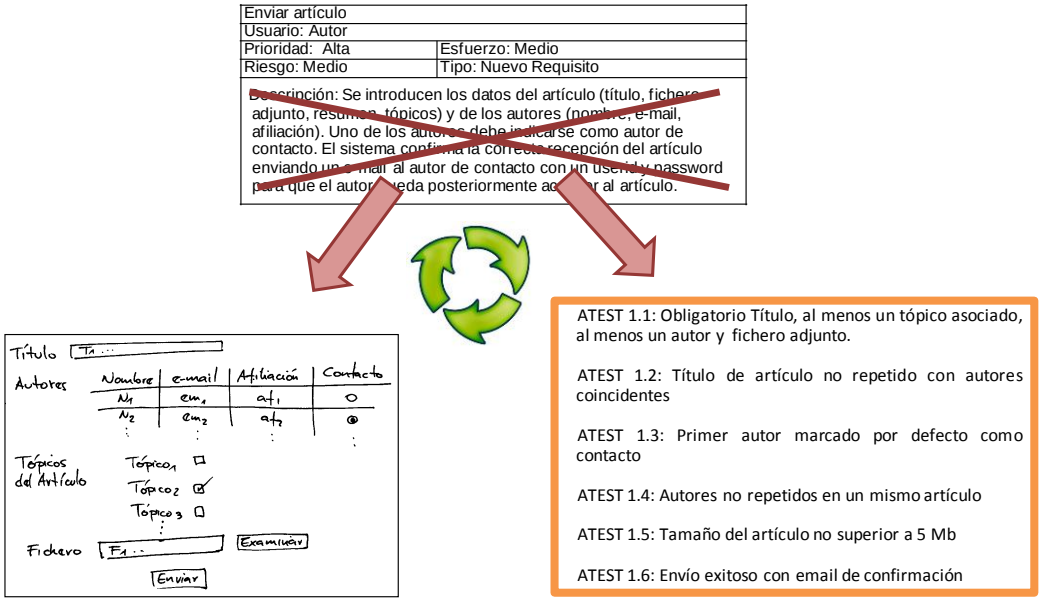


Figura 3. Especificación de requisitos en TDRE

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

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

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

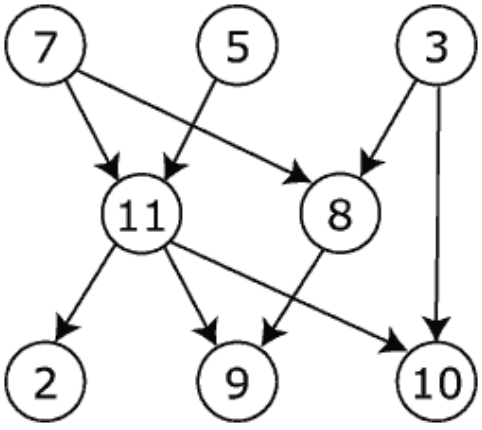
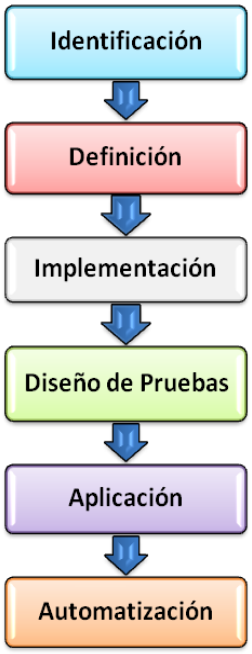


Figura 4. Representación de un grafo acíclico dirigido

La PA cuando va pasando por el proceso de desarrollo software va cambiando de estados, dependiendo de los recursos que disponga la empresa tendremos más o menos agentes interactuando con las PAs a lo largo del proceso. La situación ideal sería tener

al menos tres roles diferentes: el **Ingeniero de Requisitos** (a nivel de empresa conocido como Analista funcional o Analista de Negocio), el **Programador** y el **Tester**. Si por falta de recursos no se tuvieran los tres roles, el mismo agente que analiza podría programar o el mismo que analiza podría testear, lo conveniente es que no testee el mismo agente que ha implementado.

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



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

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

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

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

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

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

A lo largo de este trabajo nos centraremos más detalladamente en la identificación y definición de las PAs, lo cual está más asociado al marco de trabajo del analista. El diseño y ejecución de las PAs es responsabilidad del tester y no lo veremos en detalle.

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

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

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

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

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

- **Condición.** Es opcional, y se utiliza para establecer condiciones previas antes de aplicar los pasos de la prueba. Normalmente se refieren al estado de la BD antes de ejecutar la prueba y/o la navegación necesaria en la IU para localizarse en el punto adecuado para realizar la prueba (cuando no sea obvio)
- **Pasos.** Son las acciones de interacción del actor con el sistema. Cuando son varias acciones, éstas pueden ponerse en una lista numerada. Deben ser simples, del estilo "seleccionar…", "introducir...", evitando hacer referencia explícita a

controles de interfaz o cualquier tecnicismo que dificulte su validación con el usuario

- **Resultado esperado**. Es el efecto de las acciones de interacción del actor. Cada acción puede provocar uno o más resultados. Es importante que cuando se trate de mensajes al usuario se incluya el texto como parte del resultado esperado, así el programador dispone de dicha información ya validada con el cliente
- **Observaciones**. Es un apartado opcional, son recomendaciones que el analista estima conveniente hacerle al tester y/o programador

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

En el Anexo I "Pautas para la definición de PAs" se explican detalladamente las pasos que hay que seguir para especificar PAs correctamente. A continuación se presenta como ejemplo la definición de la PA "*Reintegro saldo < cantidad*".

CONDICIÓN

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

PASOS

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

RESULTADO ESPERADO

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

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

directa con los requisitos, éstas se ven favorecidas porque en TDRE los requisitos están especificados como pruebas. Por otra parte, el tester diseñará, aplicará, y documentará una o más ejecuciones instanciadas (con valores concretos) para cada PA.

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

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

Capítulo 3. TUNE-UP

3.1. Metodología TUNE-UP

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

TUNE-UP se inspira en la esencia de **PSP** (Personal Software Process) [42], donde se destaca que la base del éxito radica en una disciplina de trabajo y productividad individual centrada en la gestión de los compromisos. En cuanto a gestión del tiempo y de priorización del trabajo personal TUNE-UP está alineada con los principios de **GTD** (Getting Things Done)[43] y **The Seven Habits of Highly Effective People** (44). Para la organización y fácil acceso al trabajo de un agente TUNE-UP aporta una innovadora variación de **sistema Kanban**, el cual se integra con un simple pero potente mecanismo de workflows flexibles, permitiendo orquestar de forma automática gran parte de la colaboración necesaria entre actividades. TUNE-UP ayuda en cada momento del proyecto a responder a la pregunta: *¿conseguiré cumplir con los plazos de entrega de mis tareas? o ¿seremos capaces de cumplir con los plazos de entrega al cliente?* Estas simples preguntas inquietan a cualquier gestor o participante de un proyecto. No contar con una respuesta acertada, y sobre todo oportuna, conlleva en la mayoría de los casos graves complicaciones en el proyecto.

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

Después de trabajar tanto con metodologías tradicionales **(RUP y Métrica)** como ágiles **(XP y SCRUM)**, y utilizado muchas herramientas de apoyo para diferentes actividades en el proceso de desarrollo (herramientas para modelado o para pruebas, IDEs para programación, etc.), la principal lección aprendida es que los lenguajes de programación y notaciones, así como sus herramientas de apoyo, aunque contribuyen al éxito de un proyecto no son tan decisivos como lo es el proceso de desarrollo en sí. Sin embargo, siendo las metodologías las protagonistas al respecto, las propuestas actuales continúan

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

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

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

- **Proceso de desarrollo dirigido por las pruebas** (Test-Driven). La definición de una WU es básicamente la especificación de sus pruebas de aceptación acordadas con el cliente. A partir de ahí, todo el proceso gira en torno a ellas, se estima el esfuerzo de implementar el comportamiento asociado y de aplicar las

pruebas, se diseñan las pruebas e implementa dicho comportamiento, y finalmente, se aplican las pruebas sobre el producto para garantizar su correcta implementación.

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

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

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

TUNE-UP es una metodología que incorpora aspectos de metodologías ágiles y de metodologías tradicionales. Las dos primeras características (proceso iterativo e

incremental, y proceso centrado en las pruebas de aceptación) clasifican a TUNE-UP como metodología ágil, sin embargo, las otras características están más próximas de lo que sería una metodología tradicional.

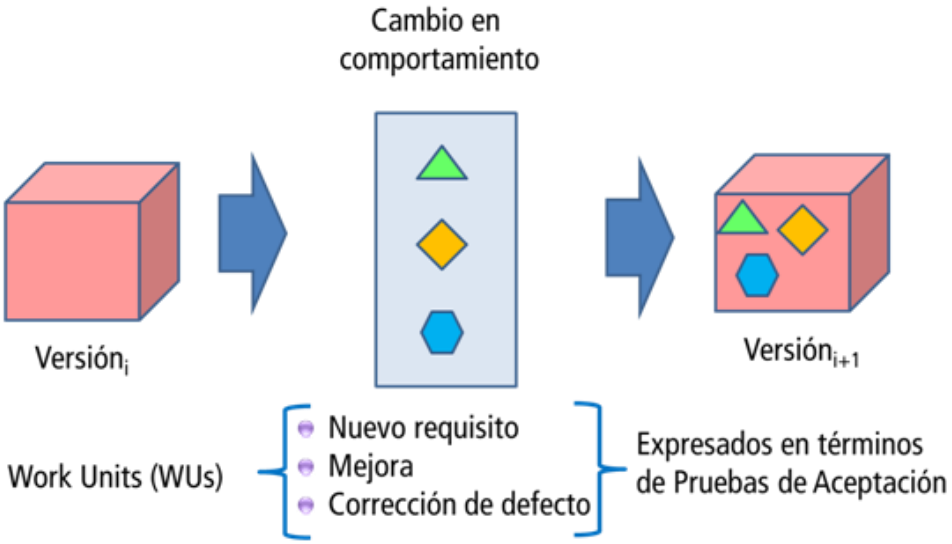


Figura 5. Estrategia Global de TUNE-UP

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

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

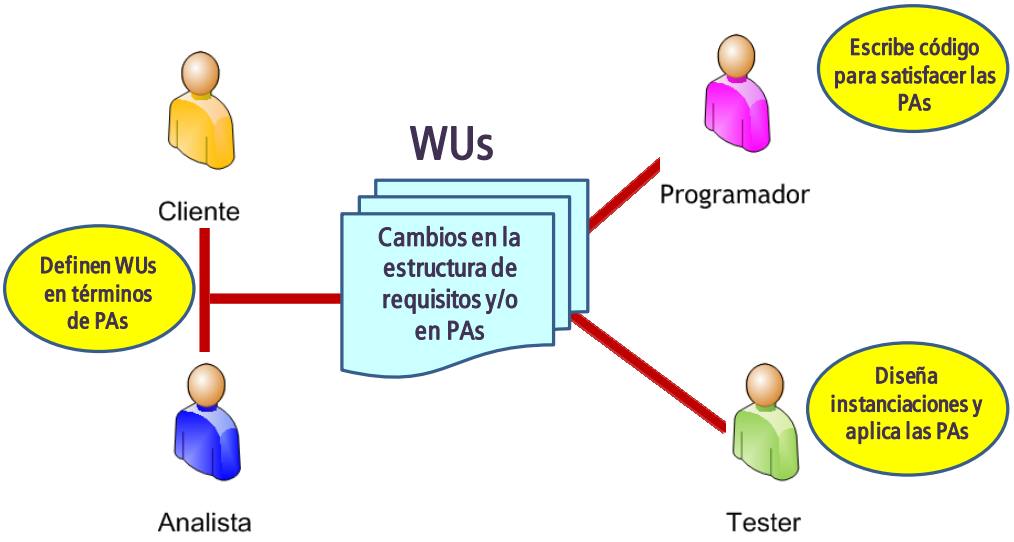


Figura 6. Proceso dirigido por las PAs

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

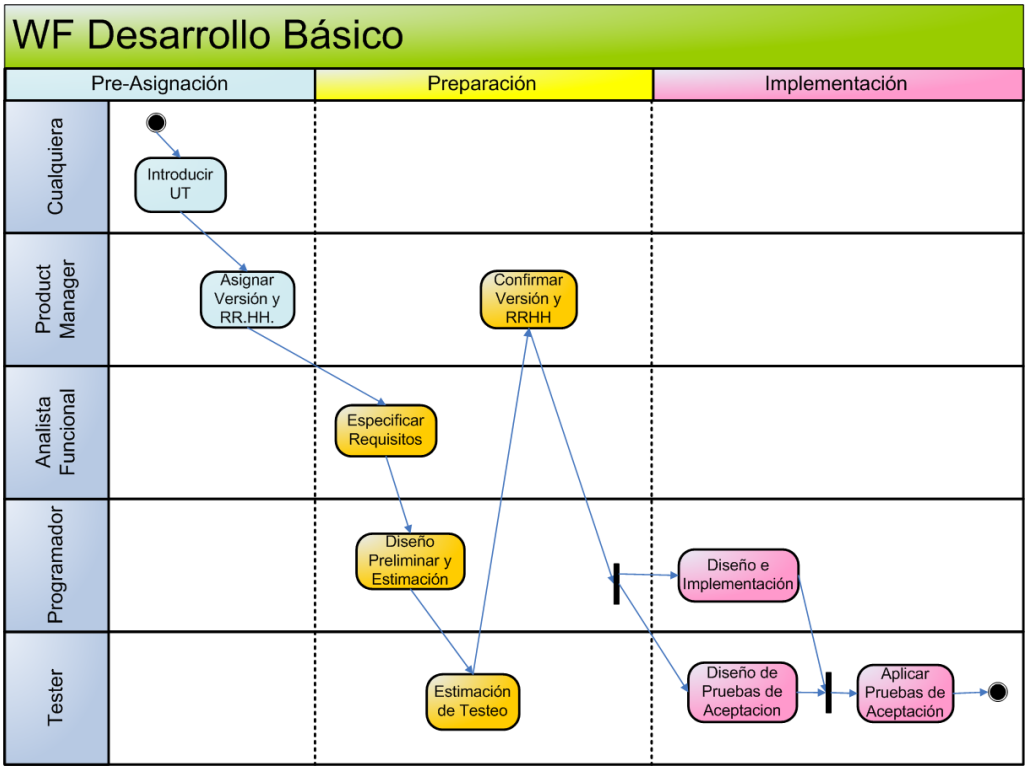


Figura 7. Un ejemplo de Workflow simple

En TUNE-UP se utiliza la siguiente terminología para referirse a comportamiento no deseado del producto: Fallo, Defecto y Error. El **fallo** es el síntoma de comportamiento no deseado que manifiesta el producto en su utilización. El **defecto** es lo que produce el fallo. Los defectos pueden estar en el código del producto o en cualquiera de sus especificaciones, por ejemplo puede que el código esté correcto respecto de la prueba, pero la prueba puede no ser correcta. Finalmente, el **error** es la acción humana (realizada o no realizada por los desarrolladores), es lo que originó el defecto. En la Figura 8 se ilustra cómo en TUNE-UP se lleva a cabo un tratamiento integral de los fallos. Todo fallo se especifica como una PA con un KO contenida en una WU(se verá en detalle en el apartado 4.1). En caso que el fallo se resuelva en una misma WU (sea esta de cualquier tipo), se llevará a cabo el correspondiente re-trabajo (volviendo a realizar el análisis o programación requerida). Si el fallo no es abordado en la WU donde se detecta simplemente si la detección del fallo se hace fuera del contexto de una WU, entonces se creará una nueva WU de tipo fallo que se llevará a cabo en la versión actual o en alguna versión futura.

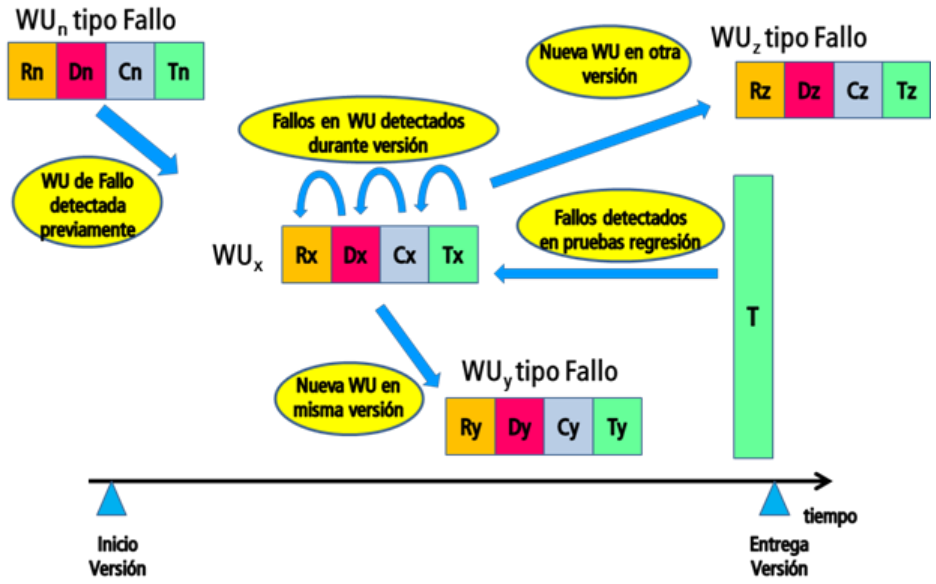


Figura 8. Gestión de Fallos en TUNE-UP

3.2. Herramienta Tune-UP Software Process

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

Personal Planner (PEP)

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

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

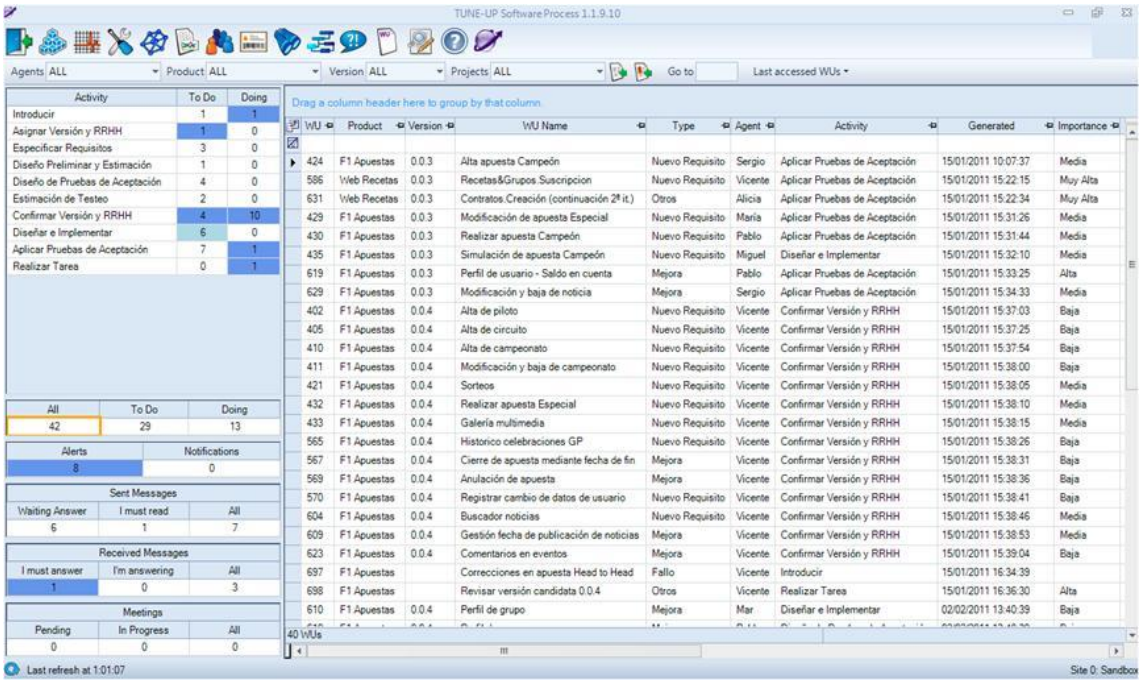


Figura 9. Personal Planner (PEP)

Work Unit Manager (WUM)

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

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

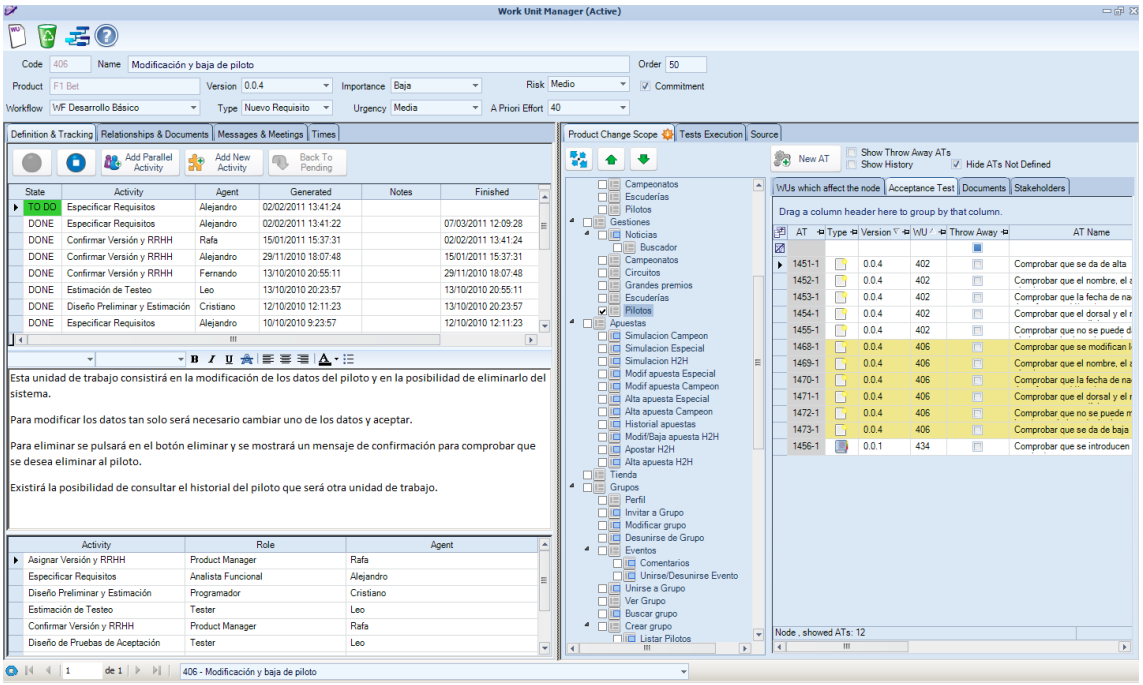


Figura 10. Work Unit Manager (WUM)

Requirements Manager (REM)

