CONTENIDO

TABLA DE IMÁGENES

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Los proyectos de software actuales no experimentan las mismas necesidades ni realidades que los proyectos de software de hace varios años. La evolución del mercado actual es mucho más rápida que en aquellos tiempos. Hoy en día, la mayoría de los productos que utilizan las personas regularmente, como por ejemplo, compra de productos, trámites, pago de servicios, correo, etc., disponen su propio sistema informático. Algunos específicamente destinados a un producto en particular, mientras que otros, de forma más hibrida, encargados de un bloque de productos.

Entonces, es casi imposible pensar que existe una organización que disponga el monopolio de algún rubro, comercio o cliente. Es normal en el mundo del software comercial, que exista competencia entre los productos de software más utilizados. Es por esto, que las organizaciones tienden a buscar características que les brinden una distinción en sus productos, para poder mantenerse en lo más alto del mercado. En repetidas oportunidades, esta distinción no es visible, o es difícil de demarcar. Por ejemplo, muchas organizaciones apuestan a la eficiencia de un plan de trabajo, en lugar de a una característica del producto.

Para poder destacarse en el mercado, la industria del software tiende a priorizar la agilidad y flexibilidad en el desarrollo, en lugar de los procesos largos y poco estructurados. Es decir, se construye el producto al mismo tiempo en que se modifican e introducen nuevos requisitos. Esto ocasiona que los procesos de gestión del proyecto también se vean modificados. Las organizaciones se ven en la obligación de utilizar estrategias que les permitan tomar decisiones correctas, con el objetivo de mantenerse en lo más alto del mercado. Es por esto que las herramientas de apoyo en la gestión del proyecto, pasaron a ocupar un rol fundamental en la administración del mismo.

1.2. OBJETIVOS

1.2.1. OBJETIVO GENERAL

Desarrollar una herramienta de software que asista en al seguimiento y documentación de las actividades que se llevan a cabo durante el proceso de gestión del proyecto, utilizando Trello como gestor de tareas de base y Github como repositorio.

Se analizarán las herramientas de gestión más utilizadas actualmente desde el punto de vista de la configuración inicial, la gestión de personal, administración, verificación de tareas y la posibilidad de vincular las historias de usuarios con los archivos en el repositorio. Detectando las falencias que presentan y proponiendo un conjunto de soluciones a los problemas encontrados. Agilizando el proceso de desarrollo y facilitando las tareas a los usuarios desarrolladores.

1.3. ESTRUCTURA DE LA TESINA

En el capítulo 2 se abordará todas las cuestiones relacionadas a la "Gestión de proyectos". Por ejemplo, procesos que se llevan a cabo, etapas que intervienen, nivel de complejidad, metodologías agiles y tradicionales, herramientas mayormente utilizadas, tipos de proyectos, estrategias de gestión del personal y del trabajo, etc. En el siguiente capítulo, se indicarán los problemas de los proyectos de software complejos, las actividades que se ven mayormente afectadas, el nivel de complejidad de un proyecto, cómo afecta dicho nivel al éxito del mismo, los problemas que genera al proceso de gestión, a qué se debe, etc. A continuación se describirán siete problemas detectados en dos de las herramientas de software más populares de gestión de proyecto. Analizado los problemas, en el capítulo 5 se presenta "TG – Project Management", describiendo sus funcionalidades y la solución a los problemas descriptos en el capítulo anterior. Finalmente, se presentarán las conclusiones y trabajos futuros de este trabajo.

2. GESTIÓN DE PROYECTOS

La gestión de proyectos de software es una parte esencial de la ingeniería de software. Es la aplicación de conocimiento, habilidades, herramientas y técnicas a las actividades necesarias para alcanzar los objetivos del proyecto. La buena gestión no puede garantizar el éxito. Sin embargo, la mala gestión usualmente lleva al fracaso. El software es entregado tarde, los costes son mayores que los estimados y los requerimientos no se cumplen. (Pablo Lledó y Gustavo Rivarola, 2007)

La administración eficiente del proyecto, implica la utilización de *procesos de gestión* durante todo el ciclo de vida del mismo. Dentro de los procesos que se llevan a cabo, podemos identificar dos tipos, administración general y administración de proyectos. Aunque ambos procesos son similares, no debemos confundirlos, pues se basan en supuestos diferentes. El proceso de administración general está pensado como un sistema de gestión de una organización, cuya duración es extensa y desconocida. La administración de proyectos se orienta, fundamentalmente, a gestionar emprendimientos de carácter finito y con objetivos específicos, los que una vez cumplidos determinan su finalización.

Tanto la administración general, como la administración de proyectos, se nutren de la planificación, organización y dirección de recursos –humanos y materiales–, con el objetivo de mantener bajo control la ejecución. Es por ello que el conocimiento de los procesos de administración general, es un fundamento necesario, aunque no suficiente, para asegurar una administración exitosa de los proyectos. (Pablo Lledó y Gustavo Rivarola, 2007)

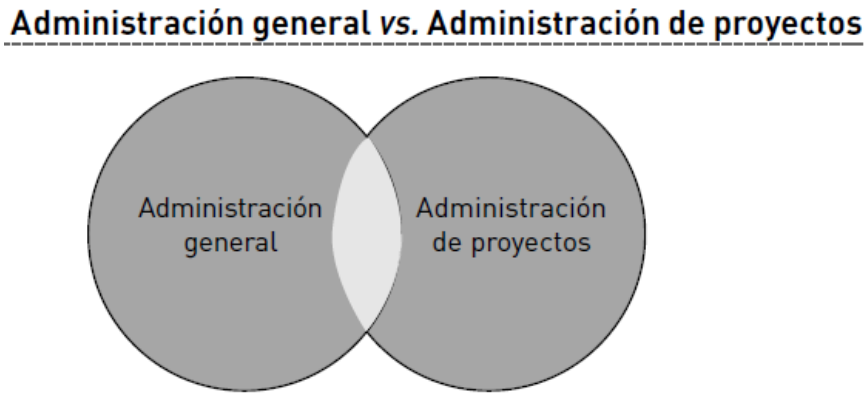

Ilustración 1 – Administración general vs Administración de proyectos

La dirección formal de un proyecto se compone de diferentes procesos de gestión (alcance, costos, recursos, cronograma, etc.), los cuales tiene como objetivos definir una metodología para conseguir los objetivos del proyecto y asegurar que éste se ejecute de acuerdo a lo definido. (Garriga, 2006). La administración eficiente del proyecto implica la utilización de *procesos de gestión* durante todo el ciclo de vida del proyecto. (Pablo Lledó y Gustavo Rivarola, 2007)

Los gestores de software son responsables de la planificación y temporización del desarrollo de los proyectos. Por lo tanto, supervisan el trabajo para asegurar que se lleve a cabo conforme a

los estándares requeridos y supervisan el progreso para comprobar que el desarrollo se ajusta al tiempo previsto y al presupuesto.

Hay varios tipos de proyectos de software, con una gama amplia de funcionalidades. Esta idea nos hace pensar que para diferentes proyectos son apropiados diferentes objetivos de desarrollo. Tratar de aplicar los mismos procesos de gestión a todos los proyectos de software es tan mala idea como tratar de aplicar *code-and-fix* para desarrollar cualquier tipo de proyecto (Schaull, 2011)

Las herramientas de administración de proyectos sirven para proporcionar a los miembros del equipo de trabajo la estructura, la flexibilidad y el control necesarios para alcanzar resultados extraordinarios a tiempo y dentro del presupuesto (Pablo Lledó y Gustavo Rivarola, 2007)

2.1. METODOLOGÍA DE GESTIÓN DE UN PROYECTO

Para que un proyecto resulte exitoso, es necesario que la administración del mismo sea eficiente. Para ello es fundamental la experiencia de las personas involucradas en los equipos de trabajo. La experiencia está ligada al conocimiento, las buenas prácticas y las actividades y procesos que han rendido buen resultado en un determinado contexto y que se espera que, en contextos similares, rindan similares resultados.

La metodología de gestión de un proyecto ha sido siempre un elemento clave para la administración del trabajo. Hace referencia al conjunto de todos los procedimientos utilizados para alcanzar el objetivo del proyecto, es decir, para gestionar nuestras actividades siguiendo unos requisitos y pasos, con el fin de encontrar rutas de trabajo optimizadas. La elección errónea de la metodología a utilizar, es uno de los principales factores que puede llevar a un proyecto al fracaso. Esta decisión es uno de los primeros grandes desafíos que tiene que cumplir un director de proyecto.

Una metodología debe reflejar el tamaño, la duración y la complejidad de cada proyecto individual, y adaptarse a la industria, la cultura organizacional y el nivel de madurez de gestión de proyectos de la organización. Una metodología puede ser amplia o mínima; rigurosa o sencilla; compleja o simple; lineal o altamente iterativa; descrita en fases o descrita para todo el ciclo de vida del proyecto. Todos los proyectos de software son diferentes, por lo tanto no es correcto utilizar la misma metodología de gestión a todos los proyectos.

No debe llamarse metodología a cualquier procedimiento. Se trata de un concepto que en la gran mayoría de los casos resulta demasiado amplio. Sin embargo, los directores de proyectos de hoy en día suelen basarse en dos metodologías principales, de las cuales posteriormente se derivan en varias "sub-metodologías" o sub procedimientos. Estas metodologías son la **metodología tradicional** o la **metodología ágil**. (Alexander Menzinsky, Gertrudis López, Juan Palacio, 2016)

2.1.1 - METODOLOGÍA TRADICIONAL

Dentro de la metodología tradicional, el desarrollo en cascada es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de *software*, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Una etapa es un conjunto de actividades del proyecto relacionadas entre sí y que, en general, finaliza con la entrega de un producto parcial o completo**.** Hay proyectos sencillos que sólo requieren de una etapa, y otros de gran complejidad que requieren más. A estas etapas en su conjunto se las denomina *ciclo de vida del proyecto*.

Con el objetivo de mantener el proyecto bajo control, los administradores del proyecto tienen la responsabilidad de predecir lo que pasará en la siguiente etapa, antes de terminar con la anterior. Entonces necesitan apoyarse en el uso de técnicas para minimizar errores y aumentar la eficacia del proceso de gestión del proyecto.(Almunia, 2016)

Si bien las etapas de cada proyecto en particular tienen similares nombres, todos los proyectos son distintos. La mayoría comprende cuatro o cinco etapas, pero algunos pueden tener varias fases adicionales en función del tamaño del emprendimiento.

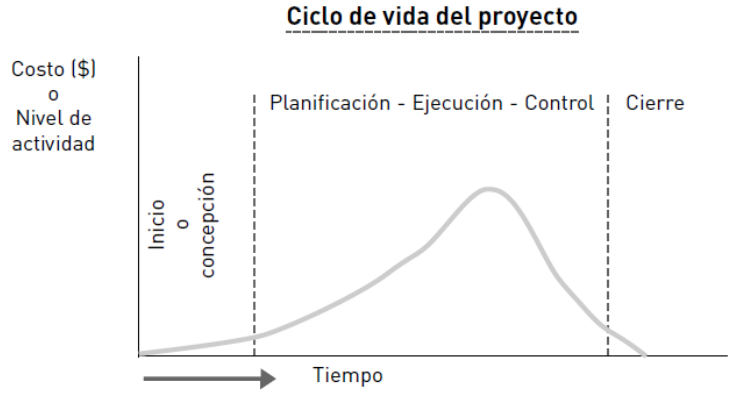

Ilustración 2 – Ciclo de vida del proyecto

Cada proyecto en particular podrá tener su propio ciclo de vida, que puede diferir de la Ilustración 2. (Pablo Lledó y Gustavo Rivarola, 2007)

El problema que presenta este tipo de metodología es que muchos proyectos software fracasan porque se planifica en profundidad al principio y se pretende reducir los cambios en fase de ejecución. El método de trabajo inflexible, orientado a cumplir exhaustivamente la planificación, hace que cuando estos grandes proyectos terminan, muchas veces el cliente no obtiene lo que esperaba, lo que suele explicarse porque una vez se supera la planificación, ya no hay comunicación eficaz con el cliente. (Barato, 2013)

2.1.2 - METODOLOGÍA ÁGIL

La forma ágil de trabajar consiste en lo siguiente: En vez de pasar mucho tiempo **planificando** un proyecto al principio, luego **ejecutando** el plan sin revisión ni comunicación frecuente con el cliente, se planifica una pequeña funcionalidad que se puede entregar rápidamente.

A diferencia de lo que ocurría con los métodos tradicionales, con los métodos ágiles (o adaptativos) se puede empezar el proyecto sin un plan completo. Dicho plan irá surgiendo poco a poco, según se avanza. En los métodos predictivos, donde se tiene un plan completo desde el principio, los cambios se toleran.

En los **métodos ágiles**, se comienza planificando una pequeña parte inicial, se asigna un equipo con las habilidades necesarias y comienza a trabajar. No se pierde tiempo esperando a que alguien termine algo en otra parte de la organización. Se desarrolla el elemento de software y se va probando según se avanza. Al final de una corta iteración, hay algo que poder enseñar al cliente, ya sea un prototipo o una demo. El cliente percibe un valor en esa entrega y transmite buena realimentación sobre la funcionalidad y la usabilidad. Después comienza una siguiente iteración incremental. (Barato, 2013)

Por lo tanto, gracias a centrar el punto de interés en la ejecución de proyectos y el foco en la productividad y en la entrega temprana, hizo que para la gestión de desarrollo de software sea más aceptada una metodología de tipo "ágil", como por ejemplo Scrum y Kanban. (Figuerola, Gestionando pequeños proyectos, 2015)

2.1.2.1 - SCRUM

Scrum es un modelo de desarrollo ágil caracterizado por:

1. Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
2. Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
3. Solapamiento de las diferentes fases del desarrollo, en lugar de realizarlas una tras otra en un ciclo secuencial o de cascada.

El modelo Scrum se caracteriza por el protagonismo de equipos brillantes, autos organizados y motivados. Abordan el desarrollo de sistemas complejos partiendo de una visión general y solapando las fases del desarrollo. Trabajan con autonomía, solapando las fases de desarrollo y compartiendo el conocimiento y aprendizaje de forma abierta.

Scrum está formado por un conjunto de prácticas y reglas que dan respuesta a los siguientes principios del desarrollo ágil:

1. Gestión evolutiva del producto, en lugar de la tradicional o predictiva.
2. Calidad del resultado basado en el ***conocimiento tácito de las personas***, antes que en el explícito de los procesos y la tecnología empleada.

3. Estrategia de ***desarrollo incremental a través de iteraciones*** (Sprints).

Tienen que dividir cada ciclo o iteración de trabajo en Sprints para producir cada incremento. (Schwaber, 1995)

2.1.2.2 - KANBAN

Kanban utiliza técnicas de representación visual de información para mejorar la eficiencia en la ejecución de las tareas representada por Scrum. Esto se debe a que uno de los principales problemas que se pueden observar, es que suelen formarse cuellos de botella que bloquean el avance de los desarrolladores produciendo tiempos muertos dentro de los Sprints.

El uso de tableros Kanban muestra y gestiona el flujo de avance y entrega, y ayuda a evitar los dos problemas más importantes: cuellos de botella y tiempos muertos. El desarrollo ágil de software emplea prácticas de gestión visual por ser las que mejor sirven a los principios de comunicación directa y simplicidad en la documentación y gestión.

Las prácticas Kanban son válidas para gestión evolutiva con entrega continua. Deben emplearse con criterios de flexibilidad, sin considerar prescripciones ni excepciones en el método de trabajo, para lograr la implementación personalizada, que dé la mejor respuesta a los principios ágiles o de síntesis de ambos, con los que trabaje la organización. (Figuerola, Kanban - Su uso en el desarrollo de software, 2016)

2.2. HERRAMIENTAS

Dentro del ciclo de vida de un proyecto se usan herramientas de distinta índole, empleadas para diferentes objetivos. Este informe se centra en la descripción de herramientas destinadas a la gestión de proyectos y en herramientas utilizadas para el control de versiones de código.

2.2.1 - HERRAMIENTAS PARA LA GESTIÓN DE PROYECTOS

Una de las ventajas de usar una herramienta de apoyo en la gestión de proyectos, es que esta puede enriquecer y potenciar el talento de los usuarios. Los gestores de proyectos asisten en el seguimiento del avance del proyecto, a gestionar las tareas del personal, a realizar la estimación de esfuerzo y cronograma y a gestionar la comunicación con el cliente. (Pfleeger, 2002)

Disponemos de un gran abanico de posibilidades para la elección de una herramienta destinada a la gestión de proyectos. Dos de las herramientas más populares en el mercado, son Trello y Jira.

2.2.1.1 -TRELLO

Una de las herramientas más conocidas para la gestión de proyectos es Trello. Trello provee un entorno de trabajo totalmente personalizable. Le permite a una organización definir sus propias reglas de negocio, métricas, acomodarse a las necesidades del proyecto y gestionar las

demandas del cliente. Trello dispone de una interfaz de usuario muy sencilla para su uso, logrando que los miembros de la organización se adapten rápidamente al uso de la herramienta. Otra característica importante de esta herramienta es que permite que todos los miembros de un equipo puedan planificar y supervisar las tareas a realizar. (Stella, 2010)

2.2.1.2 - JIRA

Muchas organizaciones optan por Jira para la gestión de sus proyectos. Jira provee un entorno de trabajo muy estructurado, con reglas ya definidas y poco flexible. La interfaz de usuario es muy compleja, requiere de una curva de aprendizaje medianamente elevada para aprender de su uso. Jira permite a la organización que todos los miembros de un equipo de software puedan planificar y supervisar las tareas a realizar. Provee un gran conjunto de métricas que ayudan a la organización a llevar el control de sus productos. (Atlassian, 2004)

2.2.2 - HERRAMIENTAS PARA EL CONTROL DE VERSIONES

Cuando no existe un control previo predefinido que administre el trabajo realizado desde el punto de vista del desarrollo, se dice que los programadores generan código espagueti. Por lo tanto, los desarrolladores tienen la necesidad de disponer de un sistema o de una estructura de control que apoye a la gestión del desarrollo del código.

Es usual que las organizaciones utilicen una metodología de "control de versiones" para administrar y controlar el desarrollo. Este sistema apoya a los programadores del proyecto a proponer soluciones de manera colaborativa, evitando tener conflictos entre ellos.

Existen numerosas tecnologías de control de versiones que son utilizadas por la comunidad desarrolladora. Este informe se centra en la explicación del sistema de control de versiones Git.

1.2.1 - GIT

Git es un sistema de control de versiones que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, permitiendo recuperar versiones específicas de en cualquier momento. Permite revertir archivos y también el proyecto a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más. (Straub, 2014)

Disponemos de un gran abanico de posibilidades en la elección de herramientas que implementan el sistema de control de versiones basándose en Git. Este informe centra su foco de atención en la descripción de GitHub.

1.2.2 - GITHUB

GitHub es una plataforma de **desarrollo colaborativo de software** que permite alojar proyectos utilizando el sistema de control de versiones Git.

GitHub es mucho más que un servicio de alojamiento de código, también ofrece varias herramientas útiles para el **trabajo en equipo**. Entre ellas, es necesario destacar:

- Una **wiki** para el mantenimiento de las distintas versiones de las páginas.
- Un **sistema de seguimiento de problemas** que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
- Una **herramienta de revisión de código**, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
- Un **visor de ramas** donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

El flujo de trabajo propuesto por GitHub consiste en la bifurcación (fork) de un repositorio por desarrollador. Cuando se hace un fork de un repositorio, se hace una copia exacta en crudo del repositorio original que luego podemos utilizar como un repositorio git cualquiera. El objetivo principal de los forks es permitirle a los desarrolladores contribuir a un proyecto de forma segura.

GitHub propone que los desarrolladores contribuyan o colaboren con el repositorio original a través de los Pull Request. Un Pull Request es una petición que el propietario de un fork hace al propietario del repositorio original para que este último incorpore los commits que están en el fork. Una vez abierto el Pull Request, se puede discutir y revisar los cambios potenciales con los colaboradores y agregar confirmaciones de seguimiento antes de que los cambios se fusionen en el repositorio. (GitHub, GitHub help, 2013)

3. GESTIÓN DE PROYECTOS COMPLEJOS Y DE AMPLIO ALCANCE

Para que un proyecto resulte exitoso, es necesario que la administración del mismo sea eficiente. Es fundamental la experiencia y la buena comunicación de las personas involucradas dentro de los equipos de trabajo. Los proyectos de software actuales no experimentan las mismas necesidades ni realidades que los proyectos de hace varios años. Ahora se prioriza más la agilidad y flexibilidad en el desarrollo, que los procesos estructurados de larga duración. Se construye el producto al mismo tiempo que se modifican e introducen nuevos requisitos. El cliente siempre parte de una visión medianamente clara, pero el nivel de innovación que requiere y la velocidad a la que se mueve el entorno de su negocio, no le permiten prever con detalle cómo será el resultado final. Es necesario actuar rápidamente, en lugar de llevar largas actividades de administración y control. Por esto las organizaciones se ven en la obligación de utilizar tácticas de gestión orientadas a la respuesta ágil y flexible, necesaria para trabajar en mercados de evolución rápida.

A medida que transcurre el ciclo de vida de un proyecto, todas estas características ocasionan que la complejidad de los proyectos aumente. Esto genera que las estrategias de gestión necesiten modificarse. Entonces el nivel de adaptación al cambio que disponga la organización, está atado al nivel de escalabilidad que posea. Esta escalabilidad define la capacidad que tiene para mejorar y crecer sin perder la calidad del producto.

