
Grado en Ingeniería Informática

Trabajo Final de Grado

# Propuesta e implementación de mejoras del proceso de producción de Software en una empresa

*Autora:*
Rosa Torres Hudencial

*Supervisor:*
Julián Mulet Escrig

*Tutora académica:*
Cristina Campos Sancho

Fecha de lectura: 17 de Julio de 2018
Curso académico 2017/2018

## Resumen

Se presenta el trabajo Final de Grado realizado durante la estancia en prácticas en la empresa Trading on Machine SL. Estas prácticas vienen motivadas por la necesidad que tiene esta empresa de automatizar lo máximo posible el proceso de producción de *software*. Para lograr esto, se explica el análisis y estudio realizado del proceso, identificando requisitos, necesidades y deficiencias, para así, proponer una serie de optimizaciones sobre el flujo del proceso. Luego, se describe la selección y evaluación de herramientas escogidas a integrar en el proceso, con la finalidad de lograr las mejoras identificadas. Además, se presenta la programación efectuada para la funcionalidad no cubierta por las herramientas evaluadas.

## Palabras clave

Reingeniería de procesos, *Scrum*, Control de versiones, Taiga.

## Keywords

Process Reengineering, *Scrum*, Version Control, Taiga.

# Índice general

# Índice de tablas

# Índice de figuras

# Capítulo 1

# Introducción

## 1.1. Contexto y motivación del proyecto

El proyecto propuesto se realiza en el marco de las prácticas para el proyecto Final de Grado de Ingeniería Informática. La motivación de estas prácticas viene dada por la posibilidad de aplicar los conocimientos adquiridos durante el grado, así como poner en práctica conceptos más específicos del itinerario Sistemas de Información.

La empresa para la cual se ha realizado las prácticas, está desarrollando un sistema para generar estrategias financieras, el cual le permitirá tomar decisiones sobre inversiones y operar en el mercado financiero. Las características del desarrollo de este sistema, provocó la necesidad de mejorar el proceso de gestión de las tareas que participan en la producción de *software*, mediante la automatización de las herramientas que se utilizan en el proceso. El estudio de cómo sucede el proceso, la propuesta de mejoras, así como la selección, evaluación e implementación de herramientas a incorporar para lograr optimizar el proceso, resulta de gran interés, ya que da la oportunidad de realizar el estudio y mejora de un proceso en un entorno real.

## 1.2. Objetivos del proyecto

El objetivo de este proyecto es proponer e implementar mejoras al proceso actual de producción de *software* de la empresa Trading on Machine SL (con nombre comercial Trading by Machine o TbM). Para lograr esto, la empresa ha propuesto automatizar, en la medida de lo posible, las tareas que participan en el proceso de desarrollo de *software*.

El proceso deberá ser estudiado y analizado, para así lograr entender cómo se realiza. Además, se identificarán los roles implicados y las herramientas utilizadas. Una vez estudiado el proceso, se podrán identificar necesidades y deficiencias, y se propondrán una serie de mejoras para cada caso. A continuación, se deberán recoger los requisitos que debe cumplir el proceso mejorado, y teniendo en cuenta estos requisitos, seleccionar, evaluar e implementar las herramientas necesarias que permitan optimizar el proceso de desarrollo de *software*.

## 1.3. Estructura de la memoria

La estructura de esta memoria comienza exponiendo, la motivación y objetivos, que impulsaron la elección del proyecto para la realización de las prácticas. A continuación, se describe la empresa donde se ha realizado la estancia, el alcance del proyecto, el proceso de producción de *software* en la empresa TbM y, se hace mención a las herramientas y metodologías empleadas para estudiar y mejorar el proceso.

A continuación, se expone cómo se planificó temporalmente el proyecto, el cálculo de estimación de costes, en el cual se hace una valoración considerando los gastos de personal y de herramientas y, el seguimiento del proyecto efectuado.

Después se llega a la parte central de este documento, en la cual se describe con la ayuda de un lenguaje de notación gráfica, cómo sucede el proceso de producción de *software* para el desarrollo de un sistema en la empresa TbM. En las secciones siguientes, se explican los problemas y necesidades identificados en el proceso, la mejora propuesta para optimizar dicho proceso y, la selección, evaluación e implementación de herramientas y funcionalidades, llevadas a cabo, con el objetivo de mejorar el proceso estudiado.

Por último, se exponen las pruebas realizadas sobre la nueva funcionalidad implementada, se exponen impresiones sobre los resultados obtenidos y se realiza una conclusión.

# Capítulo 2

# Descripción del proyecto

El sistema, TbM System, que está desarrollando la empresa Trading by Machine, está destinado al análisis y a las operaciones sobre finanzas. Debido a esto, la empresa ha detectado la necesidad de automatizar acciones y usar herramientas robustas para el desarrollo de su sistema. De esta manera, la empresa TbM, espera cumplir con ciertas normas de seguridad y calidad para poder pasar futuras auditorías, así como optimiar los tiempos de entrega de las tareas, mejorar la resolución de incidencias y obtener un flujo de desarrollo más ágil y escalable. Es en este trabajo de mejora, mediante la automatización de acciones y el estudio, selección y evaluación de herramienas que se adapten al desarrollo de *software* de la empresa TbM, donde se ha propuesto el proyecto.

## 2.1. Descripción de la empresa

La empresa donde se realiza la estancia en prácticas es Trading By Machine (TbM), se encuentra ubicada en Valencia y cuenta con una sede en Castellón, en el edificio Espaitec 2 situado dentro del Campus del Riu Sec. Es en esta sede donde se realiza la estancia en prácticas.

TbM se dedica a ofrecer servicios financieros mediante el uso de nuevas tecnologías, por lo que se define a sí misma, como una empresa de carácter *Fintech*. En la actualidad, está desarrollando su propio sistema de *trading*, llamado TbM System, el cual consistirá en una plataforma que ayudará a tomar decisiones sobre estrategias financieras y, a realizar, operaciones a través de internet. TbM System recogerá datos de tipo financiero, efectuará análisis sobre dichos datos y los transformará en información, de manera que se pueda permitir a inversores, tomar decisiones sobre finanzas a través de la generación de estrategias de inversión. Este sistema, también permitirá operar en el mercado: se podrán realizar operaciones de compra y venta de activos financieros. Se trata pues, en su mayoría, de un Sistema de Soporte a la toma de Decisiones (DSS) [1], ya que se encargará de proporcionar información, con el objetivo de facilitar decisiones frente a operaciones financieras.

Además del sistema TbM System, la empresa también prevé crear una red social colaborativa con el objetivo de compartir recursos, experiencias y asesorar a posibles clientes que quieran

adquirir activos financieros a través de ellos.

## 2.2. Alcance del proyecto

El alcance de este proyecto incluye el estudio del proceso de desarrollo de *software* en su estado antes de la mejora, la identificación de deficiencias y necesidades para, a continuación, ofrecer una serie de optimizaciones y realizar la propuesta del proceso mejorado. A continuación, se seleccionarán y se evaluarán posibles herramientas existentes en el mercado como solución a las deficiencias anteriormente identificadas, o incluso si es necesario, se propondrá una solución a medida. Por último, se deberá implementar la solución propuesta.

## 2.3. Proceso de desarrollo de *software* en la empresa

La empresa TbM realiza el desarrollo de *software* para su sistema mediante metodología *Scrum*. Se trata de una metodología en la cual se realizan entregas parciales y regulares en las que se revisa el desarrollo del producto [2], [3]. A través de estas entregas parciales, consiguen obtener cada cierto período de tiempo, un incremento del producto, el TbM System. Esta forma de organizar el trabajo en períodos de tiempo, son los denominados *Sprints* [4]. Cada *Sprint* se define en una reunión en la que participa, la persona responsable de que se sigan las prácticas descritas en la metodología *Scrum* (*Scrum Master*), el miembro del equipo encargado de definir y dirigir los objetivos del proyecto (*Product Owner*) y, los desarrolladores.

La reunión se divide en dos partes. En la primera parte se decide qué tareas se llevarán a cabo en el *Sprint*. Para esto, se tiene en cuenta la lista de prioridades de los requisitos del usuario. Esta lista, llamada *Product Backlog*, puede verse representada al principio de la Figura 2.1. Cada requisito se define en pocas frases utilizando lenguaje común, creando de esta forma, las historias de usuario [5]. Además, se establece a qué miembro del equipo de desarrollo pertenece cada historia de usuario, y por último, se define la duración del *Sprint*, normalmente de diez días. Después se realiza una pausa en la reunión, en la que los desarrolladores descomponen las historias de usuario que les pertenecen en tareas, e indican la duración en el tiempo para cada una de ellas, obteniendo así, la lista de tareas a realizar durante el *Sprint*. Esta lista de tareas, llamada *Sprint Backlog* [6], puede verse también en la Figura 2.1 precedida por el *Product Backlog*. Una vez que se han descompuesto las historias de usuario en tareas, se continúa con la segunda parte de la reunión, en la que se revisan las tareas creadas entre todos los miembros y se finaliza la reunión, dando lugar al comienzo del *Sprint*. Durante la duración del *Sprint*, tal y como puede verse en la Figura 2.1, se realiza un seguimiento en reuniones diarias, *Scrum* diario, de pocos minutos.

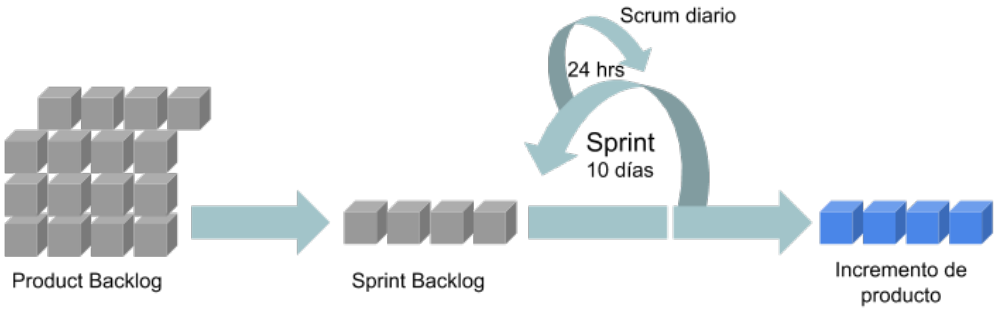



Figura 2.1: Proceso de desarrollo con *Scrum*.
*Fuente: Elaboración propia.*

Cuando se finaliza el período del *Sprint*, en el cual se han implementado las funcionalidades descritas en las tareas en el sistema en desarrollo, se obtiene un incremento del producto. Puede verse este incremento representado al final de la Figura 2.1.

## 2.4. Metodología empleada en el proyecto

El proyecto propuesto consiste en mejorar un proceso, la producción de *software*, realizado por la empresa Trading by Machine, para desarrollar su sistema TbM System. Para la realización de las actividades que han permitido obtener este trabajo final, se han adaptado las fases de un proyecto de Reingeniería de Procesos de Negocio (*Business Process Reeingeniering*, BPR), ya que el BPR consiste en el análisis y rediseño de procesos de negocio, para alcanzar mejoras en medidas críticas y contemporáneas de rendimiento, tales como costes, calidad, servicio y rapidez [7]. Las fases son [8]:

1. Tener el equipo de reingeniería.
2. Selección del proceso a rediseñar.
3. Diagnóstico de procesos/comprender los procesos existentes.
4. Rediseño del proceso.
5. Prototipos - Diseñar y construir un prototipo del nuevo proceso.
6. Implementación a gran escala.

Aún así, se debe tener en cuenta que debido a las propias condiciones del proyecto se ha prescindido de los puntos: 1. Tener el equipo de reingeniería, ya que no se ha llevado a cabo la selección de un equipo para realizar este proyecto, y el punto 5. Prototipos - Diseñar y construir un prototipo del nuevo proceso, debido a que este proyecto tiene un alcance e implementación reducido. Además, el punto 6. Implementación a gran escala, se debe entender como sólo *implementación* debido a que no se ha tratado de una implementación compleja a gran escala. Finalmente, las fases realizadas han sido:

1. Selección del proceso a rediseñar. Ha consistido en el estudio y mejora del proceso de desarrollo de *software* realizado por la empresa TbM.

2. Diagnóstico de procesos/comprender los procesos existentes. Para ello se ha estudiado el proceso en el estado antes de la mejora, se han detectado y analizado los problemas presentes y se ha representado el proceso.

3. Rediseño del proceso. Se ha propuesto las mejoras para los problemas hallados, y se ha realizado la representación del proceso mejorado.

4. Implementación. Se han adaptado e implementado las herramientas necesarias para lograr las mejoras previamente propuestas.

Para representar y optimizar el flujo de trabajo del proceso, se ha utilizado la Gestión de Procesos de Negocio *(Business Process Management*, BPM). Según la definición oficial en [9], «es una disciplina que implica cualquier combinación de modelado, automatización, ejecución, control, medición y optimización de flujos de actividades de negocios, en apoyo de objetivos empresariales, sistemas integrales, empleados, clientes y socios dentro y más de los límites de la empresa.»

El lenguaje de notación empleado, incluido en el estándar del BPM y que permite modelar procesos de negocio en un formato de flujo de trabajo, ha sido el Modelo y Notación de Procesos de Negocio (*Business Process Model and Notation, BPMN*). La versión utilizada de BPMN en este proyecto es la 2.0.

También se debe tener en cuenta que, la gestión del trabajo y el desarrollo de las mejoras se ha realizado según la metodología *Scrum*, a fin de seguir el mismo modo de trabajo de la empresa.

## 2.5. Herramientas empleadas en el proyecto

Se deben distinguir, las herramientas empleadas para la realización del proyecto, de las herramientas utilizadas en el proceso de producción de *software* de la empresa TbM. Las herramientas empleadas en la realización del proyecto han sido:

- El *Integrated Development Environment* IDE Pycharm, para la programación de la nueva funcionalidad.
- El panel de *Scrum* Taiga para gestionar las tareas durante el proyecto.
- Git, *software* de control de versiones para llevar un control de la implementación realizada.
- Aplicación web Bitbucket, para almacenar el código de la implementación realizada en remoto.
- Modelización del proceso con BPMN, a través de la aplicación *online Camunda Modeler* [10].

- Elaboración de las matrices de evaluación con la aplicación Hojas de Cálculo de Google.

Se exponen a continuación las herramientas seleccionadas, adaptadas e integradas en el proceso mejorado. La motivación de escoger estas herramientas se explica en el capítulo 5. *Implementación de las mejoras del proceso de producción de software*, en las secciones 5.1.3. y 5.2.3.*Selección, evaluación e implementación.*

- El IDE Pycharm, para el desarrollo de TbM System.
- El panel de *Scrum* Taiga para gestionar las tareas que realiza la empresa para el desarrollo de su sistema.
- Git, para el control de versiones de su sistema.
- Conjunto de extensiones de comandos Git, Git-flow.
- Aplicación web Bitbucket, para almacenar el código de TbM System en remoto.

# Capítulo 3

# Planificación del proyecto

## 3.1. Planificación

Inicialmente la planificación del proyecto se realizó mediante un diagrama de Gantt, el cual puede verse en la Tabla 3.1. En dicho diagrama consta la fecha de inicio de las prácticas, el 29 de enero del 2018, y la fecha de finalización, el 23 de mayo del mismo año. Se incluyeron las tareas que inicialmente se consideraron necesarias para llevar a cabo el proyecto según la limitación del tiempo de las prácticas, 300 horas.

Sin embargo, esta planificación inicial, a proposición de la empresa, se cambió para desarrollar el proyecto según la metodología *Scrum*; de esta forma se adaptó la planificación a la forma de trabajar de la empresa. En la Tabla 3.2 se resumen los *Sprints* llevados a cabo para realizar este proyecto. El contenido completo de los *Sprints* puede verse en el Anexo A. *Sprints planificados para la realización del proyecto.*

| Id | Nº | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|---|
| 1 | **1** | **Proyecto** | **300 horas** | **30/01/18** | **23/05/18** | |
| 2 | **1.1** | **Documentar y planificar el proyecto** | **20 horas** | **30/01/18** | **02/02/18** | |
| 3 | 1.1.1 | Revisar el contexto y buscar información | 10 horas | 30/01/18 | 31/01/18 | |
| 4 | 1.1.2 | Identificar alcance y objetivos | 10 horas | 01/02/18 | 02/02/18 | 3 |
| 5 | **1.2** | **Planificar el proyecto** | **25 horas** | **05/02/18** | **09/02/18** | |
| 6 | 1.2.1 | Definir tareas | 10 horas | 05/02/18 | 06/02/18 | |
| 7 | 1.2.2 | Crear el diagrama de Gantt | 5 horas | 07/02/18 | 07/02/18 | 6 |
| 8 | 1.2.3 | Documentar la propuesta del proyecto | 10 horas | 08/02/18 | 09/02/18 | 7 |
| 9 | 1.2.4 | Entregar la propuesta del proyecto | 0 horas | 09/02/18 | 09/02/18 | |
| 10 | **1.3** | **Desarrollo técnico del proyecto** | **15 horas** | **12/02/18** | **14/02/18** | |
| 11 | 1.3.1 | Definir requisitos del proyecto | 15 horas | 12/02/18 | 14/02/18 | |
| 12 | 1.3.1.1 | Crear diagrama de Casos de uso | 5 horas | 12/02/18 | 12/02/18 | |
| 13 | 1.3.1.2 | Definición de requisitos de alto nivel | 10 horas | 13/02/18 | 14/02/18 | 12 |
| 14 | **1.4** | **Estudio y propuesta de mejoras del flujo de trabajo** | **85 horas** | **15/02/18** | **04/04/18** | |
| 15 | 1.4.1 | Modelización de la situación actual | 15 horas | 15/02/18 | 19/02/18 | |
| 16 | 1.4.2 | Propuesta de mejoras | 25 horas | 20/02/18 | 13/03/18 | 15 |
| 17 | 1.4.3 | Análisis de las necesidades de adaptación | 25 horas | 14/03/18 | 21/03/18 | 16 |
| 18 | 1.4.4 | Evaluación y viabilidad | 20 horas | 22/03/18 | 04/04/18 | 17 |
| 19 | **1.5** | **Evaluación y selección de herramientas** | **35 horas** | **05/04/18** | **13/04/18** | |
| 20 | 1.5.1 | Preselección de herramientas a evaluar | 10 horas | 05/04/18 | 06/04/18 | |
| 21 | 1.5.2 | Evaluación de las herramientas | 15 horas | 09/04/18 | 11/04/18 | 20 |
| 22 | 1.5.3 | Selección de la propuesta | 10 horas | 12/04/18 | 13/04/18 | 21;20 |
| 23 | **1.6** | **Implementación** | **120 horas** | **16/04/18** | **23/05/18** | |
| 24 | 1.6.1 | Adaptación de la funcionalidad | 80 horas | 16/04/18 | 11/05/18 | |
| 25 | 1.6.2 | Integración y pruebas | 15 horas | 14/05/18 | 16/05/18 | 24 |
| 26 | 1.6.3 | Formación usuarios | 10 horas | 17/05/18 | 18/05/18 | 25 |
| 27 | 1.6.4 | Documentación | 15 horas | 21/05/18 | 23/05/18 | 24;25;26 |
| 28 | 1.6.5 | Entrega final | 0 horas | 23/05/18 | 23/05/18 | 27 |

Tabla 3.1: Diagrama de Gantt.

Tabla 3.2: *Sprints* llevados a cabo para realizar el proyecto.

| | |
|---|---|
| **Sprint 1** | Familiarizarse con el flujo de trabajo. |
| **Período** | 30 enero - 9 febrero. |
| **Sprint 2** | Evaluar herramientas para el flujo de trabajo. |
| **Período** | 12 febrero - 22 febrero. |
| **Sprint 3** | Estudiar herramientas para integrar el canal de comunicación para el flujo de trabajo, estudio de *Git-flow* para control de versiones. |
| **Período** | 6 marzo - 23 marzo. |
| **Sprint 4** | Realizar matrices de evaluación para las herramientas a incorporar al flujo de trabajo, y realizar pruebas con los *plugins* de Pycharm-GitFlow y Pycharm-Taiga. |
| **Período** | 26 marzo - 13 abril. |
| **Sprint 5** | Automatizar flujo diario de Git. |
| **Período** | 16 abril - 27 abril. |
| **Sprint 6** | Terminar flujo Git TbM. |
| **Período** | 30 abril - 11 mayo. |
| **Sprint 7** | Dar soporte de Taiga para el *branching*. |
| **Período** | 14 mayo - 23 mayo. |

## 3.2. Estimación de costes del proyecto

Se presenta a continuación la estimación de costes del proyecto. Esta estimación se calcula teniendo en cuenta el coste humano y las herramientas empleadas. Como para la realización del proyecto se han destinado dos personas, la estudiante y el supervisor, será este número el que se tenga en cuenta al calcular el coste humano, considerando una persona para realizar el trabajo, programadora junior y, otra para la supervisión del proyecto. Además, para el cálculo de las herramientas empleadas, sólo se tendrán en cuenta las que tengan licencia de pago.

- Coste humano:
  - Programadora junior: 1200 € mes a 40 horas semana.
  - Supervisor del proyecto: 2000 € mes a 40 horas semana.
- Coste de licencias:
  - Licencia IDE Pycharm: 649 € anual.
- Coste equipo informático:
  - Equipo: 1200 €. Se debe tener en cuenta que se amortiza en 4 años.

Dado que las prácticas tienen una duración de 300 horas realizadas en 12 semanas, se calculan las cantidades correspondientes para los costes de personal, licencias de *software* y equipo informático:

**Coste humano:**

Programadora junior:

$$Horas\ mes = 40\ horas\ *\ 4\ semanas = 160\ horas/mes \tag{3.1}$$

$$Precio\ hora = [1200\ euros/mes]\ /\ [Horas\ mes] = 7,5\ euros/hora \tag{3.2}$$

Teniendo en cuenta las 300 horas de duración de las prácticas:

$$Coste\ programadora = 300\ horas\ *\ [Precio\ hora] = 2250\ euros \tag{3.3}$$

Supervisor del proyecto:

$$[Horas\ mes] = Valor\ de\ (3.1) \tag{3.4}$$

$$Precio\ hora = [2000\ euros/mes]\ /\ [Horas\ mes] = 12,5\ euros/hora \tag{3.5}$$

Sabiendo que dedica 8 horas semanales al proyecto, y que la duración de las prácticas fue de 12 semanas:

$$Horas\ dedicación = 8\ horas/semana\ *\ 12\ semanas = 96\ horas \tag{3.6}$$

$$Coste\ supervisor = [Horas\ dedicación]\ *\ [Precio\ hora] = 1200\ euros \tag{3.7}$$

**Coste de licencias de *software*:**

$$Precio\ mes = [649\ euros/anuales]\ *\ [1\ año\ /\ 12\ meses] \simeq 54,08\ euros/mes \tag{3.8}$$

$$Meses\ trabajo = [12\ semanas]\ *\ [1\ mes\ /\ 4\ semanas] = 3\ meses \tag{3.9}$$

$$Coste\ licencias = [Precio\ mes]\ *\ [Meses\ trabajo] = 162,24\ euros \tag{3.10}$$

**Coste del equipo informático:**

Teniendo en cuenta que un equipo informático se amortiza en 4 años:

$$Precio\ mes = [1200\ euros]\ *\ [1\ año\ /\ 12\ meses]\ /\ [4\ años] = 25\ euros/mes \tag{3.11}$$

$$Meses\ trabajo = Valor\ de\ (3.9) \tag{3.12}$$

$$Coste\ equipo\ informático = [Precio\ mes]\ *\ [Meses\ trabajo] = 75\ euros/mes \tag{3.13}$$

**Total costes:**

Tabla 3.3: Cálculo total de costes para el proyecto

| **Costes** | **Precios** |
|---|---|
| **Programadora** | 2250.00 € |
| **Supervisor** | 1200.00 € |
| **Licencias** | 162.24 € |
| **Equipo informático** | 75.00 € |
| **Total** | 3687.24 € |

Tal y como puede verse en la Tabla 3.3, se han obtenido los siguientes costes de personal: Programadora junior 2250.00 € y Supervisor del proyecto 1200.00 €, sumando un coste humano de 3450.00 €. Las licencias tienen un coste de 162.24 € y el equipo informático de 75.00 €. El total de costes asciende a 3687.24 €.