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

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

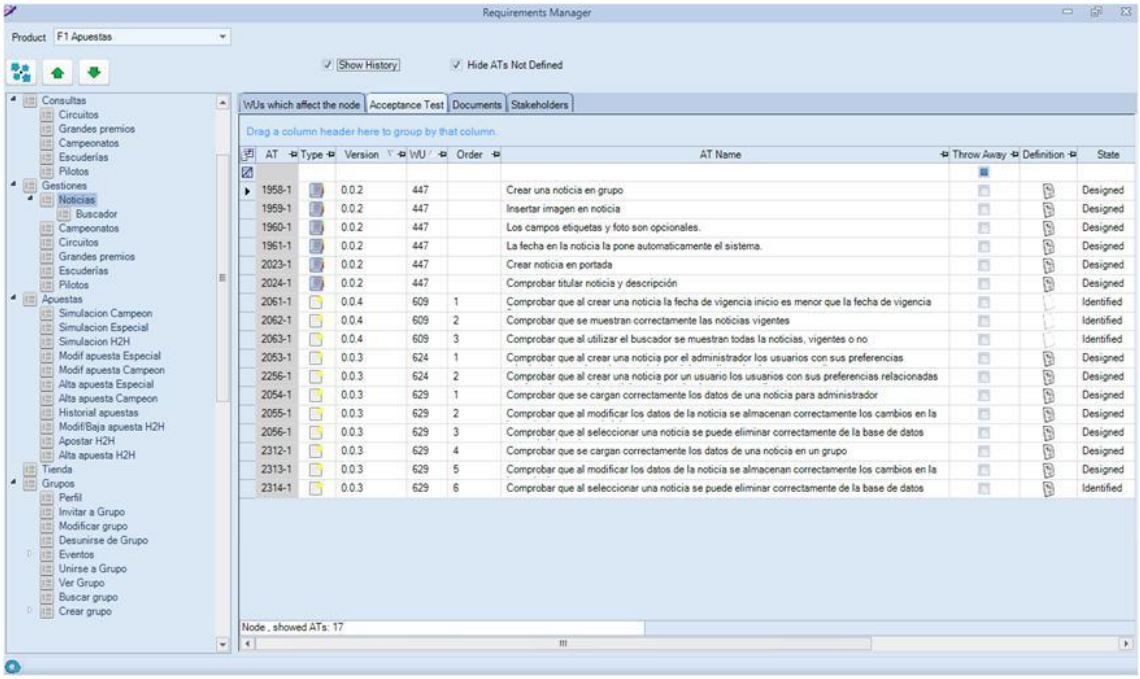


Figura 11. Requirements Manager (REM)

Version Contents & Tracking (VCT)

En este módulo encontramos facilidades para la planificación y seguimiento de las versiones. En la Figura 12 se muestra el contenido de **la pestaña 'Agent Workload'**, en ella se puede conocer en cualquier momento la holgura simple de los agentes respecto de sus actividades en una versión del producto. En esta interfaz se ofrecen potentes mecanismos de filtros y agrupaciones por columnas. Cuando una versión tiene problemas de holgura, en esta misma interfaz el **Product Manager** puede cambiar el agente asignado a la actividad (para balancear la carga de un agente) o cambiar de versión alguna unidad de trabajo. Otras alternativas son modificar la fecha de término de la versión, asignar más recursos humanos a la versión o dividir unidades de trabajo para realizarlas incrementalmente en varias versiones. Además disponemos de otras pestañas que tienen gran utilidad a la hora de planificar una versión como **'Relationships'**, **'WUs in Version'** y **'Affected Requirements'**, las dos últimas las describiremos con detalle en el apartado 4.3 del Capítulo 4.

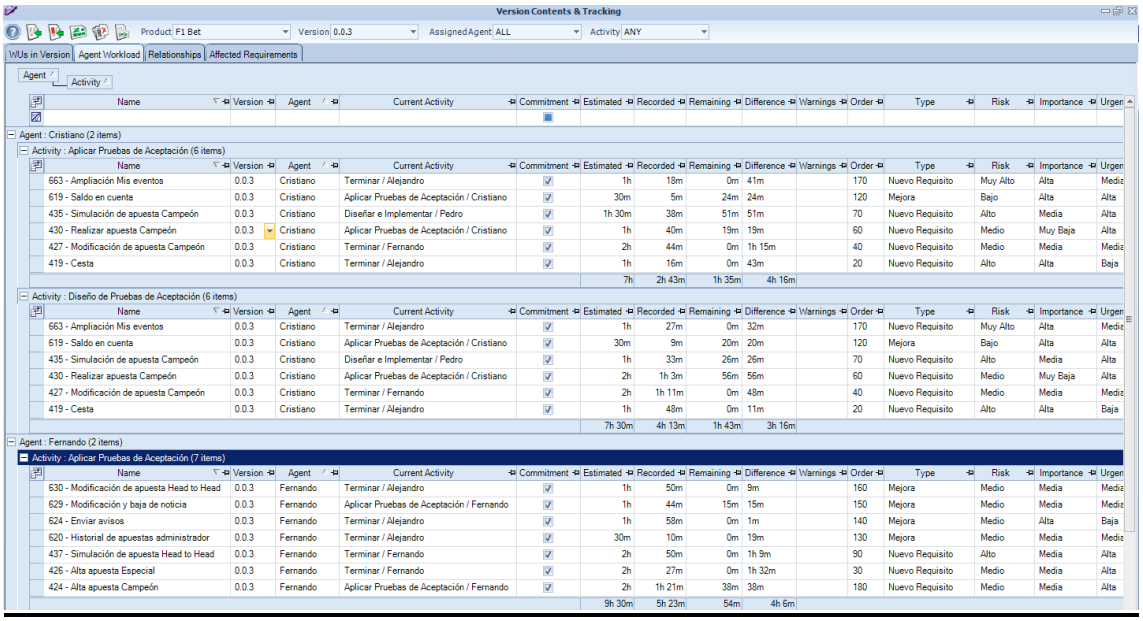


Figura 12. Versión Contents & Tracking (VCT)

3.3. Seguimiento de una iteración con TUNE-UP

En TUNE-UP el seguimiento de una iteración incluye los siguientes mecanismos de apoyo:

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

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

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

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

- **La Gráfica WUs Finished/Unfinished** (lateral inferior izquierdo de la figura 13) muestra por día las Wus que tenemos finalizadas y las que no están finalizadas que cumplen los filtros que se encuentran en la parte superior de la figura 13.
- **La gráfica de estado de PAs** (lateral inferior derecho de la figura 13) ilustra el estado de las PAs para cada nivel de testeo de las Wus que cumplen los filtros que se encuentran en la parte superior de la figura 13 (esta gráfica la veremos más en detalles en el apartado 4.3)

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

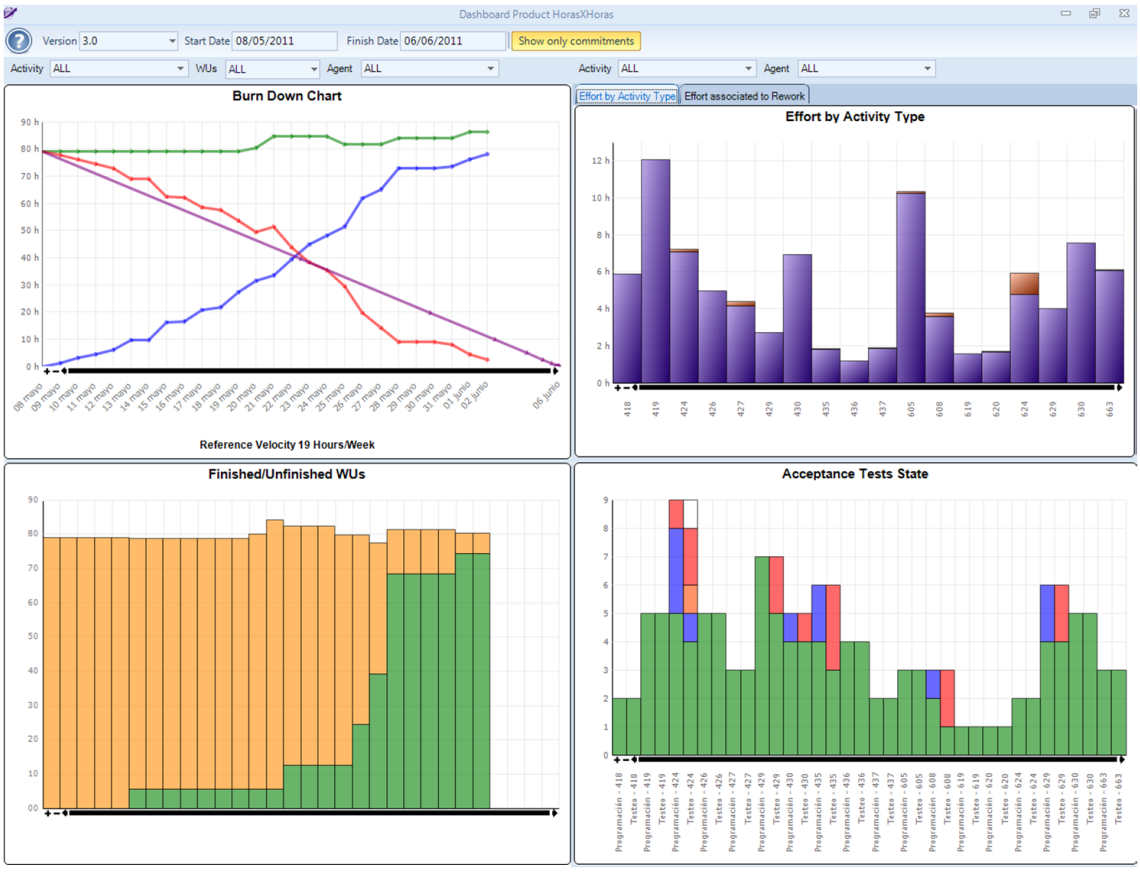


Figura 13. Dashboard asociado a un producto

La **Gráfica Burn Down de TUNE-UP** (lateral superior izquierdo de la Figura 13) incluye dos gráficas en una; una gráfica básica **Burn Down** (asociada a la línea serpenteante descendente) y una gráfica **Burn Up** (asociada a la línea ascendente

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

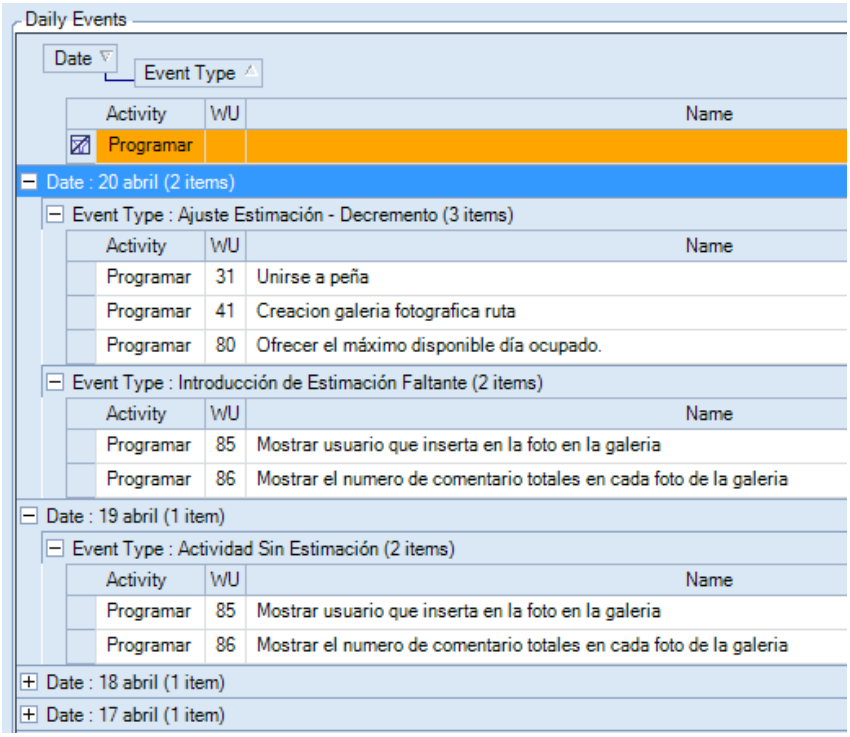


Figura 14. Tabla Daily Events

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

la gráfica de la Figura 13 se despliega en la tabla de la Figura 14 la lista de eventos ocurridos entre el día previo y el día seleccionado. Para cada evento se indica el miembro del equipo, la actividad e información de la WU donde se produce. En TUNE-UP se supervisan todos los eventos que pueden influir en la correcta interpretación del esfuerzo restante, dichos eventos se describen a continuación:

Eventos que invalidan la lectura del esfuerzo restante:

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

Eventos que provocan una variación en el esfuerzo restante observado:

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

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

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

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

Capítulo 4. TDRE en Tune-Up

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

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

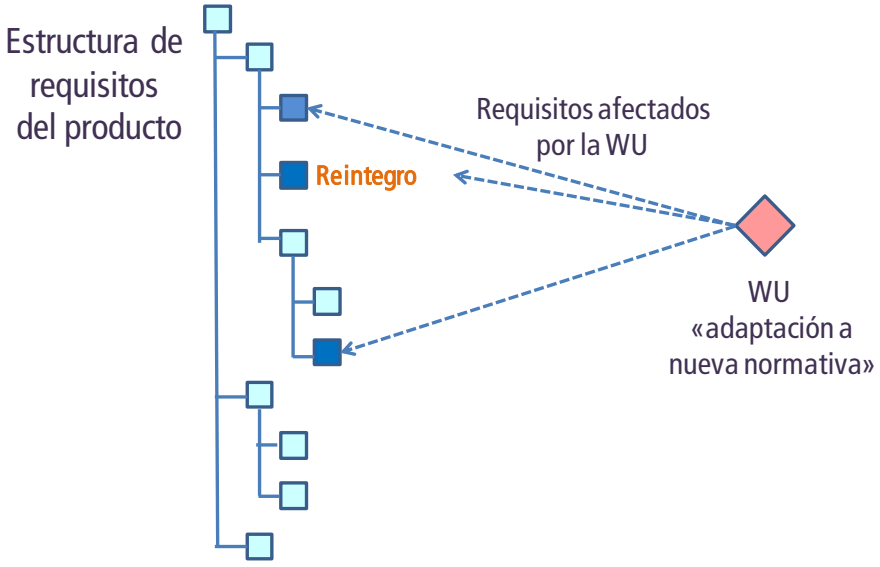


Figura 15. Estructura de Requisitos del producto

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

4.1. Proceso dirigido por las Pruebas de Aceptación

El primer paso a realizar cuando el equipo de desarrollo o el cliente quieren hacer algún cambio de comportamiento dentro del producto es crear la WU correspondiente. Las WUs se pueden crear desde diferentes sitios en TUNE-UP (PEP, WUM y REM). El contenido del REM es muy similar al de la **pestaña Product Change Scope** dentro del

WUM, la única diferencia es que el REM está orientado a consultar el comportamiento pasado, actual y previsto del producto sin estar en el contexto de una unidad de trabajo determinada. Esta información es muy útil en el momento de proponer un nuevo cambio en el producto para por ejemplo, evitar solapes e inconsistencias entre cambios o programar de forma racional futuros cambios del producto prestando atención al conjunto de cambios pendientes de implementar.

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

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

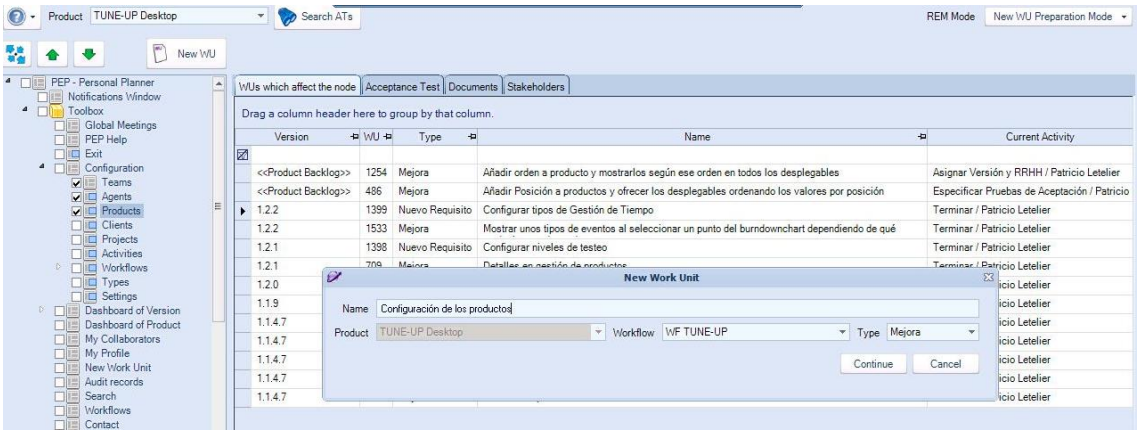


Figura 16. Creación de una WU desde el REM

En TUNE-UP el módulo WUM apoya a los agentes en sus actividades sobre una determinada unidad de trabajo. Desde este módulo y para cada unidad de trabajo, se puede acceder a la **pestaña Product Change Scope** (ver figura 16), donde el analista revisaría las marcas de los nodos que se verán afectados y si es necesario tendría que

modificarlas (podría darse el caso que el agente que ha creado la WU no conociera suficientemente la estructura de requisitos y se haya dejado nodos por marcar o haya puesto las marcas en niveles superiores). Una vez las marcas ya sean las correctas el analista se encargaría de definir los cambios en las PAs de dichos nodos.

