UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Integración de un editor gráfico de workflows en una herramienta de apoyo a la gestión ágil del trabajo

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Jesús Martínez Hernández

**Tutor**: Dr. Patricio Letelier Torres

Curso 2018-2019

# Resumen

En el presente Trabajo de Fin de Grado se trata el desarrollo de un nuevo componente Angular para su posterior integración en la versión web de Worki, herramienta para la gestión ágil del trabajo que emplea TUNE-UP Process, un framework propio, para la definición de metodologías ágiles. Dicho componente complementa al editor tabular de Worki y, al igual que éste, permite al usuario gestionar los workflows del sistema. Estos workflows son clave para el correcto funcionamiento de la herramienta, pues definen el flujo de las actividades que pasan por el tablero kanban.

**Palabras clave:** Workflow, Editor gráfico, Tablero kanban, Angular.

# Abstract

This Final Degree Project talks about the development of a new Angular component for the web version of Worki, a tool to manage work in an agile way that uses TUNE-UP Process, its own framework, to define agile methodologies. Said component supplements Worki's grid-based editor and allows the user to manage the system workflows. These workflows are key to make the tool work properly since they define the flow activities will follow through the kanban board.

**Keywords :** Workflow, Editor gráfico, Tablero kanban, Angular.

# Agradecimientos

En especial a mis padres y familiares que me han acompañado y apoyado hasta aquí desde el día en el que mi corazón empezó a latir por su cuenta.

A todos aquellos amigos que marcaron un antes y un después en mi vida y que me ayudaron a superarme y crecer como persona.

A Patricio Letelier y al equipo de desarrollo de Worki web por darme esta oportunidad, depositar su confianza en mí y ayudarme durante toda la creación de este trabajo de fin de grado.

Por último, me gustaría dar las gracias a todos los amigos que he hecho durante el grado que me han brindado risas y buenos momentos a lo largo de estos cuatro años de esfuerzo, estrés y crecimiento personal.

# Tabla de contenidos





# Capítulo 1 Introducción

El uso de metodologías ágiles [23] en el entorno de desarrollo de software está cada vez más extendido y aceptado como alternativa a métodos más tradicionales. Uno de los factores que han propiciado esta situación ha sido el desarrollo y uso de herramientas de apoyo a la gestión ágil del trabajo, que facilitan la implementación de prácticas ágiles dentro del ámbito empresarial ayudando, así, a un mejor seguimiento y cuantificación del trabajo realizado.

Worki es una de esas herramientas, la cual da apoyo a un framework propio llamado TUNE-UP Process[1]. Este framework permite a un equipo de desarrollo establecer una metodología adaptada a las necesidades de su contexto de trabajo.

Una buena metodología de trabajo se basa en unos procesos bien definidos que ayudan a la coordinación de los equipos de trabajo. En la actualidad se ha popularizado la visualización del trabajo mediante tableros kanban [24]. Un tablero kanban muestra en columnas una secuencia de actividades, es decir, un proceso secuencial. El diseño de tableros kanban para un contexto de trabajo no es más que el diseño de los workflows [22] que luego se representarán en dichos tableros kanban. Algunas herramientas ya incorporan editores de workflows para este propósito, sin embargo, presentan deficiencias en cuanto a la coherencia del uso de tableros respecto de: uso al mismo tiempo de Sprints, situaciones multi-proyecto, excepciones al flujo secuencial (actividades en paralelo), etc.

TUNE-UP Process utiliza un sistema kanban en el cual se muestra el trabajo representado por unidades de trabajo. Éstas pasan por diferentes estados, o actividades de un workflow, hasta su finalización.

## 1.1. Motivación

Worki es una herramienta de apoyo a la gestión ágil de proyectos desarrollada y utilizada académicamente en el ámbito de la escuela de informática de la UPV. En Worki se ha implementado una variante de tableros kanban que resuelve las deficiencias antes mencionadas a través de la idea de “workflows flexibles” [21].

El diseño y mantenimiento de los workflows es fundamental para el correcto funcionamiento de la metodología. Worki ofrece funcionalidades para la creación y edición de los elementos metodológicos, pero la gestión de workflows se lleva a cabo dentro de una tabla que muestra la información del workflow, una forma poco amigable y visual que choca con la dinámica flexible y ágil de la herramienta.

[1] http://www.tuneupprocess.com/

## 1.2. Objetivos

El objetivo de este TFG es integrar en Worki un editor gráfico de workflows que facilite su gestión. Para esto se evaluarán componentes que realicen diagramas para aplicaciones web, en particular que se puedan integrar fácilmente en Angular (la tecnología front-end de Worki). El desafío es que las funcionalidades asociadas a dicha idea de “workflows flexibles” en Worki se mantenga también en el editor gráfico de workflows. La validación del editor gráfico se realizará con su utilización en los proyectos que se están gestionando actualmente con Worki.

## 1.3. Estructura del trabajo

En este trabajo se muestra el proceso de desarrollo del componente Angular para la gestión gráfica de workflows. Seguidamente, se presenta la estructura de este trabajo de fin de grado.

En el capítulo uno se hace una introducción a la motivación del trabajo, explicando qué se pretende conseguir de él y en qué consiste.

En el capítulo dos se realiza una introducción a BPMN y se explica la relación que guardan con los workflows.

En el capítulo tres se habla de las metodologías para el desarrollo software, centrándonos en las ágiles y comentando brevemente las tradicionales. Además se ponen ejemplos de metodologías y se explican sus características y filosofía de trabajo.

En el capítulo cuatro se comenta brevemente la importancia que tienen los workflows dentro de las metodologías ágiles. Además se explica los diferentes elementos de la herramienta Worki y cómo estos están relacionados con los workflows.

En el capítulo cinco se habla sobre la tecnología que se va a emplear para el desarrollo del componente y se realiza un estudio para comparar y elegir un componente que nos ayude a crear el editor gráfico.

En el capítulo seis se detalla la especificación del componente a desarrollar a partir del sistema de edición actual. Se determinan los casos de uso y se habla sobre el diseño de la solución, en esta última parte se tratan los diferentes retos que han surgido durante el desarrollo y cómo se han solucionado.

En el capítulo siete se proporciona una pequeña guía de uso del editor desarrollado, mostrando su interfaz, sus diferentes componentes y las verificaciones que realiza.

En el capítulo ocho se realiza una conclusión del trabajo tras finalizar el proceso de desarrollo y se habla sobre el trabajo futuro surgido y lo aprendido durante este trabajo de fin de grado.

# Capítulo 2 Business Process Modeling Notation

Se entiende como workflow, o flujo de trabajo, a la automatización parcial o total de un proceso de negocio, es decir, una colección de actividades o tareas relacionadas y estructuradas que en una secuencia específica produce un servicio o producto.

Business Process Modeling Notation (BPMN) es la notación de referencia para el diseño de workflows. Para entender el modelado de workflows hemos de entender BPMN y, por tanto, en este capítulo se hablará en detalle de él.

## 2.1. Introducción a BPMN

Los contenidos de este capítulo han sido extraídos y traducidos del artículo "Introduction to BPMN" de Stephen A. White [2].

BPMN fue desarrollada por la *Business Process Management Initiative* (BPMI). La especificación de la versión 1.0 de BPMN salió al público en Mayo de 2004. Dicha especificación representa más de dos años de esfuerzo por parte de la *BPMI Notation Working Group*. Su principal objetivo era ofrecer una notación capaz de ser entendida por todos los participantes del proyecto, desde los analistas del negocio que crear los borradores iniciales de los procesos, los desarrolladores técnicos responsables de implementar la tecnología que llevará a cabo esos procesos, hasta el cliente del negocio que gestionará y monitorizará dichos procesos. BPMN está apoyado en un modelo interno que genera el ejecutable BPEL4WS. Así, BPMN crea un enlace estandarizado que acerca el proceso de diseño al de implementación.

BPMN define un *Business Process Diagram* (BPD), el cual está basado en una técnica de grafos de flujo diseñada para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de elementos gráficos, que son actividades y los controles de flujo que definen su orden de ejecución.

## 2.2. Fundamentos del BPMN

Un BPD está compuesto por una serie de elementos gráficos. Estos elementos permiten el fácil desarrollo de diagramas simples que serán familiares a la mayoría de analistas de negocio. Los elementos fueron elegidos para ser distinguibles los unos de los otros y utilizando formas familiares para la mayoría de modeladores. Por ejemplo, las actividades son rectángulos, y las decisiones son rombos. Cabe resaltar que una de las motivaciones para el desarrollo del BPMN es crear un mecanismo simple que sirva para crear modelos de procesos de negocio y que, al mismo tiempo, sea capaz de tolerar la inherente complejidad de los procesos de negocio. El enfoque que se tomó para solucionar el problema que generan estos dos requisitos en conflicto fue organizar los aspectos gráficos de la notación en categorías específicas. Esto ofrece

un pequeño set de categorías de notaciones para que el sistema de renderizado de un BPD pueda fácilmente reconocer los tipos básicos de elementos y entender el diagrama. En estas categorías básicas de elementos, se puede añadir variaciones e información adicionales que apoyen los requerimientos más complejos sin cambiar drásticamente el *look-and-feel* básico del diagrama. Las cuatro categorías de elementos son:

- Objetos de flujo
- Objetos conectores
- *Swimlanes*
- Artefactos

### 2.2.1. Objetos de flujo

Un BPD tiene un set de (tres) elementos básicos, que son los objetos de flujo, que hacen que los modeladores no tengan que aprender y reconocer una gran cantidad de distintas formas. Los tres objetos de flujo son:

- **Evento:** un evento se representa mediante un círculo y es algo que "ocurre" durante el curso de un proceso de negocio. Estos eventos afectan al flujo del proceso y normalmente tienen una causa (activador) y un efecto (resultado). Los eventos son círculos con el centro libre para permitir a los marcadores internos diferenciar entre activadores y resultados. Hay tres tipos de eventos, basados en el momento en el que afectan al flujo: Inicio, Intermedio y Fin.


**Figura 1 Tipos de evento**

- **Actividad:** una actividad se representa usando un rectángulo de esquinas redondeadas y un término genérico sinónimo del trabajo que una organización realiza. Una actividad puede ser atómica o no atómica (compuesta). Los tipos de actividad son: tareas y subprocesos. Los subprocesos se distinguen usando una pequeño símbolo de más en la parte inferior central de la figura.


**Figura 2 Representación de una actividad**

- **Gateway (compuerta):** una *gateway* se representa con el típico rombo y se usa para controlar la divergencia y convergencia de flujos de secuencia. Así,

éste determina la decisiones tradicionales, así como la creación, fusión o unión de caminos. Los marcadores internos indicarán el tipo de control de comportamiento.

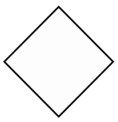


**Figura 3 Representación de una gateway**

## 2.2.2. Objetos conectores

Los objetos de flujo son conectados entre sí en un diagrama para crear el esqueleto básico de un proceso de negocio. Hay tres objetos conectores que hacen esta función. Estos conectores son:

- **Flujo de secuencia:** un flujo de secuencia se representa con una línea sólida con una punta de flecha también sólida y se usa para mostrar el orden (de la secuencia) en el que las actividades se ejecutarán dentro de un proceso. Nótese que el término “flujo de control” no es generalmente usado en BPMN.
- **Flujo de mensaje:** un flujo de mensaje se representa con una línea discontinua con una punta de flecha vacía y se usa para mostrar el flujo de mensajes entre dos participantes del proceso separados que envían y reciben. En BPMN, dos *pools* separadas en el diagrama representan a los dos participantes.
- **Asociación:** una asociación se representa con una línea de puntos con punta de flecha de una sola línea y se usa para asociar datos, texto y otros artefactos con objetos de flujo. Las asociaciones se usan para mostrar los *inputs* y *outputs* de las actividades.


**Figura 4 Objetos conectores**

Para los modeladores que necesitan o desean un bajo nivel de precisión a la hora de crear modelos de proceso para la documentación y comunicación, los elementos básicos junto a los conectores dan la posibilidad de crear diagramas comprensibles de forma sencilla.

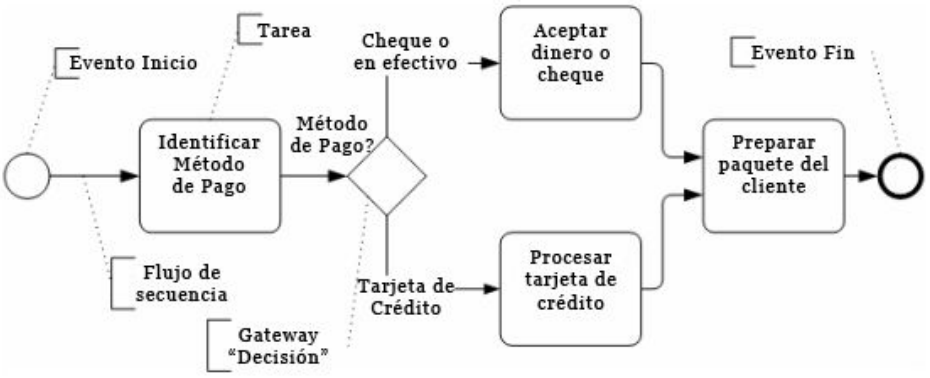



**Figura 5 Ejemplo de un diagrama de baja precisión**

Para los modeladores que necesitan o desean un mayor nivel de precisión a la hora de crear modelos, que serán sujetos a un análisis minucioso o administrados por un *Business Process Management System* (BPMS), los detalles adicionales se pueden añadir a los elementos básicos y mostrarse mediante marcadores internos.

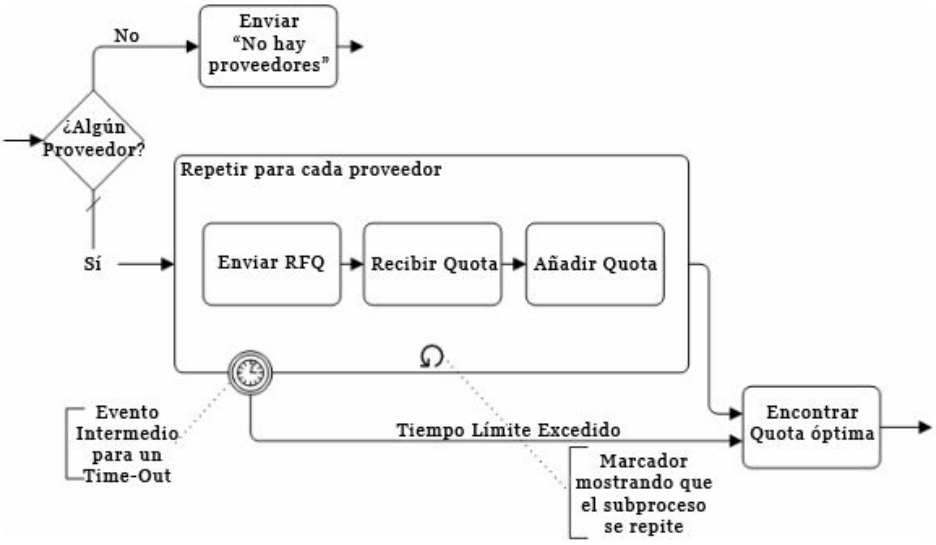



**Figura 6 Ejemplo de un diagrama de alta precisión**

## 2.2.3. Swimlanes (canales)

Muchas metodologías de modelado de procesos utilizan el concepto de *swimlanes* como un mecanismo para organizar actividades en categorías visuales separadas para ilustrar distintas capacidades funcionales o responsabilidades. BPMN soporta *swimlanes* con dos constructores principales. Los dos tipos de objetos *swimlane* en un BPD son:

- **Pool:** una *pool* representa a un participante en un proceso. También actúa como un contenedor gráfico para particionar un set de actividades de otras *pools*, normalmente en el contexto de situaciones B2B.


**Figura 7 Representación de una pool**

- **Lane:** una *lane* es una sub-partición dentro de una *pool* que extiende la longitud de la misma, ya sea verticalmente u horizontalmente. Las *lanes* se usan para organizar y categorizar actividades.


**Figura 8 Representación de dos lanes dentro de una pool**

Las *pools* se usan cuando un diagrama involucra dos entidades o participantes separadas y que están físicamente separadas en el diagrama. Las actividades dentro de *pools* separadas se consideran procesos autocontenidos. Por tanto, los flujos de secuencia no deben cruzar los límites de una *pool*. Los flujos de mensajes son definidos como el mecanismo para mostrar la comunicación entre dos participantes y, por tanto, debe conectar dos *pools* (u objetos dentro de las *pools*).

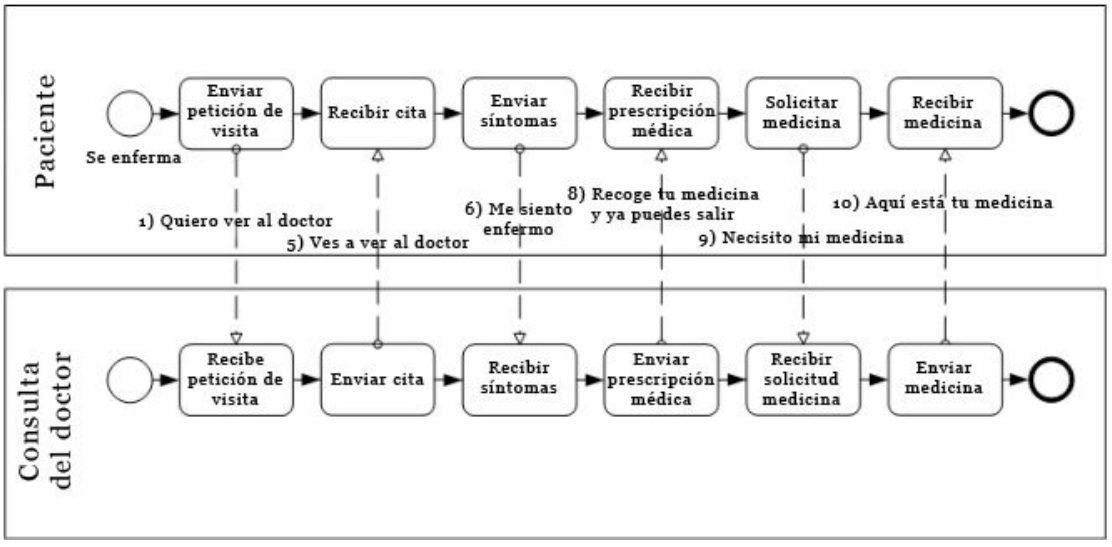



**Figura 9 Ejemplo de un diagrama con swimlanes**

Las *lanes* están más estrechamente relacionadas con las metodologías de modelado de procesos con *swimlanes* tradicionales. Las *lanes* se usan comúnmente para separar las actividades asociadas con un rol o función de una organización específica. Los flujos de secuencia pueden cruzar los límites de las *lanes* dentro de

una *pool*, pero los flujos de mensaje no pueden ser usados entre objetos de flujo dentro de las *lanes* de la misma *pool*.

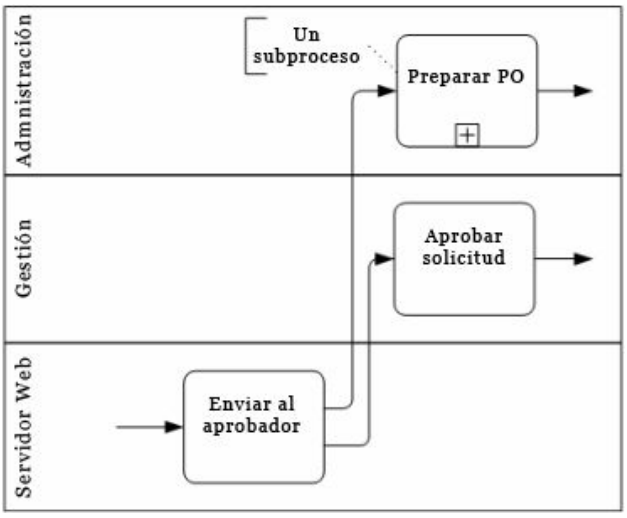



**Figura 10 Ejemplo de un diagrama con cruce entre pistas**

## 2.2.4. Artefactos

BPMN fue diseñado para dar a los modeladores y a las herramientas de modelado algo de flexibilidad para extender la notación básica y ofrecer la posibilidad de añadir contexto adecuado a situaciones de modelado específicas, como la de un mercado vertical (por ejemplo, aseguradoras y bancos). Se puede añadir tantos artefactos a un diagrama como el contexto del proceso de negocio requiera. La versión actual de la especificación BPMN sólo tiene tres tipos de artefactos BPD predefinidos:

- **Data Object:** los objetos de datos son un mecanismos que muestran cuántos datos son necesarios o producidos por las actividades. Estos están conectados a las actividades mediante asociaciones.
- **Group:** un grupo se representa con un rectángulo con esquinas redondeadas y con un borde discontinuo. La agrupación se puede usar con fines de documentación o análisis, pero sin afectar a los flujos de secuencia.
- **Annotation:** las anotaciones son un mecanismo que usan los modeladores para proporcionar información textual adicional para el lector del diagrama BPMN.