## 3.3. Seguimiento del proyecto

El proyecto se ha desarrollado a lo largo de siete *Sprints*. El tiempo de realización para cada *Sprint* de forma general fue de diez días, sin embargo, algunos tuvieron un período mayor debido al tiempo necesario para realizar las tareas.

En la Tabla 3.4 se muestra qué tareas se han realizado en cada *Sprint*. Las tareas marcadas en color azul son tareas realizadas parcialmente durante el *Sprint*, mientras que las tareas de color verde son las completadas. Puede apreciarse cómo algunas tareas abarcaron más de un *Sprint*. Por ejemplo, en el *Sprint* 2, se aprecia cómo la mayoría de las tareas iniciadas en el *Sprint* 1, se terminaron en este período. Esto fue debido, a la necesidad de comprender el flujo del proceso de producción de *software*, del cual fue necesario realizar varias revisiones del análisis. Lo mismo sucedió con la identificación de problemas, ya que para reconocer varios de los inconvenientes, se requirió observar y reproducir cómo se realiza el trabajo varias veces. Por último, se inició la descripción de las mejoras, la recogida de requisitos generales y la selección de herramientas a evaluar; tareas que se finalizaron en el *Sprint* 3.

Tabla 3.4: Evolución de las tareas a lo largo de los *Sprints*.

| Tareas / Sprint | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 |
|---|---|---|---|---|---|---|---|
| Análisis del proceso | | | | | | | |
| Modelización de la situación actual AS-IS | | | | | | | |
| Diagnóstico e identificación de problemas | | | | | | | |
| Rediseño del proceso TO-BE | | | | | | | |
| Descripción de las mejoras para el proceso | | | | | | | |
| Requisitos generales | | | | | | | |
| Selección de las herramientas a evaluar | | | | | | | |
| Evaluación de las herramientas | | | | | | | |
| Selección de la propuesta | | | | | | | |
| Implementación y adaptación de la propuesta | | | | | | | |
| Pruebas sobre el flujo de trabajo | | | | | | | |

| | |
|---|---|
| | Realización parcial |
| | Tarea completada |

# Capítulo 4

# BPR aplicado al proceso de producción de *software*

El proceso de producción de *software* se describe en las Tablas 4.1 y 4.2, en las que se especifican las actividades y tareas, el personal, los recursos y las reglas que se deben cumplir.

Tabla 4.1: Descripción del proceso de producción de *software*

| | |
|---|---|
| **Proceso** | Producción de *software*. |
| **Propósito** | Control y producción de código para crear el sistema. |
| **Actividades y tareas** | ▪ Crear *Sprints*:<br>• Definir cada *Sprint*.<br>• Crear las historias de usuario asociadas al *Sprint*.<br>• Crear las tareas para cada historia de usuario.<br>• Crear incidencias.<br>▪ Desarrollo dirigido por test (TDD):<br>• Crear las pruebas automatizadas.<br>• Producir el código.<br>• Comprobar el código a través de las pruebas.<br>▪ Control de reposiciones mediante Git:<br>• Gestionar las ramas.<br>• Actualizar el repositorio en local.<br>• Actualizar el repositorio en remoto.<br>▪ Notificar incidencias mediante Slack. |
| **Personal** | *Scrum Master*.<br>*Product Owner*.<br>Desarrolladores. |

Tabla 4.2: Continuación de la Tabla 4.1 Descripción del proceso de producción de *software*

| | |
|---|---|
| **Recursos necesarios** | Sistema operativo Linux, distribución Ubuntu 16.04.<br>Panel de *Scrum* Taiga.<br>IDE Pycharm.<br>Bitbucket para gestionar y alojar el repositorio en remoto.<br>Plugin para la conexión entre Pycharm y Taiga.<br>Plugin para la conexión entre Pycharm y Bitbucket.<br>Canal de comunicación para incidencias, Slack.<br>Git, para el control de versiones. |
| **Reglas** | Cada desarrollador debe crear sus tareas por cada historia de usuario que le pertenezca.<br>Al cerrar una tarea se deben pasar los test de forma automática, y en caso de ser satisfactorios, actualizar el repositorio del proyecto en remoto.<br>Un desarrollador no podrá dar por finalizada una tarea ni subirla a remoto hasta que se pasen los test de forma satisfactoria.<br>El desarrollador se debe encargar de la gestión del control de versiones con Git al producir código. |

## 4.1. Modelización con BPMN2 de la situación actual (*AS IS*)

El proceso de producción de *software* realizado en la empresa TbM para desarrollar el sistema TbM System, se ha representado *AS IS* en la Figura 4.1. Para obtener esta representación gráfica del proceso se ha empleado la notación BPMN2. La descripción de los elementos utilizados puede encontrarse en la Tabla B.1 del Anexo B. *Elementos de la notación BPMN2 usados para la representación del proceso*. También se proporciona en el Anexo B, ampliaciones de partes de la Figura 4.1 en la Figura B.1 y Figura B.2 para mejorar su lectura.

Se han identificado cuatro áreas o *pools*. Cada *pool* agrupa las tareas que lleva a cabo un participante, por este motivo se utiliza para organizar el estudio del proceso de producción de *software* en la empresa TbM. Estos *pools* son: *Scrum Master*, *Product Owner*, *Developer Team* y *Remote Repository*.

### 4.1.1. *Pool Scrum Master*

En este *pool* actúa el *Scrum Master*. Las tareas que realiza son, recibir aviso de la realización de las historias de usuario procedente del *pool Product Owner*, para a continuación revisarlas y dar una retroalimentación. La comunicación con el *pool Product Owner* se ha representado mediante unas flechas discontinuas. No se detallan dentro del *pool* el resto de actividades realizadas por el *Scrum Master* debido a que los únicos eventos de interés para el proceso son recibir y enviar mensajes al *pool Developer Team*.

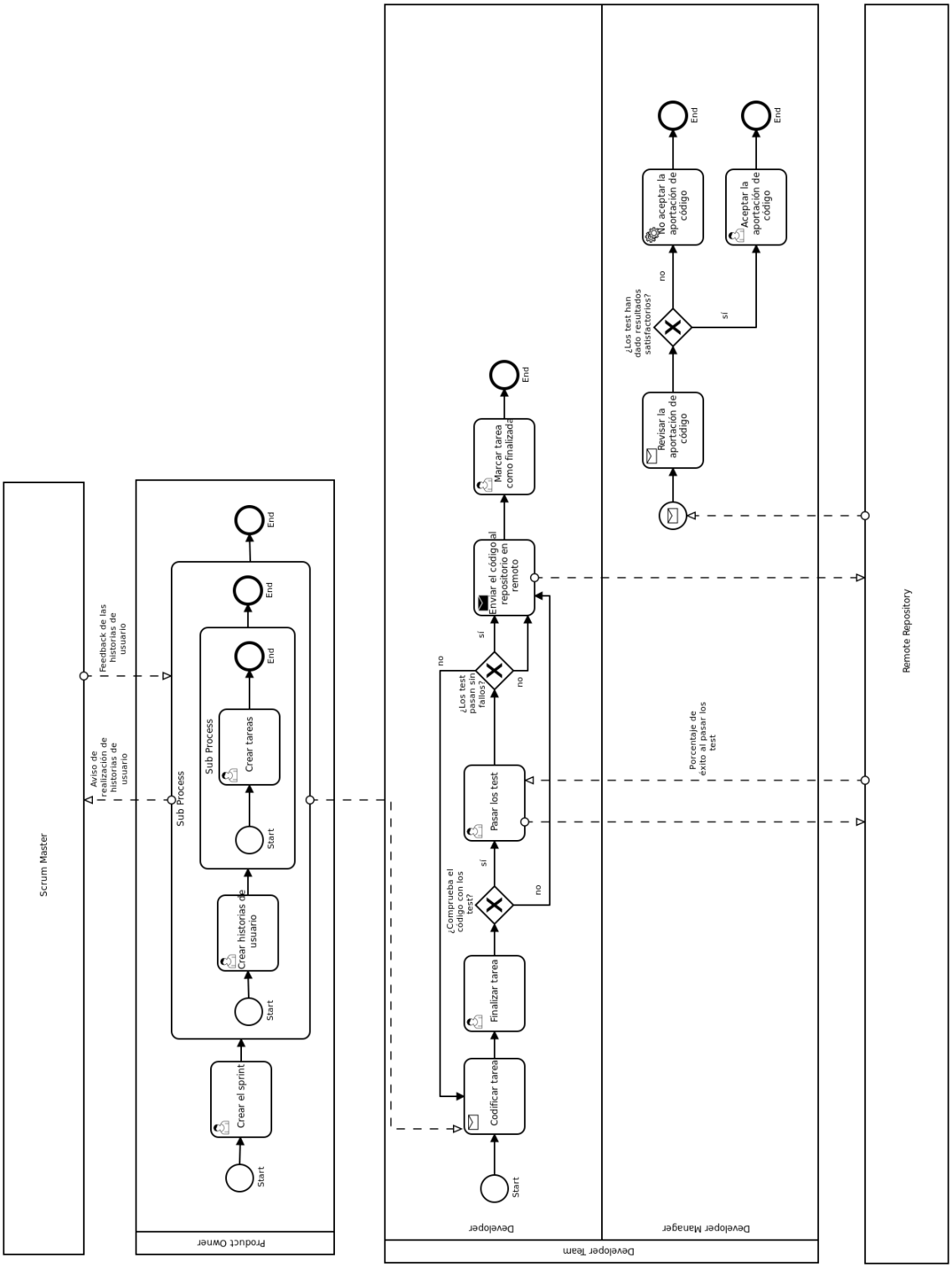



Figura 4.1: Modelización *AS IS* con BPMN2.

### 4.1.2. *Pool Product Owner*

En este *pool* se lleva a cabo la creación de los *Sprints*. El flujo comienza indicado por un círculo y la palabra *Start*. La creación del *Sprint* conlleva la realización de un subproceso, en el cual se crean las historias de usuario, para a su vez, realizar otro subproceso, en el que se crean las tareas a partir de las historias de usuario. Las actividades realizadas en este *pool* son tareas que realiza un usuario (*User task*) con la ayuda de una aplicación web: el panel de *Scrum* Taiga. El subproceso en el cual se crean las historias de usuario, se comunica con el *pool Scrum Master* y con el *pool Developer Team*. En concreto, en el *pool Developer Team* la comunicación se realiza con los desarrolladores, representados por un carril (*lane*) en el *pool* llamado *Developer*.

### 4.1.3. *Pool Developer Team*

Este *pool*, se puede ver en la Figura 4.1, como está dividido en dos partes: *Lane Developer* y *Lane Developer Manager*. Cada una de estas áreas, es el elemento del BPMN llamado, *Lane*. Este elemento se utiliza para mostrar tareas de un *pool* que fueron asignadas a personas particulares, en este caso, los desarrolladores y el responsable del equipo de desarrolladores [11].

***Lane Developer:***

Cuando el mensaje procedente del subproceso del *pool Product Owner* llega al *lane Developer*, da comienzo la actividad de codificar. Una vez realizada por el desarrollador, continúa el flujo con la finalización de la codificación de las tareas, *Finalizar tarea*, y a continuación se toma la decisión de comprobar el código con los test. Si el desarrollador decide efectuar las pruebas, realiza los test y obtiene un porcentaje de éxito. El nivel de porcentaje de éxito es decisivo para incluir el código en el repositorio en remoto. Aún así, dado que pasar los test y enviar el código a remoto depende de la decisión tomada por el desarrollador, también es posible enviar el código a remoto, sin llevar a cabo los test, o incluso sin que pasen satisfactoriamente. Esto puede verse representado mediante los nodos de decisión *¿Comprueba el código con los test?* y *¿Los test pasan sin fallos?* Una vez enviado el código a remoto el desarrollador concluye la tarea, y el flujo del proceso en este *lane* finaliza.

***Lane Developer Manager:***

En este *lane* actúa el desarrollador con papel de *Developer Manager*. El proceso da comienzo al recibir un mensaje del *pool Repository Remote*, entonces el *Developer Manager* efectúa la revisión de la aportación de código en la cual se evalúa si el nuevo código supera los test. Si las pruebas pasan satisfactoriamente, se realiza la siguiente actividad, *Aceptar la aportación del código* y finaliza el proceso. En caso de ser insatisfactorios, de forma automática el código se rechaza y el proceso también finaliza.

### 4.1.4. *Pool Remote Repository*

La actividad de pasar los test en el *lane Developer* conlleva el enviar el nuevo código al *pool Remote Repository*, del cual se obtiene el porcentaje de éxito al efectuar las pruebas sobre el

código. Posteriormente, cuando el flujo llega a la actividad de *Enviar el código al repositorio en remoto*, el código es enviado nuevamente a este *pool*. A continuación, el *pool Remote Repository* manda un mensaje al *lane Developer Manager* para dar comienzo a la revisión de la aportación de código.

## 4.2. Diagnóstico e identificación de problemas

Una vez estudiado y modelado el proceso de control de producción de *software*, se han podido identificar problemas y necesidades. Siguiendo la representación BPMN de la Figura 4.1, los problemas encontrados son:

### 4.2.1. *Pool Product Owner*

**No se diferencian las tareas de *tipo problema***. Se han identificado dos tipos de tareas, unas son las creadas a partir de las historias de usuario, y otras son las creadas a partir de problemas detectados en el código. Para estas tareas de tipo problema se ha observado que sería necesario distinguir a qué tipo de problema se refieren.

**Las tareas no tienen *prioridad***. En la actualidad, además del problema explicado en el punto anterior, una tarea que se deba realizar para solucionar un problema no posee un nivel de importancia o prioridad. Esto provoca no disponer de un criterio para anteponer unas tareas a otras.

**Las tareas no tienen *severidad***. Sería necesario identificar las tareas para solucionar problemas según el grado de impacto del defecto en el proyecto o severidad. En la actualidad no se dispone de esta información, y no se tiene un criterio definido para escoger la tarea a realizar.

### 4.2.2. *Pool Developer Team*

**No es posible trabajar sobre una tarea de tipo problema.** En la actualidad el IDE Pycharm dispone de funcionalidad, aunque limitada, para ofrecer conexión entre Taiga y Pycharm. Esta conexión sólo tiene capacidad para mostrar tareas normales. El panel de *Scrum* Taiga dispone de una sección con un listado para introducir las tareas de tipo problema, y estas tareas no se muestran en el IDE para trabajar sobre ellas. Además Pycharm, dispone de la funcionalidad básica para realizar el control de versiones sobre el código, y al crear una tarea de forma manual, conlleva, realizar manualmente la acción necesaria para obtener el seguimiento del control de versiones que, normalmente realizaría el IDE.

**El cambio de estado de una tarea a *closed* es manual.** Las tareas presentes en el panel de *Scrum* y que se utilizan para añadir funcionalidad al sistema en desarrollo, pueden adquirir tres tipos de estados a medida que se realizan:

- *new*, para las tareas nuevas sin empezar.
- *in progress*, para las que se están realizando.
- *closed*, para las tareas finalizadas.

Cuando se finaliza una tarea, no se cambia el estado de forma automática, sino que el desarrollador debe dirigirse al panel de *Scrum* y cambiar el estado a *closed*. Esto provoca que el estado real de la tarea dependa de la actuación del desarrollador.

**No se filtran las tareas por historia de usuario.** Se muestran en el IDE todas las tareas mediante una lista. No existe filtro alguno para mostrar las tareas, y tampoco se sabe a qué historia de usuario pertenece. Cuando un desarrollador debe realizar las tareas pertenecientes a una historia de usuario, debe dirigirse primero al panel de *Scrum* y visualizarlas, para a continuación, volver al IDE y buscar cada tarea en el listado.

**Realización manual de los test**. Se deben pasar los test antes de enviar nuevo código al repositorio en remoto, pero esta acción es manual. Esto implica que un desarrollador pueda enviar el código al repositorio del proyecto, sin haber realizado los test, o habiéndolos pasado, pero sin obtener el nivel mínimo de cobertura establecido como satisfactorio.

**Control inadecuado de versiones**. El uso actual del control de versiones, si bien proporciona un mínimo control del proyecto, este resulta insuficiente e inadecuado para trabajar en equipo.

El control de versiones, resulta de gran importancia, ya que se utiliza para gestionar los cambios sobre el sistema en desarrollo TbM System. Esto quiere decir, que en un momento determinado del desarrollo o modificación del sistema, este se encuentra en un estado definido por una versión, revisión o edición [12].

El control de versiones utilizado a través del IDE Pycharm, es el *software* Git. Este *software*, aunque existan una gran variedad de interfaces de usuario, se utiliza principalmente por línea de comandos, ya que así se dispone de todas las funcionalidades. En la empresa TbM, este *software* se usa principalmente a través del IDE Pycharm, y la línea de comandos. La diferencia de Git frente a otros sistemas de versiones, es que este, en lugar de mantener una lista con los cambios, los almacena como un conjunto de instantáneas [13].

Para lograr un entendimiento de cómo se trabaja con Git en la empresa TbM, es necesario explicar antes, qué es una rama, (*branch*). En el ámbito de Git, una rama es un apuntador móvil apuntando a una confirmación. La rama por defecto creada con la primera confirmación (al iniciar por primera vez Git en un proyecto) es la rama *master*. Puede verse esta rama en la Figura 4.2. Al realizar cada confirmación la rama irá avanzando automáticamente [14].

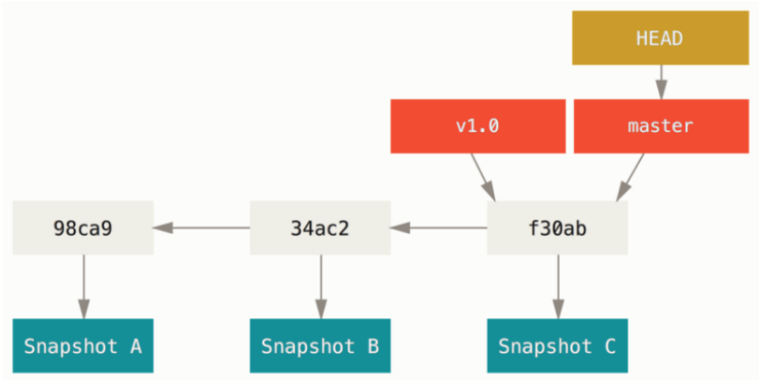



Figura 4.2: Una rama y su historia de confirmaciones.
*En [14]. Bajo Creative Commons License.*

Una confirmación o *commit*, es un comando de Git para registrar cambios. Un desarrollador realiza cambios sobre archivos, prepara estos cambios para que Git realice un seguimiento y, después, confirma estos cambios mediante el comando *commit*. Con la confirmación se debe introducir un mensaje describiendo los cambios realizados [15].

Los problemas hallados en el proceso de producción de *software* en la empresa TbM, referentes al control de versiones son:

- Los desarrolladores trabajan sobre una única rama, la *master*. Esta forma de trabajar resulta inadecuada al desarrollar en equipo, debido a que toda la aportación de código se realiza sobre esta única rama, lo que genera conflictos entre versiones de código, además de suponer un riesgo al no disponer de otra copia del proyecto.

- Difícil seguimiento de la historia de desarrollo del proyecto. Al trabajar sobre una única rama, no se distingue el tipo de aportación al código y todo el código incorporado es de tipo *producción*. Es decir, una rama *master* debería contener el proyecto en producción y si, por ejemplo, se aporta un código para el que se ha solucionado un error, debería indicarse mediante una rama creada expresamente para esa actividad.

- No se dispone de versión de proyecto. Esto implica no identificar el estado del proyecto. Además, dado que aún está en desarrollo y se prevé que sea un proyecto a largo plazo y que pase por varias fases, se debe considerar establecer y mantener versiones.

Además de los problemas presentes en los *pools Product Owner* y *Developer Team* explicados, se han encontrado otras deficiencias referentes a la comunicación:

A pesar de disponer del servicio de comunicación *Slack*, se puede afirmar que no se utiliza. Esta aplicación no se encuentra integrada con el resto de herramientas, y aunque se disponga de una cuenta y un canal o chat a disposición del equipo, el envío de mensajes, si se realiza a través de la aplicación, no es automático. Por este motivo tampoco se ha representado en el Figura 4.1 *Modelización AS IS con BPMN2*.

## 4.3. Rediseño del proceso *TO BE*

Definido el proceso e identificados los problemas y necesidades, a continuación se proponen una serie de mejoras para el proceso. Para ello se ha modelado el proceso incluyendo las mejoras; puede verse la representación BPMN correspondiente en la Figura 4.3. Se proporciona en el Anexo B partes de la figura 4.3 ampliadas, en la Figura B.1 y Figura B.3 para mejorar su lectura.

### 4.3.1. *Pool Product Owner*

**Diferenciar los tipos de tarea**. Dada la existencia de dos tipos de tarea, las resultantes de las historias de usuario y las de tipo problema, se propone que estas tareas aparezcan bien diferenciadas en el panel de *Scrum*. El nombre escogido para las tareas se corresponde también con el nombre que se suele utilizar a la hora de aplicar Git para el control de versiones, de esta manera se logra una mayor coherencia. A partir de ahora se tendrán tareas de tipo *feature* para las derivadas de las historias de usuario, y tareas de tipo *issue* para las de tipo problema. Además, para las tareas de tipo *issue*, se definen las siguientes categorías de problemas:

- *Enhanced*, para propuestas de mejoras.
- *Bugifx*, para errores menores, normalmente errores encontrados en desarrollo.
- *Hotfix*, para errores mayores, normalmente errores encontrados en producción.