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

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

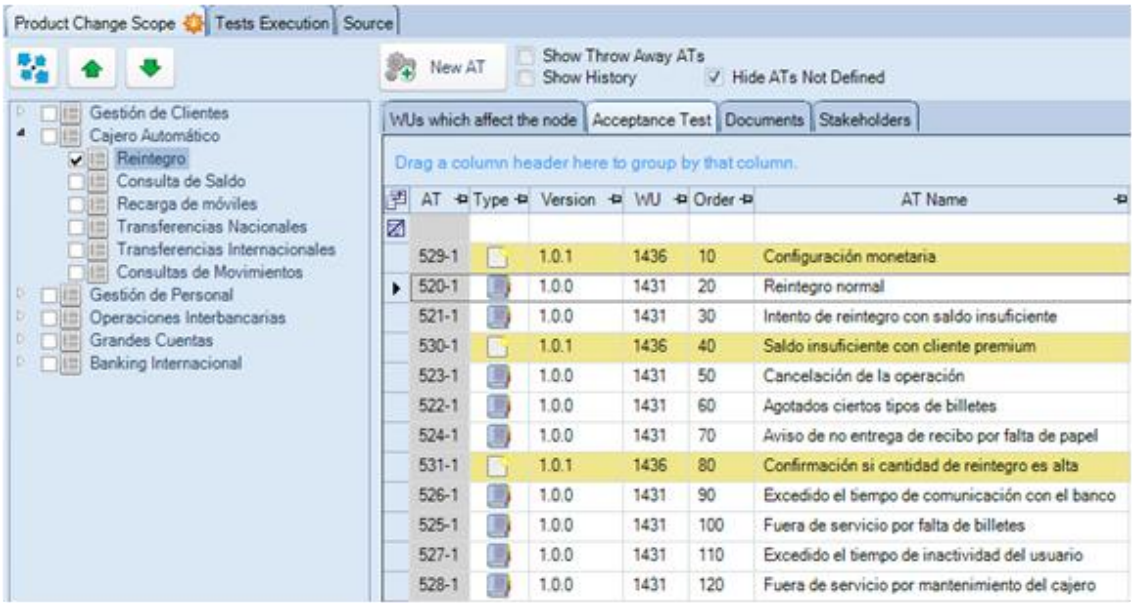


Figura 17. Pestaña Product Change Scope

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

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

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

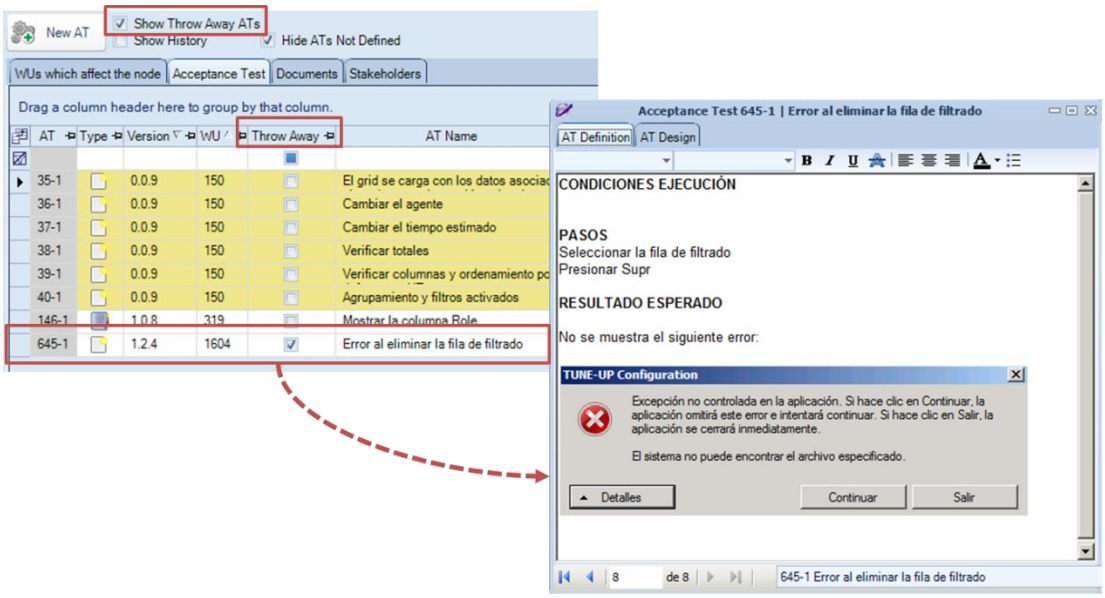


Figura 18. Ejemplo Prueba Throw Away

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

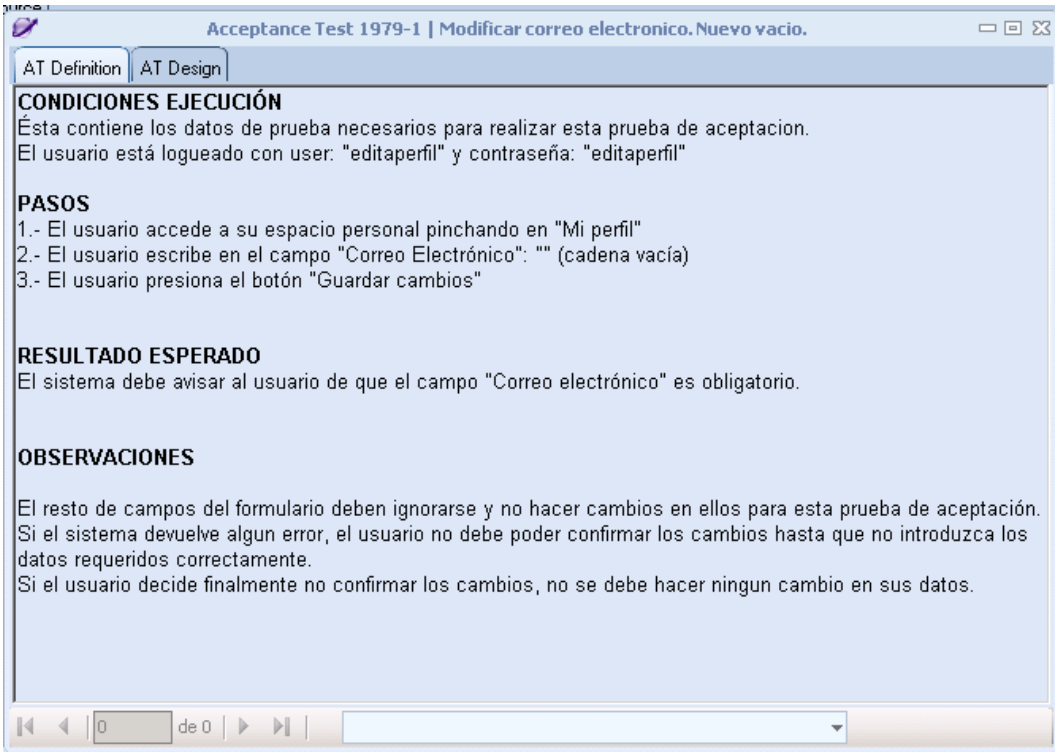


Figura 19. Formulario de Acceptance Test

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

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

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

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

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

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

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

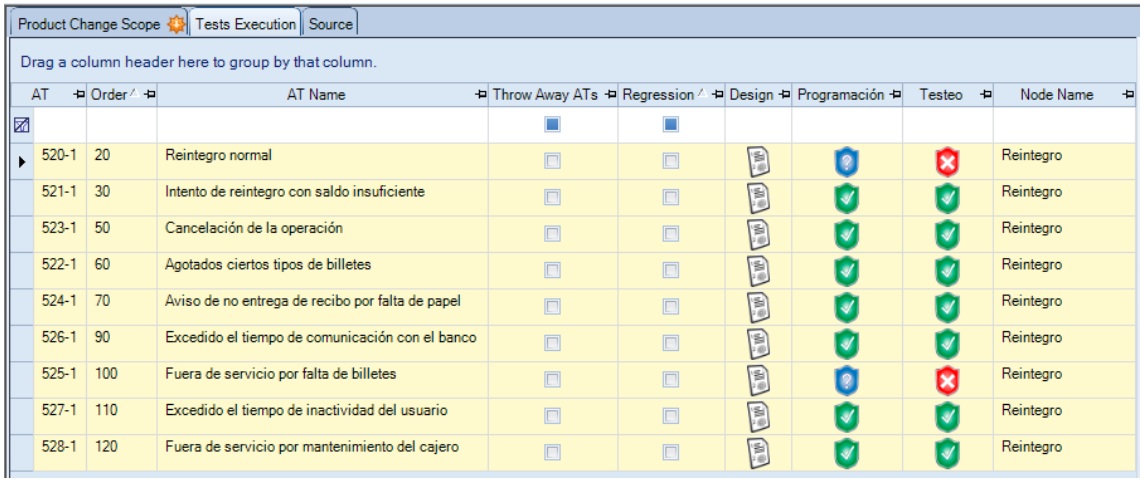


Figura 20. Pestaña Test Execution

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

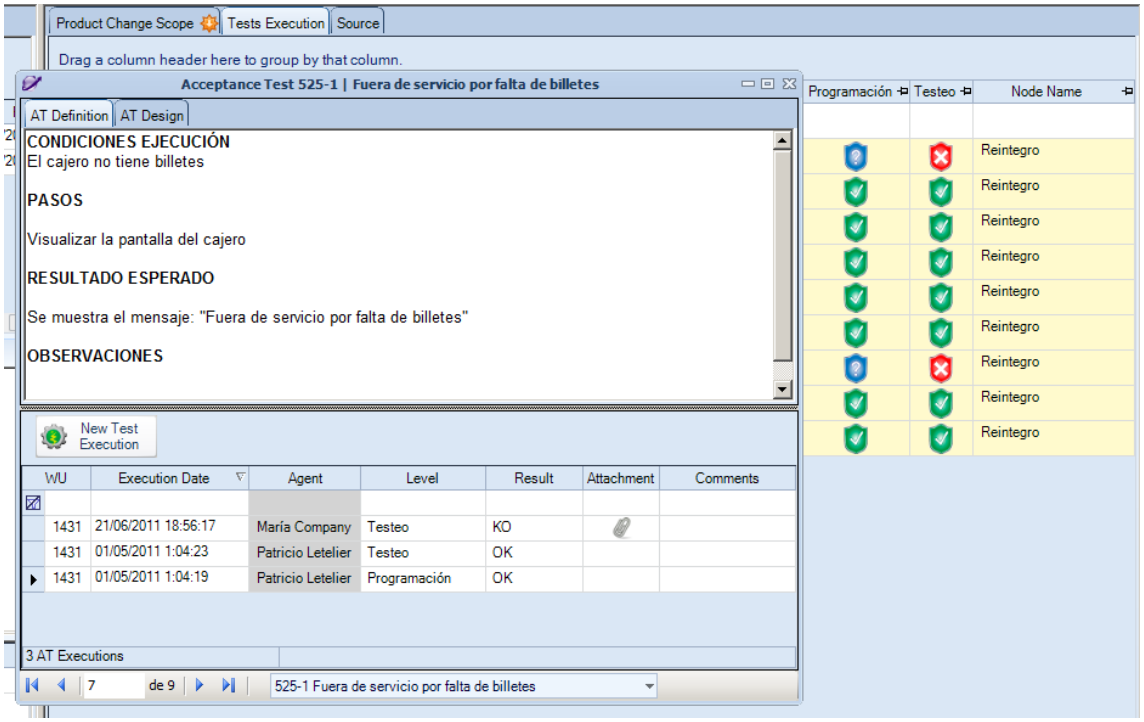


Figura 21. Ejecución de la PA

4.2. Pruebas de Aceptación y referencias a entidades

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

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

En TUNE-UP la definición de una PA se compone de tres apartados: Condiciones, Pasos y Resultado. El cuerpo de cada apartado se escribe en lenguaje natural pero referenciando entidades del modelo de dominio del producto. Esto nos permite establecer automáticamente relaciones entre PAs que tienen referencia a las mismas entidades. Una entidad puede contener atributos o relaciones con otras entidades, el modelo de dominio que vamos a utilizar en este trabajo se muestra en la Figura 22.

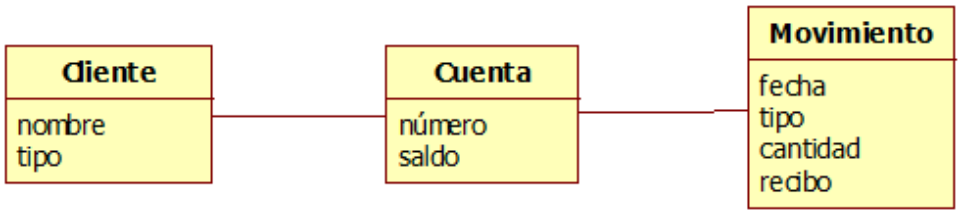


Figura 22. Modelo de dominio de ejemplo

En la Figura 32 se muestra el ejemplo de la PA anterior pero esta vez modificando parte de su contenido para incluir las referencias a entidades del modelo de dominio. En el Capítulo 6 se mostrará cómo se referencian las entidades en la definición de una PA con

la ayuda de **intellisense**, el cual nos ofrece facilidades para navegar por la estructura del modelo de dominio, introduciendo las referencias correspondientes.

En la Figura 23 se muestra un diagrama que ilustra las relaciones entre el atributo Saldo de la entidad Cuenta y las PAs de los nodos Reintegro, Gestión de Clientes y Transferencia. La relación entre una PA y una entidad surge cuando en la especificación de dicha PA, en alguno de sus tres apartados, se ha referenciado dicha entidad.

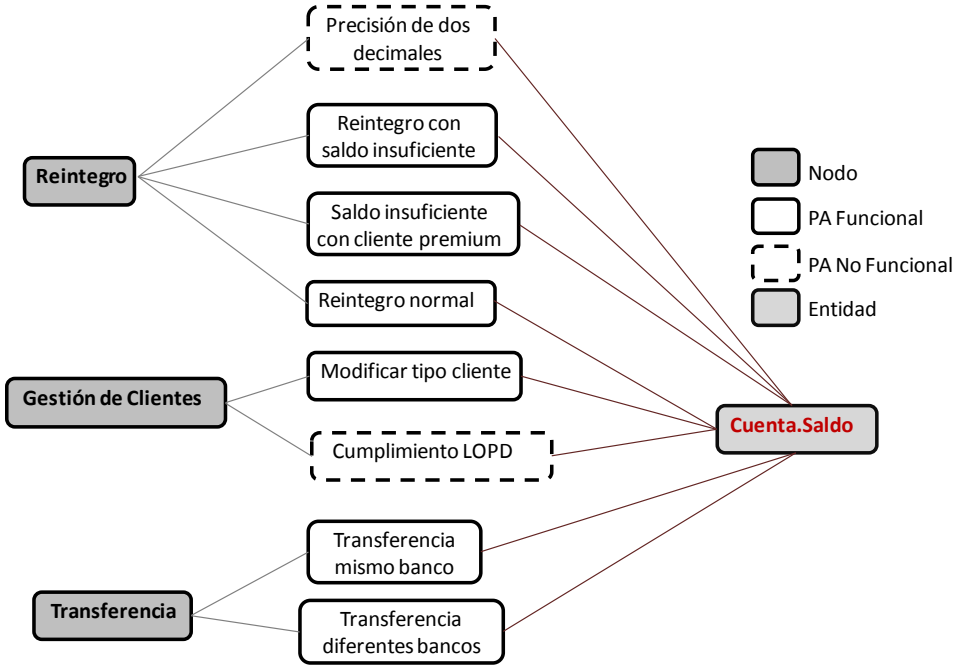


Figura 23. Representación relaciones PAs, Nodos y Entidades

4.3. Seguimiento de una iteración con Tune-Up mediante PAs

En TUNE-UP, para realizar la planificación y seguimiento de las iteraciones, el jefe de proyecto utiliza el **módulo Version Content & Tracking**. La Figura 24 muestra parte de la **interfaz llamada "Version Content",** la cual contiene la lista de unidades de trabajo de la versión de un producto. El jefe de proyecto puede consultar los datos resumidos de cada unidad de trabajo en la versión, en particular, puede conocer la actividad actual en que se encuentra dentro del workflow, su orden dentro de la versión, el esfuerzo que implica elaborar la unidad de trabajo, el agente asignado a las actividades principales del workflow, etc. Además, por cada nivel de testeo existirá una columna que refleja el estado de aplicación de las PAs de cada unidad de trabajo, mostrando el porcentaje obtenido a partir del número de pruebas con estado de aplicación OK dividido por el total de PAs. Con toda esta información el jefe de

proyecto tiene una visión más detallada del estado de avance de la unidad de trabajo. Esta vista es muy reveladora, lo normal es que los porcentajes de cada nivel se vayan completando de forma secuencial, puesto que la unidad de trabajo va pasando por diferentes actividades realizadas por agentes diferentes. Sin embargo, cuando aparecen las inevitables situaciones de re-trabajo, éstas quedan en evidencia por la forma incompleta que presentan los porcentajes de pruebas OK en cada nivel de testeo. La línea subrayada en la figura 24 es un caso frecuente de retrabajo generado por saltos atrás en el proceso, en este caso en concreto el tester ha empezado a aplicar las PAs de la WU pero ha detectado fallos (ha marcado una prueba con KO) y por lo tanto le ha devuelto al programador la WU a la actividad Diseñar e Implementar para que solucione la prueba que tiene marcadas con interrogante.

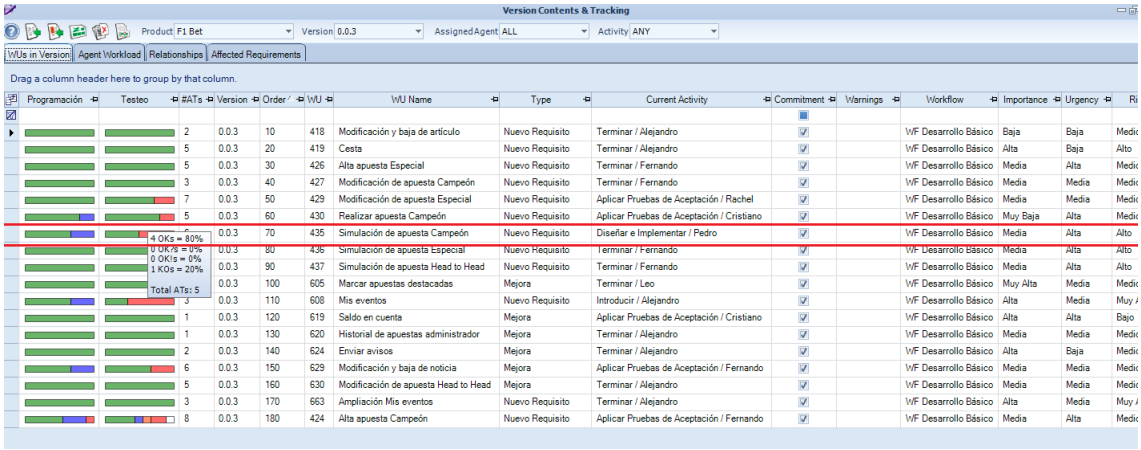


Figura 24. Pestaña Wus in Version del VCT

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

El jefe de proyecto también puede disponer del estado de avance de las PAs de una versión con la **gráfica de estado de PAs** que se encuentra en el Dashboard. En la gráfica de la Figura 25 la línea vertical representa el número de PAs de una WU y la línea horizontal tiene una barra para cada nivel de testeo de una WU. En el ejemplo de la figura sólo tenemos dos niveles de testeo, por lo tanto cada WU tiene representado el estado de las PAs en los niveles de programación y testeo. Esta vista ofrece una visión

en forma de gráfica y por lo tanto más fácil de interpretar que las columnas Programación y Testeo de la figura 24. La única diferencia es que en VCT sólo se puede ver el estado de PAs de WUs de una versión determinada y en la gráfica de estado de PAs se pueden configurar, a petición del usuario a través de filtros, las WUs que se quieren mostrar en la gráfica.

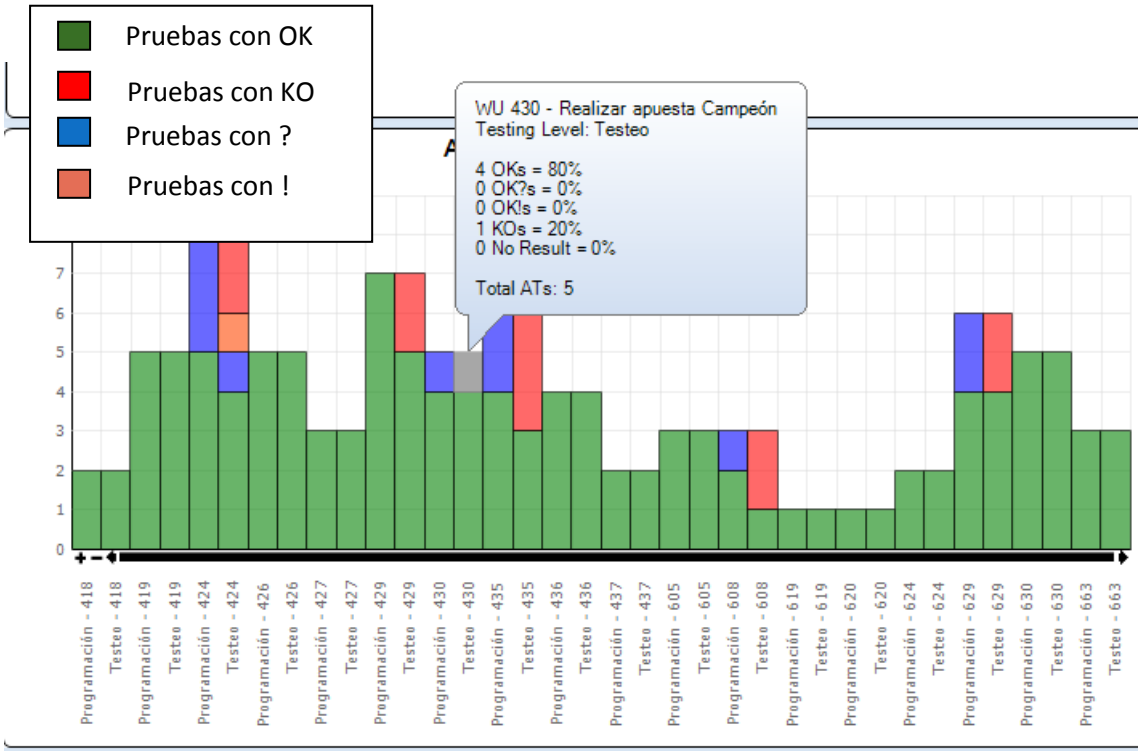


Figura 25. Gráfica de Estado de PAs

La pestaña **Affected Requirements** (Figura 26) es de gran utilidad para el jefe de proyecto ya que permite detectar posibles conflictos o solapes entre las WUs contenidas en la misma iteración o entre WUs de diferentes iteraciones. En la estructura del producto aparecen seleccionados los nodos que se ven afectados en incidencias de la versión y entre paréntesis delante del nombre del nodo el número de WUs de la versión que modifican su comportamiento. Al activar un nodo de la estructura de requisitos (panel de la izquierda), en el panel de la derecha se cargan las WUs que afectan a dicho nodo en la versión (WUs con fondo oscuro) y en otras versiones (WUs con fondo blanco). Esta información ofrece las siguientes ventajas:

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

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

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

Figura 26. Pestaña Affected Requirements de VCT

Capítulo 5. Trabajos Relacionados TDRE

En TDD las pruebas unitarias son definidas por el programador y no están necesariamente relacionadas con los requisitos. Existen dos enfoques cercanos a TDRE que también han llevado la filosofía de TDD al terreno de los requisitos, estos son Acceptance Test Driven Development (ATDD) y Behaviour Driven Development (BDD), los cuales describiremos a continuación.

5.1. Acceptance Test Driven Development(ATDD)

ATDD [9] es una evolución de TDD, orientada a resolver la falta de comunicación con el cliente y abordar adecuadamente los cambios que pueda introducir a lo largo del proceso de desarrollo. Según Koskela, en [37], el TDD puro ayuda a los desarrolladores de software a producir código de mejor calidad y facilidad de mantenimiento. El problema es que los clientes rara vez se interesan en el código. Ellos quieren que los sistemas sean más productivos y generen retornos sobre la inversión. El ATDD ayuda a coordinar los proyectos de software para entregar lo que el cliente desea.

En ATDD el proceso de desarrollo está dirigido por las pruebas de aceptación, las cuales deberían ser escritas por el cliente con la ayuda de un desarrollador. Según Reppert [18] las pruebas de aceptación proporcionan al cliente un contrato legible y ejecutable que el programador tiene que cumplir, y son usadas como documentación en un entorno ágil, dejando de ser únicamente un artefacto de pruebas como se utiliza en TDD.

El ciclo de desarrollo ATDD se encuentra representado en la figura 27:

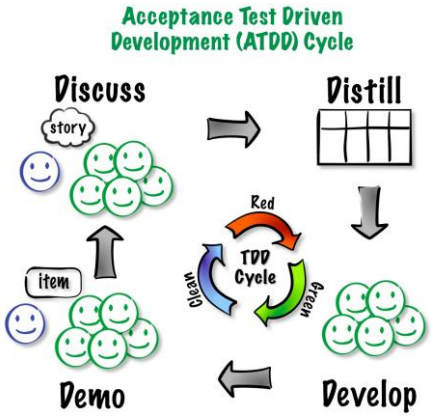


Figura 27. Clico de desarrollo con ATDD

1. **Discusión**. Desarrolladores, testers, expertos de dominio y el dueño de producto se juntan y discuten la funcionalidad para obtener un criterio de aceptación.

2. **Destilar**. Una vez que se tienen las pruebas se capturan éstas en un formato compatible con un framework de automatización de pruebas. Se intenta que las pruebas queden expresadas lo más claro y entendible posible, sin preocuparse (todavía) en cómo se van a automatizar. El foco está en la esencia de la prueba y se ignoran los detalles de implementación.

3. **Desarrollo**. Los desarrolladores siguen el enfoque de primero probar, ejecutan las pruebas de aceptación y ven que fallan. Luego automatizan las pruebas escribiendo lo necesario para conectar las pruebas al código. El siguiente paso es implementar el código mediante TDD. Finalmente se vuelven a pasar las pruebas de aceptación.

4. **Demostración**. Se realizan pruebas de exploración para revelar fallos en los criterios de aceptación y descubrir riesgos que no se habían pensado aún. Una vez que todas las pruebas pasan la historia puede ser demostrada al dueño de producto, avisándole de los riesgos potenciales descubiertos durante la implementación y exploración.

Para la mayoría de autores [6,10], un requisito en ATDD es una Historia de Usuario, especificada con una breve descripción narrativa, y a la cual se le asocia un conjunto de pruebas de aceptación, escritas normalmente en lenguaje natural como lo detalla Roger en [19].

5.2. Behaviour Driven Development (BDD)

BDD es introducido por North en [16,17] y surge como otra propuesta con el propósito de conectar el TDD con los requisitos. BDD es un enfoque que empieza desde fuera identificando la funcionalidad (valor de negocio) que el cliente quiere obtener, y después profundiza en el conjunto de requisitos que esta funcionalidad alcanzará. Cada requisito es capturado como una Historia. Los Escenarios definidos por una Historia proporcionan el criterio de éxito que será usado para determinar cuándo una Historia está completada. Por tanto, en BDD el proceso de desarrollo está dirigido por Historias y Escenarios que definen el comportamiento del producto.

En BDD se utilizan unas plantillas específicas para describir las Historias y los escenarios.