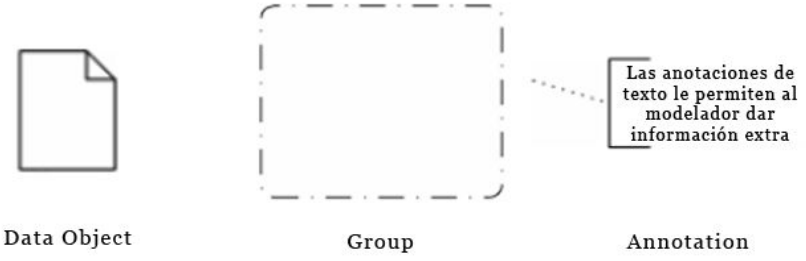



**Figura 11 Tipos predefinidos de artefactos**

Los modeladores pueden crear sus propios tipos de artefactos, los cuales pueden añadir más detalles sobre cómo se realiza el proceso (frecuentemente para mostrar los *inputs* y *outputs* de las actividades del proceso). Sin embargo, la estructura básica del proceso, como determinan las actividades, *gateways*, y flujos de secuencia, no se ve alterada al añadir artefactos al diagrama, como bien muestran las figuras 10 y 12.

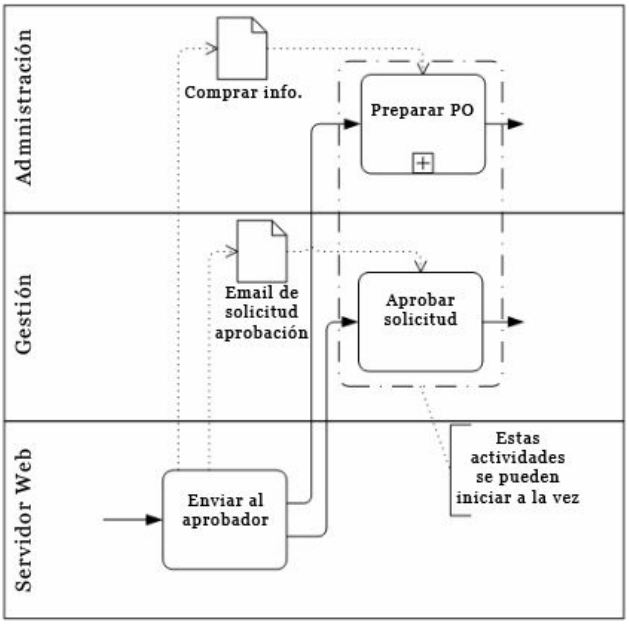



**Figura 12 Ejemplo de un diagrama con artefactos**

## 2.3. Usos generales del BPMN

El modelado de procesos de negocio se usa para comunicar una amplia variedad de información a diferentes públicos. BPMN está diseñado para abarcar muchos tipos de modelado y permite la creación de segmentos de proceso así como procesos de negocio *end-to-end*, con diferentes niveles de fidelidad. De entre los distintos objetivos del modelado de procesos, hay dos tipos básicos de modelos que pueden ser creados con un BPD:

- Procesos B2B colaborativos.
- Procesos de negocio internos.

### 2.3.1. Procesos B2B colaborativos

Un proceso B2B colaborativo muestra la interacción entre dos o más entidades del negocio. Los diagramas para estos tipos de procesos son generalmente desde un punto de vista global. Esto quiere decir que no toman el punto de vista de ninguno de los participantes, pero muestran las interacciones entre ellos. Las interacciones se muestran como una secuencia de actividades y patrones de intercambio de mensajes entre los participantes. Las actividades de los participantes de una colaboración pueden ser consideradas los "*touch-points*" entre éstos; por tanto, el proceso define las

interacciones que son visibles al público para cada participante. Cuando observamos el proceso descrito sólo en una *pool*, al proceso público también se la llama proceso abstracto. Los procesos reales (internos) son dados a tener más actividades y detalle del mostrado en procesos B2B colaborativos.

La figura 9, mostrada anteriormente, se repite en la figura 13 para servir de ejemplo de un proceso B2B colaborativo.

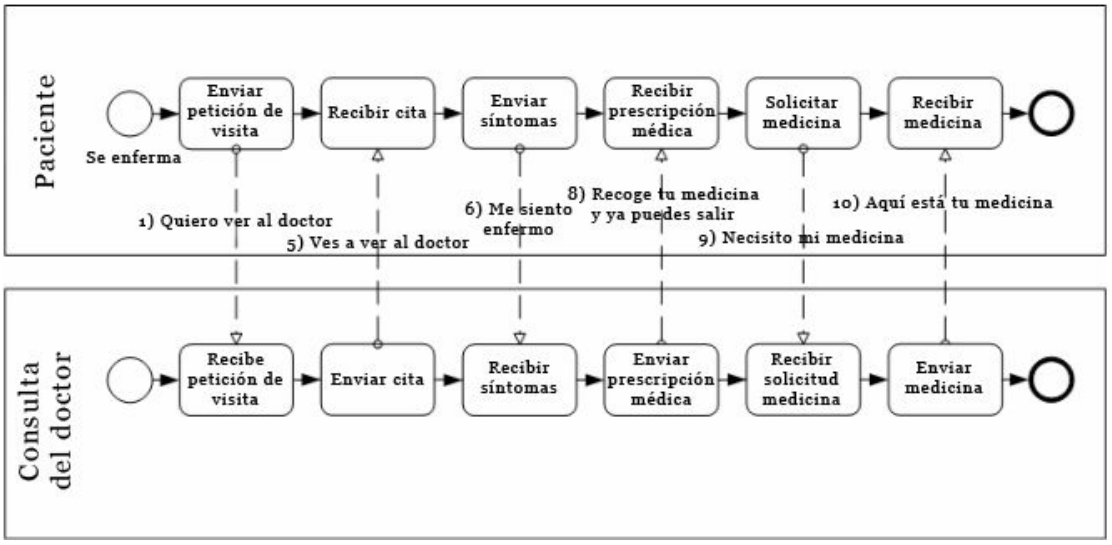



**Figura 13 Ejemplo de un proceso B2B colaborativo**

### 2.3.2. Procesos de negocio internos

Un proceso de negocio interno normalmente se centrará en el punto de vista de una sola organización de negocio. Aunque los procesos internos a menudo muestran interacciones con participantes externos, éstos definen las actividades que normalmente no son visibles al público y son, por tanto, actividades privadas. Si se utilizan *swimlanes*, entonces un proceso de negocio interno estará contenido dentro de una sola *pool*. Los flujos de secuencia del proceso está por tanto dentro de ésta y no pueden salir de sus límites. Los flujos de mensaje pueden cruzar los límites de la *pool* para mostrar interacciones existentes entre procesos de negocio internos separados.

## 2.4. Diferentes niveles de precisión

El modelado de procesos de negocio suele empezar identificando las actividades de alto nivel y a partir de éstas bajar a mayores niveles de precisión en diagramas separados. Puede que haya múltiples niveles de diagramas, dependiendo de la metodología empleada para el desarrollo del modelo. Sin embargo, BPMN es independiente de todas las metodologías de modelado de procesos específicas.

La figura 14 muestra un ejemplo de un proceso a alto nivel, realizado para un caso de estudio de BPMN, el cual es básicamente una serie de subprocesos con tres puntos de tomas de decisiones dentro del proceso.

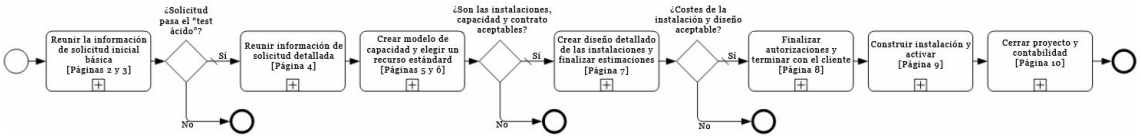



**Figura 14 Ejemplo de un proceso de negocio a alto nivel**

La figura 15 muestra los detalles del primer subproceso de la figura 14. Este diagrama emplea dos *pools*, una para el cliente y otra para la organización que proporciona el servicio. Nótese que este diagrama muestra el proceso de negocio interno de la organización y también muestra el proceso abstracto del cliente (por ejemplo, el proceso del cliente sólo incluye las actividades usadas para comunicarse a través de los flujos de mensaje hacia la organización). Las actividades dentro de la organización son particionadas por *lanes* para mostrar los departamentos o roles responsables en su ejecución.

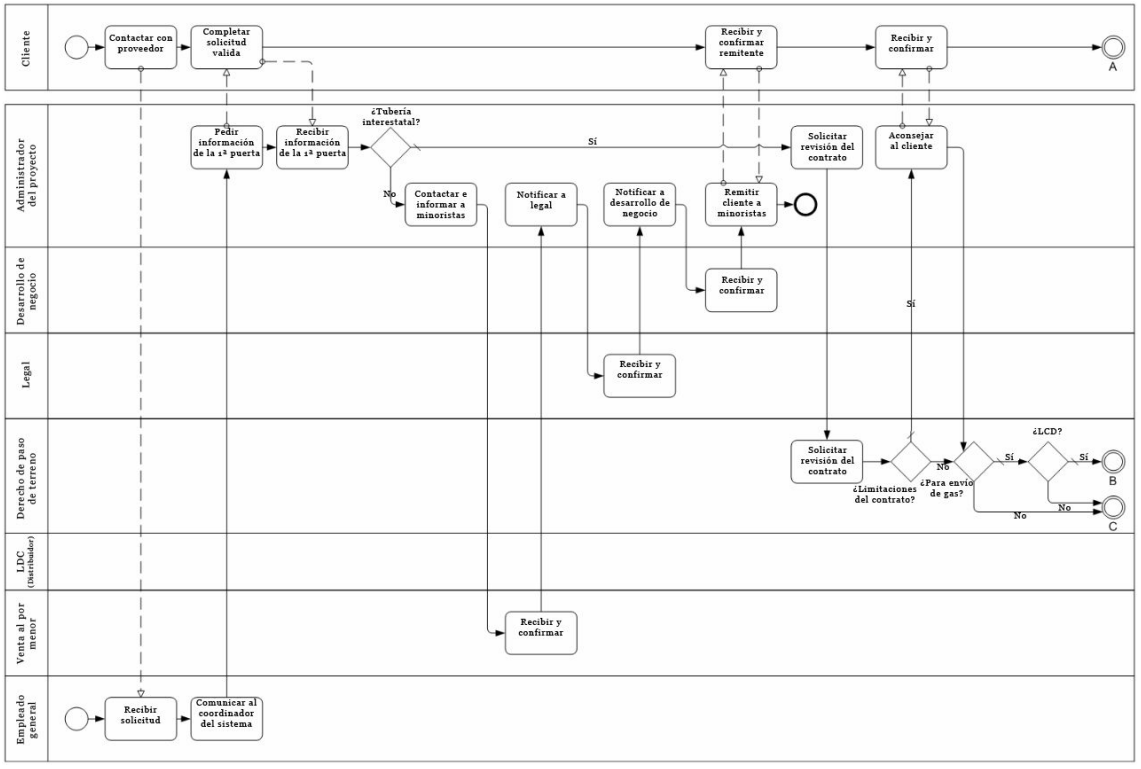



**Figura 15 Proceso de negocio a más bajo nivel del ejemplo de alto nivel**

## 2.5. Valor de modelar en BPMN

Los miembros del *BPMI Notation Working Group* representan una gran parte de la comunidad de modelado de procesos de negocio, y han llegado a un consenso para presentar BPMN como la notación de modelado de procesos de negocio estándar. El desarrollo de la BPMN es un importante paso para reducir la fragmentación que existe por culpa de los cientos de herramientas de modelado de procesos y notaciones.

El *BPMI Notation Working Group* ha demostrado tener amplios conocimientos y experiencia con muchas de las notaciones existentes y ha conseguido consolidar las mejores ideas de todas estas notaciones divergentes en una única simple notación estándar. Algunos ejemplos de otras notaciones y metodologías que se revisaron fueron: *UML Activity Diagram*, *UML EDOC Business Processes*, IDEF [1], ebXML BPSS [9], *Activity-Decision Flow (ADF) Diagram*, RosettaNet, LOVeM, y *Event-Process Chains* (EPCs). Esta fragmentación ha obstaculizado la aceptación de sistemas de administración de procesos de negocio interoperables. Una notación de modelado estándar ampliamente soportada reduciría la confusión entre negocios y usuarios finales de las TI.

Otro factor que motivó el desarrollo de BPMN es que, históricamente, los modelos de proceso de negocio desarrollados por gente de negocios han estado técnicamente separados de las representaciones de los procesos requeridas por los sistemas diseñados para implementar y ejecutar dichos procesos. Por tanto, no hay necesidad de traducir manualmente los modelos de proceso de negocio originales a modelos de ejecución. Dichas traducciones están sujetas a errores y dificultan a los propietarios de los procesos a entender la evolución y ejecución de los procesos que ellos mismos han desarrollado.

## 2.6. Mapear de BPMN a BPEL4WS

Para ayudar a reducir la brecha tecnológica en modelado, una meta clave para el desarrollo de BPMN era crear un puente entre la notación de modelado de procesos orientada a negocios y los lenguajes de ejecución orientados a las TI que implementarán los procesos dentro de un sistema de administración de procesos de negocio. Los objetos gráficos de BPMN, soportados por un amplio set de atributos de objeto, han sido mapeados a la *Business Process Execution Language for Web Services* (BPEL4WS v1.1), el estándar de facto para la ejecución de procesos. La figura 16 muestra un ejemplo de un segmento de un proceso de negocio marcando el mapeado de a los elementos de ejecución del BPEL4WS.

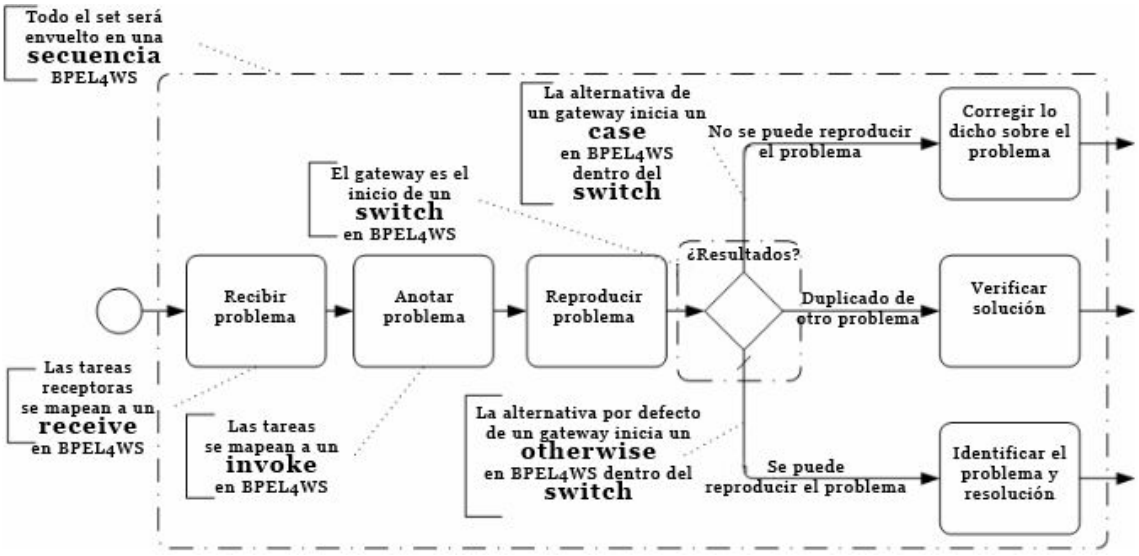



**Figura 16 Ejemplo de un diagrama mapeado a BPEL4WS**

## 2.7. Comentarios finales

Más adelante se explicará cómo en los métodos ágiles, especialmente en los tableros kanban, hay por detrás una especificación de proceso, o workflow, que idealmente suele estar formado por una secuencia de actividades muy simple pero que, en la práctica, requieren de mucha flexibilidad.

Reconocer que un tablero kanban es un workflow es entender que se puede utilizar una notación gráfica como BPMN para diseñar estos tableros, que es el propósito de este TFG.

# Capítulo 3 Metodologías ágiles

Parte del contenido de este capítulo ha sido extraído y traducido del estudio "*A Comparison between Agile and Traditional Software Development Methodologies*" de M. A. Awad [10].

El desarrollo de software empezó como un proceso caótico, principalmente basado en el conocido "*code and fix*" (programar y arreglar). El código se confeccionaba con poca o ninguna planificación, y el diseño global del sistema no estaba bien pensado y terminaba siendo determinado por decisiones a corto plazo. Esta forma de trabajo no era mala para sistemas de dimensiones reducidas, pero éstos se volvían cada vez más grandes y ambiciosos. Por tanto, arreglar y expandir dichos sistemas era una tarea cada vez más difícil debido a su alta complejidad y baja mantenibilidad. Finalmente, como respuesta se introdujeron alternativas a esta forma de desarrollo conocidas como metodologías. Las metodologías imponen una proceso disciplinado al desarrollo de software con el objetivo de crear un software de mayor calidad.

En la actualidad se consideran dos categorías principales de metodologías:

- Metodologías tradicionales.
- Metodologías ágiles.

## 3.1. Metodologías tradicionales

Las metodologías tradicionales dependen mucho de establecer una serie de requisitos estables desde el principio y que éstos no cambien a lo largo del proyecto. Dentro de las metodologías tradicionales encontramos muchas variedades, una de las más conocidas y utilizadas es el Desarrollo en Cascada.

### 3.1.1. Desarrollo en cascada

Como alternativa al enfoque "*code and fix*", Winston Royce propuso en el año 1970 la metodología de desarrollo en cascada [11]. Esta metodología se centra en avanzar de forma secuencial por unas fases determinadas de desarrollo. Dentro de estas fases nos encontramos una serie de tareas y objetivos que han de cumplirse antes de avanzar a la siguiente. Las fases pueden tener nombres diferentes cada vez, pero las primera fase siempre se concentra en determinar qué es lo que se quiere que el sistema haga, y la segunda en en cómo se diseñará. La tercera fase es en la que se empieza a escribir el código para crear el sistema software deseado, en la cuarta se lleva a cabo el testeo del sistema y en la última fase realizan tareas de implementación. La figura 17 muestra un ejemplo de un ciclo de vida del desarrollo en cascada, donde los porcentajes marcan el tiempo ha invertir en cada fase.

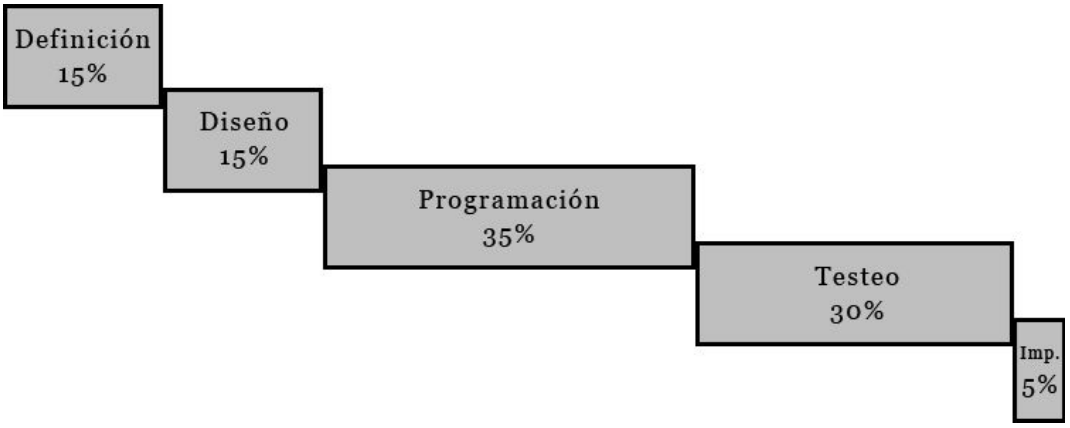



**Figura 17 Ejemplo de un ciclo de vida del desarrollo en cascada**

El desarrollo en cascada presenta una serie de claras debilidades:

- La definición y el diseño son totalmente anticipados y no consideran futuros cambios en los requisitos.
- Detección tardía de fallos en la especificación, implementación, etc.
- No contempla la posibilidad de re-trabajo.

