UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Convivencia de metodologías: Scrum y Rup en un proyecto de gran escala

Autores: Juan Manuel Fernández, Sebastián Cadelli

Director: Dra. Claudia Pons

Carrera: Licenciatura en Sistemas

Resumen

Hoy en día existen diversas alternativas a la hora de elegir una metodología para desarrollar software. Años atrás imperaban las metodologías rígidas, también conocidas como las metodologías tradicionales, en las cuales abunda la documentación, los modelados, las actividades, actores y/o roles. A partir del 2001, surge un punto de inflexión, con el surgimiento de otro tipo de metodologías conocidas como las metodologías ágiles para el desarrollo de software. Estas fueron creadas para ser aplicadas en proyectos pequeños/medianos y que tienen requerimientos volátiles o que cambian con frecuencia. Mientras que las metodologías tradicionales, suelen ser más apropiadas para grandes proyectos y donde sus requerimientos son más resistentes a los cambios.

A partir de estos conceptos, queremos explicar en esta tesis, que si bien no existen recetas magistrales para aplicar metodologías a un determinado proyecto, es posible detectar buenas prácticas y otros patrones que hacen posible decidir acerca de los métodos a utilizar. Es decir, que a partir de nuestra experiencia en un proyecto vamos a explicar cómo pueden convivir dos tipos de metodologías (Scrum y Rup). Esto es, tomando lo que se sea necesario de cada metodología y adaptándola al proyecto en cuestión.

Palabras Claves

-Proyecto
-Software
-Convivencia
-Metodologías de desarrollo
-Metodologías tradicionales
-Metodologías ágiles
-Scrum
-Rup
-Scrum de Scrum

Trabajos Realizados

- Análisis y comparación de las metodologías Scrum y Rup.

- A partir de nuestra experiencia laboral en el proyecto eSidif, explicamos la convivencia de ambas metodologías descriptas anteriormente.

- Adaptación de ambas metodologías al proyecto en cuestión.

Conclusiones

No existen recetas para aplicar metodologías de desarrollo de software a los proyectos. Cada metodología posee sus principios, y por ende sus ventajas y/o desventajas. Por eso, es importante tener en cuenta que aspectos pueden necesitarse de una metodología y de esta forma aplicarla a un proyecto tomando las cosas que sean necesarias. Por otra parte, es importante también resaltar que dos metodologías distintas y de distintos principios pueden convivir, como es el caso del proyecto que presentamos.

Trabajos Futuros

Planteamos como posible trabajo futuro un análisis y desarrollo de proyectos que utilicen las nuevas alternativas de gestión ágiles como por ejemplo Kanban y su fusión con Scrum denominada "ScrumBan". Definiendo ventajas y desventajas, detallando qué aspectos de Scrum son comunes con Kanban y que otros aspectos particulares agrega Kanban a la gestión.

Realizar una comparación detallada de Kanban con las diferentes alternativas de gestión ágiles más utilizadas en la actualidad, como pueden ser, Scrum, XP, Lean; entre otras.

Fecha de la presentación: Junio 2014

Convivencia de metodologías: Scrum y Rup en un proyecto de gran escala

TESINA DE GRADO DE LICENCIATURA EN SISTEMAS

Alumnos

Juan Manuel Fernández

Sebastián Cadelli

Director

Dra. Claudia Pons

2014

Facultad de Informática
Universidad Nacional de La Plata

Agradecimientos

Queremos agradecer a Claudia nuestra directora de tesis, que nos guió, indicó y sugirió lo adecuado para finalizar este trabajo. Agradecer a todos aquellas personas que nos ayudaron y de una u otra manera nos brindaron su apoyo para llegar hasta acá.

Agradecimientos de Juan

A mi compañero de tesis y amigo, que sin su ayuda no hubiésemos podido culminar este trabajo, gracias Sebita. A mi mamá, mi papá, a mis abuelos, a Silvia, a Edu, y a todos mis familiares gracias eternas por estar siempre, ayudarme a poder realizar esta carrera y apoyarme en todos los momentos durante este camino. A mi novia que también me apoyo y bancó en las buenas y en las malas. A todos ellos les agradezco de corazón porque sin ellos hubiera sido muy difícil culminar esta etapa.

Gracias a todos.

__ Juan Manuel

Agradecimientos de Sebastián

A Juancito, por las pilas que le metió para terminar esta tesis, gracias amigo. A mi familia, mis viejos, mi abuela, por acompañarme tantos años. A mi hna por empujarme, sino todavía no hubiese definido qué carrera seguir. A Lu mi compañera y amiga de estudio durante todo el camino. A todos los que bancaron, gracias de corazón.

__Sebastián

Dedicatorias

Dedicatoria de Juan Manuel

__Juan Manuel

Dedicatoria de Sebastián

__Sebastián

INDICE GENERAL

CAPÍTULO 1. Objetivos, Estructura y Motivación

1.1 Objetivos

"A través de esta tesis se realizará una documentación formal de la adaptación y convivencia de las metodologías identificando las ventajas y/o desventajas de cada una, como así también la ventaja de la utilización de "scrum de scrum" en proyectos de gran tamaño".

Este trabajo formaliza y enriquece a la metodología utilizada en nuestro proyecto, ya que, si bien esta metodología está siendo utilizada de manera eficiente y exitosa, nuestra tesis se encarga de:

- Analizar sus dos visiones complementarias. Es decir, desde la visión tradicional (como es el caso de Rup), y la visión ágil (como es el caso de Scrum y/o Scrum de Scrum);
- Describir detalladamente cada una de sus componentes, interacciones y comunicación;
- Analizar sus ventajas y desventajas;
- Plantear mejoras y trabajos futuros;
- Documentarla de manera formal, fundamentándola en base a la teoría de Rup y Scrum.

A partir de nuestra participación y experiencia en un proyecto de gran escala, tenemos como objetivo explicar la convivencia de dos tipos de metodologías que tienen distinto enfoque pero que pueden convivir dentro de un proyecto.

Inicialmente el proyecto se regía bajo los principios de la metodología RUP, con el advenimiento del nuevo concepto ágil en el desarrollo de software comenzamos a analizar los beneficios de la inclusión de dicho concepto como parte del proceso. Aquí nos surgió la inquietud de poder consumir las características más beneficiosas de ambas metodologías y adaptarlas en un nuevo proceso de desarrollo donde convivan dichas metodologías con enfoques dispares. Este proceso de fusionar ambas metodologías que parecen opuestas, es lo que se va a explicar en el desarrollo de este documento.

Como consecuencia de la evolución y el crecimiento de nuestro proyecto, la cantidad de recursos fue aumentando paulatinamente dificultando la organización del mismo. Para disminuir esta dificultad decidimos implementar la alternativa "Scrum de Scrum", la cual vamos a explicar de forma detallada, indicando su estructura, organización y comunicación adaptadas a las necesidades de nuestro proyecto.

1.2 Estructura de la tesis

La tesis se encuentra organizada en 8 capítulos, los cuales se describen brevemente a continuación:

- **Capítulo 1 Objetivos, Estructura y Motivación**: el presente capítulo se basa en explicar cuáles son los objetivos de la tesis, así como también de mencionar la forma en la cual se encuentra estructurada la misma. Además se presenta la motivación, es decir, un panorama del contexto actual que da una introducción al origen de nuestro trabajo de tesis.
- **Capítulo 2 Metodologías de desarrollo**: este capítulo realiza una introducción a las metodologías utilizadas en el desarrollo de SW. Explicando sus características, brindando los distintos tipos de clasificaciones que existen y explicando cada tipo de taxonomía.

  Además se encarga de abarcar a las metodologías tradicionales (conocidas como rígidas y/o pesadas) a través de una descripción de las mismas.

  También se realiza una introducción a las metodologías ágiles. Desde cómo surgió la necesidad de desarrollar un nuevo tipo de metodología, el nacimiento del concepto ágil, hasta los principios que se incluyen en el manifiesto ágil.

  Por último, se realiza una comparación entre las metodologías tradicionales y las metodologías ágiles donde se contrasta a ambos tipos de metodologías a través de una comparación de los diferentes aspectos y enfoques que presentan.
- **Capítulo 3 Scrum y Rup**: por un lado se explica la metodología ágil denominada Scrum. El capítulo se encarga de describir varios aspectos de esta metodología tales como: su flujo de trabajo, los elementos que lo componen, los roles de las personas involucradas, las estimaciones, etc.