HISTORIAS	ESCENARIOS
Un rol “X”	Dado (Given), un contexto inicial “A”
Quiere un requisito “Y”	Cuando (When) un evento “B” se produce
Para obtener un beneficio “Z”	Entonces (Then) obtenemos un resultado “C”

A continuación se muestra un ejemplo real de una historia y uno de sus posibles escenarios:

HISTORIA

- Un cliente
- Quiere sacar dinero del cajero automático,
- De modo que no necesite pasar tiempo en la cola

ESCENARIO 1

- **Dado** que la cuenta posee crédito
- **Y** la tarjeta es válida
- **Y** el cajero automático tiene dinero
- **Cuando** el cliente pide dinero
- **Entonces**, asegurare que la cuenta sea debitada
- **Y** asegure que el dinero sea entregado
- **Y** asegure que la tarjeta sea devuelta.

BDD, mediante las plantillas, especifica un lenguaje común entre el cliente y los desarrolladores, de esta forma el cliente entiende que es lo que se va a desarrollar. Una historia se desglosa en diferentes escenarios. Cada escenario define una acción específica para cumplir con la característica pactada, el escenario se describe en lenguaje coloquial, por lo que el cliente lo entiende, especificando los pasos para lograrlo.

5.3. Diferencias de ATDD y BDD con TDRE

Ambos, ATDD y BDD, son muy similares en cuanto a que están orientados a la automatización de pruebas y generación de código (al menos parcial), aunque BDD ofrece más facilidades para especificar el comportamiento del sistema mediante el uso de plantillas y criterios de éxito. En BDD y ATDD las pruebas se automatizan inmediatamente al ser identificadas, no existe separación temporal en cuanto a la

especificación de la prueba y su correspondiente implementación. Por esto es necesario instanciar las PAs o escenarios en un formato que pueda ser usado por el equipo de desarrollo para automatizarlas. Esta instanciación inmediata durante la captura de requisitos implica un esfuerzo considerable que puede no ser conveniente ni posible en dicho momento. BDD y ATDD no están definidos en el marco de una metodología de desarrollo, cuando esto se hiciera debería resolverse cómo hacer factible que todo el proceso de desarrollo gire en torno a la interacción entre el cliente y el desarrollador para que este último, prueba a prueba, vaya implementando inmediatamente el comportamiento asociado a la prueba. Esto es difícil de llevarlo a la práctica cuando intervienen varios clientes y/o desarrolladores y cuando por envergadura del sistema se requiere mayor especialización en cuanto a roles. Por otra parte, normalmente para la automatización de una prueba de aceptación se necesita que la funcionalidad que se quiere probar se haya implementado antes. Nuestro enfoque TDRE no obliga a instanciar las pruebas cuando se definen los requisitos ya que dejará en manos de los testers el determinar las mejores combinaciones de datos y el decidir si es conveniente (y posible respecto de los recursos de testeo) la automatización de la prueba dentro de la misma iteración en la cual se incorpora el cambio asociado a la prueba.

Finalmente, los enfoques ATDD y BDD no abordan la problemática de gestionar la gran cantidad de pruebas que se generan. Es esencial disponer de mecanismos para manipular de forma ágil las PAs, particularmente en un contexto de mantenimiento continuo de un producto. En TDRE, y como hemos comentado antes, conscientes de la importancia de este aspecto, se ofrecen mecanismos que abordan esta necesidad.

5.4. Ejemplos Herramientas

ATDD y BDD están soportados por herramientas comunes, entre las que destaca el Framework for Integrated Test (FIT), detallado en [7,11], el cual apoya la generación de código y la automatización de PAs. Existen varias herramientas basadas en FIT, tales como FitNesse (fitnesse.org), JBehave (jbehave.org) y GreenPepper (greenpepper-software.com).

En FitNesse las pruebas se expresan como tablas que contienen los datos de entrada y los datos de salida que se esperan. En la primera fila de la tabla se indica la ubicación del **Fixture Code** (clase que procesa el contenido de la tabla). A continuación vamos a

ilustrar como se especificaría en FitNesse la siguiente historia de usuario: *"Se puede añadir jugadores al juego y preguntarle cuántos jugadores están jugando".*

org.fitnesse.triviaGameExample.fitnesseFixtures.AddRemovePlayerFixture		
playerName	addPlayer?	countPlayers?
Al	true	1
Bertha	true	2

Figura 28. Tabla de decisión en FitNesse

La tabla de decisión de la figura 28 en resumen dice que si añadimos exitosamente un jugador con nombre Al al juego, el número de jugadores en total será 1, y después si añadimos una jugadora con nombre Bertha, el total de jugadores será 2.

En la primera fila de la Figura 28 "*org.fitnesse.triviaGameExample.fitnesseFixtures"* se especifica el paquete Java (o el namespace de otro lenguaje), y *"AddRemovePlayerFixture"* la clase que será llamada para procesar la tabla. El resto de filas, cada una representa un escenario completo de ejemplo con columnas de datos de entrada (columna *"playerName"*) y columnas de salidas o resultados esperados (columnas "*addPlayer*" y "*countPlayers*"), normalmente las columnas de salida se diferencian porque contienen algún carácter especial al final.

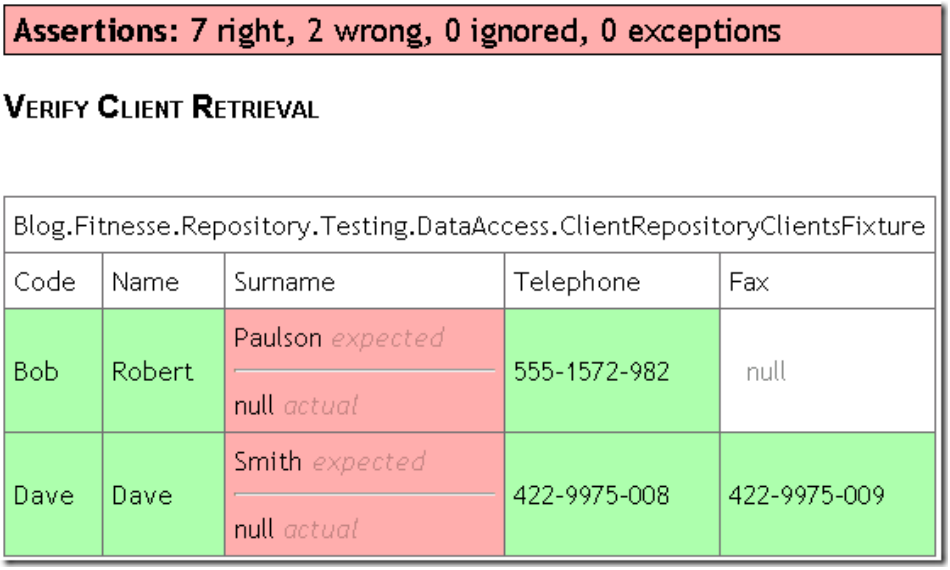
Una vez ya se ha especificado la tabla de decisión se implementa el Fixture Code que para el ejemplo de la figura 28 podría ser el siguiente:

```
import fit.ColumnFixture;

public class AddRemovePlayerFixture {
  private String playerName;
  private Game theGame;

  public void setPlayerName(String playerName) {
    this.playerName = playerName;
  }

  public boolean addPlayer() {
    theGame = StaticGame.theGame;
    Player thePlayer = theGame.addPlayer(playerName);
    return theGame.playerIsPlaying(thePlayer);
  }

  public int countPlayers() {
    return theGame.getNumberOfPlayers();  }
}
```

El **fixture addPlayer** es realmente sencillo, lo único que hace es llamar a las funciones "*addPlayer"* y "*playerIsPlaying"* de la clase Game, la cual hace el trabajo real. La clase Game con la implementación de la historia de usuario sería la siguiente:

```
public class Game {
  private ArrayList players;

  public Game() {
    players = new ArrayList();
  }

  public Player addPlayer(String aPlayerName) {
    Player aPlayer = new Player(aPlayerName);
    players.add(aPlayer);
    return aPlayer;
  }

  public boolean playerIsPlaying(Player aPlayer) {
    return players.contains(aPlayer);
  }

  public int getNumberOfPlayers() {
    return players.size();
  }
}
```

Una vez ya tenemos todo esto ejecutamos la tabla de decisión y si se colorean todas las celdas en verde, quiere decir que nuestro código nos ha devuelto el mismo resultado que esperábamos. Si la celda está marcada en rojo algo ha ocurrido y el código no nos devuelve el resultado que esperábamos, en este caso en la tabla se ha incluido una nueva fila debajo de la fila que ha fallado indicando el resultado que ha devuelto el código (ver Figura 29).



Assertions: 7 right, 2 wrong, 0 ignored, 0 exceptions

VERIFY CLIENT RETRIEVAL

Blog.Fitnesse.Repository.Testing.DataAccess.ClientRepositoryClientsFixture				
Code	Name	Surname	Telephone	Fax
Bob	Robert	Paulson *expected* null *actual*	555-1572-982	null
Dave	Dave	Smith *expected* null *actual*	422-9975-008	422-9975-009

Figura 29. Tabla de fallos en FitNesse

Estas herramientas resultan útiles cuando es la misma persona la que se encarga de definir las pruebas con el cliente, programarlas y automatizarlas, ya que al automatizar una prueba, además de conocer el dominio del problema, se requiere tener conocimiento de detalles internos de cómo está implementado el producto. Esto es un obstáculo en un contexto industrial y colaborativo, donde el trabajo está especializado en diferentes roles, y no todos tienen por qué conocer el código del producto. En TDRE las PAs actúan como elemento integrador del trabajo de los diferentes roles; el analista identifica y define las PAs, el programador debe implementar comportamiento para satisfacer las PAs y el tester debe preparar datos y ejecutar las PAs para validar el comportamiento del producto.

Capítulo 6. Infraestructura para Análisis de Impacto

En esta sección se presenta el concepto de base de datos orientado a grafos (BDOG), infraestructura software que se ha implementado en TUNE-UP para dar soporte a la especificación de PAs con referencias a entidades, mostrando sus ventajas e inconvenientes, las principales diferencias respecto de las bases de datos relacionales (BDR) y las diferentes propuestas existentes en el mercado centrándonos principalmente en Neo4j que es la opción elegida para la implementación del proyecto.

6.1. BD orientada a grafos

La capa de persistencia en nuestra versión actual está soportada por una base de datos orientada a grafos (BDOG), ya que se trata de la manera más natural e intuitiva de representar el sistema de relaciones de trazabilidad. Las BDOG almacenan la información utilizando una de las estructuras de datos más común y estudiada del mundo de la informática y las matemáticas, los grafos.

Los grafos son una potente herramienta que nos permite modelar una gran cantidad de sistemas, su utilización y estudio a lo largo de los últimos tres siglos nos ha permitido estudiar problemas matemáticos que actualmente se encuentran representados en muchos ámbitos. Las BDOG utilizan toda esta experiencia previa para crear un modelo de base de datos robusto y flexible gracias a las propiedades innatas de los grafos. Además, dependiendo de las diferentes implementaciones, se han ido añadiendo mecanismos utilizados actualmente en las BDR como el control de las transacciones y concurrencia, mecanismos de persistencia, índices, recuperación de datos o incluso lenguajes del tipo *query*.

Las BDOG nos aportan un nuevo modelo de datos completamente diferente al modelo relacional predominante en el mundo de la informática. Sus principales diferencias nos pueden resolver problemas y reducir el coste computacional en determinados ámbitos, destacando principalmente aquellos donde la explotación de las relaciones sea predominante.

A continuación se van a explicar las principales características de las BDOG.

- **Flexibilidad a cambios en la estructura de datos.** Los cambios en el modelo producen modificaciones en el conjunto de nodos, relaciones o propiedades que

se corresponden con sus operaciones básicas permitiendo una rápida evolución. En las BDR las modificaciones sobre el modelo se corresponden con modificaciones de las estructuras fijas de las tablas, las claves ajenas correspondientes, así como la actualización de los datos actuales y de las restricciones.

- **Conjunto de datos semi-estructurados.** La definición de los modelos no es prefijada como lo son las tablas de las BDR. Los nodos y relaciones de las BDOG contienen pocos campos obligatorios (identificadores), a los cuales se les añaden propiedades (pares *{key, value}*) en función de las necesidades del modelo. Esta característica además de ofrecernos una gran flexibilidad para representar nuestro sistema nos mejora el coste espacial de la información a almacenar. Es importante destacar que las propiedades son de longitud variable, evitando tener que definir su tamaño y también se pueden combinar valores complejos y multi-evaluados.
- **Intuitivo en modelos Orientados a Objetos.** En las BDR es necesario definir un *mapping* entre los objetos de la capa de dominio y las tablas del modelo relacional de persistencia, sin embargo no existe un desajuste inherente entre el modelo de objetos y el orientado a grafos. Además en las BDOG la representación de los datos, los esquemas y sus consultas están unificados en la figura del grafo.
- **Consultas optimizadas para la explotación de relaciones.** Las consultas realizadas en las BDOGs no están enmarcadas por tablas como las de las BDRs, evitando así las costosas operaciones join. Las consultas están optimizadas para sistemas fuertemente interconectados, recorriendo las relaciones entre los nodos siguiendo unos criterios de búsqueda y dando como resultado un subgrafo con los nodos y relaciones que hayan satisfecho los criterios de búsqueda.

Las BDR pueden ser más eficientes que las BDOG en los casos donde los elementos del sistema estén débilmente relacionados, así como en todas aquellas consultas que impliquen pocas tablas con independencia del número de registros que contengan. Por otra parte, también cabe destacar la madurez y estabilidad de los sistemas de gestión de base de datos relacionales que aportan herramientas adicionales al propio modelo que mejoran la usabilidad. Índices, triggers, mecanismos de modelado o generación automática de consultas por ejemplo, son elementos de los cuales disponemos

normalmente en estos sistemas de gestión y aún no están implementados en los sistemas de BDOGs.

Las BDR no son ni mejor ni peor que las BDOG pero se tiene que elegir correctamente cual será la más apropiada para modelar nuestro sistema. En nuestro caso hemos elegido un modelo de base de datos siguiendo los siguientes criterios:

- Nuestro sistema se puede modelar de forma natural, elegante y simple como un grafo donde los nodos son PAs y Elementos del Dominio. Los nodos están conectados mediante diferentes tipos de relaciones a explotar posteriormente por el análisis de impacto.
- Modificación incremental del dominio. Los principales elementos a tener en cuenta en la versión inicial de nuestro análisis de impacto son las PAs y las entidades del modelo de dominio, pero se espera que el análisis se pueda extender para incluir otros elementos como nodos, clientes o incluso clases del código. Esto sería fácil de incorporar debido a la flexibilidad de cambio en la estructura que nos ofrecen las BDOG.
- Heterogeneidad de las diferentes entidades del modelo de dominio. Las entidades del modelo de dominio pueden ser muy diversas y cada una será representada por un conjunto distinto de propiedades. Por ejemplo la entidad *Factura* no tiene las mismas propiedades que la entidad *Cliente.* A diferencia de las BDR donde no hay una manera simple de representar estas diferencias (a no ser que se desperdicie espacio con campos nulos), en las BDOG los nodos no están tipados y pueden tener diferentes conjuntos de propiedades de manera natural.
- Importancia de las relaciones en nuestro sistema. Nuestro sistema se quiere diseñar principalmente para explotar las relaciones de trazabilidad entre los artefactos, las BDOG están especialmente orientadas a la navegación de estas relaciones.

Una vez ya teníamos claro el modelo de base de datos, el siguiente paso al que nos enfrentamos era elegir concretamente cuál de las BD existentes en el mercado cubría mejor nuestras necesidades y restricciones. Aunque las BDOG son un enfoque relativamente nuevo en el ámbito de las BD (al menos en la práctica), en los últimos años han salido muchos proyectos relacionados con este modelo. Para estudiar cuál era

la BDOG a utilizar, se definieron una serie de criterios basados en las restricciones tecnológicas y la madurez del proyecto de BD.

En nuestro proyecto se utiliza .NET de Microsoft, por lo tanto era importante utilizar alguna BDOG que nos ofreciera una API de comunicación en c# o al menos algún elemento de comunicación mediante servicios. Otro aspecto a considerar era la madurez del proyecto (estabilidad y completitud), la documentación asociada y en general el volumen de usuarios que tenía la comunidad. Por último también nos interesaba el tipo de licencia opensource que nos permitiera utilizarla sin ningún coste adicional. Siguiendo estos criterios, estudiamos un conjunto de opciones buscando por internet y poniéndonos en contacto con los desarrolladores. En el resultado de este estudio obtuvimos las siguientes BDOG: **DEX, HiperGraph, Infogrid, Neo4j, InfiniteGraph, AllegroGraph y GraphDB**. Es importante tener en cuenta que las BDOG evaluadas, actualmente pueden tener características diferentes a cuando se tomó la decisión.

Al final de la evaluación decidimos utilizar la BDOG **Neo4j**. Esta es una BDOG desarrollada en Java por la compañía sueca NeoTechnology, y probablemente es la BDOG con más desarrolladores, mayor estabilidad y robustez.

Cómo todas la BDOG su modelo de datos está basado en grafos y toda su información está representada por nodos, relaciones y propiedades. Una relación se encarga de conectar dos nodos, y ésta puede tener propiedades, dirección y obligatoriamente tendrá un tipo que se utilizará posteriormente para la navegación. Los nodos no tienen tipo y disponen de un conjunto de propiedades asociado exclusivamente a la instancia del nodo. En conclusión el modelo de datos de Neo4j representa la información como una red fuertemente conectada (Figura 30).

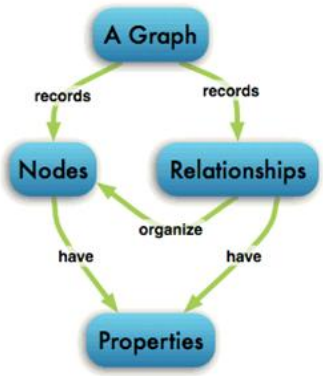


Figura 30. Representación del modelo de datos de Neo4j

A continuación se van a enumerar las principales características de Neo4j remarcando aquellas que nos han hecho decidirnos finalmente para utilizarla:

- **Representación intuitiva.**
- **Almacenamiento persistente** basado en disco, completamente optimizado tanto en rendimiento como en escalabilidad para guardar estructuras basadas en grafos.
- **Escalabilidad**. Actualmente Neo4j puede soportar varios billones de nodos, relaciones y propiedades en una única máquina, además puede ser fragmentada para acceder a través de varios servidores.
- **Flexibilidad en el modelo**. Además de la principal ventaja de las BDOG (flexibilidad para modelar nuestro sistema de forma incremental), Neo4j nos aporta otro nivel de flexibilidad que no tienen la resta de implementaciones y se trata de inexistencia de nodos tipados.
- **Transacciones ACID**. Neo4j tiene un sistema de transacciones que soporta completamente las propiedades ACID, es decir, Atomicidad, Consistencia, Aislamiento y Durabilidad.
- **API de acceso simple**. En nuestro caso no utilizaremos directamente esta funcionalidad porque accederemos a la BD mediante servicios web.
- **StandAlone Server**. Neo4j permite utilizar una base de datos en modo servidor en vez de estar integrada dentro de la aplicación. Para comunicarse en la BD nos ofrece unos sistemas de configuración de puertos y una API REST basada en llamadas http. Este es el mecanismo que utilizamos en el proyecto para comunicarnos y evitar la incompatibilidad tecnológica.
- **Potente mecanismo de recorrido**. En la mayoría de implementaciones de BDOG los elementos más utilizados para recorrer el grafo son los *Traversals.* Éstos se encargan de explorar los datos utilizando búsquedas en nivel o profundidad, donde se definen algunos criterios de búsqueda como el tipo de relaciones a recorrer, su dirección, la profundidad desde el nodo inicial, el valor de un cierto atributo,… y devuelven el subgrafo resultante del recorrido guiado.
- **Sistema completo de indexación**. Para acceder más eficientemente a los elementos (nodos y relaciones) siguiendo unos criterios de búsqueda parecidos a los utilizados en una BDR (por ejemplo búsquedas por *string*), Neo4j nos permite utilizar Apache Lucene que nos permite crear y organizar índices de

nodos y relaciones. Esta función aún no la utilizamos en nuestro proyecto, debido a que se ha incorporado en versiones recientes de Neo4 y se ha dejado pausado hasta que no se normalice su funcionamiento en versiones futuras.

- **Herramientas adicionales**. Una de las características más interesantes de Neo4j es la cantidad de herramientas que tenemos a nuestra disposición para hacer un buen uso de la BD. Sin tener en cuenta las ya nombradas anteriormente, esta BDOG nos aporta un sistema incremental de backups (en la edición *Enterprise*), herramientas de monitorización, algoritmos ya implementados de búsqueda de caminos más cortos disponibles desde las dos APIs, incorporación de plugins definidos para el usuario para extender la funcionalidad del API actual (plugins *Gremlin* y *Cypher* dos lenguajes del tipo *query* y *pattern matching* para tener una mayor expresividad en las consultas) y un administrador Web que nos ofrece el servidor y nos permite monitorizar y visualizar los datos en todo momento haciendo uso exclusivamente de un navegador. En la Figura 31 los datos se pueden representar utilizando distintas representaciones incluyendo una visualización en forma de grafo.

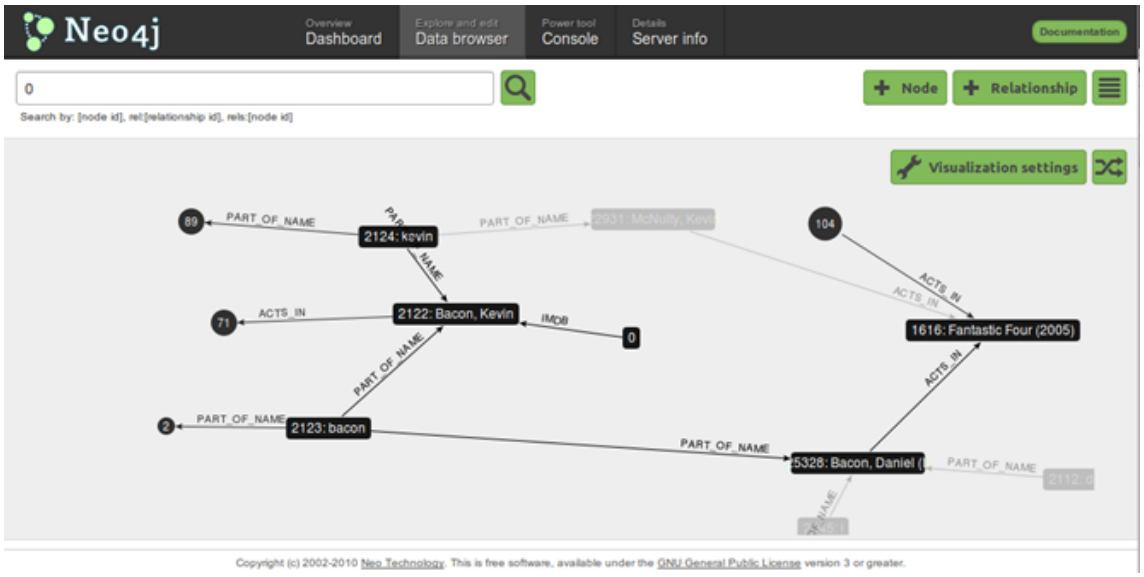


Figura 31. WebAdmin ofrecido por Neo4j

- **Licencia dual.** Esta BDOG es un proyecto de código abierto que ofrece dos tipos de licencias completamente diferenciadas (ediciones no comerciales y ediciones comerciales). En nuestro caso estamos utilizando la edición *Community* bajo una licencia GPL que nos permite utilizar Neo4j sin ningún tipo de restricción, con la única excepción de que no se puede distribuir la

aplicación que utiliza la BD con código cerrado, en este caso tendríamos que adquirir una licencia comercial.