## 3.2. Metodologías ágiles

Las metodologías ágiles surgen como respuesta a las tradicionales, pues uno de sus principales problemas es la falta de adaptabilidad a cambios en los requisitos. Para solucionar estos problemas, las metodologías ágiles se basan en ciclos de desarrollo cortos, iterativos e incrementales, de forma que al final de cada iteración se tiene una versión del producto utilizable que se va acercando cada vez más a la visión final del mismo. Además, estas metodologías intentan involucrar al cliente en el proyecto lo máximo posible y, al siempre haber una versión usable del producto, validar los requisitos con él y acordar posibles cambios y mejoras. Algunas de las metodologías ágiles más conocidas son:

- Extreme Programming
- Scrum
- Kanban
- Lean Development

### 3.2.1. Extreme Programming

El proceso de Extreme Programming (XP) [13] se caracteriza por sus cortos ciclos de desarrollo, continuo feedback, software incremental, y diseño evolutivo. Gracias a estas características se la considera una metodología capaz de responder en entornos de continuos cambios. Las prácticas y terminología de XP son las siguientes:

- **Planificación:** el programador estima el esfuerzo que será necesario para la implementación del software.
- **Pequeñas entregas:** a medida que el software va creciendo se hacen entregas frecuentes del mismo al cliente.
- **Metáfora:** el software se define a partir de una serie de metáforas que describen cómo funciona éste entre el cliente y los programadores.
- **Diseño simple:** los diseños deben ser lo más simples posibles.
- **Refactoring:** es la reestructuración del software para eliminar redundancias, simplificar su estructura, etc. pero sin alterar la funcionalidad.
- **Programación en pareja:** el código se escribe en pareja en un solo ordenador.
- **Propiedad colectiva:** nadie en concreto es el responsable de ciertas partes del código, todo el mundo puede modificar como quiera el mismo.
- **Integración continua:** el código se integra con la versión del software actual tan pronto como esté listo.
- **Semana de 40 horas:** Una semana debe tener como mucho 40 horas de trabajo.
- **Cliente interno:** el cliente debe estar disponible para hablar con el equipo de desarrollo en todo momento.
- **Estándares de programación:** deben haber unas reglas de estilo consensuadas a la hora de programar.

El ciclo de vida en XP (figura 18) se divide en seis partes: exploración, planificación, iteraciones hasta la entrega, producción, mantenimiento y muerte.

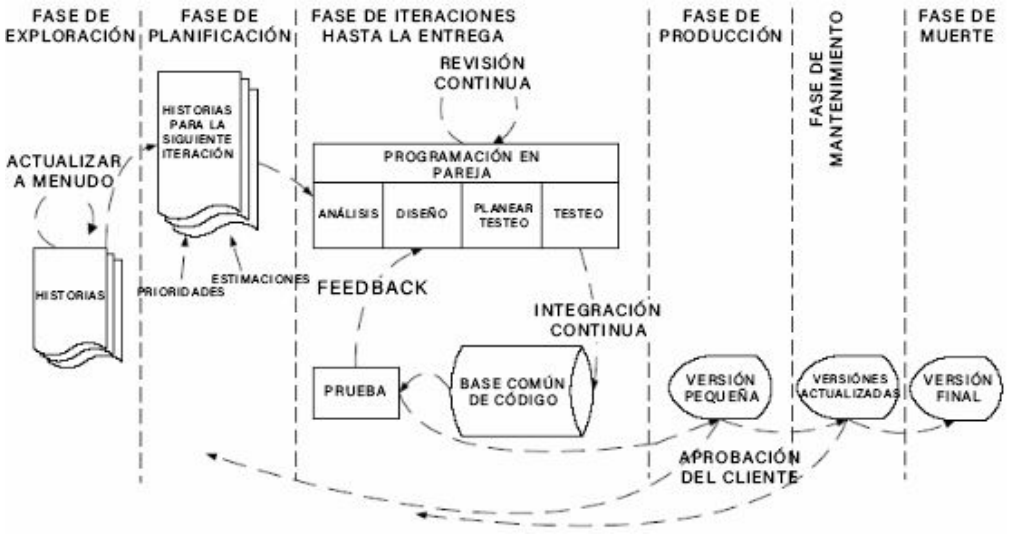



**Figura 18 Ciclo de vida de los procesos XP [3]**

## 3.2.2. Scrum

Scrum [12] es un proceso iterativo e incremental para el desarrollo de productos. Una de sus características principales es su enfoque en cómo un equipo debe trabajar para crear un sistema flexible dentro de un entorno de constantes cambios. Al final de una iteración la funcionalidad del producto debe de haber aumentado, acercandolo a la

visión final del mismo. Esto se puede apreciar mejor si observamos el ciclo de vida de Scrum de la figura 19.

Las claves de la metodología Scrum son las siguientes:

- **Sprints:** son periodos de 30 días en el cual se adapta el producto a los cambios solicitados y se aumenta su funcionabilidad, de forma que al final de los 30 días se ha conseguido una versión incrementada de la anterior lista para mostrar al cliente.
- **Backlog del producto:** es la lista de las funciones, mejoras y cambios deseados que todavía faltan por introducir.
- **Reunión para la planificación de sprints:** es dónde los clientes, usuarios, el *product owner* y el *Scrum team* acuerdan una serie de objetivos. Para luego acordar cómo se realizará la implementación en el sprint.
- **Backlog del sprint:** es la lista de funcionalidades asociadas a dicho sprint.
- **Scrum diaria:** es una reunión diaria de unos 15 minutos y que sirven para mantener a todo el equipo informado del estado del proyecto, identificar posibles problemas, etc.


**Figura 19 Proceso Scrum [4]**

### 3.2.3. Kanban

Kanban [25] es una metodología simple y muy visual que permite, de un vistazo, comprobar el estado actual del trabajo. En kanban el trabajo se divide en tareas que han de seguir un proceso (el workflow) hasta su finalización. Sus principales características son cuatro:

- **Visualización del workflow:** como se puede ver en la figura 20 visualizar el proceso que siguen las actividades y su estado en un tablero kanban es muy sencillo.
- **Límite a las tareas en WIP:** es un sistema que limita la cantidad de trabajo que se hace en ciertas etapas del workflow, de esta forma se asegura que el equipo de desarrollo no excede su capacidad de trabajo y que las tareas pendientes no se acomulan.
- **Control del flujo:** el proceso es modificable y, por tanto, si algo falla se puede mejorar cambiando partes del mismo. El flujo es completamente personalizable y adaptable a diferentes contextos.
- **Mejora del proceso:** el uso de kanban es un ciclo evolutivo, a medida que se va utilizando se van descubriendo posibles mejoras y cambios que harán que el proceso sea más eficiente y productivo.

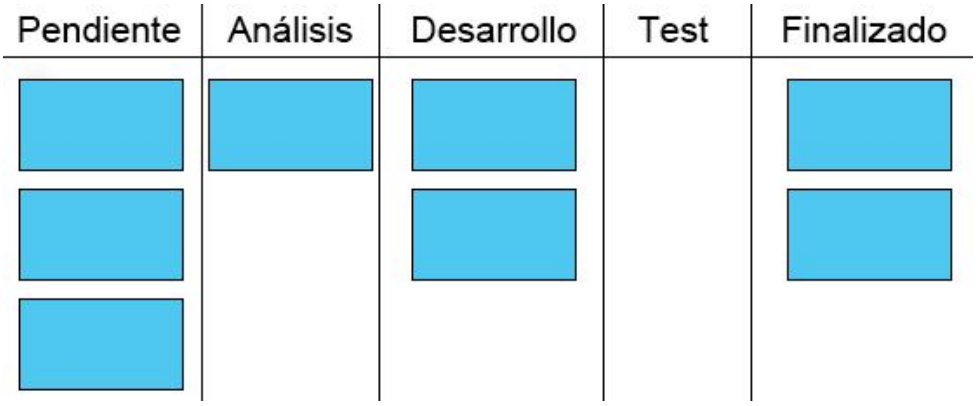



**Figura 20 Tablero kanban donde las tareas son los rectángulos azules**

### 3.2.4. Lean Development

Lean Development [26] se basa en una serie de principios para reducir costes, esfuerzo y optimizar recursos con el objetivo de construir justo lo que el cliente quiere, también llamado menor producto viable. Estos principios aplicados al desarrollo de software son:

- **Eliminar los desperdicios:** todo aquello que no tenga valor para el cliente debe ser eliminado como, por ejemplo, código innecesario.
- **Amplificar el aprendizaje:** es importante obtener feedback durante el desarrollo para así refinar el proceso de desarrollo y, por tanto, mejorar la eficiencia y demás aspectos importantes de éste.
- **Decidir lo más tarde posible:** el desarrollo de software se realiza en un contexto de continuos cambios, por tanto al retrasar lo máximo posible las decisiones se puede reaccionar mejor a éstos.
- **Entregar tan rápido como sea posible:** es interesante realizar versiones estables para poder probarlas, recibir feedback y pasar a la siguiente iteración.

- **Capacitar al equipo:** el equipo de desarrollo tiene voz y es el que dice a los directivos qué posibilidades hay para el desarrollo o qué mejoras serían interesantes.
- **Construir integridad intrínseca:** cada uno de los componentes del sistema funcionan adecuadamente juntos.
- **Véase todo el conjunto:** el sistema software ha de verse como la suma de sus partes y como el producto de sus interacciones. Es importante no sólo realizar pruebas a las pequeñas partes sino al conjunto como un todo.

# Capítulo 4 Workflow para desarrollo ágil

Parte del contenido de este capítulo se ha obtenido de la ayuda de Worki [27] disponible desde la propia herramienta.

Se entiende como workflow a la automatización parcial o total de un proceso de negocio, es decir, una colección de actividades o tareas relacionadas y estructuradas que en una secuencia específica produce un servicio o producto. Estas actividades pueden implican una interacción automática o manual con el usuario.

## 4.1. Introducción a TUNE-UP Process

Como se ha mencionado anteriormente, TUNE-UP Process es un framework que permite a un equipo de desarrollo establecer una metodología ágil adaptada a las necesidades de su contexto. Sus principales características son:

- Un modelo de proceso de desarrollo de software iterativo e incremental.
- Proceso de desarrollo dirigido por pruebas.
- Workflows flexibles que permiten la coordinación del esfuerzo asociado a cada una de las unidades de trabajo.
- Planificación y seguimiento del proceso de desarrollo continuo.
- Cuantificación del esfuerzo (tiempo) realizado.

Durante el resto de este capítulo nos centraremos en ver cómo se incorpora la gestión de los workflows dentro de Worki.

## 4.2. Worki

Worki está formado por 3 módulos principales:

- Tablero kanban.
- Gestor de unidades de trabajo.
- Gestor de la estructura.

### 4.2.1. Tablero kanban

El tablero kanban es la herramienta que resume todo el trabajo, muestra todas las Unidades de Trabajo (UT) no terminadas, en las que el usuario está participando, ha participado o participará. Es lo primero que vemos al abrir la herramienta (figura 21), de forma que, nada más entrar, tenemos una lista del trabajo a realizar, promoviendo la realización de trabajo no finalizado así como la colaboración durante el ciclo de desarrollo de una UT.

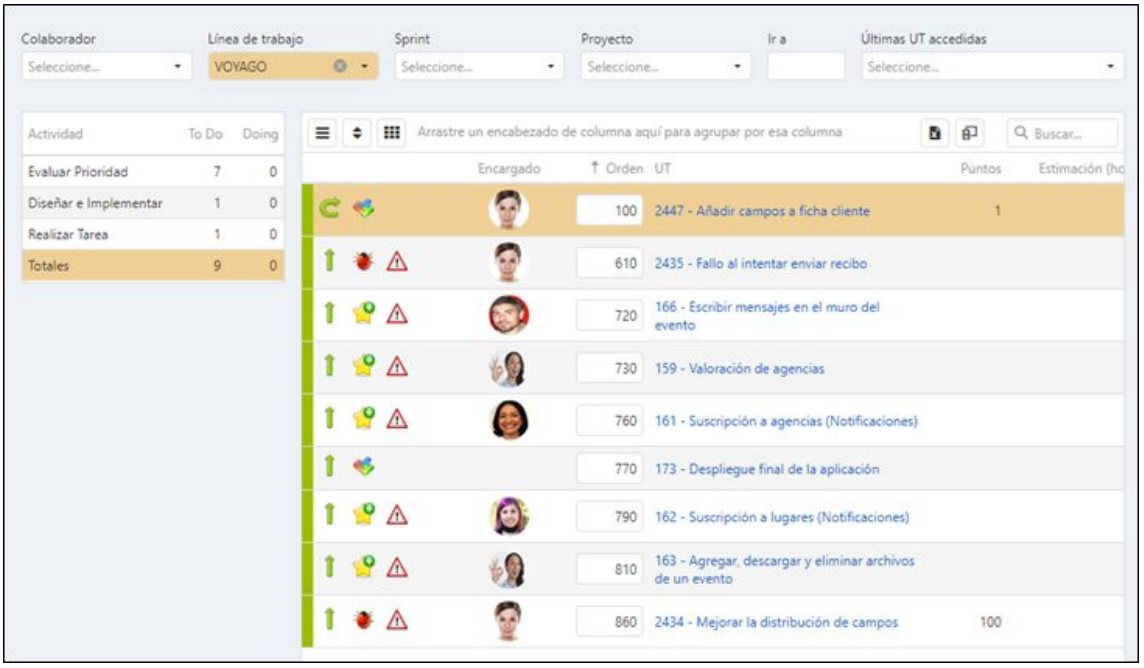



**Figura 21 Imagen de la interfaz del Tablero kanban**

## 4.2.2. Gestor de unidades de trabajo

El gestor de UTs, como bien indica su nombre, es el agente que nos permite gestionar todo aquello relacionado con las UTs. Éste, además, posee distintas herramientas que nos permiten cuantificar el esfuerzo realizado en cada UT, su estado y demás cosas a tener en cuenta:

- **Gestión de seguimiento:** esta herramienta nos ofrece información sobre el estado de la UT: actividades por las que ha pasado, quienes han estado involucradas en su realización y su estado actual. Además, también nos permite editar el estado de la UT. La figura 22 muestra la interfaz del gestor.

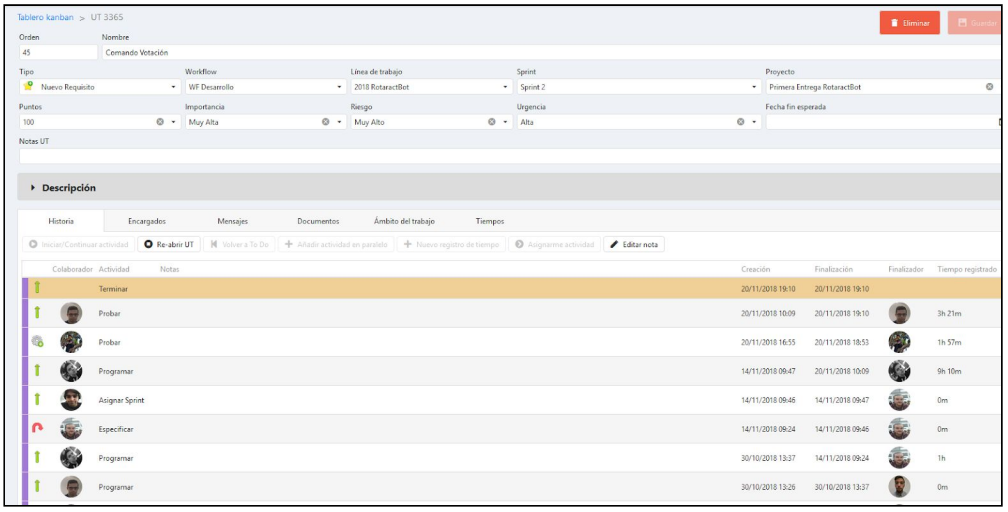



**Figura 22 Interfaz del gestor de seguimiento**

- **Gestión de tiempos:** herramienta que permite consultar el tiempo empleado por los diferentes agentes en la realización de las actividades dentro de la UT como se muestra en la figura 23.

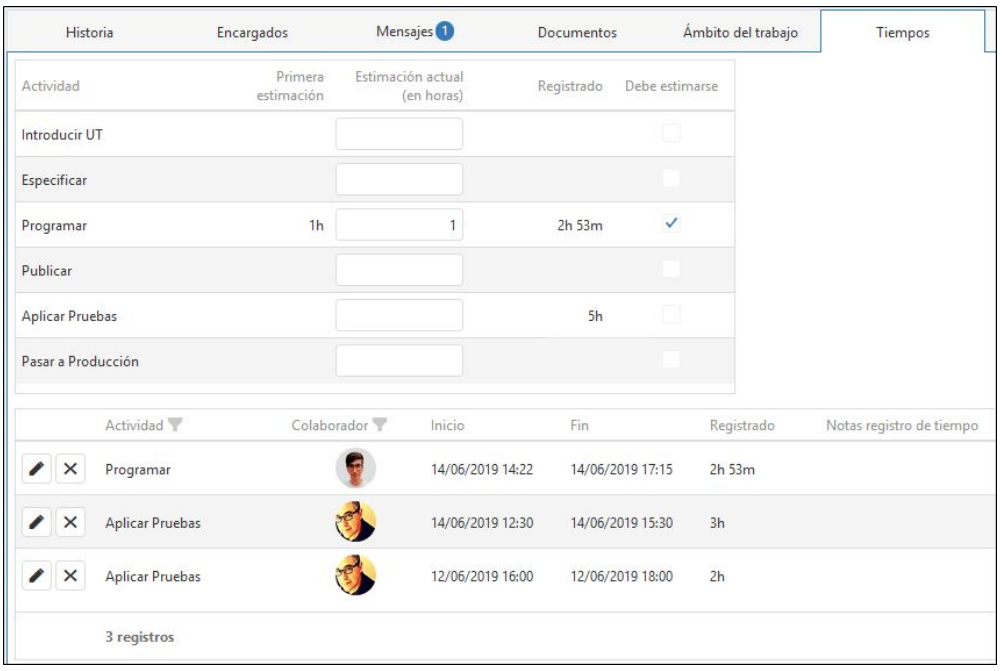



**Figura 23 Interfaz del gestor de tiempos**

- **Gestión de documentos:** una UT puede tener documentos asociados, de tenerlos se mostrarán en la pestaña “Documentos” como bien muestra la figura 24.

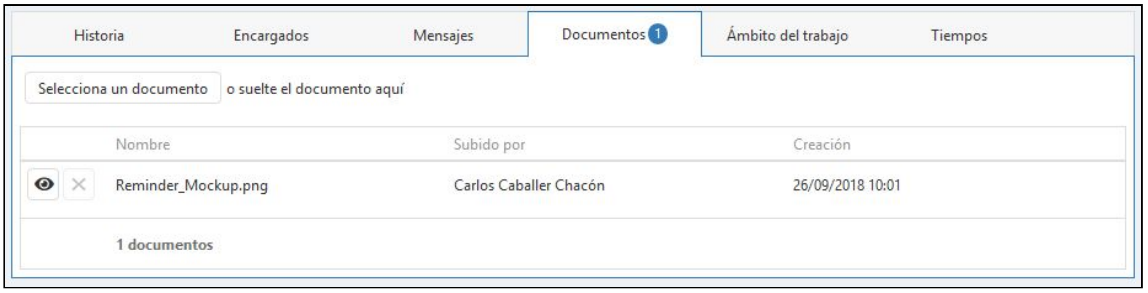



**Figura 24 Interfaz del gestor de documentos**

## 4.2.3. Gestor de la estructura

En Worki, las líneas de trabajo tienen una estructura en forma de árbol. Normalmente, en el caso de productos software, la estructura representa la organización funcional del producto. Siendo la idea que, al consultar el trabajo realizado en una línea de trabajo, su estructura nos ayude en la planificación y toma de decisiones.

Worki incorpora un gestor de estructuras que nos permite modificar y crearlas a nuestro antojo, tal y como muestra la figura 25.