  Por otra parte, introduce a la metodología pesada Rup, describiendo las distintas etapas que tiene en el desarrollo de SW (inicio, elaboración, construcción, transición). También se explican los principios de dicha metodología.

  Por último se realiza una comparación de los dos tipos de metodologías descriptas anteriormente.
- **Capítulo 4 Nuestro Proyecto de referencia: eSidif**: introduce al proyecto a partir del cual se va a explicar la convivencia de los dos tipos de metodologías. Se detallan las características principales del mismo, la arquitectura y lenguajes utilizados, la conformación de los grupos de trabajo, las herramientas que se utilizan, etc.
- **Capítulo 5 Nuestra propuesta de Proceso de desarrollo combinado**: en este capítulo se va a explicar la convivencia de Scrum y Rup en el proyecto Esidif. Es decir, se va a detallar el funcionamiento de cada metodología adaptada al proyecto en cuestión.

  También se va a introducir el concepto de "Scrum de Scrum" y cómo este nuevo enfoque es aplicado a nuestro proyecto.

  Todo esto nos permite puntualizar a este como uno de los capítulos más interesantes de la tesis.
- **Capítulo 6 Trabajos Relacionados**: contiene trabajos relacionados con el tema de esta tesis.
- **Capítulo 7 Conclusiones**: se detallan las conclusiones que derivan del trabajo de tesis finalizado, como así también los posibles *trabajos futuros*.

1.3 Motivación

Hasta no hace mucho tiempo, el desarrollo de software tenía una tendencia marcada en el control de los procesos a través de metodologías rígidas, dirigidas por excesiva documentación, modelados, actividades, actores y/o roles. A partir del año 2001, tras una reunión realizada en los EE.UU, surgió el concepto de ágil. En dicha reunión, (de la cual formaron parte varios expertos de la industria del software) se comenzó a proyectar una alternativa a las metodologías tradicionales de software.

El primer avance significativo fue la creación de una organización sin fines de lucro, dedicada a promover conceptos relacionados con el desarrollo ágil, denominada "*The Agil Alliance*", la cual creó un documento en donde se sientan las bases, o la "filosofía" que persigue este tipo de metodologías alternativas, es decir, las metodologías ágiles. A contraposición de las tradicionales, las metodologías ágiles se centran en el equipo de desarrollo y sus interacciones, y en la implementación de software por sobre la documentación del mismo.

A partir de aquí, surge entonces el debate: ¿bajo qué condiciones es más adecuado utilizar un tipo de metodología u otro? Es decir, en qué tipo de proyecto vamos a obtener resultados más eficientes utilizando una metodología tradicional y en cual va a funcionar de mejor manera una metodología ágil.

Es cierto que las metodologías ágiles fueron creadas originalmente para proyectos pequeños/medianos y que tienen requerimientos volátiles y/o cambiantes; y que las metodologías tradicionales suelen ser más eficientes en proyectos más grandes y donde su entorno y requisitos son resistentes al cambio.

Pero no existe una "receta" magistral que nos indique específicamente cuando se debe utilizar una u otra metodología. Es más, en un mismo proyecto podemos utilizar una combinación de ambas, aprovechando las diferentes ventajas de cada una. Es decir, no son excluyentes.

He aquí el motivo de nuestra tesis, de cómo se pueden aplicar y convivir dos metodologías en un proyecto.

CAPÍTULO 2: Metodologías de desarrollo

2.1 Metodologías de desarrollo de Software

2.1.1 Introducción

Un proceso de software detallado y completo suele denominarse "Metodología". Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, espiral entre otros). Adicionalmente una metodología debe definir con precisión los artefactos, los roles y las actividades involucradas, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. Habitualmente se utiliza el término "método" para referirse a técnicas, notaciones y guías asociadas, que son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño.

La comparación y/o clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas. A grandes rasgos, considerando su filosofía de desarrollo, aquellas metodologías con mayor énfasis en la planificación y control del proyecto, en la especificación precisa de requisitos y modelado, reciben el nombre de Metodologías Tradicionales (o también denominadas Metodologías Pesadas, o Peso Pesado).

Otras metodologías, denominadas Metodologías Ágiles, están más orientadas a la generación de código con ciclos muy cortos de desarrollo, y se dirigen a equipos de desarrollo pequeños. Estas hacen especial hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso.

Una metodología de desarrollo de software se refiere al entorno que se usa para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información. Una gran variedad de metodologías se han desarrollado a lo largo de los años, cada una de ellas con sus fortalezas y debilidades, con sus ventajas y desventajas.

2.1.2 Características deseables de una metodología

A continuación se mencionan distintas características que se requieren deseables a la hora de elegir una metodología.

- Existencia de reglas predefinidas, fases y sub-fases, tareas, productos intermedios, técnicas y herramientas tales que se amolden a cualquier desarrollo.
- Cobertura total del ciclo de desarrollo.
- Verificaciones intermedias.
- Planificación y control.
- Comunicación efectiva.
- Utilización sobre un abanico amplio de proyectos.

- Fácil formación.
- Herramientas CASE[i].
- La metodología debe contener actividades que mejoren el proceso de desarrollo.
- Soporte al mantenimiento. Por ejemplo, la reingeniería.
- Soporte de la reutilización de software, no solo reutilización de código.
- Actualmente, se huye de métodos muy burocráticos o monolíticos.

2.1.3 Metodologías secuenciales / Iterativas

2.1.3.1 Metodologías secuenciales

En las metodologías secuenciales, el proceso de desarrollo de software se divide en varia fases (o pasos). Cada fase tiene un conjunto de metas a cumplir. El fin de cada fase delimita el comienzo de la fase siguiente. Aunque es normal la superposición de fases, estas metodologías proponen una gran fase de análisis de requerimientos, una fase de diseño, una fase de construcción y por último una de pruebas. El alcance de cada fase es la totalidad de los requerimientos de un proyecto.

2.1.3.2 Metodologías Iterativas

En las metodologías iterativas se divide el proyecto en entregas (o iteraciones). Si cada iteración define un conjunto de metas a cumplir, podríamos pensar que no hay una gran diferencia con la metodología secuencial. No obstante, cada iteración define como entregable un software testeable por el usuario. Entonces hay etapas de análisis de requerimientos, diseño, construcción y prueba en cada iteración. Además, cada iteración permite revisar y cambiar los requerimientos a resolverse.

2.1.4 Metodologías orientadas: al proceso/ a las personas

Otra taxonomía que divide las metodologías de desarrollo de SW, es el grado de importancia que le dan a dos aspectos:

- al proceso de desarrollo
- y a las personas que ejecutan ese plan.

2.1.4.1 Metodologías orientadas al proceso

Si bien la mayoría de las metodologías contemplan tanto la serie de pasos que conforman el proceso como qué tipo de tareas deben desarrollar las personas (en base a sus perfiles), hay metodologías en las que el proceso está por encima de las personas. Dicho de otra manera, "respetar el proceso garantiza el éxito del proyecto". El margen de discrecionalidad (cuánto puedo salirme del libreto) es mínimo, sólo en lo operacional (en el día

a día). Por eso es importante para estas metodologías poder medir cada tarea del proyecto, sea un ejecutable o documentación.

2.1.4.2 Metodologías orientadas a las personas

Las metodologías orientadas a las personas consideran que éstas definen el éxito o fracaso de un proyecto. Les asignan un grado mayor de decisión en cada tarea y confían en su capacidad de resolución de un problema antes que en las métricas que dan los indicadores. Esto no quiere decir que el proceso no importe, sino que ocupa un puesto de menor relevancia en la alcance de un logro, de una meta.

2.1.5 Metodologías orientadas: a la documentación/ al producto

2.1.5.1 Metodologías orientadas a la documentación

Hay metodologías que sostienen que un producto de software bien elaborado nace de una documentación extensa y que contempla todas las decisiones que surgieron del análisis y del diseño. De esa manera el desarrollador no tendrá dudas ni excusas a la hora de escribir cada línea de código. Estas metodologías son las *metodologías orientadas a la documentación.*

2.1.5.2 Metodologías orientadas al producto

Por el contrario, hay metodologías que privilegian tener un software testeable para el usuario antes que tener el documento que respalde ese software que está corriendo. Esto no significa que haya que programar sin tener una especificación, sino que:

- tomamos una documentación que puede tener definiciones pendientes, estar incompleta en su diseño o que sepamos que esté sujeta a cambios,
- construimos el software,
- y luego actualizamos las decisiones principales en el documento con la ventaja de tener la certeza de que lo qué hace el sistema es eso.