Además, para cada *issue* será necesario definir una severidad y prioridad. Las severidades establecidas son:

- *Whishlist*, para características que podrían mejorarse en el proyecto.
- *Normal*, para establecer una preferencia común.
- *Minor*, para problemas de una importancia menor.
- *Major*, para problemas con una necesidad mayor de solución.

Los tipos de prioridad definidos son:

- *High*. Prioridad alta.
- *Low*. Prioridad baja.

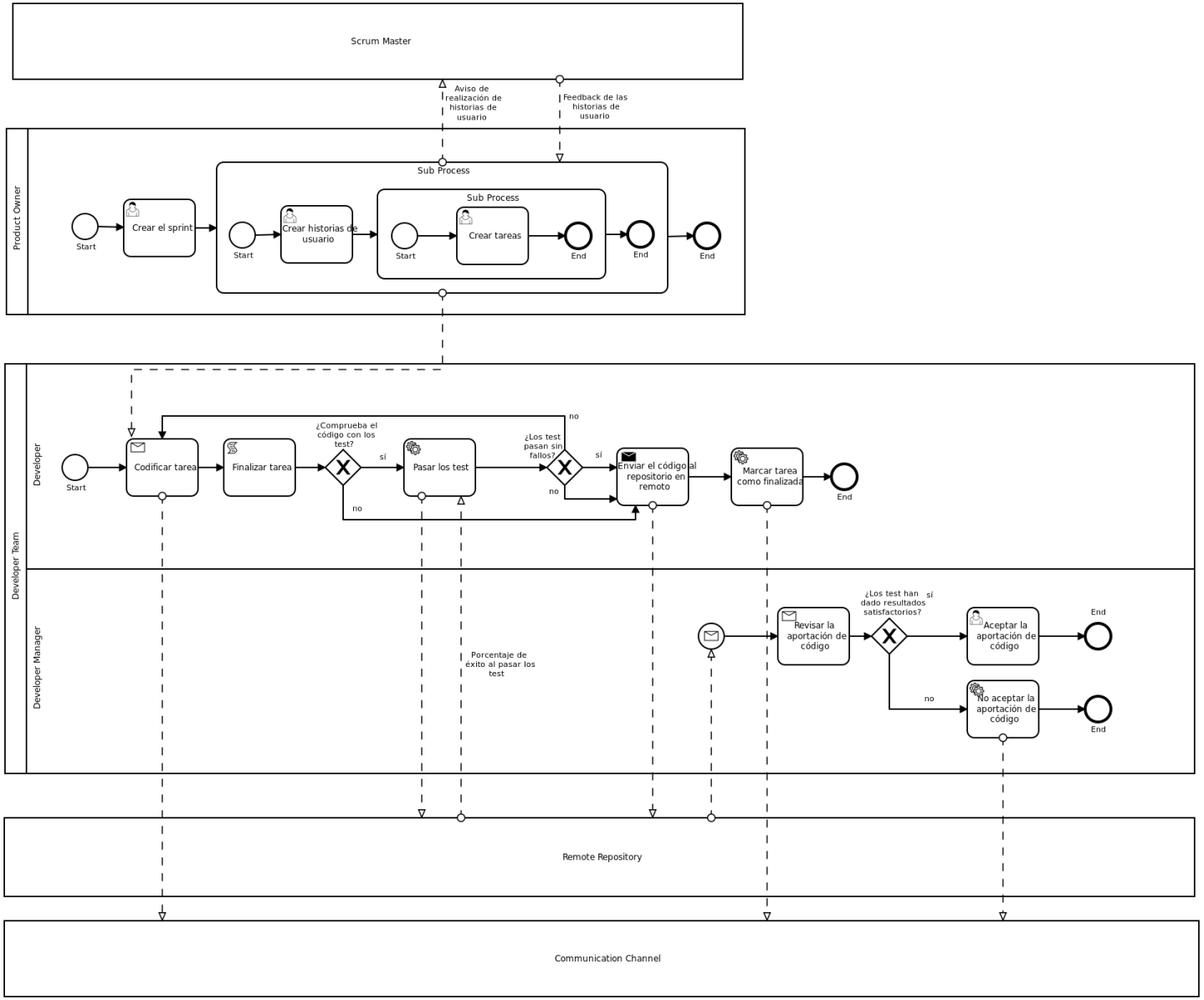



Figura 4.3: Modelización *TO BE* con BPMN2.

### 4.3.2. *Pool Developer Team*

Las mejoras presentadas a continuación, corresponden a las actividades del *pool Developer Team*: *Finalizar tarea*, *Pasar los test* y *Marcar tarea como finalizada*.

**Finalizar tarea de forma automática.** La actividad *Finalizar tarea*, antes llevada a cabo de forma manual, ahora pasar a ser una tarea automática realizada por un *script*.

**Pasar los test de forma automática**. Ahora el pasar los test no depende de la actuación del desarrollador, sino que esto se produce de forma automática, solucionando la inclusión de código en remoto sin cumplir con la cobertura mínima de los test, o incluso sin haberlos realizado.

**Cambiar el estado de una tarea a *closed*.** Una vez subido el código a remoto, la tarea cambia al estado *closed* de forma automática en el panel de *Scrum*. Esta acción conlleva el evitar incoherencias debido a descuidos por parte del desarrollador a la hora de cambiar el estado de la tarea.

Además de estas mejoras para resolver los problemas anteriormente explicados, se proponen también para este *pool* las siguientes mejoras:

**Mostrar las tareas por historia de usuario**. Las tareas sobre las que trabajar deberán mostrarse por historia de usuario, de esta manera un desarrollador, podrá seleccionar una historia de usuario y a continuación se le ofrecerán las tareas correspondientes para trabajar.

**Mostrar las tareas de tipo problema, *issue***. Se dispondrá de otro listado en el que aparecerán las tareas identificadas como problema. Así, se evita la creación de este tipo de tareas de manera manual, facilitando el trabajo del desarrollador.

**Control de versiones**. Se propone la creación de distintas ramas según el trabajo a realizar. Dado que ahora se disponen de diversos tipos de tarea es posible crear diversas ramas según el tipo de actividad a realizar, de esta forma se soluciona la mayoría de conflictos entre las distintas aportaciones de código de los desarrolladores al no realizarlo sobre una única rama y, además, se dispone de varias versiones del código según si está en producción o desarrollo, suponiendo esto una medida de mayor seguridad y mejorando la historia de aportaciones al proyecto.

**Versionar el proyecto**. Mantener números de versiones a medida que evoluciona el proyecto. Los números de versión se efectuarán de manera automática, aprovechando la disponibilidad de las ramas debido a que es posible aumentar el número de versión del proyecto al finalizar una tarea como, por ejemplo, una tarea de tipo *bugfix*.

### 4.3.3. *Pool Communication Channel*

Se añade este nuevo *pool* ante la necesidad de efectuar envíos de notificaciones.

**Notificaciones automáticas**. Tanto al abrir como al cerrar una tarea, se enviará una

notificación a través de la aplicación escogida para el canal de comunicación. También se debería enviar una notificación al desarrollador responsable de una inclusión de código deficiente al repositorio en remoto. Estas acciones se han representado mediante envíos de mensajes entre el *pool Communication Channel* y el *pool Developer Team* en la Figura 4.3.

# Capítulo 5

# Implementación de las mejoras del proceso de producción de *software*

En este apartado se describe para cada mejora identificada anteriormente la evaluación e implementación. Dado que podemos diferenciar varios tipos *pools* en el flujo de trabajo, se muestra, a continuación, la descripción de las mejoras propuestas para cada uno de ellos, así como los requisitos generales y la selección, evaluación e implementación realizada. Las mejoras se dividen en los siguientes *pools*:

- *Product Owner*.
- *Developer Team*.
- *Communication Channel*.

## 5.1. *Pool Product Owner*

En esta sección, se proponen las mejoras que es necesario aplicar para mejorar la funcionalidad del panel de *Scrum* y la disponibilidad de las tareas a realizar en el IDE Pycharm.

### 5.1.1. Descripción de las mejoras

**Panel de *Scrum*:**

El panel de *Scrum* utilizado en la actualidad es Taiga. Los aspectos que se desean mejorar son:

- Distinguir las tareas de tipo problema o *issue* de las tareas *normales* o *feature*.

- Se desea establecer tipos de *issues*: *enhanced*, *bugfix* y *hotfix*.
- Establecer categorías de severidades: *whishlist*, *normal*, *minor* y *major*.
- Crear prioridades para los tipos de *issues*: *low* y *high*.

**Conexión entre el panel de *Scrum* y el IDE Pycharm:**

En la actualidad el IDE Pycharm, utiliza el *plugin taiga*. Este sólo muestra un listado de las tareas con estado *new* o *in progress*. Además, sólo es posible cambiar el estado de una tarea desde el IDE a *new*. Las mejoras para estos problemas son:

- Mostrar las tareas por historia de usuario y por tareas de tipo *issue*.
- Una vez finalizada una tarea, cambiar el estado a *closed* de forma automática sin salir del IDE.

### 5.1.2. Requisitos generales

**Panel de *Scrum*:**

El panel de *Scrum* utilizado en la actualidad no cumple con todas las necesidades. Debido a esto, se realizará un estudio sobre el mismo, para ver si es posible adaptar la configuración, y además se propondrán otras herramientas destinadas al mismo propósito para someterlas a evaluación. En caso de ser necesario, se cambiará la herramienta para adoptar la que sea más adecuada al proceso según el resultado de la evaluación. Los requisitos mínimos que deben cumplir estas herramientas son:

- La aplicación debe ser de tipo *cloud*.
- Se debe poder conectar con las aplicaciones Bitbucket y Slack.
- *Applicacion Programming Interface* (API) pública.
- Poder establecer roles de usuario.
- Crear tareas.
- Crear historias de usuario.
- Establecer estados para las tareas (*features*).
- Crear tareas de tipo problema (*issues*).
- Establecer tipos, severidades y prioridades para los *issues*.
- Poder enviar notificaciones.
- Configurar permisos según el tipo de rol.

- Crear equipo de trabajo.

**Conexión del panel de *Scrum* al IDE Pycharm.**

Es necesario establecer conexión entre el panel de *Scrum* y el IDE Pycharm para poder trabajar sobre las tareas del panel y cambiar el estado de estas. Los requisitos son:

- Mostrar un listado con las historias de usuario.
- Mostrar un listado con los *issues*.
- Mostrar las tareas *features* por historia de usuario.
- Mostrar las tareas de tipos *issue* en un listado a parte de las tareas *feature*.
- Cambiar el estado de una tarea a *closed* sin salir del IDE de forma automática.

### 5.1.3. Selección, evaluación e implementación

La selección de las herramientas se realiza para escoger las que deban evaluarse, y así obtener la más adecuada al proceso. Luego, se realizará la implementación de la herramienta seleccionada.

**Panel de *scrum*:**

Selección:

Los paneles sometidos a evaluación han sido:

- Taiga. Aplicación web para realizar la gestión de proyectos. De los paneles que ofrece esta aplicación, se evalúa el panel de Scrum [16].
- Jira.Herramienta para gestionar el desarrollo de *software* [17].
- Trello. Aplicación que proporciona un tablero para gestionar proyectos [18].
- ScrumDo. Herramienta para planificar, administrar y mejorar el trabajo [19].
- QuickScrum. Herramienta de gestión de proyectos [20].

Evaluación:

La evaluación se ha realizado a través de matrices de valoración. Se ha escogido una versión de cada producto en concreto para evaluar. Se debe tener en cuenta que, la decisión de escoger cada versión de producto proviene de las características que ofrece cada uno. Así, por ejemplo, se ha escogido la versión *Free* de Taiga, ya que cumple con los requisitos necesarios, tal y como

se podrá ver en la evaluación realizada, mientras que para Trello fue necesaria una versión de pago.

Los criterios evaluados para cada herramienta han sido: económicos, tecnológicos, de adaptabilidad, requisitos funcionales y requisitos generales de cada producto. La evaluación completa de las herramienta se puede ver en el Anexo C, apartado C.1. *Evaluación de los paneles de Scrum.*

*Criterios económicos.*

Para la evaluación de los criterios económicos, se tuvo que elegir una versión de cada producto, ya que la mayoría de ellos ofrecen distintos precios según lo que se desee contratar. De esta manera, se seleccionó para Taiga la versión *Free*, para Jira la versión *Jira Software*, para Trello la versión *Business Class*, para ScrumDo la versión *Scrumban* y para QuickScrum la versión *Cloud*. Los criterios que se tuvieron en cuenta fueron la periodicidad de realización del pago, la cantidad del pago y el número de usuarios. Cabe mencionar que la frecuencia de pago se consideró como un requisito para disminuir el número de gestiones con las facturas.

Como resultado de la evaluación se obtuvo que Taiga es mejor económicamente con una puntuación de 1,95, seguido de Jira y ScrumDo con 1,00 punto cada uno, QuickScrum con una puntuación 0,65 y por último Trello con 0,60 puntos.

*Criterios tecnológicos.*

Para evaluar los criterios tecnológicos se tuvieron en cuenta aspectos como el tipo de servicio, ya que debe ser *cloud*, las conexiones necesarias con otras aplicaciones (mínimo con Bitbucket y Slack), API pública, mantener el proyecto privado o el almacenamiento.

El resultado de la evaluación, fue que Trello obtuvo la mejor puntuación con 1,85, seguido de Taiga con 1,80 y Jira con 1,70 puntos. Se obtuvo 0,95 puntos para ScrumDo y por último 0,65 para QuickScrum. Como resultado de esta evaluación se ha obtenido que Trello cumple mejor con los criterios tecnológicos.

*Criterios de adaptabilidad.*

A la hora de realizar la evaluación de adaptabilidad de cada producto, se tuvieron en cuenta los requisitos generales descritos en el apartado 5.1.2. De esta forma se ha evaluado si en la herramienta se podían configurar los *issues*, para poder definir el tipo, la prioridad y la severidad. Además, aunque con menor grado de importancia, también se han evaluado características como configurar el aspecto del tablero.

La puntuación obtenida para los paneles de *Scrum* fue: para Taiga de 0,99, seguida de Jira con 0,90 puntos, 0,55 para QuickScrum, 0,44 para ScrumDo, y Trello con la menor puntuación de 0,25. Tal y como se refleja en la puntuación obtenida, al realizar las pruebas se obtuvo que Taiga admite la configuración deseada, mientras que Trello (con la menor puntuación) no soporta la adaptación necesaria.

*Criterios generales.*

Se ha sometido a evaluación ciertos criterios generales ofrecidos por las herramientas, ya que podrían ser de interés para mejorar el flujo de trabajo. Algunos de estos criterios han sido: si el manejo resulta intuitivo, si dispone de aplicación móvil o adjuntar archivos. Los resultados obtenidos han sido: 0,99 para Taiga, 0,90 para Jira, 0,55 para QuickScrum, 0,44 para ScrumDo y 0,25 para Trello.

*Criterios funcionales.*

En los criterios funcionales se han evaluado características como poder establecer un equipo de trabajo, si las tareas tienen un código para identificarlas o si es posible mantener un estado para las tareas.

Las puntuaciones han sido de 1,00 para Taiga y de 0,96 para Jira. A continuación, QuickScrum ha obtenido una puntuación de 0,75, y ScrumDo 0,60 puntos, mientras que Trello ha obtenido la menor calificación con 0,24 puntos.

*Resultado.*

Una vez alcanzados los resultados de cada criterio, se ha obtenido el resultado final de la evaluación. En total han sido cinco tipos de criterio tal y como se puede ver en la Tabla 5.1. Todos tienen el mismo el peso, ya que se les otorga la misma importancia, excepto para los criterios económicos, donde el valor es menor, debido a que la empresa estaría dispuesta a pagar una cantidad mayor a cambio de mejorar el resto de criterios. Así pues, se ha obtenido la mayor puntuación para Taiga de 1,31 puntos frente al resto de herramientas. En segundo lugar, le sigue Jira con una puntuación de 1,10, después ScrumDo con 0,72 puntos, Trello con 0,69 puntos y finalmente con la menor puntuación, QuickScrum con 0,62 puntos.

Tabla 5.1: Totales de la evaluación de los paneles de *Scrum*.

| Criterios | Peso | Taiga | Jira | Trello | ScrumDo | QuickScrum |
|---|---|---|---|---|---|---|
| **Versión del producto** | - | *Free* | *Jira Software* | *Business Class* | *Scrumban* | *Cloud* |
| **Económicos** | 16% | 1,95 | 1,00 | 0,60 | 1,00 | 0,65 |
| **Generales** | 21% | 0,97 | 0,92 | 0,51 | 0,68 | 0,50 |
| **Tecnológicos** | 21% | 1,80 | 1,70 | 1,85 | 0,95 | 0,65 |
| **Adaptabilidad** | 21% | 0,99 | 0,90 | 0,25 | 0,44 | 0,56 |
| **Funcionales** | 21% | 1,00 | 0,96 | 0,24 | 0,62 | 0,75 |
| **Total** | 100% | **1,31** | 1,10 | 0,69 | 0,72 | 0,62 |

**Nota.** Se debe tener en cuenta que la elección de la versión de cada producto a evaluar está condicionada por los propios requisitos. Por ejemplo, para la versión de Trello se seleccionó la versión *Business Class*, ya que la versión gratuita sólo ofrecía la conexión a un servicio externo y se necesita más de una conexión.

Implementación:

Dados los resultados de la evaluación, se concluye que se seguirá trabajando con Taiga ya que ha conseguido la mayor puntuación. Se procede a configurar las mejoras necesarias en la herramienta:

- Se establecen los tipos de *issues*: *enhancement*, *bugfix* y *hotfix*.
- Los tipos de severidad: *whislist*, *normal*, *minor*, *major*.
- Se establecen las prioridades: *low*, *high*.

Además, el grado de severidad y prioridad tendrá asociado un color, de manera que el grado resulte visualmente representativo. Puede verse en la Tabla 5.2 cómo se ha establecido la configuración:

Tabla 5.2: Configuración de los *issues* para el panel de *scrum*.

| Característica | Nombre color | Color |
|---|---|---|
| **Tipo de *issue*** | | |
| *Enhancement* | azul | |
| *Bugfix* | amarillo | |
| *Hotfix* | rojo | |
| **Tipo de *severity*** | | |
| *Whishlist* | gris | |
| *Normal* | azul | |
| *Minor* | amarillo | |
| *Major* | rojo | |
| **Tipo de *priority*** | | |
| *Low* | azul | |
| *High* | rojo | |

Cabe comentar que el estado de los *issues* (*new*, *in progress*, *closed*) no fue necesario configurarlo, ya que la herramienta ofreció de forma predeterminada los mismos estados que el de las tareas derivadas de las historias de usuario.

En la Figura 5.1, se muestra el panel Taiga funcionando con la configuración llevada a cabo.

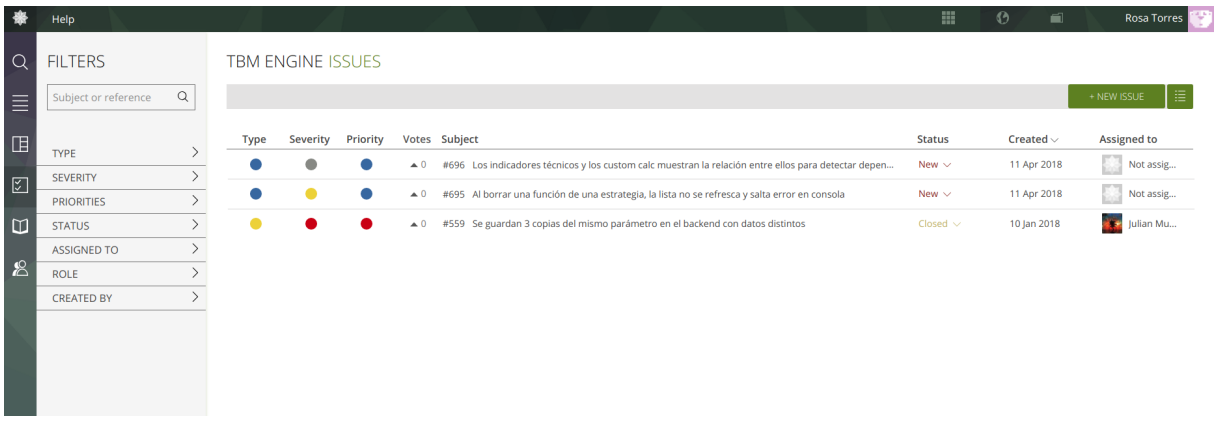



Figura 5.1: Resultado de la configuración para los *issues* en Taiga.

**Conexión del panel de *Scrum* al IDE Pycharm:**

A continuación, se selecciona, evalúa e implementa la funcionalidad necesaria para cubrir los requisitos respecto a la conexión entre el panel de *Scrum* Taiga y el IDE Pycharm.

Selección:

En este caso no se realiza selección, dado que sólo existe un *plugin* para realizar la conexión entre el IDE y el panel de *Scrum*. Por tanto, se realizará la evaluación sobre este *plugin*.

Evaluación:

La evaluación realizada ha consistido en probar las acciones ofrecidas por esta herramienta. El *plugin* ofrece un listado de las tareas con estado *new* e *in progress*, y actualiza el estado a *in progress* en el panel de *Scrum* al iniciar una tarea. Las tareas que ofrece, son sólo las de tipo *feature*, y éstas no se pueden distinguir por historia de usuario. El menú ofrecido por el *plugin* puede verse en la Figura 5.2:

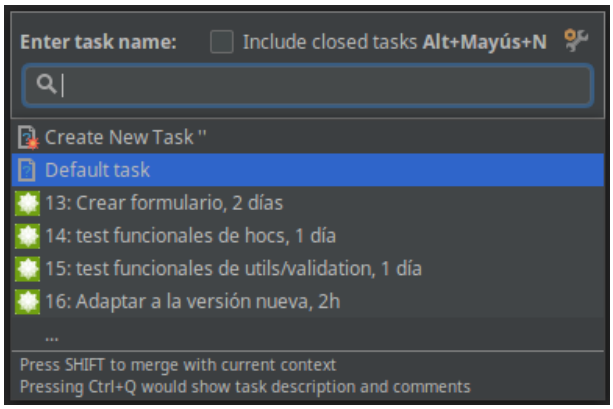



Figura 5.2: Menú para realizar las tareas ofrecido por el *plugin taiga*.

Dado que el *plugin* no cubre todas las necesidades identificadas, se decide programar esta funcionalidad.

Implementación:

La siguiente adaptación fue realizar la programación para, desde Pycharm, disponer de un menú donde realizar las acciones que involucran al panel de *Scrum*, sin abandonar el entorno de trabajo del IDE. Para la realización de la programación que permite la comunicación con Taiga, se consultó la documentación referente a su *Application Programming Interface* (API) [21], [22].