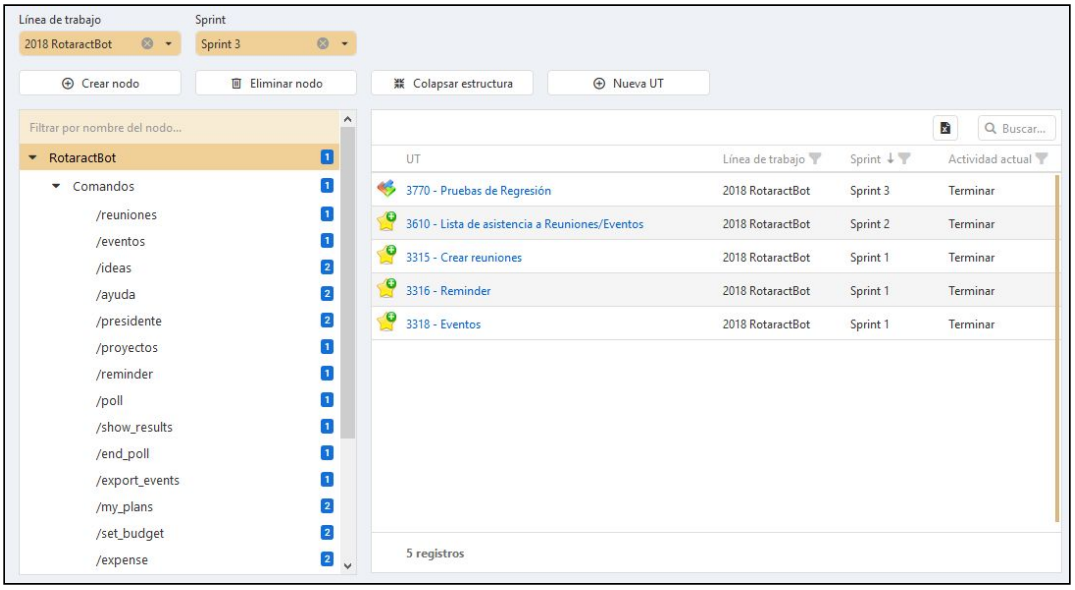



**Figura 25 Interfaz del gestor de la estructura**

Con esto concluye la presentación de los tres módulos principales de Worki. Más adelante se hablará en profundidad sobre el gestor de workflows de Worki.

# Capítulo 5 Tecnología para el desarrollo de editores gráficos

## 5.1. Angular para desarrollo web

La versión web de Worki sobre la que vamos a trabajar está hecha en Angular [5] (v1.6.6), es por eso que, para facilitar la integración, nuestro editor gráfico también ha de ser construido utilizando Angular.

Angular es un framework de desarrollo, gratuito y open source, creado por Google [14] para JavaScript. Fue creado con la finalidad de facilitar el desarrollo de aplicaciones web SPA (Single-page Application) que sean eficientes y simples a nivel de código.

En Angular las aplicaciones web están compuestas por componentes, estos son partes de código que, como piezas de un puzle, se pueden reutilizar en diferentes partes de la aplicación, permitiendo un desarrollo mucho más ágil.

En este apartado trataremos el estudio previo realizado para determinar qué componente utilizar para facilitar la tarea de crear un editor gráfico de workflows que emplee diagramas. El tipo de componente que buscamos es un componente Angular para su fácil implementación dentro de Worki. Los requisitos que debe cumplir el componente son:

- Permitir la edición de diagramas.
- Permitir la personalización de los elementos del diagrama.
- Control intuitivo y simple para el usuario.
- Facilidad de aprendizaje.

## 5.2. Componentes para editar diagramas

A la hora de buscar componentes nos encontramos dos alternativas: software de código abierto y comercial. El software de código abierto es gratuito y permite a sus usuarios realizar modificaciones al mismo dando un alto grado de personalización que el código cerrado o comercial no pueden ofrecer pero, por otra parte, la documentación suele ser escasa, no tiene versiones muy estables o están aún en desarrollo. Por otra parte, el código comercial suele ser más estable y robusto, además de contar con una documentación amplia, ejemplos y demás información que facilita el aprendizaje, pero, como es de esperar, requiere de un desembolso económico y modificar el código puede ser complicado.

### 5.2.1. NGX-Graph[2]

NGX-Graph [15] es un componente creado y utilizado por Swimlane [6] quienes, como parte de su compromiso por la comunidad *open source*, liberaron el código de NGX-Graph el 22 de Febrero del 2017.

Una de las ventajas de NGX-Graph es su sencillo e intuitivo control de la interfaz, navegar por el diagrama, acercarse o alejarse de éste se puede hacer de forma simple sin tener que configurar ningún parámetro.

Por el lado negativo, tenemos que la personalización de los nodos puede llegar a ser tediosa, teniendo que hacer uso de los templates de Angular para personalizar la forma y apariencia de los nodos, idealmente queremos que toda la personalización sea autocontenida dentro del componente. Además, la documentación [16], aunque existente, es escasa, confusa y en ciertas partes desactualizada. Esto dificulta mucho el aprendizaje de este componente. Finalmente cabe mencionar que no fue diseñado con el modelado de procesos de negocio en mente.

A continuación, en la figura 26, se muestra un diagrama sencillo creado con NGX-Graph.

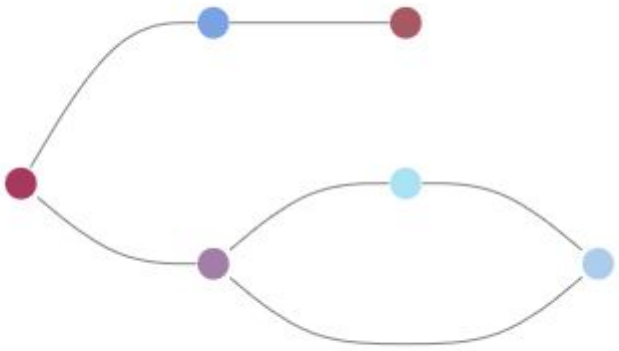


**Figura 26 Ejemplo de un diagrama en NGX-Graph**

Para crear un nodo en NGX-Graph podemos hacer uso del siguiente código:

```
const newNode: Node = { id: '1' };
this.nodes.push(newNode);
this.nodes = [...this.nodes];
```

Siendo “this.nodes” una lista de nodos ligada a la del diagrama, sólo tenemos que introducir el nodo recién creado dentro de la lista “nodes” y crear una nueva instancia

[2] https://github.com/swimlane/ngx-graph

de la misma (tercera línea) para así activar el detector de cambios que se encargará de actualizar la apariencia del diagrama para reflejar su nuevo estado.

Por otro lado, la creación de conexiones entre nodos se hace de forma similar con el siguiente código:

```
const newEdge: Edge = { id: '1', source: 'node1', target: 'node2' };
this.links.push(newEdge);
this.links = [...this.links];
```

Siendo “this.links” una lista de conexiones ligada a la del diagrama, siguiendo la misma idea que en para los nodos, creamos una nueva instancia de “links” para actualizar el diagrama.

Como conclusión, hemos podido observar la gestión de los nodos y sus conexiones se puede realizar de una forma sencilla y rápida, pero su excesiva simpleza, escasa documentación y falta de opciones a la hora de personalizar el diagrama hacen que no sea la opción más alta en la lista.

### 5.2.2. yFiles for HTML[3]

yFiles for HTML es un software comercial creado por yWorks [7] que dispone de una amplia gama de componentes. Para hacer uso de éstos se ha de usar JavaScript, pero su sitio web dispone de una guía [17] para la integración de sus componentes en Angular.

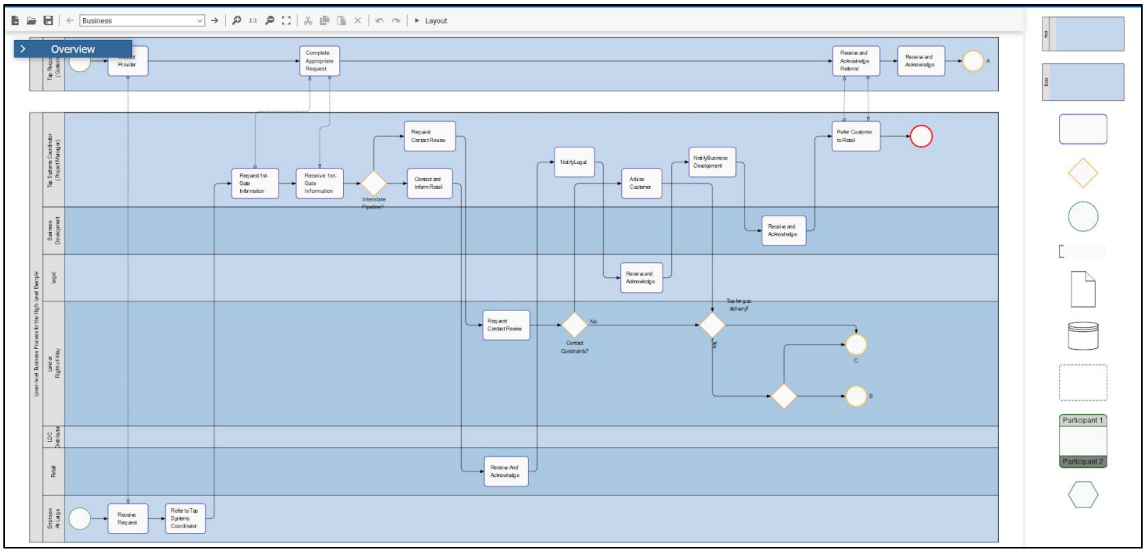



**Figura 27 Ejemplo de un proceso de negocio en yFiles for HTML**

Como se puede apreciar en la figura 27, yFiles for HTML ofrece gran variedad de herramientas que permiten al equipo de desarrollo crear cualquier tipo de diagrama que deseen. Además, la documentación [18] (figura 28) es muy completa, con

[3] https://www.yworks.com/products/yfiles-for-html

múltiples ejemplos y demos. Pero, así como es el más completo y prometedor, también es el más caro (más de 10.000€), es por eso que no se ha podido profundizar más en él y se sitúa al final de la lista de opciones.

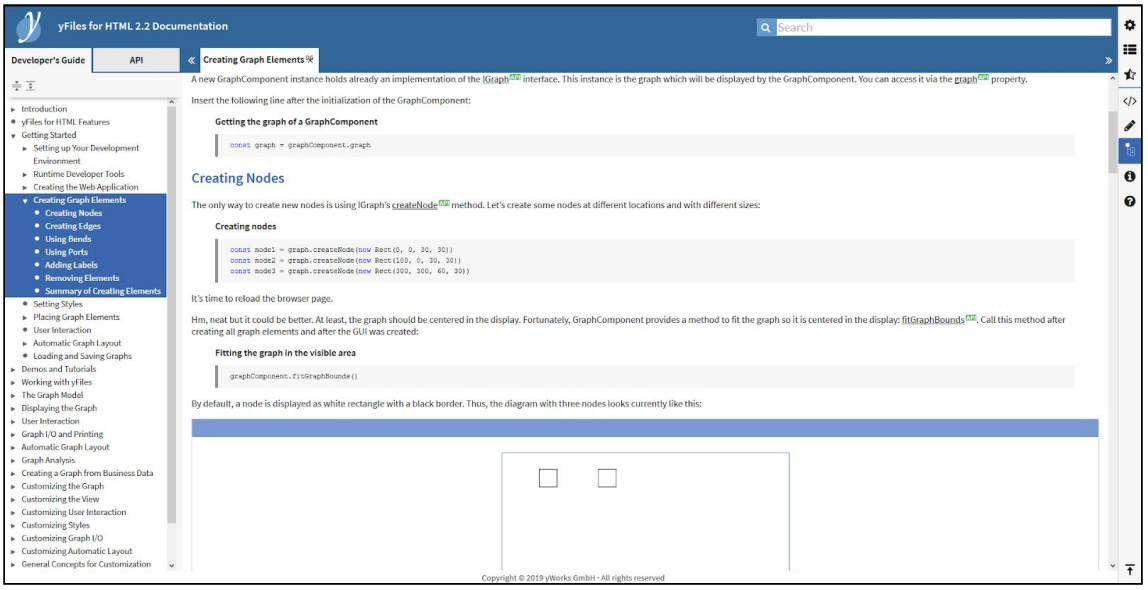



**Figura 28 Documentación de yFiles for HTML**

## 5.2.3. Syncfusion Essential JS 2 Diagram[4]

Software comercial que forma parte de Essential Studio, una serie de componentes gráficos desarrollado por Syncfusion [8]. Es un componente bastante completo, ofrece muchas posibilidades para personalizar la apariencia y funcionalidad de nuestro diagrama, haciendo que lo podamos modelar a nuestras necesidades. El precio de todo el pack de Essential Studio es de 445€, un precio razonable dado lo que ofrece.

La documentación (figura 29) es muy completa, con algún que otro ejemplo. La documentación se divide en dos partes: API [19], donde podemos encontrar los aspectos más técnicos de los componentes, y otra que ayuda al usuario [20] en la instalación de los componentes así como a familiarizarse con los mismos, dando ejemplos e instrucciones.

[4] https://www.syncfusion.com/

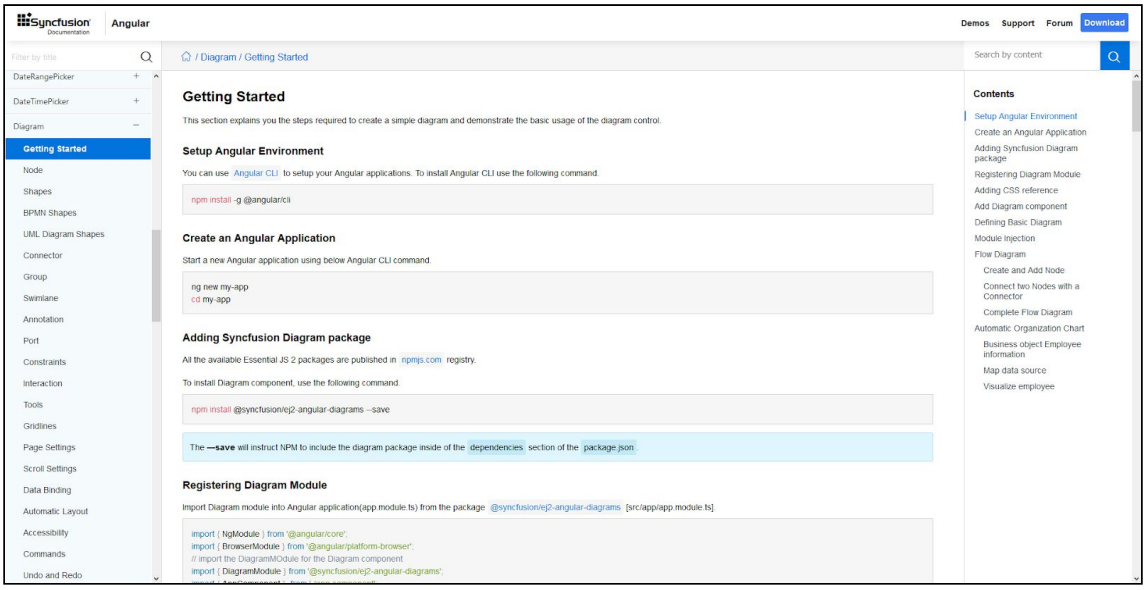



**Figura 29 Documentación de Syncfusion Essential JS 2**

Haciendo buen uso de sus características podemos conseguir un alto grado de personalización. Como podemos ver en la figura 30, podemos personalizar el diagrama y sus elementos de distintas formas, pudiendo crear desde figuras simples hasta las más complejas propias de BPMN así como modificar su apariencia a voluntad.

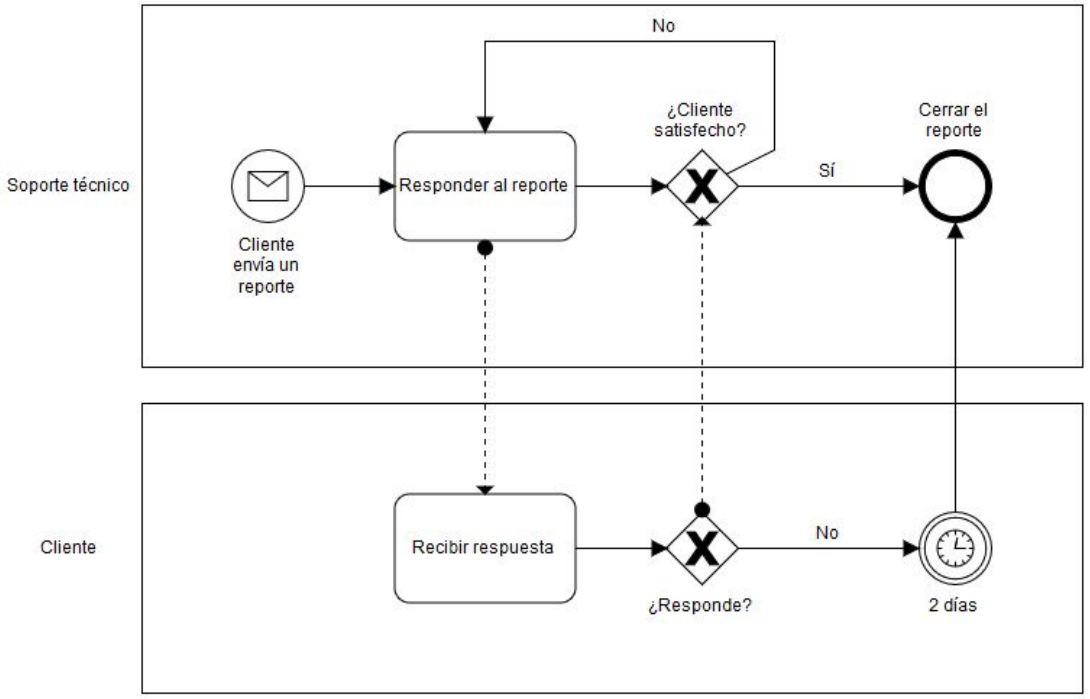



**Figura 30 Ejemplo de un diagrama creado usando Syncfusion Essential 2**

Para insertar un nodo evento de inicio con disparador de mensaje de BPMN como el del diagrama anterior utilizamos el siguiente código:

```
const nodeStart: NodeModel = {
  height: 50,
  width: 50,
  offsetX: 250,
  offsetY: 250,
  shape: {
    type: 'Bpmn',
    shape: 'Event',
    event: {
      event: 'Start',
      trigger: 'Message'
    }
  },
  annotations: [ { content: 'Cliente envía un reporte', margin: { top: 55 }
} ]
};
this.diagram.add(nodeStart);
```

De forma similar las conexiones se pueden crear de la siguiente forma:

```
const connector: ConnectorModel = {
  sourceID: 'node1',
  targetID: 'node2'
};
this.diagram.add(connector);
```

Como podemos ver la gestión de nodos y conexiones dentro del diagrama es sencilla y fácil de implementar. Aunque la gran cantidad de atributos que los nodos poseen puede llevar a declaraciones bastante largas, haciendo el código más extenso y difícil de leer de lo necesario.

El control por defecto no es el mejor, pero esto se puede arreglar haciendo algunos cambios en la configuración del diagrama. Se puede, por ejemplo, añadir una *overview* del diagrama, permitir hacer zoom, etc.

En resumen, es un buen componente que cumple todos nuestros requisitos. Parece robusto, versátil y relativamente fácil de aprender. Por tanto, lo consideramos como un fuerte candidato a ser el elegido.

## 5.3. Conclusión

Después de examinar y probar los componentes anteriores se llegó a una serie de conclusiones:

- El componente debe ofrecernos la posibilidad de personalizar de forma sencilla los elementos del diagrama así como su comportamiento.

- Debe de ser fácil de aprender, con una documentación completa para que el desarrollo sea fluido y sin más dificultades de las necesarias.

Es por eso que podemos reducir nuestras opciones a sólo dos: Syncfusion Essential JS 2 Diagram e yFiles for HTML. Las dos tienen una amplia documentación, ofrecen una gran personalización tanto en funcionalidad como en apariencia del diagrama y sus componentes. yFiles for HTML parece ser la mejor en esos campos pero, su alto precio y no tener una versión para Angular hace que descartemos dicha opción.

Por último, sólo nos queda una opción restante: Syncfusion Essential JS 2 Diagram. Cumple con todos los requisitos además de tener un precio razonable.

# Capítulo 6 Editor gráfico de workflows

## 6.1. Especificación de requisitos

El propósito de este proyecto es la creación de un editor gráfico de workflows haciendo uso de diagramas para su posterior integración dentro de la aplicación web de Worki. En este primer apartado se busca especificar y dar forma a los requisitos del nuevo editor.