Estas son las *metodologías orientadas al producto.*

2.1.6 Metodologías adaptativas/ predictivas

¿Qué ocurre con los cambios que piden los usuarios mientras se va construyendo el software? ¿Cómo se manejan?

Dependiendo de las respuestas a las dos preguntas anteriormente planteadas, las metodologías pueden ser:

- adaptativas
- predictivas

2.1.6.1 Metodologías adaptativas

Consideran que el cambio es inevitable, que no tiene sentido resistirse a él. De manera que el proceso mismo contempla momentos en los que el usuario puede modificar los requerimientos: esto implica agregar nuevos requerimientos, descartar otros y/o modificarlos (no importa si ya fueron construidos o no). "El usuario tiene derecho a cambiar de opinión", es lo que se sostiene en este tipo de metodologías.

2.1.6.2 Metodologías predictivas

Las metodologías predictivas sostienen que es posible anticipar los cambios a través de un buen análisis y un buen diseño que contemple diferentes alternativas. Este enfoque no es inocente, sabe perfectamente que el usuario puede cambiar de opinión, que las disposiciones legales e impositivas sufren modificaciones y que los proyectos están siempre sujetos a vaivenes políticos. Pero justamente por eso busca minimizar los cambios para conservar lo más estable posible el entorno. Resistirse al cambio es su naturaleza.

2.2 Metodologías tradicionales

2.2.1 Introducción/Características

Las metodologías tradicionales, o también denominadas metodologías pesadas o de peso, son las metodologías orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.

Son las más tradicionales, ya que, se centran en:

- la definición detallada de los procesos y tareas a realizar,
- herramientas a utilizar,
- y requieren una extensa documentación, ya que, pretenden prever todo de antemano.

Este tipo de metodologías son más eficaces y necesarias utilizar cuanto mayor es el proyecto en cuanto al tiempo y a recursos que son necesarios emplear. Es decir, donde se requiere una gran organización.

Inicialmente el desarrollo de software era artesanal en su totalidad. Por la fuerte necesidad de mejorar el proceso y llevar a los proyectos a la meta deseada, tuvieron que importarse la concepción y los fundamentos de metodologías existentes en otras áreas y adaptarlas al desarrollo de software. Esta nueva etapa de adaptación contenía el desarrollo dividido en etapas de manera secuencial que de algo mejoraba la necesidad latente en el campo del software.

Entre las principales metodologías tradicionales tenemos a dos conocidos como lo son: RUP y MSF[ii] entre otros. Estos centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto (definido en la fase inicial del desarrollo del proyecto). Otra de las características importantes dentro de este enfoque son los altos costos al implementar un cambio y al no ofrecer una buena solución para proyectos donde el entorno es volátil.

Las metodologías tradicionales (formales) se focalizan en documentación, planificación y procesos (Plantillas, técnicas de administración, revisiones, etc.).

2.3 Metodologías ágiles

2.3.1 Introducción/Características

Durante el transcurso de los años 90 el ambiente cambiante y turbulento era cada vez más la regla que la excepción. Por lo tanto surgió la necesidad de desarrollar metodologías livianas y maniobrables que pudieran operar en ese ambiente. Estas metodologías son conocidas colectivamente hoy en día como "metodologías ágiles". En la figura 1 se puede observar una línea cronológica de cómo fueron evolucionando y surgiendo los distintos tipos de metodologías de desarrollo de software, desde las metodologías orientadas a los procesos, hasta las agiles como por ejemplo: DSDM, SCRUM, FDD, etc.

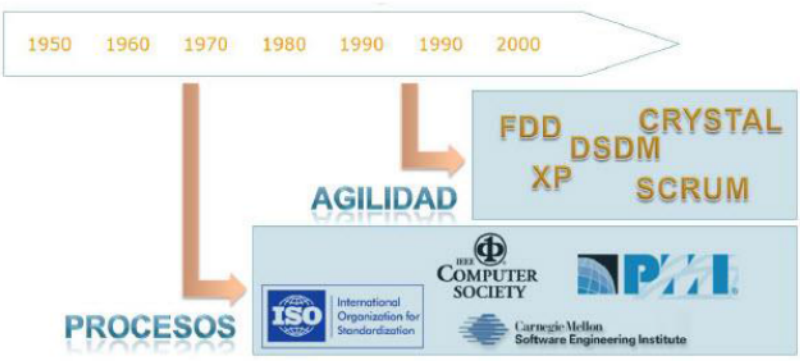


Figura 1 – Evolución de las metodologías de desarrollo de SW

¿Qué es una metodología ágil?

Lo ágil se define (por los mismos agilistas) como la habilidad de responder de forma versátil, al cambio para maximizar los beneficios. Las metodologías ágiles varían en su forma de responder al cambio, pero en general comparten las siguientes características:

- Los individuos y sus interacciones son más importantes que los procesos y las herramientas.
- El software que funciona correctamente es más importante que la documentación exhaustiva.
- La colaboración con el cliente en lugar de la negociación de contratos.
- La respuesta al cambio en lugar de aferrarse a un plan.

En una reunión celebrada en febrero de 2001 en Utha (Estados Unidos), nace el término "ágil" aplicado al desarrollo de software. En esta reunión participaron un grupo de 17 expertos de la industria del software. Su objetivo fue esbozar los valores y principios que debían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que pueden surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. Tras esa reunión se creó "*The Alliance*"; una organización dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que los adopten. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía ágil.

Los valores anteriores inspiran los *doce principios del manifiesto.* Éstas son las características que diferencian un proceso ágil de uno tradicional. Los dos primeros son generales y resumen gran parte del espíritu ágil.

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten un valor. Un proceso es ágil si a las pocas semanas de empezar, ya se entrega software que funcione aunque sea rudimentario. El cliente decide si pone en marcha dicho software con la funcionalidad que ahora se proporciona o simplemente lo revisa e informa de posibles cambios a realizar.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva. Los cambios en los registros deben verse como algo positivo. Estos cambios le van a permitir al cliente aprender más, y lograr una mayor satisfacción.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas. Se insiste en que las entregas al cliente sean software y no planificaciones, ni documentación de análisis o de diseño.
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto. El proceso de desarrollo necesita ser guiado por el cliente, por lo que la interacción con el equipo es muy frecuente.

5. Construir el proyecto entorno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo. Los miembros de equipo deben hablar entre ellos. Éste es el principal y mejor modo de comunicación.

7. El software que funciona es la medida principal de progreso. El estado de un proyecto no viene dado por la documentación generada o la fase en la que se encuentre, sino por el código generado y el funcionamiento.

8. Los procesos ágiles promueven un desarrollo sostenible. Los desarrolladores y usuarios deberían ser capaces de mantener el ritmo de desarrollo durante toda la ejecución del proyecto, asegurando en todo momento que la calidad es máxima.

9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad. Producir código claro y robusto es la clave para avanzar más rápidamente en el proyecto.

10. La simplicidad es esencial. Tomar los caminos más simples que sean consistentes con los objetivos perseguidos. Si el código producido es simple y de alta calidad será más sencillo adaptarlo a los cambios que puedan surgir.

11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos. Todo el equipo es informado de las responsabilidades y éstas recaen sobre todos sus miembros. Es el propio equipo el que decide la mejor forma de organizarse, de acuerdo a los objetivos que se persigan.

12. En intervalos regulares, el equipo reflexiona respecto en cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. Puesto que el entorno está cambiando continuamente, el equipo también debe ajustarse al nuevo escenario de forma continua. Puede cambiar su organización, sus reglas, sus convenciones, sus relaciones, etc., para seguir siendo ágil.

2.3.4 Comparación de Metodologías Tradicionales Vs. Metodologías Ágiles

Como se explicó en el capítulo 2, existen distintos tipos de metodologías de desarrollo de software y se pueden clasificar dependiendo diversos aspectos. Pero como se aclaró, una de las clasificaciones más relevantes es la que se refiere a la *filosofía de desarrollo* que persiguen las mismas.

Encontramos entonces a estos dos tipos de metodologías:

- Metodologías de desarrollo Tradicionales (pesados, o de peso)
- Metodologías de desarrollo ágiles (livianos o ligeros)

En este capítulo se va a realizar una comparación de estas metodologías de desarrollo del SW.