- **Comunidad y documentación**. Se dispone en la Web de una extensa documentación con un manual muy completo y varios ejemplos de implementaciones reales. Además hay una lista de correos muy activa donde se puede expresar cualquier duda o conocer las últimas novedades.
- **Entregas de versiones frecuentes**. Neo4j es una BD que basa su desarrollo en ciclos cortos de aproximadamente 6 meses, donde cada mes se entrega una versión *milestone* para que la comunidad vaya probando su rendimiento. Los desarrolladores implementan en cada versión lo más demandado por la comunidad.

6.2. Gestión del Modelo de Dominio

Al inicio de nuestra propuesta, el modelo de entidades estaba pobremente definido como un conjunto no ordenado de términos en un documento de Microsoft Word compartido mediante un servidor SharePoint. Con el fin de poder gestionar las entidades de dominio en el contexto de TUNE-UP los analistas pidieron un nuevo módulo que permitiera definir y organizar los términos con facilidad, accediendo a él desde el formulario donde se definen las PAs.

El contenido del Gestor de Términos se almacena en la BDOG. Pensando en el uso que se iba a dar a los términos se han definido dos niveles de entidades de dominio. En primer nivel disponemos de los términos que son entidades independientes con propiedades y relaciones con otros elementos del mismo nivel. Además tenemos los atributos que son entidades de segundo nivel que contienen propiedades y sólo tienen sentido en el contexto de un término, y por tanto están relacionadas con un único término.

Todos los términos y atributos pueden ser referenciados en una PA para posteriormente realizar el análisis de impacto. Sin embargo, las propiedades de un término o de un atributo únicamente muestran las descripciones o restricciones de cada elemento que se pueden asociar a una PA, de esta forma toda persona involucrada en el desarrollo del producto dispone de una información completa para entender la PA.

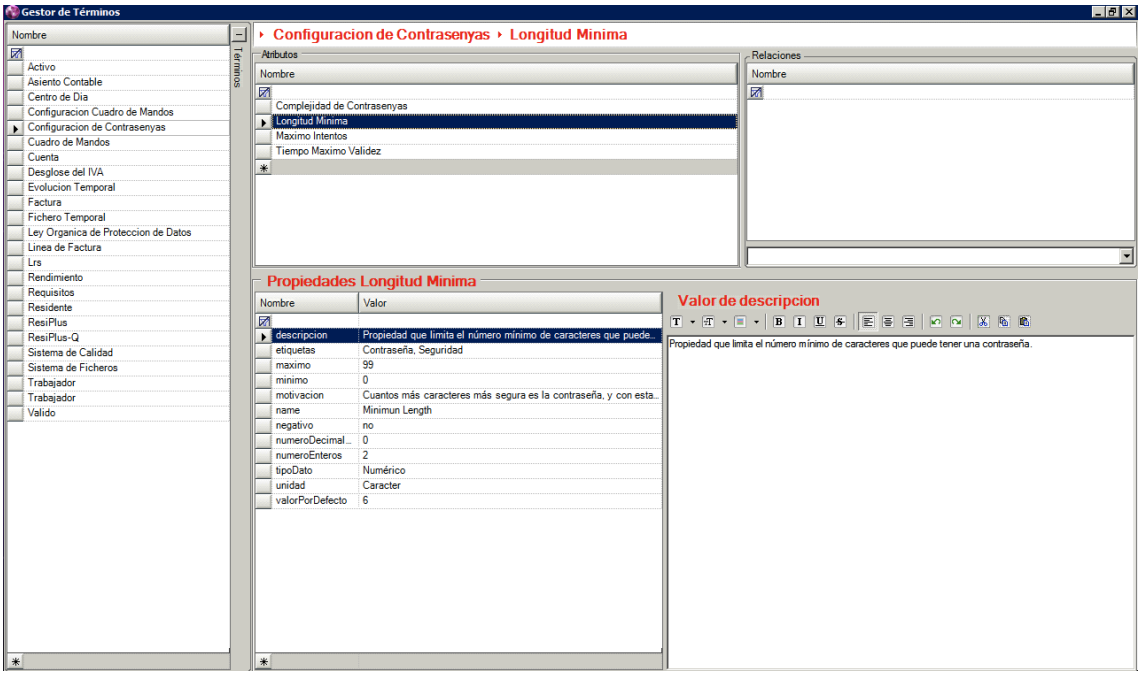


Figura 32. Gestor de Entidades del Modelo de Dominio

Después de varias propuestas y reuniones con el equipo de analistas, se definió el Gestor de Términos que se muestra en la Figura 32.

En la parte derecha del gestor hay un panel ocultable con un grid que muestra todos los términos que hay actualmente en la BDOG, teniendo la opción de introducir nuevos. La información que se observa en las otras secciones depende de la fila que se tenga seleccionada en el grid de términos, es decir, en seleccionar una fila los datos de los otros grids se actualiza con las propias características del término.

En la parte superior se encuentran las dos secciones relacionadas directamente con el término que se encuentra seleccionado: el área de atributos y relaciones. En la primera se muestran en un grid los atributos o elementos de segundo nivel relacionados con el término y nos permite añadir nuevos. En la segunda sección se encuentra otro grid con las relaciones del término con otros términos del mismo nivel, teniendo la opción de añadir nuevas relaciones mediante un listado que nos muestra todos los términos de primer nivel que se encuentran en la BDOG.

Por último en la parte inferior se encuentra la sección de las propiedades que muestra mediante un grid todas las propiedades que se han definido con su valor correspondiente. El valor de las características se puede editar desde un control de texto rico (TX Text Control) para poder añadir formato. Los datos que se muestran en esta

sección pueden estar relacionados tanto a términos como atributos dependiendo del elemento que se haya seleccionado.

Para minimizar posibles confusiones al reutilizar la misma sección para propiedades de elementos de diferentes niveles se han añadido indicadores visuales en el navegador superior de la Figura 32. Concretamente se ha añadido un breadcrumb que nos indica en todo momento cual es el elemento contextual del formulario y además nos permite una nueva forma de navegar por los diferentes términos y atributos del modelo de domino, recordando aquellos elementos que ya han sido visitados.

Actualmente existen restricciones de eliminación y modificación sobre los elementos del gestor de término. Los nombres de los términos y atributos no pueden ser modificados ni eliminados si ya han sido referenciados en alguna PA. Esta limitación temporal ha sido aceptada por todos los usuarios debido a la inconsistencia que se produciría en los textos de la PA y en la posterior explotación de Análisis de Impacto. En futuras versiones se implementará un mecanismo automático de actualización, el cual se encargara de modificar la referencia en la PA.

6.3. Parser para Pruebas de Aceptación

El mecanismo ***intellisense*** permite fácilmente acceder a las entidades del modelo de dominio al introducir texto en cualquiera de los apartados de una PA. En TUNE-UP hemos implementado un *intellisense* tal como se ofrece en los entornos de desarrollo: se activa con ctrl + space, se quita con ESC, al escribir se filtra el contenido y con el punto se navega al siguiente nivel. Cuando se selecciona una entidad automáticamente se introduce en el cuadro de texto con un formato diferente (itálica y color de fuente azul). Para detectar las entidades en el texto de los diferentes apartados de la PA se ha implementado un parser del contenido de cada apartado de la PA (en formato rtf), en el cual se identifican las entidades y atributos referenciados. En el anexo II se ha incluido el código del parser con algunas anotaciones para que se pueda entender.

Cuando una PA contiene en su especificación alguna referencia a una entidad, automáticamente al aceptar los cambios se crea la relación PA-Entidad en la BDOG. Utilizamos atributos en las relaciones para establecer el apartado en el cual se referencia la entidad dentro de la PA. Esto nos permite identificar dependencias más específicas entre PAs, lo cual veremos en detalle en la siguiente sección.

En la Figura 33 se muestra, como se introduciría en TUNE-UP, la definición de la prueba “Intento de reintegro con saldo insuficiente” que se había mostrado antes, pero ahora referenciando las entidades del modelo de dominio. En itálica y azul se observan dichas referencias. Puede observarse que aunque se ha modificado la especificación original incluyendo las referencias a entidades, la especificación mantiene prácticamente el mismo grado de flexibilidad y legibilidad del lenguaje natural.

En una PA mediante el intellisense y navegando con el ‘.’ se pueden incluir referencias de los siguientes tipos:

- **Entidad**. Se referencia en la PA una entidad del modelo de dominio. Al abrir el intellisense con ctrl + esp se cargan todas las entidades y para añadirla al texto de la prueba se tiene que hacer doble click sobre ella o presionar enter. Un ejemplo podría ser si en una PA se referenciara la entidad ‘Cuenta’.

- **Atributo de una entidad**. En una PA se puede hacer referencia a un atributo de una entidad del modelo de dominio. Por ejemplo, en la PA de la Figura 33 se hace referencia a ‘Cliente.Tipo’, donde Tipo es un atributo de la entidad Cliente. Para añadir un atributo en el texto de la PA es necesario incluir primero la entidad contenedora del atributo. Cuando se selecciona la entidad, al poner el ‘.’ automáticamente el intellisense se recarga y muestra todos los atributos y entidades asociadas a la entidad previamente seleccionada.

- **Entidad asociada a otra entidad.** En la PA se ha referencia a una entidad que está asociada a otra entidad. Por ejemplo ‘Cliente.Cuenta’, en este caso la entidad Cliente está relacionada con la entidad Cuenta. Para acceder a Cuenta es necesario acceder a través de la entidad Cliente.

- **Atributo de una entidad asociada a otra entidad.** Se hace referencia en la PA a un atributo de una entidad que está asociada a otra entidad. Por ejemplo, en la PA de la Figura 33 se hace referencia a Cliente.Cuenta.Saldo, donde Saldo es un atributo de la entidad Cuenta que está relacionada con la entidad Cliente.

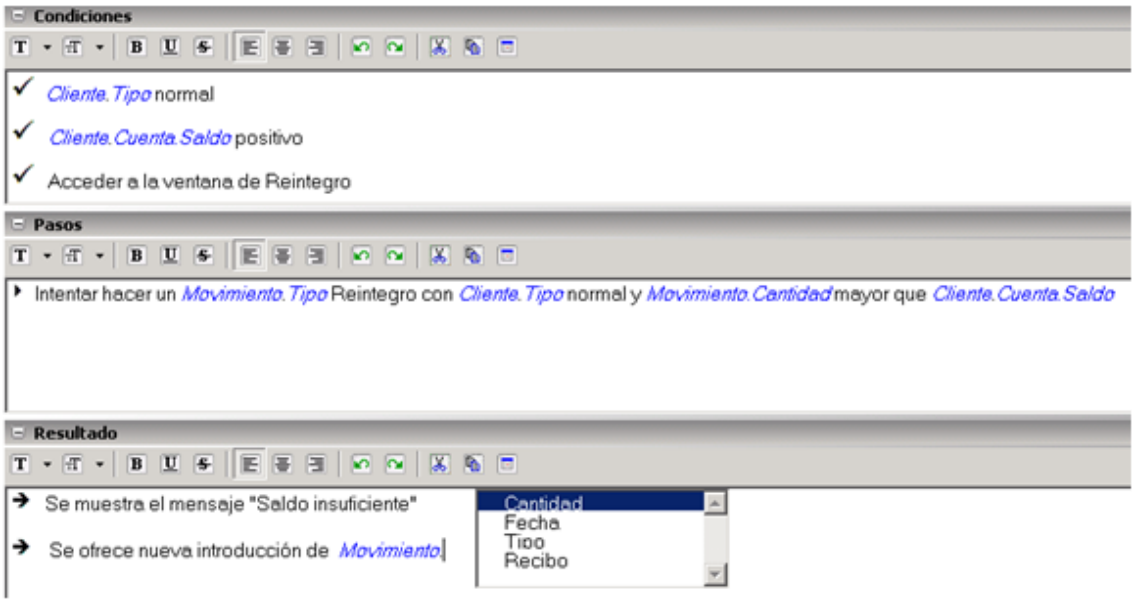


Figura 33. Ejemplo de PA con referencias a entidades

Capítulo 7. Análisis de Impacto

En TUNE-UP el Análisis de Impacto está centrado en el comportamiento externo del producto, es decir, queremos determinar el conjunto completo de requisitos que pueden verse afectados o involucrados por un cambio. En TUNE-UP las PAs definen el comportamiento actual del producto, pero también un cambio asociado a una WU se especifica en términos de PAs añadidas, modificadas o eliminadas.

El Análisis de Impacto nos da información de qué artefactos se pueden ver afectados y por lo tanto se tienen que revisar cuando hacemos un cambio. El conjunto de artefactos que nos devuelve el análisis de impacto puede ser bien porque tienen una relación de trazabilidad directa o indirecta. En la Figura 34 se muestra un ejemplo de cómo es el Análisis de Impacto directo en TUNE-UP. Una WU afecta a un Nodo porque se han añadido, modificado o eliminado PAs en ese Nodo, por lo tanto ese cambio puede impactar sobre los otros tres nodos de la Figura 33 porque a través de sus PAs está relacionado directamente con ellos.

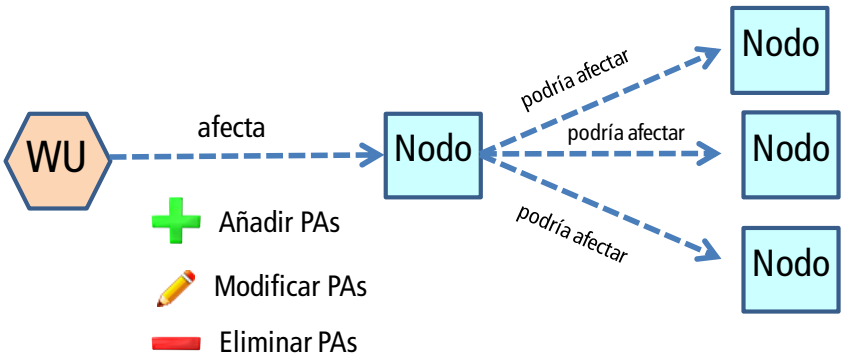


Figura 34. Impacto directo en TUNE-UP

En TUNE-UP sólo hemos considerado por ahora el Análisis de Impacto directo, pero somos conscientes que no es suficiente y en versiones futuras incluiremos las relaciones indirectas. En TUNE-UP se incluirán estas relaciones pero de una manera opcional y siempre a petición del usuario, él decidirá el nivel de profundidad que quiere explorar dependiendo de la sofisticación del cambio. En la Figura 35 por ejemplo se muestra el resultado del Análisis de Impacto con un nivel de profundidad 2, las relaciones de color rojo serían aquellas que aunque no están relacionadas directamente con el nodo podrían verse afectadas porque tiene una cierta relación indirectamente.

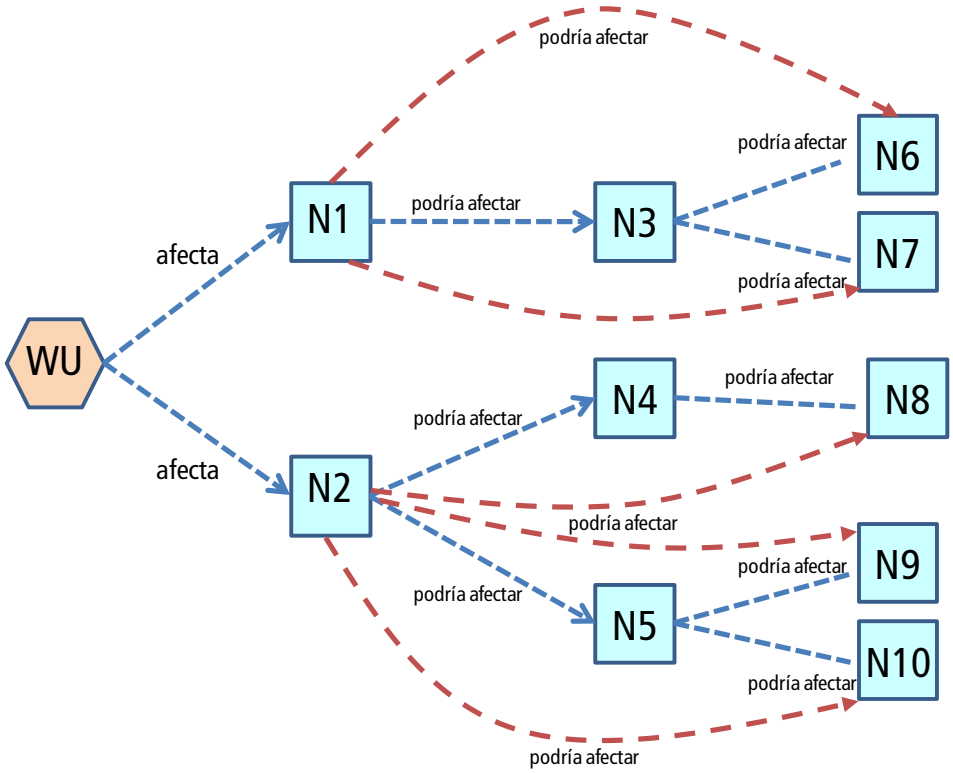


Figura 35. Análisis de impacto indirecto en TUNE-UP

En nuestra propuesta gracias a las relaciones directas creadas de forma semiautomática del tipo PA-Entidad podemos hacer Análisis de Impacto de dos tipos, los cuales hemos denominado Interdependencia Simple y Dependencia Causa-Efecto. Estos dos tipos de Análisis de Impacto se explican a continuación.

7.1. Interdependencia Simple

En TUNE-UP los requisitos se corresponden con nodos de la estructura del producto y el comportamiento asociado a cada nodo está determinado por el conjunto de PAs contenidas en el nodo. Cuando se van a añadir, modificar o eliminar PAs (que tendrán ciertas referencias a entidades), todas las otras PAs (del mismo nodo o de otros nodos) que tengan referencias hacia las mismas entidades deberían ser revisadas para verificar si se verán afectadas. Si posteriormente se determina que efectivamente el cambio afecta a una o varias PAs, puede que dicho cambio deba propagarse y dichas PAs afectadas también tengan que ser modificadas, o que la evaluación del impacto nos lleve a desistir del cambio y buscar una alternativa.

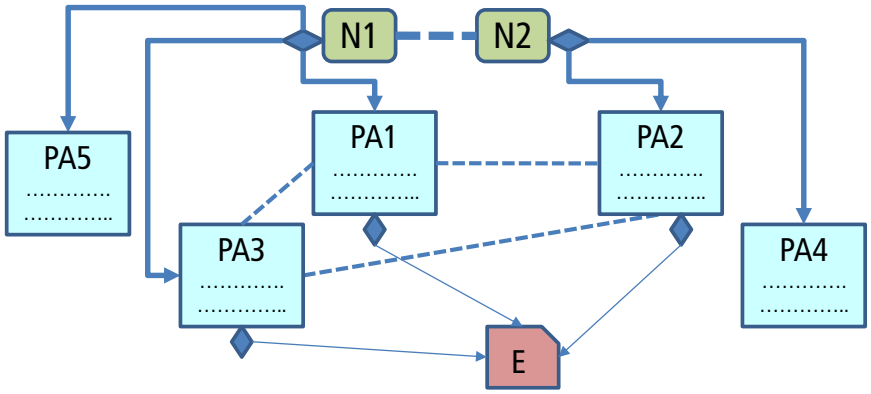


Figura 36. Ejemplo genérico de Interdependencia Simple

En la Figura 36 se muestra un ejemplo genérico de cómo funciona la interdependencia simple. En este ejemplo tenemos dos nodos (N1 y N2) con sus respectivas PAs. Las PAs: PA1, PA2 y PA3 referencian a la Entidad (E), por lo tanto estas tres pruebas están relacionadas entre ellas. Si se hiciera un cambio en cualquiera de ellas sería conveniente que se revisarán las otras dos. Además, como las PAs se encuentran en diferentes nodos (N1 y N2) se deduce que dichos nodos están relacionados, con lo cual de forma más global se podría deducir que un cambio en uno de dichos nodos podría afectar al otro.

A continuación en la Figura 37 con el fin de que sea más entendible se muestra un ejemplo real que representa este tipo de dependencia. Este ejemplo instancia con datos el ejemplo de la Figura 36, pero para simplificar sólo hemos considerado la PAs que están relacionadas entre sí (PA1, PA2 y PA3). La PA "Reintegro con saldo insuficiente" y la PA "Reintegro Normal" pertenecen al nodo "Reintegro" y la PA "Transferencia mismo banco" pertenece al nodo "Transferencia". Las tres pruebas están relacionadas entre ellas ya que referencian a la misma entidad "Cliente.Cuenta.Saldo" y consecuentemente los nodos contenedores de dichas PAs también están relacionados.