La experiencia es un factor clave dentro de la administración del proyecto, el éxito del mismo depende de ella. El problema está en que la experiencia está atada directamente al contexto. Las buenas prácticas de ingeniería de software sobre gestión de proyectos, indican a que todas las actividades y procesos que hayan rendido buenos resultados en un determinado contexto, se espera que en contextos similares rindan de la misma manera. Aun así todos los proyectos de software son distintos. Entonces esta desigualdad ocasiona que no se pueda utilizar esta premisa durante todo el proceso de gestión, permitiendo aplicarla en partes muy puntuales del proyecto.

Aun así, existen varios atributos dentro del contexto de cualquier proyecto, en el que la experiencia todavía sigue cumpliendo un papel importante, el primero de ellos es la gestión del personal del proyecto.

3.1. GESTIÓN DEL PERSONAL DEL PROYECTO

Dentro de un proyecto de software participan usuarios de todo tipo, ya sea por estar vinculados directamente con el mismo, o por participar de forma indirecta en el proceso de desarrollo. Dentro de los usuarios que están vinculados directamente con el proyecto de software, es decir, los que aportan al producto final, se encuentran los desarrolladores, diseñadores, gerentes, administradores de producto, arquitectos de software, etc. Todos ellos comunicándose cotidianamente entre sí.

Los proyectos de software complejos, se encuentran en una constante evolución. Estos tipos de proyectos priorizan la agilidad sobre los procesos estructurados. Entonces, una técnica muy

utilizada dentro de la gestión de proyectos, es contar con un gran número de personal calificado, capacitado y con el objetivo de resolver ciertos problemas del producto. De esta forma, cada persona puede enfocarse en un trabajo específico sin tener la responsabilidad de cumplir el rol de distintos puestos. Con esto se pretende agilizar el desarrollo de producto, a consta de invertir en mano de obra.

El problema es que a mayor mano de obra, mayor es el control necesario. A su vez, no siempre es sencillo conseguir un gran número de personas capacitadas. Lo normal es que uno se forme dentro de la organización, haga carrera y aprenda de los compañeros. Entonces la falta de experiencia se combate con el apoyo de los más experimentados.

Todo esto se consigue entablando una buena comunicación dentro del proyecto.

3.1.1. VARIABLES QUE AFECTAN AL DESEMPEÑO INDIVIDUAL

Es necesario contar con personas predispuestas, interesadas y que sepan trabajar en equipo. El desempeño de una persona depende de muchos factores: Como se resume en la Ilustración 3, algunos de estos están relacionados con el individuo y otros se vinculan al entorno laboral. La interacción entre estos factores determinan el nivel de motivación y satisfacción de cada una de las personas que integran un equipo de proyecto y, consecuentemente, tendrá un impacto fundamental en sus desempeños individuales dentro del mismo.

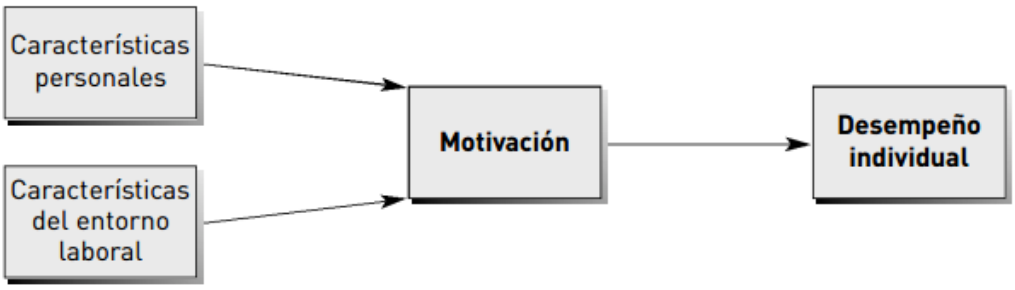

Ilustración 3 – Motivación

Una persona experimenta necesidades de diversa índole, tales como necesidades fisiológicas (por ejemplo, comer), necesidades sociales (trabajar en equipo), necesidades de estima (sentirse reconocido) y necesidades de autorrealización (ser el mejor director de proyectos de la compañía). Estas necesidades influencian la motivación del individuo y originan comportamientos dirigidos a satisfacerlas. Entre los comportamientos dirigidos a satisfacer las necesidades de un individuo está el trabajo.

En la medida en que su trabajo dentro del proyecto le permita satisfacer estas necesidades, el individuo se sentirá motivado, de tal manera que su desempeño seguramente será adecuado y beneficioso para sí mismo, para el proyecto y para la empresa u organización en la que trabaja.

(Pablo Lledó y Gustavo Rivarola, 2007)

3.1.2. GESTIÓN DEL PERSONAL DE PROYECTOS DE SOFTWARE COMPLEJOS Y DE AMPLIO ALCANCE

Los proyectos de software se encuentran en constante evolución. Esta evolución, ya sea para bien o para mal, ocasiona que el proyecto aumente en tamaño. Uno de los factores que ocasionan que el proyecto aumente en tamaño, es la necesidad de agregar nuevas funcionalidades al producto, por ejemplo a consta de un cambio en el mercado o de la propia evolución del mismo. Toda funcionalidad que se agregue necesita gente que la trabaje. Aunque las organizaciones suelen utilizar la rotación de personal, lo estrategia más común que suelen utilizar es la contratación, de esta forma no vacían ningún cupo.

A su vez, el producto siempre se encuentra en proceso de mantenimiento. Para esto se requiere gente que se encargue de atender todos los problemas y requerimientos que puedan llegar a surgir. Muchos productos entran en el estado que se conoce como barrera de mantenimiento, es decir se encuentran en un estado en el porcentaje de recursos necesarios para el mantenimiento se incrementa a tal punto, que imposibilitan la generación de nuevo software sin la contratación de personal.

Otra cuestión a tener en cuenta es la posibilidad de pérdida de personal a causa de un cambio de empleo. Si una persona que hacía 6 años que trabajaba dentro del mismo proyecto, decide dejar la organización, probablemente abra un cupo que pueda requerir más de una persona. Una persona con estas características, suele conocer cuestiones muy complejas no solo del producto, sino también del proceso y del negocio. Entonces llenar un lugar así, puede necesitar el ascenso de alguien que a su vez abriría otro cupo, o la contratación de mayor personal.

En consecuencia, los proyectos de software constantemente tienen la necesidad de aumentar el número de personas a causa del aumento en tamaño y alcance del producto. A medida que un proyecto de software aumenta en tamaño y alcance, también aumenta en complejidad, ocasionando que las estrategias de gestión utilizadas deben cambiar.

Por ejemplo, en un proyecto de software no es lo mismo gestionar 15 personas, que gestionar 100. Probablemente para el proyecto formado por 15 personas, se disponga comunicación directa y personal. Pero para el caso del proyecto formado por 100 personas, seguramente no se encuentren todos en la misma oficina. Hasta podrían estar en diferentes ciudades. Esto genera que se necesite otro medio para gestionar el personal.

3.1.3. GESTIÓN DEL TRABAJO

Uno de los problemas que ocurre cuando estamos ante proyectos de software complejos y de amplio alcance, está dado por cómo administrar la asignación y distribución del trabajo. En un proyecto formado por 15 personas, probablemente todos los empleados conozcan todas las reglas y requerimientos del producto. Pero si estamos ante un proyecto formado por más de 100 personas, el conocimiento del negocio debe organizarse de distinta manera. Por esto la estrategia más utilizada para administrar el trabajo, es organizar el personal en equipos autos organizados, con un objetivo y alcance específico dentro del proyecto. De esta forma, cada

equipo se responsabiliza del desarrollo y mantenimiento de lo que se encuentra dentro de su alcance y puede desinteresarse del resto.

En un proyecto típico de software, la mayor cantidad del personal que integra el mismo son IT, es decir desarrolladores y programadores. A su vez, es necesario optimizar la etapa de ejecución del proyecto. Es por esto que las organizaciones suelen utilizar la mayor cantidad de personas con este perfil. Así que supongamos un caso límite, un proyecto formado por 500 personas en el que el 80% del personal que integra el mismo, son desarrolladores. Eso nos daría un total de 400 personas. Las metodologías agiles proponen que los equipos deben ser chicos, no más de 10 personas. De esta forma se logra optimizar la comunicación más eficientemente. Supongamos entonces equipos típicos de 5 personas. Estaríamos hablando de 80 equipos. Tener que gestionar 80 equipos en conjunto, parece seguir siendo un trabajo un poco arduo de realizar. Es por esto que ante proyectos de este estilo, no alcanza simplemente la división del personal en equipos, es necesario apoyarse en otra estrategia de gestión para disminuir la probabilidad de riesgos dentro del proceso de desarrollo.

Por esta cuestión los proyectos se "*modularizan*", es decir, sub-dividirse de acuerdo a algún tipo de criterio. Por ejemplo, verticales, horizontales u otra estrategia de gestión del trabajo que le permita a la organización definir una estructura de organización piramidal. De esta forma se logra acotar el alcance de las responsabilidades de un equipo y definir estrategias de control y administración específicas para cada módulo, agregando también mecanismos para administrar todos los módulos en conjunto.

Aun así, la modularización tiene sus problemas, por ejemplo el acoplamiento. El acoplamiento existente entre módulos, indica el nivel de dependencia existente entre cada uno de ellos. Entre mayor sea el grado de acoplamiento, mayor será la necesidad de que ambos módulos convivan y funcionen en conjunto. El bajo acoplamiento entre las unidades de software es el *estado ideal* que siempre se intenta obtener para lograr una buena programación o un buen diseño. Cuanto *menos dependiente* sean las partes que constituyen un sistema informático, *mejor* será el resultado. Sin embargo, es imposible un desacoplamiento total de las unidades. Por ejemplo, si es necesario desarrollar una caja de búsqueda *cross* a toda la aplicación, no tiene sentido que el módulo de paquetes la desarrolle con su propio diseño y que el módulo de hoteles defina el suyo. Para mantener consistencia, lo correcto es que ambas implementaciones sean iguales, o simplemente debería ser una única implementación y ambos módulos reutilicen las soluciones.

Por ello, el objetivo final del diseño de software es reducir al máximo el acoplamiento entre componentes. Para ello, lo más importante es saber eliminar el acoplamiento que no sea funcional o arquitectónico.

El caso del acoplamiento funcional, puede ser por ejemplo que un componente de cálculo de probabilidades dependa de un componente de cálculo matemático básico. Esto se debe a que para calcular probabilidades será necesario aplicar fórmulas matemáticas.

El bajo acoplamiento permite:

- Mejorar la mantenibilidad de las unidades de software.

- Aumentar la reutilización de las unidades de software.
- Evitar el efecto onda, es decir, evitar que un defecto en una unidad puede propagarse a otras, haciendo incluso más difícil de detectar dónde está el problema.
- Minimiza el riesgo de tener que cambiar múltiples unidades de software cuando se debe alterar una.

Entonces si llevamos este caso a los proyectos de software complejos y de amplio alcance, resulta impensado contar con un acoplamiento cero. Imagínese un caso real, en el que se comienza un proyecto desde cero, con pocas personas. En las metodologías agiles en lugar de pasar mucho tiempo **planificando** y luego **ejecutando** el plan, se planifica una pequeña parte y se comienza a trabajar. Entonces resulta impensado que al principio disponiendo 5 personas, sea posible suponer que en 1 o 2 años pueda llegar a crecer tanto como para necesitar 200 o 300 personas. Normalmente las arquitecturas de las aplicaciones suelen irse adaptando de acuerdo al avance del proyecto.

Tanto Scrum como Kanban utilizan pizarras para gestionar y controlar el personal y todas las tareas que se necesitan llevar a cabo dentro del proyecto. A medida que aumenta la complejidad del proyecto, aumenta el personal y probablemente las necesidades del mismo. A mayor personal, mayor necesidad de control. Con esto surge la necesidad de apoyarse en herramientas de gestión de proyectos que faciliten y aumenten la eficiencia del desarrollo de software. Es por esto que normalmente el número de pizarras depende directamente de la estrategia de gestión utilizada. Para el caso de los proyectos de software complejos que utilizan la estrategia de modularización, es común observar una pizarra por módulo.

3.1.4. CONCLUSIÓN

La gestión del trabajo apunta a la definición de estrategias para dividir el proyecto, con el objetivo de facilitar el control de las actividades que se realizan para producir el producto. En contra parte, la gestión del personal apunta a las estrategias utilizadas para dividir el personal que integran el proyecto en pequeños grupos manejables y coordinarles. Cuando hablamos de proyectos de software complejos, ambos términos están relacionados. Esto se debe a que normalmente la sub-división del personal se realiza de acuerdo a la estrategia utilizada para dividir el proyecto.

Otro de los atributos dentro del contexto del proyecto, en el que la experiencia aún sigue cumpliendo un papel importante, es la gestión de las tareas.

3.2. GESTIÓN DE LAS TAREAS

En general, las tareas suelen confundirse con los proyectos. Esto se debe a que ambos términos comparten muchas características. Por ejemplo, se llevan a cabo personas, están limitados por recursos escasos (tiempo, dinero, etc.) y necesitan ser planificados y controlados. La principal diferencia radica en que las tareas son repetitivas y se mantienen en el tiempo, mientras que los proyectos son únicos y temporales.

Todo proyecto tiene al menos una persona responsable del control y administración de todas las actividades a realizar durante todo el proceso de gestión. Esta persona es conocida como director del proyecto. Desde un punto de vista formal podríamos definir al director del proyecto como la persona responsable de la consecución de los objetivos temporales, económicos y de alcance del mismo. Lo que desde un punto de vista funcional se acaba traduciendo en que el director del proyecto asume una figura de integrador y facilitador, tanto en lo referente al propio proyecto, como al equipo humano relacionado con este. (Lledó, 2007)

En Scrum y Kanban cuando estamos ante proyectos de software complejos, la dirección suele ser piramidal. Todos los equipos Scrum disponen un Master o líder de equipo. El Master es el responsable del cumplimiento de las reglas de negocio, asegurando que se entienden en la organización, y se trabaja conforme a ellas. Es responsable de:

1. Realizar la reunión de planificación del sprint antes de cada cierre de sprint.
2. Asegurar que se cuenta con una pila del producto preparada por el propietario del producto.
3. Ayudar a mantener el diálogo entre el propietario del producto y el equipo.
4. Asegurar que se llegue a un acuerdo entre el propietario del producto y el equipo respecto de lo que incluirá el incremento.
5. Ayudar al equipo a comprender la visión y necesidades de negocio del cliente.
6. Asegurar que el equipo ha realizado una descomposición y estimación del trabajo realistas, y ha considerado las posibles tareas necesarias de análisis, investigación o apoyo.
7. Asegurar que al final de la reunión están objetivamente determinados:
    a. Los elementos de la pila del producto que se van a ejecutar.
    b. El objetivo del sprint.
    c. La pila del sprint con todas las tareas estimadas.
    d. La duración del sprint y la fecha de la reunión de revisión.

A su vez proporciona:

- Asesoría y formación al equipo para trabajar de forma auto organizada y con responsabilidad de equipo.
- Revisión y validación de la pila del producto.
- Moderación de las reuniones.
- Resolución de impedimentos que en el sprint pueden entorpecer la ejecución de las tareas.

- Gestión de las dificultades de dinámica de grupo que se puedan generar en el equipo.
- Configuración, diseño y mejora continúa de las prácticas de Scrum en la organización. Respeto de la organización y los implicados, con las pautas de tiempos y formas de Scrum.

Kanban en este sentido es un bastante menos estructurado en cuanto a los procesos y a las responsabilidades. Todos dentro del equipo son responsables de todo aunque normalmente existe un seudo Master que es el responsable de la interacción con el personal externo al equipo, que conoce mejor el proyecto, conoce mejor las reglas de negocio, etc.

De esta forma tanto Scrum como Kanban gestionan la comunicación dentro y fuera del equipo. Aun así, si volvemos al ejemplo del apartado anterior, un proyecto formado por 500 personas podría llegar a estar formado por hasta 80 equipos, demuestra que por más experiencia que disponga el Master, sería imposible interactuar con otras 79 personas de su mismo rol. Es por esto que normalmente se utilizan personas encargadas de controlar la comunicación hacia fuera de los módulos. Los mismos suelen recibir el nombre de arquitectos de modulo, directores de modulo, gerentes de modulo, etc.

Entonces supongamos que los 80 equipos se dividen en 8 módulos, es decir, 10 equipos para cada uno de ellos. Siguiendo esta aproximación tendríamos una pirámide de 5 personas por equipo con un líder, dándonos un total de 10 líderes por modulo y 8 gerentes de proyecto. Siguiendo con la idea de que se prioriza la agilidad dentro de los proyectos, y que tanto Scrum como Kanban proponen utilizar una pizarra por modulo, disponer módulos formados por más de 100 personas resulta prácticamente imposible de controlar. 100 personas compartiendo el mismo tablero, con sus tareas y requerimientos, ocasionaría un caos para el encargado de interpretar o leer las mismas. En consecuencia se necesita definir otra estrategia para supervisar todos los equipos pertenecientes al mismo módulo.

4. GESTORES DE PROYECTOS

Disponemos de un gran abanico de posibilidades en la elección de una herramienta de apoyo en la gestión de proyectos. Es por esto que la primera gran decisión que debe abordar la organización, es definir cuál de estas herramientas se adaptará mejor a las expectativas del proyecto. Dos de las herramientas de software más populares en el ámbito empresarial son Trello y Jira. Ambas destinadas a gestionar proyectos de software, con sus estrategias y metodologías.

Todos los proyectos de software son diferentes. Aun así, las organizaciones suelen optar por una de ellas debido a la popularidad que poseen, sin tener en cuenta que esta decisión debería depender de las características del proyecto.

Tanto Trello como Jira presentan inconvenientes que la organización necesita evaluar. Trello está implementado con el objetivo de proveer agilidad y facilidad en el uso. A consta de esto, pierde la posibilidad de poder adaptarse a cualquier tipo proyecto. Jira en contra parte, tiene como principal objetivo el poder gestionar cualquier tipo de proyecto. A consta de esto, dispone demasiada estructura, configuraciones y complejidad, perdiendo agilidad y facilidad en el uso.

4.1. CONFIGURACIÓN POR DEFECTO DE LOS GESTORES DE PROYECTOS

Con el objetivo de lograr abarcar la mayor cantidad de tipos de proyectos de software distintos, tanto Trello como Jira disponen configuraciones por defecto demasiado genéricas. Con esto pretenden brindar la posibilidad de poder adaptar la herramienta a cualquier tipo de proyecto. Estas situaciones agregan complejidad en el uso. Resultaría más eficiente disponer de una herramienta que para determinados tipos de proyectos, sea la que mejor se adapte a las expectativas del mismo. De esta forma se ahorraría el tener que repetir o redefinir configuraciones cuando se encuentren ante proyectos similares.

El problema que presenta esta situación, es que para proyectos similares seguramente se terminen utilizando diferentes configuraciones, con el tiempo que conlleva el proceso de definir las mismas. A su vez, si de un proyecto al otro la herramienta presenta cambios considerables, los usuarios probablemente necesiten capacitarse en el uso. Esto ocasiona que la herramienta se convierta en un cuello de botella.

4.1.1. TRELLO

Trello no dispone el concepto de proyecto. Permite crear tableros o pizarras Kanban con una configuración por defecto muy simple. Al dar inicio a un proyecto, la primera cuestión necesita definir la organización, es cómo representará al mismo dentro de la herramienta. Las dos soluciones posibles son a través de un conjunto de tableros, o simplemente acotar el concepto a un único tablero.

En Trello, los tableros son independientes uno del otro. Esto quiere decir que cualquier configuración definida, usuarios asignados, tarjetas creadas, listas, etc., conviven únicamente dentro de ese tablero. En consecuencia, si utilizáramos la primera opción, se generarían varios problemas. Por ejemplo, perder la posibilidad de relacionar historias de usuario, siendo este un factor muy importante dentro de la gestión de proyectos.

Si suponemos que la configuración definida es un tablero por equipo, otro problema sería que si un usuario cambiara de equipo, necesitaría darse de baja en un tablero y agregarse en el otro. A su vez, si el proyecto fuese grande y complejo, estaría formado por muchos equipos. Esto ocasionaría que el número de tableros sea inmanejable.

Trello está ideado para proyectos de todo tipo, pero resulta conveniente su utilización cuando estamos ante un proyecto chico y simple, o directamente para uso personal. La simpleza que dispone la herramienta y la necesidad de tener que definir una configuración diferente para cada tipo de proyecto, ocasiona que no sea una buena elección cuando estamos ante un proyecto grande y complejo.

4.1.2. JIRA

Jira, a diferencia con Trello, si define el concepto de proyecto. Permite crear, modificar y eliminar proyectos de acuerdo a las necesidades de la organización. El problema, del cual se queja la mayoría de las personas que lo utilizan, es la complejidad que presenta la herramienta.

Jira permite adaptarse a cualquier tipo de proyecto, configurarse de acuerdo a las especificaciones del mismo, a las necesidades de la organización, etc. Por ejemplo, permite definir pizarras Kanban o Scrum, asociadas a un mismo proyecto. Es decir, no obliga a la organización a tener que optar por una de las dos metodologías agiles mayormente utilizadas. El problema que presenta con respecto a esto, es que está pensado para que cada equipo tenga su propia pizarra, con su configuración, métricas y reglas.