Por un lado, las metodologías tradicionales para el desarrollo de software se caracterizan por tener un proceso disciplinado y sistemático con el objetivo de hacer el trabajo más predecible, eficiente y planificado.

RUP es un claro ejemplo de lo que se denomina una metodología pesada o tradicional, ya que, es un proceso riguroso de seguimiento de pasos metódicos y propulsor de una documentación extensa. Además no permite entregar un producto útil al cliente en etapas tempranas de su proceso de desarrollo y tampoco permite realizar cambios en los requerimientos. Estas son algunas características que se podría decir son comunes en las metodologías pesadas.

Por otro lado, las metodologías ágiles se sustentan en los siguientes puntos:

- Los individuos y sus iteraciones son más importantes que los procesos y herramientas.
- Un software que funcione es más importante que su documentación.
- La colaboración con los clientes es más importante que la negociación de los contratos.
- La respuesta ante cambios es más importante que el seguimiento de un plan.

Estos puntos están relacionados con los postulados de la alianza ágil, los cuales siguen los 12 principios del manifiesto ágil (que se mencionaron en la sección 2.3.1).

Los procesos de desarrollo ágil, generalmente son procesos iterativos en los que se entrelazan la especificación, el diseño, el desarrollo y las pruebas. El software se va desarrollando a través de incrementos, en donde cada incremento incluye nuevas funcionalidades al sistema.

Como se dijo que RUP es una de las metodologías pesadas, tenemos que decir que entre las metodologías ágiles tenemos como ejemplos más conocidos a SCRUM, XP[iii], Crystal Methodologies[iv], DSDM[v], ASD[vi], FDD[vii], LD[viii], entre otras.

La diferencia principal entre los métodos pesados y ligeros es la siguiente: mientras los métodos llamados pesados proponen un desarrollo a través del orden y la documentación extensa (lo cual garantiza la calidad del producto de software en un tiempo y costo determinado) las metodologías ligeras tratan de mejorar la calidad del software a través de la comunicación directa con el cliente, de entregas tempranas de avances al mismo y de solo la documentación necesaria.

La figura 2 y la figura 3 realizan comparaciones de las metodologías de software tradicionales y las metodologías de software ágiles.

	Métodos Ágiles	Métodos Tradicionales
Enfoque	Adaptación	Predictivo
Éxito de Medición	Valor del Negocio	Conformación de planificar
Tamaño del proyecto	pequeño	grande
Estilo de gestión	Descentralizada	Autocrático
Perspectiva para el Cambio	Cambio y Adaptabilidad	Cambio y Sostenibilidad
Cultura	Liderazgo-Colaboración	Comandos de control
Documentación	Bajo	Pesado
Énfasis	Orientada a las personas	Orientado a los procesos
Ciclos	Muchos	Limitado
Dominio	Impredecible exploratorio	Previsible
Planificación por adelantado	Mínimo	Exhaustivo
Retorno de la Inversión	A principios de Proyecto	Fin de Proyecto
Tamaño del equipo	Pequeños / Creatividad	Grande

Figura 2 – Comparación de metodologías ágiles y metodologías tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 3 - Comparación de metodologías ágiles y metodologías tradicionales

CAPÍTULO *3. Scrum y Rup*

3.1 Scrum

Scrum es una metodología ágil de gestión de proyectos cuyo objetivo primordial es elevar al máximo la productividad de un equipo. Pone su atención y hace foco sobre valores y prácticas de gestión, en vez de requerimientos, prácticas de desarrollo, implementación y otras cuestiones técnicas. Ésta metodología delega completamente en el equipo la responsabilidad de decidir la mejor manera de trabajar para ser lo más productivos posibles, es decir, que es flexible y los integrantes del equipo pueden optar por organizar la forma de interactuar entre ellos.

La terminología "Scrum" procede del deporte llamado rugby, donde se designa al acto de preparar el avance del equipo en unidad pasando la pelota a uno y otro jugador. Scrum es adaptable, ágil, auto-organizado y con pocos tiempos muertos. Esta metodología ágil fue desarrollada por Jeff Sutherland[ix] y elaborada más formalmente por Ken Schwaber[x]. Poco tiempo después Sutherland y Schwaber se unieron para refinar y extender Scrum. Se la ha llegado a conocer como una herramienta de hiper-productividad. Schwaber se dio cuenta entonces de que un proceso necesita aceptar el cambio, en lugar de esperar predictibilidad. Se enfoca en el hecho de que procesos definidos y repetibles sólo funcionan para atacar problemas definidos y repetibles con gente definida y repetible en ambientes definidos y repetibles. Toma el cambio como una forma de entregar al final del desarrollo algo más cercano a la verdadera necesidad del Cliente. Puede ser aplicado teóricamente a cualquier contexto en donde un grupo de gente necesita trabajar junta para lograr una meta común.

Se basa en los principios ágiles:

- Privilegiar el valor de la gente sobre el valor de los procesos.
- Entregar software funcional lo más pronto posible.
- Predisposición y respuesta al cambio.
- Fortalecer la comunicación y la colaboración.
- Comunicación verbal directa entre los implicados en el proyecto.
- Simplicidad; eliminación de artefactos innecesarios en la gestión del proyecto.

3.1.1 La esencia de Scrum

- Más que una metodología de desarrollo es una herramienta para gestionar proyectos.
- No contiene definiciones en áreas de ingeniería.
- Con visión de que el trabajo es efectuado por equipos auto-organizados y auto-dirigidos, logrando motivación, responsabilidad y compromiso.
- Está basada en un proceso constructivo iterativo e incremental donde las iteraciones tienen duración fija.

- Contiene definición de roles, prácticas y productos de trabajo escritas de forma simple.
- Está basada en un conjunto de valores y principios.

3.1.2 Flujo SCRUM

En la figura 4 se puede visualizar el flujo de trabajo de la metodología Scrum. Es decir, desde el product backlog, la duración del sprint, las iteraciones, etc.

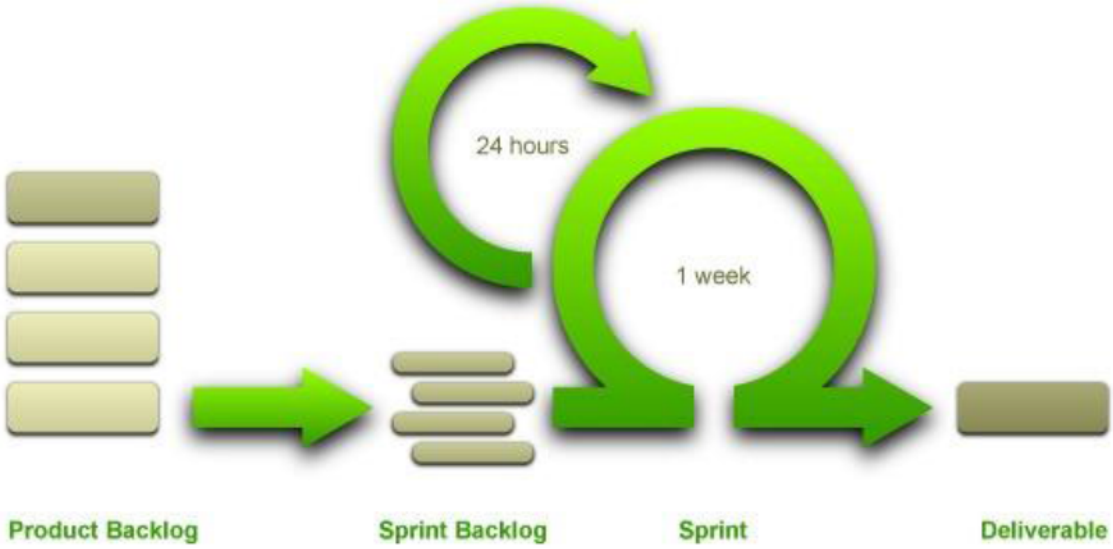


Figura 4 – Flujo de trabajo de Scrum

3.1.3 Elementos de SCRUM

Los elementos que componen Scrum son diversos. A continuación se mencionan y posteriormente se realiza una explicación de los mismos.

- Roles
  - Product Owner
  - Scrum Master
  - Team (Equipo)

- Poda de requerimientos
- Product Backlog
- Sprint
  - Planificación (Planning)
  - Sprint Backlog
  - Scrum
  - Estimaciones

  - Builds continuos
  - Revisión del Sprint
  - Reunión retrospectiva
- Valores
  - Foco, comunicación, respeto y coraje.