### 6.1.1. Editor de workflows actual

El motor de workflows de Worki se encarga de que las UTs se realicen en el orden adecuado y, además, permite que éstas realicen saltos a cualquier actividad del workflow, admitiendo el paralelismo, la bifurcación y la secuenciación de las actividades.

El usuario puede definir los workflows haciendo uso de una tabla (figura 31). Los workflows que se pueden definir son muy flexibles y ajustados a las necesidades del proyecto del usuario. Pero usar una tabla para definirlos no es muy visual y puede llegar a confundir al usuario, es por eso que en este proyecto se plantea la creación e implementación de un editor gráfico que emplee diagramas.

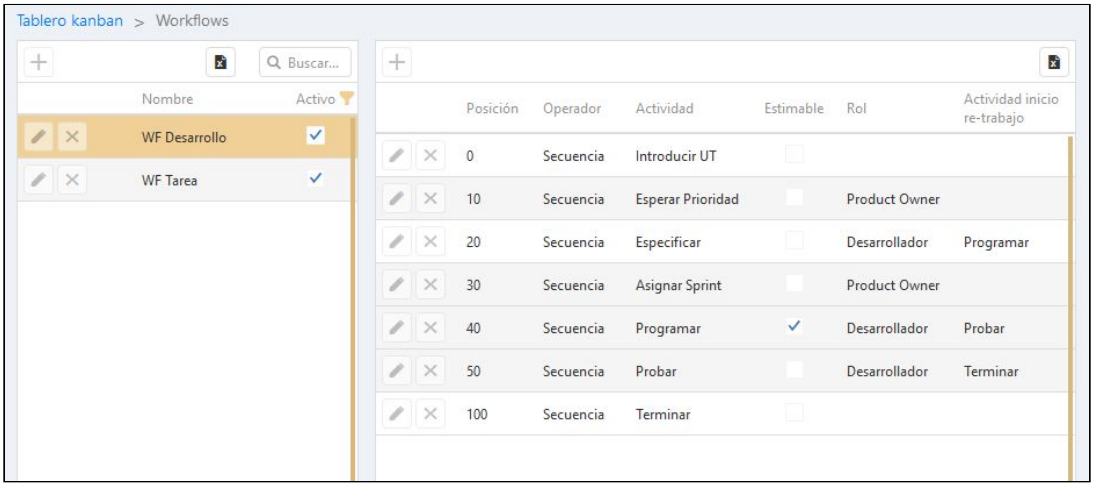



**Figura 31 Interfaz del gestor de workflows**

Al crear un workflow se le puede asociar una línea de trabajo, una vez creado vendrá con una actividad inicio y otra final. A partir de este punto el usuario puede editar libremente el workflow añadiendoles actividades, cambiando sus propiedades, o eliminandolas si éstas no se encuentran en una UT.

El workflow de una UT representa el camino que ésta sigue en su procesamiento. Sin embargo, se permite que una UT se salte actividades o se devuelva a una

actividad anterior. Todas las actividades por las que pasa una UT quedan registradas en su historial.

El sistema de edición de workflows actual sigue un modelo tabular, en el que las actividades del workflow se muestran en una tabla, pudiendo ver todos sus parámetros (posición, operador, actividad, estimable, rol y actividad re-trabajo) de un sólo vistazo.

Esta tabla permite tanto consultar el estado del diagrama como editarlo. La edición del diagrama se realiza mediante un formulario modal con una serie de restricciones para asegurar la consistencia en la base de datos y diferenciar entre el modo de editar y el de añadir una actividad. Al realizar un cambio, añadir o eliminar una actividad éste se refleja inmediatamente en la base de datos, por lo que el estado del workflow ha de ser siempre consistente.

### 6.1.2. Casos de uso

El componente tiene como finalidad permitir la edición de los workflows mediante el uso de un diagrama y sus elementos. El editor nuevo debe tener las mismas funcionalidades que el actual. Aunque, por limitaciones en la forma de visualización de un diagrama, la información visible sobre las actividades se verá reducida ligeramente.

Siguiendo lo dicho anteriormente, podemos concluir que los requisitos funcionales del sistema nuevo y actual son muy parecidos, obteniendo el diagrama de casos de uso de la figura 32.

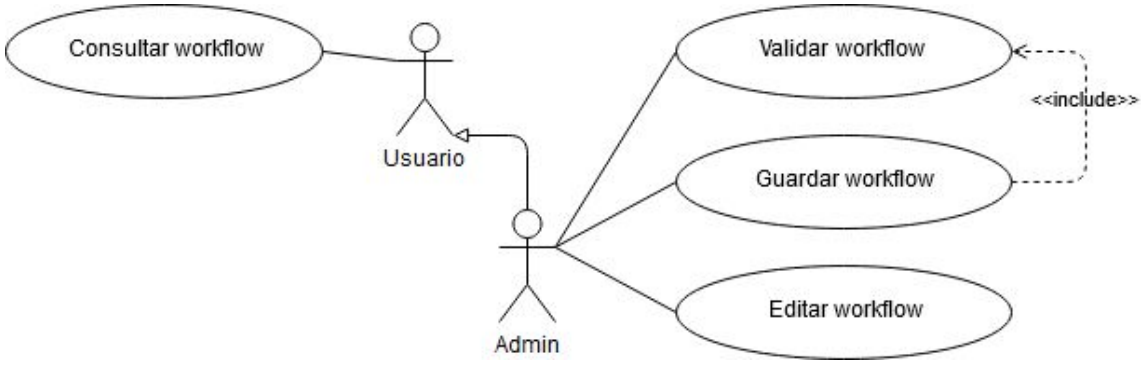



**Figura 32 Diagrama de casos de uso**

**Caso de uso 01 - Consultar workflow**

Desde la herramienta Worki el usuario puede acceder el editor desde “Elementos del sitio > Workflows”, una vez allí se mostrará un listado de todos los workflows disponibles, el usuario elegirá uno y, al cambiar a la pestaña “Diagrama”, se le mostrará el diagrama que representa el workflow seleccionado.

Pruebas de aceptación que ha de cumplir:

- Comprobar que no se puede modificar el diagrama si no se tiene permisos de administrador.

- Comprobar que tras cargar, el diagrama representa correctamente el workflow cargado.

**Caso de uso 02 - Validar workflow**

Desde dentro del editor de workflows el usuario puede clicar en un botón de nombre “Validar”, una vez hecho esto el sistema comprobará si el estado actual del diagrama es válido, si se han encontrado problemas se le mostrará al usuario una lista con los problemas encontrados y, de ser válido se le comunicará al usuario.

Pruebas de aceptación que ha de cumplir:

- Comprobar que el diagrama no es válido si hay nodos inconexos en el diagrama.
- Comprobar que el diagrama no es válido si el nodo inicial tiene entradas o más de una salida.
- Comprobar que el diagrama no es válido si el nodo final tiene salidas o más de una entrada.
- Comprobar que el diagrama no es válido si alguno de los nodos actividad no tienen exactamente una entrada y salida.
- Comprobar que el diagrama no es válido si el valor del atributo posición de alguna de las actividades en secuencia es mayor o igual que el de la siguiente actividad.
- Comprobar que el diagrama no es válido si el valor del atributo posición de alguna las actividades que sean paralelas o de selección entre sí es diferente.
- Comprobar que el diagrama no es válido si un nodo de apertura de paralelo o selección no tiene ninguna entrada o tiene menos de dos salidas.
- Comprobar que el diagrama no es válido si un nodo mezclador no tiene ninguna salida o tiene menos de dos entradas.
- Comprobar que el diagrama no es válido si un grupo de nodos que sean paralelos o de selección entre sí provienen de distintos nodos de apertura de paralelo o selección o si conducen a distintos nodos mezcladores.

**Caso de uso 03 - Guardar workflow**

Desde dentro del editor de workflows el usuario puede clicar en un botón de guardado, una vez hecho esto el sistema validará la correctitud del diagrama y, de no haber ningún problema, se guardarán los cambios realizados en la base de datos.

Pruebas de aceptación que ha de cumplir:

- Comprobar que los cambios se han guardado correctamente y se corresponden a lo mostrado por la versión tabular del editor.
- Comprobar que, en caso de que el diagrama no sea válido, no se guardan los cambios.
- Comprobar que si hay cambios no guardados al intentar abandonar el editor se advierte.

**Caso de uso 04 - Editar workflow**

Desde dentro del editor de workflows el usuario puede clicar en un botón que abre un formulario modal. Tras rellenar los campos pertinentes y clicar en “Guardar” un nodo actividad que representa a la actividad elegida aparece en medio del diagrama.

El usuario puede seleccionar una o más actividades y, clicando en “Eliminar” desde el menú contextual o pulsando la tecla “Suprimir”, eliminar las actividades seleccionadas.

El usuario puede seleccionar una actividad y, clicando en “Editar” desde su menú contextual o haciendo doble clic en ella, abrir un formulario modal que permite editar la actividad del workflow. Tras modificar los campos pertinentes y clicar en “Guardar” la actividad se guarda internamente y refleja, de ser necesario, en su apariencia los cambios.

El usuario puede clicar en una serie de botones con los siguientes nombres: “Paralelo”, “Selección” y “Mezclador”. Tras clicar aparece en medio del diagrama un nodo del tipo indicado por el nombre (por ejemplo un nodo apertura de paralelo al clicar el botón “Paralelo”).

El usuario puede clicar en un botón que activará el modo dibujo de conectores, este modo está activo hasta que el usuario dibuje, arrastrando el ratón dentro del diagrama, un solo conector.

Pruebas de aceptación que ha de cumplir:

- Comprobar que el nodo representa correctamente la actividad y opciones que el usuario ha elegido en el formulario.
- Comprobar que no se puede eliminar ninguna actividad de tipo inicial o final.
- Comprobar que no se puede eliminar ninguna actividad que tenga alguna UT abierta.
- Comprobar que los cambios se han realizado a nivel interno del nodo.
- Comprobar que la apariencia del nodo se actualiza correctamente al aceptar los cambios.
- Comprobar que al clicar uno de los botones aparece el nodo deseado en el diagrama.
- Comprobar que los conectores se dibujan correctamente.

## 6.2. Diseño de la solución

En este apartado se hablará de los detalles en la implementación y diseño del nuevo editor. En esta fase el objetivo es diseñar una solución para satisfacer todos los requisitos especificados.

### 6.2.1. Arquitectura de la aplicación

Los workflows están definidos por las tablas de la base de datos presentes en la figura 33.

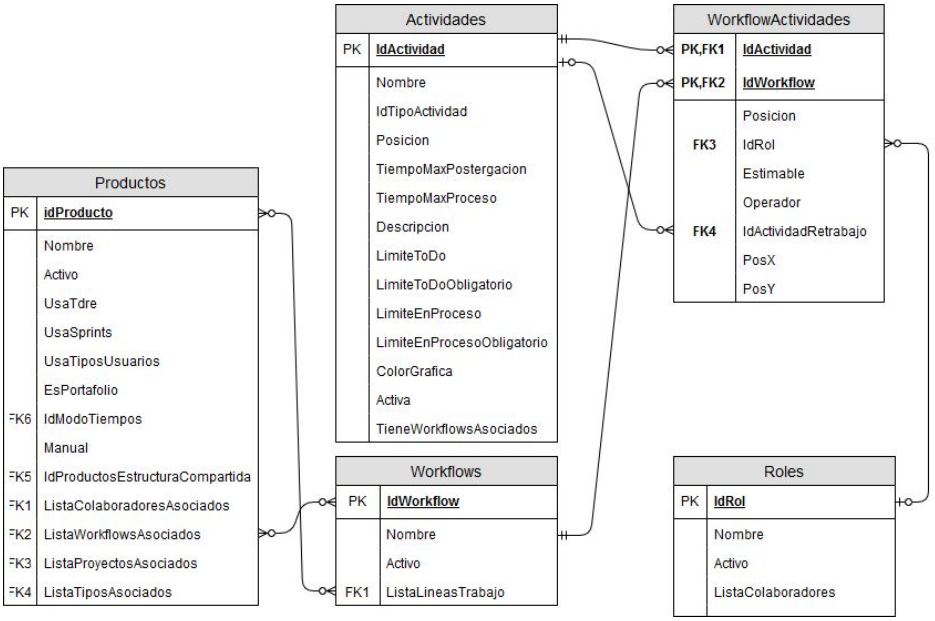



**Figura 33 Diagrama relacional simplificado de la base de datos**

Uno de los mayores retos de la creación del editor ha sido mantener la consistencia con la base de datos. El editor actual realiza los cambios directamente sobre ésta y, por tanto, siempre está en un estado consistente. Por otra parte, en un diagrama pueden existir estados inconsistentes o de caos, es por eso que fue necesario crear un estado diferenciado del de la base de datos para admitir la posibilidad de estos estados caóticos o incompletos mientras se termina de editar el workflow.

Otro reto interesante ha sido la validación del diagrama para verificar que se encuentra en un estado completo y que, por tanto, está listo para reflejar esos cambios en la base de datos. En un diagrama el usuario tiene libertad absoluta para realizar todo tipo de combinaciones de nodos y conexiones, es por eso que la validación se debe realizar teniendo en cuenta todas estas posibilidades y combinaciones que el usuario tiene disponibles.

Finalmente, el último reto digno de mención ha sido el guardado del workflow en la base de datos. El lado servidor de Worki realiza sus propias comprobaciones y modificaciones para asegurar que la base de datos es consistente tras cada operación. Este sistema funciona bien para el editor actual que sigue los mismos principios de consistencia a cada movimiento realizado, pero el editor nuevo guarda todos los cambios realizados de golpe. Por eso, si, por ejemplo, queremos introducir tres actividades recién añadidas y paralelas entre ellas y, al introducirlas en la base de

datos la primera de ellas tiene el atributo operador a “1” (Paralelo) el servidor nos devolverá error diciendo que no es posible que una sola actividad esté en paralelo.

A continuación se comentarán tres ejemplos relevantes de la implementación ya que son fundamentales para la resolución de los retos ya comentados anteriormente.

El primero de los tres trata sobre la carga de los workflows desde la base de datos para su posterior representación en forma de diagrama dentro del editor.

### Carga y representación en forma de diagrama de los workflows

Uno de los primeros problemas que nos encontramos ha sido que cuando se crea un workflow nuevo o éste nunca ha sido cargado como diagrama, los atributos *posX* y *posY* de éste valen *null*. Por eso, durante la carga y representación en forma de diagrama de alguno de los workflows se ha de comprobar esto. Si alguno de las actividades tiene uno de esos dos atributos a *null* el sistema se encargará de automáticamente posicionar los nodos de forma que el diagrama sea legible. Los siguientes fragmentos de código muestran una pequeña parte del mecanismo que hace esto posible:

```
if (i < actividades.length - 1 && actividades[i].operador !== 0) {
  // Si las actividades son paralelas o de selección
  [...]
  // Por cada una de las actividades en paralelo o selección
  for (const actividad of actividadesSimultaneas) {
    const posX = arePosXYNull ? offsetX : actividad.posX;
    const posY = arePosXYNull ? offsetY : actividad.posY;
    newNode = this.generateNodeFromActividad(actividad, posX, posY);
    targetNode = this.diagram.add(newNode);
    newConnector = { ... };
    this.diagram.add(newConnector);
    newConnector = { ... };
    this.diagram.add(newConnector);
    offsetY += 105;
  }
  offsetX += 150;
  [...]
} else {
  // Si la actividad está en secuencia
  const posX = arePosXYNull ? offsetX : this.actividadesDatasource[i].posX;
  const posY = arePosXYNull ? 250 : this.actividadesDatasource[i].posY;
  const newNode: NodeModel = this.generateNodeFromActividad(
    this.actividadesDatasource[i], posX, posY
  );
  targetNode = this.diagram.add(newNode);
  if (sourceNode) {
    const newConnector: ConnectorModel = { ... };
    this.diagram.add(newConnector);
  }
  sourceNode = targetNode;
}
offsetX += 150;
```

La idea es simple pero efectiva, por cada nodo secuencial creado incrementamos el *offset* del eje horizontal en 150 píxeles y, si los nodos son paralelos o de selección entre ellos los ponemos uno encima del otro incrementando el *offset* del eje vertical en 105 píxeles cada vez que se añade uno de estos nodos. Finalmente cuando se va a definir la posición del nodo se comprueba se comprueba si *arePosXYNull* (el resultado de la comprobación previa de si alguna de las actividades tiene *posX* o *posY* a nulo) es verdadero o falso, si es verdadero se utiliza el *offsetX* o *offsetY* calculado por el sistema y, de ser falso, se utilizan las posiciones registradas en la base de datos.

La generación de un nodo a partir de una actividad se realiza mediante la función "generateNodeFromActividad", éste toma como primer argumento el objeto que representa a la actividad y, como segundo y tercer argumento, la posición horizontal y vertical respectivamente en la que estará el nodo y devuelve un nodo que contiene internamente toda la información sobre la actividad y está listo para su inserción en el diagrama.

Crear nodos secuenciales e introducirlos en el diagrama, como se puede ver en el fragmento de código anterior, es trivial, pero la generación de nodos paralelos o de selección entre sí tiene una complejidad añadida, pues se necesita de dos nodos especiales adicionales. El siguiente fragmento de código muestra cómo se realiza la inserción de todos estos nodos:

```
// Ordenamos las actividades por posición para facilitar la generación en
// paralelo y selección
const actividades = this.actividadesDatasource.sort(
  (a, b) => a.posicion - b.posicion
);
for (let i = 0; i < actividades.length; i++) {
  if (i < actividades.length - 1 && actividades[i].operador !== 0) {
    // Si las actividades son paralelas o de selección
    let newNode: NodeModel;
    if (actividades[i].operador === 1) {
      // Si el nodo está en paralelo
      newNode = {
        [...],
        addInfo: {
          tipo: 'Paralelo'
        }
      };
    } else if (actividades[i].operador === 2) {
      // Si el nodo es de selección
      newNode = {
        [...],
        addInfo: {
          tipo: 'Seleccion'
        }
      };
    }
```

```
    offsetX += 150;
    targetNode = this.diagram.add(newNode);
    // Crear enlace entre nodo paralelo/selección al nodo anterior
    let newConnector: ConnectorModel = { ... };
    this.diagram.add(newConnector);
    // Conseguir una lista de todas las actividades con la misma posición
    const actividadesSimultaneas = this.actividadesDatasource.filter(
      (actividad) => actividad.posicion ===
      this.actividadesDatasource[i].posicion
    );
    // Creamos el nodo mezclador que pondrá fin al paralelismo o la
selección
    const mergeNode: NodeModel = {
      [...],
      addInfo: {
        tipo: 'Mezclador'
      }
    };
    const mezclador = this.diagram.add(mergeNode);
    sourceNode = targetNode;
    let offsetY = 145;
    // Recorremos el array con las actividades con misma posición
    for (const actividad of actividadesSimultaneas) {
      const posX = arePosXYNull ? offsetX : actividad.posX;
      const posY = arePosXYNull ? offsetY : actividad.posY;
      // Creamos el nodo actividad
      newNode = this.generateNodeFromActividad(actividad, posX, posY);
      targetNode = this.diagram.add(newNode);
      // Conectamos el nuevo nodo al nodo paralelo/selección
      newConnector = { ... };
      this.diagram.add(newConnector);
      // Conectamos el nuevo nodo al nodo mezclador
      newConnector = { ... };
      this.diagram.add(newConnector);
      offsetY += 105;
    }
    offsetX += 150;
    sourceNode = mezclador;
    // Se le resta uno para compensar por el i++ del for
    i += actividadesSimultaneas.length - 1;
  } else {
    // Si la actividad está en secuencia
    [...]
  }
  offsetX += 150;
}
```

El anterior fragmento de código también nos sirve para ver cómo se generan los conectores entre nodos. Resumiendo su funcionamiento, cuando se añade un nuevo nodo, éste pasa a ser el *targetNode* el cual se conecta al nodo inmediatamente anterior, el nodo *sourceNode*. De esta forma a medida que se añaden nodos nuevos el sistema los conecta inmediatamente al anterior.

### Guardado de los workflows

Como se ha comentado anteriormente, la parte servidor de la aplicación Worki realiza sus propias comprobaciones y modificaciones para asegurar que a cada cambio el estado de la base de datos sigue siendo consistente. Esto ha supuesto una serie de problemas, especialmente con nodos que son paralelos o de selección, que requieren que las operaciones se realicen en un orden específico y de que se hagan comprobaciones adicionales para evitar que el servidor deniegue nuestras solicitudes.