Supongamos que la organización decide utilizar la misma estructura para todos los equipos por cuestiones de análisis de métricas. Por ejemplo impone que la metodología a utilizar sea Scrum y define que el tiempo de Sprint es de 2 semanas. Un primer problema que se puede observar en este punto, es que se tendría que definir la misma configuración por cada equipo. Es decir, por cada pizarra creada dentro de la herramienta. Si estuviéramos ante un proyecto complejo, formado por ejemplo por 100 equipos, este proceso terminaría resultando lento y poco eficiente.

En Jira una historia de usuario perteneciente a una pizarra, puede tener dependencia o relaciones solo con otra/s historia(s) de usuario dentro de la misma pizarra. Utilizando el esquema de una pizarra por equipo, se perdería la posibilidad de definir relaciones entre las historias de usuario de los diferentes equipos. Supongamos que se está trabajando en una pantalla, en la que un equipo es el responsable del desarrollo de la vista y otro es el responsable del desarrollo de la lógica. Agreguemos que el requerimiento del desarrollo de la vista está dividido en 10 historias de usuario, mientras que el de la lógica solo en 5. Si existiera dependencia entre las historias correspondientes a cada tópico, los equipos se verían

imposibilitados de trabajar en paralelo, teniendo la necesidad de tener que contactarse para supervisar el estado del trabajo del otro.

4.2. CONTROL DEL PROYECTO

A medida que los proyectos aumentan en complejidad y tamaño, las organizaciones suelen utilizar diferentes estrategias de gestión y control para cada una de las partes que integran el mismo. Una de las principales cuestiones a tener en cuenta, es la división del trabajo.

Los proyectos de software complejos suelen "modularizarse". Es decir, sub-dividirse de acuerdo a algún tipo de criterio, como por ejemplo, verticales, horizontales u otra estrategia que defina la organización. Este método permite acotar las estrategias de control a cada módulo específico y definir mecanismos para administrar todos los módulos en conjunto. Tanto Trello como Jira permiten configurarse para poder adaptarse a este fin, logrando administrar cada módulo de forma independiente. Por ejemplo, una posible arquitectura que se podría utilizar, sería un tablero/pizarra por cada módulo definido. En ambas herramientas todas las configuraciones se definen de manera independiente por cada tablero o pizarra. Entonces esta solución permite que cada módulo pueda disponer su propia configuración, logrando que cada uno de ellos sea independiente al resto.

Cuando existe un alto acoplamiento entre los distintos sub-proyectos, la organización necesita definir estrategias para supervisar todos los módulos en conjunto. Esto agrega complejidad al uso de la herramienta y le quita agilidad al proceso de gestión del proyecto.

4.2.1. TRELLO

Trello permite que ante un proyecto complejo y de gran tamaño, cada sub-proyecto tenga su propio tablero Kanban, con sus reglas, usuarios y definiciones. Es decir, permite tratar a cada sub-proyecto de manera independiente, dejando de lado los puntos en común que pueda tener cada uno. Esto genera varios problemas:

- Se deben redefinir todas las configuraciones y reglas que sean compartidas para cada tablero.
- Si un usuario por cualquier motivo necesita cambiar de sub-proyecto, sería necesario darlo de baja del tablero actual y agregarlo al nuevo tablero. En proyectos en los que esta situación sucede muy a menudo, el proceso de alta y baja de un usuario terminaría resultando tedioso.
- Trello permite marcar dependencias entre tareas pertenecientes al mismo tablero. Si una tarea perteneciente a un determinado sub-proyecto, tuviera dependencia con otra externa al mismo, sería imposible marcarla de algún modo.

4.2.2. JIRA

Jira, en contraparte con Trello, permite que ante un proyecto con un alto grado de acoplamiento o relación entre cada sub-proyecto, exista un único tablero unificado, con sus reglas y

configuraciones compartidas. El problema de esta solución, es el medio que utiliza para diferenciar cada sub-proyecto dentro de la pizarra.

Jira utiliza filtros y tags como medio de agrupamiento de historias de usuario, con el objetivo de beneficiar al proceso de búsqueda. De esta forma, cada sub-proyecto podría disponer su propio tag, para así poder identificarlo rápidamente dentro del tablero. El problema que presenta esta aproximación es que los tags están pensados para otro propósito. Por ejemplo, agrupar historias según el modulo al que pertenecen, características de las mismas, etc. Si existieran 20 sub-proyectos, existirían 20 tags simplemente para diferenciar un sub-proyecto del otro. También habría que agregar todos los tags definidos para agrupar las historias de usuario. Entre más tags disponga la herramienta, mayor complejidad se agrega al proceso de creación y búsqueda de una historia de usuario, ocasionando que ambos procesos sean poco eficientes.

Otro problema que se observa es que los usuarios deberían conocer todos los tags definidos dentro del sistema, para así poder filtrar historias de usuario con mayor agilidad. Para un proyecto en el que se produce un alto grado de modificación en los equipos, esto terminaría resultando poco práctico para los usuarios.

4.3. EQUIPOS Y ROLES

Otro de los factores a tener en cuenta a la hora de definir estrategias de gestión y control para el proyecto, es la división del personal. Este punto suele estar muy relacionado al anterior: Primero suele determinarse cuáles serán las personas asignadas a cada uno de los módulos definidos y luego se define que función o tarea cumplirán cada una de las personas dentro de cada uno de ellos. Aun así, a medida que aumenta la complejidad y el tamaño de los proyectos, también aumenta la complejidad y tamaño de cada uno de los módulos.

Normalmente la complejidad y tamaño de los proyectos, ocasionan que la modularización necesite apoyarse en otra estrategia de división del trabajo. Disponer módulos formados por más de 100 personas resulta prácticamente imposible de controlar.

Dentro de un proyecto participan usuarios de todo tipo. Entonces resultaría imprescindible la sub-división del personal dentro de cada módulo, según algún tipo de criterio. La alternativa mayormente utilizada es la de utilizar equipos auto organizados, formados por un número acotado de personas.

Tanto Trello como Jira disponen soluciones para administrar cada equipo de forma independiente. Por ejemplo, otra posible arquitectura que se podría utilizar, sería un tablero/pizarra por cada equipo definido. De esta forma, todas las tareas y configuraciones que integren la pizarra o el tablero, serian propias de cada equipo.

La diversidad de equipos que suelen existir dentro de un proyecto, por ejemplo equipos de diseño, equipos IT, equipos de administración, equipos de soporte, etc., hace sonar lógico el utilizar una pizarra por equipo, con sus propias tareas y configuraciones. El disponer esta arquitectura permite que las configuraciones y reglas de un tablero de soporte, sean diferentes a

las de un tablero de diseño. A su vez, cada equipo puede optar por utilizar la metodología de gestión de proyectos que mejor se adapte a su forma de trabajo.

4.3.1. TRELLO

Si nos referimos al control del personal, el problema que presenta utilizar Trello como herramienta de apoyo en la gestión de proyectos complejos de software, es que no está preparada para este propósito. Trello permite asignar personas a un tablero, pero no permite definir roles ni organizar a los mismas en equipos para facilitar el control.

4.3.2. JIRA

Jira, en contra parte, permite definir equipos de desarrollo dentro de los tableros. De esta forma facilita el control del personal dentro del proyecto. También permite definir roles y permisos asociados a los usuarios. El problema que dispone Jira, es la complejidad que dispone para definir todas estas configuraciones dentro de la herramienta.

4.4. CREACIÓN DEL PROYECTO

Querer brindar soporte a la sub-división del proyecto en módulos, a la sub-división del personal en equipos, utilizar roles, definir permisos y definir un mecanismo para controlar el desarrollo, agrega la necesidad de tener que configurar los tableros, posteriormente a la creación del proyecto.

4.4.1. TRELLO

Trello por su lado hace fuerza en la sencillez de la interfaz y en la agilidad del uso. Entonces al momento de la creación de un proyecto, ofrece un tablero por defecto demasiado genérico. Con esto consigue que para proyectos de pequeño porte sea una de las herramientas más utilizadas. Lógicamente esta sencillez le hace perder fuerza en la gestión de proyectos complejos.

En Trello los tableros son independientes uno del otro. Entonces resulta prácticamente imposible realizar la gestión de un proyecto sub-dividido en módulos, donde el personal se encuentra organizado en equipos. Para lograr este cometido, sería necesaria apoyarse en plugins externos o definir estrategias o configuraciones que permitan adaptar la herramienta.

4.4.2. JIRA

Jira en primera instancia dispone un proceso sencillo para la creación de un proyecto. El administrador crea el mismo simplemente indicando un nombre que lo identifique. Luego necesita crear tableros asociados, cada cual con su metodología de gestión específica. Cada tablero dispone su propia configuración y es independiente uno del otro.

El problema que dispone Jira es que la configuración por defecto definida para cada tablero, en proyectos de software complejos siempre resulta acotada. En consecuencia la organización en

todos estos casos se ve en la obligación de definir sus propias reglas, con el objetivo de adaptar la herramienta.

Que un tablero sea independiente al otro, ocasiona que por cada configuración definida, resulte necesario generar la misma configuración en cada tablero por separado. En este caso, cada vez que se da inicio a un proyecto, resulta necesario definir una configuración específica para cada tablero. Para el caso en el que se modifique una regla de negocio, también se debe replicar el cambio a todos los tableros asociados. Si estuviéramos ante un proyecto complejo, formado por muchos módulos y equipos como el del ejemplo anterior, el proceso de creación del proyecto y el de modificación de los tableros, sería exhaustivo.

4.5. PROCESO DE DESARROLLO Y ETAPA DE PLANIFICACIÓN Y VERIFICACIÓN DEL SOFTWARE

Las empresas buscan a toda costa mejorar sus procesos para mantenerse competitiva. Pero existe un gran interés en lograr la agilidad organizacional, con la esperanza de acortar la implementación de cambios urgentes y alcanzar las metas. Aunque el desarrollo ágil esté de moda, existen proyectos en el que resulta conveniente aplicar un "mix" entre lo tradicional y lo ágil. Es aquí donde el factor humano y la experiencia del Project Manager terminan resultando esencial para el bien estar del proyecto.

Seleccionar las herramientas adecuadas que permitan llevar al proyecto a buen fin y obtener la satisfacción de todos los Stakeholders implicados, son dos de los grandes retos que necesitan contemplar los Project Manager.

La diferencia entre las metodologías agiles y las metodologías tradicionales están en la forma en que se aborda el alcance del proyecto. La forma tradicional indica que no se puede planificar un proyecto hasta no tener detallado completamente su alcance. Esta metodología funcionaba correctamente para proyectos de industria y para proyectos en donde el alcance era relativamente sencillo de determinar. Pero en la industria del software el alcance de un proyecto no siempre se puede calcular, generando la necesidad de disponer una alternativa a los procesos de gestión de proyectos.

Debido a la velocidad a la que se mueve el mercado del software y a la incertidumbre que disponen los usuarios finales, se prefiere construir el producto al mismo tiempo que se modifican e introducen nuevos requisitos. Normalmente el entorno del negocio del cliente se mueve a una velocidad tal que no le permiten prever con detalle cómo será el resultado final. Esto ocasiona requerimientos cambiantes y vulnerables, ocasionando que la producción de software prefiera utilizar estrategias para el lanzamiento de productos orientadas a la entrega temprana de resultados tangibles, y a la respuesta ágil y flexible. Esto no quiere decir que se haya dejado de lado la planificación, sino que en lugar de llevarse a cabo al inicio del proyecto, se realiza de forma sucesiva durante todo el ciclo de vida.

¿Ahora, qué sucede en los proyectos de software en los que no se cuenta con un cliente real o físico, sino que el sistema es de ámbito comercial y está dirigido a todas las personas? En estos

tipos de sistemas, los requerimientos parten a partir de un análisis de producto realizado por la organización. Por ejemplo, para un sistema de venta de viajes, puede surgir la idea de vender el hotel, el transporte desde el aeropuerto al hotel, un auto para trasladarse, etc. Estas cuestiones surgen luego de haber hecho un análisis del mercado. El empezar a desarrollar sin tener en claro el alcance de la funcionalidad, produce pérdida de tiempo, y en el mundo del software, el tiempo es dinero.

El ciclo de vida de un proyecto se encuentra en función del costo y del tiempo. En un proyecto típico de software, los costos de la fase de ejecución, es decir, la mano de obra, pueden llegar casi al 100% del costo total del proyecto. Esto se debe a que la fase de ejecución se corresponde casi al 100% del ciclo de vida del proyecto. En consecuencia los procesos de administración, independientemente de la metodología que se utilice, se enfocan en optimizar la etapa de desarrollo del *proyecto*, es decir, reducir los tiempos de desarrollo. El tener bien en claro que es lo que se quiere desarrollar, como desarrollarlo y en qué orden, mejora notablemente los tiempos de desarrollo.

Siguiendo el ejemplo anterior, suponga el caso en el que la organización decide encarar el módulo de hoteles. Hay cuestiones mínimas que necesitan planificarse, previo a la codificación. Por ejemplo, definir si conviene tercerizar la venta de hoteles cobrando una pequeña comisión por cada venta, o directamente comprar habitaciones y revenderlas desde la aplicación. Otro ejemplo sería analizar si conviene proveerle descuentos al usuario que ya compro más de una vez el mismo hotel, recomendarle hoteles según el destino del vuelo al que se dirija, etc. Probablemente esta decisión termine influenciando al desarrollo del producto final, entonces conviene tener bien planificado el desarrollo, para no perder tiempo re-implementando lo mismo.

Otro de los problemas que se suelen observar en proyectos de este estilo, es que las organizaciones suelen omitir la etapa de verificación. Los sistemas que están desarrollados con un propósito comercial, en lugar de ser objeto de la necesidad de una empresa, tienden a pensar que el cliente final no es tan "crítico". Es decir, meter un error en producción dentro del módulo de hoteles, generaría en el peor de los casos que no se puedan venderse hoteles. Pero meter un error en un sistema desarrollado para una empresa particular, podría llegar a ocasionar que la misma decida desvincularse de la relación con la organización, generando la baja del proyecto.

Las organizaciones tienden a dejar de lado los procesos tradicionales, ya sea la etapa de planificación y verificación. Si la organización quisiera poner énfasis a estas dos etapas, probablemente necesitarían apoyarse en una herramienta específica desarrollada para este propósito. El problema que trae está aproximación, está dado por la necesidad de gestionar el software a través de múltiples herramientas.

4.5.1. TRELLO

Si quisiéramos gestionar el proceso de requisitos en Trello, deberíamos utilizar un tablero diferente específicamente para este propósito. Esto se debe a que las configuraciones dentro de

la herramienta están ligadas a un único tablero. Entonces no sería recomendable que conviva una estructura compartida entre las historias de usuario y los requerimientos del sistema.

Si aun así, decidiéramos ir por este esquema, se perdería la posibilidad de relacionar historias de usuario con los requerimientos. Es necesario recordar que de un requerimiento se derivan múltiples historias de usuario. En consecuencia, esta solución perjudicaría al proceso de búsqueda y al proceso de gestión del proyecto.

Si quisiéramos gestionar el proceso de verificación en Trello, deberíamos definir una configuración dentro de la herramienta específicamente para este propósito. Trello no dispone un medio nativo para realizar este proceso, generando el problema de tener que pensar o implementar un medio para conseguirlo.

4.5.2. JIRA

Jira permite crear pizarras configuradas de acuerdo a la metodología Scrum o Kanban. Al crear la pizarra, define por defecto una estructura según la metodología elegida. El problema que presenta con respecto al proceso de licitación de requerimientos, es que no define una pila de producto en ninguna de los dos esquemas. En Scrum, el ciclo de iteración comienza en la Pila de Sprint, nombrada como “Backlog”, mientras que en Kanban comienza en la lista “Por hacer”.

En consecuencia, si quisiéramos gestionar el proceso de requerimientos, deberíamos utilizar un tablero diferente y configurarlo para darle soporte a este proceso. Este tablero tendría que configurase de acuerdo a alguna de las dos metodologías soportadas.

Con respecto a la verificación del software, Jira al igual que Trello, no dispone un mecanismo para realizar este proceso. La diferencia radica en que Jira define un flujo de trabajo por defecto que permite modificar, agregar estados y procesos al mismo. Entonces la organización podría utilizar el flujo de trabajo y agregar el proceso de verificación en el mismo. El problema está en que no es tenido en cuenta en primera instancia y puede llegar a generar complicaciones al proceso de gestión del proyecto.

4.6. BÚSQUEDA DE HISTORIAS DE USUARIO

Cuando hablamos de proyectos de software complejos y de gran tamaño, la búsqueda de historias de usuario es otro de los factores que quitan agilidad al proceso de desarrollo. Las historias de usuario relacionadas pueden resultar beneficiosas para el proceso de análisis. Esto se debe a que normalmente muchos de los requerimientos no funcionales, características y soluciones suelen compartirse. Tanto Trello como Jira disponen motores de búsqueda muy poderosos, en donde el usuario ingresa características de la historia que desea encontrar y rápidamente obtendrá resultados. El problema de este esquema es que la mayoría de las veces no se conoce ninguna característica asociada a la historia de usuario, que permita realizar la búsqueda fácilmente.

4.6.1. TRELLO

Trello dispone un motor de búsqueda de tarjetas muy poderoso que permite buscar por nombre, usuario asignado, etc. A su vez permite guardar búsquedas. El principal problema que presenta es que deja de lado la posibilidad de marcar dependencias o relaciones entre tarjetas. Este esquema resultaría muy útil cuando el usuario no conoce ninguna característica relacionada a la misma.

4.6.2. JIRA

Jira también permite buscar tarjetas utilizando un motor de búsqueda muy poderoso. Dispone un motor de búsqueda mucho más poderoso que Trello. Si los usuarios que configuran el tablero definen correctamente agrupamientos en tags y categorías de historias de usuario, posteriormente se puede filtrar por las mismas.

Jira en contra parte con Trello, si permite definir relaciones entre historias de usuario. Cada pizarra permite crear y modificar dependencias, y a su vez, permite utilizar las definidas por defecto por la herramienta. Por defecto permite indicar si una historia de usuario depende de otra, si fue creada a partir de otra, si es una historia clonada, si se necesita para el desarrollo de otra, si bloquea otra, etc. Esta aproximación está buenísima y cuando se usa bien, puede resultar muy útil. El problema es que la complejidad del proceso de creación de una historia de usuario, agregando la complejidad que genera que los usuarios puedan definir sus propias dependencias, terminan ocasionando que no se utilice.

Por último, jira no tiene en cuenta las relaciones que se dan por defecto al momento de crear una historia de usuario. Por ejemplo, todas las historias de usuario deben ser creadas a partir de un requerimiento específico. De esta forma se agilizaría el proceso de búsqueda permitiendo agrupar todas las tareas a partir del mismo.

A su vez, en muchas ocasiones el mismo proceso de creación de una historia de usuario debería definir relaciones automáticamente. De esta forma se evitaría el delegar la responsabilidad de este proceso al usuario que utiliza la herramienta.

4.7. VINCULACIÓN DE HISTORIAS DE USUARIO CON PULL REQUEST

Cuando hablamos de proyectos de software complejos y de gran tamaño, en los que colaboran muchas personas, es normal que la arquitectura de los mismos también sea compleja. Normalmente suelen estar formados por N repositorios, con muchas tecnologías de desarrollo, capas, paradigmas y millones de líneas de código cada uno. Estas situaciones originan que los desarrolladores, previo al proceso de implementación, se vean en la necesidad de adentrarse en un proceso de análisis con el objetivo de determinar si:

1. Existe algo ya hecho para reutilizar o simplemente es una funcionalidad nueva. Normalmente este proceso consiste en la detección de Pull Requests relacionados con el tópico a desarrollar. Los Pull Request sirven como guía para el proceso de codificación.

2. Investigar si existen reglas de negoció que se aplicaran al desarrollo, es decir, determinar todos los requerimientos no funcionales asociados a la tarea. Por ejemplo, autenticaciones, gama de colores, casos de uso, etc.
3. Investigar si la funcionalidad se encuentra bloqueada por otra tarea que necesita ser desarrollada previamente. Este bloqueo puede deberse a la necesidad de una funcionalidad que aún no se encuentra finalizada, a que la versión actual del sistema no soporta esta nueva característica, a que este cometido podría traer problemas en determinados ambientes de desarrollo, etc.
4. Si habláramos de un proyecto formado por un gran número de equipos:
    - investigar cual es equipo encargado del mantenimiento de dichas páginas. Suele ocurrir que la tarea está asignada a una persona perteneciente al equipo equivocado.
    - Investigar si el desarrollo de esta funcionalidad genera problemas a otro equipo.
5. Etc.

En este tipo de arquitecturas se crean y se fusionan muchos pull Request a diario, ocasionando que el proceso de búsqueda de los mismos resulte una tarea engorrosa de realizar. Github provee un motor de búsqueda muy poderoso, el problema es que la complejidad de los proyectos y la falta de buenas prácticas en las definiciones de los pull Request, generan que el proceso de búsqueda sea muy complicado.

Si la historia de usuario es lo suficientemente atómica como especifica la teoría definida por las metodologías agiles, la misma debería poder realizarse a través de un único pull Request. A lo sumo, se debería crear una incidencia posteriormente, si se encontraron bugs en un ambiente superior de testing. Por lo tanto, resultaría conveniente poder vincular ambos conceptos.