3.1.4 Roles

La dimensión del equipo total de Scrum no debería ser superior a veinte personas. Si hay más, lo más recomendable es formar varios equipos. No hay una técnica oficial para coordinar equipos múltiples, pero se han documentado experiencias de hasta 800 miembros, divididos en Scrum de Scrum definiendo un equipo central que se encarga de la coordinación, las pruebas cruzadas y la rotación de los miembros. Scrum tiene una estructura muy simple. Todas las responsabilidades del proyecto se reparten en 3 roles:

- **Product owner (Dueño del producto)➔** Representa a todos los interesados en el producto final. Es el responsable oficial del proyecto, gestión, control y visibilidad de la lista de acumulación o lista de retraso del producto (Product Backlog). Toma las decisiones finales de las tareas asignadas al registro y convierte sus elementos en rasgos a desarrollar.

Sus áreas de responsabilidad son:

- Financiación del proyecto.
- Requisitos del sistema.
- Retorno de la inversión del proyecto.
- Lanzamiento del proyecto.
-

- **Scrum Master (Líder del proyecto)➔** Responsable del proceso Scrum, de cumplir la meta y resolver los problemas. Así como también, de asegurarse que el proyecto se lleve a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que progrese según lo previsto. Interactúa con el cliente y el equipo. Coordina los encuentros diarios, y se encarga de eliminar eventuales obstáculos. Debe ser miembro del equipo y trabajar a la par.

- **Team (Equipo)➔** Responsable de transformar el Backlog de la iteración en un incremento de la funcionalidad del software, es decir, de convertir el product backlog en un software entregable. El equipo tiene la autoridad para reorganizarse y definir las acciones necesarias o sugerir eliminación de impedimentos.

Características del equipo:

- Auto-gestionado
- Auto-organizado
- Multi-funcional

El número ideal para la conformación de un equipo es entre 8 y 10 personas.

3.1.5 Poda de requerimientos

La primer actividad es armar una lista exhaustiva de los requerimientos originales del sistema. Luego se procede a ver qué requerimientos son realmente necesarios, cuáles pueden posponerse y cuáles eliminarse. Para ello debe identificarse un representante con capacidad de decisión, priorizar los requerimientos en base a su importancia y acordar cuáles son los prioritarios para la fecha de entrega. La poda de requerimientos es una buena práctica implícita en modelos ágiles, se hace lo que el cliente realmente desea, no más. Es decir, se realiza una priorización de requerimientos.

3.1.6 Product Backlog

Con los requerimientos priorizados y podados armamos el Backlog de Producto. Este es una forma de registrar y organizar el trabajo pendiente para el producto (actividades y requerimientos). Es un documento dinámico que incorpora constantemente las necesidades del sistema. Por lo tanto, nunca llega a ser una lista completa y definitiva. Se mantiene durante todo el ciclo de vida (hasta finalizar el sistema) y es responsabilidad del Product Owner.

3.1.7 Sprint Scrum

Está basado en el control empírico de procesos. Se utiliza cuando la capacidad de predicción es vaga, la incertidumbre alta o el proceso es demasiado complejo para ser modelado y definido. En el enfoque empírico de control de procesos se establecen reglas simples y se crea una disciplina de inspección frecuente para adaptarse rápidamente a situaciones imprevistas o problemas.

Un Sprint es el período de tiempo durante el que se desarrolla un incremento de funcionalidad. Constituye el núcleo de Scrum, que divide de esta forma el desarrollo de un proyecto en un conjunto de pequeñas "carreras".

Características del Sprint:

- Duración máxima de 30 días.
- Durante el Sprint no se puede modificar el trabajo que se ha acordado en el Backlog.
- Sólo es posible cambiar el curso de un Sprint, abortándolo, y sólo lo puede hacer el Scrum Master si decide que no es viable por alguna de las razones siguientes:
  - La tecnología acordada no funciona.
  - Las circunstancias del negocio han cambiado.
  - El equipo ha tenido interferencias.

3.1.8 Planificación

Se planifica en detalle el trabajo al inicio de cada Sprint asumiendo que los objetivos no van a cambiar durante el mismo. De esta manera se atenúa el riesgo.

Aspectos a tener en cuenta sobre la planificación de un Sprint:

- Una determinación general de alcance, frecuentemente basada en una EDT (Estructura de División del Trabajo).
- Estimaciones de esfuerzo de alto nivel realizadas durante la etapa de concepción del proyecto.
- Esfuerzo dedicado a labores de soporte o de preparación de los ambientes requeridos por el proyecto.
- Esfuerzo asociados a las reuniones diarias, de planificación y de revisión.
- Requerimientos de recursos de infraestructura o logísticos (máquinas, redes, licencias, papel, pizarras, etc.).
- Habilidades presentes y necesarias en el equipo.
- Restricciones asociadas al conocimiento del negocio, la tecnología o externas (legales, reglamentarias, estándares, etc.).

Rol del Scrum Master durante la planificación:

- Dirige la planificación.
- Es vínculo entre el equipo y el Product Owner del proyecto.
- Registra problemas y riesgos detectados durante la planificación.
- Registra las tareas, asignaciones y estimaciones.
- Inicia el Backlog del Sprint.

3.1.9 Sprint Backlog

Se refiere al trabajo o tareas determinadas por el equipo para realizar en un Sprint. Es decir, la conversión de tareas a un producto funcional.

Características:

- Las tareas se estiman en una duración entre 1 a 20 horas de trabajo. Las de mayor duración deben intentar descomponerse en sub-tareas de ese rango de tiempo.
- La estimación se actualiza día a día.

3.1.10 Scrum diario (Daily)

Scrum asume que el proceso es complejo y que es necesario inspeccionarlo frecuentemente, por eso se realiza una reunión diaria de seguimiento. El encuentro diario impide caer en el dilema señalado por Fred Brooks: "¿Cómo es que un proyecto puede atrasarse un año? Un día a la vez".

El foco de la reunión es determinar el avance en las tareas y detectar problemas o "bloqueos" que estén haciendo lento el progreso del equipo o que eventualmente impidan a un

equipo cumplir con la meta del Sprint. La idea es que ningún problema quede sin resolver o, por lo menos, sin iniciar alguna acción de respuesta dentro de las 24 horas después de su detección. La reunión es además un espacio definido para que cada miembro del equipo comunique a los demás el estado de su trabajo y por lo tanto reafirme el compromiso.

3.1.11 Rol del Scrum Master durante el Scrum

Dentro de los roles del Scrum Master durante el Scrum encontramos los siguientes:

- Dirigir la reunión y mantener el foco de atención.
- Realizar preguntas para aclarar dudas.
- Registrar, escribir y/o documentar los problemas para su resolución después de la reunión.
- Asegurarse que los miembros cuenten con el ambiente adecuado para la reunión.

La figura 5 sintetiza a Scrum desde el proceso y flujo de trabajo, los roles participantes, las componentes, los distintos tipos de reuniones existentes y los valores que sigue esta metodología ágil de desarrollo de software.