Con esta programación se ha logrado disponer de un menú en la consola del entorno de desarrollo. El menú permite seleccionar el tipo de trabajo que debe realizarse dividido en dos grupos: historias de usuario o *issues*. Además, sólo se muestran los asignados al usuario. Una vez seleccionado uno de estos dos grupos, se muestra a continuación un submenú con las tareas. Las tareas mostradas son las que poseen estado nuevo o en progreso, es decir, en el menú no se mostrarán las tareas cerradas, ya que no se va a trabajar sobre ellas. Si se diera la excepción de tener que reabrir una tarea cerrada, sólo habría que ir al panel de *Scrum*, cambiar el estado a nuevo o en progreso, y entonces se mostraría en el menú. Puede verse cómo se muestra este menú en el IDE en la Figura 5.3, en la cual primero se ha accedido a ver las historias de usuario y se ha seleccionado una de ellas, para a continuación mostrar el listado de las tareas que tiene asociadas.

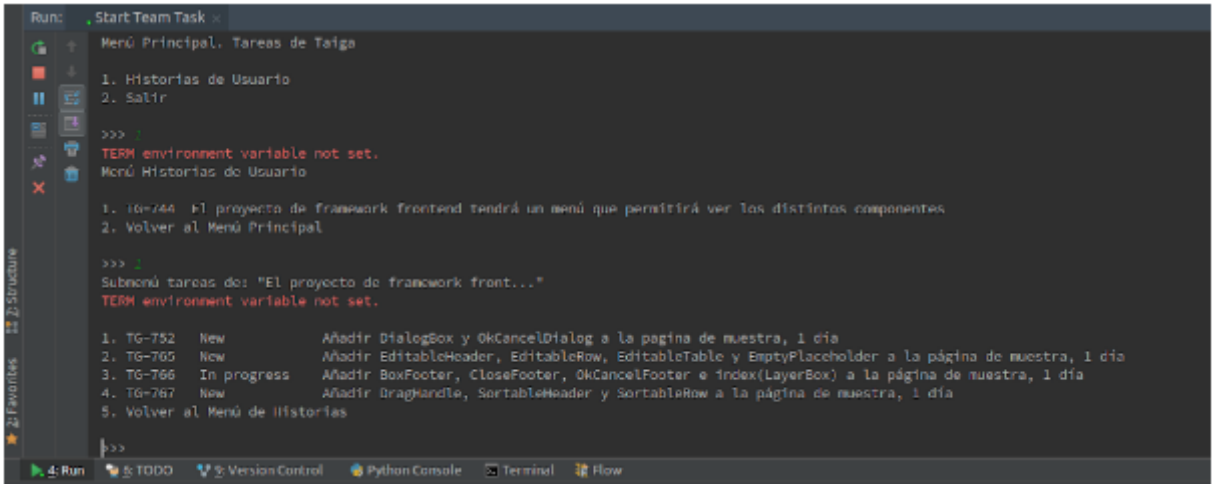



Figura 5.3: Menú de *Scrum* en Pycharm.

Para iniciar el menú mostrado en la Figura 5.3, realizar la sincronización del proyecto (sincronizar el proyecto en local con el remoto), iniciar y finalizar tareas de tipo *feature*, *bugfix*, *hotfix* y *release*[1], se ha añadido un nuevo menú con botones en el IDE Pycharm. Los iconos para los botones se han escogido de manera que resulten visualmente representativos, teniendo en cuenta incluso el color de éstos, como por ejemplo para los tipos de tarea. Estas se han representado con chinchetas y se ha escogido el color según su importancia: azul para las tareas

[1]Una nueva versión del proyecto para producción.

de tipo *feature*, amarillo para los de tipo *bugfix* y rojo para las de tipo *hotfix*. Se muestra en la siguiente Tabla 5.3 la descripción de cada botón.

Tabla 5.3: Botones configurados para el menú del IDE Pycharm.

| Icono | Definición |
|---|---|
|  | Sincronizar el trabajo con remoto. |
|  | Iniciar el menú de *scrum*. |
|  | Finalizar una rama de tipo *feature*. |
|  | Finalizar una rama de tipo *bugifx*. |
|  | Finalizar una rama de tipo *hotfix*. |
|  | Iniciar una rama de tipo *release*. |
|  | Finalizar una rama de tipo *release*. |

A continuación, se muestra en la Figura 5.4, el resultado del menú integrado en el IDE Pycharm. Pueden verse los botones del menú en la parte superior derecha de la imagen.

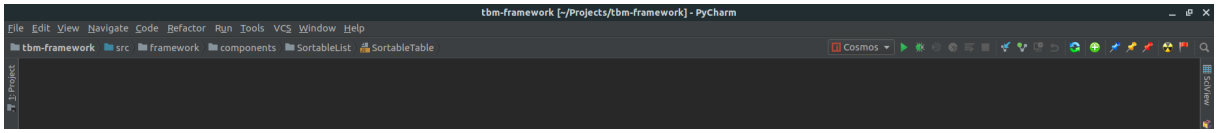



Figura 5.4: Menú añadido en Pycharm.

## 5.2. Pool Developer Team

En este *pool* se describen las mejoras para poder establecer el cierre de las tareas, así como la ejecución de los test y el manejo del control de versiones. No se realiza evaluación del IDE Pycharm dado que la empresa no desea cambiarlo.

### 5.2.1. Descripción de las mejoras

Se desean conseguir las siguientes mejoras:

Mejorar el uso del control de versiones mediante Git. Para ello se propone utilizar varias ramas:

- Una rama *master*, para la versión del proyecto en producción.

- Una rama *develop*, para la versión del proyecto en desarrollo.
- Utilizar una rama por cada tarea que deba realizar el desarrollador, con el mismo nombre que el identificador de la tarea en el panel de *Scrum.* Por ejemplo, si una tarea tiene el identificador 807, la rama se nombrará *feature/807.*
- Ramas *release*, para incorporar las nuevas versiones del proyecto desde desarrollo a producción. La rama *release* se creará con el nombre *release* seguido del número de versión correspondiente al estado del proyecto.

Además, se pretende automatizar la creación de las ramas, así como las *fusiones* (*merge*) entre ellas como, por ejemplo, incorporar el código de una rama de una tarea (o rama de tipo *feature*) a la rama *develop*, o lo que es lo mismo, finalizar una tarea. Al finalizar una tarea, se deben pasar los test y si son satisfactorios incorporar el código al repositorio en remoto.

Mejorar la realización de los test. Para evitar que un desarrollador suba código a remoto sin haber pasado los test, o sin que estos sean satisfactorios, se propone realizar los test de forma automática, y en caso de que se pasen correctamente, subir también el código a remoto y finalizar la tarea. En caso de que no se obtenga un resultado óptimo al realizar los test, se cancela el cierre de la tarea, obligando al desarrollador a revisar el código.

Formatear e identificar los *commits* adecuadamente. Para cerrar una rama antes se deben confirmar los cambios; esto se realiza mediante el comando *commit*, y el desarrollador, además, debe incluir una descripción de los cambios e incorporaciones realizados en el código. Esto es de especial importancia ya que estos *commits* describen las incorporaciones realizadas, lo que genera una *historia* de desarrollo del proyecto, y de esta manera se ayuda también a los desarrolladores a saber qué se ha realizado en cada momento. Cada *commit* deberá tener el siguiente formato:

*TG-[número de tarea] [tipo de rama]: Descripción.*

Establecer números de versión para el proyecto. Para esto se ha establecido que las tareas *release*, *bugfix* y *hoftix*, incrementarán la versión. El formato base para la versión será v.x.x.x, y los incrementos se realizarán de la siguiente forma:

- *Release.* Incrementará el segundo dígito y, además, se le concatenará *rc*: v.x.+1.x-rc.
- *Hotfix.* Incrementará el tercer dígito: v.x.x.+1
- *Bugfix.* Incrementará también el tercer dígito: v.x.x.+1

### 5.2.2. Requisitos generales

Se han de tener en cuenta los siguientes requisitos para lograr las mejoras anteriormente propuestas.

- Para mejorar y facilitar el control de versiones, se debe trabajar con Git de la siguiente manera:

    - Los desarrolladores deben trabajar en una rama separada de la rama principal. De esta forma las aportaciones de código son más seguras, y se pasan los test por cada nueva aportación de código.
    - Es deseable tener varios tipos de ramas según el tipo de tarea que deba realizarse (en adelante al referirse a un tipo de tarea, también se entenderá que se hace referencia a la rama del mismo tipo). Por ejemplo, si la tarea es de tipo problema *bugfix*, deberá realizarse dicha tarea sobre una rama del mismo tipo, *bugfix*.
    - Distinguir los trabajos que se realicen para generar una nueva versión del proyecto mediante ramas creadas para tal fin, es decir, ramas *release*.
    - Obtener números de versión adecuados para el proyecto.
- Automatizar la realización de los test.
- Automatizar la incorporación del nuevo código en remoto si el resultado de los test es satisfactorio, o cancelar el cierre de la tarea, si resulta insatisfactorio.
- Formatear e identificar de forma adecuada los *commits*.

### 5.2.3. Selección, evaluación e implementación

**Selección:**

Para adaptar el control de reposiciones, la empresa sugirió estudiar el modelo de ramificación Git-flow. Este estudio se ha incorporado en la memoria en el Anexo D. *Git-flow. Modelo de ramificación*. Debido a que se consideró la forma de trabajar que propone el modelo Git-flow, se seleccionaron herramientas que le diesen soporte. Las herramientas son:

- *Software* Git-flow.
- *Plugin Git-flow* para el IDE Pycharm.

**Evaluación:**

Git-flow. Modelo de ramificación

A continuación, se presenta cómo propone el modelo Git-flow realizar la ramificación.

- Rama *master*. Versión del proyecto para producción.
- Rama *develop*. Versión del proyecto para desarrollo.
- Rama *release*. Para incluir una nueva versión del proyecto.
- Rama *feature*. Rama para realizar las tareas.
- Rama *hotfix*. Para realizar las tareas identificadas como errores del proyecto en producción.
- Rama *bugfix*. Para realizar tareas identificadas como errores en desarrollo.

De esta forma se cumplirían los requisitos de trabajar con distintas ramas, diferenciar los tipos de tarea y mantener una versión del proyecto para desarrollo y otra para producción. Además, este modelo contempla el uso de *tags* para establecer las versiones del proyecto.

Plugin Git-flow

Para utilizar el *plugin* Git-flow, dado que Git-flow proporciona una conjunto de extensiones sobre Git, primero se instaló en el sistema operativo, y luego se realizó la instalación del *plugin* en el IDE Pycharm. Lo siguiente fue realizar la configuración. En la configuración se pudo apreciar cómo ofrecía soporte para mantener una rama en producción, una rama para desarrollo, además de las ramas de tipo *feature*, *release*, *hotfix* y *bugfix*, de manera que un desarrollador podría trabajar en distintas ramas según el tipo tarea, y mantendría una versión del proyecto para producción y otra para desarrollo. Puede verse esta configuración en la Figura 5.5.

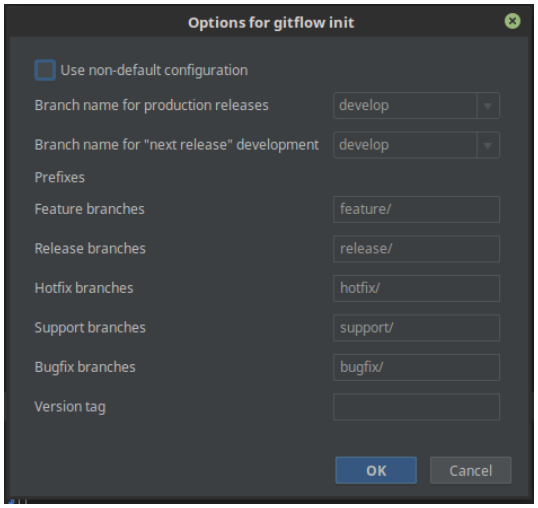



Figura 5.5: Configuración inicial del *plugin* Git-flow para Pycharm.

Sin embargo, una vez instalado, las opciones fueron limitadas. Inicialmente permite crear una rama de tipo *feature*, *release* o *hotfix*, tal y como se muestra en la Figura 5.6 en el menú izquierdo. No se muestra la opción de crear una rama de tipo *bugfix*. El motivo es que el modelo Git-flow establece que inicialmente una rama de tipo *bugfix* se crea a partir de una *release*; entonces, esto implica crear primero una rama de tipo *release*. Esta forma de trabajar, aunque es la que se propone inicialmente en el modelo, admite una variante, y es crear una rama *bugfix* desde una rama *develop*. Esta variante es la que se decidió adoptar para aplicar la modelización de ramas establecida por Git-flow al proceso y que el *plugin* no permite configurar.

Otra característica que se debe destacar, es que una vez iniciada una rama, el menú pasa a ofrecer otras opciones para finalizar dicha rama. Por ejemplo, al finalizar una tarea de tipo *feature* (y realizada sobre el mismo tipo de rama), la finalización del trabajo se debe realizar en dos pasos: uno en el que se cierra la rama (*Finish*) y otro en el que se publica (*Publish*) o lo que es lo mismo, enviar el código al repositorio en remoto. Esto puede verse en la Figura 5.6 en el menú de la derecha.

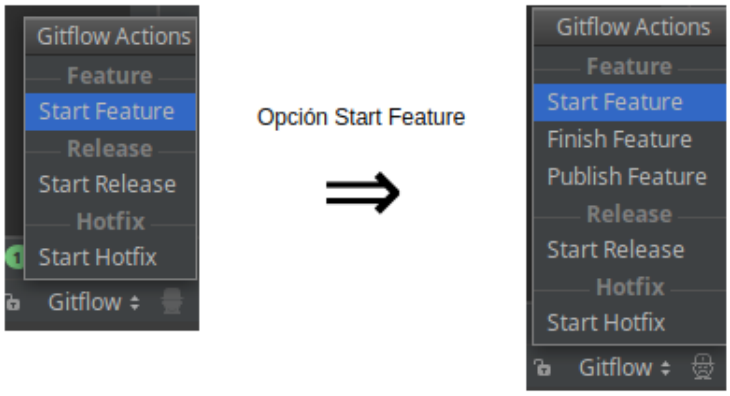



Figura 5.6: Opciones del *plugin Git-flow* tras haber creado una rama de tipo *feature*.

En la siguiente Tabla 5.4 se muestran los requisitos evaluados en las pruebas, y si son soportados o no.

Tabla 5.4: Requisitos soportados por el *plugin Git-flow*.

| Requisitos | Soportado |
|---|---|
| Crear rama *feature* | sí |
| Finalizar rama *feature* | sí |
| Crear rama *release* | sí |
| Finalizar rama *release* | sí |
| Crear rama *hotfix* | sí |
| Finalizar rama *hotfix* | sí |
| Crear rama *bugfix* | no |
| Finalizar rama *bugfix* | no |
| Crear un nuevo *tag* | no |
| Publicar el *tag* en remoto | no |
| Crear más de una rama *feature* | no |

**Nota.** Se añade como requisito *Crear más de una rama feature*, dado que se considera esta posibilidad, y el *plugin* no permite crear más de una rama del mismo tipo.

Después de realizar las pruebas, se ha obtenido que el *plugin Git-flow* no cubre todos los requisitos, tal y como se puede ver en la anterior Tabla 5.4. Además, la forma en que se realizan algunas tareas a través del *plugin*, hace que se dependa de la actuación del desarrollador. Por ejemplo, para terminar una tarea en una rama de tipo *feature*, este puede seleccionar la opción *Finish*, lo que finalizará el trabajo en local, pero hasta que no seleccione la opción *Publish*, no se incorporará al repositorio en remoto. Otro problema hallado y considerado grave, es que durante las pruebas realizadas el *plugin* mostró un comportamiento errático como, por ejemplo, no ofrecer las opciones de *Finish* y *Publish* después de crear una rama *feature*, lo que conllevó tener que reiniciar el IDE para que se mostrasen las opciones.

Puesto que el *plugin* no realiza toda la funcionalidad deseada, y además depende bastante de la actuación del desarrollador, se decide no utilizar esta herramienta y programar la adaptación necesaria. De esta forma se automatizarán las acciones, facilitando el trabajo del desarrollador y disminuyendo la posibilidad de cometer errores.

### Implementación

La implementación ha consistido en programar la adaptación necesaria para lograr automatizar las acciones que se realizan para llevar a cabo el control de versiones. Para esto, fue necesario estudiar previamente el modelo de ramificación Git-flow [23], [24], [25], [26], para el que después se realizaron algunas modificaciones debido a los requisitos. Por ejemplo, al realizar el cierre de una rama de tipo *feature*, se ha incorporado la automatización de los test, de manera que ahora la acción ya no depende del desarrollador. Otro de los requisitos conseguidos ha sido mantener un número de versión del proyecto (*tag*), de manera que corresponda con el estado actual del proyecto, y además se incremente el número de versión en caso de crear, por ejemplo, una rama de tipo *release* o una rama *bugfix*. Se presenta en la Tabla 5.5, un resumen del flujo general de Git-flow y las decisiones de adaptación que se tomaron frente a este modelo.

Se debe tener en cuenta que, además, se ha cambiado cómo se realizan los test. Inicialmente se realizaban a través del servicio web Bitbucket. Este aspecto se consideró en un principio, mantenerlo y así se reflejó en el proceso mejorado, pero al automatizar el cierre de las ramas y la inclusión de código a remoto, se aprovechó este aspecto para efectuar los test antes de enviarlos al repositorio en Bitbucket, ahorrando de esta forma un paso: el envío de código para ejecutar los test en remoto. Ahora, para el cierre de una tarea un desarrollador realiza un *commit* y agrega una descripción de los cambios realizados en el código. A continuación, los test se pasan de forma automática, y si son satisfactorios se envía el código a Bitbucket y se cierra la rama. En caso de que éstos no sean satisfactorios, se cancela el envío y el cierre, y se muestra un mensaje de error al usuario para que proceda a revisar el código.

Tabla 5.5: Adaptación del flujo general del modelo de ramificación Git-flow.

| **Flujo general según la metodología git-flow** | **Adaptación del flujo** |
|---|---|
| **Rama *develop*** | |
| Se crea la rama *develop* desde *master* | *Sin cambios* |
| **Rama *feature*** | |
| Se crea la rama *feature* desde *develop* | - Se crea la rama *feature* desde *develop*<br>- Se publica la rama *feature* en remoto. |
| La rama *feature* al finalizar se fusiona con *develop* y se elimina. | - Pasar los test.<br>- Si son satisfactorios:<br>- Fusionar la rama con *develop*.<br>- Enviar los cambios a remoto.<br>- Eliminar la rama |
| **Rama *release*** | |
| Se crea la rama *release* desde *develop* | *Sin cambios* |
| La rama *release* al finalizar se fusiona con *develop* y *master*, y se elimina. | - Comprobar que la rama existe en remoto.<br>- Pasar los test.<br>- Incrementar la versión del proyecto.<br>- Fusionar la rama con *develop*.<br>- Crear un *pull request*(1) hacia *master*. |
| **Rama *hotfix*** | |
| Si se detecta un error, se crea una rama *hotfix* desde *master* | Si se detecta un error en la rama *master* se crea una rama *hotfix*. |
| La rama *hotfix* al finalizar se fusiona con *develop* y *master*. A excepción de si existe una rama *release*, en cuyo caso se fusionará con esta. | - Pasar los test. Si son satisfactorios:<br>- Incrementar la versión del proyecto.<br>- Fusionar con *develop*.<br>- Crear un *pull request*(1) hacia *master*. |
| **Rama *bugfix*** | |
| Si se detecta un error en una rama *release*, se crea una rama *bugfix* desde *release* | *Sin cambios* |
| La rama *bugfix* al finalizar se fusiona con *release*. | - Pasar los test.<br>- Incrementar la versión del proyecto.<br>- Incorporar los cambios de la rama *bugfix* en la rama *release*.<br>- Eliminar la rama *bugfix*.<br>- Incorporar los cambios de la rama *release* en *develop*. |

**Nota**. (1) Un *pull request* es un método para enviar el nuevo código a remoto mediante una petición. Dicha petición queda pendiente de aprobación por parte del responsable del repositorio, lo que supone una seguridad para evitar la incoporación de código no deseado al proyecto.

Por último, se configuró el formato de los mensajes para realizar los *commits*. Para ello se ha hecho uso de una plantilla [27] en el IDE Pycharm. Fue necesario configurar la plantilla para que mostrase el nombre de la rama y el número de la tarea para que el desarrollador sólo tuviese que añadir la descripción de los cambios realizados. Puede verse la plantilla en la Figura 5.7, en la que aparece primero *TG-794 feature:*, correspondiente al identificador y tipo de una tarea, para incluir a continuación la descripción.

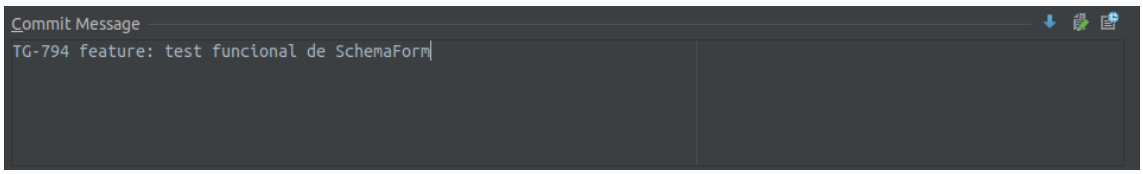



Figura 5.7: Plantilla *commit message* para introducir la descripción de los cambios.

## 5.3. Pool Communication Channel

La herramienta disponible en la empresa en la actualidad es Slack, aunque no se utiliza de forma habitual. Dado que se desconocen todas las funcionalidades de esta herramienta y será necesario estudiarla, se escogen también otras aplicaciones destinadas a la comunicación para someterlas a estudio y evaluación junto a Slack. De esta forma se obtendrá la herramienta más adecuada para el flujo de trabajo. Después, se debería realizar la implementación y configuración de la aplicación seleccionada.

### 5.3.1. Descripción de las mejoras

Las mejoras identificadas son:

- Si se detecta una incidencia al incorporar código en remoto, se debe notificar al desarrollador implicado.
- Cuando se crea una tarea de tipo *issue* en el panel de *Scrum* Taiga, se debe enviar una notificación al desarrollador implicado.
- Se deben mostrar notificaciones en un canal general disponible para todo el equipo de desarrolladores, al iniciar y al terminar cada tarea.

### 5.3.2. Requisitos generales

- Publicar un mensaje en el canal general cuando un desarrollador empiece o finalice una tarea.
- Generar notificaciones desde el control de reposiciones, es decir, desde el servicio Bitbucket.
- Generar notificaciones desde el panel de *Scrum* Taiga.
- Enviar una notificación a la persona implicada en caso de incidencia.