4.7.1. TRELLO

Trello no dispone un medio para realizar este cometido. Uno debería apoyarse en herramientas de terceros para implementar esta característica.

4.7.2. JIRA

Jira dispone de un medio nativo para vincular historias de usuario con Pull Request. El problema de esta característica está en la complejidad que posee. Jira permite prácticamente manejar GitHub desde Jira.

El problema es que las herramientas de gestión de proyectos, no son únicamente usadas por usuarios desarrolladores. También son utilizadas por usuarios de producto, diseñadores, etc. Estos tipos de usuario no suelen tener conocimiento técnico de un sistema de control de versiones. Si nosotros activamos este plugin, todos los usuarios agregados al tablero visualizaran información relacionada a las discusiones del código, soluciones, commits realizados, etc.

4.8. CONCLUSIÓN

Los gestores de proyectos presentan estructuras por defecto demasiado genéricas, permitiendo configurarse de acuerdo a las expectativas de casi cualquier tipo de proyecto. Con esto pretenden lograr abarcar la mayor cantidad de tipos de proyectos distintos. Tanto Trello como Jira están basados en una metodología específica (Trello), o hibrida (Jira). Cada una de ellas apoya a la organización en el seguimiento del avance del proyecto, a gestionar las tareas del personal, a realizar la estimación de esfuerzo y cronograma, a gestionar la comunicación con el cliente y a gestionar la documentación generada durante todo el proceso de gestión, dejando de lado muchas otras cuestiones muy importantes, como el proceso de licitación, la verificación del software, agilidad en distintos puntos del desarrollo, etc.

Aun así, debido a la heterogeneidad, complejidad y tamaño que disponen los proyectos de software, para seguir este camino es necesario dejar de lado cuestiones muy importantes. Es por esto que se propone el desarrollo de una herramienta, a partir de ahora llamada TG Project Management, que tiene como principal objetivo apoyar a la organización en el proceso de gestión proyectos, haciendo foco en solucionar todos los problemas planteados.

5. TG PROJECT MANAGEMENT

TG Project Management – (TG), utiliza diversas estrategias para realizar la gestión del proyecto, haciendo foco en aumentar la eficiencia del proceso de gestión del desarrollo que dispone Jira y Trello. De esta forma, el principal objetivo de la herramienta consiste en solucionar los problemas planteados, proponiendo acotar considerablemente la complejidad de diversos procesos.

5.1 CONFIGURACIONES POR DEFECTO

5.1.2 METODOLOGÍA DE GESTIÓN DE PROYECTOS

Scrum y Kanban tienen puntos fuertes y débiles en el manejo de cada proceso de la gestión del proyecto. Es por esto que TG utiliza las mejores prácticas de cada alternativa, aplicando cada método según resulte ventajoso al proceso de gestión del proyecto. Con esto consigue aumentar la agilidad del proceso de gestión y eliminar la necesidad de definir una configuración, para adaptar la herramienta a una metodología de gestión de proyectos diferente.

5.1.3 GESTIÓN DEL TRABAJO Y DEL PERSONAL

TG define métodos para organizar y administrar el trabajo dentro del proyecto. Por un lado percibe roles al igual que Scrum. Los roles dentro de la herramienta, ayudan a controlar y administrar el trabajo, restringiendo acciones dentro del producto. Por otro, permite organizar y administrar el personal a través de equipos y módulos.

5.1.4 PILA DE PRODUCTO

TG dispone un medio para definir requerimientos dentro de la herramienta. No solo utiliza esta característica para crear requerimientos, sino que aprovecha su potencial en el proceso de búsqueda de historias de usuario.

5.1.5 ITERACIONES

TG se basa en iteraciones (Sprints) de igual forma que lo define Scrum. La diferencia con el esquema que propone Scrum, está en que propone que el trabajo en progreso (WIP), debe estar limitado por estados del flujo de trabajo, de igual forma que lo define Kanban. El limitar el WIP por estados del flujo de trabajo trae grandes ventajas; por un lado, no fuerza a la organización a definir una duración fija de Sprint, para luego tener que adaptar el WIP a un lapso de tiempo definido, de esta forma se evitan tiempos muertos y sobreestimaciones de esfuerzo. Por otro lado, no descarta la ventaja que trae el organizar las historias de usuario en iteraciones más marcadas como es el Sprint, previendo que salgan características del producto no deseadas a los ambientes superiores.

Dentro de cada sprint las historias de usuario atraviesan un conjunto de etapas. Las mismas representan el estado en el que se encuentra cada tarea dentro del proceso de desarrollo, de forma muy similar a como lo propone Kanban. Estas etapas permiten controlar el flujo de desarrollo y estimar con mayor seguridad los tiempos.

En TG cada equipo puede dar por finalizado el Sprint en el momento en el que le resulte conveniente, pudiendo agregar o quitar historias de usuario, independizando su trabajo y velocidades con los demás equipos. Es decir, cada sprint es independiente al de los demás equipos. Esto permite mejorar la agilidad del proceso de desarrollo.

5.1.6 TIPO DE PROYECTO

TG apunta a proyectos de software complejos. Proyectos formados por un gran número de personas y tareas entre otras cuestiones. Aun así, se comporta perfectamente en etapas tempranas del proyecto, cuando la complejidad del mismo aún no se encuentra tan demarcada. De esta forma evita tener que definir configuraciones que le permitan adaptar la herramienta según el momento en el que se encuentre proyecto.

Unificar el foco de la herramienta trae grandes ventajas. Por un lado, la misma ya está preparada desde primer momento a soportar un gran número de personal dentro de la misma, disponiendo soluciones para controlar y gestionar dicha característica. Con esto se elimina la necesidad de definir configuraciones extras cada vez que el personal aumente.

Por otro lado, la herramienta también ya está preparada para soportar un gran número de historias de usuario, disponiendo estrategias para manejar sus estados, dependencias, filtros y relaciones, durante todo el ciclo de vida de la misma.

5.1.9 TIPOS DE USUARIO

TG está pensada para ser utilizada por usuarios desarrolladores. Es decir, centra todas las características de la herramienta, a la perspectiva del mismo, consiguiendo grandes beneficios al proceso de gestión del proyecto. Dispone soluciones por defecto que aumentan la agilidad del proceso de desarrollo. Por ejemplo, la vinculación de historias de usuario con Pull Request. A su vez, elimina aquellas funcionalidades que resulten innecesarias para el cumplimiento de la función del usuario desarrollador. De esta forma consigue disminuir considerablemente la complejidad de la herramienta. Por ejemplo, acota la estructura o definición de las historias de usuario, a lo que realmente necesita el mismo.

5.1.9 CONTROL DEL PROYECTO

TG utiliza numerosas estrategias para realizar el control del proyecto dentro de la herramienta:

- Define roles para administrar los permisos y responsabilidades de los usuarios.
- Utiliza el sprint para administrar el trabajo, permitiendo que cada equipo sea el propio responsable de gestionar su WIP.

- Permite visualizar los ítems que se realizaron en cada sprint, aunque el mismo este cerrado.
- Dispone medios agiles y avanzados para encontrar historias de usuario.
- Permite dar por finalizado un sprint según le resulte conveniente al equipo y/o organización.
- Gestiona el trabajo a través de módulos.
- Gestiona el personal a través de equipos.
- Gestiona el orden en el que se tienen que resolver las historias de usuario, a través de un esquema de prioridades.
- Etc.

5.2 CREACIÓN DE UN PROYECTO

TG dispone un proceso de creación del proyecto simple y ágil. Este proceso es llevado a cabo tal como lo representa la Ilustración 4 - Creación del proyecto:

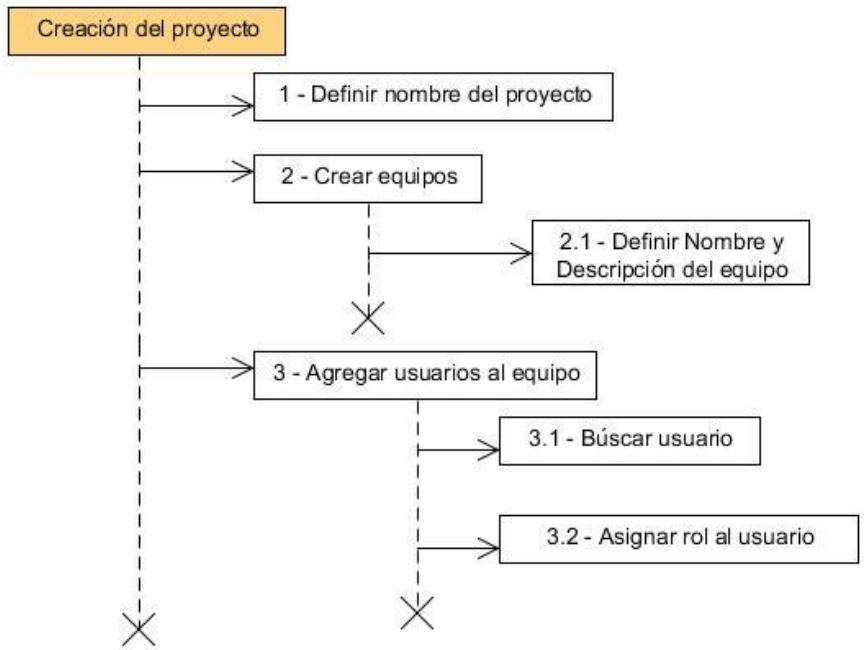

Ilustración 4 - Creación del proyecto

5.2.1 PROCESO DE CREACIÓN DEL PROYECTO

Al crear un nuevo proyecto dentro de la herramienta, primero hay que definir un nombre para identificar al mismo. Este nombre permitirá posteriormente acceder al proyecto:

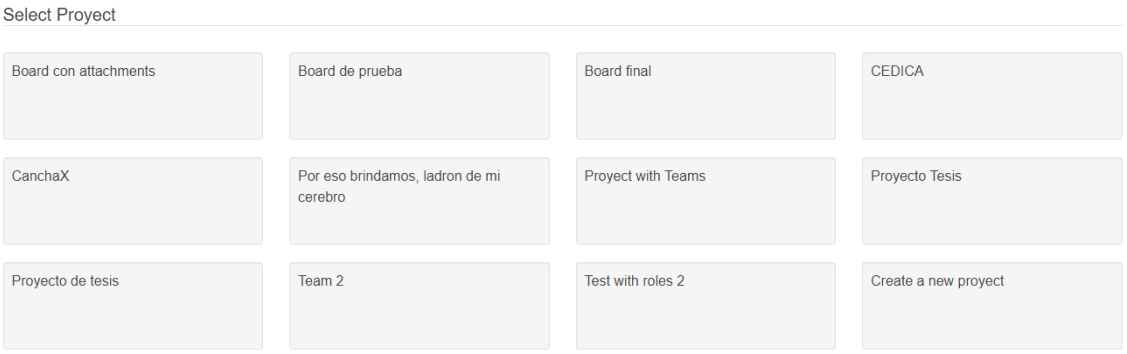

Ilustración 5 – Listado de proyectos

Simplemente con definir un nombre, ya disponemos un proyecto creado dentro de la herramienta, con todas sus configuraciones por defecto definidas. De esta forma no es necesario definir ninguna configuración extra al proyecto.

5.3 GESTIÓN DEL TRABAJO Y DEL PERSONAL

TG apunta a proyectos de software complejos y de gran tamaño. Es por esto que brinda la posibilidad de sub-dividir el proyecto en módulos y sub-dividir el personal en equipos.

Luego de creado el proyecto, es necesario definir el personal que integrará el mismo, el equipo al que pertenecerá cada usuario y el modulo al que pertenecerá cada equipo.

5.3.1 ALTA DE EQUIPO

TG dispone un medio para crear equipos al momento de la creación del proyecto, o posteriormente durante la etapa de mantenimiento. Los mismos pueden definirse simplemente indicando un nombre y una descripción. El nombre es un medio para identificar unívocamente al equipo dentro del proyecto. Es por esto que TG no permite 2 equipos con el mismo nombre. La descripción es el principal atributo que utiliza la herramienta para gestionar el personal. Dentro de este atributo debe especificarse información que permita organizar el personal. Por ejemplo, el modulo o vertical del sistema que dicho equipo realizará el mantenimiento y/o desarrollo, conjunto de páginas o rutas, objetivo del equipo, etc. Esta información posteriormente ayudará al desarrollador en la búsqueda de un equipo específico, como así también en la búsqueda de historias de usuario. Así que es muy importante utilizar correctamente este atributo.

La Ilustración 6 – Creación de equipos, muestra un ejemplo de cómo se vería un proyecto recién creado, luego de haber definido un conjunto de equipos.

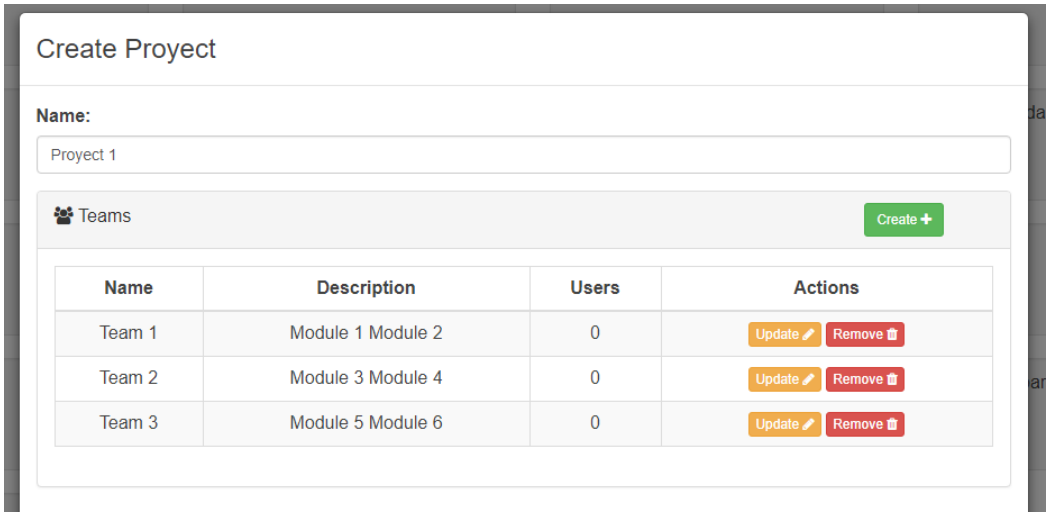

Ilustración 6 – Creación de equipos

5.3.2 AGREGAR USUARIOS AL EQUIPO

Luego de definidos los equipos, TG dispone de un medio para asignar usuarios a los mismos. En este punto hay que tener en cuenta varias cuestiones:

- Es necesario que un equipo este creado con anterioridad.
- Hay que tener en cuenta que si un equipo es borrado del sistema, todos los usuarios dentro del mismo se perderán.
- No puede haber un mismo usuario asignado a dos equipos distintos.
- Todo equipo debe tener al menos un Master.
- Los equipos no tienen restricción de cantidad de usuarios

Para asignar un usuario al equipo, la aplicación solicita que se ingrese el email del mismo. De esta forma se verifica que el usuario esté dado de alta en Trello y en GitHub con anterioridad.

Ilustración 7 – Agregar usuario 1

Al hallar el usuario, el sistema solicita que se le asigne un rol dentro del sistema. Los roles disponibles dentro del sistema son: "*Master*", "*User*", "*Owner*" y "*Other*".

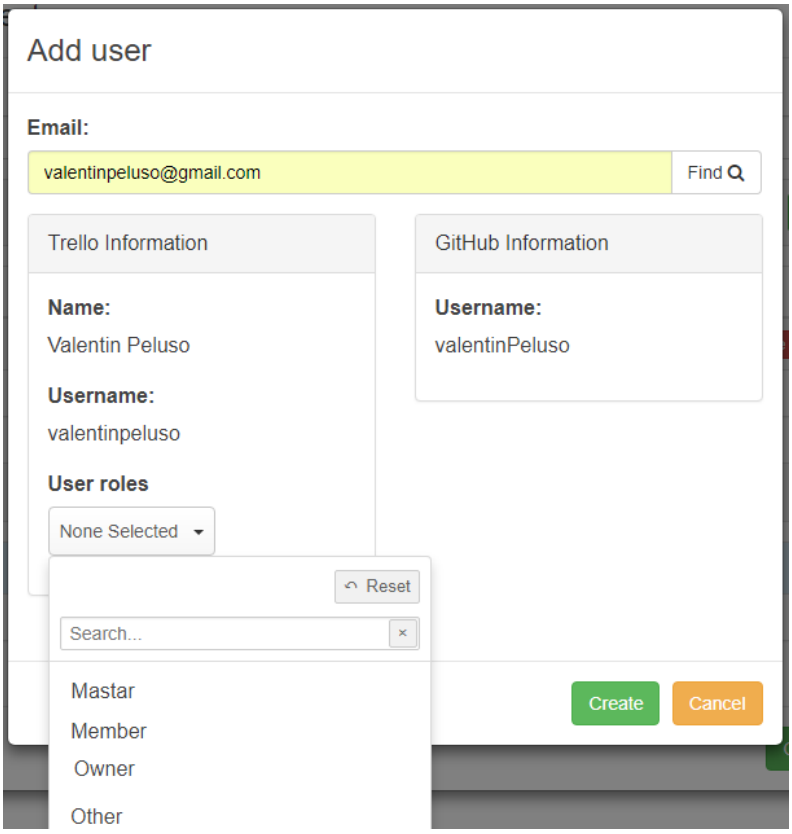

Ilustración 8 – Agregar usuario 2

Luego de asignado el rol, se habilita la opción de agregar el usuario al equipo indicado. De esta forma la distribución de los equipos para el proyecto se vería de una manera similar a la Ilustración 9:

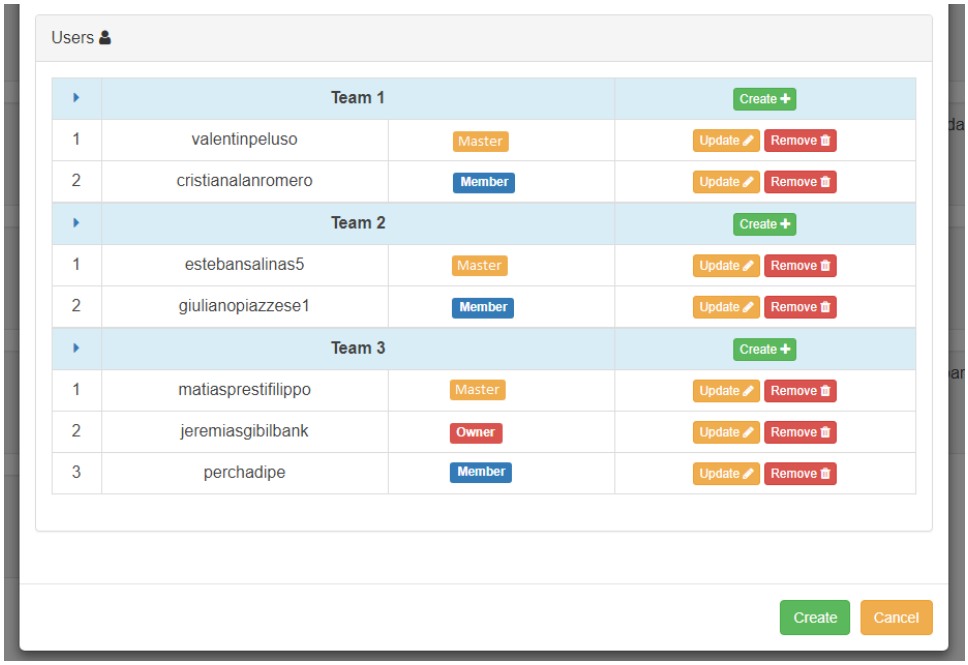

Ilustración 9 – Listado de usuarios agregados al proyecto

5.3.2.1 ROLES

TG utiliza los roles para realizar la gestión del trabajo dentro de la herramienta. Utilizando esta característica y la descripción del equipo, realiza la gestión del personal dentro del proyecto.

5.3.2.1.1 MASTER

Es el responsable del cumplimiento de las reglas predefinidas por la organización. Particularmente es responsable de:

1. Marcar la finalización del Sprint.
2. Crear historias de usuario a partir de los requerimientos del cliente.
3. Asignar las historias de usuario a los desarrolladores.
4. Asignar las historias de usuario al Sprint.
5. Marcar la validación y verificación de cada historia de usuario.

5.3.2.1.2 USER

Los miembros del equipo son los que están destinados a realizar el desarrollo de las historias de usuario. Son los responsables de agregar valor al producto. Se les permite visualizar todas las tareas definidas, pero no se les permite asignar tareas, ni indicar en cual Sprint se llevará a cabo

el desarrollo cada una. El rol de User también es el que tiene la responsabilidad de sincronizar una historia de usuario con un Pull Request.

5.3.2.1.3 OWNER

En la ingeniería de software tradicional los requisitos del sistema forman parte del proceso de adquisición, siendo por tanto responsabilidad del cliente la definición del problema y de las funcionalidades que debe aportar la solución. No importa si se trata de gestión tradicional o ágil, la definición de la pila del producto siempre es responsabilidad del cliente. En TG los usuarios con rol Owner son los responsables de interpretar los requerimientos del cliente y darlos de alta dentro de la herramienta.

5.3.2.1.4 OTHER

En TG a los usuarios con rol "Other" se les permite observar los detalles del sistema. Es decir, examinar el avance del proyecto. No se les permite realizar acciones que aporten al proceso de desarrollo.