Uno de los problemas que surgen si no realizamos las operaciones en un orden específico es que si, por ejemplo, eliminamos del diagrama un actividad secuencial con posición 10 e introducimos una diferente pero con posición 10 y a la hora de realizar los cambios en la base de datos introducimos la actividad nueva antes de eliminar la antigua, el servidor nos devolverá un error. Por tanto, después de analizar los posibles problemas, se determinó que el orden adecuado para realizar las operaciones es: primero las operaciones de borrar, luego las de actualizar y finalmente las de añadir. De esta forma, tras finalizar todas las operaciones de un mismo tipo, el estado se mantiene consistente.

Finalmente, podemos hablar de los problemas que surgen si no se realizan esas comprobaciones y modificaciones adicionales sobre las actividades del diagrama que se han mencionado anteriormente. Si añadimos al diagrama un set nuevo de nodos paralelos o de selección entre sí, a la hora de insertarlos en la base de datos el primero hace que el servidor nos de error. Esto se debe a que a la hora de insertarlo, su atributo operador vale, dependiendo del caso, 1 ó 2 y, como ya hemos mencionado, no puede haber un sólo nodo en paralelo o de selección por lo que el servidor lo rechaza. La siguiente función se encarga del guardado del workflow, en ella se puede apreciar el orden de las operaciones las comprobaciones adicionales necesarias para resolver el problema que se acaba de mencionar y otro:

```
async saveWorkflow() {
  // Comprobamos si el diagrama es válido
  if (this.isDiagramValid()) {
    // Conseguimos una lista de las actividades en el diagrama
    let actividadesWorkflow: WorkflowActividad[] =
      this.diagram.nodes.map((node) => {
        if (node.addInfo['infoActividad']) {
          const actividad = node.addInfo['infoActividad'];
          actividad.posX = node.offsetX;
          actividad.posY = node.offsetY;
          return actividad;
        }
```

```
      });
    actividadesWorkflow = actividadesWorkflow.filter((act) => act);
    // Creamos una lista que contiene las actividades en común entre la
base de
    // datos y el diagrama.
    const actividadesComunes: WorkflowActividad[] =
actividadesWorkflow.filter(
      (act) => this.actividadesDatasource.map((a) =>
        a.idActividad).indexOf(act.idActividad) > -1
      );
    // Creamos una lista que contiene las actividades que sólo están en el
    // diagrama.
    let actividadesUnicas = actividadesWorkflow.filter(
      (act) => actividadesComunes.map((a) =>
        a.idActividad).indexOf(act.idActividad) <= -1
      );
    actividadesUnicas = actividadesUnicas.concat(
      this.actividadesDatasource.filter(
       (act) => actividadesComunes.map((a) =>
         a.idActividad).indexOf(act.idActividad) <= -1
      )
    );
    // Determinamos qué actividades van a ser eliminadas de la base de
datos
    const actividadesEliminar = actividadesUnicas.filter(
      (act1) => this.actividadesDatasource.map((act2) =>
        act2.idActividad).indexOf(act1.idActividad) > -1
    );
    // Determinamos qué actividades van a ser añadidas a la base de datos
    const actividadesAñadir = actividadesUnicas.filter(
      (act1) => this.actividadesDatasource.map((act2) =>
        act2.idActividad).indexOf(act1.idActividad) < 0
    );
    // Eliminamos las actividades
    for (const actividad of actividadesEliminar) {
      await this.workflowsService.deleteWorkflowActividad(
        this.idWorkflow, actividad.idActividad
       ).toPromise();
    }
    // Actualizamos las actividades en común
    for (const actividad of actividadesComunes) {
      // Primera comprobación adicional
      if (
        actividad.operador !== 0 &&
        actividadesComunes.filter((act) => act.posicion ===
         actividad.posicion).length <= 1
      ) {
        actividad.operador = 0;
      }
      await this.workflowsService.updateWorkflowActividad(
        this.idWorkflow, actividad.idActividad, actividad
```

```
        ).toPromise();
    }
    // Añadimos las actividades
    const listaPosicionesArregladas = [];
    for (const actividad of actividadesAñadir) {
      // Segunda comprobación adicional
      if (
        actividad.operador !== 0 &&
        this.actividadesDatasource.filter((act) =>
          act.posicion === actividad.posicion
        ).length === 0 &&
        listaPosicionesArregladas.indexOf(actividad.posicion) < 0
      ) {
        listaPosicionesArregladas.push(actividad.posicion);
        actividad.operador = 0;
      }
      await this.workflowsService.createWorkflowActividad(
        this.idWorkflow, actividad
      ).toPromise();
    }

    this.workflowsService.getActividades(
      this.idWorkflow, false
    ).subscribe((ret) => {
      this.actividadesDatasource = ret;
    });
    this.toastService.success('Se ha guardado el workflow', undefined);
    this.workflowsDiagramaService.existenCambios = false;
  }
}
```

Podemos observar que en total se han realizado dos comprobaciones adicionales. La segunda de ellas resuelve el problema ya descrito anteriormente. La primera, sin embargo, evita que el servidor nos de error si eliminamos y dejamos sólo una de las actividades paralelas o de selección entre sí, para luego añadir nuevas actividades que reemplacen a las eliminadas. El error se produce porque, tras eliminar de la base de datos las actividades, el servidor pone el valor “operador” de la actividad restante que estaba en paralelo/selección a cero, pero en el diagrama este atributo “operador” vale uno, para paralelo, o dos, para selección y, dado que no puede haber una sola actividad paralela o de selección, el servidor rechaza la petición.

### Validación del workflow

Como se ha mencionado al principio, uno de los problemas que surgen al verificar un diagrama ha sido que el usuario tiene un alto grado de libertad. Un alto número de posibilidades implica riesgo en que alguna de ellas no sea un estado correcto pero aún así pase el test de validación. Por tanto, ha sido importante diseñar un buen sistema de validación que esté blindado ante esta posibilidad.

Para conseguir esto se han realizado dos validaciones, una a nivel individual de cada nodo y otra a nivel de recorrido (yendo desde el nodo inicial hasta el final). Primero hablaremos de la validación a nivel de nodos. Cada tipo de nodo tiene diferentes reglas que ha de cumplir para que se le considere válido. A continuación se muestra el código que valida un nodo actividad normal:

```
private isTaskNodeValid(node: Node): Array<string> {
  const messages: string[] = [];
  // Comprueba que el nodo tenga exactamente una entrada y salida
  if (node.outEdges.length === 1 && node.inEdges.length === 1) {
    const outConnector = <Connector>this.diagram.connectors.find(
      (connector) => connector.id === node.outEdges[0]
    );
    const inConnector = <Connector>this.diagram.connectors.find(
      (connector) => connector.id === node.inEdges[0]
    );
    // Comprueba que la entrada y salida conecten a otro nodo
    if (outConnector.targetID === '' || inConnector.sourceID === '')
{
      messages.push(
        '- El conector de salida del nodo ' +
        node.addInfo['infoActividad'].nombreActividad +
        ' no está conectado a otro nodo.'
      );
    }
  } else {
    messages.push(
      '- El nodo ' +
      node.addInfo['infoActividad'].nombreActividad +
      ' debe tener sólo un conector de entrada y salida.'
    );
  }
  return messages;
}
```

De esta forma se comprueba que un nodo actividad normal sólo tenga una entrada y salida y que, además, estas estén conectadas a otro nodo y que no simplemente “cuelgan” de éste. Esta función devuelve una lista de mensajes de error que la función padre “isDiagramValid” añadirá a la suya propia para luego mostrarle los mensajes al usuario y devolver verdadero o falso en base a la longitud de dicha lista.

La validación de los nodos inicio y final es trivial y muy parecida a la del nodo actividad normal, pero con un pequeño cambio en la condición del número de entradas y salidas. En el caso del nodo inicial se cambia la condición de “exactamente una entrada y salida” a “exactamente una salida y cero entradas” y, en el caso del nodo final se cambia dicha condición por “exactamente una entrada y cero salidas”.

Por otro lado, la validación de los nodos de apertura de paralelo y selección así como el mezclador, o de cierre, es interesante y digna de mención. A continuación se muestra el código encargado de la validación de los nodos de apertura de paralelo y selección:

```
private isParaleloNodeValid(node: Node): Array<string> {
  const messages: string[] = [];
  // Comprobamos que haya una entrada y más de una salida
  if (node.inEdges.length === 1 && node.outEdges.length >= 2) {
    let nodoMezclador: Node = null;
    let numConectoresInconexos = 0;
    for (const connectorID of node.outEdges) {
      let connector = <Connector>this.diagram.connectors.find(
        (con) => con.id === connectorID
      );
      // Comprobamos si alguno de las salidas está inconexa
      if (connector.targetID === '') {
        numConectoresInconexos++;
      } else {
        // Conseguimos el nodo al que está conectado el nodo paralelo
        const nodoIntermedio = <Node>this.diagram.nodes.find(
          (n) => n.id === connector.targetID
        );
        // Comprobamos que el nodo sea válido y, por tanto, tenga una salida
        if (this.isTaskNodeValid(nodoIntermedio).length <= 0) {
          connector = <Connector>this.diagram.connectors.find(
            (con) => con.id === nodoIntermedio.outEdges[0]
          );
          const targetNode = <Node>this.diagram.nodes.find(
            (n) => n.id === connector.targetID
          );
          // Comprobamos que el nodo está conectado a un nodo mezclador
          if (targetNode.addInfo['tipo'] === 'Mezclador') {
            // Si es el primer nodo mezclador que encontramos lo guardamos
            nodoMezclador = nodoMezclador == null ? targetNode : nodoMezclador;
            // Comprobamos que el nodo mezclador es el mismo para todos los
            // nodos en paralelo/selección
            if (nodoMezclador.id !== targetNode.id) {
              messages.push( ... );
            }
          } else {
            messages.push( ... );
          }
        }
      }
    }
    if (numConectoresInconexos > 0) {
      messages.push( ... );
    }
  } else {
    messages.push( ... );
  }
  return messages;
```

```
}
```

La validación de los nodos mezcladores es muy similar a la mostrada por el fragmento de código anterior pero el proceso es justamente el inverso, yendo hacia atrás en el diagrama para comprobar la validez del nodo.

A continuación, hablaremos sobre la validación a nivel de recorrido, esta se encarga de asegurar que haya un camino desde el nodo inicial al final, de que el atributo “posicion” de las actividades sea el correcto y, ya que recorre el diagrama entero, verifica y cambia el atributo “operador” de las actividades para que sea el correcto. A continuación tenemos el código encargado de realizar estas comprobaciones:

```
private arePositionsCorrect() {
  const nodes = <Node[]>this.diagram.nodes;
  const connectors = <Connector[]>this.diagram.connectors;
  // Recupera el nodo inicial
  let nodoActual = nodes.find((node) => node.addInfo['tipo'] === 'Start');
  let posicionAnterior = -1;
  // Continuar hasta que se alcance el nodo final
  while (nodoActual.addInfo['tipo'] !== 'End') {
    if (
      nodoActual.addInfo['tipo'] === 'Task' ||
      nodoActual.addInfo['tipo'] === 'Start'
    ) {
      // Si se trata de un nodo normal o inicio
      nodoActual.addInfo['infoActividad'].operador = 0;
      // Comprobar que su posición es mayor a la del anterior
      if (nodoActual.addInfo['infoActividad'].posicion <= posicionAnterior) {
        return false;
      }
      posicionAnterior = nodoActual.addInfo['infoActividad'].posicion;
      const conector = connectors.find(
        (connector) => connector.id === nodoActual.outEdges[0]
      );
      // Avanzamos al siguiente nodo
      nodoActual = nodes.find((node) => node.id === conector.targetID);
    } else if (nodoActual.addInfo['tipo'] === 'Mezclador') {
      // Si se trata de un nodo mezclador simplemente avanzamos al siguiente
      const conector = connectors.find(
        (connector) => connector.id === nodoActual.outEdges[0]
      );
      nodoActual = nodes.find((node) => node.id === conector.targetID);
    } else if (nodoActual.addInfo['tipo'] === 'Paralelo' ||
               nodoActual.addInfo['tipo'] === 'Seleccion') {
      // Si se trata de un nodo apertura de paralelo o de selección
      let posicion;
      // Por cada uno de las salidas
      for (let i = 0; i < nodoActual.outEdges.length; i++) {
        const connector = nodoActual.outEdges[i];
        const conectorSaliente = connectors.find(
          (con) => con.id === connector
```

```
        );
        // Obtenemos el nodo actividad al que llevan
        const node = nodes.find((n) => n.id === conectorSaliente.targetID);
        // Aseguramos que el operador sea el correcto
        node.addInfo['infoActividad'].operador =
          nodoActual.addInfo['tipo'] === 'Paralelo' ? 1 : 2;
        if (!posicion) {
          // Si es el primer nodo que encontramos guardamos su posición
          posicion = node.addInfo['infoActividad'].posicion;
        }
        // Comprobamos que las posiciones sean la mismas para los nodos
        // paralelos o de selección entre sí y que sean mayores a las del nodo
        // anterior al nodo paralelo
        if (
          node.addInfo['infoActividad'].posicion <= posicionAnterior ||
          posicion !== node.addInfo['infoActividad'].posicion
        ) {
          return false;
        }
      }
      let conector = connectors.find(
        (connector) => connector.id === nodoActual.outEdges[0]
      );
      const nodo = nodes.find((node) => node.id === conector.targetID);
      conector = connectors.find(
        (connector) => connector.id === nodo.outEdges[0]
      );
      // Avanzamos al nodo siguiente
      nodoActual = nodes.find((node) => node.id === conector.targetID);
      posicionAnterior = posicion;
    }
  }
  return true;
}
```

Todas las funciones mencionadas anteriormente son auxiliares de la función “isDiagramValid” que, como se ha dicho anteriormente, se encarga de mostrar la lista de errores al usuario y devolver verdadero o falso en función de si el diagrama es válido o no. Aquí se muestra el código entero de la función:

```
isDiagramValid() {
  let listaProblemas: string[] = [];
  for (const node of <Node[]>this.diagram.nodes) {
    // Comprobar si el nodo es inconexo
    if (node.outEdges.length === 0 && node.inEdges.length === 0) {
      if (
        node.addInfo['tipo'] === 'Task' ||
        node.addInfo['tipo'] === 'Start' ||
        node.addInfo['tipo'] === 'End'
      ) {
        listaProblemas.push('- El nodo ' +
          node.addInfo['infoActividad'].nombreActividad +
```

```
          ' no está conectado.');
      } else {
        listaProblemas.push('- Al menos uno de los nodos ' +
          node.addInfo['tipo'] +
          ' no está conectado.');
      }
    } else {
      let messages = [];
      // Validar el nodo en base a su tipo
      switch (node.addInfo['tipo']) {
        case 'Task':
          messages = this.isTaskNodeValid(node);
          break;
        case 'End':
          messages = this.isEndNodeValid(node);
          break;
        case 'Start':
          messages = this.isStartNodeValid(node);
          break;
        case 'Mezclador':
          messages = this.isMezcladorNodeValid(node);
          break;
        case 'Paralelo':
        case 'Seleccion':
          messages = this.isParaleloNodeValid(node);
          break;
      }
      if (messages.length > 0) {
        listaProblemas = listaProblemas.concat(messages);
      }
    }
  }
  // Si no hay problemas se hace la validación a nivel de recorrido
  if (listaProblemas.length === 0 && !this.arePositionsCorrect()) {
    listaProblemas.push('- Las posiciones de los nodos son erroneas,' +
      ' deben ir de menor a mayor.');
  }
  this.listaProblemas = listaProblemas;
  if (listaProblemas.length > 0) {
    // Si hay problemas se abre una ventana emergente con la lista de problemas
    this.confirmDialogService.open({
      header: 'Se han encontrado problemas',
      message: this.listaProblemas.join('<br />'),
      buttons: [
        {
          label: 'Aceptar',
          icon: 'fa fa-check'
        }
      ]
    });
  }
  // Si no hay problemas se considera el diagrama como válido
  return listaProblemas.length === 0;
}
```

### Estadísticas de la implementación

El componente tiene un total de 31 funciones, 27 atributos o propiedades y un total de 1304 líneas de código, de las cuales 1197 fueron destinadas a funciones. Se han realizado los 4 casos de uso y las 21 pruebas de aceptación. La distribución de las líneas de código por cada función ha sido la siguiente:

| Función | Nº Líneas |
|---|---|
| ngOnInit | 150 |
| doubleClick | 11 |
| collectionChange | 8 |
| contextMenuOpen | 25 |
| contextMenuClick | 15 |
| addNew | 13 |
| onEditClick | 13 |
| customRemove | 30 |
| connectionChange | 7 |
| toggleDrawnToolOn | 9 |
| loadData | 132 |
| abrirAdvertenciaGuardado | 22 |
| getActividadesRetrabajoDesplegable | 21 |
| getActividadesDesplegable | 41 |
| defineModal | 109 |
| defineCRUD | 11 |
| getOperadores | 16 |
| setNombreOperador | 12 |
| addSymbol | 61 |
| save | 43 |
| saveWorkflow | 64 |
| checkConnectorValidity | 25 |

| | |
|---|---|
| isDiagramValid | 53 |
| isTaskNodeValid | 21 |
| isStartNodeValid | 20 |
| isEndNodeValid | 20 |
| isMezcladorNodeValid | 45 |
| isParaleloNodeValid | 49 |
| arePositionsCorrect | 45 |
| generateNodeFromActividad | 93 |
| getFontColor | 13 |

A continuación, en la figura 34 se presenta un diagrama de las funciones más relevantes que muestra cómo están relacionadas las unas a las otras, donde una función emplea a las que sus conectores salientes están apuntando.

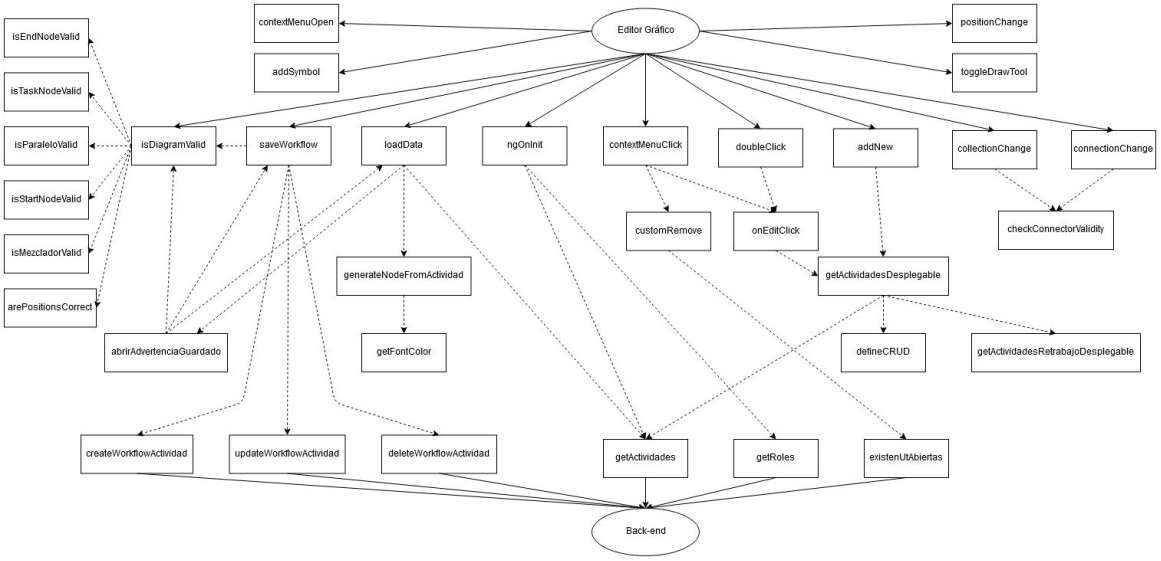



**Figura 34 Diagrama de las funciones relevantes**

# Capítulo 7 Editor gráfico de workflows en Worki

En este capítulo se ofrece una pequeña guía de uso del editor desarrollado en este trabajo de fin de grado.

## 7.1. Guía de uso

Una vez creado un nuevo workflow, se nos mostrará algo parecido a la figura 35, un diagrama con las dos actividades incluidas en el workflow por defecto.