### 5.3.3. Selección, evaluación e implementación

Se listan a continuación las herramientas escogidas para el estudio:

**Selección:**

Las herramientas evaluadas han sido:

- Slack. Aplicación de comunicación que permite crear canales en los que puede participar todo el equipo de desarrollo [28].
- Stride. Herramienta de comunicación para equipos de trabajo [29].
- Flock. Aplicación que permite la comunicación y colaboración de un equipo de trabajo [30].
- Fleep. Es una aplicación que permite la comunicación de organizaciones de trabajo [31].
- Ryve. Permite la comunicación incluyendo equipos de trabajo, y así organizar la realización de las tareas [32].

**Evaluación:**

Para la evaluación se han utilizado matrices de valoración. La elección de las versiones gratuitas para todas las herramientas se ha debido a que la empresa prefiere ahorrar por ahora en este servicio.

Los criterios evaluados para cada herramienta han sido los criterios funcionales, ya que la herramienta debe cumplir con las conexiones a los servicios Taiga y Bitbucket. También se han considerado criterios generales para comparar las características de las herramientas, y de esta forma escoger la que ofrezca unos servicios más adaptados al flujo del proceso. La evaluación completa se puede ver en el Anexo C, apartado C.2 *Evaluación de las aplicaciones de comunicación.*

*Criterios funcionales.*

En los criterios funcionales se ha evaluado la conexión con Bitbucket, con Taiga y el poder configurar el envío de mensajes automatizados, tanto a un canal general como a un determinado usuario, de esta forma se podría notificar a un desarrollador en caso de que su aportación de código al repositorio en remoto fuese rechazada.

Se debe distinguir que algunos servicios ofrecían como opción vincularse a Bitbucket, de forma que sólo había que seleccionar esta opción para establecer la conexión. Otras herramientas, sin embargo, no disponían de este servicio, pero sí permitían conexión a través de *webhook*[2]. Aún así, con las aplicaciones Fleep y Ryver no fue posible recibir notificaciones; a través de

[2]«Un Webhook es una manera de ser notificado cuando un evento ha ocurrido en tu aplicación o la de un tercero. Es básicamente una solicitud POST que se envía a una URL específica.» [33]

Taiga para Fleep, y de Bitbucket y Taiga para Ryver. Sí se consiguió con éxito para Slack con Taiga. Esto ha quedado reflejado en la puntuación obtenida, pues Slack tiene un total 0,68 puntos, frente a Stride, Flock y Fleep con 0,34 puntos, ya que en estas tres sólo fue posible enviar notificaciones con el servicio Bitbucket, y 0 puntos para Ryver, debido a que no se logró con ninguno de los dos servicios.

*Criterios generales.*

Para los criterios generales se han evaluado aspectos comunes a todas las herramientas. Destacan los siguientes criterios: las herramientas han de ser de tipo aplicación web, necesidad de integración con otros servicios y aplicaciones (enviar notificaciones desde Bitbucket y Taiga), la cantidad de aplicaciones que pueden ser integradas, las notificaciones y el formateo para escribir mensajes con fragmentes de código, para que estos resulten de más claridad. Los servicios con mayor puntuación han sido Flock y Fleep, aún así se deberá tener en cuenta que Fleep no ofrece formatear mensajes con código. Después siguen Slack y Strice con 0,87 puntos, y por último Ryver con 0,72 puntos.

*Resultado.*

El resultado obtenido se presenta en la Tabla 5.6. Se puede ver cómo se ha otorgado un peso mayor a los criterios funcionales con 55 %, por considerarlos imprescindibles, y un peso de 45 % para los criterios generales. Así pues, de esta manera se ha obtenido que Slack obtiene la mayor puntuación con 1,55 puntos, seguido de Flock y Fleep, ambos con 1,24 puntos, Stride con 1,55 puntos, y finalmente Ryver con la menor puntuación de 0,72 puntos. Se concluye con el resultado de la evaluación que Slack será la herramienta que se integre como canal de comunicación para el proceso. Aun así, se debe tener en cuenta que ninguna aplicación logró enviar notificaciones a un usuario en concreto, pudiéndose enviar sólo al canal general. (Esto se probó enviando notificaciones tanto desde el panel de *Scrum* Taiga, como desde el control de versiones Bitbucket).

Tabla 5.6: Resultado de la evaluación de las aplicaciones de comunicación.

| Criterios | Peso | Slack | Stride | Flock | Fleep | Ryver |
|---|---|---|---|---|---|---|
| **Versión del producto** | - | *Gratuito* | *Free* | *Plan gratuito* | *Basic* | *Free* |
| **Funcionales** | 55% | 0,68 | 0,34 | 0,34 | 0,34 | 0,00 |
| **Generales** | 45% | 0,87 | 0,87 | 0,90 | 0,90 | 0,72 |
| **Total** | 100% | **1,55** | 1,21 | 1,24 | 1,24 | 0,72 |

**Nota.** Todas las versiones de los productos evaluados han sido versiones gratuitas.

**Implementación:**

Debido a la restricción del tiempo para la realización de las prácticas y, a que ninguna herramienta cumplió del todo con los requisitos recogidos, se prescindió de la adaptación de Slack para el flujo de trabajo. Pese a esto, se configuraron los servicios Taiga y Bitbucket para recibir notificaciones a través de correo electrónico, y se estableció como canal de comunicación para reuniones y envío de mensajes la aplicación Skype. Se deja como extensión para este proyecto la integración y adaptación del servicio de comunicación, además se debe tener en cuenta que, de cambiar la situación de la empresa, (aumento de empleados, cambios en las

herramientas utilizadas, etc.) se debería realizar una nueva evaluación por si fuese necesario seleccionar, otras versiones de los productos, debido a que el estudio se limitó únicamente a las versiones gratuitas.

# Capítulo 6

# Pruebas

Las pruebas han consistido en reproducir parte del trabajo diario que se espera realizar. Se ha utilizado el panel de *Scrum* Taiga y el IDE Pycharm con el nuevo menú creado para poder acceder a las tareas desde el IDE y realizar el control de versiones de forma automática. Estas pruebas se realizaron y se presentaron al supervisor de las prácticas en la estancia en la empresa TbM. Como se deseaban realizar todas las tareas desde el propio IDE sin tener que abandonarlo, se realizan las pruebas desde el *pool Developer Team*. A continuación, se describen las pruebas realizadas:

## 6.1. Pruebas realizadas en el *pool Developer Team*

Para la realización de las pruebas, se creó primero las historias de usuario con sus tareas de tipo *feature* y tareas de tipo *issue* en el panel de Taiga para que estuvieran disponibles desde el menú creado en el IDE para tal fin. Además, se crearon *issues* de distintos tipos, combinando las severidades y prioridades.

### 6.1.1. Se puede sincronizar el proyecto en local con el remoto

**Objetivo:**

Para esta prueba se espera poder actualizar el código en local con el disponible en remoto. Para esto se hará uso del botón existente en el menú implementado en el entorno de desarrollo.

**Pasos realizados:**

Se pulsa el botón correspondiente del menú y se observa cómo el código en local es actualizado con el código disponible en remoto.

### 6.1.2. Es posible iniciar el menú para tareas *Scrum*

**Objetivo:**

Inicializar y mostrar el menú creado para realizar las tareas del panel de *Scrum*, a través del botón disponible en el menú integrado en el IDE Pycharm.

**Pasos realizados:**

Se pulsa el botón correspondiente en el menú de Pycharm y se observa cómo se muestra el menú para realizar las tareas del panel de *Scrum*.

### 6.1.3. Se pueden iniciar tareas de tipo *feature*

**Objetivo:**

En esta prueba se espera poder acceder al listado de las tareas a través de las historias de usuario, seleccionar la tarea para trabajar sobre ella y que, además, se cree la rama correspondiente para iniciar el seguimiento de la codificación a través del control de versiones de forma automática.

**Pasos realizados:**

Primero se inició el menú, mostrando el listado de las historias de usuario, y se seleccionó una historia, para la que a continuación el menú ofreció sus tareas. Al seleccionar una de ellas, se creó de forma automática una rama con el mismo nombre que el tipo de la tarea, *feature*, seguido del identificador numérico que le correspondía, y se cerró el menú automáticamente. Se comprobó también que las tareas se muestran tanto en el estado *new* e *in progress*, y que no se muestran las tareas con estado *closed*.

### 6.1.4. Se pueden iniciar tareas de tipo *issue*

**Objetivo:**

Se espera poder trabajar sobre tareas de tipo *issue* seleccionándolas a través del menú creado para este próposito en el IDE. En el menú, junto al nombre de la tarea, se debe mostrar su prioridad y severidad para ayudar a escoger la tarea sobre la que trabajar.

**Pasos realizados:**

Nuevamente se inició el menú. Esta vez se seleccionó la opción para mostrar el listado de *issues*, y se escogió una tarea de tipo problema para trabajar sobre ella. De forma automática se creó una rama con el nombre del tipo del problema, *bugfix* en este caso, seguido del número de tarea. Se repitieron estos pasos para probar los distintos tipos de *issue* con diferentes priori-

dades y severidades, obteniendo el comportamiento esperado. También se comprobó que no se muestran las tareas cerradas.

### 6.1.5. Se pueden cerrar tareas de tipo *feature*, *bugfix* y *hotfix*

Se agrupa a continuación en una sola explicación las pruebas realizadas para estos tipos de tarea, debido a que las pruebas son idénticas.

**Objetivo:**

Se debe realizar primero un *commit*. Luego, se debe poder seleccionar la opción de cerrar la tarea, a través del botón creado expresamente para esta función y, que se mostró en la explicación de la implementación. Al cerrar se deben pasar los test, incrementar el número de versión del proyecto y, enviar el código al repositorio en remoto. Por último, se debe cerrar la rama. En caso de no superar los test, se aborta el cierre y se debe mostrar un mensaje de error al usuario.

**Pasos realizados:**

Lo primero fue comprobar que no es posible cerrar una tarea si antes no se ha realizado un *commit*. Al pulsar el botón de la función de cierre, se mostró un mensaje de error, obteniendo el comportamiento esperado. Para continuar, se procedió a crear el *commit* haciendo uso de la plantilla. Se comprueba, además, que la plantilla muestra correctamente el identificador numérico de la tarea y el nombre del tipo de tarea que se está a punto de finalizar. A continuación, nuevamente se hace el intento de cerrar la tarea, y esta vez sí da inicio al cierre. Se ve cómo pasan los test con éxito, se incrementa el número de versión del proyecto y se envía el código a remoto. Obtenido el comportamiento deseado por esta parte de la implementación, se procedió también a comprobar el caso en el que falla algún test. En esta parte se obtuvo un mensaje de error y se abortó el cierre de la rama, no se incrementó el número de versión, no se envió el código a remoto y se mostró un mensaje de error.

### 6.1.6. Se puede iniciar una tarea de tipo *release*

**Objetivo:**

Esta prueba debe proceder de igual forma que el inicio de las tareas de tipo *release*, *bugfix* y *hotfix*. La única diferencia es que el nombre de la rama tiene que estar formado por la palabra *release* seguido del siguiente número de versión del proyecto. Es decir, se debe poder obtener el número actual, incrementarlo y utilizar el número obtenido como parte del nombre de la rama.

**Pasos realizados:**

Se utiliza el botón creado para esta funcionalidad y dar paso al proceso de inicio de este tipo de tarea. Se comprueba, con éxito, que se crea una rama para dar soporte al control de versiones con el formato esperado.

### 6.1.7. Se puede finalizar una tarea de tipo *release*

**Objetivo:**

A través del botón creado para esta función, se espera realizar el cierre de la tarea de tipo *release*. Se debe generar un *commit*, pasar los test y, a continuación, incrementar el número de versión, enviar el código a remoto y cerrar la rama.

**Pasos realizados:**

Los pasos realizados para esta prueba son los mismos que los efectuados para el cierre de las tareas *feature*, *bugfix* y *hotfix*. En lo único que difiere es en el tipo de envío de código a remoto. Dado que este tipo de tarea se realiza para obtener una versión del proyecto para producción, el código se debe enviar a través de una petición *pull-request*. Con esta petición el código no se incorpora a desarrollo, sino que se hace una petición de incorporación a producción. De esta forma el código queda pendiente de ser aprobado por el responsable del repositorio, el *Product Owner*. Esta prueba se logra con éxito.

# Capítulo 7

# Impresiones sobre los resultados obtenidos

Se destacan algunos de los resultados obtenidos, ya que han resultado de interés, gracias a que, con la implementación de las herramientas, se ha logrado crear un marco integrado de desarrollo de *software* para la empresa TbM. Estos resultados, se apreciaron después de realizar las pruebas descritas en el capítulo 6, *Pruebas*.

Este marco integrado, facilita trabajar con las tareas del panel de *Scrum* en el IDE Pycharm y, automatiza el control de versiones para el desarrollo del sistema. Debido a esto se consigue mantener también, una historia limpia y coherente sobre las incorporaciones de cambios al desarrollo del sistema TbM System.

Al disponer de las tareas en el entorno de desarrollo y crear las ramas con la misma identificación que en el panel de *Scrum*, la aplicación Taiga hace un seguimiento de las actividades. De esta forma, en el propio panel de *Scrum*, al seleccionar una tarea sobre la que ya se ha trabajado, se muestra la descripción de los *commits* creados, ordenados por fecha y hora. Cada descripción de *commit* muestra al desarrollador que llevó a cabo la tarea, e incluso, si se ha cambiado el propietario de la tarea para que otro miembro del equipo continúe con el trabajo, se identifica también este usuario en la historia de la tarea y se indica, además, su aportación realizada.

Con esto se ha logrado mantener, la información de cómo evoluciona el proyecto ligada a su tarea en el panel de *Scrum*. De esta manera cualquier miembro del equipo puede consultar de forma ordenada el trabajo realizado y saber qué funcionalidad se ha desarrollado en esa tarea. Se muestra un fragmento de la información de la historia de desarrollo de una tarea en la Figura 7.1.

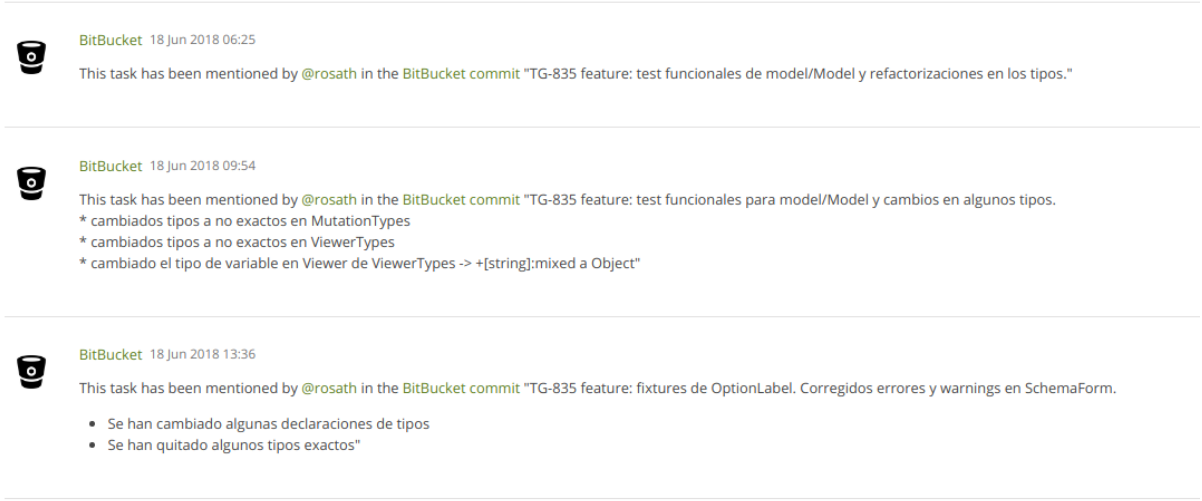



Figura 7.1: Parte de la historia de una tarea en el panel de *Scrum* Taiga.
*Imagen obtenida con permiso de la empresa TbM.*

Otra forma de poder ver la historia de desarrollo del proyecto es a través del servicio *Bitbucket*. En donde se muestra la información de las acciones realizadas desde el ámbito del control de versiones. La historia ahora es mucho más clara: se muestran las ramas creadas por cada tipo de tarea realizada, identificadas con el nombre del tipo y numeración de la tarea, gracias a la plantilla que se incorporó en la implementación, además de la descripción de los *commits*. También es posible consultar las modificaciones realizadas en el código a partir de los *commits* generados. Esto hace que un desarrollador al consultar un *commit* pueda saber en qué tarea se realizó la modificación, por quién, leer la descripción para entender el motivo y visualizar la información. Además, dado que se han integrado las herramientas, también es posible acceder directamente al *commit* de Bitbucket desde el servicio Taiga. Los enlaces que permiten acceder a visualizar el *commit*, se muestran en la Figura 7.1, y son, *@Rosa* y *Bitbucket commit*. En la Figura 7.2, se muestra un fragmento de cómo se han generado los *commits* para el DSS sobre estrategias financieras en desarrollo de la empresa TbM.

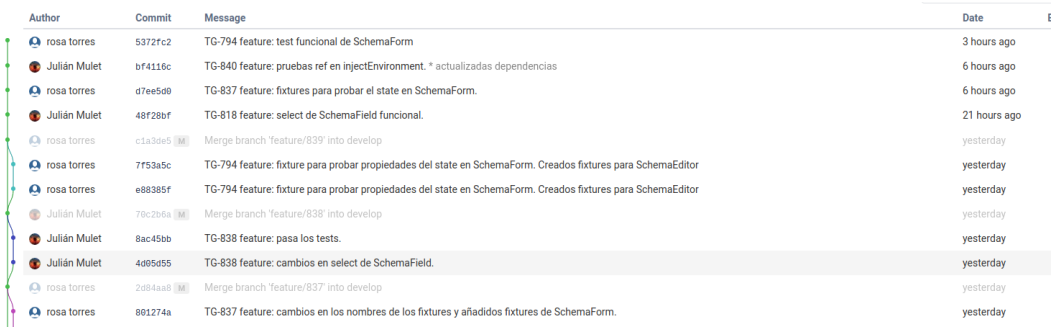



Figura 7.2: Parte de la historia del DSS sobre estrategias financieras en desarrollo en Bitbucket.
*Imagen obtenida con permiso de la empresa TbM.*

# Capítulo 8

# Conclusiones

En este proyecto se ha aplicado metodologías y conceptos aprendidos durante el grado, en concreto en las asignaturas EI1029 Sistemas de Información en las Organizaciones, EI1037 Gestión de Proyectos de Sistemas de Información y EI1046 Sistemas de Información Integrados. Los conocimientos que se han podido aplicar son: *Decision Support System* (DSS), desarrollo mediante Scrum,*Business Process Reengineering* (BPR), *Business process management* (BPM) y *Business Process Model and Notation* (BPMN). Los cuales han permitido realizar el estudio del proceso presentado en este proyecto, representarlo y proponer mejoras.

Para la implementación de las mejoras propuestas en la realización del proyecto, se han podido aplicar numerosos conocimientos adquiridos durante estos años en el grado, como por ejemplo, poder realizar la programación para añadir la nueva funcionalidad, conceptos de sistemas operativos aplicados a la lógica de la programación realizada, o lograr el entendimiento para ampliar y adquirir nuevos conceptos como ha sido el modelo de ramificación Git-flow, o la documentación de las *Application Programming Interface* (API) para la integración de las herramientas.

Sobre el resultado obtenido: la nueva funcionalidad añadida al entorno de desarrollo, la configuración añadida al panel de Scrum y la automatización del control de versiones; aprovecho para comentar, el valor añadido que ha supuesto al proceso de desarrollo del sistema que está llevando a cabo la empresa *Trading by Machine*. Tras las prácticas me incorporé como trabajadora a la empresa. El poder crear y finalizar tareas desde el entorno de desarrollo y, automatizar el control de versiones, ha propiciado que me pueda abstraer de estas acciones; apenas se producen conflictos al realizar aportaciones de código, se ha ganado en tiempo, y se tiene toda la información sobre cómo se realiza el desarrollo, de forma clara y ordenada, lo que hace fácil seguir el trabajo realizado por cualquier miembro del equipo.

Como ampliación a este proyecto, queda la integración de comunicación de notificaciones. En un principio, parte de la propuesta de este proyecto abarcaba la implementación de una herramienta de comunicación para el envío de notificaciones. Si bien, se realizó la selección de posibles aplicaciones para incorporar al flujo de trabajo, la evaluación de las aplicaciones seleccionadas y, se obtuvo como resultado una aplicación a integrar; no se incorporó finalmente al flujo de trabajo. Esto se debió a diversos motivos: la restricción de tiempo para las prácticas y

a que, la empresa consideró que en su estado actual, al ser pocos miembros, era prescindible, por lo que se decidió no realizar la implementación para llevar a cabo el resto de mejoras propuestas. Por esta razón, se deja como extensión a este trabajo la posibilidad de integrar una herramienta de comunicación de notificaciones.

Sobre la estancia en prácticas, desde el primer día me trataron como integrante más de la empresa. Me mostraron todas las herramientas con las que se trabaja y me explicaron también, cómo están desarrollando el sistema TbM System. Durante toda la estancia no me ha faltado soporte por parte del supervisor, quién si algún día no se pudo presentar en la oficina, se preocupó por contactar mediante Skype para supervisar el trabajo que estaba realizando. Sobre la planificación de los *Sprints* y el desarrollo mediante *Scrum*, al principio me resultó costoso. Además, mis tareas para realizar el proyecto, se añadían junto con las tareas a realizar para el desarrollo del sistema. Sin embargo, a medida que pasaron las semanas, esta manera de trabajar, resultó más favorable, en todo momento se tenían presentes las actividades a realizar, y además, al cerrar tareas se tenía sensación de avanzar sobre el trabajo que se estaba realizando.

Quisiera aprovechar, para expresar mi agradecimiento a mi supervisor Julián Mulet, por el interés y dedicación prestados para que aprendiera y para que la estancia en prácticas resultase provechosa; a mi tutora, Cristina Campos, por los consejos, conocimientos y correcciones realizados en este proyecto. A los profesores y profesoras del Grado en Ingeniería Informática, por las enseñanzas transmitidas durante el grado.

Agradecer también, a mi compañero Rafael Jurado, por haberlo sido, y por ponerme en contacto con la empresa donde hice la estancia en prácticas. A Aglaya Pons, por haber sido tan buena compañera de grupo durante estos últimos cursos.

Por último, agradecer a todas mis amistades de Castellón e Ibiza, por todo el apoyo prestado, sobre todo al final de la carrera. Y finalmente, a mi marido, Diego Garzón, por el ánimo y apoyo recibido, que no ha sido poco, durante estos años.

# Bibliografía


[1] Joaquin Guiral Rafael Lapiedra, Carlos Devece. *Introducción a la gestión de sistemas de información en la empresa*. Publicacions de la Universitat Jaume I, Campus del Riu Sec. Edifici Rectorat i Serveis Central, 1207 Castelló de la Plana, 2011.