5.4 GESTIÓN DEL DESARROLLO

Luego de haber creado el proyecto y haber definido el personal que integrará el mismo, comienza el proceso de gestión del desarrollo. TG se adhiere al modelo de desarrollo en etapas de ingeniería de software:

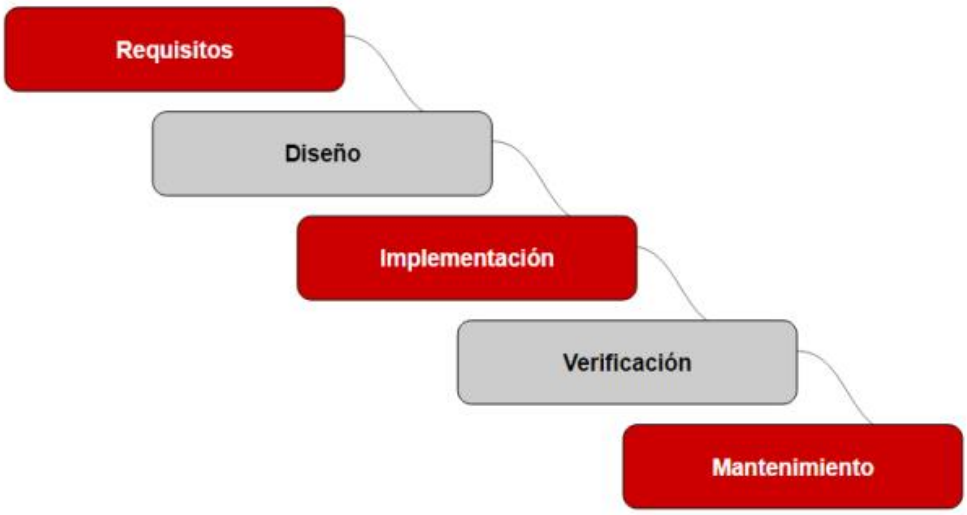

Ilustración 10 – Desarrollo de software en etapas

Dispone numerosas estrategias para supervisar todo estos procesos, comenzando en la etapa de licitación de requerimientos, hasta la etapa del mantenimiento del producto:

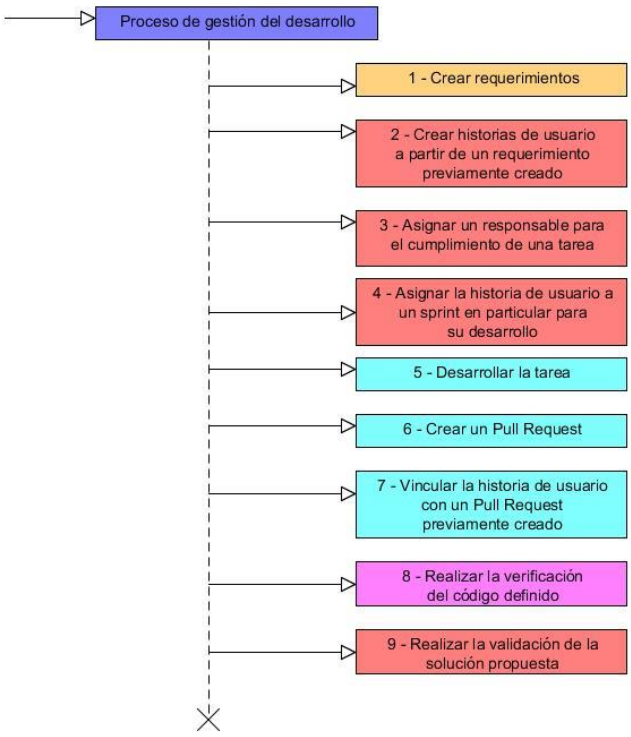

Ilustración 11 - Gestión del desarrollo

- Los colores de los recuadros de la Ilustración 11 - Gestión del desarrollo, se diferencian según el rol destinado a realizar dichas acciones. El cuadro 1 es responsabilidad del rol Owner, los cuadros 2, 3, 4 y 9 son responsabilidades del rol Master, 5, 6 y 7 son responsabilidades del rol User y el 8 puede ser resuelto por cualquier rol del sistema.
- Estos procesos están puestos en el orden en que deben ser realizados. Esto no quiere decir que, por ejemplo, luego de creado un requerimiento, sea necesario crear una historia de usuario. El usuario podría crear varios requerimientos, luego crear varias historias de usuario y después vuelto a crear aún más requerimientos. Es decir, varias de estas actividades pueden realizarse en conjunto.

5.4.1 ORGANIZACIÓN DE LA HERRAMIENTA

Ilustración 12 – Organización de la herramienta

TG se encuentra dividido en 4 grandes secciones o vistas, consiguiendo de esta manera independizar diversos procesos y reducir la complejidad de la herramienta. El acceso a dichas vistas esta manipulado por una barra de navegación en la parte superior de la pantalla, tal como se observa en la Ilustración 12 – Organización de la herramienta.

Dichas vistas son accesibles según el rol del usuario que utiliza la herramienta. El rol Master, por ser el líder y el encargado de gestionar el trabajo del equipo, puede acceder a todas las vistas disponibles. En cambio, el rol Owner y el rol Member, tienen una visión más acotada de la herramienta. Al rol Owner solo se le permite acceder a la vista de Product Backlog. Esto se debe a que dichos usuarios son los responsables de generar y visualizar el estado de todos los requisitos del sistema. Al rol User solo se le permite acceder a la vista de Development. Esto se debe a que son los responsables de desarrollar las historias de usuario. Al rol Other se les permite acceder a la vista de Product Backlog y a la vista de Sprints. Se les permite observar detalles del sistema, es decir, examinar el avance del proyecto. No se les permite realizar acciones que aporten al proceso de desarrollo.

5.4.2 VISTA 1: PRODUCT BACKLOG

En la vista de Product Backlog se permiten crear, modificar y borrar requerimientos de usuarios:

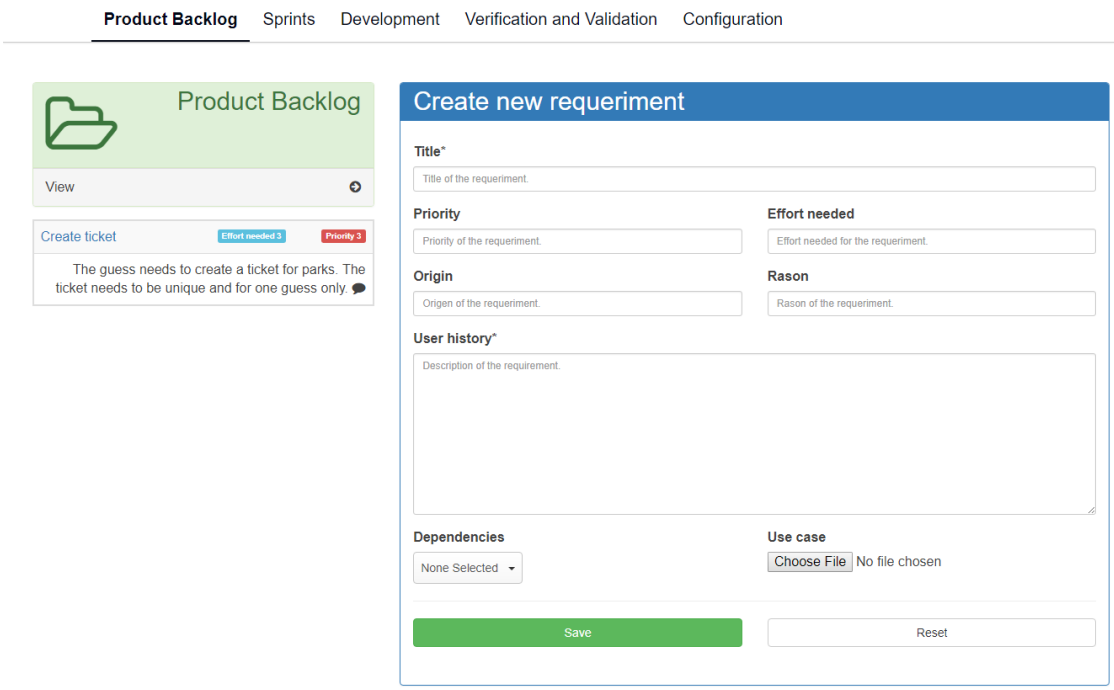

Ilustración 13 - Product Backlog

La primera vista de la herramienta, representada por la Ilustración 13 - Product Backlog, solo puede ser accedida por los roles “Master” y “Owner”. La manipulación de los requisitos del sistema, son responsabilidad de dichos roles. El rol “Owner” es el responsable de la interpretación de los requerimientos y reflejarlos dentro de la herramienta. Es decir, crearlos y modificarlos acorde a las necesidades. El rol “Master” es el encargado de supervisar y manipular los procesos en nombre del equipo que lidera, para luego definir historias de usuario asociadas a cada uno de ellos.

5.4.2.1. CREACIÓN DE UN REQUERIMIENTO

Como se observa en la Ilustración 14 - Creación de un requerimiento, el proceso de creación de un requerimiento es un proceso simple. Dicho proceso debe ser llevado a cabo por el rol “Owner”:

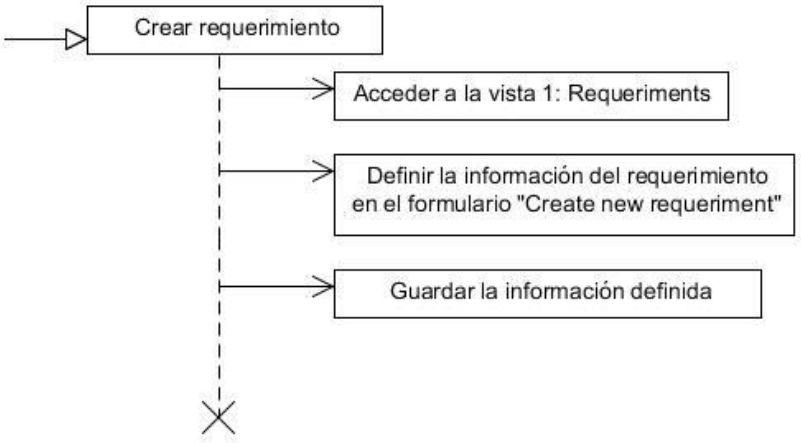

Ilustración 14 - Creación de un requerimiento

5.4.2.1.1. TITULO

El título del requerimiento, es simplemente un medio para identificarlo unívocamente dentro de la herramienta. El mismo debe contener información que luego pueda utilizarse dentro del proceso de búsqueda de historias de usuario.

5.4.2.1.2. DESCRIPCIÓN DE LA FUNCIONALIDAD/REQUISITO

Habitualmente, la pila del producto se empieza a elaborar como resultado de una reunión de “tormenta de ideas”, o "fertilización cruzada", o un proceso de “Exploración” (eXtreme Programming) donde colabora todo el equipo. El campo de descripción representa un medio para definir todas las cuestiones abordadas en estas reuniones, pudiendo escribir libremente los temas tratados y todas aquellas características que posteriormente beneficien al proceso de gestión de historias de usuario.

5.4.2.1.3. PRIORIDAD

Todo requerimiento surge con un indicador de prioridad que representa la urgencia del desarrollo del mismo. Es por esto que TG dispone este mecanismo para representar dicha prioridad sobre cada requerimiento.

5.4.2.1.4. PRE ESTIMACIÓN DEL ESFUERZO NECESARIO

Esta característica se utiliza para definir un pre estimación del esfuerzo necesario que requerirá el desarrollo del requerimiento. Es necesario utilizar este campo para representar cuestiones como estimaciones de tiempo, estimaciones de cantidad de personar necesario, etc.

5.4.2.1.5. RAZÓN DE LA EXISTENCIA

Esta característica se utiliza para definir la razón de la existencia del requerimiento. La misma puede ser porque un cliente solicito, o simplemente porque se quiere evaluar la mejora económica que infiere el agregado de dicha funcionalidad.

5.4.2.1.6. ORIGEN

Esta característica se utiliza para definir el origen del requerimiento. El mismo puede ser por parte del cliente, o por parte de la propia organización.

5.4.2.1.7. POSIBLES DEPENDENCIAS.

Esta característica se utiliza para marcar dependencias entre requerimientos.

5.4.3 VISTA 2: SPRINTS

La segunda vista de la herramienta, representada por la Ilustración 15, recibe el nombre de Sprints e integra principalmente la gestión de las historias de usuario. Esta vista únicamente puede ser accedida por el rol Master y está destinada a administrar el trabajo de cada equipo, manipulando las tareas y asignaciones de los usuarios.

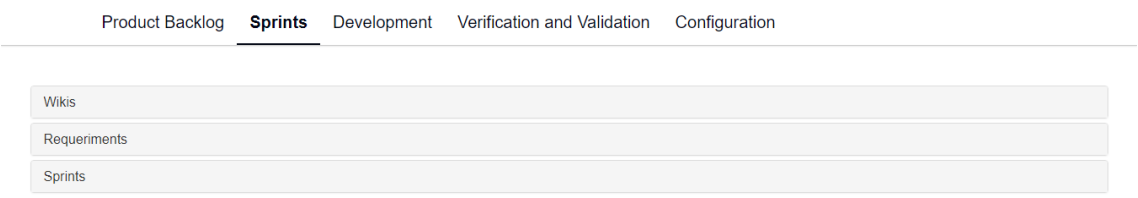

Ilustración 15 – Vista 2: Sprints

5.4.3.1 SECCIÓN 1: WIKIS

Una wiki es un concepto que se le da a documentos que pueden ser modificados directamente desde un sistema informático, donde usuarios crean, modifican o eliminan contenidos que, generalmente, son compartidos. TG utiliza las wikis para apoyar al desarrollador en el cumplimiento de los objetivos de una historia de usuario. Es habitual que los wikis incluyan información que apoye al desarrollador al cumplimiento del objetivo de la historia de usuario. Como por ejemplo, información de archivos necesarios a modificar, información de equipos encargados del mantenimiento del módulo, reglas de negocio, etc. Esto reduce el tiempo del proceso de investigación previo al proceso de desarrollo.

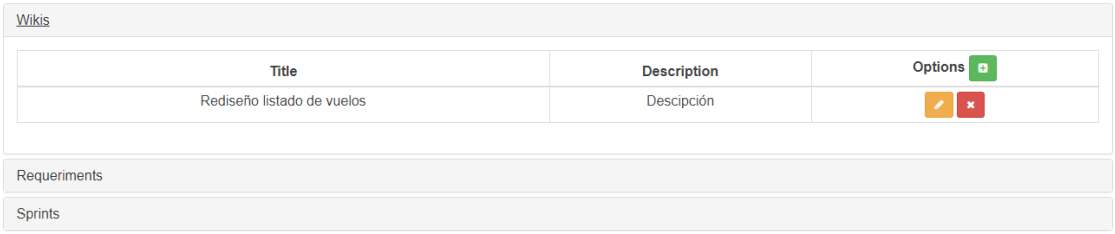

Ilustración 16 – Sección 1: Listado de wikis

Por otro lado, la wiki puede ser utilizada para difundir información del diseño de una vista, la arquitectura de la respuesta de un servicio de api, el diagrama de clases de un modelo de backend, etc. La forma más común de difundir esta información es a través de imágenes.

Generalmente las personas encargadas de definir estas características, utilizan sus propias herramientas. Por ejemplo, los diseñadores suelen utilizar Photoshop para organizar los elementos de una vista, mientras que los arquitectos de api rest, suelen utilizar herramientas como UMLet para definir los diagramas de clases. Todas estas herramientas, generan imágenes con toda la información definida, que necesita adjuntarse a la historias de usuario. Es por esto que TG define esta característica como estrategia de apoyo en la gestión del desarrollo.

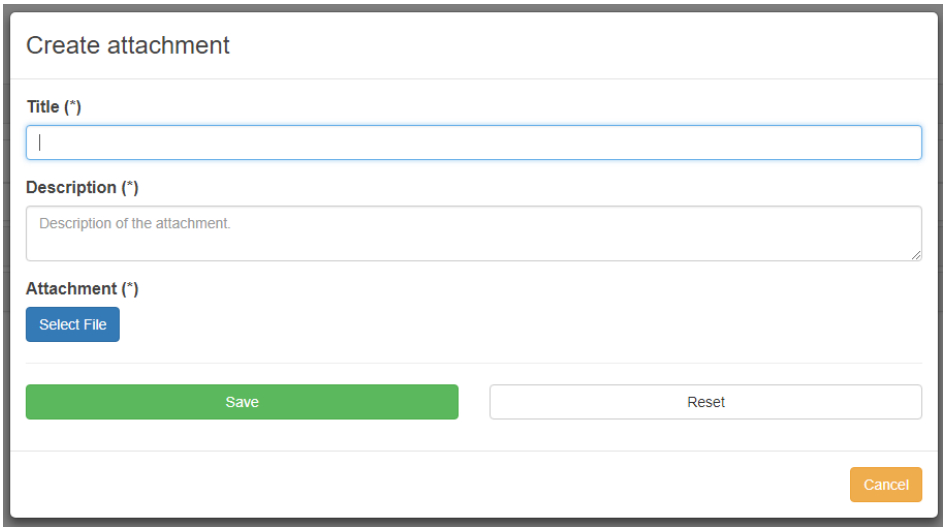

Ilustración 17 – Crear wiki

5.4.3.2 SECCIÓN 2: REQUERIMENTS

En esta sección TG lista todos los requerimientos de la pila de producto, pero desde la perspectiva de un usuario con rol Master. Es decir, únicamente contiene la información necesaria para que el usuario Master pueda identificarlo dentro del sistema. De esta forma, TG consigue aumentar la agilidad del proceso de identificación de un requerimiento particular.

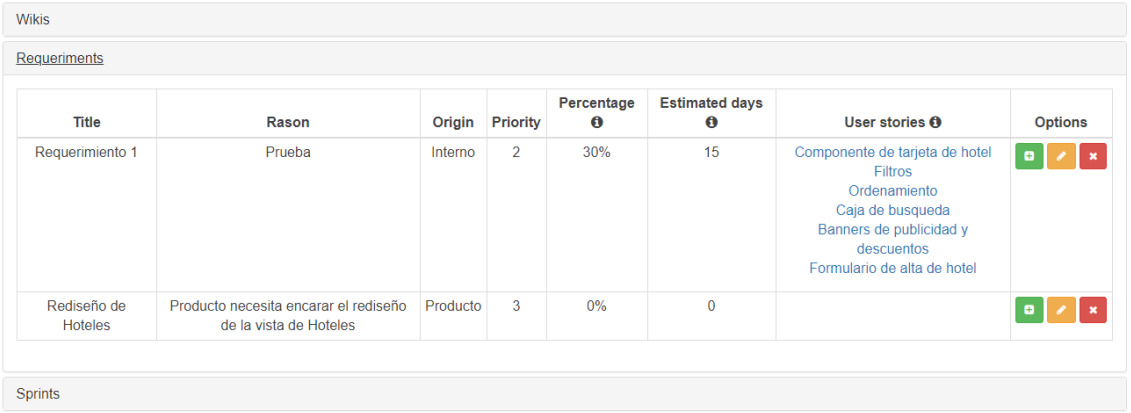

Ilustración 18 – Sección 2: Listado de requerimientos

Como se observa en la Ilustración 18 – Sección 2: Listado de requerimientos, se muestran varias características de cada uno de ellos, aunque no se dispone la totalidad de la información. Particularmente, hay tres atributos que es necesario remarcar.

1. **Porcentaje:** TG registra el porcentaje de resolución de cada requerimiento. Esta propiedad la calcula en base a los porcentajes de resolución de cada historia de usuario asociada.
2. **Días estimados:** TG registra la cantidad de días estimados para la resolución de un requerimiento. Esta propiedad la calcula en base a los días estimados de cada historia de usuario asociada.
3. **Historias de usuario:** TG lista todas las historias de usuario creadas a partir de cada requerimiento, permitiendo acceder a un detalle en el que se pueda observar todas las características de la misma:

Ilustración 19 – Detalle de historia de usuario

5.4.3.2.1 PROCESO DE CREACIÓN DE UNA HISTORIA DE USUARIO

Las historias de usuario son el elemento principal de comunicación en las metodologías agiles. Son el medio que utilizan los procesos de gestión del software para brindar toda la información que necesita saber un desarrollador y para incorporar o modificar una funcionalidad del sistema de software. En TG todas las historias de usuario son creadas a partir de un requerimiento de la pila de producto. Es por esto que la opción está vinculada a cada requerimiento en particular.