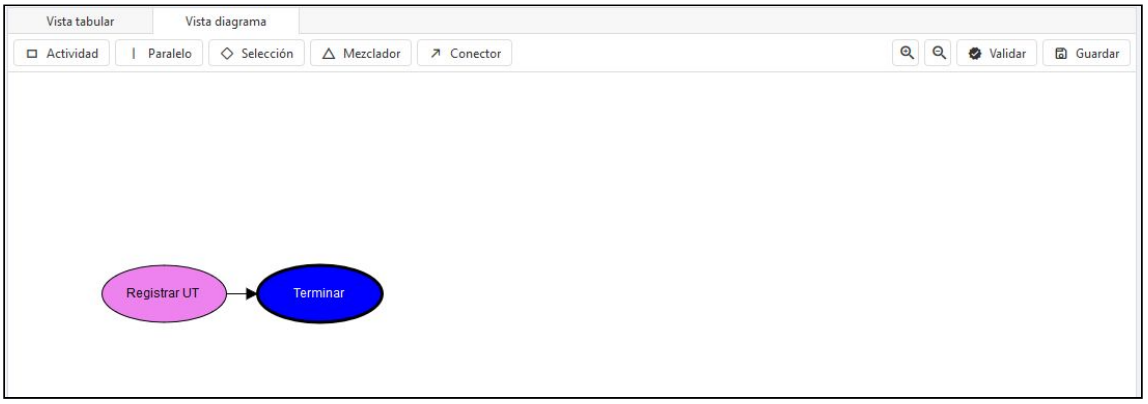



**Figura 35 Diagrama tras la creación de un workflow**

Para añadir una nueva actividad al workflow clicamos en el botón “Actividad” del editor, esto muestra un formulario como el de la figura 36 que, tras rellenar como deseemos y clicar en “Guardar” añadirá la actividad al workflow, resultando en algo parecido a la figura 37.

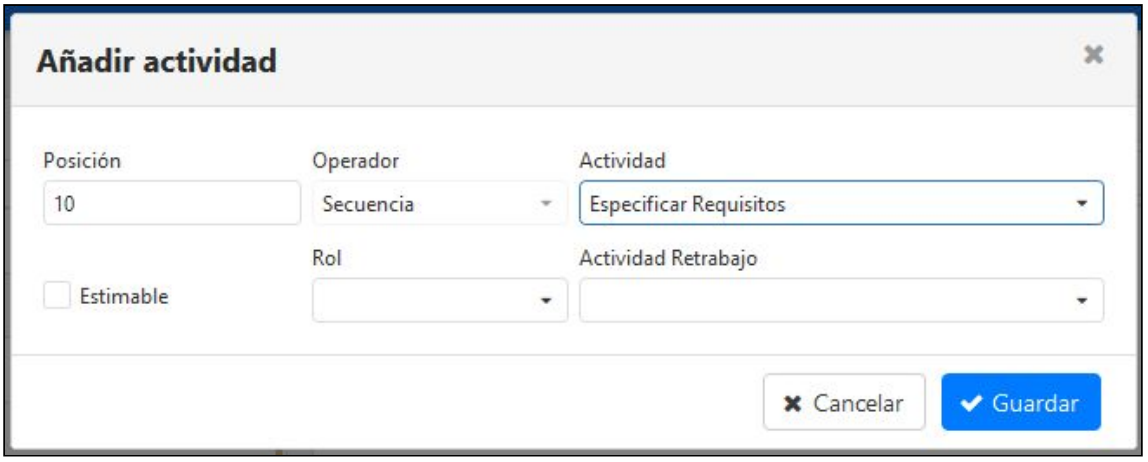



**Figura 36 Formulario para añadir una actividad**


**Figura 37 Diagrama tras añadir una actividad**

Podemos incluir la nueva actividad en la secuencia conectandola a las otras utilizando el botón “Conector” y dibujando los conectores pertinentes. La figura 38 muestra el resultado.

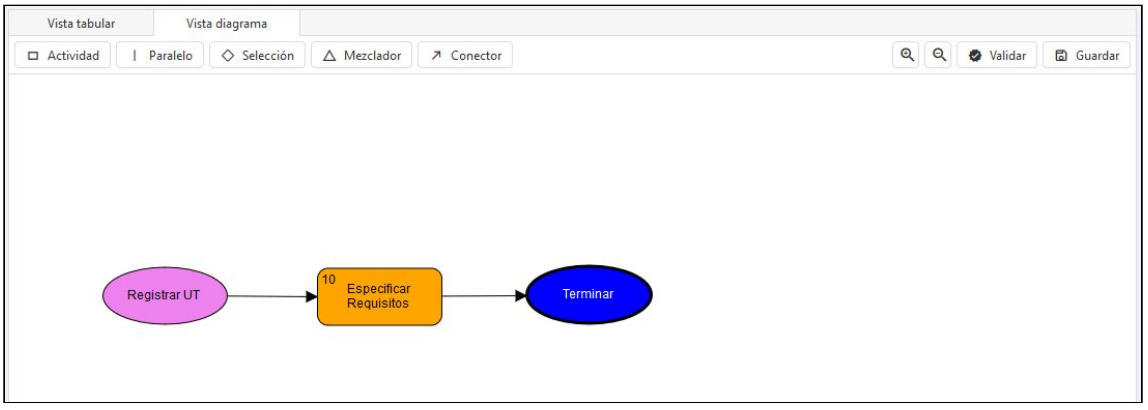



**Figura 38 Diagrama tras conectar el nodo nuevo a los demás**

Para crear un grupo de nodos en paralelo primero clicamos en el botón “Paralelo” y el botón “Mezclador” para crear un nodo de apertura de paralelo y un nodo mezclador respectivamente. A continuación, los conectaremos de forma pertinente, resultando en algo parecido a la figura 39.

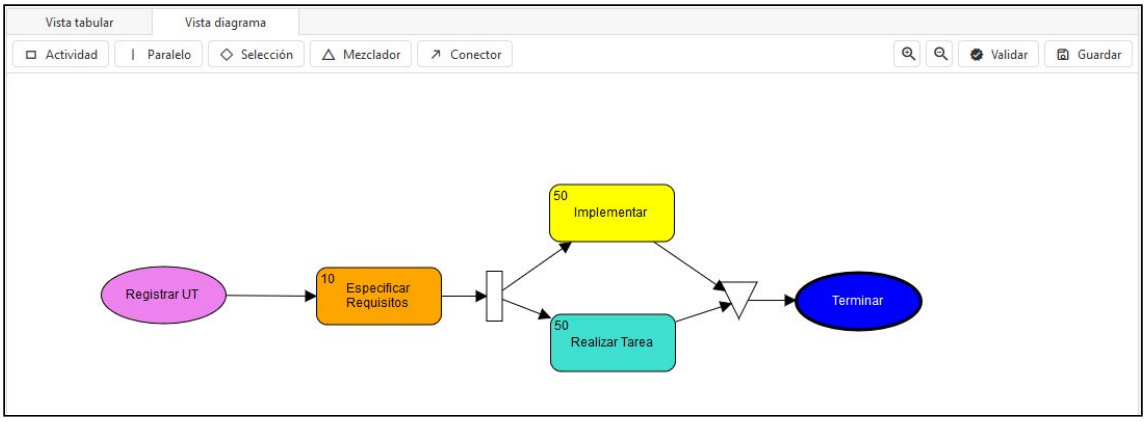



**Figura 39 Diagrama tras crear un grupo de nodos en paralelo**

Si queremos editar alguna actividad basta con hacer doble clic sobre ella, esto abrirá un formulario muy similar al de la figura 36, tras realizar las modificaciones deseadas y clicar en “Guardar” los cambios se aplicarán a la actividad.

Finalmente, si clicamos en el botón “Guardar” el sistema guardará el workflow, si hay algún problema con el estado actual se mostrará una lista de problemas como la de la figura 40.

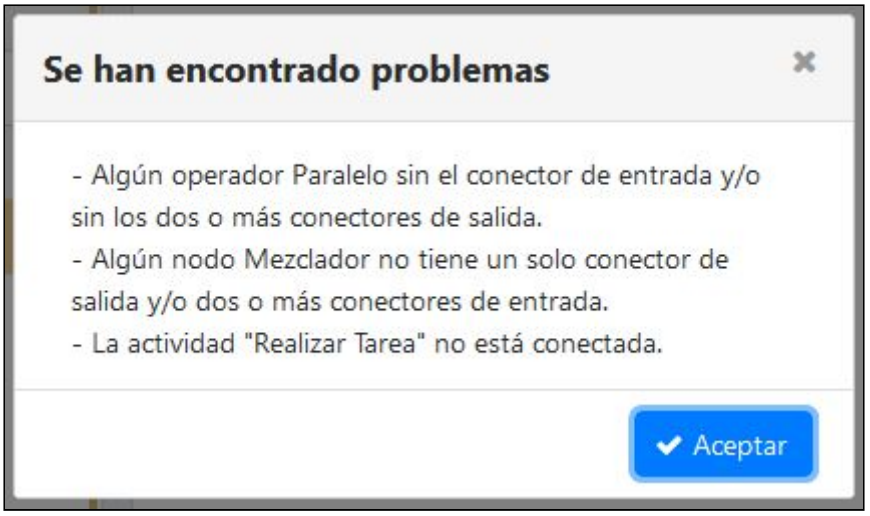



**Figura 40 Lista de problemas**

## 7.2. Verificaciones del diagrama

A continuación se muestran ejemplos de diagramas incorrectos y de cómo el sistema de verificación detecta y reporta los errores al usuario.

En el ejemplo que muestra la figura 41 se muestra cómo el sistema identifica y reporta cuatro errores en el diagrama: la actividad inicial “Registrar UT” no está conectada, la actividad “Especificar Requisitos” no tiene ninguna entrada, la actividad “Probar” no tiene ninguna salida y el conector de entrada de la actividad final “Terminar” no está conectado a otra.

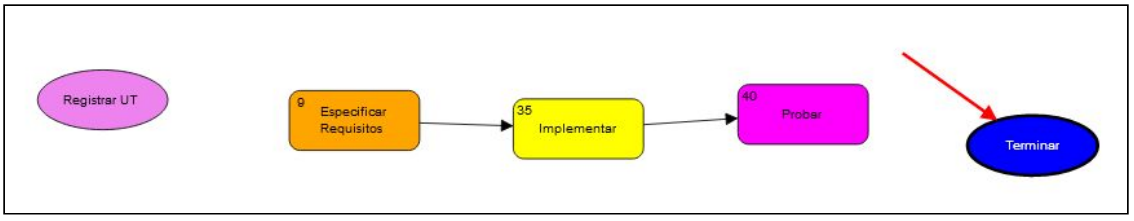



**Figura 41 Primer ejemplo de diagrama erróneo**

Tras pulsar el botón de validar o guardado se nos muestra el modal de la figura 42 con una lista de todos los errores encontrados en el diagrama.

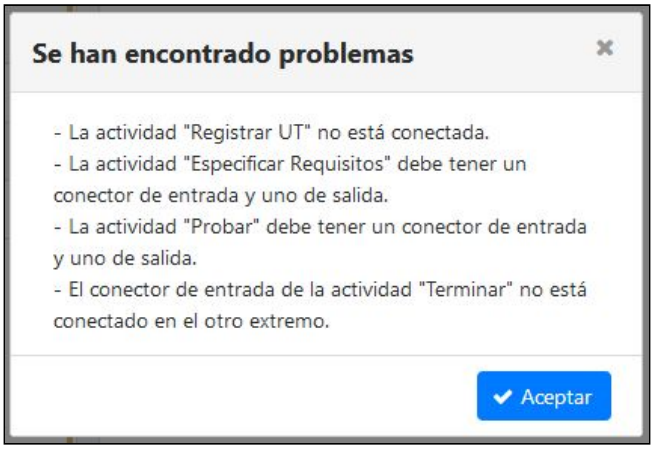



**Figura 42 Modal con la lista de problemas del diagrama de la figura 41**

La figura 43 muestra un diagrama con dos errores: el nodo selección (representado por un rombo) sólo tiene un conector de salida y el nodo mezclador (representado por un triángulo) tiene dos conectores de entrada inconexos.

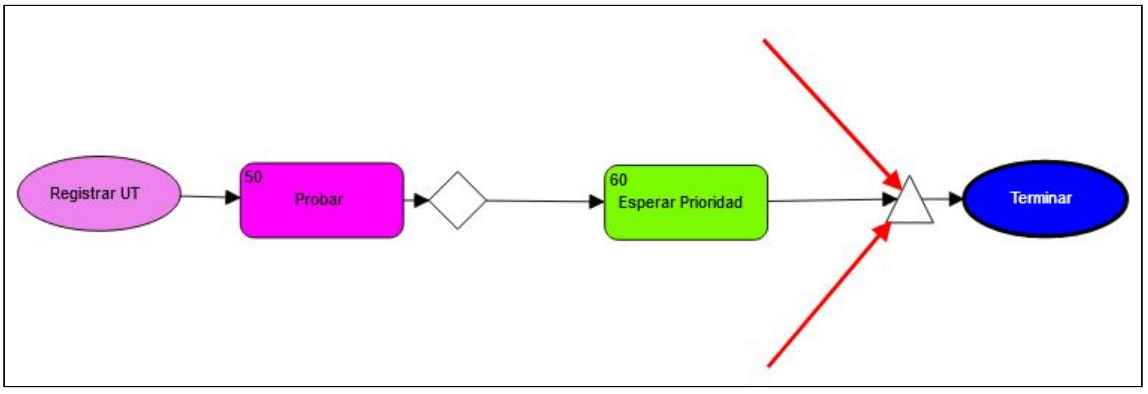



**Figura 43 Segundo ejemplo de un diagrama erróneo**

Tras pulsar el botón de validar o guardado se nos muestra el modal de la figura 44 con una lista de todos los errores encontrados en el diagrama.

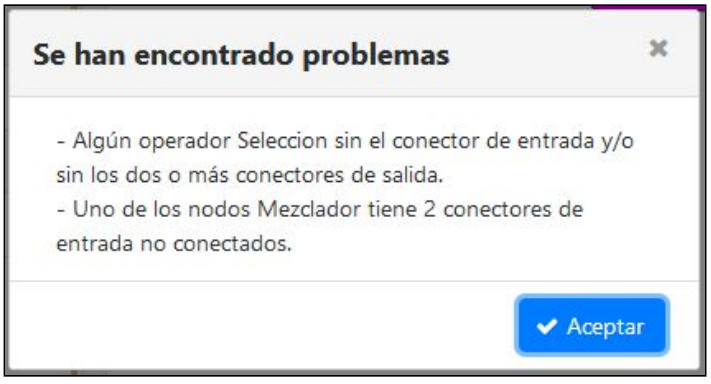



**Figura 44 Modal con la lista de problemas del diagrama de la figura 43**

Finalmente la figura 45 nos muestra un diagrama más complejo con múltiples problemas listados en la figura 46.

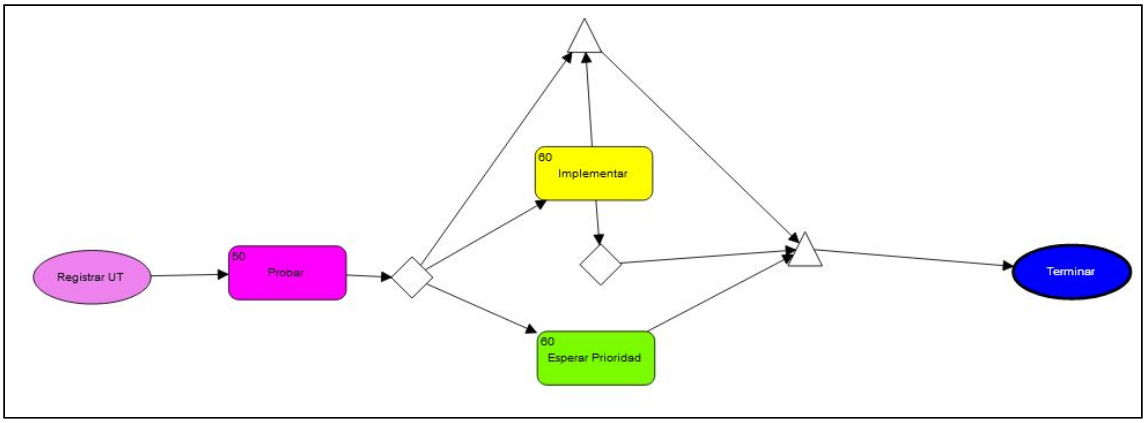



**Figura 45 Tercer ejemplo de un diagrama erróneo**

Tras pulsar el botón de validar o guardado se nos muestra el modal de la figura 46 con una lista de todos los errores encontrados en el diagrama.

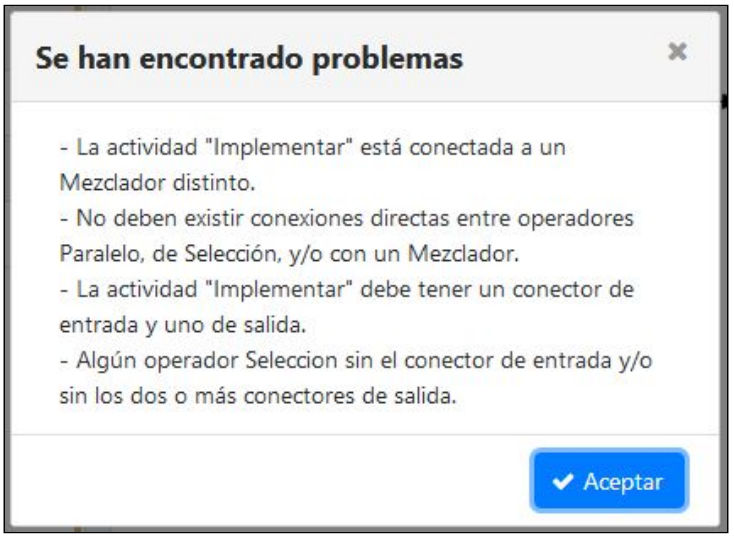



**Figura 46 Modal con la lista de problemas del diagrama de la figura 45**

# Capítulo 8 Conclusiones

## 8.1. Conclusiones y trabajo futuro

Para una correcta y efectiva gestión de proyectos es necesario utilizar herramientas de apoyo y metodologías que faciliten la planificación y el seguimiento de éstos. Considerando el proceso de desarrollo de software como un proceso de negocio y, como tal, automatizable y controlable a través del uso de workflows. Éstos han de ser flexibles, pues el desarrollo de software es un entorno dado a los cambios en los requisitos y necesidades del proyecto, y han de ayudar al desarrollo ofreciendo información valiosa sobre el estado del mismo.

Worki es la herramienta de apoyo a TUNE-UP Process, en Worki los workflows son protagonistas para la gestión del trabajo.

Consideramos que los objetivos del presente TFG se han cumplido. Se ha creado un nuevo editor de workflows que ha supuesto una mejora respecto al editor actual, ya que el modelado y visualización de workflows, o procesos, es más natural e intuitivo cuando se realiza sobre un diagrama. Contar con un editor sencillo pero potente donde se puedan definir estos workflows de forma simple y cómoda supone una gran mejora para la herramienta Worki.

Para la creación del componente fue necesario investigar el estado del arte de componentes gráficos para Angular centrados en la creación y edición de diagramas. Esto nos llevó a comparar distintas posibilidades, tanto soluciones comerciales como de código abierto. También se realizó un pequeño ejemplo para evaluar la complejidad a nivel de código de tareas simples como la creación e inserción dentro del diagrama de un nodo y un conector. Al final se acabó eligiendo Syncfusion Essential JS 2 Diagram por su alto grado de completitud, documentación y precio.

El componente del editor desarrollado se encuentra actualmente completamente integrado y probado dentro de la propia herramienta de Worki web y se está utilizando en implantaciones reales de Worki.

Desde el punto de vista de la experiencia personal, la realización de este trabajo de fin de grado ha permitido al autor colaborar dentro de un proyecto real, ampliar y profundizar los conocimientos sobre procesos de negocio, metodologías y workflows aprendidos durante el grado en asignaturas como Proyecto de Ingeniería de Software o Desarrollo de software dirigido por modelos, y aprender nuevas tecnologías para el desarrollo web tales como Angular y sobre alguna de las distintas librerías gráficas disponibles para ésta. Todo esto no hubiera sido posible sin la colaboración del equipo de desarrollo de la versión web de Worki.

Dentro de las futuras líneas de trabajo se tiene:

- Mejorar el sistema de validación para que éste marque en rojo y/o con un *tooltip* los elementos problemáticos del diagrama.

- Modificar el sistema para eliminar el atributo posición de las actividades, permitiendo así un modelado más rápido de los workflows, aunque esto puede suponer un reto pues este atributo es usado por otras partes de la herramienta.
- Mejorar la navegabilidad dentro del diagrama para que esta sea más fluida.
- Introducir una pequeña ayuda para que el usuario pueda entender el funcionamiento de los nodos especiales mejor.