[2] Sinnaps. Metodología scrum. `https://www.sinnaps.com/blog-gestion-proyectos/metodologia-scrum`. [fecha de consulta: 10 de junio, 2018].

[3] ProyectosAgiles.org. Qué es scrum. `https://proyectosagiles.org/que-es-scrum/`. [fecha de consulta: 10 de junio, 2018].

[4] Ken Schwaber y Jeff Sutherland. La guía de scrum. `http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf`. [fecha de consulta: 10 de junio, 2018].

[5] La enciclopedia libre Wikipedia. Historias de usuario. `https://es.wikipedia.org/w/index.php?title=Historias_de_usuario&oldid=104901942`. [fecha de consulta: 10 de junio, 2018].

[6] ProyectosAgiles.org. Lista de tareas de la iteración (sprint backlog). `https://proyectosagiles.org/lista-tareas-iteracion-sprint-backlog/`. [fecha de consulta: 10 de junio, 2018].

[7] La enciclopedia libre Wikipedia. Reingeniería de procesos. `https://es.wikipedia.org/wiki/Reingenier%C3%ADa_de_procesos`. [fecha de consulta: 26 de junio, 2018].

[8] Rajesh Ray. *Enterprise resource planning : text cases / Rajesh Ray.* New Delhi [etc.] : Tata McGraw-Hill Education, cop. 2011, 2011.

[9] Nathaniel Palmer. What is bpm? `https://bpm.com/what-is-bpm`. March 26, 2014 [fecha de consulta: 10 de junio, 2018].

[10] Camunda. bpmn.io. `https://bpmn.io/`. [fecha de consulta: 8 de febrero, 2018].

[11] Camunda. Bpmn 2.0 symbol reference. `https://camunda.com/bpmn/reference/`. [fecha de consulta: 8 de febrero, 2018].

[12] La enciclopedia libre Wikipedia. Control de versiones. `https://es.wikipedia.org/w/index.php?title=Especial:Citar&page=Control_de_versiones&id=108573911`. [fecha de consulta: 10 de junio, 2018].

[13] Empezando - fundamentos de git. `https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git`, [fecha de consulta: 8 de febrero, 2018].

[14] 3.1 git branching - branches in a nutshell. https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell. [fecha de consulta: 9 de junio, 2018].

[15] Fundamentos de git - guardando cambios en el repositorio. https://git-scm.com/book/es/v1/Fundamentos-de-Git-Guardando-cambios-en-el-repositorio. [fecha de consulta: 8 de febrero, 2018].

[16] Taiga Agile. Taiga. https://taiga.io/. [fecha de consulta: 9 de abril del 2018].

[17] Atlassian. Jira software. https://es.atlassian.com/software/jira. [fecha de consulta: 5 de marzo, 2018].

[18] Atlassian. Trello. https://trello.com/home, [fecha de consulta: 5 de marzo, 2018].

[19] Scrumdo. https://www.scrumdo.com/. [fecha de consulta: 5 de marzo, 2018].

[20] Quickscrum. https://www.quickscrum.com/. [fecha de consulta: 5 de marzo, 2018].

[21] Taiga. Taiga rest api. https://taigaio.github.io/taiga-doc/dist/api.html. [fecha de consulta: 30 de abril, 2018].

[22] Python community. python-taiga. https://pypi.org/project/python-taiga/. [fecha de consulta: 30 de abril, 2018].

[23] Vincent Driessen. A successful git branching model. https://nvie.com/posts/a-successful-git-branching-model/. [fecha de consulta: 10 de abril, 2018].

[24] Jeff Kreeftmeijer. Using git-flow to automate your git branching workflow. https://jeffkreeftmeijer.com/git-flow/. [fecha de consulta: 10 de abril, 2018].

[25] Atlassian. Gitflow workflow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. [fecha de consulta: 11 de abril, 2018].

[26] Daniel Kummer. git-flow cheatsheet. https://danielkummer.github.io/git-flow-cheatsheet/. [fecha de consulta: 11 de abril, 2018].

[27] Jan Gatting. Git commit message plugin. https://plugins.jetbrains.com/plugin/10100-git-commit-message-plugin. [fecha de consulta: 20 de abril, 2018].

[28] Slack. https://slack.com/intl/es-es. [fecha de consulta: 8 de febrero, 2018].

[29] Atlassian. Stride. https://www.stride.com/. [fecha de consulta: 6 de marzo, 2018].

[30] Flock. https://flock.com. [fecha de consulta: 6 de marzo, 2018].

[31] fleep. https://fleep.io/. [fecha de consulta: 7 de marzo, 2018].

[32] Ryver. https://ryver.com/. [fecha de consulta: 7 de marzo, 2018].

[33] Jean Manzo. Api vs webhooks. https://medium.com/@JManzoSistemas/api-vs-webhooks-4745bffcfa65. [fecha de consulta: 9 de junio, 2018].

[34] bpmn. Bpmnposter. http://www.bpmb.de/index.php/BPMNPoster. [fecha de consulta: 8 de febrero, 2018].

[35] Vincent Driessen. A successful git branching model. https://nvie.com/posts/a-successful-git-branching-model/. [fecha de consulta: 30 de abril, 2018].

[36] Peter van der Does. What is git-flow avh edition. https://github.com/petervanderdoes/gitflow-avh/wiki. [fecha de consulta: 30 de abril, 2018].

[37] Alfonso. Aprende git. http://aprendegit.com/tag/git-flow/. [fecha de consulta: 9 de abril, 2018].

# Anexo A

# *Sprints* planificados para la realización del proyecto

Se presentan los *Sprints* llevados a cabo durante la duración del proyecto. La información de los *Sprints* se ha extraído del panel de *Scrum* Taiga y se ha introducido en este trabajo con permiso de la empresa *Trading on Machine*. Para cada *Sprint* se ha especificado el período de duración, las historias de usuario creadas y, las tareas derivadas de cada historia de usuario. En total han sido siete *Sprints*. Pueden verse de la Tabla A.1 a la Tabla A.7.

Tabla A.1: *Sprint* 1. Historia de usuario y tareas realizadas.

| | |
|---|---|
| ***Sprint* 1** | Familiarizarse con el flujo de trabajo. |
| **Período** | 30 enero - 9 febrero. |
| **Historia de usuario** | Analizar flujo de trabajo panel de Scrum e interacción con entorno IDE. |
| **Tareas** | Dar de alta cuenta de Github, Bitbucket, Slack TbM y Taiga.<br>Crear tareas para historia de usuario de añadir flujo de trabajo Scrum + IDE.<br>Familiarizarse panel de Scrum Taiga.<br>Familiarizarse flujo actual de trabajo IDE.<br>Investigar recursos de trabajo con slack.<br>Crear cuenta para tomar licencia del IDE y usar correo corporativo.<br>Tarea de prueba para slack #prueba. |

Tabla A.2: *Sprint* 2. Historias de usuario y tareas realizadas.

| | |
|---|---|
| ***Sprint* 2** | Evaluar herramientas para el flujo de trabajo. |
| **Período** | 12 febrero - 22 febrero. |
| **Historia de usuario** | Búsqueda de herramientas para el flujo de trabajo panel de scrum. |
| **Tareas** | Investigar otras herramientas para incorporar al flujo de trabajo. |
| **Historia de usuario** | Analizar Jira como gestor de proyectos y comparar con solución actual. |
| **Tareas** | Crear cuenta de Jira y realizar pruebas. |
| **Historia de usuario** | Analizar Trello como gestor de proyectos y comparar con solución actual. |
| **Tareas** | Crear cuenta de Trello y realizar pruebas. |

Tabla A.3: *Sprint* 3. Historias de usuario y tareas realizadas.

| | |
|---|---|
| ***Sprint* 3** | Estudiar herramientas para integrar el canal de comunicación para el flujo de trabajo, estudio de git-flow para control de versiones. |
| **Período** | 6 marzo - 23 marzo. |
| **Historia de usuario** | Buscar herramientas para canal de comunicación para el flujo de trabajo. |
| **Tareas** | Realizar búsqueda de herramientas para canal de comunicación. |
| **Historia de usuario** | Estudiar Stride para canal de comunicación. |
| **Tareas** | Crear cuenta en Stride y realizar pruebas. |
| **Historia de usuario** | Estudiar Flock para canal de comunicación. |
| **Tareas** | Crear cuenta Flock y realizar pruebas. |
| **Historia de usuario** | Estudiar Fleep para canal de comunicación. |
| **Tareas** | Crear cuenta en Fleep y realizar pruebas. |
| **Historia de usuario** | Estudiar Ryver para canal de comunicación. |
| **Tareas** | Crear cuenta en Ryver y realizar pruebas. |
| **Historia de usuario** | Estudiar flujo de trabajo estándar git-flow. |
| **Tareas** | Búsqueda y lectura del flujo de trabajo estándar git-flow.<br>Documentar cómo se trabaja con git-flow.<br>Crear repositorio en bitbucket para realizar pruebas usando git-flow. |

Tabla A.4: *Sprint* 4. Historias de usuario y tareas realizadas.

| | |
|---|---|
| ***Sprint* 4** | Realizar matrices de evaluación para las herramientas a incorporar al flujo de trabajo, y realizar pruebas con los plugins de Pycharm-GitFlow y Pycharm-Taiga. |
| **Período** | 26 marzo - 13 abril. |
| **Historia de usuario** | Crear matriz de evaluación para las herramientas para gestionar el proyecto. |
| **Tareas** | Establecer los criterios que se van a evaluar para las herramientas.<br>Introducir los criterios para evaluar las herramientas.<br>Puntuar los criterios de cada herramienta. |
| **Historia de usuario** | Analizar uso del plugin para conexión entre IDE y Taiga. |
| **Tareas** | Realizar pruebas del plugin de conexión IDE-Taiga y analizar si cubre todos los casos. |
| **Historia de usuario** | Instalar y estudiar funcionalidad del plugin Git-Flow para Pycharm. |
| **Tareas** | Instalar plugin Git-Flow en Pycharm.<br>Estudiar funcionalidad plugin. |

Tabla A.5: *Sprint* 5. Historias de usuarios y tareas realizadas.

| | |
|---|---|
| ***Sprint* 5** | Automatizar flujo diario de git. |
| **Período** | 16 abril- 27 abril. |
| **Historia de usuario** | Start Feature: sincroniza con remoto, usar Git flow para crear Feature de Develop. |
| **Tareas** | Codificar script Start Feature de git-flow para el IDE. |
| **Historia de usuario** | Inicio de jornada de desarrollo. |
| **Tareas** | Estudiar documentación y pruebas con Interactive Rebase desde rama pública.<br>Codificar script de "Inicio de jornada en Feature". |
| **Historia de usuario** | Finish Feature. |
| **Tareas** | Codificar script Finish Feature. |
| **Historia de usuario** | Analizar funcionalidad git flow publish para estudiar su interés de cara a integrarlo en el flujo de trabajo. |
| **Tareas** | Búsqueda y lectura de documentación sobre Publish. Realización de pruebas si resulta de interés. |
| **Historia de usuario** | Estudiar el uso de Release para crear scripts de trabajo. |
| **Tareas** | Estudio y pruebas sobre el uso de la rama Release. |
| **Historia de usuario** | Estudiar el uso de Hotfix para crear los scripts de trabajo. |
| **Tareas** | Estudio y pruebas del uso de ramas Hotfix. |

Tabla A.6: *Sprint* 6. Historias de usuarios y tareas realizadas.

| | |
|---|---|
| ***Sprint* 6** | Terminar flujo git TbM. |
| **Período** | 30 abril - 11 mayo. |
| **Historia de usuario** | Inicio de jornada: borra rama release si está huérfana. |
| **Tareas** | Borra la rama release si está huérfana.<br>Explorar API de Taiga. |
| **Historia de usuario** | Start Release: número de versión actual +0.1 y truncando el tercer número a 0, ejemplo desde 1.3.5 => 1.4.0, crear y publicar tag a 1.4.0-rc. |
| **Tareas** | Realizar start release desde rama develop.<br>Establecer el tag version con el formato: 0.1 y realizar publish release.<br>Establecer tag para versión de release candidate con el format "vx.x.x.-rc". Y realizar push de tags. |
| **Historia de usuario** | Start Bugfix: usando git flow. |
| **Tareas** | Start rama bugfix desde release mediante comandos git flow. Realizar git flow publish de la rama bugfix creada. |
| **Historia de usuario** | Finish Bugfix: si pasa los tests, crear y publicar tag añadiendo 0.0.1 al último rc, ejemplo 1.4.0-rc => 1.4.1-rc, usar git flow finish y push desde release. |
| **Tareas** | Realizar la ceremonia de inicio de jornada, pasar los test y establecer el tag con el formato "vx.x.x.-rc".<br>Finalizar rama bugfix mediante comando git flow. Enviar a remoto el tag y realizar push de la rama bugfix. |
| **Historia de usuario** | Finish Release: si pasa los test, crear y publicar tag, ultimo rc sin el rc, ejemplo: 1.4.1.rc => 1.4.1, actualizar tag_version, hacemos pull request a master y fusionar sobre develop. |
| **Tareas** | Realizar inicio de jornada, pasar los test y realizar pull request hacia master.<br>Establecer tag con formato "vx.x.x" a partir del último tag que tendrá formato "vx.x.x-rc" y enviar a remoto el tag.<br>Cambiar a rama develop, fusionar la rama release en master y realizar push desde develop. |
| **Historia de usuario** | Start Hotfix: usar git flow. |
| **Tareas** | Crear rama hotfix desde master, y realizar push mediante comandos git flow. |
| **Historia de usuario** | Finish Hotfix: si pasa los tests, crear y publicar tag, ultimo de master +0.0.1, ejemplo: 1.4.1 => 1.4.2, actualizar tag_version, hacemos pull request a master y fusionar sobre develop, borramos rama hotfix. |
| **Tareas** | Realizar inicio de jornada, pasar los test y realizar pull request hacia master.<br>Incrementar el tag actual en master: 0.0.1 y subir el tag a remoto.<br>Cambiar a rama develop, fusionar la rama hotfix en master, eliminar la rama hotfix y realizar push. |

Tabla A.7: *Sprint* 7. Historias de usuarios y tareas realizadas.

| | |
|---|---|
| ***Sprint* 7** | Dar soporte de Taiga para el branching. |
| **Período** | 14 mayo - 23 mayo. |
| **Historia de usuario** | Establecer y configurar en Taiga los issues: tienen type, severity y priority. |
| **Tareas** | Establecer los tipos de issues y asociarlos a colores y configurarlo en Taiga.<br>Establecer los tipos de severidad y asociarlos a colores y configurarlo en Taiga.<br>Establecer los tipos de prioridad y asociarlos a colores y configurarlo en Taiga. |
| **Historia de usuario** | El menú de tareas de taiga tiene 4 categorías: Sprint tasks, Enhancements, bugfixes y hotfixes. |
| **Tareas** | Crear menú principal.<br>Crear submenú para bugfixes.<br>Crear submenú para Sprint tasks.<br>Crear submenú para Enhancements.<br>Crear submenú para hotfixes.<br>Ajustar y copiar plantilla de commit a project files. |
| **Historia de usuario** | Los items de taiga cerrados no se muestran. |
| **Tareas** | No mostrar los items cerrados de taiga. |
| **Historia de usuario** | Los sprint tasks crean ramas feature en git. |
| **Tareas** | Al seleccionar una task del menú, se crea su rama feature a través del script para git. |
| **Historia de usuario** | Los enhancements crean ramas feature en git. |
| **Tareas** | Al seleccionar un enhancement del menú, se crea una rama feature a través del script para git. |
| **Historia de usuario** | Los bugfix crean ramas bugfix en git. |
| **Tareas** | Al seleccionar un bugfix del menú, se crea su rama bugfix a través del script para taiga. |
| **Historia de usuario** | Las tareas del sprint están agrupadas en historias de usuario: Sprint tasks => User Stories => Tasks. |
| **Tareas** | Realizar el submenú para mostrar las historias de usuario de un sprint.<br>Crear el submenú para mostrar las tareas de una historia de usuario. |
| **Historia de usuario** | Las tareas, en el menú, muestran su estado actual. |
| **Tareas** | Mostrar en el menú el estado actual de las tareas. |
| **Historia de usuario** | Los issues, en el menú, muestran su prioridad y su estado actual. |
| **Tareas** | Mostrar en el menú el estado y la prioridad de lo issues. |
| **Historia de usuario** | Los issues, se muestran ordenados por prioridad. |
| **Tareas** | Mostrar en el menú los issues por orden de prioridad. |
| **Historia de usuario** | Cuando seleccionamos un elemento del menú sea tarea o issue, crea su rama git y cambia es estado a "in progress". |

Tabla A.8: Continuación Tabla A.7. *Sprint* 7.

| | |
|---|---|
| **Tareas** | Al seleccionar un enhancement del menú, se crea una rama feature a través del script para git. |
| **Historia de usuario** | Los bugfix crean ramas bugfix en git. |
| **Tareas** | Al seleccionar un bugfix del menú, se crea su rama bugfix a través del script para taiga. |
| **Historia de usuario** | Las tareas del sprint están agrupadas en historias de usuario: Sprint tasks => User Stories => Tasks. |
| **Tareas** | Realizar el submenú para mostrar las historias de usuario de un sprint.<br>Crear el submenú para mostrar las tareas de una historia de usuario. |
| **Historia de usuario** | Las tareas, en el menú, muestran su estado actual. |
| **Tareas** | Mostrar en el menú el estado actual de las tareas. |
| **Historia de usuario** | Los issues, en el menú, muestran su prioridad y su estado actual. |
| **Tareas** | Mostrar en el menú el estado y la prioridad de lo issues. |
| **Historia de usuario** | Los issues, se muestran ordenados por prioridad. |
| **Tareas** | Mostrar en el menú los issues por orden de prioridad. |
| **Historia de usuario** | Cuando seleccionamos un elemento del menú sea tarea o issue, crea su rama git y cambia es estado a "in progress". |
| **Tareas** | Al seleccionar una tarea, llamar al script de start_feature y cambiar su estado a "in progress".<br>Al seleccionar un issue llamar al script correspondiente para crear su rama y cambiar el estado a "in progress". |
| **Historia de usuario** | Cuando cerramos una rama feature, bugfix o hotfix pasa su entidad en Taiga a "closed". |
| **Tareas** | Al seleccionar la opción de cerrar una rama de tipo feature, bugfix o hotfix desde el QuickMenú, se lanza su script correspondiente y se cambia el estado de la tarea a "closed". |
| **Historia de usuario** | Los hotfix crean ramas hotfix en git. |
| **Tareas** | Al seleccionar un hotfix se lanza el script de start_hotfix. |
| **Historia de usuario** | Crear menú con iconos en el IDE para realizar las acciones. |
| **Tareas** | Mostrar iconos para las acciones en la barra menú del IDE. |
| **Historia de usuario** | Realizar pruebas sobre flujo de trabajo en el IDE. |
| **Tareas** | Clonar proyecto tbm-fronted.<br>Crear documentación con los casos que deben pasar las pruebas.<br>Al realizar las acciones mediante "menú scrum" se comprueba la funcionalidad del flujo de trabajo. |

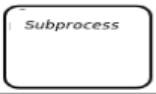
# Anexo B

# Elementos BPMN2 y ampliación de figuras

## B.1. Elementos de la notación BPMN2 usados para la representación del proceso

Se muestran en la Tabla B.1 los elementos de la notación BPMN utilizados para realizar las representaciones gráficas del proceso estudiado en el estado *AS IS* y *TO BE* en el Capítulo 4. *BPR aplicado al proceso de producción de software*. La información completa sobre los elementos existentes en la notación BPMN puede encontrarse en [11], [34]. Las figuras utilizadas para la creación de la Tabla B.1 se han extraído de la herramienta Camunda Modeler [10].

Tabla B.1: Descripción de los elementos BPMN utilizados en las representaciones.

| Elemento | Descripción | Notación |
|---|---|---|
| *Pool* | Contenedor de los flujos de secuencia entre actividades. |   |
| *Subprocess* | Es una actividad en la cual se han representado los elementos internos. |   |
| *Start event* | Indica el comienzo de un proceso. |  |
| *End event* | Indica dónde termina un proceso. |  |
| *Message Start Event* | Se usa cuando llega un mensaje de un participante y desencadena el inicio del proceso. |  |
| *Exclusive Gateway* | Para crear opciones de decisión exclusivas. |  |
| *User task* | Tareas de usuario realizadas de forma manual mediante un *software*. |  |
| *Service task* | Tareas realizadas de forma automática a medida que se ejecuta el proceso. |  |
| *Receive task* | Tara para indicar que a partir de la llegada de un mensaje, se modela como una tarea separada. |  |
| *Send task* | Tarea para enviar un mensaje a un participante externo. |  |
| *Sequence Flow* | Para mostrar el flujo de las actividades en el proceso. |  |
| *Message Flow* | Para mostrar el flujo de mensajes entre dos entidades que están preparadas para enviar y recibir. |  |

## B.2. Ampliación de figuras *AS IS* y *TO BE*

Para facilitar la visualización de los elementos utilizados para las representaciones *AS IS* y *TO BE* se muestra, a continuación, el *pool Product Owner* común a ambas figuras, el *pool Developer Team* del modelo *AS IS* y el *pool Developer Team* del modelo *TO BE*. Debe entenderse que en las siguientes figuras se han eliminado algunos *pools* y mensajes de flujo, debido a que el objetivo es mostrar partes de las figuras ampliadas para facilitar su lectura.

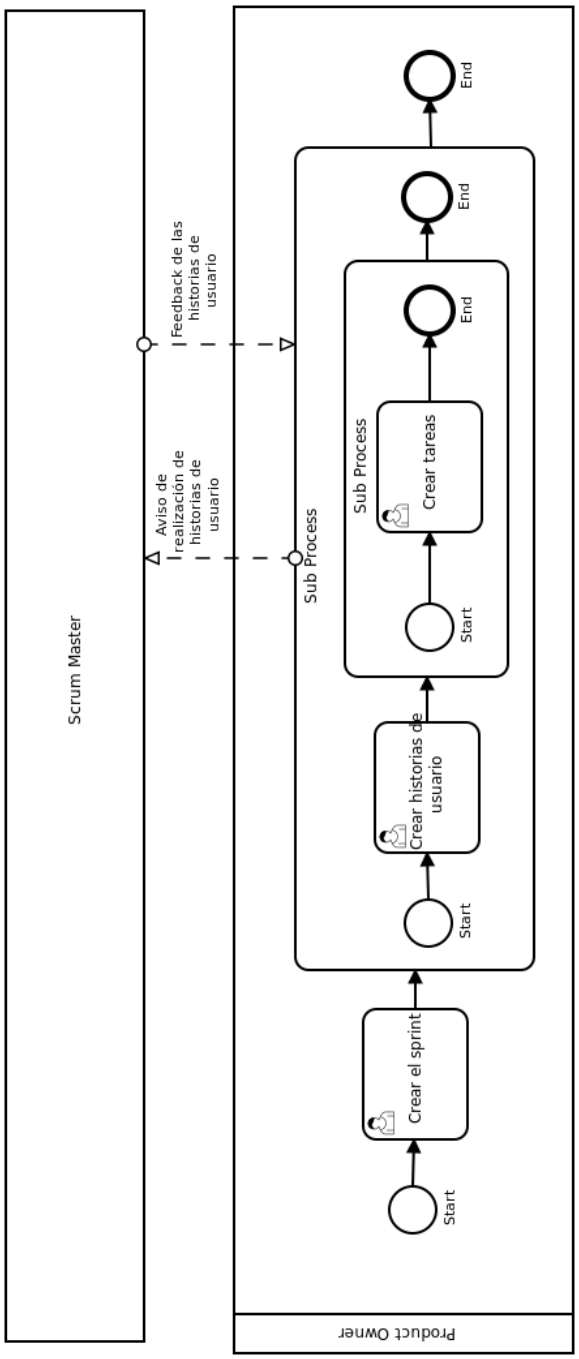



Figura B.1: *Pool Product Owner* del modelo *AS IS* y *TO BE*.