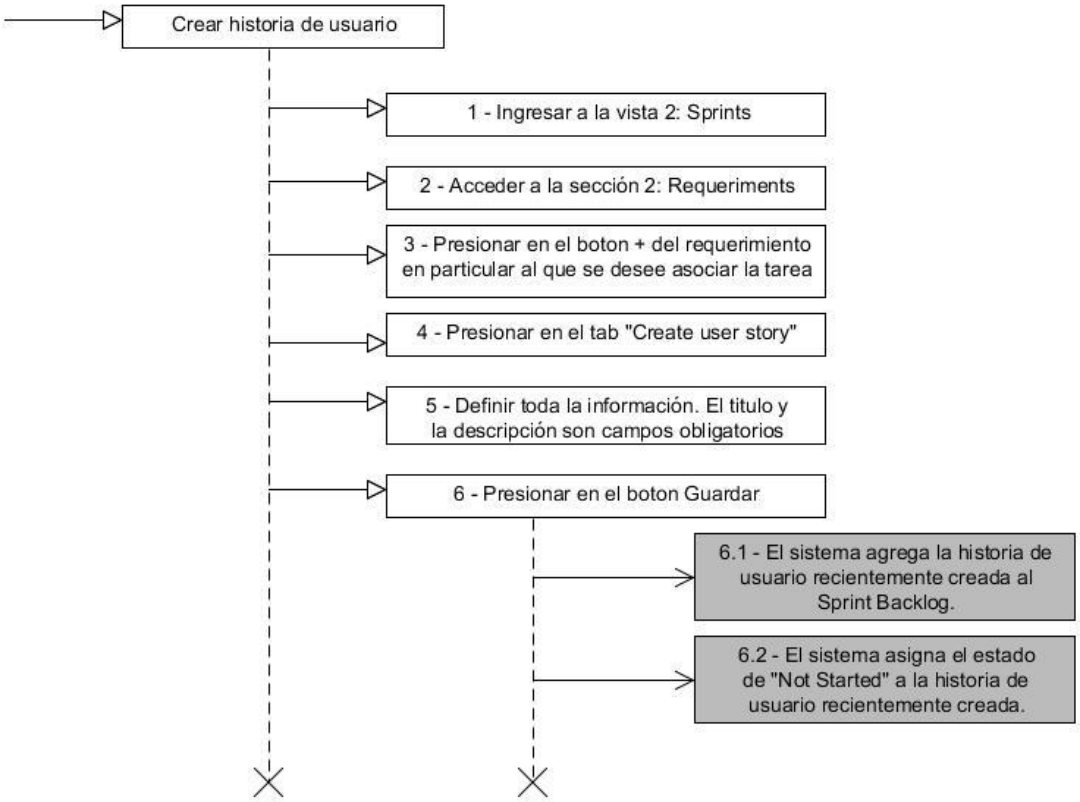

Ilustración 20 - Crear historia de usuario

5.4.3.2.1.1 DETALLE DEL REQUERIMIENTO

Luego de presionar el botón para crear una historia de usuario, TG muestra una vista con 3 sub-secciones. En la primera, se visualiza un detalle del requerimiento particular al que se terminará vinculando cada historia de usuario creada en este punto. A diferencia de Trello y Jira, TG permite crear múltiples historias de usuario dentro de la misma vista. Con esto consigue aumentar la agilidad del proceso de creación de historias de usuario.

La idea de mostrar este detalle, es proveer un medio ágil para corroborar toda la información definida en el requerimiento. De esta forma si dispusiera casos de uso, dependencias, o

simplemente una descripción lo suficientemente clara, el usuario puede apoyarse en este detalle para corroborar información a medida que crea cada historia de usuario.

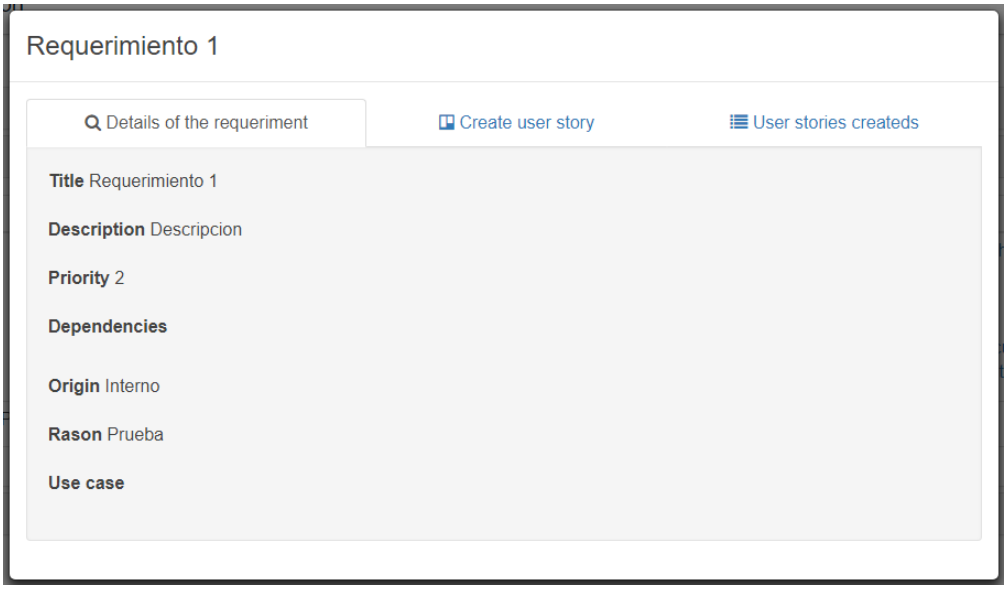

Ilustración 21 – Detalle de un requerimiento

5.4.3.2.1.2 CREAR HISTORIA DE USUARIO

La segunda pestaña se utiliza para dar de alta una historia de usuario a partir del requerimiento específico.

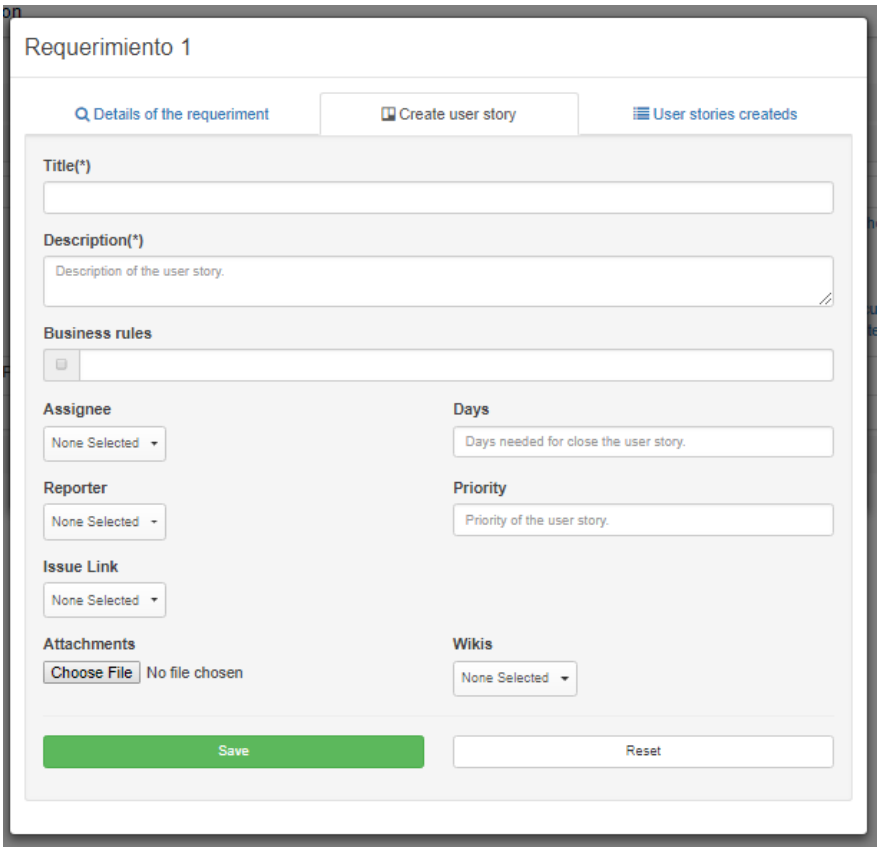

Ilustración 22 – Crear historia de usuario

Es necesario observar que la información pretendida no se corresponde con la establecida por Scrum o Kanban, ni por Jira o Trello. La causa de esto, es que se agregó y/o modifico el comportamiento de ciertos atributos que regularmente caen en desuso, con el objetivo de agilizar el proceso de creación.

5.4.3.2.1.2.1 TÍTULO

Al igual que en Scrum y Kanban, el título de la historia de usuario se corresponde con una breve descripción para identificar la mismo dentro del sistema. Es necesario que sea conciso y resuma brevemente la descripción de la tarea.

5.4.3.2.1.2.2 DESCRIPCIÓN

También al igual que en Scrum y Kanban, La descripción indica brevemente el tópico a desarrollar. Debe contener toda la información necesaria para desarrollar la historia de usuario.

En TG una historia de usuario puede ser creada simplemente especificando el título y la descripción. Esto se debe a que habitualmente los equipos suelen reunirse y planificar tareas dentro de reuniones distendidas, en las que no pueden darse el lujo de perder tiempo definiendo toda la información detallada para cada tarea pensada. Debido a la agilidad que disponen Jira o Trello, los Master no suelen definir las historias de usuario en medio de estas reuniones., en su lugar suelen contar de su memoria, o anotar puntos clave en papel. Entonces luego de terminada la reunión, pasan toda la información en limpio, es decir, cada funcionalidad que se conversó y se decidió desarrollar en el sistema. TG dispone este medio, para que los Master a medida que transcurren estas reuniones, puedan especificar de forma resumida cada historia de usuario y luego complementar con información más específica sin necesidad de apoyarse en ninguna otra alternativa.

5.4.3.2.1.2.3 BUSINESS RULES

Esta propiedad se utiliza para definir el desglose de todas las acciones que deben realizarse para el completar la tarea. De esta forma, el desarrollador puede llevar el control de las sub-tareas que va completando y el Master puede llevar registro del desarrollo.

5.4.3.2.1.2.4 ASSIGNE

Esta propiedad se utiliza para asignar la historia de usuario a un responsable para su desarrollo. Aunque en primera instancia es probable que el Master no conozca que desarrollador es el indicado para el cumplimiento de la tarea, suele ocurrir en determinadas situación que una historia de usuario es la continuación de otra, o simplemente están relacionadas directamente. En estos casos el Master puede no necesitar analizar con anticipación quien debe ser el responsable del desarrollo de la misma.

5.4.3.2.1.2.5 DAYS

Esta propiedad se utiliza para estimar el tiempo necesario para la resolución de una tarea. TG propone que el margen sea de 1 – 10. Scrum específica que si el desarrollo de una tarea está estimado en más de 10 días, es porque está mal definida. Entonces se utilizó esta aproximación.

La herramienta utiliza esta propiedad para estimar el tiempo de resolución de un Requerimiento específico. Supongamos que el Requerimiento “Req1” tiene asignado las historias de usuario “Hist1”, “Hist2” y “Hist3”. Supongamos también que “Hist1” tiene estimada una resolución de 3 días, “Hist2” 2 días e “Hist3” 4 días. También supongamos que “Hist1” e “Hist2” están asignadas a “Usr1”, mientras que “Hist3” está asignada a “Usr2”. Como “Hist1” e “Hist2” están asignadas al “Usr1”, TG asume que ambas tareas no pueden realizarse en paralelo. Pero “Hist3” si podría realizarse al mismo tiempo que “Hist1” e “Hist2”. Entonces en este ejemplo, la herramienta asumirá un tiempo de resolución de 5 días.

5.4.3.2.1.2.6 REPORTER

Se utiliza para mantener notificado usuarios sobre el estado de una historia de usuario. En muchas ocasiones una tarea se encuentra bloqueada porque necesita que previamente se finalice la resolución de otra. TG utiliza está propiedad para este cometido. El usuario responsable de la historia de usuario bloqueada, puede ser agregado como "Reporter" y así realizar el seguimiento de la misma.

5.4.3.2.1.2.7 PRIORITY

Se utiliza para controlar el orden de prioridad de las historias de usuario.

5.4.3.2.1.2.8 ISSUE LINK

La sección de Issue link se utiliza para marcar relaciones entre historias de usuario. Esta propiedad es la principal característica a tener en cuenta durante el proceso de búsqueda de una historia de usuario en particular. Si esta propiedad se utiliza con cuidado, un desarrollador simplemente con navegar a través de las tareas relacionadas, podría encontrar la historia de usuario que está buscando.

5.4.3.2.1.2.9 ATTACHMENTS

Cuando estamos ante un proyecto de software complejo y de gran tamaño, los procesos de gestión suelen considerar eficaz utilizar equipos para definir, administrar y controlar los diseños del sistema. Lo ideal sería contar con una herramienta que apoye a los diseñadores a definir estos diseños, devolviendo el código estático html y css del tópico en particular.

Dentro del proceso de gestión del proyecto, los diseñadores generan imágenes de las vistas del sistema. Muchas veces estas imágenes contienen información que es interpretada por los desarrolladores para cumplir su trabajo. Por ejemplo, tamaños, fuentes, colores, espaciados, etc. TG utiliza la sección de Attachments para adjuntar estas imágenes a las historias de usuario.

5.4.3.2.1.2.10 WIKIS

TG utiliza las Wikis como medio para difundir información que apoye al desarrollador al cumplimiento del objetivo de una historia de usuario. Esta información puede estar compuesta por documentos de negocio, reglas o procesos (requerimientos no funcionales), información de archivos necesarios a modificar, información de equipos encargados del mantenimiento del módulo, como por ejemplo nombre de los equipos, nombre del Master, información de contacto, etc. En consecuencia, la propiedad de Wikis se utiliza para seleccionar wikis dadas de alta previamente en el sistema y vincularlas a las historias de usuario.

5.4.3.2.1.2.11 AGILIDAD DEL PROCESO DE CREACIÓN DE HISTORIAS DE USUARIO

Hay otras cuestiones que también benefician a la agilidad del proceso de creación de las historias de usuario. Primero que nada, el listar todos los requerimientos del sistema y definir un medio para crear tareas a partir de cada uno de ellos, trae grandes ventajas:

- Elimina la necesidad de tener que vincular un requerimiento luego o durante el proceso de creación.
- Permite llevar control del estado en el que se encuentra cada requerimiento.
- Permite realizar la estimación del tiempo necesario para la finalización de cada requerimiento con mayor grado de certeza.
- Permite organizar todas las historias de usuario por requerimiento, agilizando considerablemente el proceso de búsqueda de una en particular.

Otra característica que posee TG, es que dispone un medio para crear múltiples tareas utilizando la mínima información, dentro de la misma vista. Una historia de usuario puede ser creada simplemente especificando el título y la descripción. Luego de haber llenado esos dos campos y de haber presionado en el botón de guardar, la herramienta creará la tarea con toda la información definida y limpiará el formulario para que se puedan seguir agregando tareas al al proyecto. De esta forma, se permiten definir múltiples historias de usuario de forma resumida. TG dispone un medio para luego complementar cada tarea con información más detallada.

Otra característica que dispone TG en este punto, es que permite navegar entre pestañas manteniendo toda la información especificada. De esta forma si el usuario necesitara acceder al detalle del requerimiento para corroborar una característica, o a la pestaña del listado de todas las tareas creadas a partir de dicho requerimiento, no perdería la información definida.

5.4.3.2.1.3 HISTORIAS DE USUARIO CREADAS

La última pestaña lista la totalidad de historias de usuario creadas a partir del requerimiento.

Ilustración 23 – Listado de historias de usuario creadas a partir de un requerimiento

5.4.3.2.1.3.1. ESTADOS

En la Ilustración 23 se puede observar que todas las historias de usuario contienen el label "**Not started**". Este label representa el estado en el que se encuentra dicha historia de usuario.

En TG, todas las historias de usuario pasan por los estados representados en la Ilustración 24

Ilustración 24 – Listado de estados de historias de usuario

TG utiliza dichos estados para realizar la gestión del desarrollo. Cada uno de ellos representa la etapa en la que se encuentra dicha historia de usuario, es decir el porcentaje de resolución de la misma.

1. **Not started:** 0%
2. **Ready for dev:** 25%
3. **Dev completed:** 50%
4. **Ready for test:** 75%
5. **Closed:** 100%

TG utiliza este porcentaje, para determinar el porcentaje de resolución del requerimiento al que están vinculados.

Un usuario no puede manipular por sistema los estados de las historias de usuario. Los mismos son manipulados automáticamente por la herramienta. De esta forma, pueden ser utilizados por los analistas del producto como guía, apoyando a los mismos a determinar o estimar el tiempo restante para el cumplimiento de una tarea.

El primer estado en el que se puede encontrar una historia de usuario es el de **"Not Started"**. Este estado se asigna al momento en que se crea una historia de usuario. Como lo indica su nombre, el mismo representa que la historia de usuario no se encuentra asignada a ningún Sprint para su desarrollo.

El segundo estado en el que se puede encontrar una historia de usuario es el de **"Read for Dev".** Este estado se utiliza para indicar que la historia de usuario está lista para ser desarrollada. El mismo se asigna al momento en que una historia de usuario es movida del sprint Backlog a un Sprint específico. Si resultará necesario mover de nuevo la historia de usuario al Sprint Backlog, la misma pasaría a estar de nuevo en el estado de **"Not Started"**.

El tercer estado en el que se puede encontrar una historia de usuario es el de "**Dev Completed**". Este estado representa que la historia de usuario fue desarrollada y que dispone un Pull Request asociado a la misma.

El cuarto estado en el que se puede encontrar una historia de usuario es el de **"Ready for test"**. Este estado representa que el pull Request asociado a la tarea fue fusionado con el repositorio remoto. Entonces en este punto, lo único que faltaría para completar el proceso de gestión del desarrollo, es la validación del Master de que el desarrollo propuesto está acorde con lo especificado en los ambientes superiores.

El último estado en el que se puede encontrar una historia de usuario puede ser tanto el de **"Closed"**. Dicho estado representa que la historia de usuario cumplió con las expectativas del cliente. Entonces en este punto, la misma se considera completada.

Existen 2 labels más dentro de la herramienta, que también representan estados de la historia de usuario, pero los mismos se consideran fuera del proceso de gestión del desarrollo. El primero de ellos es el estado de "**Carry Over**". Dicho estado representa que una historia de usuario no alcanzó a completarse previo al proceso de finalización del Sprint. Entonces la misma necesita

ser desarrollada en el siguiente Sprint. TG utiliza esta marca para identificar las historias de usuario que se encuentran en dicho estado.

El último estado en el que se puede encontrar una historia de usuario es el de "Pull Request". Dicho estado representa que la tarea tiene un Pull Request asociado.

5.4.3.3 SECCIÓN 3: SPRINTS

La sección "Sprints" está formada por los Sprints del sistema. Dentro de esta sección el Master realiza la gestión de las historias de usuario.

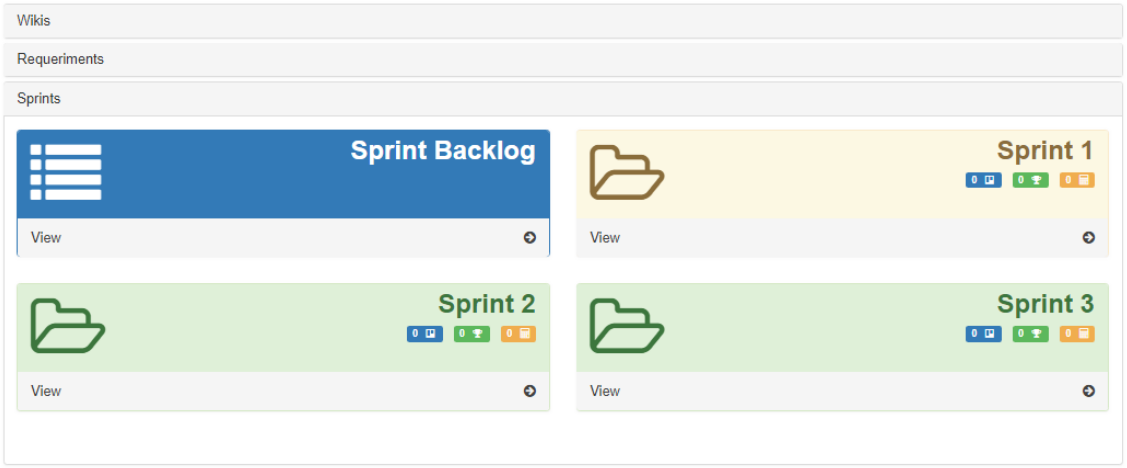

Ilustración 25 – Sección 3: Sprints

La Ilustración 25 – Sección 3: Sprints, representa la vista de la sección Sprints. Hasta el momento pueden observarse 4 listas. El Sprint Backlog como su nombre lo indica representa la pila de sprint. La lista amarilla, en este caso el Sprint 1, representa el sprint activo dentro del sistema, es decir, el sprint actual. Mientras que los Sprints verdes, en este caso los Sprints 2 y 3, representan Sprints futuros.

5.4.3.3.1 SPRINT BACKLOG

Como se observa en la Ilustración 26 – Sprint backlog, en la pila de Sprint se almacenan todas las historias de usuario que fueron recientemente creadas, pero que aún no fueron seleccionadas para su desarrollo. Es por esto que el estado en el que se encuentra cada una de ellas es "Not Started". A su vez, en la pila de Sprint pueden existir tareas sin usuario asignado, tal como las tareas 3, 4, 5 y 6. Esto se debe a que aún no se ha indicado quien es el usuario responsable del desarrollo de las tareas faltantes.