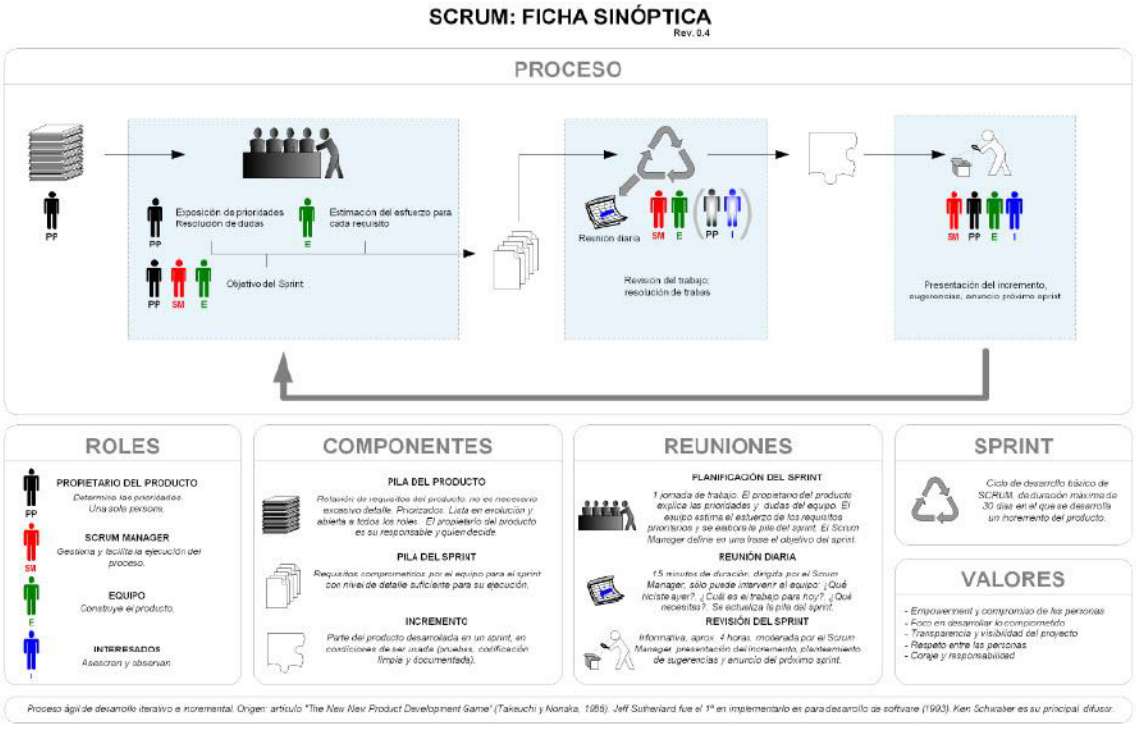


Figura 5 – Síntesis General de Scrum

3.1.12 Estimaciones

Las estimaciones se realizan por primera vez en la reunión de planificación al inicio del Sprint. Posteriormente, las tareas se re-estiman todos los días y se registran sus cambios en el Backlog del Sprint.

Esta actividad puede ser realizada inmediatamente antes o después del Scrum diario.

Algunas claves para la estimación son las siguientes:

- Siempre se realizan estimaciones de esfuerzo, no de duración.
- Siempre se estima el esfuerzo total pendiente para terminar la tarea, no se estima el esfuerzo consumido.
- Se buscan unidades manejables, lo usual es que estén en un mínimo de 2 horas y un máximo de 20. Si la tarea es muy corta se trata de juntarla con otras relacionadas. Si la tarea es muy grande se trata de descomponerla.

3.1.13 Builds continuos y pruebas básicas

La estrategia que generalmente se utiliza es la de Builds continuos y "smoke test" (prueba básica para la funcionalidad del sistema).

El procedimiento de Builds continuos es el siguiente:

i. Los programadores desarrollan según el Backlog del Sprint, y al finalizar, notifican al integrador.
ii. El integrador toma el código y lo integra con el resto del producto.
iii. Se compila el software y se prueba "por arriba" el producto, para verificar que no se haya roto.
iv. Si se encuentran problemas se devuelve al desarrollador.
v. Se notifica al equipo que hay una nueva versión "estable" del código para usar como base.

3.1.14 Revisión del Sprint

El objetivo de la reunión de revisión es presentar el producto o porción del producto desarrollada por el equipo a los usuarios. La reunión se utiliza para detectar inconformidades mayores que se vuelven elementos del Backlog de Producto y que eventualmente se resuelven en el siguiente Sprint. A la reunión asisten el equipo, el Scrum Master, el Product Owner y todas las personas implicadas en el proyecto.

3.1.15 Reunión retrospectiva

Scrum involucra el concepto de mejora continua a través de las reuniones de retrospección. Las reuniones buscan detectar los puntos positivos y negativos del Sprint para generar propuestas de mejora para futuros Sprints. Las reuniones de retrospección son el

concentrador del aprendizaje organizacional sobre el Scrum. Los puntos positivos y negativos se registran y se definen ítems de acción para cada uno. Los ítems de acción definidos se toman en cuenta en los siguientes Sprints.

A este tipo de reuniones asisten el equipo y el Scrum Master, y opcionalmente el Product Owner del Producto.

3.2 Rup

El Proceso Unificado Racional (Rational Unified Process en inglés, y sus siglas RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP no es un sistema con pasos firmemente establecidos, sino que trata de un conjunto de metodologías adaptables al contexto y necesidades de cada organización, donde el software es organizado como una colección de unidades atómicas llamados objetos, constituidos por datos y funciones, que interactúan entre sí.

RUP es un proceso para el desarrollo de un proyecto de software que define claramente *quien, cómo, cuándo* y *qué debe hacerse en el proyecto.*

RUP como proceso de desarrollo

- RUP es explícito en la definición de software y su trazabilidad, es decir, contempla en relación causal de los programas creados desde los requerimientos hasta la implementación y pruebas.
- RUP identifica claramente a los profesionales (actores) involucrados en el desarrollo del software y sus responsabilidades en cada una de las actividades.

3.2.1 Fases de desarrollo

Rup comprende las siguientes fases para el desarrollo de SW:

1) *Inicio*
2) *Elaboración*
3) *Construcción*
4) *Transición*

Fase de inicio

Se realiza un plan de fases, donde se identifican los principales casos de uso y se identifican los riesgos. Se concreta la idea, la visión del producto, cómo se enmarca en el negocio, y el alcance del proyecto. El objetivo en esta etapa es determinar la visión del proyecto.

Modelado del negocio

En esta fase, el equipo se familiariza más al funcionamiento de la empresa, sobre conocer sus procesos.

Objetivos y características:

- Entender la estructura y la dinámica de la organización para la cual el sistema va a ser desarrollado.
- Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.

Requisitos

En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que se especifican.

Objetivos y características:

- Establecer y mantener un acuerdo entre clientes y otros stakeholders sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.
- Definir el ámbito del sistema.
- Proveer una base para estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

Fase de elaboración

Se realiza el plan de proyecto, donde se completan los casos de uso y se mitigan los riesgos. Se planifican las actividades necesarias y los recursos requeridos, especificando las características y el diseño de la arquitectura. En esta etapa el objetivo es determinar la arquitectura Óptima.

Análisis y Diseño

En esta actividad se especifican los requerimientos y se describen sobre cómo se van a implementar en el sistema.

Objetivos y características:

- Transformar los requisitos al diseño del sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación.

Fase de construcción

Se basa en la elaboración de un producto totalmente operativo y en la elaboración del manual de usuario. Construir el producto, la arquitectura y los planes, hasta que el producto está listo para ser enviado a la comunidad de usuarios. En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.

Implementación

Se implementan las clases y objetos en archivos fuente, binarios, ejecutables y demás. El resultado final es un sistema ejecutable.

Objetivos y características:

- Planificar qué subsistemas deben ser implementados y en qué orden deben ser integrados, formando el Plan de Integración.
- Cada desarrollador decide en qué orden implementa los elementos del subsistema.
- Si encuentra errores de diseño, los notifica.
- Se integra el sistema siguiendo el plan.

Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del producto que se está desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino para ir integrado en todo el ciclo de vida.

Objetivos y características:

- Encontrar y documentar defectos en la calidad del software.
- Generalmente asesorar sobre la calidad del software percibida.
- Proveer la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- Verificar las funciones del producto de software según lo diseñado.
- Verificar que los requisitos tengan su apropiada implementación.

Fase de transición

El objetivo es llegar a obtener el release del proyecto. Se realiza la instalación del producto en el cliente y se procede al entrenamiento de los usuarios.

Objetivos y características:

- Realizar la transición del producto a los usuarios, lo cual incluye: manufactura, envío, entrenamiento, soporte y mantenimiento del producto, hasta que el cliente quede satisfecho. Por ende, en esta fase suelen ocurrir cambios.

Despliegue

Esta actividad tiene como objetivo producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

- Probar el producto en su entorno de ejecución final.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.
- Formar a los usuarios y al cuerpo de ventas.
- Migrar el software existente o convertir bases de datos.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

A medida que se avanza en el proyecto, es decir, cuando se va pasando de una fase a otra, la importancia relativa de cada uno de los flujos de trabajo va cambiando. Así, en las iteraciones de la fase de inicio, el trabajo se centra principalmente en el modelamiento del negocio y en la captura y especificación de requisitos. Pero en la fase de construcción, el desarrollo está enfocado en la implementación (codificación) y, en menor medida, en el diseño.

En la figura 6, se pueden visualizar las distintas fases de Rup y su relación con los flujos de trabajo que se realizan a través de las correspondientes iteraciones.