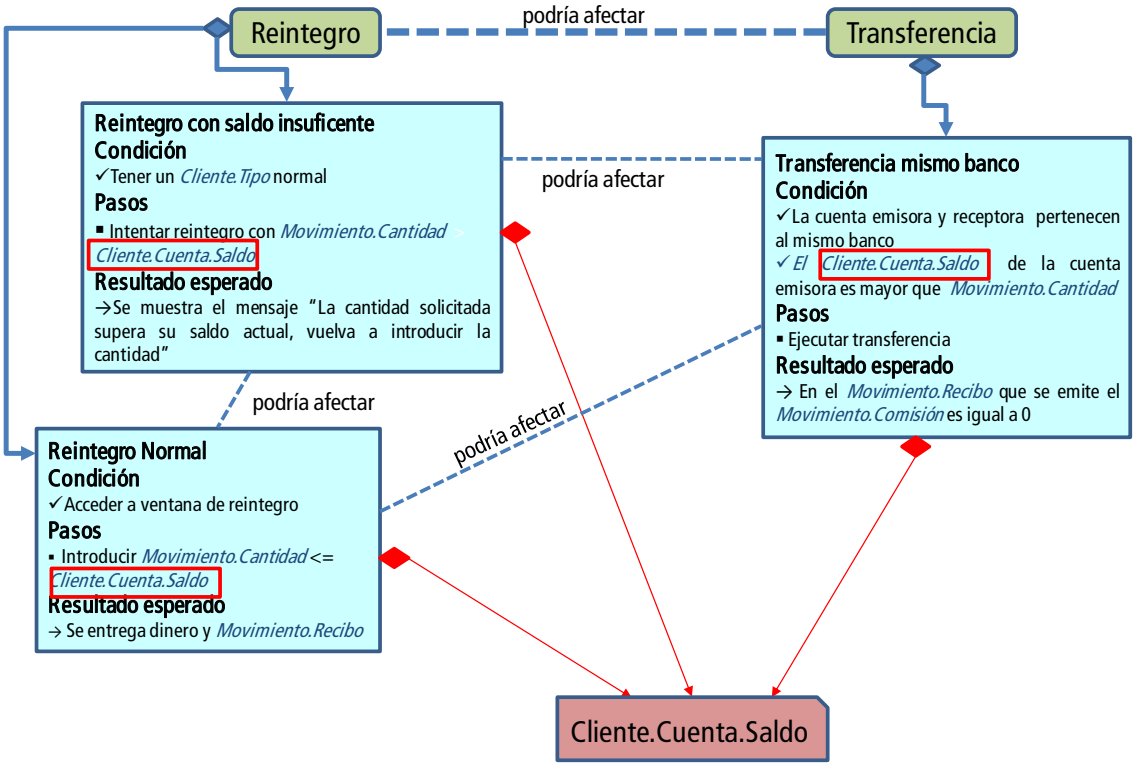


Figura 37. Ejemplo Interdependencia Simple

Resulta evidente que lo más adecuado para visualizar las relaciones es un grafo, similar al de la Figura 36. Sin embargo, como primera versión hemos presentado la información en grids. Se ofrecen dos vistas de explotación, en la primera, dada una PA se muestran otras PAs que se relacionan con ella a través de una entidad. Este vista de explotación se puede ver en la figura 38 donde se muestra como la PA seleccionada se relaciona con los nodos Gestión de Clientes, Transferencia y Reintegro a través de unas determinadas PAs, esta relación es debido a que esas PAs y la PA seleccionada tienen una asociación con la misma entidad. Por ejemplo la PA seleccionada se relaciona con el nodo "Gestión de Clientes" a través de la propuesta I-14914.20 'Modificar tipo cliente' porque ambas tienen en su descripción una relación del tipo afecta con la entidad 'Cuenta.Saldo'.

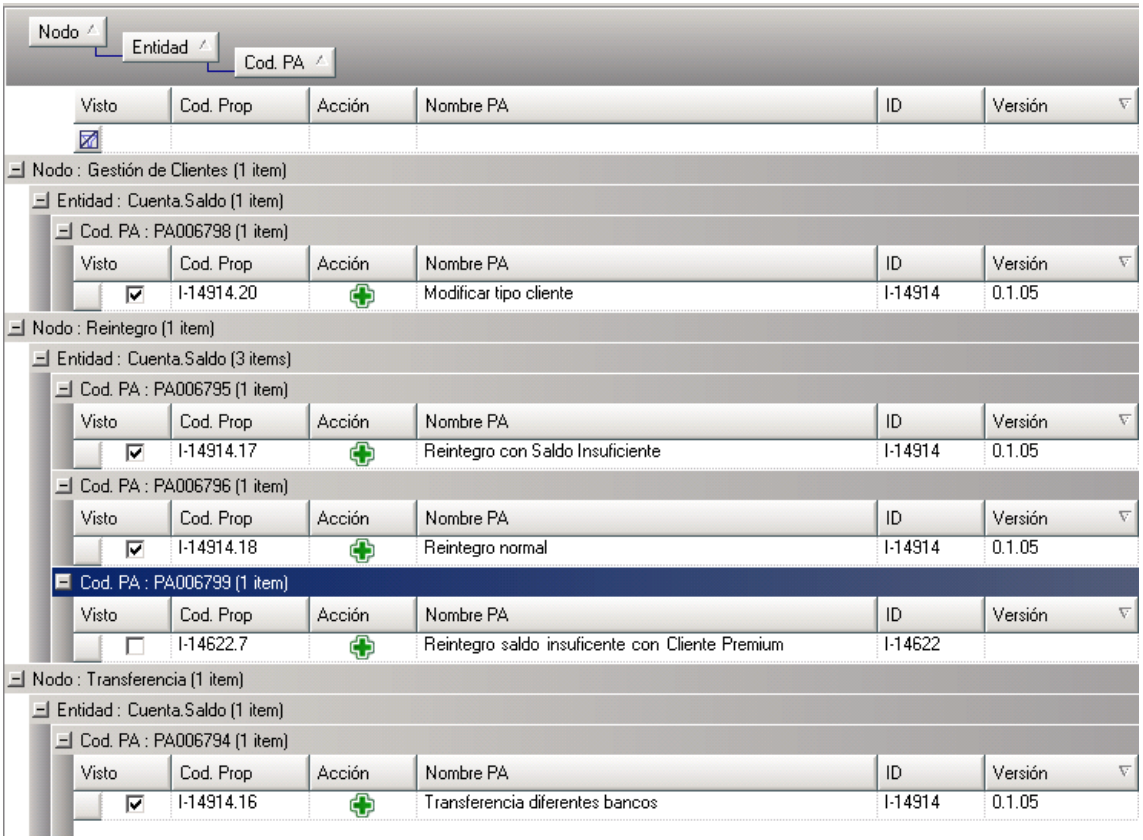


Figura 38. Explotación Análisis de Impacto en Interdependencia Simple (PA-PAs)

En la otra vista de explotación (Figura 39), teniendo seleccionado un nodo, se muestra qué otros nodos del producto se relacionan con él, desplegando además las PAs que hacen que los nodos se relacionen.

En el segundo nivel del grid de la Figura 39 se muestran las PAs que dada su referencia a una misma entidad hacen que se relacionen los nodos. Además, se proveen casillas para que se marque el nodo como revisado. Cuando esto se hace, automáticamente se marcan como revisadas todas las PAs del nodo. También se puede ir prueba a prueba revisándolas y cuando estén todas las pruebas del nodo marcadas se marcará automáticamente el nodo como revisado. En ambas vistas de explotación haciendo doble clic en una PA se accede a un formulario con todos los detalles de la especificación de dicha PA.

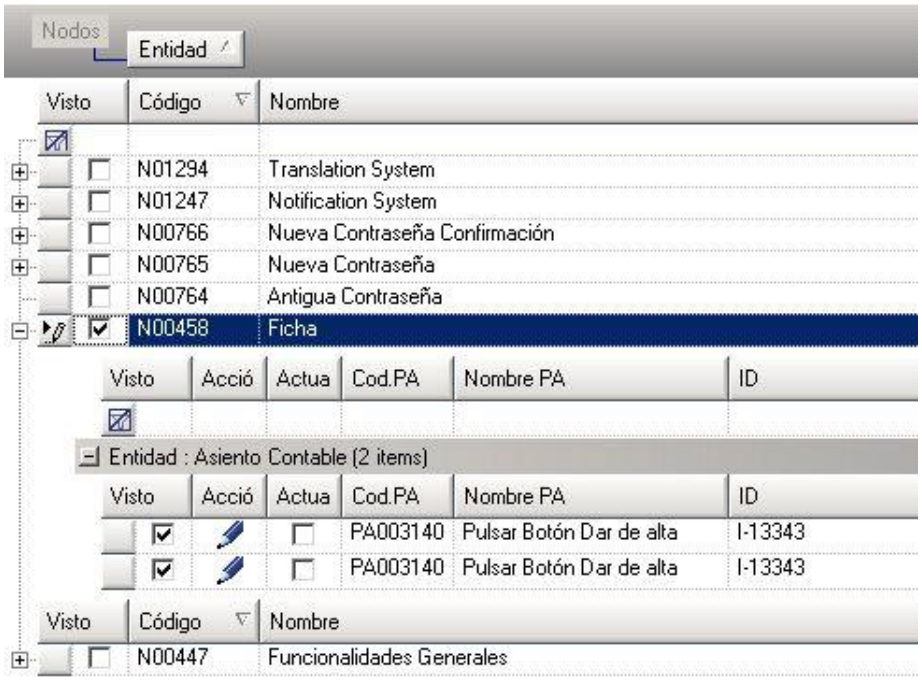


Figura 39. Explotación Análisis de Impacto en Interdependencia Simple (Nodo-Nodos)

7.2. Dependencia Causa-Efecto

La Interdependencia Simple no considera dirección en las relaciones entre PAs o entre nodos, así cualquier elemento podría verse afectado si alguno de los elementos relacionados se modifica. El segundo tipo de Análisis de Impacto que soporta nuestra propuesta lo hemos llamado Dependencia Causa-Efecto pues conlleva una direccionalidad desde uno de los elementos relacionados.

Al etiquetar con atributos las relaciones (facilidad que ofrece Neo4j) podemos diferenciar desde qué apartados de la PA se hace referencia a una Entidad. En la Figura 40 se ilustra la Dependencia Causa-Efecto. En este ejemplo tenemos dos pruebas (PA1 y PA2) con sus tres apartados. En el apartado Resultado de la PA1 se referencia a la entidad (E) y en el apartado condiciones de la PA2 se referencia también esa entidad. Así, al ejecutarse el comportamiento de la PA2 debe darse una condición que está relacionada con el resultado de la ejecución de la PA1. Por lo tanto, hay una mayor probabilidad de que si se modifica la PA1 esto va a tener un efecto en la PA2 (PA1 puede afectar PA2). Además, como dichas PAs se encuentran en nodos diferentes podemos deducir que el cambio en el nodo N1 puede afectar al nodo N2.

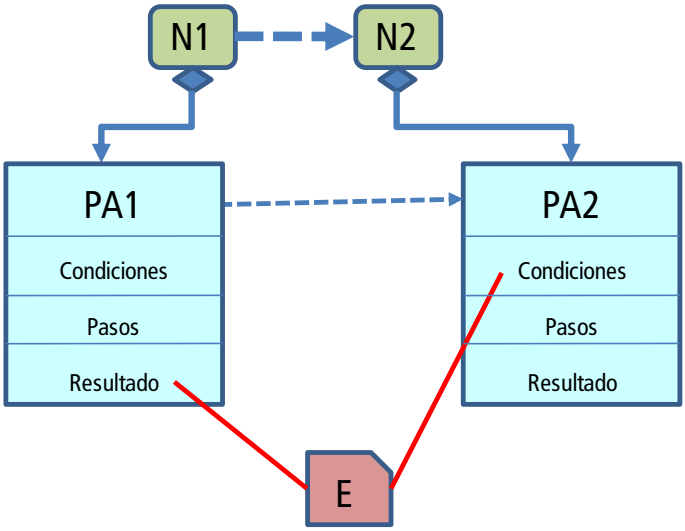


Figura 40. Ejemplo genérico Dependencia Causa-Efecto

A continuación en la Figura 41 se ilustra esta dependencia con un ejemplo real. En la PA1 "Edición atributo Premium" se marca a un cliente de tipo Premium referenciando en el apartado Resultado a la entidad "Cliente.Tipo". En la PA2 "Reintegro cliente premium" se hace un reintegro con un cliente del tipo Premium y en el apartado Condición se ha hecho referencia a la misma entidad "Cliente.Tipo". Las dos PAs están relacionadas con una dependencia Causa-Efecto, por lo tanto hay más probabilidad de que sí se modifica el comportamiento de edición del cliente Premium esto afecte a la PA que define otro comportamiento teniendo como condición el tener un cliente del tipo Premium.

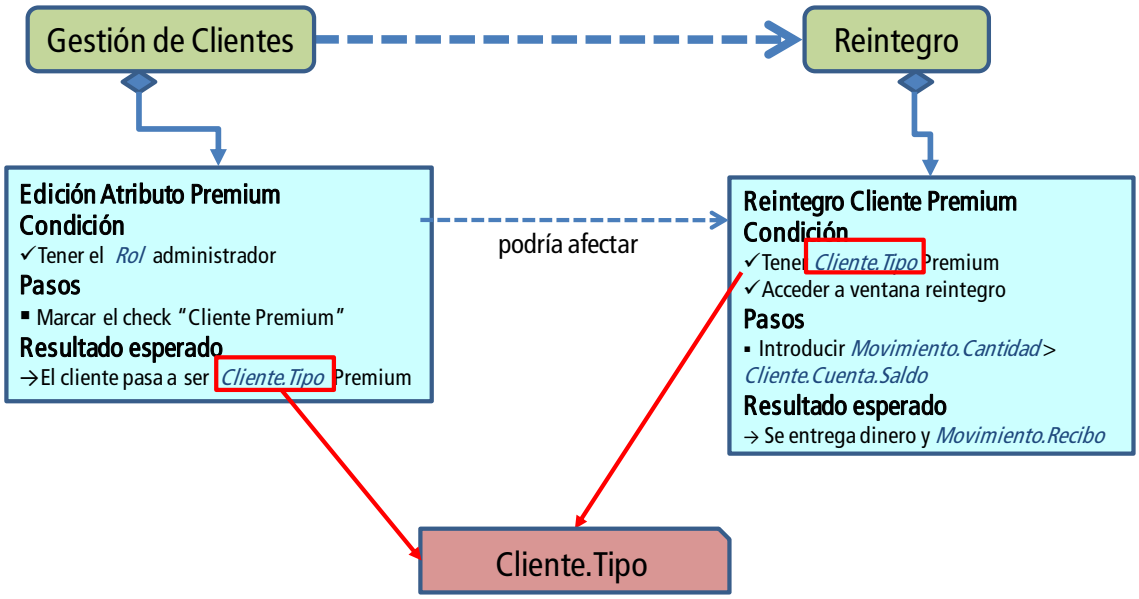


Figura 41. Ejemplo con dependencia Causa-Efecto

Un caso típico de Dependencia Causa-Efecto es el que se establece entre nodos (y sus PAs) en los cuales se realiza la configuración de un producto, con respecto de otros nodos (y sus PAs) sobre los cuales actúan los parámetros de configuración.

Las relaciones que obtenemos por Dependencia Causa-Efecto son un subconjunto de las que se obtienen mediante Interdependencia Simple, con lo cual en la explotación de la información las primeras son remarcadas, puesto que tienen mayor probabilidad de evidenciar la necesidad de propagación de cambios a otros requisitos.

Capítulo 8. Trabajos relacionados con Análisis de Impacto

La mayoría de trabajos que abordan el análisis de impacto con automatización para el mantenimiento de los enlaces de trazabilidad se centran en un análisis a nivel de diseño y código. A continuación se describen algunos de estos trabajos.

En [24] se presenta el prototipo *What-If Impact Analysis tool* que se ha implementado para analizar a priori el análisis de impacto a través de las dependencias del código, aunque se ha desarrollado el prototipo no se ha integrado en ningún entorno de desarrollo y tampoco ha sido testeado en ningún proyecto real. En [28] se presenta la técnica *PathImpact*, la cual mientras el sistema se está ejecutando, se construye una representación del comportamiento del sistema (utilizando un grafo acíclico dirigido propuesto por Larus en [32]) y la utiliza para estimar el impacto. En dicho trabajo los autores comparan su técnica con otras técnicas que predicen el impacto del cambio. Lee y Offut en [21] analizan las características del software orientado a objetos para definir cómo la composición, la herencia y el polimorfismo pueden influenciar en el Análisis de Impacto. Para ello categorizan los diferentes tipos de cambios que podrían ser aplicados en un sistema orientado a objetos y asignan a cada tipo un atributo que indica la influencia del cambio en otros objetos del sistema.

Las aproximaciones presentadas en [21,24,28] tienen como objetivo evaluar a priori el impacto de un cambio en el software, es decir, hacen un análisis predictivo para trabajar con un plan antes de tratar con las consecuencias.

Por otra parte, gran parte de los trabajos asociados a Análisis de Impacto intentan abarcar todos los posibles artefactos del proceso, a continuación comentamos dos que nos parece interesante mencionar.

Göknil en [22] presenta una técnica basada en la formalización de relaciones de requisitos en el contexto de Model Driven Depevelopment. Allí se presenta un metamodelo que incluye los conceptos comunes extraídos de las aproximaciones más importantes de Ingeniería de Requisitos. Los cuatro tipos de relaciones que se han considerado en el metamodelo (Requires, Refines, Conflicts y Contains) son utilizadas para hacer la traza y obtener los requisitos que se tendrían que modificar. En dicho trabajo se presentan estas cuatro relaciones formalizadas junto con unas restricciones que hay que tener en cuenta y basándose en esta formalización se definen las reglas de

impacto para los requisitos. En aproximaciones como ésta, en las cuales se propone trabajar con diferentes tipos de relaciones de trazabilidad entre los artefactos, aparece como problema añadido el diferenciar semánticamente dichos tipos de relaciones. Un caso extremo ocurre en [36] donde se proponen alrededor de 50 diferentes tipos de relaciones entre artefactos. En nuestro contexto y centrados en Análisis de Impacto hasta ahora sólo hemos consideramos el tipo de relación *Afecta*, es decir, una PA puede afectar otra PA, o un requisito (nodo) puede afectar a otro.

En [23] se presenta un enfoque para Análisis de Impacto considerando diferentes niveles de profundidad del alcance del cambio. El Análisis de Impacto se obtiene analizando las relaciones de dependencia entre métodos de componentes conectados. De manera similar en [38] se introduce la medida Distancia para ser capaz de evaluar el impacto según su probabilidad de ocurrencia y basándose en la distancia entre los elementos modificados y los elementos impactados.

En propuestas como las descritas en [23,22] las relaciones de trazabilidad entre los artefactos se definen manualmente, lo cual es inabordable en proyectos medianos o grandes, a menos que dicha trazabilidad sea fácilmente configurable para adaptarla a las necesidades del proyecto, restringiendo los tipos de artefactos y de relaciones que se identificarán y mantendrán actualizados [33,34].

Todos los enfoques antes descritos para Análisis de Impacto son teóricos, no están soportados por una herramienta, o tienen un prototipo, pero hasta donde llega nuestro conocimiento, en ningún caso han sido utilizados industrialmente.

Las herramientas comerciales específicas para gestión de requisitos ofrecen funcionalidad muy similar respecto de Análisis de Impacto, las cuáles normalmente se encuentran descritas en la literatura y están basadas en Matrices de Trazabilidad [29,30].

En la siguiente tabla se muestra una comparativa realizada en [40], sobre cómo las principales herramientas comerciales de gestión de requisitos (las cuatro herramientas que cumplen la mayoría de las funciones en una encuesta realizada por INCOSE [41]) realizan la trazabilidad (creación de relaciones y visualización).

IRqA	Rational Requisite Pro	DOORS	CaliberRM
Permite la creación de relación de trazabilidad entre requisitos y: -otros requisitos -elementos del dominio del problema (conceptos, entidades) -elementos de la especificación de la solución -escenarios -IRqA test cases y TestDirectos tests -clases de implementación -código fuente a través de la asociación con archivos externos VISUALIZACION -vistas de elementos relacionados -matriz de trazabilidad	Permite la creación de relaciones de trazabilidad entre los tipos de requisitos gestionados por RequisitePro VISUALIZACION -propiedades de los requisitos -matriz de trazabilidad	Permite la creación de relación de trazabilidad entre cualquier par de objetos contenidos en cualquier módulo dentro del repositorio de DOOR. La relación entre objetos está definida por el usuario en el link de módulos. Son establecidos diferentes tipos de relaciones entre un par de módulos (entre sus objetos) . Permite la definición de atributos por cada tipo de relación. La relación puede ser establecida desde: -los módulos que contienen los objetos que el usuario quiere relacionar -desde linkset of link module. VISUALIZACION -matriz de trazabilidad	Permite el establecimiento de relación de trazabilidad a través: -Trazabilidad tabs:es posible establecer trazabilidad entre requisitos y requisitos del proyecto VISUALIZACION -matriz de trazabilidad

La única diferencia entre cómo estas herramientas realizan el Análisis de Impacto y Trazabilidad son las diferentes maneras que tienen para generar y mantener las matrices de trazabilidad. Sin embargo todas ellas coinciden en que estas relaciones de trazabilidad se tienen que establecer y mantener manualmente. Este aspecto nosotros lo consideramos fundamental y por tanto la automatización es una de las principales ventajas de nuestra propuesta.

En la Figura 42 se muestra la matriz de trazabilidad de las características de software que relaciona a éstas con los casos de uso, de tal manera que se puede conocer qué caso de uso deriva de qué característica. Aunque la Figura 42 es una captura de la herramienta RequisitePro (ibm.com/software/awdtools/reqpro/), las herramientas DOORS (ibm.com/software/awdtools/doors/), IRQA (visuresolutions.com/irqa-requirements-tool), CaliberRM (borland.com/us/products/caliber/) y Conture (jamasoftware.com) la ofrecen de forma muy similar.

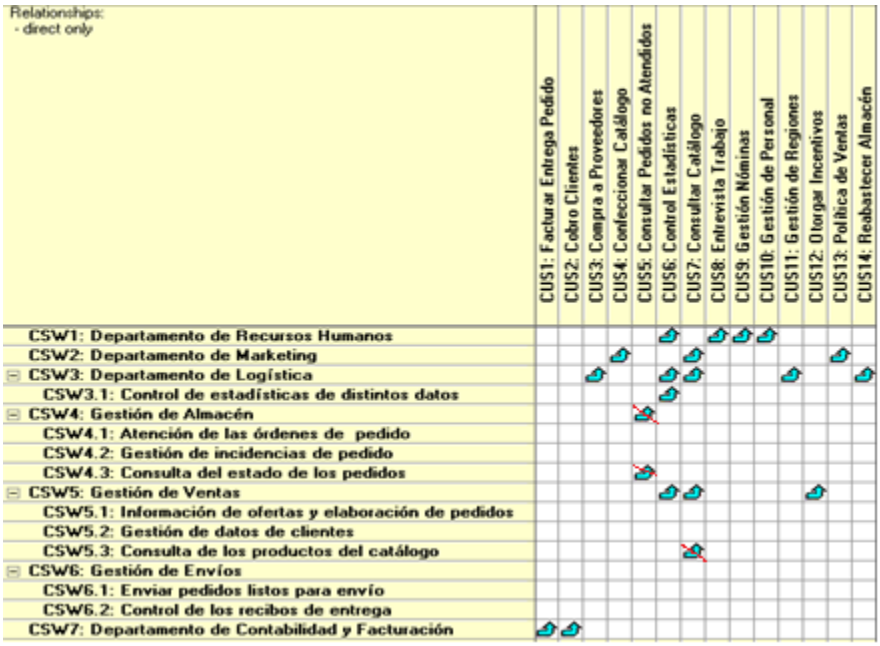


Figura 42. Matriz de trazabilidad de features con casos de uso en Requisite pro

Finalmente, desde el punto de vista de la especificación textual de requisitos y de los inconvenientes que conlleva utilizar lenguaje natural, existen muchos trabajos en los cuales, utilizando restricciones de sintaxis o incluso lenguajes formales, intentan evitar o reducir dichos inconvenientes. La motivación de estas propuestas normalmente es generar otros artefactos a partir de especificaciones de requisitos (usando técnicas de transformación de modelos) o realizar análisis de consistencia de la especificación de requisitos. En nuestro caso, el interés está centrado en el Análisis de Impacto, y sólo en el ámbito de los requisitos y pruebas de aceptación, sin pretender limitar o restringir el uso del lenguaje natural en la especificación textual de las PAs.