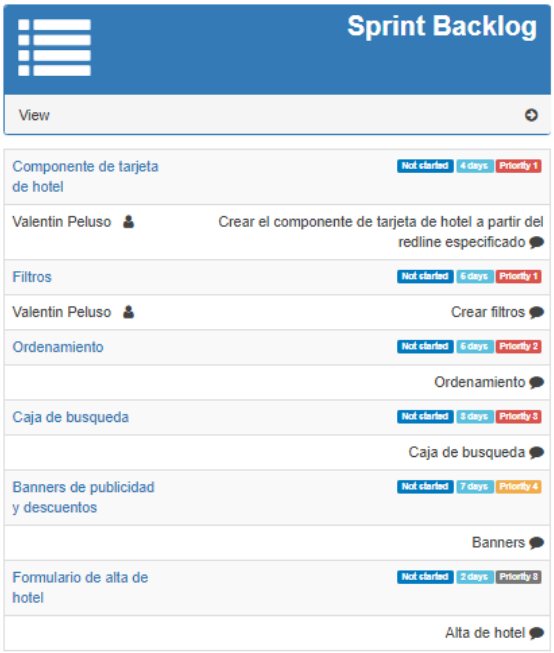

Ilustración 26 – Sprint backlog

5.4.3.3.1.1. ASIGNAR UN RESPONSABLE PARA EL DESARROLLO DE UNA TAREA

Cada tarea dentro de la pila de Sprint dispone un link en el nombre de la misma. Presionando en ese link, se abre el mismo formulario de creación de una historia de usuario, pero con toda la información pre-cargada. De esta forma el Master no solo puede asignar un usuario para su desarrollo, sino que también puede modificar o agregar otros atributos, aumentando la velocidad del proceso de creación de cada una de ellas:

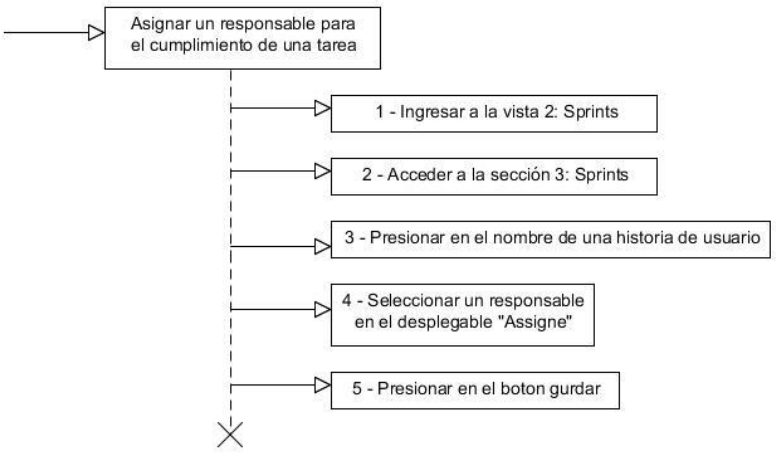

Ilustración 27 – Seleccionar un responsable para el cumplimiento de una tarea

5.4.3.3.1.2. ORDENAMIENTO DEL SPRINT BACKLOG

Dentro del Sprint Backlog, las historias de usuario se encuentran ordenadas por prioridad. TG utiliza esta propiedad como medio de ordenamiento del Backlog, consiguiendo que se visualicen primero las tareas con mayor prioridad. Otra característica que dispone TG, es que solo permite asignar historias de usuario a un sprint específico, según el orden de prioridad que poseen. Es decir, si un usuario N tiene asignada una historia de usuario con prioridad 4 en el sprint 1, al este usuario no se le podrán asignar historias de usuario que dispongan prioridad menor a 4 dentro del mismo sprint. De esta forma se fuerza que las tareas más prioritarias sean las primeras en resolverse.

Habitualmente la prioridad de una tarea es relativa al momento del proyecto. Es por esto que TG permite modificar este atributo para que no quede una historia de usuario en inanición. Queda bajo la responsabilidad del usuario Master el decidir cuándo es necesario subirle o bajarle la prioridad a una tarea.

Aun así, existe una situación en la que el usuario Master se ve imposibilitado de modificar la prioridad de una tarea, debido al estado en el que se encuentra la misma. Cuando una tarea no pudo ser cerrada antes de la finalización del sprint, la misma entra en el estado de Carry Over. TG interpreta a las historias de usuario con dicho estado como prioridad 0, es decir, las considera con mayor prioridad que cualquier otra tarea.

5.4.3.3.1.3. ASIGNAR UNA HISTORIA DE USUARIO A UN SPRINT EN PARTICULAR

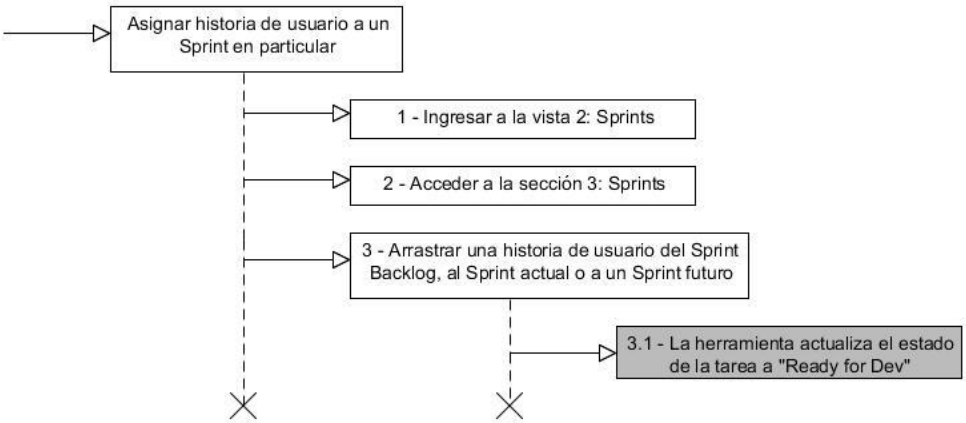

Ilustración 28 - Asignar historia de usuario a un Sprint en particular

Otra de las operaciones permitidas dentro de la pila de sprint, es asignar una historia de usuario a un sprint en particular. Esta acción se corresponde con una de las operaciones más básicas de todas las herramientas de gestión de proyectos. En TG las tareas son arrastrables, con el objetivo de darle mayor agilidad a esta operación. Las mismas pueden desplazarse de la pila de sprint a un sprint en particular, entre Sprints, o también retornarla de un sprint al Backlog. Estas

acciones repercuten al estado en el que se encuentra cada una de ellas. Cuando una tarea se encuentra en el Sprint Backlog, dispone el estado de *"Not Started"*. Cuando la misma es desplazada a un sprint en particular, TG le asigna el estado a *"Ready for dev"*. Caso similar ocurre cuando se decide regresar una tarea a la pila de Sprint, la misma vuelve al estado de *"Not Started"*.

5.4.3.3.2 SPRINT

La propiedad de prioridad, los estados en los que se encuentran las historias de usuario y los roles que disponen cada usuario, son las características más importantes dentro del proceso de gestión del desarrollo. A través de estas 3 propiedades, TG gestiona y administra todas las historias de usuario, el personal y mantiene el proyecto de software bajo control.

Ilustración 29 – Sprint

Como se observa en la Ilustración 29 – Sprint, en un sprint se almacenan las historias de usuarios que están listas para ser desarrolladas.

5.4.3.3.2.1 DETALLE DEL SPRINT

TG utiliza los 3 labels que se observan en la parte inferior del nombre del sprint, para reflejar información general del estado en el que se encuentra cada uno de ellos. Con estos 3 labels el Master pude obtener un panorama de la etapa en el que se encuentra dicho Sprint, sin necesidad observar y hacer un análisis de todas las historias de usuario asignadas. También puede utilizarlo para tomar decisiones, como por ejemplo definir en qué momento resulta conveniente cerrar el sprint o indicar la cantidad de días estimados para la finalización del mismo. De esta forma se reduce la complejidad del proceso de gestión del desarrollo y se aumenta la agilidad del proceso de gestión del proyecto.

El primer label ***1** representa la cantidad de historias de usuarios asignadas al sprint. Comúnmente las organizaciones suelen tener predefinido la cantidad de historias de usuario que se incorporaran a cada incremento, siempre por cuestiones de tiempos, estadísticas y presupuestos. Este label brinda esta información para poder mantener los estándares de la empresa y utilizarlos como guía del grado de mejora que se logra con el correr del tiempo.

El segundo label ***2** representa la cantidad de historias de usuario cerradas sobre el total de asignadas. Este label apoya a la organización a determinar si se está trabajando dentro de los tiempos estimados, si el Sprint esta sobre valuado o si es necesario agregar o quitar tareas del mismo. De esta forma puede obtenerse un panorama del estado en el que se encuentra el sprint y tomar decisiones en base a esta característica.

El tercer label ***3** representa la cantidad de días estimados para la finalización del sprint. TG calcula esta propiedad a partir de la suma de los días estimados de cada historia de usuario, utilizando el método de "camino crítico" a nivel de Sprint. Es decir asume que las tareas asignadas a diferentes usuarios pueden realizarse al mismo tiempo y en paralelo. Pero cuando un usuario tiene asignada más de una tarea, TG asume que hasta no tener finalizada una de ellas, no podrá comenzar con la siguiente. Entonces en este caso, se sumarán los tiempos de ambas tareas.

5.4.3.3.2.2 ESTADOS DE LOS SPRINTS

Los Sprints del sistema pueden encontrarse en 3 posibles estados: ser el *"Sprint actual"*, ser un *"Sprint futuro"*, o ser un *"Sprint cerrado"*. Dichas distinciones ayudan a la gestión del proyecto a controlar el estado del proyecto.

En TG solo se puede disponer un único *"Sprint actual"*, representado por el color amarillo. Dicho estado representa, como el nombre lo indica, al Sprint actual del sistema. La mayoría de las operaciones que se realizan con los Sprints, por ejemplo finalizarlo, vincular un pull Request con una historia de usuario, marcar la validación y verificación del desarrollo de una tarea, etc., se habilitan únicamente en este estado.

Todos los Sprints posteriores al "Sprint actual", son "Sprints futuros" y están representados por el color verde. A este tipo de Sprints solo se les permite asignar o remover tareas.

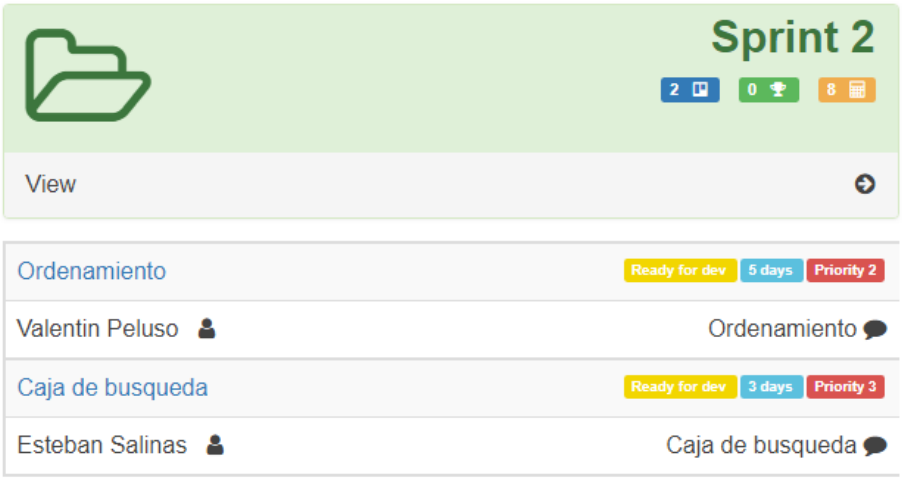

Ilustración 30 – Sprint futuro

Todos los Sprints predecesores al "Sprint actual", son "Sprints cerrados". Estos Sprints representan incrementos generados. Entonces los mismos pueden utilizarse como guía para desarrollos futuros, para el análisis de rendimiento por la organización, etc.

5.4.3.3.2.3 FINALIZACIÓN DEL SPRINT

La finalización del Sprint es un proceso que puede llevarse a cabo en el momento en el que el Master le resulte apropiado. Dicho acción se lleva a cabo presionando en el botón que aparece abajo del "Sprint Actual" y es lo que termina generando el cambio de estado del mismo.

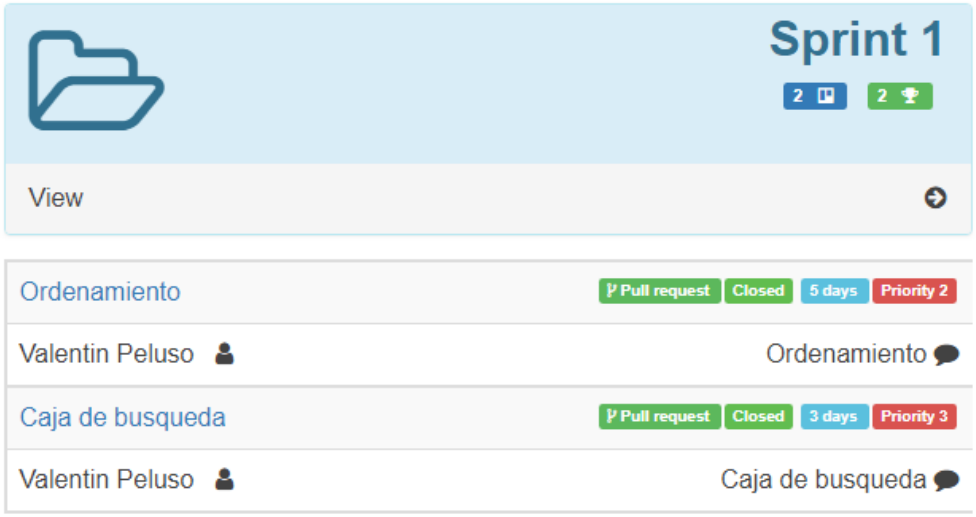

Ilustración 31 – Sprint cerrado

Únicamente se puede tener un "Sprint actual", siendo el mismo el único que puede finalizarse. Esta acción transforma al "Sprint actual" en un "Sprint cerrado" y al siguiente "Sprint Futuro"

en el próximo "Sprint actual". De esta forma TG consigue administrar y organizar el desarrollo de forma lineal, manteniendo el control dentro del proyecto.

Utilizar esta estrategia trae grandes ventajas al proceso de gestión del proyecto. Por un lado, quita la necesidad que define Scrum de tener que adaptarse a un calendario, consiguiendo que cada equipo sea el propio responsable de tomar la decisión del momento en que resulta conveniente cerrar el sprint. Esto elimina la posibilidad de tener un sprint sin ningún incremento generado. También permite adaptarse a determinaciones propuestas por la propia organización, como por ejemplo forzar el cierre de un Sprint por razones burocráticas o de negocio. De esta forma se elimina la necesidad de tener que esperar hasta la finalización del sprint para solucionar un bug fix.

En TG cuando el usuario Master decide finalizar el Sprint, todas las historias de usuario que aún no se encontraban en el estado de *"Closed"*, se auto asignan al siguiente Sprint, es decir, al próximo "Sprint Actual". Dicha acción agrega el estado de *"Carry Over"* a la tarea, manteniendo el estado en el que se encontraban, tal como se observa en la Ilustración 32 – Sprint actual. En este punto, dicha historia de usuario se encontrará en 2 estados, *"Ready for Dev"* y *"Carry Over"*. Todas las tareas en estado de *"Carry Over"*, son consideradas como prioridad 0. Entonces si un usuario tiene asignada una tarea con dicho estado, no se le podrán asignar nuevas tareas hasta no tener la misma finalizada.

Ilustración 32 – Sprint actual

TG utiliza esta estrategia porque trae varias ventajas al proceso de gestión del desarrollo. Por un lado, fuerza a que todas las historias de usuario con estado de *"Carry over"* sean las más prioritarias dentro del sprint. De esta forma una tarea no puede atravesar varios Sprints sin ser

finalizada. A su vez, consigue que los días estimados para la finalización del Sprint, estén más acorde a lo propuesto, pudiendo reflejar con mayor grado de certeza los tiempos de desarrollo.

5.4.4 VISTA 3: DEVELOPMENT

La tercera vista que dispone TG, representada por Ilustración 33, recibe el nombre de "Development", e integra principalmente la gestión del proceso de desarrollo en la etapa de implementación, permitiendo vincular historias de usuario con Pull Request. Esta vista únicamente puede ser accedida por el rol de Member. Está destinada a que cada integrante del equipo, sea el responsable de administrar su desarrollo y de vincular cada historia asignada al Pull Request propuesto.

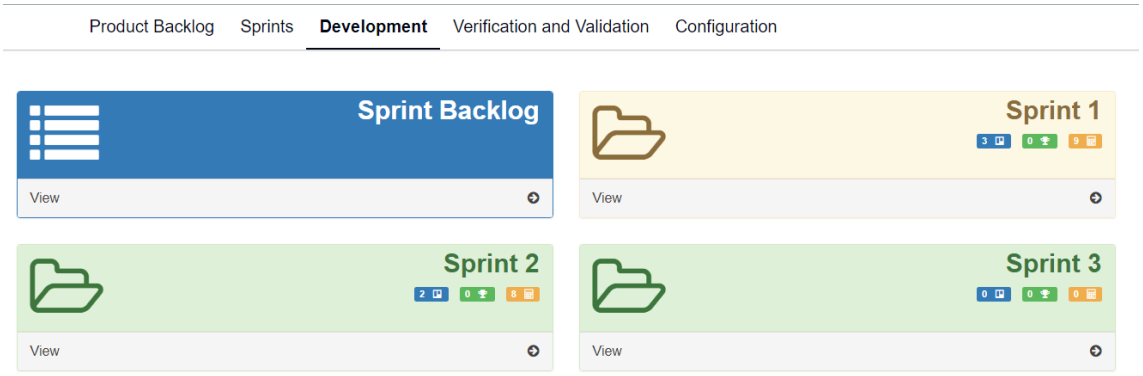

Ilustración 33 – Vista 3: Development

Como se observa en la Ilustración 33, el usuario User también visualiza todos los Sprints del sistema, es decir, el "Sprint actual", los dos Sprints Futuros y en caso de disponer, los Sprints Cerrados.

En numerosas oportunidades un usuario necesita detectar o recopilar información de una tarea. También a veces necesita identificar un Pull Request relacionado que lo beneficie en el desarrollo. Entonces visualizando todos los Sprints del sistema, un User puede entrar a todas las historias de usuario, visualizar su detalle y acceder a los Pull Request vinculados a cada una de ellas.

5.4.4.1 VINCULACIÓN DE HISTORIAS DE USUARIO CON UN PULL REQUEST

TG potencia la creación de tareas atómicas. Entonces, según lo que especifica la teoría definida por las metodologías agiles, la misma debería poder realizarse a través de un único Pull Request. Dado esto, la herramienta dispone un medio para vincular una historia de usuario con un único Pull Request, siendo este el responsable de desarrollar la tarea relacionada. De esta forma no necesita apoyarse en herramientas externas, evitando así aumentar la complejidad de la herramienta.

Un usuario User puede realizar dicha acción, accediendo al detalle de una historia de usuario que tenga asignada. Dentro de esta vista, la herramienta dispone un medio para seleccionar el Pull Request deseado. La herramienta solo permite vincular Pull Request a una historia de usuario que se encuentren dentro del "Sprint Actual" y en estado de "Ready For Dev".

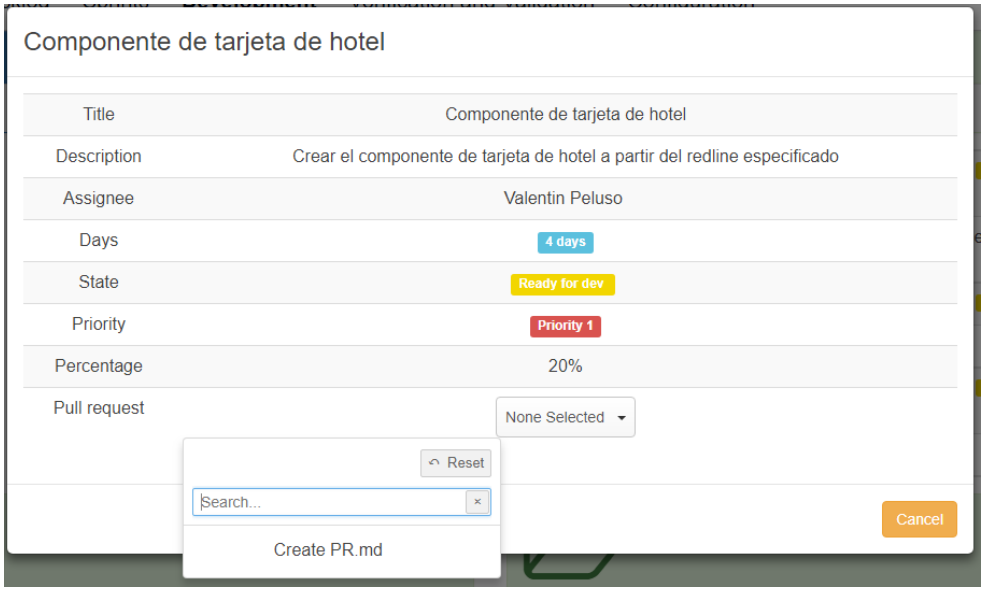

Ilustración 34 – Vinculación de historia de usuario con PR

Otra precondición es que dicho Pull Request necesita estar creado con anterioridad. La herramienta no permite crear pull Request, solo permite vincular historias de usuario con los mismos. También, TG lista únicamente los Pull Request creados por el usuario User asociado a la tarea. Dado que un usuario no suele tener muchos Pull Request abiertos para un repositorio en particular, la identificación del mismo resulta rápida y sencilla. De esta forma se consigue aumentar la agilidad de este proceso.