# Anexo: Pruebas de aceptación

A continuación se muestran las pruebas de aceptación realizadas sobre el editor gráfico creado.

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 01 | **Código del Caso de Uso:** 01 |
| **Descripción de la prueba:**<br><br>Comprobar que no se puede modificar el diagrama si no se tiene permisos de administrador. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y que el usuario que accede al workflow no sea administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario intenta realizar alguna modificación dentro del diagrama.<br>3. El usuario intenta clicar en el botón de guardado. | |
| **Resultado esperado:**<br><br>El resultado esperado es que el usuario no puede hacer clic en el botón de guardado pues este se encuentra desactivado y, por tanto, no se puede modificar el estado del workflow. | |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 02 | **Código del Caso de Uso:** 01 |
| **Descripción de la prueba:**<br><br>Comprobar que tras cargar, el diagrama representa correctamente el workflow cargado. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos. | |

| |
|---|
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario comprueba que el diagrama se ha generado exitosamente. |
| **Resultado esperado:**<br><br>El diagrama carga sin problemas sus contenidos que reflejan con precisión el workflow guardado en la base de datos. |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 03 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:**<br><br>Comprobar que el diagrama no es válido si hay nodos inconexos en el diagrama. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y el usuario debe ser administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario añade una actividad nueva al diagrama.<br>3. El usuario añade un nodo especial al diagrama.<br>4. El usuario clica en el botón de validar. | |
| **Resultado esperado:**<br><br>El resultado del paso 4 es que se muestra en una ventana una lista de problemas y, dos de los mensajes de la lista son “El nodo <nombre actividad> no está conectado.” y “Al menos uno de los nodos <tipo nodo especial> no está conectado.” | |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 04 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:** | |

| |
|---|
| Comprobar que el diagrama no es válido si el nodo inicial tiene entradas o más de una salida. |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y el usuario debe ser administrador. |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario añade un conector entrante al nodo inicial.<br>3. El usuario clica en el botón de validar.<br>4. El usuario elimina el conector entrante creado.<br>5. El usuario añade dos conectores de salida al nodo inicial.<br>6. El usuario clica en el botón de validar. |
| **Resultado esperado:**<br><br>El resultado en los pasos 3 y 6 es el mismo: se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “El nodo <nombre actividad> debe tener sólo un conector de salida y ninguno de entrada.” |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 05 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:**<br><br>Comprobar que el diagrama no es válido si el nodo final tiene salidas o más de una entrada. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y el usuario debe ser administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario añade un conector de salida al nodo final.<br>3. El usuario clica en el botón de validar.<br>4. El usuario elimina el conector de salida creado.<br>5. El usuario añade dos conectores entrante al nodo final. | |

| |
|---|
| 6. El usuario clica en el botón de validar. |
| **Resultado esperado:**<br><br>El resultado en los pasos 3 y 6 es el mismo: se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “El nodo <nombre actividad> debe tener sólo un conector de entrada y ninguno de salida.” |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 06 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:**<br><br>Comprobar que el diagrama no es válido si alguno de los nodos actividad no tienen exactamente una entrada y salida. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow con al menos tres actividades en la base de datos y el usuario debe ser administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario elimina uno de los conectores de alguno de los nodos de actividad.<br>3. El usuario clica en el botón de validar. | |
| **Resultado esperado:**<br><br>El resultado en el paso 3 es que se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “El nodo <nombre actividad> debe tener sólo un conector de entrada y salida.” | |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 07 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:** | |

| |
|---|
| Comprobar que el diagrama no es válido si el valor del atributo posición de alguna de las actividades en secuencia es mayor o igual que el de la siguiente actividad. |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow con al menos cuatro actividades en la base de datos y el usuario debe ser administrador. |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario edita uno de los nodos para que la posición de éste sea mayor que la del nodo siguiente.<br>3. El usuario clica en el botón de validar. |
| **Resultado esperado:**<br><br>El resultado en el paso 3 es que se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “Las posiciones de los nodos son erróneas, deben ir de menor a mayor.” |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 08 | **Código del Caso de Uso:** 02 |
| **Descripción de la prueba:**<br><br>Comprobar que el diagrama no es válido si el valor del atributo posición de alguna las actividades que sean paralelas o de selección entre sí es diferente. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow con al menos dos actividades en paralelo en la base de datos y el usuario debe ser administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario edita uno de los nodos en paralelo para cambiar su posición a una distinta a la de los otros nodos en paralelo.<br>3. El usuario clica en el botón de validar. | |

**Resultado esperado:**

El resultado en el paso 3 es que se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “Las posiciones de los nodos son erróneas, deben ir de menor a mayor.”

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 09 | **Código del Caso de Uso:** 02 |

**Descripción de la prueba:**

Comprobar que el diagrama no es válido si un nodo de apertura de paralelo o selección no tiene ninguna entrada o tiene menos de dos salidas.

**Condiciones de ejecución:**

Debe existir un workflow con al menos dos actividades en paralelo en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario elimina el conector de entrada del nodo de apertura de paralelo.
3. El usuario clica en el botón de validar.
4. El usuario restaura el conector de entrada del nodo de apertura de paralelo.
5. El usuario elimina los conectores de salida del nodo de apertura de paralelo hasta sólo dejar uno.
6. El usuario clica en el botón de validar.

**Resultado esperado:**

El resultado en los pasos 3 y 6 es el mismo: se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “Uno de los nodos Paralelo no tiene un solo conector de entrada y/o dos o más de salida.”

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 10 | **Código del Caso de Uso:** 02 |

**Descripción de la prueba:**

Comprobar que el diagrama no es válido si un nodo mezclador no tiene ninguna salida o tiene menos de dos entradas.

**Condiciones de ejecución:**

Debe existir un workflow con al menos dos actividades en paralelo en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario elimina el conector de salida del nodo mezclador.
3. El usuario clica en el botón de validar.
4. El usuario restaura el conector de salida del nodo mezclador.
5. El usuario elimina los conectores de entrada del nodo mezclador hasta dejar sólo uno.

**Resultado esperado:**

El resultado en los pasos 3 y 6 es el mismo: se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “Uno de los nodos Mezclador no tiene un solo conector de salida y/o dos o más de entrada.”

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 11 | **Código del Caso de Uso:** 02 |

**Descripción de la prueba:**

Comprobar que el diagrama no es válido si un grupo de nodos que sean paralelos o de selección entre sí provienen de distintos nodos de apertura de paralelo o selección o si conducen a distintos nodos mezcladores.

**Condiciones de ejecución:**

Debe existir un workflow con dos grupos de dos nodos paralelos entre sí en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario cambia el conector de salida de uno de los nodos del primer grupo paralelo para que lleve al nodo mezclador del segundo grupo.

3. El usuario cambia el conector de salida de uno de los nodos del segundo grupo paralelo para que lleve al nodo mezclador del primer grupo.
4. El usuario clica en el botón de validar.
5. El usuario deshace los cambios realizados en los pasos 2 y 3.
6. El usuario cambia el conector de entrada de uno de los nodos del primer grupo paralelo para que venga del nodo apertura de paralelo del segundo grupo.
7. El usuario cambia el conector de entrada de uno de los nodos del segundo grupo paralelo para que venga del nodo apertura de paralelo del primer grupo.
8. El usuario clica en el botón de validar.

**Resultado esperado:**

El resultado en el paso 4 es que se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “El nodo en paralelo <nombre actividad> está conectado a un nodo Mezclador distinto.”

El resultado en el paso 8 es que se muestra en una ventana una lista de problemas y, uno de los mensajes de la lista es “El nodo en paralelo <nombre actividad> está conectado a un nodo Paralelo distinto.”

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 12 | **Código del Caso de Uso:** 03 |

**Descripción de la prueba:**

Comprobar que los cambios se han guardado correctamente y se corresponden a lo mostrado por la versión tabular del editor.

**Condiciones de ejecución:**

Debe existir un workflow en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario realiza alguna modificación al diagrama que lo mantenga en un estado válido.
3. El usuario clica en el botón de guardado.
4. El usuario recarga el workflow.

**Resultado esperado:**

El resultado en el paso 3 es que aparace un *toast* en la esquina superior derecha informando al usuario del éxito de la operación como el de la figura 47. Tras realizar el paso 4 el usuario puede comprobar que el diagrama refleja los cambios realizados.


**Figura 47 Toast que informa al usuario del éxito en el guardado**

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 13 | **Código del Caso de Uso:** 03 |

**Descripción de la prueba:**

Comprobar que, en caso de que el diagrama no sea válido, no se guardan los cambios.

**Condiciones de ejecución:**

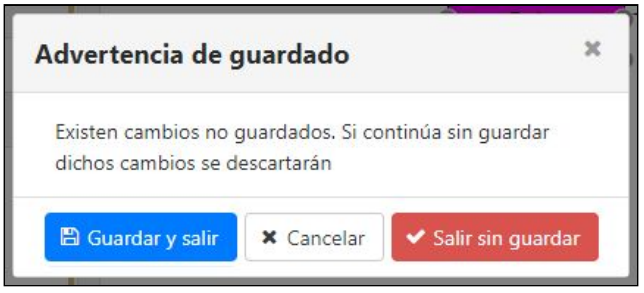
Debe existir un workflow en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

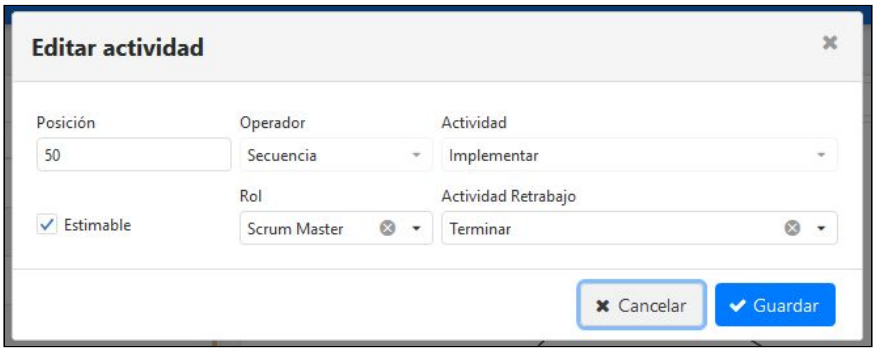
1. El usuario accede al editor de workflows.
2. El usuario realiza alguna modificación al diagrama que lo deja en un estado no válido.
3. El usuario clica en el botón de guardado.
4. El usuario recarga el workflow.

**Resultado esperado:**

El resultado en el paso 3 es que se muestra en una ventana una lista de problemas. Tras realizar el paso 4 el usuario puede comprobar que el diagrama no refleja los cambios realizados.

| Caso de prueba de aceptación | |
|---|---|
| **Código del Caso de PA:** 14 | **Código del Caso de Uso:** 03 |
| **Descripción de la prueba:**<br><br>Comprobar que si hay cambios no guardados al intentar abandonar el editor se advierte. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos. | |
| **Entrada/Pasos de ejecución:**<br><br>3. El usuario accede al editor de workflows.<br>4. El usuario modifica el workflow de alguna forma.<br>5. Sin guardar el usuario intenta abandonar el editor. | |
| **Resultado esperado:**<br><br>El resultado en el paso 5 es que aparecerá un modal como el de la figura 48 avisando al usuario de que hay cambios sin guardar.<br><br>  <br><br>**Figura 48 Advertencia de guardado** | |

| Caso de prueba de aceptación | |
|---|---|
| **Código del Caso de PA:** 15 | **Código del Caso de Uso:** 04 |
| **Descripción de la prueba:** | |

| |
|---|
| Comprobar que el nodo representa adecuadamente la actividad y opciones que el usuario ha elegido en el formulario. |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y el usuario debe ser administrador. |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario clica en botón de añadir actividad.<br>3. El usuario rellena el formulario de forma válida y clica en “Guardar.”<br>4. El usuario hace doble clic sobre el nodo recién creado. |
| **Resultado esperado:**<br><br>El resultado en el paso 3 es que aparece un nodo nuevo que representa a la actividad añadida. Tras realizar el paso 4 aparece un formulario (figura 49) idéntico al utilizado para crear la actividad y que tiene los mismo valores en los campos que los que estableció el usuario en el paso 3.<br><br>  <br>**Figura 49 Formulario de edición/creación de las actividades** |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 16 | **Código del Caso de Uso:** 04 |
| **Descripción de la prueba:**<br><br>Comprobar que no se puede eliminar ninguna actividad de tipo inicial o final. | |
| **Condiciones de ejecución:** | |

Debe existir un workflow en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario selecciona el nodo inicio.
3. El usuario pulsa la tecla suprimir.
4. El usuario selecciona el nodo final.
5. El usuario pulsa la tecla suprimir.

**Resultado esperado:**

El resultado en los pasos 3 y 5 es el mismo: aparece una toast en la esquina superior derecha como la de la figura 50.

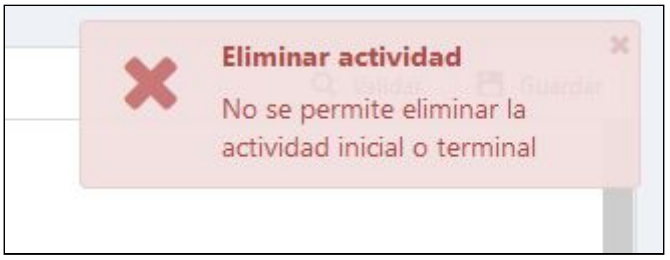



**Figura 50 Toast que notifica al usuario que no se puede eliminar un nodo inicial o final**

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 17 | **Código del Caso de Uso:** 04 |

**Descripción de la prueba:**

Comprobar que no se puede eliminar ninguna actividad que tenga alguna UT abierta.

**Condiciones de ejecución:**

Debe existir un workflow con al menos una actividad con una UT abierta en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario selecciona una de las actividades con una UT abierta.
3. El usuario pulsa la tecla suprimir.

**Resultado esperado:**

El resultado en el paso 3 es que aparece una toast en la esquina superior derecha como la de la figura 51.

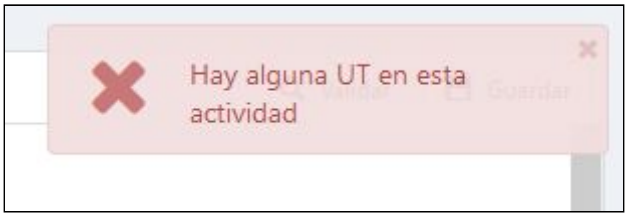



**Figura 51 Toast que notifica al usuario que la actividad tiene una UT abierta**

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 18 | **Código del Caso de Uso:** 04 |

**Descripción de la prueba:**

Comprobar que los cambios se han realizado a nivel interno del nodo.

**Condiciones de ejecución:**

Debe existir un workflow en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario hace doble clic sobre alguna de las actividades.
3. El usuario modifica el formulario de forma válida y clica en “Guardar.”
4. El usuario hace doble clic sobre la actividad que acaba de editar.

**Resultado esperado:**

Tras realizar el paso 2 aparecerá el formulario ya mostrado en la figura 49. El resultado en el paso 4 es que se abre otra vez el formulario y éste refleja los cambios realizados por el usuario en el paso 3.

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 19 | **Código del Caso de Uso:** 04 |
| **Descripción de la prueba:**<br><br>Comprobar que la apariencia del nodo se actualiza correctamente al aceptar los cambios. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow con al menos tres actividades y dos actividades iniciales con colores diferentes en la base de datos y el usuario debe ser administrador. | |
| **Entrada/Pasos de ejecución:**<br><br>1. El usuario accede al editor de workflows.<br>2. El usuario hace doble clic sobre el nodo inicio.<br>3. El usuario cambia la actividad a la otra en el formulario y clica en “Guardar.”<br>4. El usuario hace doble clic sobre el nodo de una actividad que no sea inicial o final.<br>5. El usuario cambia la posición en el formulario y clica en “Guardar.” | |
| **Resultado esperado:**<br><br>El resultado en el paso 3 es que el color y la anotación central del nodo cambian al color que representa la actividad elegida y su nombre respectivamente.<br><br>El resultado en el paso 5 es que la anotación en la parte superior izquierda cambia para reflejar el nuevo valor de la posición. | |

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 20 | **Código del Caso de Uso:** 04 |
| **Descripción de la prueba:**<br><br>Comprobar que al clicar uno de los botones aparece el nodo deseado en el diagrama. | |
| **Condiciones de ejecución:**<br><br>Debe existir un workflow en la base de datos y el usuario debe ser administrador. | |

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario hace clic en el botón “Paralelo.”
3. El usuario hace clic en el botón “Selección.”
4. El usuario hace clic en el botón “Mezclador.”

**Resultado esperado:**

El resultado en el paso 2 es que aparece un nodo de apertura de paralelo en el diagrama. De forma similar, al finalizar el paso 3 aparece un nodo de apertura de selección en el diagrama. Finalmente, tras llevar a cabo el paso 4 aparece un nodo mezclador en el diagrama.

| **Caso de prueba de aceptación** | |
|---|---|
| **Código del Caso de PA:** 21 | **Código del Caso de Uso:** 04 |

**Descripción de la prueba:**

Comprobar que las conexiones se dibujan correctamente.

**Condiciones de ejecución:**

Debe existir un workflow en la base de datos y el usuario debe ser administrador.

**Entrada/Pasos de ejecución:**

1. El usuario accede al editor de workflows.
2. El usuario clica en el botón “Conector.”
3. El usuario clica en un punto del diagrama y manteniendo pulsado arrastra el ratón hasta otro punto y suelta.

**Resultado esperado:**

Tras realizar el paso 2 el cursor del ratón cambia mientras está dentro del diagrama a una cruz similar a un signo de suma. El resultado en el paso 3 es que a medida que el usuario arrastra el ratón por el diagrama se va dibujando un conector y, al soltar el conector aparece en el diagrama.

# Anexo: Referencias


1. Integrated Definition Methods. http://www.idef.com/
2. White, S. Introduction to BPMN. IBM Corp.
3. K. Beck, Extreme Programming explained: Embrace change.Reading, Mass., Addison-Wesley, Nov16, 2004
4. Pete Deemer et al., Información básica de SCRUM.
5. Página web de Angular. https://angular.io/
6. Página web de Swimlane. https://swimlane.com/
7. Página web de yWorks. https://www.yworks.com/
8. Página web de Syncfusion. https://www.syncfusion.com/
9. ebXML Business Process Specification Schema. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-bp
10. M. A. Awad, A Comparison between Agile and Traditional Software Development Methodologies.
11. Royce, W., “Managing the Development of Large Software Systems”, Proceedings of IEEE WESCON 26, pp.1-9,1970.
12. Schwaber K., Beedle M. Agile Software Development with Scrum. Prentice Hall, Series in Agile Software Development. 2001.
13. Extreme Programing. http://www.extremeprogramming.org/
14. Página web de Google. https://www.google.com/
15. Repositorio de NGX-Graph. https://github.com/swimlane/ngx-graph
16. Documentación de NGX-Graph. https://swimlane.github.io/ngx-graph/
17. Guía para la integración de yFiles for HTML en Angular CLI. https://live.yworks.com/demos/toolkit/angular/index.html
18. Documentación de yFiles for HTML. https://docs.yworks.com/yfileshtml/#/dguide/introduction
19. API de Syncfusion Essential JS 2. https://ej2.syncfusion.com/angular/documentation/api/diagram/
20. Guía de ayuda de Syncfusion Essential JS 2. https://ej2.syncfusion.com/angular/documentation/introduction/

21. Letelier Torres, Patricio (29 de octubre de 2011). Agilismo at work. Blog. Recuperado de http://agilismoatwork.blogspot.com/2011/10/workflows-flexibles-parte-i.html

22. Workflow. (18 de mayo de 2019) Wikipedia, La enciclopedia libre. Fecha de consulta: 22:54, julio 7, 2019 from https://en.wikipedia.org/wiki/Workflow

23. Letelier Torres, Patricio (22 de marzo de 2012). Agilismo at work. Blog. Recuperado de http://agilismoatwork.blogspot.com/2012/03/agilismo-en-pocas-palabras.html

24. Letelier Torres, Patricio (28 de octubre de 2011). Agilismo at work. Blog. Recuperado de http://agilismoatwork.blogspot.com/2011/10/desafios-para-la-aplicacion-efectiva-de.html

25. David J. Anderson. (2010) Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.

26. Mary Poppendieck; Tom Poppendieck (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley Professional. ISBN 978-0-321-15078-3.

27. Ayuda de la herramienta Worki. https://cliente.tuneupprocess.com/help/web/