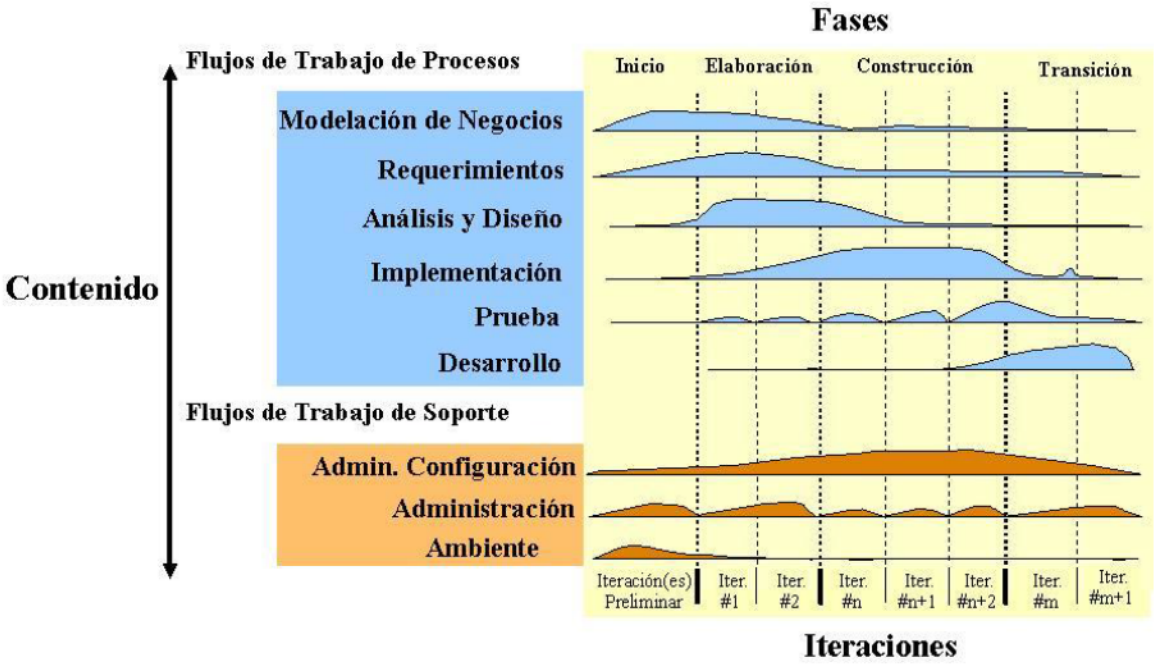


Figura 6 – Fases y flujos de trabajo en Rup

3.2.3 Principios de RUP

Rup presenta diversos principios entre los que se destacan los siguientes:

- ***Metodología basada en casos de uso.***

Según [5] "los casos de uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema".

En RUP los Casos de Uso se utilizan para especificar los requisitos del sistema y además guían su diseño, implementación y prueba.

- **Adaptación del proceso**

El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

- **Balancear prioridades**

Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

- **Colaboración entre equipos**

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc.

- **Demostrar valor iterativamente**

Los proyectos se entregan, aunque sea de modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

- **Elevar el nivel de abstracción**

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o frameworks por nombrar algunos. Estos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

- **Enfocarse en la calidad**

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

3.2.4 Elementos de RUP

Un proceso de desarrollo de software define 4 aspectos: "Quién" hace, "Qué" hace, "Cómo" hace, "Cuándo" hace.

De acuerdo a lo anteriormente dicho, Rup define 4 elementos:

- *Roles*: que responden a la pregunta ¿Quién?
- *Actividades*: que responden a la pregunta ¿Cómo?
- *Productos(o artefactos)*: que responden a la pregunta ¿Qué?
- *Flujos de trabajo*: que responden a la pregunta ¿Cuándo?

Roles

Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas.

Las responsabilidades de un rol son las siguientes: la de llevar a cabo un conjunto de actividades y la de ser el dueño de un conjunto de artefactos.

Según [6] RUP define grupos de roles, agrupados por participación en actividades relacionadas. Estos grupos son los que se mencionan a continuación:

Analistas:

- Analista de procesos de negocio.
- Diseñador del negocio.
- Analista de sistema.
- Especificador de requisitos.

Desarrolladores:

- Arquitecto de software.
- Diseñador.
- Diseñador de interfaz de usuario.
- Diseñador de cápsulas.
- Diseñador de base de datos.
- Implementador.
- Integrador.

Gestores:

- Jefe de proyecto.
- Jefe de control de cambios.
- Jefe de configuración.

- Jefe de pruebas.
- Jefe de despliegue.
- Ingeniero de procesos.
- Revisor de gestión del proyecto.
- Gestor de pruebas.

Apoyo:

- Documentador técnico.
- Administrador de sistema.
- Especialista en herramientas.
- Desarrollador de cursos.
- Artista gráfico.

Especialista en pruebas:

- Especialista en Pruebas (tester).
- Analista de pruebas.
- Diseñador de pruebas.

Otros roles:

- Stakeholders.
- Revisor.
- Coordinación de revisiones.
- Revisor técnico.

Productos/Artefactos

Un producto o artefacto es un trozo de información que es producido, modificado o utilizado durante el proceso de desarrollo de software. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final.

Un artefacto puede ser cualquiera de los siguientes:

- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de Casos de Uso o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un Caso de Uso o un subsistema.

Flujos de trabajo

Con la enumeración de roles, actividades y artefactos no se define un proceso, se necesita contar con una secuencia de actividades realizadas por los diferentes roles, así como la relación entre los mismos. Un flujo de trabajo es una relación de actividades que producen resultados observables.

Los flujos de trabajo son las 4 fases que realiza Rup como proceso de desarrollo, que fueron explicadas anteriormente:

- Inicio
- Construcción
- Elaboración
- Transición

3.2.5 Productos/Artefactos de Rup

De acuerdo a las fases y/o flujos de trabajo de Rup se generan distintos productos o artefactos. A continuación se mencionarán los mismos de acuerdo a las fases de la metodología Rup.

Inicio

- Documento Visión.
- Diagramas de caso de uso.
- Especificación de Requisitos.
- Diagrama de Requisitos.

Elaboración

Documento de arquitectura que trabaja con las siguientes vistas:

Vista Lógica

- Diagrama de clases.
- Modelo E-R (Si el sistema así lo requiere).

Vista de Implementación

- Diagrama de Secuencia.
- Diagrama de estados.
- Diagrama de Colaboración.

Vista Conceptual

- Modelo de dominio.

Vista física

- Mapa de comportamiento a nivel de hardware.
- Diseño y desarrollo de casos de uso, o flujos de casos de uso arquitectónicos.
- Pruebas de los casos de uso desarrollados, que demuestran que la arquitectura documentada responde adecuadamente a requerimientos funcionales y no funcionales.

Construcción

- Especificación de requisitos faltantes.
- Diseño y desarrollo de casos de uso y/o flujos de acuerdo con la planeación iterativa.
- Pruebas de los casos de uso desarrollados, y pruebas de regresión según sea el caso.

Transición

- Pruebas finales de aceptación
- Puesta en producción
- Estabilización

3.3 Comparación Scrum Vs. Rup

En la tabla 1 se realiza una comparación de Scrum y Rup teniendo en cuenta diversos aspectos.

	SCRUM	RUP
Clasificación	Metodología Ágil	Metodología tradicional y/o rígida
Enfoque	Iterativo	Iterativo
Ciclo	Cada Sprint (iteración)es un ciclo completo.	Consta de 4 fases: Inicio, Elaboración, Construcción y transición, aunque algunas pueden realizarse concurrentemente.
Planificación	Por cada sprint existe una fecha de entrega que estipula el Scrum Master durante la planificación. Durante la misma se detalla el trabajo a	Se basa en un plan de proyecto formal asociada a múltiples iteraciones. Existe una fecha final así como también hitos [xi]

	realizar a través de la estimación de esfuerzo para cada tarea. También cabe mencionar que se fija un alcance, es decir, que los objetivos durante el sprint no se van a modificar.	intermedios.
Alcance	Los objetivos se determinan en el "sprint backlog" (durante la planning) y no pueden ser modificados.	Está predefinido en el documento de alance antes del inicio del proyecto.
Elementos	-Roles(Product owner, Scrum Master, Team) -Poda de requerimientos -Product Backlog -Sprint(Planificación, Sprint backlog, Scrum, estimaciones, builds continuos, revisión del sprint, reunión retrospectiva) -valores (foco, comunicación, respeto)	- Roles(Analistas, Desarrolladores, Gestores, Apoyo, Especialista en pruebas) - Actividades - Productos/Artefactos (Documento de visión, Diagramas de CU., Diagrama de Clases, Modelo E-R, Diagrama de secuencia, Diagrama de estados, etc) - Flujos de trabajo(Inicio, Elaboración, Construcción, Transición)

Tipo de proyectos	Para proyectos que necesitan mejoras rápidas y organizaciones que no dependen de una fecha límite. Así como también se amoldan en proyectos en los cuales los requerimientos son flexibles y/o el ámbito el tendiente a cambios.	Óptimos en grandes proyectos y/o proyectos a largo plazo. También aplicables en proyectos de media/alta complejidad. En cuanto a los requerimientos es mejor para proyectos con requerimientos rígidos y/o proyectos que no tienden a modificar y/o agregar requerimientos(resistentes al cambio).
Documentación	Baja cantidad de documentación.	Exhaustiva y detallada documentación.
Énfasis	Orientada a las personas.	Orientada a los procesos.