Capítulo 9. Conclusiones y Trabajo Futuro

Este Trabajo Final de Máster se ha estructurado en dos grande bloques, en primer lugar se ha presentado el innovador enfoque TDRE y en segundo lugar se ha expuesto cómo se ha incorporado Análisis de Impacto de Requisitos en el contexto de TDRE.

TDRE es un enfoque para ingeniería de requisitos basado en Pruebas de Aceptación. TDRE es una extrapolación del enfoque TDD hacia el ámbito de la ingeniería de requisitos y planificación del desarrollo del software. A diferencia de otros enfoques para gestión de requisitos, TDRE está totalmente integrado en un proceso de desarrollo (TUNE-UP) que tiene como protagonistas a las PAs. El analista especifica los requisitos en términos de PAs, posteriormente todo el trabajo de desarrollo de un cambio o un nuevo requisito se realiza a nivel de cada prueba de aceptación y teniendo como criterio de éxito su satisfacción.

En esencia, TDRE es simple pero a la vez potente, un requisito es un contenedor de comportamiento expresado como PAs. La estructura de requisitos del producto es un grafo dirigido que va evolucionando a lo largo de la vida del producto, especialmente en cuanto a la cantidad de nodos y los niveles de profundidad del grafo. La creación de un nuevo nodo se establece de forma pragmática según la relevancia del requisito y de la extensión (cantidad) y complejidad de las PAs que lo definen.

Para facilitar la gestión de una gran cantidad de PAs los nodos se pueden ir descomponiendo para reorganizar las PAs de una forma más cómoda y legible tanto para el equipo de desarrollo como para el cliente. Para especificar un requisito lo esencial es establecer las pruebas de aceptación que permitirán validar que el comportamiento esperado se ha incluido en el producto. Así, un cambio o nuevo requisito se define básicamente añadiendo, modificando o eliminando nodos, y consecuentemente añadiendo, modificando o eliminando PAs en dichos nodos.

Los enfoques tradicionales para la gestión de requisitos mantienen el artefacto requisito y las actividades para su elaboración separadas de otros artefactos y actividades del proceso de desarrollo. Esto dificulta la trazabilidad y obliga a un mayor esfuerzo en cuanto a validar la implementación con respecto de los requisitos. Debido a que las modificaciones o mejoras de requisitos tienden a ser mucho más frecuentes que los nuevos requisitos, si no se tiene una adecuada granularidad y modularidad de la

especificación de requisitos, es muy difícil definir y validar dichas modificaciones. Basta con pensar, por ejemplo, cuando se utiliza un Modelo de Casos de Uso, lo complicado que puede resultar organizar una iteración n-ésima de un producto en la cual se deben implementar muchas modificaciones y/o mejoras de Casos de Uso ya implementados en versiones previas. Lo que se debe planificar es el trabajo asociado a los cambios que debe realizarse sobre el producto, y en este caso dichos cambios no corresponderían a nuevos Casos de Uso. Por ejemplo, si la especificación de cada Caso de Uso se hace mediante plantillas, habría que comparar la plantilla original respecto de la que se propone modificada para así comprender las diferencias en el comportamiento. Además, la tendencia natural es que dichos Casos de Uso y sus especificaciones vayan creciendo y haciéndose cada vez más difíciles de gestionar. Todo esto nos confirma que esta estrategia no es sostenible en mantenimiento. Claro está, esto no suele cuestionarse en gran parte de los desarrollos industriales de software puesto que el mantenimiento suele hacerse directamente sólo en el código o de forma muy limitada (o inexistente) en el resto de artefactos.

En un principio nos resistimos a que la interfaz de usuario del producto condicionara su estructura de requisitos. Sin embargo, con el tiempo (en el ámbito del mantenimiento) esto resultó lo más acertado puesto que facilitaba, para el cliente y el equipo de desarrollo, el contextualizar los cambios de comportamiento que solicitaba el cliente. TDRE no obliga a que la organización siga siempre o en su totalidad dicha estrategia. Además, las facilidades ofrecidas para reorganizar la estructura de requisitos hacen que la estrategia de organización pueda cambiarse sin mayores inconvenientes.

TDRE ha evolucionado en un contexto de una PYME de desarrollo de software en el marco de varios proyectos universidad-empresa. TDRE se encuentra validado en uso desde hace más de cuatro años en la empresa con la cual colaboramos, en el proyecto industrial más grande en el cual se utiliza tenemos incluidas más de 10.000 PAs. En los primeros dos años de implantación de TDRE las PAs se especificaban en un documento de texto. De esta forma el equipo de desarrollo se fue adaptando al enfoque y el cambio no fue tan radical cuando se pasó a gestionar las Pas de forma más estructurada y precisa en formularios de TUNE-UP. Desde los últimos dos años ya se introducen las pruebas dentro de TUNE-UP mediante el módulo REM y mes a mes sacamos versiones mejorando e integrando nuevas funcionalidades.

Una vez ya teníamos TDRE implantado, la incorporación de AI resultaba relativamente sencillo, incluyendo en la especificación de las PAs referencias a entidades del Modelo de Domino. Considerando que toda especificación de requisitos incluye referencias al Modelo de Dominio de la aplicación, nuestra hipótesis de trabajo es "Si dos o más PAs comparten en su descripción elementos del Modelo de Dominio, un cambio en una de dichas PAs puede afectar a las otras". Así nuestra propuesta de Análisis de Impacto que constituye una solución novedosa, pragmática, y además con soporte automatizado para la identificación y mantenimiento de las relaciones de trazabilidad.

El Análisis de Impacto es un tema de gran interés y una de las principales motivaciones para definir y mantener trazabilidad entre artefactos software. Esta importancia es reconocida por toda la literatura de Ingeniería de Requisitos, sin embargo, el estado del arte en la teoría y en la práctica continúa siendo el mismo de hace dos décadas. Existen dos factores que explican esta situación; por un lado resulta evidente que disponer de relaciones de trazabilidad entre artefactos en un producto software requiere mucho esfuerzo, tanto en su definición como en su mantenimiento. El esfuerzo invertido no es fácil de rentabilizar cuando la explotación de dicha información es rudimentaria y poco detallada, tal como sucede al utilizar Matrices de Trazabilidad como soporte para dicha información. Por otra parte, los enfoques clásicos para Análisis de Impacto son demasiado ambiciosos en cuanto a intentar cubrir todo tipo de artefactos, desde requisitos hasta el código, lo cual incrementa el problema de esfuerzo antes mencionado. Otros enfoques para Análisis de Impacto ofrecen automatización, salvando así el inconveniente del esfuerzo en definición y mantenimiento de la trazabilidad, pero están centrados sólo en artefactos del ámbito del código. Esto no representa una significativa contribución teniendo en cuenta todas las facilidades que suelen incluir los IDEs actuales para determinar relaciones entre piezas de código.

Nuestro enfoque salva los impedimentos asociados al esfuerzo invertido mediante automatización en la definición y mantenimiento de la información necesaria para Análisis de Impacto, ya que las relaciones de trazabilidad se generan de forma automática al aplicar el parser a las PAs. También nos ofrece una serie de ventajas respecto de la especificación de requisitos, aunque estamos incluyendo referencias a entidades del Modelo de Dominio del tipo "Entidad.Entidad.Atributo" en la especificación de la prueba, se mantiene prácticamente el mismo grado de flexibilidad y legibilidad del lenguaje natural. Otra de las ventajas es que se gana en estandarización

de la especificación de requisitos ya que todos los analistas utilizan los mismos términos para hacer referencia a elementos del Modelo de Dominio en el contenido de la prueba.

Nuestra propuesta de AI enfocada en facilitar el trabajo de definición de requisitos, ámbito en el cual el Análisis de Impacto resulta muy apreciado. El Análisis de Impacto en requisitos resulta conveniente pues está en un nivel de abstracción más cercano a como vienen definidos los cambios solicitados por el cliente.

El Análisis de Impacto que ofrecemos en nuestra propuesta es más preciso que en los otros enfoques existentes, por el nivel de granularidad de las Pruebas de Aceptación y por la utilización del Modelo de Dominio. Cuando mostramos los grids de explotación e indicamos que dos PAs se encuentran relacionadas, fácilmente se puede saber el elemento del Modelo de Dominio que establece dicha relación.

TUNE-UP, con el enfoque TDRE incluido, está disponible para ser descargado gratuitamente desde www.tuneupprocess.com. La funcionalidad asociada a Análisis de Impacto desde hace un año se está utilizando y refinando en un proyecto I+D+I de la empresa con la cual colaboramos. Ha sido tan buena la aceptación de la propuesta que el equipo de analistas nos vienen solicitando mejoras continuas en cuanto a la infraestructura (Gestión del Modelo de Dominio, Intellisense, etc..), lo cual constituye en sí misma importantes mejoras para la especificación de PAs. Por este motivo estamos aplazando las mejoras de la explotación de la información, lo cual constituye gran parte de nuestros trabajos futuros. En breve la funcionalidad para Análisis de Impacto estará incorporada en la versión descargable de TUNE-UP.

Así, de momento no tenemos resultados definitivos respecto de la validación del enfoque de Análisis de Impacto en cuanto a su explotación (sólo hemos validado en uso la infraestructura), aunque tenemos garantizado que representará una mejora significativa pues los futuros usuarios de la herramienta han participado activamente en su concepción. Sin embargo, ya hemos detectado algunos aspectos importantes para mejorar y que constituyen nuestros próximos desafíos. Ellos son los siguientes:

- Se generan muchísimas relaciones entre PAs (y por consiguiente entre nodos). Aunque el estudio de las Dependencias Causa-Efecto permite centrarse en un subconjunto relevante de relaciones, aún resulta laborioso estudiar una por una las PAs posiblemente afectadas. Habría que incorporar otros mecanismos que

permitan determinar un subconjunto menor en el cual concentrar el estudio. Por ejemplo, se podrían utilizar pesos para las PAs y/o para las relaciones de forma que aquellas con mayor peso se remarquen. De forma similar podríamos utilizar la información histórica asociada a artefactos efectivamente afectados y a falsos positivos para destacar el conjunto de artefactos candidatos para ser revisados.

- Asociado a lo anterior, es muy importante ofrecer una forma de presentación más visual de los elementos y sus relaciones. Debería ser lo más cercano a un grafo, tal como está estructurada la información en la BDOG.
- Sería conveniente tener un mecanismo para realizar análisis predictivo (what if), sin tener que modificar el estado actual de la especificación, puesto que a veces debido a los resultados del Análisis de Impacto se decide explorar otras alternativas de cambio o simplemente descartar la propuesta de cambio debido al impacto que puede tener.

Finalmente quisiera destacar la experiencia profesional que ha significado para mí el desarrollo de este Trabajo Final de Máster. Desde hace dos años estoy trabajando con un equipo de trabajo desarrollando y mejorando la metodología TUNE-UP. Nos enfrentamos a un marco de trabajo en el que cada 3 semanas realizamos entregas de funcionalidad de la herramienta TUNE-UP Process Tool. Además, convivimos con los usuarios de la herramienta, y les ofrecemos soporte, entrenamiento y apoyo necesario, convivimos dentro de un entorno real de empresa. La implantación de TDRE en TUNE-UP ha sido un proceso costoso, debido al cambio que suponía en la forma de trabajar de los agentes, por este motivo se fueron haciendo cambios progresivamente y siempre escuchando las principales necesidades de usuarios que lo estaban utilizando o lo iban a utilizar. Actualmente TDRE está fuertemente implantado en la empresa y nadie pone en entredicho las múltiples ventajas que ha aportado, tanto a los usuarios que la utilizan día a día como a los jefes de proyecto.

Además del resultado práctico de este Trabajo Final de Máster se han generado tres publicaciones de investigación directamente asociadas a este trabajo, ellas son:

- Marante, M. Company, M. Letelier, P. Suarez, F. Gestión de requisitos basada en pruebas de aceptación: Test-Driven en su máxima expresión. XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010).

- Company, M., Letelier, P., Marante, M., Suarez, F. Análisis de impacto en requisitos soportado de forma semi-automática y en un marco de desarrollo TDD basado en pruebas de aceptación. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)
- Marante, M., Company, M., Letelier, P. Seguimiento ágil de proyectos de desarrollo de software utilizando Gráficas Burn Down. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)

Tanto los trabajos de investigación como el desarrollo de este trabajo han motivado a la autora a continuar su trabajo de investigación relacionado con la Ingeniería de Requisitos dirigida por las Pruebas de Aceptación.

Referencias

1. Marante, M. Company, M. Letelier, P. Suarez, F. Gestión de requisitos basada en pruebas de aceptación: Test-Driven en su máxima expresión. XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010).
2. Company, M., Letelier, P., Marante, M., Suarez, F. Análisis de impacto en requisitos soportado de forma semi-automática y en un marco de desarrollo TDD basado en pruebas de aceptación. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)
3. Marante, M., Company, M., Letelier, P. Seguimiento ágil de proyectos de desarrollo de software utilizando Gráficas Burn Down. XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2011)
4. Marante, M., Letelier, P., Suarez, F. *TUNE-UP: Seguimiento de proyectos software dirigido por la gestión de tiempos.* XIV Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2009). ISBN 978-84-692-4211-7 (pág. 57-68), Septiembre 2009.
5. Marante, M., Letelier, P., Suarez, F. *TUNE-UP: Un enfoque pragmático para la planificación y seguimiento de proyectos de desarrollo y mantenimiento de software*. I Congreso Iberoamericano SOCOTE – SOporte al COnocimiento con la TEcnología (SOCOTE 2009). ISBN 978-84-613-8585-0, Noviembre 2009.
6. Andersson, J., Bache, G., Sutton, P. XP with Acceptance-Test Driven Development: A Rewrite Project for a Resource optimization System, XP 2003, LNCS 2675, pp. 180–188, 2003.
7. Adzic, G. Test Driven .NET Development with FitNesse: second edition. Neuri Limited, ISBN: 978-0-9556836-2-6, 2009.
8. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley. 2000.
9. Beck, K. Test Driven Development: By Example. Addison Wesley, 2002
10. Cohn, M. User Stories Applied for Agile Software Development, Person Education, Inc., 2004.
11. Deng, C., Wilson, P., Maurer, F. *Fitclipse: A fit-based eclipse plug-in for executable acceptance test driven development*. In Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming. Springer, 2007.

12. Haugset, B., Hanssen, G., Automated Acceptance Testing: A Literature Review and an Industrial Case Study, Proc. of Agile 2008, pp. 27-38
13. Forsberg, K., Harold M. "*The Relationship of Systems Engineering to the Project Cycle*" Engineering Management Journal, 4, No. 3, pp. 36-43, 1992.
14. IEEE 830-1993, IEEE Recommended Practice for Software Requirements Specifications
15. Kroll P., Kruchten P., Booch G. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley Professional. 2003.
16. North, D. *Introducing BDD,* Better Software, March, 2006.
17. North, D. *What's story?* February 2007.
18. Reppert, T. *Don't Just Break Software, Make Software*, Better Software Magazine, July/August 2004
19. Roger, S. Acceptance testing vs. unit testing: A developer's perspective. Lecture Notes in Computer Science, 2004 - Springer
20. Schwaber K., Beedle *M. Agile Software Development with Scrum.* Prentice Hall, Series in Agile Software Development. 2001
21. Lee, M., Offutt, J., and Alexander, R. Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software. 34th International Conference on Technology of Object-Oriented Languages and Systems, 2000, pp. 61-70.
22. Göknil, A. and Kurtev, I. and van den Berg, K.G. Change Impact Analysis based on Formalization of Trace Relations for Requirements. European Conference on Model Driven Architecture (ECMDA) Traceability Workshop 2008, pp. 59-75
23. Gupta, C. Singh, Y. Chauhan, D. Dependency based Process Model for Impact Analysis: A Requirement Engineering Perspective. International Journal of Computer Applications 2010 , vol. 6, issue 6, pp. 28-33
24. Ajila, S. Software Maintenance: An Approach to Impact Analysis of Objects Change. Software: Practice and Experience. Volume 25, Issue 10, pp. 1155–1181, October 1995
25. Turver, R. Munro, M. An early impact analysis technique for software maintenance. Journal of Software Maintenance: Research and Practice. Volume 6, Issue 1, pp. 35–52, 1994
26. Jun Han. Supporting Impact Analysis and change propagation in SW Engineering Environments. 8th International In Proceedings. Workshop on Software

Technology and Engineering Practice incorporating Computer Aided Software Engineering 1997, pp. 172-182.

27. James S. O'Neal, Doris L. Carver. Analyzing the Impact of Changing Requirements. 17th IEEE International Conference on Software Maintenance (ICSM) 2001, pp. 190-195
28. Law, J. Rothermel, G. Whole program Path-Based dynamic impact analysis. 25th International Conference on Software Engineering (ICSE) 2003, pp. 308-318
29. Sommerville, I. Sawyer, P. Requirements Engineering: A Good Practice Guide. Chichester, England: John Wiley & Sons. 1997
30. Wiegers, K. Software Requirements. Microsoft Press 2003.
31. Bohner, S and Arnold, R. *Software Change Impact Analysis* IEEE Computer Society Press, Los Alamitos, CA, USA 1996.
32. J. Larus. Whole Program Paths. ACM SIGPLAN 1999 conference on Programming language design and implementation, pp. 259-269, Atlanta, GA, May 1999. ACM.
33. Letelier, P. A Framework for Requirements Traceability in UML-based Projects. 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, U.K. 2002 , pp. 32-41.
34. Letelier, P. Navarro, E. Anaya, V. Customizing Traceability in a Software Development Process. Information System Development: Advances in Theory, Practice and Education. O. Vasilecas, A. Caplinskas,W. Wojtkowski, W. Gregory, J. Zupancic, S. Wrycza (Editores). Springer. 2005 pp. 137-148.
35. Ramesh, B. Factors Influencing Requirements Traceability Practice. Communications of the ACM. 1998, pp. 37-44.
36. Ramesh, B., Jarke, M. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Vol. 27, No. 1, January 2001.
37. Koskela, L. Test Driven: Practical TDD and Acceptance TDD for Java Developers. October, 2007. ISBN: 1-932394-85-0
38. Briand, L.C. Labiche, Y. O'Sullivan, L. Impact analysis and change management of UML models. Proceedings of the International Conference on Software Maintenance (ICSM'03). 2003, pp. 256-266

39. Hayes, J.H. Dekhtyar, A. Sundaram, S. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. IEE Transactions on Software Engineering, Vol. 32, No. 1, January 2006.
40. Comparativa en herramientas
41. INCOSE, The International Council on Systems Engineering Requirements Management Tools Survey. Disponible en http://www.incose.org.
42. Humphrey, Watts S. The Personal Software Process. Software Process Newsletter, Technical Council on Software Engineering, IEEE Computer Society, Volume 13, No. 1, Sept. 1994, pp SPN 1-3.
43. Allen, D. Getting Things Done: The Art of Stress-Free Productivity. Published in Penguin Books 2003. ISBN 0 14 20.0028 0
44. Covey, S. The Seven Habits of Highly Effective People. Published Simon & Schuster, Limited. Edition Softcover. ISBN 0743269519

Anexo I – Pautas para la definición de PAs

1. **Ubicación de la PA**

Las pruebas deben especificarse en el nodo en el que se tengan que ejecutar los pasos a seguir para realizar la prueba.

Duda que suele surgir: ¿El efecto provocado por una opción de configuración se comprueba en el nodo de la opción de configuración o en donde se produce el efecto? En este caso la ubicación de la prueba dependerá de cómo queramos que funcione la opción. Las dos opciones posibles son las siguientes:

- El efecto de marcar la opción es inmediato y no depende de ninguna otra acción.

  Ejemplo: Opción "Concatenar Apellidos, Nombre o Nombre Apellidos" → al cambiar el valor de la opción, el efecto de cambiar la concatenación a todas las fichas se produce en el momento, aunque no entremos a las fichas, por lo que la prueba estaría en el nodo de configuración. La prueba sería similar a:

PASOS
✓ Cambiar la opción Concatenar Nombres
RESULTADO
→ Los nombres de las fichas de Clientes se muestran según la opción de concatenación seleccionada

- El efecto de marcar la opción se verá o afectará sólo al realizar otra acción.

  Ejemplo: Opción "Fecha fin de inscripciones [dd/mm/aaaa]" → el efecto se produce al acceder a una inscripción, borrarla o crear una inscripción: Estaría en el nodo de Inscripciones. La prueba sería similar a:

CONDICIONES
✓ Tener establecida una fecha fin de inscripciones mayor que la actual.
PASOS
• Crear Inscripción
RESULTADO
→ No se crea la inscripción y nos aparece un mensaje…

En todo caso, no representa mayor inconveniente el hecho que en caso como este se definan pruebas similares (especificando la misma funcionalidad) pero en nodos diferentes.

2. **Identificación de la PA**

En la identificación de la PA se pone un nombre a la PA. Algunas de las consideraciones o pautas que se deben tener en cuenta a la hora de definir el nombre a una PA son las siguientes:

- Los nombres de las PAs deberían ser lo más cortos y descriptivos posibles. Por ejemplo, "Interfaz" → Las PAs que sólo comprueban la interfaz se nombran así ya que es lo suficiente para saber a qué hace referencia la PA, ya que el lugar que estamos comprobando viene indicado por la ruta en la que se encuentra la prueba.

- El nombre debe ser intuitivo, para facilitar la localización de la PA por medio de búsquedas por nombre, es decir, cuando creemos la PA tenemos que pensar en cómo la buscaríamos después.

- No se deben poner nombres de PAs idénticos dentro de un mismo nodo, ya que es otro medio para identificar PAs.

- Para PAs que indiquen acciones manuales, se indicará el paso principal de la prueba, y en caso de haber varias pruebas respecto a una misma acción, se especificará también un resumen de las condiciones u otros pasos que las diferencien. Por ejemplo: "Crear Cliente de Tipo Premium".

- Los grupos de PAs relacionadas tendrán un nombre similar, para que resulte más fácil ver la relación. Por ejemplo, todas las pruebas relacionadas con mismo botón llevarán en el nombre el texto del botón.