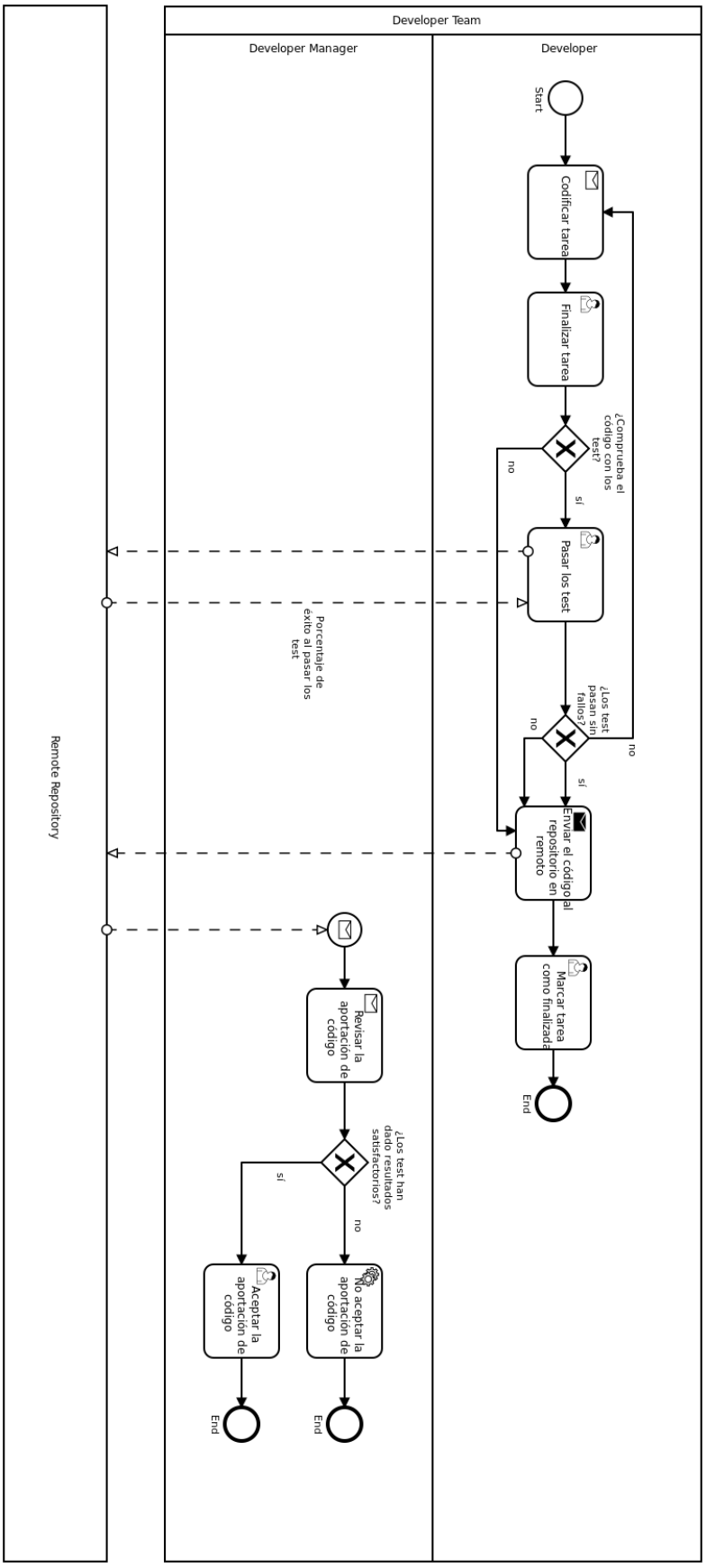



Figura B.2: *Pool Developer Team* del modelo *AS IS*.

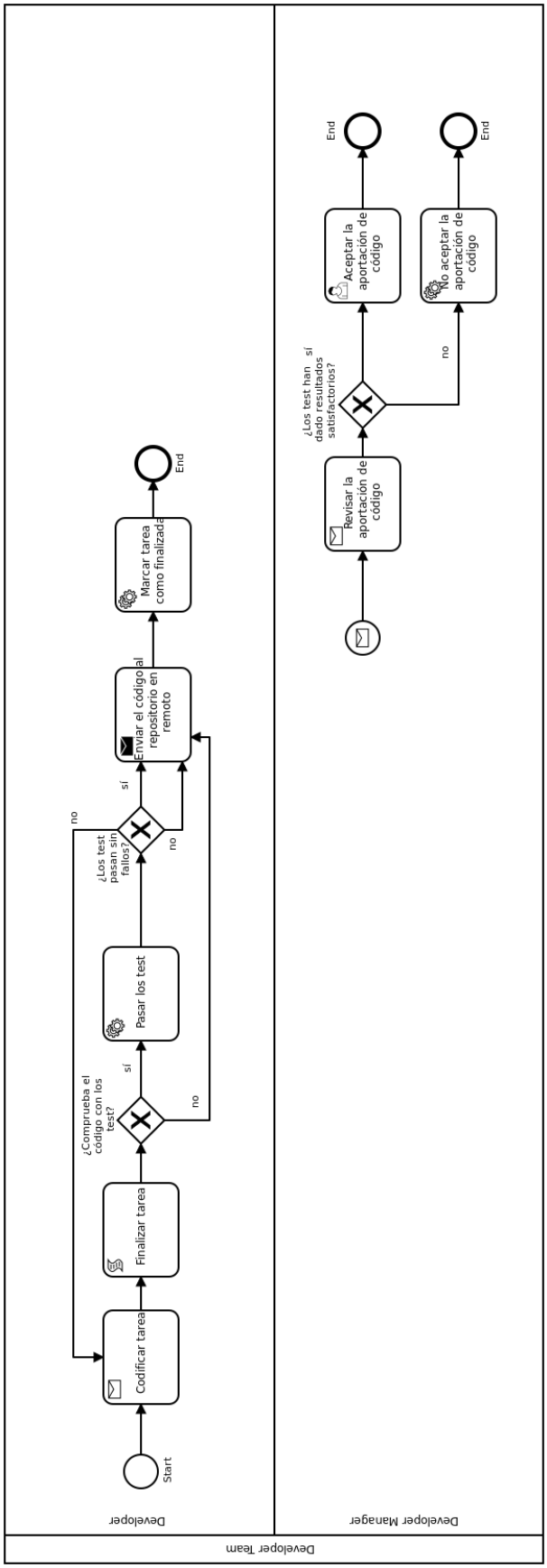



Figura B.3: *Pool Developer Team* del modelo *TO BE*.

# Anexo C

# Evaluación de las herramientas seleccionadas

## C.1. Evaluación de los paneles de *Scrum*

Se presenta a continuación, la evaluación completa realizada para la selección del panel de *Scrum*. Los criterios evaluados han sido: criterios económicos, generales, tecnológicos, de adaptabilidad y funcionales.

### C.1.1. Resultados de los criterios económicos

En la C.1, se muestran los criterios económicos evaluados, así como la puntuación obtenida por cada aplicación. La escala de valores utilizada para la evaluación de estos criterios se proporciona en la Tabla C.2.

Tabla C.1: Resultados de los criterios económicos para los paneles de *Scrum*.

| Criterios económicos | Taiga | | | Jira | | Trello | |
|---|---|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso | Información | Peso |
| **Tipo de precio** | 30% | - | 3,00 | Mensual | 1,00 | Anual | 2,00 |
| **Precio** | 35% | Free (1) | 2,00 | 10.00$ | 1,00 | 9.99$ (2) | 0,00 |
| **Número de usuarios** | 35% | 3u. | 1,00 | 10u. | 1,00 | 1u. | 0,00 |
| **Total** | 100% | | **1,95** | | **1,00** | | **0,60** |

**Nota.** (1) Para 1 proyecto privado.
(2) Precio por usuario/mes si se paga anualmente.

| Criterios económicos | ScrumDo | | | QuickScrum | |
|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso |
| **Tipo de precio** | 30% | Mensual | 1,00 | Mensual | 1,00 |
| **Precio** | 35% | 8.99$ | 1,00 | 3.00$ | 1,00 |
| **Número de usuarios** | 35% | 10u. | 1,00 | 1u. | 0,00 |
| **Total** | 100% | | **1,00** | | **0,65** |

Tabla C.2: Escala de valores para la evaluación de los criterios económicos.

| **Rango de precios** | **Peso** |
|---|---|
| Free | 2,00 |
| 1$ - 5$ mes | 1,00 |
| > 5$ mes | 0,00 |
| **Tipo de precio** | **Peso** |
| - | 3,00 |
| Anual | 2,00 |
| Mensual | 1,00 |
| **Núm. de usuarios** | **Peso** |
| >= 3u. | 1,00 |
| < 3u. | 0,00 |

### C.1.2. Resultados de los criterios generales

Como criterios generales se escogieron características comunes a casi todas las aplicaciones y, que podrían ser de interés para el caso del proceso estudiado. En la Tabla C.3 puede verse el resultado de la evaluación de estos criterios. Cabe mencionar, que para el criterio *Crear rama en repositorio de git*, aunque ScrumDo y Jira, ofrezcan esta posibilidad, no se les otorgó la misma puntuación. Esto ha sido debido a que Jira, permite la creación de ramas con Bitbucket, y este es el servicio empleado en la empresa, mientras que para ScrumDo, se hubiese tenido que migrar el repositorio a GitHub.

Otro valor a destacar, es si la aplicación resulta intuitiva. En este caso a Taiga, se le concedió una puntuación mayor. Esto fue debido a que, en comparación con el resto de aplicaciones, esta presenta una interfaz más limpia e instintiva, por lo que se quiso destacar esta característica.

La escala de valores empleada para cada criterio se presenta en la Tabla C.4.

Tabla C.3: Resultados de los criterios generales para los paneles de *Scrum*.

| Criterios generales | Taiga | | | Jira | | Trello | |
|---|---|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso | Información | Peso |
| **Intuitivo** | 6% | Muy alto | 3,00 | Medio | 1,00 | Alto | 2,00 |
| **Establecer rol de usuario** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Crear nuevos tipos de roles de usuario** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Creación de incidencias** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Poder crear varias tareas a la vez** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Aplicación móvil** | 5% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Aplicacion web adaptada a móvil** | 2% | No | 0,00 | Sí | 0,50 | Sí | 1,00 |
| **Documentación/Soporte** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Creación de wiki** | 1% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Registro de auditoría** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Importación del proyecto** | 7% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Exportación del proyecto** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Adjuntar archivos** | 3% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Control de Releases** | 2% | No | 0,00 | Sí | 1,00 | No | 0,00 |
| **Realización de informes** | 5% | No | 0,00 | Sí | 1,00 | No | 0,00 |
| **Crear tareas a través del correo electrónico** | 1% | No | 0,00 | Sí | 1,00 | Sí | 1,00 |
| **Notificaciones** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Establecer fechas de vencimiento para las tareas** | 7% | Sí | 1,00 | No | 0,00 | Sí | 1,00 |
| **Crear rama en repositorio de git** | 5% | No | 0,00 | sí, Bitbucket | 1,00 | No | 0,00 |
| **Total** | 100% | | **0,97** | | **0,92** | | **0,51** |

| Criterios generales | ScrumDo | | | QuickScrum | |
|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso |
| **Intuitivo** | 6% | Medio | 1,00 | Alto | 2,00 |
| **Establecer rol de usuario** | 7% | Sí | 1,00 | Sí | 1,00 |
| **Crear nuevos tipos de roles de usuario** | 7% | Sí | 1,00 | No | 0,00 |
| **Creación de incidencias** | 7% | No | 0,00 | No | 0,00 |
| **Poder crear varias tareas a la vez** | 7% | No | 0,00 | Sí | 1,00 |
| **Aplicación móvil** | 5% | No | 0,00 | No | 0,00 |
| **Aplicacion web adaptada a móvil** | 2% | Sí | 1,00 | No | 0,00 |
| **Documentación/Soporte** | 7% | Sí | 1,00 | Sí | 1,00 |
| **Creación de wiki** | 1% | No | 0,00 | No | 0,00 |
| **Registro de auditoría** | 7% | No | 0,00 | No | 0,00 |
| **Importación del proyecto** | 7% | Sí | 1,00 | No | 0,00 |
| **Exportación del proyecto** | 7% | Sí | 1,00 | No | 0,00 |
| **Adjuntar archivos** | 3% | Sí | 1,00 | Sí | 1,00 |
| **Control de Releases** | 2% | No | 0,00 | Sí | 1,00 |
| **Realización de informes** | 5% | Sí | 1,00 | Sí | 1,00 |
| **Crear tareas a través del correo electrónico** | 1% | No | 0,00 | No | 0,00 |
| **Notificaciones** | 7% | Sí | 1,00 | Sí | 1,00 |
| **Establecer fechas de vencimiento para las tareas** | 7% | Sí | 1,00 | No | 0,00 |
| **Crear rama en repositorio de git** | 5% | Sí, Github | 0,50 | No | 0,00 |
| **Total** | 100% | | **0,68** | | **0,50** |

Tabla C.4: Escala de valores utilizada para la evaluación de los criterios generales.

| **Intuitivo** | **Peso** |
|---|---|
| Muy alto | 3,00 |
| Alto | 2,00 |
| Medio | 1,00 |
| Bajo | 0,00 |
| **Sorportado** | **Peso** |
| Alto | 3,00 |
| Medio | 2,00 |
| Bajo | 1,00 |
| Muy bajo | 0,50 |
| No | 0,00 |

### C.1.3. Resultados de los criterios tecnológicos

Se recogen, en la Tabla C.5, los resultados alcanzados para las aplicaciones para los criterios tecnológicos. La escala de valores empleada para obtener los resultados se facilita en la Tabla C.6.

Tabla C.5: Resultados de los criterios tecnológicos para los paneles de *Scrum*.

| Criterios tecnológicos | Peso | Taiga | | Jira | | Trello | |
|---|---|---|---|---|---|---|---|
| | | Información | Peso | Información | Peso | Información | Peso |
| **Servicio Cloud** | 15% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Conexión con otras aplicaciones** | 15% | Github, Bitbucket, Slack, E-mail | 4,00 | Bitbucket, Github, Slack, E-mail | 4,00 | Bitbucket, Github, Slack, E-mail | 4,00 |
| **API pública** | 15% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Documentación para la API** | 15% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Lenguaje proporcionado en la documentación para la API** | 15% | Python | 3,00 | Java | 2,00 | Python | 3,00 |
| **Proyectos privados (o públicos)** | 20% | Privado | 1,00 | Privado | 1,00 | Privado | 1,00 |
| **Almacenamiento** | 5% | 300MB | 2,00 | 25GB | 3,00 | 25GB | 3,00 |
| **Total** | 100% | | **1,80** | | **1,70** | | **1,85** |

| Criterios tecnológicos | Peso | ScrumDo | | QuickScrum | |
|---|---|---|---|---|---|
| | | Información | Peso | Información | Peso |
| **Servicio Cloud** | 15% | Sí | 1,00 | Sí | 1,00 |
| **Conexión con otras aplicaciones** | 15% | Github, Slack | 2,00 | Bitbucket, Github | 2,00 |
| **API pública** | 15% | Sí | 1,00 | No | 0,00 |
| **Documentación para la API** | 15% | Sí | 1,00 | No | 0,00 |
| **Lenguaje proporcionado en la documentación para la API** | 15% | - | 0,00 | - | 0,00 |
| **Proyectos privados (o públicos)** | 20% | Privado | 1,00 | Privado | 1,00 |
| **Almacenamiento** | 5% | - | 0,00 | - | 0,00 |
| **Total** | 100% | | **0,95** | | **0,65** |

Tabla C.6: Escala de valores utilizada para la evaluación de los criterios tecnológicos.

| **Conexión con otras aplicaciones** | **Peso** |
|---|---|
| Slack | 1,00 |
| Bitbucket | 1,00 |
| GitHub | 1,00 |
| E-mail | 1,00 |
| **Total** (1) | 4,00 |
| **Documentación de la API** | **Peso** |
| Sí | 1,00 |
| No | 0,00 |
| **API pública** | **Peso** |
| Sí | 1,00 |
| No | 0,00 |
| **Lenguaje** | **Peso** |
| Python | 3,00 |
| Java | 2,00 |
| - (2) | 0,00 |
| **Almacenamiento** | **Peso** |
| > 5GB | 3,00 |
| 300MB - 5GB | 2,00 |
| 0 - 300MB | 1,00 |
| - (2) | 0,00 |

**Nota.** (1) Se aplica el valor de la suma de las aplicaciones que admite.
(2) Valor desconocido.

### C.1.4. Resultados de los criterios de adaptabilidad

Se presentan los criterios de adaptabilidad evaluados en la Tabla C.7. La mayoría de estos criterios son importantes, ya que permitirán obtener el comportamiento esperado por la aplicación. Esto se ha reflejado en el peso concedido a cada criterio. Por otro lado, a características menos importantes, como personalizar el aspecto del tablero, se les ha otorgado una puntuación menor. La escala de valores utilizada para las puntuaciones se puede ver en la Tabla C.8.

Tabla C.7: Resultados de los criterios de adaptabilidad para los paneles de *Scrum*.

| Criterios de adaptabilidad | Taiga | | | Jira | | Trello | |
|---|---|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso | Información | Peso |
| **Modificar/crear estados para epics** | 5% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar/crear estados para historias de usario** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar/crear estados para tareas** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar/crear estados para bugfix/hotfix/enhancements** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar/crear prioridades para bugfixes/hotfixes/enhancements** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar/crear severidad para bugfixes/hotfixes/enhancements** | 10% | Sí | 1,00 | No | 0,00 | No | 0,00 |
| **Crear/Modificar nuevos tipos de *issues*** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Crear/modificar tags** | 1% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Crear/configurar notificaciones** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Crear/modificar roles de usuario** | 10% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Modificar permisos según el tipo de rol** | 10% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Configurar el aspecto del tablero** | 6% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Personalizar el aspecto en general (cambiar fondo, colores, etc)** | 1% | No | 0,00 | Sí | 1,00 | Sí | 1,00 |
| **Total** | 100% | | **0,99** | | **0,90** | | **0,25** |

| Criterios de adaptabilidad | ScrumDo | | | QuickScrum | |
|---|---|---|---|---|---|
| | Peso | Información | Peso | Información | Peso |
| **Modificar/crear estados para epics** | 5% | No | 0,00 | Sí | 1,00 |
| **Modificar/crear estados para historias de usario** | 10% | No | 0,00 | Sí | 1,00 |
| **Modificar/crear estados para tareas** | 10% | Sí | 1,00 | Sí | 1,00 |
| **Modificar/crear estados para bugfix/hotfix/enhancements** | 10% | No | 0,00 | Sí | 1,00 |
| **Modificar/crear prioridades para bugfixes/hotfixes/enhancements** | 10% | No | 0,00 | Sí | 1,00 |
| **Modificar/crear severidad para bugfixes/hotfixes/enhancements** | 10% | No | 0,00 | Sí | 1,00 |
| **Crear/Modificar nuevos tipos de *issues*** | 10% | Sí | 1,00 | No | 0,00 |
| **Crear/modificar tags** | 1% | Sí | 1,00 | Sí | 1,00 |
| **Crear/configurar notificaciones** | 7% | Sí | 1,00 | No | 0,00 |
| **Crear/modificar roles de usuario** | 10% | Sí | 1,00 | No | 0,00 |
| **Modificar permisos según el tipo de rol** | 10% | No | 0,00 | No | 0,00 |
| **Configurar el aspecto del tablero** | 6% | Sí | 1,00 | No | 0,00 |
| **Personalizar el aspecto en general (cambiar fondo, colores, etc)** | 1% | No | 0,00 | No | 0,00 |
| **Total** | 100% | | **0,44** | | **0,56** |

Tabla C.8: Escala de valores utilizada para la evaluación de los criterios de adaptabilidad.

| Soportado | Peso |
|---|---|
| Sí | 1,00 |
| No | 0,00 |

### C.1.5. Resultados de los criterios funcionales

En la siguiente Tabla C.9, se presenta el resultado de la evaluación para los criterios funcionales. Durante la realización de las pruebas de evaluación de estos criterios, sorprende que, para las tareas de incidencia con las aplicaciones ScrumDo y QuickScrum, no fuese posible establecer la configuración para trabajar de la forma deseada. No llamó tanto al atención, el no lograrlo con Trello, ya que esta aplicación es más limitada.

Otro aspecto, que también llamó la atención, es la visualización de ScrumDo para las tareas. Así como el resto de aplicaciones presenta en el panel las historias de usuario y las tareas, ScrumDo sólo muestra las historias de usuario. Para visualizar las tareas es necesario acceder a la historia de usuario. Este comportamiento, resultó extraño, ya que se espera de este tipo de aplicaciones que las tareas estén disponibles visualmente. Este aspecto se reflejó con la puntuación concedida a ScrumDo de 0,50 puntos. La escala de valores para establecer las puntuaciones se puede ver en la Tabla C.10.

Tabla C.9: Resultados de los criterios funcionales para los paneles de *Scrum*.

| Criterios funcionales | Taiga | | | Jira | | Trello | |
|---|---|---|---|---|---|---|---|
| | Puntuación | Información | Peso | Información | Peso | Información | Peso |
| **Establecer roles de usuario** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Establecer equipo de trabajo** | 9% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Creación y visualización de historias de usuario** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Creación y visualización de tareas** | 9% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Conectividad con IDE Pycharm** | 4% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Conectividad con Bitbucket** | 3% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Conectividad con slack** | 3% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Seguimiento de issue (hotfix, bugfix, enhancement)** | 9% | Sí | 1,00 | sí (*2) | 0,50 | No | 0,00 |
| **Establecer prioridades para issue** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Establecer tipos de issue** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Estados de las tareas (new, in progress, completed...)** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Códigos para las tareas (cada tarea tiene un código único que la identifica)** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Creación de incidencias (*1)** | 9% | Sí | 1,00 | Sí | 1,00 | No | 0,00 |
| **Total** | 100% | | **1,00** | | **0,96** | | **0,24** |

**Nota.** (*1) Una incidencia puede ser de tipo bug, hotfix o enhancement. No es necesario que pertenezca a una historia de usuario. Cuando se detecta una incidencia se deja anotada en un listado.