Ilustración 35 – Historias de usuario con PR vinculado

Como se observa en la Ilustración 35, luego de vinculado el Pull Request con la historia de usuario, TG realiza dos modificaciones. Por un lado, agrega el label de "Pull Request". Este label representa que la historia de usuario tiene un Pull Request asociado, permitiendo acceder rápidamente al mismo en Github. Por otro, pasa la tarea al estado de "Dev Completed". Teóricamente un desarrollador debería crear un Pull Request cuando considera que finalizado el desarrollo de la tarea. La herramienta asume que esta acción, significa que el desarrollo se encuentra completo. De esta forma la misma avanza al 50% del porcentaje de desarrollo.

5.4.4.2 MERGE DE UN PULL REQUEST VINCULADO A UNA HISTORIA DE USUARIO

Luego de creado el Pull Request y vinculado el mismo con una historia de usuario, el siguiente paso dentro del proceso de desarrollo consiste en validar la solución propuesta por el desarrollador. Este proceso se realiza desde Github. Luego de terminado este proceso, el Pull Request se encuentra en condiciones de ser fusionado. Esta acción también se realiza desde Github.

5.4.5 VISTA 4: VERIFICATION AND VALIDATION

Que un Pull Request haya sido fusionado, no quiere decir que la historia de usuario se considere finalizada; sino que se considera aceptada la calidad de la solución propuesta. No se deben considerar como incremento a los prototipos, módulos o sub-módulos, ni partes pendientes de pruebas o integración. Entonces, luego que el Pull Request ha sido fusionado, el siguiente paso dentro del proceso de desarrollo, consiste en probar en algún ambiente superior la solución propuesta.

La vista 4 es la parte del sistema en la que el Master puede seleccionar aquellas historias que se encuentren en el "Sprint actual" en el estado de "Ready for test", y aceptar o rechazar la validación de las mismas.

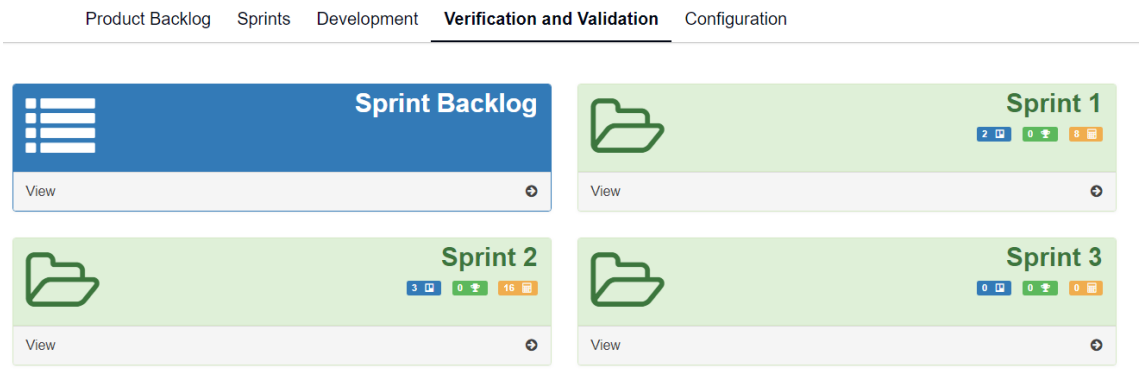

Ilustración 36 – Vista 4: Verificatión and Validation

El proceso de validación consiste en descubrir defectos funcionales, comprobando que la solución propuesta funciona correctamente en ambientes superiores. El objetivo consiste en determinar que cumple con las expectativas del cliente. Provocar fallas o revisar los productos son las dos maneras más comunes de descubrir defectos.

Luego de que el Pull Request fue fusionado, TG pasa la tarea al estado de "Ready for Test". Dentro del sistema este estado representa que la solución propuesta, desde el punto de vista de la programación, fue aceptada por el equipo pero aún está pendiente de las pruebas.

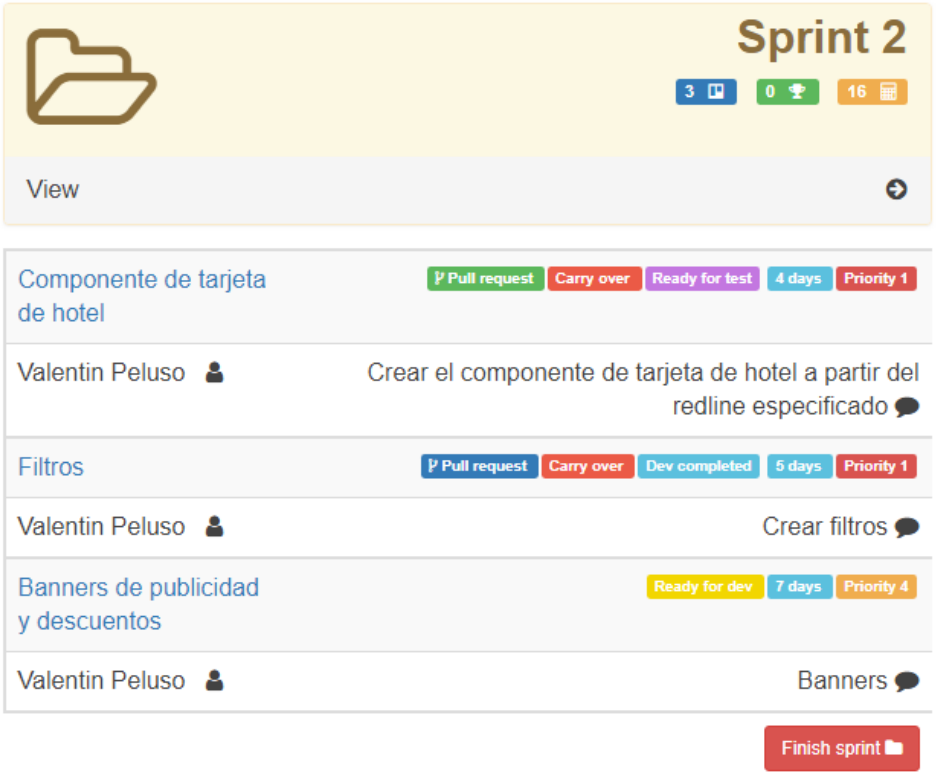

Ilustración 37 – Historia de usuario en estado de "Ready for Test"

Para realizar esta acción, el Master necesita acceder al detalle de la historia de usuario, donde la herramienta le habilita dos acciones para aceptar o rechazar la solución propuesta:

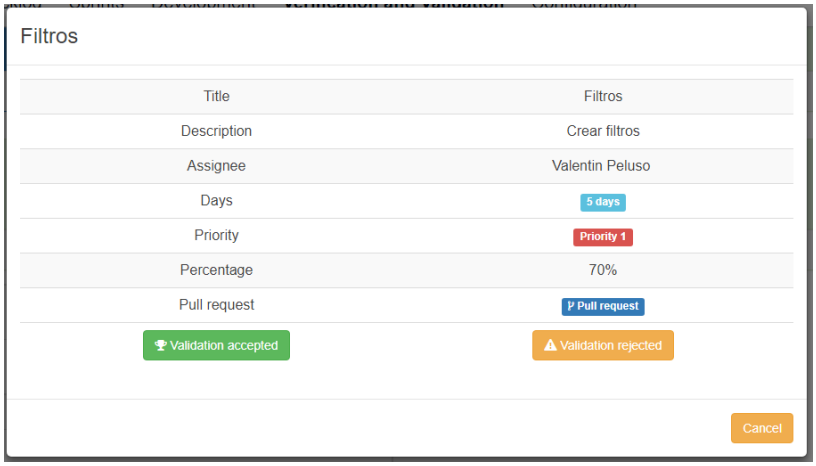

Ilustración 38 – Marcar/Rechazar la validación

5.4.5.1 VALIDACIÓN RECHAZADA

Que la validación haya sido rechazada significa que la solución propuesta no cumple con las expectativas del cliente o que no estaba acorde con alguna regla definida. Entonces cuando el Master realiza esta acción, la herramienta regresa la tarea al estado de "Ready for dev". Esto se debe a que es necesario desarrollar una nueva solución para reparar los problemas encontrados.

Ilustración 39 – Validación rechazada 1

Como se observa en la Ilustración 39, el color del label del Pull Request cambió. Esto se debe a que el usuario en algún momento tendrá que crear un nuevo Pull Request con el arreglo propuesto. Este label representa que la tarea tenía un Pull Request creado con anterioridad, pero la validación fue rechazada. De esta manera el usuario no pierde la relación entre ellos.

Luego de finalizar el nuevo desarrollo, el usuario necesita volver a crear el Pull Request en Github y luego vincular la tarea con el mismo.

Ilustración 40 – Validación rechazada 2

Como se observa en la Ilustración 40, en este punto la tarea se encuentra en el estado de "Dev Completed". Entonces para continuar con el proceso de desarrollo, entra en el proceso de verificación del código en GitHub.

5.4.5.2 VALIDACIÓN ACEPTADA

Una vez aceptada la validación, la herramienta asume que se ha generado un incremento dentro del sistema. Entonces TG actualiza el estado de la tarea a "Closed".

Cuando la historia de usuario pasa al estado de "Closed", se disparan dos actualizaciones: Por un lado, el requerimiento al que estaba asociado la tarea aumentará el porcentaje de finalización. Por otro, el "Sprint Actual" sumara 1 a la cantidad de historias de usuario completadas.

En este punto se considera finalizado el proceso de gestión del desarrollo de una tarea. Aun así, TG almacena todas las historias de usuario finalizadas. Esto se debe a que las mismas resultan beneficiosas al proceso de búsqueda de una historia de usuario en particular.

5.5 PROCESO DE BÚSQUEDA

Una historia de usuario relacionada puede resultar útil para el desarrollo de una tarea posterior. Esto se debe a que normalmente suelen compartir características, cómo reglas de negocio, implementaciones, problemas o archivos necesarios a modificar. Es necesario recordar que TG mantiene la vinculación de cada historia de usuario con el Pull Request propuesto para su desarrollo. De esta forma, encontrar una historia de usuario en particular, significa también hallar el pull Request que la implemento.

TG Project Manager dispone diversos medios para realizar la búsqueda de una historia de usuario en particular. La más sencilla y la principal característica a tener en cuenta durante el proceso de búsqueda, es analizar si la tarea a desarrollar, tiene definida una tarea relacionada mediante la propiedad Issue link. Esta propiedad se utiliza para marcar relaciones entre historias de usuario. Si esta propiedad se utiliza con cuidado, un desarrollador simplemente con navegar a través de las tareas relacionadas, podría encontrar la historia de usuario que está buscando.

Otra alternativa que dispone la herramienta, es analizar Sprints pasados. En muchas ocasiones, el equipo sabe en que Sprints se agregó un incremento, pero no es posible acordarse el título de la tarea como para buscarlo rápidamente a través de un buscador. TG almacena los Sprints finalizados, entonces un usuario puede acotar el total de historias de usuario visualizando únicamente ese Sprint en particular. En el peor de los casos, debería analizar el detalle de cada una de ellas, hasta encontrar la tarea deseada.

Otra forma que se puede utilizar, es analizar todas las historias de usuario creadas a partir del mismo requerimiento del cual fue creada la tarea actual. En TG todas las historias de usuario son creadas a partir de un requerimiento. La herramienta mantiene esta relación, y a su vez permite visualizar todas tareas agrupadas por este criterio, es posible acotar el total de historias de usuario. De esta forma, es posible encontrar una historia de usuario, visualizando el requerimiento padre de la tarea actual.

Otra característica que dispone la herramienta, es que mantiene todos los Pull Request asociados a una misma tarea. Cuando la validación de una historia de usuario es rechazada, la misma regresa al estado de "Ready for dev", ocasionando que el usuario necesite desarrollar una nueva solución, con el objetivo de reparar los problemas encontrados. Esto significa que en algún momento deberá crear un nuevo pull Request con el arreglo propuesto. TG mantiene esta relación mediante dos labels. Uno que representa al Pull Request actual, es decir el abierto. Y otro que representa todos los Pull Requests que ya fueron fusionados con el repositorio remoto, pero que fueron rechazados funcionalmente. De esta forma el usuario puede visualizar todos los Pull Request que se crearon para el desarrollo de una única tarea, encontrando rápidamente todos los archivos modificados.

Contar con más de un medio para encontrar una historia de usuario en particular o un Pull Request relacionado, agregan agilidad al proceso de búsqueda. A su vez, la simplicidad de

todos estos métodos, quitan la necesidad de tener que contar con un buscador y tener que recordar características de una historia en particular.

6. CONCLUSIÓN

A lo largo de la tesina, se realizó un análisis detallado, de la importancia de disponer una herramienta de apoyo, destinada a la gestión de proyectos. A su vez, se manifestó la eficacia que se consigue, cuando se prioriza la agilidad y flexibilidad en el desarrollo, en lugar de los procesos largos y poco estructurados. También se indicó lo imprescindible que se vuelve contar con una herramienta de apoyo, cuando el nivel de complejidad del proyecto es elevado, permitiendo administrar el desarrollo, controlar el personal y gestionar las actividades, durante todo el ciclo de vida del proyecto.

TG Project Management es una herramienta de apoyo, desarrollada con el principal objetivo de gestionar proyectos de alta complejidad. Es decir, proyectos formados por un gran número de personas, requerimientos, actividades y procesos. Las principales característica son:

1. **Configuración Inicial y Creación del proyecto**: Tener preparada la herramienta para gestionar el proyecto durante todo el ciclo de vida del mismo, definiendo estrategias tanto para los primeros pasos, como para cuando se dispone un alto nivel de complejidad, permitió evitar problemas referido a las configuraciones por defecto y al proceso de creación del proyecto.
2. **Metodología Hibrida**: Utilizar una metodología de gestión de proyectos ágil del tipo hibrida, utilizando las mejores prácticas de Scrum y Kanban, permitió conseguir numerosas ventajas al proceso de gestión del proyecto.
3. **Equipos y Roles**: Organizar al personal en equipos, utilizar roles para controlar a los usuarios, definir permisos asociadas a dichos roles, etc., facilito la administración de recursos humanos.
4. **Seguimiento completo del proyecto**: Disponer una herramienta para administrar todos los requerimientos desde la etapa inicial del sistema, permitió solucionar el problema referido a la falta de un proceso de elicitación. A su vez, se enumeraron las numerosas ventajas que dispone dentro del proceso de desarrollo y durante el seguimiento del proyecto.
5. **Control del desarrollo y vinculación con un Pull Request**. Utilizar el sprint para controlar las tareas, definir vistas accesibles por usuario, gestionar los equipos a través de módulos, administrar las tareas utilizando un esquema de prioridades, gestionar el avance de las historias de usuario a través de estados, definir diversas alternativas para realizar búsquedas de pull Request y tareas, etc., permitió controlar avance del proyecto.
6. **Búsqueda de Historias de Usuario desarrolladas**: Disponer un medio para vincular una historia de usuario con un Pull Request, permitió gestionar el proceso de desarrollo.
7. **Verificación y Validación:** Disponer un medio para marcar la verificación y validación de cada tarea, permitió completar el proceso de desarrollo, desde la etapa de licitación, hasta la etapa de mantenimiento.

Aunque los proyectos de software actuales aumenten exponencialmente en complejidad a medida que trascurre su ciclo de vida, TG está preparado para gestionar todos los procesos que se llevan a cabo, durante la gestión del proyecto. Permitió mejorar la eficiencia de muchas estrategias que utilizan Jira y Trello, durante todo el proceso, como así también, conseguir numerosas ventajas en diversos momentos del ciclo de vida del mismo. Pero su principal característica y distinción, fue permitir gestionar el proyecto de principio a fin, sin tener que preocuparse por el momento en el que se encuentra, la cantidad de personas que dispone y el tamaño que presenta. TG se adapta al proyecto, el proyecto no necesita adaptarse a la herramienta.

7. TRABAJOS FUTUROS

Como trabajo futuro, se propone ampliar la herramienta para brindar soporte a numerosas características de la gestión de proyectos, beneficiando a la agilidad de la herramienta, al seguimiento y avance del proyecto y al análisis de procesos.

- Definir gráficos estadísticos del progreso del proyecto, en base al desarrollo dentro de los Sprints, tareas, actividades, usuarios, etc., comparando una curva histórica con una curva actual.
- Definir estrategias para identificar, analizar y definir medios de respuesta a riesgos dentro del desarrollo. Esto mejoraría a la estimación de tiempos, mejorando la eficiencia en general del proyecto.
- Definir medios para analizar costos, pudiendo interpretar el avance del proyecto en función del precio que requiere.
- Definir estrategias para organizar al personal de forma hibrida, permitiendo a un desarrollador participar en diversas tareas.
- Vincular la herramienta con una herramienta de integración continua, como puede ser Jenkins. De esta forma se podrán realizar integraciones automáticas, para así poder detectar fallos antes de que la tarea se encuentre en un ambiente superior.
- Vincular la herramienta con una herramienta de servicio de correo electrónico, como por ejemplo Gmail. De esta forma se podrán avisar a los usuarios de los eventos que ocurren dentro de la herramienta.

BIBLIOGRAFÍA

Alexander Menzinsky, Gertrudis López, Juan Palacio. (2016). *Scrum Manager.* Iubaris Info 4 Media SL.

Almunia, P. (22 de Febrero de 2016). *itmplatform*. Recuperado el 25 de Junio de 2017, de itmplatform: http://www.itmplatform.com

Atlassian. (12 de Octubre de 2004). *Jira.* Recuperado el 2017 de Marzo de 5, de Jira: https://www.atlassian.com/software/jira

Barato, J. (2013). *Diferencias entre la gestión tradicional y la gestión ágil de Proyectos de Software.* España: ESI.

Barcelona, U. d. (2008). *Las 3 metodologías para la gestión de proyectos que más se utilizan.* Barcelona: OBS - Buisness School.

Collins-Sussman, Ben; Fitzpatrick, B.W. and Pilato. (2004). *Version Control with Subversion.* O'Reilly.

Ferran, G. (2017). *Desarrollar el plan de proyectos de Business Intelligence (Parte 1).*

Figuerola, N. (2015). Cómo seleccionar una Metodología de Project Management. *Project Management*, 7.

Figuerola, N. (2015). *Gestionando pequeños proyectos.*

Figuerola, N. (2016). Kanban - Su uso en el desarrollo de software. *Kanban*, 8.

Figuerola, N. (2016). *Ventajas del uso de un gestor de proyectos - La motivación.*

flow.ci. (17 de 9 de 2016). *GitHub vs. Bitbucket vs. GitLab vs. Coding*. Recuperado el 15 de 7 de 2017, de medium: https://medium.com

Garc, K. O. (2002). Manual para la presentación de proyectos comunitarios.

Garriga, A. (2006). *Director del proyecto. Funciones y responsaabilidades.*

GitHub. (2013). *Github Help*. Recuperado el 15 de 7 de 2017, de About forks: https://help.github.com/articles/about-forks/

GitHub. (2013). *GitHub help*. Recuperado el 15 de 7 de 2017, de About pull requests: https://help.github.com/articles/about-pull-requests/

Herranz, R. (2012). *Scrum Manager.*

Herrman, C. (2009). *Fundamentals of methodology.*

Highsmith, M. F. (2001). The Agile Manifesto. *The Agile Manifesto*, 7.

Kniberg, H. (2009). How to make the most of both . *Kanban vs Scrum*, 1-41.

Lledó, P. &. (2007). *Gestión de Proyectos - Cómo dirigir proyectos exitosos, coordinar los recursos humanos y administrar los riesgos.* California: Pearson. Prentice Hall.

Lorino, P. (1993). *Gestión del equipo del proyecto.*

Miranda, J. J. (2012). *Gestión de proyectos. Identificación - Formulación - Evaluación financiera - Económica - Social - Ambiental.* MM Editores .

Muhammad Ovais Ahmad, J. M. (2013). Kanban in software development. *Kanban in software development: A systematic literature review*.

Pablo Lledó y Gustavo Rivarola. (2007). *Gestión de proyectos / Pablo Lledó y Gustavo.* Buenos Aires: Prentice Hall - Pearson Education.

Pfleeger, S. L. (2002). *Ingeniería de software teoría y práctica.* Buenos Aires: Pearson Educación.

Pressman, P. R. (2003). *Ingeniería del software - Un enfoque práctico.* Mexico: Mc Graw Hill.

Schaull, S. F. (2011). EL "DESARROLLO DE SOFTWARE" COMO "INGENIERÍA DE SOFTWARE". *Ing. USBMed*, 4.

Schwaber, K. (1995). Scrum. *SCRUM Development Process* , 23.

Shingō, S. (1989). *A Study of the Toyota Production System from an Industrial Engineering Viewpoint. Productivity Press.*

sommerville, I. (2005). *Ingeniería de software. Septima edición.* Madrid (España): Pearson Educación, S.A.

Sommerville, I. (205). Ingeniería de software. Séptima edición. En I. Sommerville, *Ingeniería de software. Séptima edición.* Madrid: Pearson Addison Wesley.

Stella, F. (2010). *Trello*. Recuperado el 5 de Marzo de 2017, de Trello: https://trello.com/

Straub, S. C. (2014). *Pro Git 2nd Edition.* Apress.

Thompson, B. (2009). Lean Software Engineering. *Lean Software Engineering*, 10.

Tom Preston-Werner, C. W. (Abril de 2008). *Github*. Recuperado el 7 de Marzo de 2017, de Github: https://pages.github.com/

Turner, R., & Jain, A. (2002). Agile Meets CMMI: Culture Clash or Common Cause? En R. &. Turner, *Agile Meets CMMI: Culture Clash or Common Cause?* (págs. 153-165). XP/Agile Universe 2002.