482-1	Botón Eliminar cliente
483-1	Permiso Botón Eliminar cliente
484-1	Pulsar Botón Eliminar cliente
485-1	Pulsar Botón Eliminar cliente con rol administrador

También podría optarse, en caso de muchas PAs, por crear un subnodo que contenga la parte común, en este caso crearíamos un nodo hijo del nodo "Ficha cliente" con el nombre "Botón Eliminar cliente", y este contendría las pruebas: "Permiso", "Pulsar Botón" y "Pulsar Botón con rol administrador".

- Para PAs sobre procesos automáticos indicaremos el nombre del proceso. Por ejemplo: "Actualización", "Cobro automático", etc...

- PAs que indiquen fórmulas con las que se calculan datos, indicarán en su nombre: "Cálculo de " + dato que se calcula [+ condiciones]". Por ejemplo, "Cálculo de comisión al hacer una transferencia a una cuenta de otro banco".

3. **Definición de la PA**

Se utilizara una plantilla base para escribir las PAs, de forma que sean siempre similares.

CONDICIÓN

✓

PASOS

‣

RESULTADO ESPERADO

➔

OBSERVACIONES

ⓘ

Explicación y ejemplos de situaciones concretas en cada apartado:

- **Condiciones**

El apartado condiciones se utiliza para establecer condiciones previas, necesarias antes de realizar los pasos de la PA. **Normalmente se refieren al estado de la BD antes de realizar la PA**, por ejemplo, para aquellas PAs que requieran de alguna opción de configuración, en las condiciones tendremos todas las **opciones de configuración** que tengan que haberse modificado para poder ejecutar la PA.

El resto de información necesaria para la PA (Datos concretos de usuarios, facturas…) no se especificará en las Condiciones, es decir, todo lo que son datos concretos necesarios para conseguir el resultado que puedan tener distintos valores para pasar la PA, ya entran dentro de lo que sería **la Instanciación de Datos**. Por ejemplo, para una PA que implique acciones sobre un usuario de algún tipo, no es preciso indicar que tiene que haber un usuario de ese tipo creado. La excepción será en los casos en que el valor concreto de un dato realmente condicione el resultado, por ejemplo, "si el Cliente tiene saldo inferior a 0 aparece en el listado de morosos".

El apartado de Condiciones es opcional, ya que en algunos casos no es necesario disponer de configuración específica para conseguir el resultado, por ejemplo, para PAs que sólo comprueban que la interfaz sea la especificada.

Cada línea que comience por el símbolo "✓ ", será una condición de la PA que se debe cumplir, de esta forma se sabrá rápido de forma visual cuantas condiciones hay.

No hay que poner condiciones que nos indiquen desde qué partes de la aplicación se debe realizar la PA, ya que eso viene indicado por defecto en los diferentes caminos del nodo que contiene la PA dentro de la estructura del producto. Por ejemplo no hay que poner condiciones del estilo "Acceder a la pestaña de datos del cliente". La prueba al estar dentro del nodo "Pestaña datos del Cliente", sabemos que se tienen que ejecutar los pasos en todos los caminos de ese nodo.

En una condición podemos indicar varias sub-condiciones separadas por "o" en los siguientes casos:

- Que alternativamente producen el mismo resultado con los mismos pasos.

CONDICIONES
✓ Tener algún cliente de las siguientes provincias: Valencia, Alicante, Castellón, Ibiza, Mallorca, Barcelona, Tarragona, Lleida o Girona.
PASOS
‣ Ver ficha del cliente
RESULTADO ESPERADO
➔ Tiene la opción de recibir las notificaciones en catalán

- Para indicar que no afectan al resultado, pero que se enfatiza que se debe comprobar que realmente no afecte.

CONDICIONES

Tener un cliente que tenga:

- ✓ Marcada o desmarcada la opción "Incluir en facturación mensual".
- ✓ Conceptos facturables pendientes de facturar

PASOS

- ⏵ Realizar la facturación de Otros conceptos facturables

RESULTADO

- ➔ El cliente aparece en el árbol de conceptos facturables pendientes de facturar.

Las condiciones pueden agruparse en grupos de condiciones según a qué hagan referencia cuando queremos indicar varias condiciones sobre un mismo elemento.

CONDICIONES

✓ **Tener un cliente con:**

- Número de movil
- Fecha salida del hotel ==Fecha actual
- Cuentas pendientes de pagar

✓ **Tener en configuración:**

- Marcado check Notificar salida al cliente por sms
- Marcado check Notificar las cuentas pendientes por sms

- **Pasos**

Son las acciones de interacción del usuario con el sistema que desencadenan el proceso o procesos necesarios para que se produzca el resultado, realizadas a partir de las condiciones indicadas en la PA (Ya sea por interfaz, o por medio de otra aplicación).

En una PA se pueden especificar uno o varios pasos indicando el orden en el que deben aplicarse, deben ser simples, y comenzar con un verbo del estilo "visualizar", "seleccionar", "intentar", evitando cualquier tecnicismo que dificulte su validación con el usuario.

En caso de que existan varias formas (pasos) de conseguir un mismo resultado crearemos varias PAs, una por cada forma. Para facilitar el mantenimiento de las PAs puede separarse la especificación del resultado también en una PA independiente, de forma que en las primeras PAs se verifique que por cada camino se lanza la acción (PA 1 y 2 del ejemplo siguiente), y en la otra (PA 3 del ejemplo siguiente) se comprueben los detalles de funcionamiento de la acción resultante.

- PA 1→ Botón Nueva Inscripción

PASOS
- Pulsar botón Nueva Inscripción

RESULTADO
➔ Se realiza el **Proceso de creación de una inscripción**.

OBSERVACIONES
El proceso de creación de inscripción se comprueba en la *Prueba 3*

- PA 2 → Opción Nueva Inscripción del menú contextual

PASOS
- Pulsar botón Nueva Inscripción del menú contextual

RESULTADO
➔ Se realiza el **Proceso de creación de una inscripción.**

OBSERVACIONES
El proceso de creación de inscripción se comprueba en la *Prueba 3*

- PA 3 → Proceso de creación de inscripción

PASOS
- Realizar el **Proceso de creación de inscripción**

RESULTADO
➔ Se crea una nueva inscripción en blanco, sólo se ha rellenado el campo Fecha de alta.

Cuando se especifique un Paso que no es seguro que se vaya a completar o que no se vaya a completar indicaremos "Intentar…".

PASOS
- ~~Eliminar Factura →~~ Intentar eliminar la factura

RESULTADO
➔ Nos aparece el siguiente mensaje de confirmación: ¿Esta seguro que desea eliminar la factura?

Si especificamos "Eliminar la factura" se asume que se realiza la acción completa, con lo que ya se habría aceptado el mensaje.

En una PA sólo se debe establecer un punto de verificación al final de los pasos, es decir, podemos tener varios resultados parciales, pero todas las comprobaciones indicadas en ellos deben poder realizarse una vez completados los pasos. Por ejemplo si se tiene una prueba PA1 con Paso 1, Paso 2, Paso 3 → Resultado, pero también se quiere comprobar que tras los pasos 1 y 2 se produce el Resultado 1, tendríamos que

crear una prueba adicional PA2 con Paso 1, Paso 2 → Resultado 1. En vez de utilizar en la PA1 el formato Paso 1, Paso 2 → Resultado 1, Paso 3 → Resultado. A continuación se muestra un ejemplo real de prueba incorrecta.

CONDICIONES

- ✓ Tener permiso de Ver, Editar y Eliminar Productos
- ✓ Tener un producto dado de alta

PASOS

- Pulsar el botón derecho sobre el producto →**Resultado 1:** La opción "Asignar producto a" está habilitada
- Abrir el desplegable de "Asignar producto a" → **Resultado 2:** Aparecen por orden alfabético los nombres de todos los clientes activos
- Elegir un cliente del desplegable

RESULTADO

- ➔ La columna Stock del grid de productos se ha actualizado correctamente, restándole una unidad.

La PA anterior estaría definida incorrectamente y se tendría que estructurar para no tener resultados parciales. A continuación se muestran las tres PAs que tendrían que substituir la anterior.

- PA1 → Desplegable "Asignar producto a" habilitado

CONDICIONES

- ✓ Tener permiso de Ver, Editar y Eliminar Productos
- ✓ Tener un producto dado de alta

PASOS

- Pulsar el botón derecho sobre el producto

RESULTADO

- La opción "Asignar producto a" está habilitada

- PA2 → Formato desplegable "Asignar producto a"

CONDICIONES

- ✓ Tener permiso de Ver, Editar y Eliminar Productos
- ✓ Tener un producto dado de alta

PASOS

- Pulsar el botón derecho sobre el producto
- Abrir el desplegable de "Asignar producto a

RESULTADO

- ➔ Aparecen por orden alfabético los nombres de todos los clientes activos.

- PA3 → Asignar Producto a un Cliente

CONDICIONES

- ✓ Tener permiso de Ver, Editar y Eliminar Productos
- ✓ Tener un producto dado de alta

PASOS

- Pulsar el botón derecho sobre el producto
- Abrir el desplegable de "Asignar producto a"
- Elegir un cliente del desplegable

RESULTADO

→ La columna Stock del grid de productos se ha actualizado correctamente, restándole una unidad.

En determinados casos puede ser interesante unir en el contenido de una misma PA varias acciones con los resultados que generan, ya sean consecutivos a alternativos. Un caso claro son cuando tenemos opciones que dependen unas de otras. Por ejemplo, en casos de combinaciones de checks que impliquen que al marcar o desmarcar unos se desmarquen o marquen otros, podemos en vez de crear una PA por cada combinación crear una única PA de "Combinaciones de <Opc 1> <Opc2>". A continuación se muestra una PA modelo de este caso en particular.

- PA → Combinaciones de Opción A y Opción B

PASOS

Probar todas las combinaciones posibles de las opciones A y B

RESULTADO

Comprobar que se cumple cada caso de la siguiente tabla:

Teniendo	Probar	Resultado
Opción A desmarcada	Marcar Opción B	A y B marcadas.
Opción A marcada	Marcar Opción B	A y B marcadas.
Opción B desmarcada	Marcar Opción A	A marcada, B desmarcada.
Opción B marcada	Marcar Opción A	A y B marcadas.
Opción A desmarcada	Desmarcar Opción B	No permitido:
Opción A marcada	Desmarcar Opción B	A marcada, B desmarcada.
Opción B desmarcada	Desmarcar Opción A	A y B desmarcadas.
Opción B marcada	Desmarcar Opción A	A y B desmarcadas.

- **Resultado Esperado**

Son los efectos que deberíamos obtener tras realizar los pasos indicados, pueden producirse uno o más resultados.

Siempre que sea posible se especificarán los resultados de forma externa, desde la perspectiva de un usuario. Sólo en casos excepcionales en que no pueda hacerse de otra forma, podrá especificarse en la PA una comprobación de otro tipo, por ejemplo, comprobando en base de datos.

En principio los resultados se agruparán por funcionalidad, es decir, todos los resultados que se produzcan al realizar determinados pasos en ciertas condiciones se establecerán en una misma PA siempre que respondan a un mismo objetivo. La idea es que luego estas PAs se puedan utilizar por separado cuando una cosa no tiene por qué afectar a la otra.

Los resultados no deberían ser condicionales, es decir, no se debería indicar en un resultado "Si ocurriese tal cosa o se produjese otro paso pasaría X" sino que habría que crear una PA aparte para ese caso. A continuación se muestra un ejemplo de PA incorrecta para este caso.

- PA → No editar el grid de ventas con Fecha < Fecha Actual

CONDICIONES

✓ Tener seleccionada en el grid una venta con Fecha < Fecha Actual

PASOS

▸ Intentar editar cualquier campo del grid

RESULTADO

➔ No se permite editar ningún campo excepto que el agente que lo esté modificando sea el agente que ha realizado la venta entonces se podría editar el campo 'Notas'

En el caso de querer indicar una excepción al comportamiento general, como en la PA anterior, se deberá especificar una PA independiente para el caso excepcional. Por lo tanto la PA anterior se tendría que dividir en las dos siguientes:

- PA → No editar los campos Cantidad, Producto, Referencia e Importe del grid de ventas con Fecha < Fecha Actual

CONDICIONES

✓ Tener seleccionada en el grid una venta con Fecha < Fecha Actual

PASOS

- Intentar editar los campos Cantidad, Producto, Referencia e Importe del grid

RESULTADO

➔ No se permiten editar

- PA -> Editar el campo Notas del grid de Ventas con Fecha<Fecha actual

CONDICIONES

✓ Tener seleccionada en el grid una venta con Fecha < Fecha Actual

✓ Ser el agente que ha realizado la venta

PASOS

- Editar el campo notas

RESULTADO

➔ Se permiten editar

➔ El resultado se guarda correctamente

4. **Instanciación de Datos**

Aunque es el tester el encargado de buscar las diferentes combinaciones de datos adecuadas para las diferentes instanciaciones de la PA, es importante para la persona que define la PA conocer de qué se trata para evitar entrar en la instanciación de datos cuando se está escribiendo la PA.

Las principales diferencias entre una PA y su diseño son las siguientes:

- La PA es un escenario desde la perspectiva del cliente que solicita el comportamiento asociado, el analista define las PA y si es posible el cliente las valida
- El diseño de la PA es un escenario o más en ejecución del sistema, definidos por el tester. La PA NO está instanciada. Cada PA tendrá una o más instanciaciones de datos.

A continuación se muestra un ejemplo de una PA y posteriormente sus correspondientes instanciaciones que habrá definido el tester.

CONDICIONES

- ✓ Tener desmarcado el check "Permitir introducir el carácter + como parte del número de teléfono"

PASOS

- ▸ Intentar introducir caracteres especiales que no sean + o letras en el campo teléfono

RESULTADO

- ➔ No se escriben en el campo

La PA es muy clara ya que explica cuál es el resultado, pero deja abiertas las opciones al tester para realizar las comprobaciones. En este caso, en el diseño que se muestra a continuación se pueden ver las distintas instanciaciones de la misma PA.

Descripción del diseño	**Resultado**
Intentar insertar letras "aa" en el campo teléfono	Letras en minúsculas no se insertan y el campo permanece sin datos
Intentar insertar letras "AA" en el campo teléfono	Letras en mayúsculas no se insertan y el campo permanece sin datos
Intentar insertar caracteres "&&" en el campo teléfono	Caracteres no se insertan y el campo permanece sin datos
Intentar insertar caracteres "**" en el campo teléfono	Caracteres no se insertan y el campo permanece sin datos
Intentar insertar carácter "+" en el campo teléfono	El carácter se inserta y el campo tiene datos

Anexo II – Código del parser de texto en rtf para detectar automáticamente las entidades referenciadas en una PA

Parámetros de entrada: el control del texto, el tipo de PA (propuesta, actual), el tipo de la relación que se va a crear (dependiendo de la parte de la PA que se va a parsear)
Parámetro de referencia: una lista con las entidades que se han creado correctamente

```
public void CrearRelacionNodos(Moonlight.CodeTextBox c, string TipoPA,
GraphDB.TipoRel tipo, ref  List<string[]> lRelaciones )
{
    List<string[]> listaRelaciones = new List<string[]>();
    string[] lista = c.Text.Split(' ');
    int k;
    for ( k = 0; lista.Length > k; k++)
    {
         if (!string.IsNullOrEmpty(lista[k]))
         {
            break;
         }
    }

    if (k == lista.Length)
    {
      k--;
    }

    if (lista.Length > 0)
    {
        if (lista[k].Contains('.'))
        {
           lista = lista[k].Split('.');
        }
    }



    string textortf= "";
    c.Save(out textortf, XTextControl.StringStreamType.RichTextFormat);
    if(string.IsNullOrEmpty (lista[k]))
        return;
    //Quitamos caracteres especiales que no aparecem tal cual en el rtf
     string[] auxCarEspeciales = new string[3];
    auxCarEspeciales[0] = "\t";
    auxCarEspeciales[1] = "\r";
    auxCarEspeciales[2] = "\n";

int pos= textortf.IndexOf(lista[k].Split(auxCarEspeciales,
StringSplitOptions.RemoveEmptyEntries)[0]);
string cuerpoTexto = textortf.Substring(pos);
```

Obtenemos del texto (sin formato rtf) la primera palabra y buscamos en el texto con formato rtf la posición de la primera palabra que se ha introducido. De esta forma eliminamos la cantidad de etiquetas de formato que se introducen en el rtf al principio y no nos interesan. Al final la variable cuerpoTexto contiene la parte del texto con formato rtf que nos interesa.

```
//Creamos un array con las etiquetas que nos indican
que empieza texto con el formato en azul
string[] aux1 = new string[2];
aux1[0] = "\\plain\\f1\\fs20\\cf5";
aux1[1] = "\\plain\\f2\\fs20\\cf5";
//Separamos el texto rtf con las etiquetas
de inicio de formato
string[] lista1 = cuerpoTexto.Split(aux1,
StringSplitOptions.RemoveEmptyEntries);
for (int i = 0; i < lista1.Length; i++)
{
    string[] ls = new string[2];
//Nos interesa obtener los dos últimos niveles
de la entidad por ejemplo Cliente.Nombre
    string nombreNodoHoja = "";//Nombre
    string nombreNodoPadre = "";//Cliente
```

Separamos el rtf con las etiquetas que nos indican que empieza el formato que queremos parsear y se crean las variables donde nos guardaremos el valor de las entidades referenciadas en la prueba

Si el texto que estamos evaluando no contiene la etiqueta \cf4 (vuelta al formato normal) y no es el último item del array, eso quiere decir que ya se ha encontrado el nodo padre.

```
    if (!lista1[i].Contains("\\cf4") && (i+1)!=lista1.Length)
    {
       //lista1[i] es una clase
        nombreNodoPadre = lista1[i].Trim();

       //miramos si tiene algun atributo o asociacion con otra clase,
      por ejmplo Cuenta.Nombre. Si tiene una relación está se
      encontrara en la siguiente posición del array y empezara por
      punto y la etiqueta \cf4
        if (lista1.Length > (i + 1))
       {

         if (lista1[i + 1].TrimStart(' ').StartsWith(".\\plain"))
         {
             string[] aux2 = new string[1];
             aux2[0] = "\\cf4";
             string[] array=lista1[i + 1 ].Split( aux2,
    StringSplitOptions.RemoveEmptyEntries);
           if (array.Length > 1)
           {
             nombreNodoHoja = array[1].Trim();
           }
         }
       }
```

Si el texto que estamos evaluando contiene la etiqueta \cf4 (vuelta al formato normal) y no empieza con .\plain (sería un nodo hoja), entonces tenemos texto normal de la PA sin formato y después un nodo padre

```
    }
      else if (lista1[i].Contains("\\cf4") && !lista1[i].TrimStart('
').StartsWith(".\\plain"))
      {
         //hay texto normal y una clase
        string[] aux2 = new string[1];
        aux2[0] = "\\cf4";
        string[] array = lista1[i].Split(aux2,
StringSplitOptions.RemoveEmptyEntries );
        if (array.Length > 1)
        {
           nombreNodoPadre = array[1].Trim();
        }

        //miramos si tiene algun atributo o asociacion con otra clase
        if (lista1.Length > (i + 1))
        {
           if (lista1[i + 1].TrimStart(' ').StartsWith(".\\plain"))
           {
               string[] array1 = lista1[i + 1].Split(aux2,
StringSplitOptions.RemoveEmptyEntries);
               if (array1.Length > 1)
               {
                   if (!array1[1].Contains("\\"))
                      nombreNodoHoja = array1[1].Trim();
                   else
                   {
                     //despues de nodo hoja hay mas texto
                     nombreNodoHoja = array1[1].Split('\\')[0].Trim();
                   }
                }
            }
         }
```

Si el texto que estamos evaluando contiene la etiqueta \cf4 dos veces y además empieza por un nodo hoja (.\plain), entonces pasamos a parsearlo porque eso nos indica que al principio hay un nodo hoja que no nos interesa porque ya lo hemos parseado, pero al final hay referenciado un nodo padre.

```
      }

      else if (lista1[i].Contains("\\cf4") && lista1[i].TrimStart('
').StartsWith(".\\plain"))
    {

       string[] aux2 = new string[1];
       aux2[0] = "\\cf4";
       string[] array = lista1[i].Split(aux2,
StringSplitOptions.RemoveEmptyEntries );
       if (array.Length > 2)
       {
```

```
       //al inicio tiene un nodo hoja que ya se ha guardado y al final
      tiene un nodo padre

       nombreNodoPadre = array[2].Trim();


      //miramos si tiene algun atributo o asociacion con otra clase
      if (lista1.Length > (i + 1))
     {
        if (lista1[i + 1].TrimStart(' ').StartsWith(".\\plain"))
       {
         string[] array1 = lista1[i + 1].Split(aux2,
StringSplitOptions.RemoveEmptyEntries);
         if (array1.Length > 1)
         {
             if (!array1[1].Contains("\\"))
                 nombreNodoHoja = array1[1].Trim();
             else
             {
                 //despues de nodo hoja hay mas texto
               nombreNodoHoja = array1[1].Split('\\')[0].Trim();
             }
         }
       }
     }
   }
 }


//nombreNodoHoja y nombreNodoPadre es el nodo con el cual creamos la
relación PA-concepto
if (!string.IsNullOrEmpty(nombreNodoPadre))
{
     ls[1] = nombreNodoPadre;
     if (!string.IsNullOrEmpty(nombreNodoHoja))
     {
        ls[0] = nombreNodoHoja;
     }
}

//Añadir a las relaciones siempre que no existan todavía
bool repetido=false;
foreach (string[] s in listaRelaciones)
{
  if ((ls[0] != null && s[0] == ls[0]) &&
(ls[1] != null && s[1] == ls[1]))
  {
      repetido = true;
      break;
  }
}
if (!repetido)
{
   if(ls[1]!=null || ls[0]!=null)
      listaRelaciones.Add(ls);
}

//Mostrar mensaje parseado erroneo
if ((ls[0]!=null && ls[0].Contains('\\')) || (ls[1]!=null &&
ls[1].Contains('\\')))
{
```

Nos guardamos las entidades, siempre que no estuvieran ya, en un un array donde tenemos todas las entidades que se han encontrado al parsear un determinado apartado de la PA.

Se crean las relaciones en la BDOG, indicando el identificador de la PA, el tipo de la PA, el listado de relaciones detectadas, y el tipo con el cual se etiquetará la relación. El parámetro de salida result nos indica si se ha creado correctamente cada relación identificada.

```
 //Crear las relaciones de las PAs
 List<bool> result= new List<bool>();
 if (TipoPA == "Actual")
 {
   graphdb.añadirRelacionesPA(idPA, GraphDB.TipoPa.Actual
,listaRelaciones, tipo, out result);

 }else if (TipoPA == "Propuesta")
 {    graphdb.añadirRelacionesPA(idPA, GraphDB.TipoPa.Propuesta,
listaRelaciones, tipo, out result);
```