(*2) Es necesario configurarlo para utilizarlo de la forma desesada.

| Criterios funcionales | ScrumDo | | | QuickScrum | |
|---|---|---|---|---|---|
| | Puntuación | Información | Peso | Información | Peso |
| **Establecer roles de usuario** | 9% | Sí | 1,00 | Sí | 1,00 |
| **Establecer equipo de trabajo** | 9% | Sí | 1,00 | Sí | 1,00 |
| **Creación y visualización de historias de usuario** | 9% | Sí | 1,00 | Sí | 1,00 |
| **Creación y visualización de tareas** | 9% | Sí y no (*3) | 0,50 | Sí | 1,00 |
| **Conectividad con IDE Pycharm** | 4% | No | 0,00 | No | 0,00 |
| **Conectividad con Bitbucket** | 3% | No | 0,00 | Sí | 1,00 |
| **Conectividad con slack** | 3% | Sí | 1,00 | No | 0,00 |
| **Seguimiento de issue (hotfix, bugfix, enhancement)** | 9% | No | 0,00 | Sí | 1,00 |
| **Establecer prioridades para issue** | 9% | No | 0,00 | Sí (*4) | 0,50 |
| **Establecer tipos de issue** | 9% | Sí | 1,00 | Sí (*4) | 0,50 |
| **Estados de las tareas (new, in progress, completed...)** | 9% | Sí | 1,00 | Sí | 1,00 |
| **Códigos para las tareas (cada tarea tiene un código único que la identifica)** | 9% | Sí | 1,00 | Sí | 1,00 |
| **Creación de incidencias (*1)** | 9% | No | 0,00 | No | 0,00 |
| **Total** | 100% | | **0,62** | | **0,75** |

**Nota.** (*3) Se crean las tareas pertenicientes a una historia de usuario, pero no se visualizan en el tablero. Para acceder a las tareas hay que abrir la historia de usuario.

(*4) Es posible definirlo.

Tabla C.10: Escala de valores utilizada para la evaluación de los criterios funcionales.

| Soportado | Peso |
|---|---|
| Alto | 1,00 |
| Medio | 0,50 |
| Bajo | 0,00 |

## C.2. Evaluación de las aplicaciones de comunicación

Se muestran los resultados de los criterios funcionales y generales evaluados, de los cuales se ha obtenido el resultado final presentado en el apartado 5.3.3. *Selección, evaluación e implementación* del canal de comunicación.

### C.2.1. Resultados de los criterios funcionales

Los criterios evaluados han sido el poder conectarse a las aplicaciones Bitbucket y Taiga, y el poder enviar una notificación a un determinado usuario. Los valores de los pesos, dado que las tres características son importantes, son similares. Aún así, se ha dado un peso menor al envío a un determinado usuario, ya que prevalece la necesidad de las conexiones con Bitbucket y Taiga. Las puntuaciones obtenidas para las aplicaciones pueden verse en la Tabla C.11, así como la escala de valores empleada en la Tabla C.12.

Tabla C.11: Resultados de los criterios funcionales para las aplicaciones de comunicación

| Criterios funcionales | | Slack | | Stride | | Flock | |
|---|---|---|---|---|---|---|---|
| | Peso | Información | Puntuación | Información | Puntuación | Información | Puntuación |
| **Conexión con Bitbucket** | 34% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Conexión con Taiga** | 34% | Sí (*1) | 1,00 | No | 0,00 | No | 0,00 |
| **Configurar el envio de un mensaje a un determinado usuario** | 32% | No | 0,00 | No | 0,00 | No | 0,00 |
| **Total** | 100% | | **0,68** | | **0,34** | | **0,34** |

**Nota.** (*1) Ofrece conexión con el servicio a través de *webhook*, y se consigue con éxito.

| Criterios funcionales | | Fleep | | Ryver | |
|---|---|---|---|---|---|
| | Peso | Información | Puntuación | Información | Puntuación |
| **Conexión con Bitbucket** | 34% | Sí | 1,00 | No (*2) | 0,00 |
| **Conexión con Taiga** | 34% | No (*2) | 0,00 | No (*2) | 0,00 |
| **Configurar el envio de un mensaje a un determinado usuario** | 32% | No | 0,00 | No | 0,00 |
| **Total** | 100% | | **0,34** | | **0,00** |

**Nota.** (*2) Ofrece realizar la conexión con el servicio a través de *webhook*, pero no se consigue con éxito.

Tabla C.12: Escala de valores utilizada para la evaluación de los criterios funcionales.

| Soportado | Peso |
|---|---|
| Sí | 1,00 |
| No | 0,00 |

### C.2.2. Resultados de los criterios generales

Se presenta a continuación, los criterios generales, evaluados para las aplicaciones seleccionadas. Se puede ver en la siguiente Tabla C.13, como estos criterios, siendo normalmente comunes a este tipo de aplicaciones, en algunos casos se les otorgó una baja puntuación, como el caso del formateo del texto para escribir mensajes con fragmentos de código. También, se presenta en la Tabla C.14 la escala de valores empleada.

Tabla C.13: Resultados de los criterios generales para las aplicaciones de comunicación

| Criterios generales | | Slack | | Stride | | Flock | |
|---|---|---|---|---|---|---|---|
| | Peso | Información | Puntuación | Información | Puntuación | Información | Puntuación |
| **Aplicación web** | 7% | Sí | 1,00 | No | 0,00 | Sí | 1,00 |
| **Crear canales/chats** | 6% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Modificar un mensaje** | 5% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Compartir archivos** | 5% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de grupos de chat** | 5% | - | 0,90 | Ilimitado | 1,00 | Ilimitado | 1,00 |
| **Cantidad de usuarios** | 5% | - | 0,90 | Ilimitado | 1,00 | Ilimitado | 1,00 |
| **Intuitivo** | 5% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Aplicación móvil** | 4% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de mensajes almacenados** | 4% | 10.000 | 0,90 | 25.000 | 1,00 | 10.000 | 0,90 |
| **Integraciones de aplicaciones y servicios** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de integraciones** | 7% | 10 | 1,00 | 10 | 1,00 | 5 | 0,80 |
| **Llamadas de voz y vídeo** | 6% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Máximos participantes en videollamadas** | 6% | 2 | 0,00 | 25 | 1,00 | 4 | 0,70 |
| **Compartir pantalla** | 6% | No | 0,00 | No | 0,00 | No | 0,00 |
| **Almacenamiento de archivos** | 4% | 5GB | 1,00 | 5GB | 1,00 | 5GB | 1,00 |
| **Ayuda** | 4% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Notificaciones** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Formateo para escribir mensajes con código** | 7% | Sí | 1,00 | Sí | 1,00 | Sí | 1,00 |
| **Total** | 100% | | **0,87** | | **0,87** | | **0,90** |

**Nota.** Las información con " -" es debido a que no viene especificado, pero dado que supera las pruebas realizadas, se otorga una puntuación de 0,90

| Criterios generales | | Fleep | | Ryver | |
|---|---|---|---|---|---|
| | Peso | Información | Puntuación | Información | Puntuación |
| **Aplicación web** | 7% | Sí | 1,00 | Sí | 1,00 |
| **Crear canales/chats** | 6% | Sí | 1,00 | Sí | 1,00 |
| **Modificar un mensaje** | 5% | Sí | 1,00 | Sí | 1,00 |
| **Compartir archivos** | 5% | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de grupos de chat** | 5% | - | 0,90 | - | 0,90 |
| **Cantidad de usuarios** | 5% | 200 por equipo | 0,90 | 6 | 0,50 |
| **Intuitivo** | 5% | Sí | 1,00 | Sí | 1,00 |
| **Aplicación móvil** | 4% | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de mensajes almacenados** | 4% | - | 0,90 | Ilimitado | 1,00 |
| **Integraciones de aplicaciones y servicios** | 7% | Sí | 1,00 | Sí | 1,00 |
| **Cantidad de integraciones** | 7% | - | 0,90 | Ilimitado | 1,00 |
| **Llamadas de voz y vídeo** | 6% | Sí | 1,00 | No | 0,00 |
| **Máximos participantes en videollamadas** | 6% | 8 | 0,90 | - | 0,90 |
| **Compartir pantalla** | 6% | Sí | 1,00 | No | 0,00 |
| **Almacenamiento de archivos** | 4% | 10GB | 1,00 | Ilimitado | 1,00 |
| **Ayuda** | 4% | Sí | 1,00 | Sí | 0,50 |
| **Notificaciones** | 7% | Sí | 1,00 | Sí | 0,50 |
| **Formateo para escribir mensajes con código** | 7% | No | 0,00 | No | 0,00 |
| **Total** | 100% | | **0,90** | | **0,72** |

Tabla C.14: Escala de valores utilizada para la evaluación de los criterios generales.

| **Soportado** | **Peso** |
|---|---|
| Sí | 1,00 |
| No | 0,00 |
| **Cantidad de grupos de chat** | **Peso** |
| Ilimitado | 1,00 |
| - | 0,90 |
| **Cantidad de usuarios** | **Peso** |
| Ilimitado | 1,00 |
| >= 50 por equipo | 0,90 |
| < 50 por equipo | 0,50 |
| - | 0,90 |
| **Cantidad de mensajes almacenados** | **Peso** |
| >= 25.000 | 1,00 |
| 24.999 - 10.000 | 0,90 |
| - | 0,90 |
| **Cantidad de integraciones** | **Peso** |
| Ilimitado | 1,0 |
| >= 10 | 1,0 |
| 5 - 10 | 0,8 |
| - | 0,9 |
| **Participantes en videollamadas** | **Peso** |
| >= 25 | 1,00 |
| - | 0,90 |
| 5 - 24 | 0,90 |
| 3 - 4 | 0,70 |
| <= 2 | 0,00 |
| **Almacenamiento de archivos** | **Peso** |
| Ilimitado | 1,00 |
| >= 5GB | 1,00 |
| < 5GB | 0,00 |

# Anexo D

# Git-flow. Modelo de ramificación

## D.1. Definición

Git-flow es un modelo de ramificación que estable cómo gestionar las ramas en un repositorio de Git. Este modelo, propuesto por Vincent Driessen en su blog *A successful Git branching model* [35], es de especial importancia para equipos de desarrollo. Explica cómo gestionar ramas a través del desarrollo del producto, de forma que, por ejemplo, se pueden crear ramas para introducir nuevas funcionalidades, lanzar una nueva versión del proyecto o solucionar fallos y problemas. De esta forma es posible adaptar la creación de las ramas al desarrollo real del producto.

Las acciones realizadas con Git-flow se pueden conseguir también mediante comandos Git. Es más, para utilizar Git-flow, se debe tener instalado Git. Pero cada acción conlleva una serie de comandos, por lo que es conveniente utilizar el conjunto de extensiones ofrecido por Git-flow para no tener que ejecutar cada vez todos esos comandos.

## D.2. Ramificación

Git-flow propone mantener durante todo el proyecto dos tipos de ramas principales ***master*** y ***develop***. Estas ramas tienen vida ilimitada durante todo el proyecto. La rama *master* será utilizada para producción y la rama *develop*, contendrá el código para la siguiente versión del proyecto, es decir, la versión en desarrollo del producto.

Para desarrollar nuevas funciones para una próxima publicación o futura versión, se crean las ramas ***feature***. Estas ramas sólo existen en el repositorio del proyecto en desarrollo, nunca en producción. Por tanto, tienen vida limitada, ya que son creadas para añadir funcionalidad y eliminadas al terminar la adición de esa funcionalidad. Dado que con estas ramas se trabaja en desarrollo, se crean (bifurcan) a partir de la rama *develop* y, una vez terminada la codificación de la nueva funcionalidad, se añade (fusiona) el código a la rama *develop*. De esta forma, se consigue trabajar de manera aislada, permitiendo que varios desarrolladores puedan trabajar al

mismo tiempo.

Cuando se desea preparar una nueva versión del proyecto de producción, se crearán ramas de tipo ***release***. Debido al objetivo de este tipo de rama, preparar una nueva versión, su ciclo de vida es muy corto. En estas ramas se deben realizar las pruebas para lanzar la nueva versión y generar muy poca cantidad de código. Si fuese necesario introducir grandes cambios, se debería retroceder, eliminar esta rama, y generar los cambios a través de ramas de tipo *feature*. Estas ramas se crean a partir de la rama *develop* (a la que se han ido incorporando las características a través de ramas *feature*, y por lo tanto tiene toda la funcionalidad para la nueva versión), además, se debe crear el nuevo número de versión del proyecto o *tag*. Al terminar, se fusiona con la rama *master*, de esta forma se incorpora el código a producción.

Para solucionar problemas y fallos, se deben crear ramas de tipo ***hotfix*** o *bugfix*[1].

Las ramas de tipo ***hotfix***, se utilizan para solucionar errores encontrados en producción, es decir, en la rama *master*. Por tanto, se deben crear a partir de la rama *master* y, una vez solucionado el error, fusionar el código en la rama *master* y *develop*. Al incorporar el código a la rama *master*, se debe aumentar el número de versión del proyecto.

Ramas de tipo ***bugfix***. Este tipo de ramas existe de forma similar a las de tipo *hotfix*, con la diferencia de solucionar fallos encontrados en *develop*. Una rama *bugfix* se crea a partir de la rama *develop* y, solucionado el error, se fusiona otra vez en la rama *develop*, incrementando, además, el número de versión del proyecto. El flujo propuesto por Git-flow puede verse en la figura D.1.

## D.3. Comandos

Los comandos presentados a continuación, son los distintos comandos de Git-flow para crear ramas y establecer los números de versión del proyecto. Cada comando Git-flow supone una secuencia de comandos de Git, por lo que también se muestra la secuencia de comandos equivalentes con el objetivo de entender, qué acciones se llevan a cabo con cada comando de Git-flow. Estos comandos han sido extraídos y adaptados a partir de: [26] [37].

### D.3.1. Inicializar Git-flow

Tabla D.1: Comandos para crear el repositorio.

| Git-flow | Git |
|---|---|
| `$ git flow init` | `$ git branch develop`<br>`$ git push -u origin develop` |

[1]La introducción de ramas *bugfix* al flujo de trabajo Git-flow, es una variante propuesta por Peter van der Does en [36], quien a su vez se basó en la definición de la metodología Git-flow de Vincent Driessen.

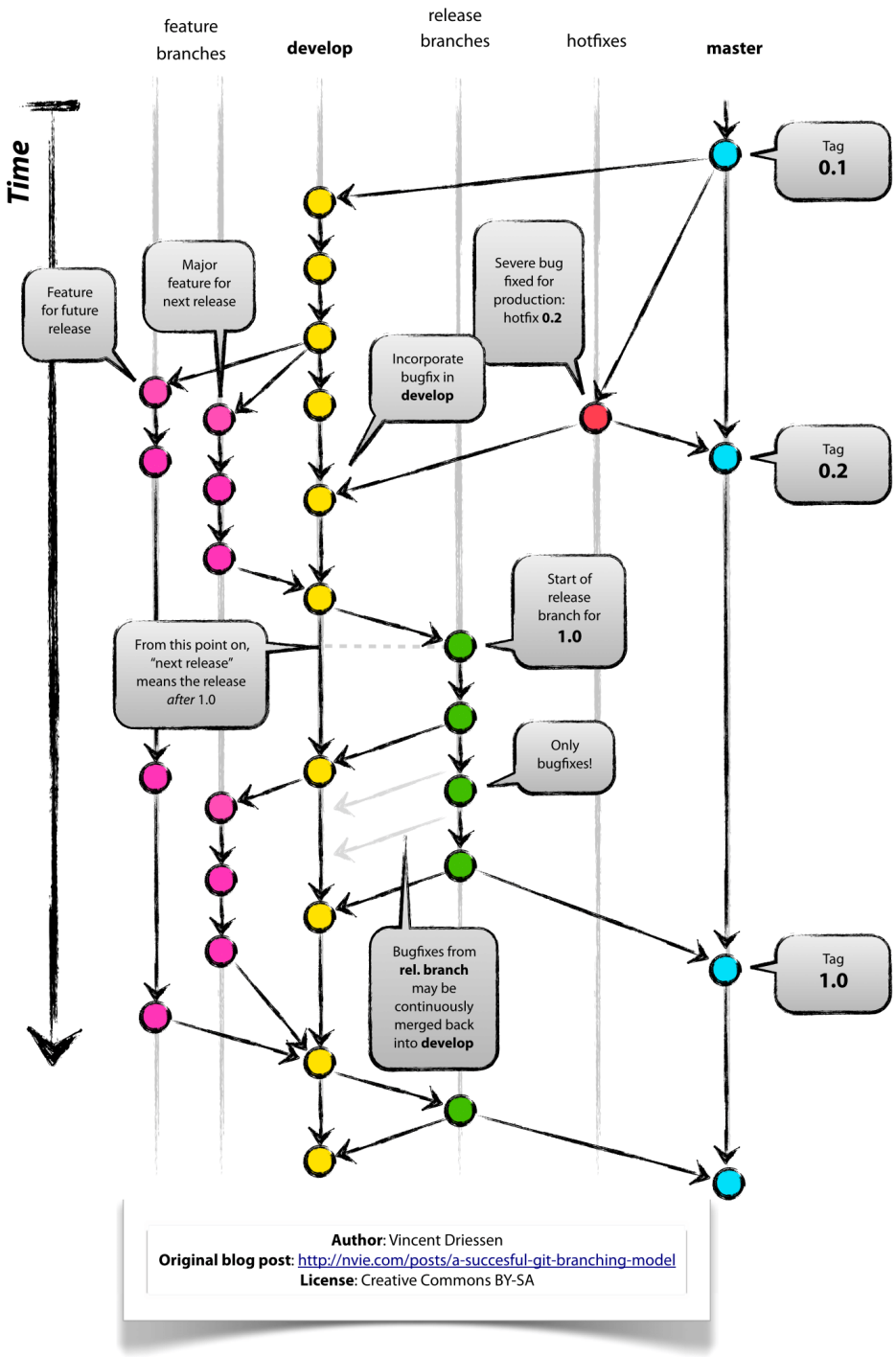



Figura D.1: Modelo de ramificación de Git.

### D.3.2. Rama *feature*

**Crear la rama *feature*:**

Tabla D.2: Comandos para crear una rama *feature*.

| Git-flow | Git |
|---|---|
| `$ git flow feature start branch_name` | `$ git checkout develop`<br>`$ git checkout -b branch_name` |

**Finalizar la rama *feature*:**

Tabla D.3: Comandos para finalizar una rama *feature*.

| Git-flow | Git |
|---|---|
| `$ git flow feature finish branch_name` | `$ git checkout develop`<br>`$ git merge branch_name`<br>`# opcional:`<br>`# git merge --no-ff branch_name`<br>`# --no-ff: siempre crea un commit`<br>`$ git branch -d branch_name`<br>`$ git push origin develop` |

### D.3.3. Rama *release*

**Crear la rama *release*:**

Tabla D.4: Comandos para crear una rama *release*.

| Git-flow | Git |
|---|---|
| `$ git flow release start 0.1.0` | `$ git checkout develop`<br>`$ git checkout -b release/0.1.0`<br>`$ ./bump-version.sh 1.2 (*1)` |

**Nota.** (*1) Ejemplo de comando para aumentar el número de versión a través de un *Script*.

**Finalizar la rama *release*:**

Tabla D.5: Comandos para finalizar una rama *release*.

| Git-flow | Git |
|---|---|
| `$ git checkout master`<br>`$ git checkout merge release/0.1.0`<br>`# opcional:`<br>`# git merge --no-ff release/0.1.0`<br>`$ git flow release finish 0.1.0` | `# se fusiona en la rama master`<br>`$ git checkout master`<br>`$ git merge --no-ff release/0.1.0`<br>`$ git tag -a 1.2`<br>`# se fusiona en la rama develop`<br>`$ git checkout develop`<br>`$ git merge release/0.1.0`<br>`# opcional:`<br>`# git merge --no-ff release/0.1.0`<br>`# borrar rama release:`<br>`$ git branch -d release/0.1.0` |

### D.3.4. Rama *hotfix*

**Crear la rama *hotfix*:**

Tabla D.6: Comandos para crear una rama *hotfix*.

| Git-flow | Git |
|---|---|
| `$ git flow hotfix finish hotfix_branch` | `$ git checkout master`<br>`$ git checkout -b hotfix_branch-0.1.1`<br>`$ ./bump-version.sh 0.1.1`<br>`$ git commit -a -m "Aumento de version"`<br>`$ git commit -m "Solucionado problema"` |

**Finalizar la rama *hotfix*:**

Tabla D.7: Comandos para finalizar una rama *hotfix*.

| Git-flow | Git |
|---|---|
| `$ git flow hotfix finish hotfix_branch` | `# Actualizar master y tag:`<br>`$ git checkout master`<br>`$ git merge hotfix_branch-0.1.1`<br>`$ git tag -a 0.1.1`<br>`# Incluir bugfix en develop: (*1)`<br>`$ git checkout develop`<br>`$ git merge hotfix_branch-0.1.1`<br>`# Eliminar rama hotfix:`<br>`$ git branch -D hotfix_branch-0.1.1` |

**Nota.** (*1) Si existe una rama *release*, se debe fusionar la rama *hotfix* en la de *release* en lugar de en la rama *develop*. El fusionar la rama *hotfix* con la de *release*, implica también el posterior fusionado con la rama *develop*, ya que la rama *release* se fusiona con la rama *develop* cuando esta ha finalizado.

### D.3.5. Rama *bufgix*

Dado que el flujo inicial de *git-flow* no contempla este tipo de rama, se presenta únicamente la secuencia de comandos Git.

**Crear la rama *bugfix*:**

Tabla D.8: Comandos para crear una rama *bugfix*.

| Git |
| --- |
| `$ git checkout develop`<br>`$ git checkout -b bugfix_branch-0.1.1`<br>`$ ./bump-version.sh 0.1.1`<br>`$ git commit -a -m "Aumento de version"`<br>`$ git commit -m "Solucionado problema"` |

**Finalizar la rama *bugfix*:**

Tabla D.9: Comandos para finalizar una rama *bugfix*.

| Git |
| --- |
| `# Actualizar develop y tag:`<br>`$ git checkout develop`<br>`$ git merge bugfix_branch-0.1.1`<br>`$ git tag -a 0.1.1`<br>`# Eliminar rama bugfix:`<br>`$ git branch -D bugfix_branch-0.1.1` |