Tabla 1 – Tabla Comparativa de Scrum y Rup

CAPITULO 4. Nuestro Proyecto de referencia: eSidif

4.1 Introducción a eSidif

En este apartado vamos a explicar y describir los aspectos fundamentales del proyecto sobre el cual se basa nuestra tesis. Es decir, el proyecto en el que se aplica la convivencia de metodologías Scrum y Rup, y en el cual participamos.

El proyecto en cuestión es denominado "***Esidif***" y su objetivo es la formulación del presupuesto nacional y registro de la ejecución presupuestaria. Este proyecto se realiza en colaboración entre la Universidad Nacional de La Plata (UNLP) y el Estado Nacional. Por ello el Laboratorio de Investigación y Formación de Informática Avanzada (LIFIA) se encuentra brindando soporte a la Secretaría de Hacienda del Ministerio de Economía de La Nación. [10]

4.2 Evolución Histórica

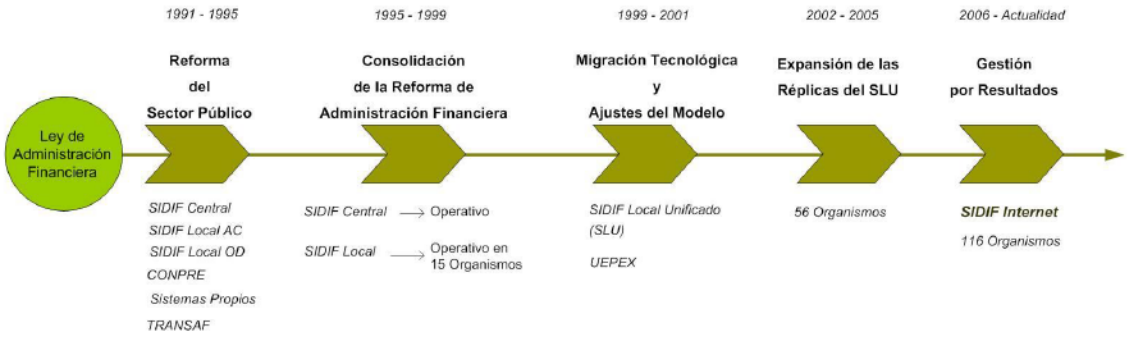


Figura 7 – Evolución del proyecto

El 30 de septiembre del año 1992 se sancionó la Ley N°24.156 de Administración Financiera y de los Sistemas de Control del Sector Público Nacional. A partir de allí, se inició el camino en la implementación del proyecto (e-Sidif) desde sus inicios y variando a través de sus diversas transformaciones, hasta lo que es en la actualidad (como se visualiza en la figura 7).

En su versión inicial, el S.I.D.I.F (Sistema Integrado de Información Financiera) fue adoptado como un sistema integrado, que estaba compuesto por varios subsistemas y/o módulos dentro de una visión funcional, que contemplaba la distribución de la base de datos lógica, en una base de datos central y tantas bases institucionales como Servicios de Administración Financiera (SAF) existieran. Esta estructura física era soportada por arquitecturas abiertas y redes locales trabajando en la modalidad "Cliente/Servidor" (C/S).

Bajo este esquema operativo, la Secretaría de Hacienda se comunicaba, a través del SIDIF Central con los sistemas periféricos instalados en los organismos (sistemas locales), con el sistema de gestión para las Unidades Ejecutoras de Préstamos Externos y demás aplicaciones que interactúan con la base de datos central. A través de esta comunicación la base central concentraba el registro de la ejecución presupuestaria. La información de gestión permanecía en las bases locales.

Dentro del S.I.D.I.F. convivían diversos sistemas locales en las distintas entidades con características diferentes. Esta diversidad hizo que se incrementara el costo en mantenimiento y de replicación de las adecuaciones en los sistemas locales, por ende en algunas oportunidades esto recaía en una demora en la disponibilidad.
A partir de los problemas y costos anteriormente mencionados comenzó una etapa de transformación. Es decir, sustituir a los diversos sistemas locales por un único sistema, esto es, a través de la implementación del SLU (S.I.D.I.F Local Unificado).
Este proceso reemplazó los sistemas locales existentes en distintos organismos por la versión unificada con mejoras en cuanto a la provisión de información confiable para la alta gerencia de las entidades, el ajuste de procedimientos de compras, presupuesto, contabilidad y tesorería, y la reducción de aplicativos complementarios requeridos para soportar la gestión.

Luego de la primer transformación, en la cual se introdujeron importantes mejoras en la gestión a través del SLU, se planteó una segunda transformación, en la cual se hizo hincapié en la extensión funcional y en la renovación tecnológica. Esta segunda transformación fue posible con el sistema "e-Sidif". Este sistema busca mejorar la calidad del S.I.D.I.F a través de la incorporación de innovaciones tecnológicas de las comunicaciones, que se han producido en la última década.

4.3 Características

A continuación se mencionan unas breves características del proyecto e-Sidif:

- Posee una base de datos única.
- Provee de gestión, registro y control simultáneos.
- Presenta mejoras en las buenas prácticas de administración financiera.
- Beneficio de ser el único sistema de administración financiera.
- Es de fácil uso.
- Incorpora seguridad y auditoría.
- Se tiene la posibilidad de acceder a distintas visiones de la información.
- Autonomía del usuario.
- Provee descentralización operativa.
- Presenta facilidad para los usuarios a la hora de la toma de decisiones.

Cuenta con una serie de sub-proyectos que son los siguientes:

- Convivencia
- Migración

- Desarrollo
- Mantenimiento de los módulos implementados

El sub-proyecto de convivencia se refiere a un conjunto de programas que mantienen sincronizadas y/o espejadas las bases de datos de los sistemas a reemplazar y el nuevo sistema. Por otra parte, migración hace referencia al procedimiento de extraer los datos del sistema actual e importarlos en el nuevo sistema, realizando las transformaciones que sean necesarias.

Por último, otra característica importante a tener en cuenta del sistema eSidif es la forma o estrategia de despliegue del mismo. Esta estrategia se basa en la *puesta en producción de manera incremental*. Esto es, a medida que se desarrollan los módulos. Cabe mencionar que se eligió esta estrategia debido a las fuertes demandas de las autoridades para *minimizar el impacto de las nuevas tecnologías en los usuarios.*

4.4 Arquitectura tecnológica, lenguaje y metodología

En esta sección explicaremos brevemente qué tipo de arquitectura, qué lenguaje y qué metodología de desarrollo es utilizada en el proyecto eSidif.

La de arquitectura utilizada es JEE[xii]. Aunque se comenzó utilizando la versión 1.4 de J2SE[xiii] y 1.3 de J2EE, pero a medida que las versiones de dichas tecnologías fueron evolucionando se migró a Java SE 5, y por último se migró a la actual, es decir, JEE, para hacer uso de las nuevas características provistas.

El lenguaje utilizado para la programación del sistema es JAVA mientras que el entorno elegido es Eclipse.

Por último, tenemos que decir que el tipo de metodología utilizada en el proyecto es RUP pero con sus adaptaciones ágiles (particularmente Scrum) relacionadas con las necesidades del proyecto. **Cabe mencionar que sobre esta metodología y la convivencia de ambas se basa nuestra tesis**, así que se retomará y explicará detalladamente este tema en los apartados siguientes.

En la *figura 8*, se visualiza un diagrama con la arquitectura de la aplicación.

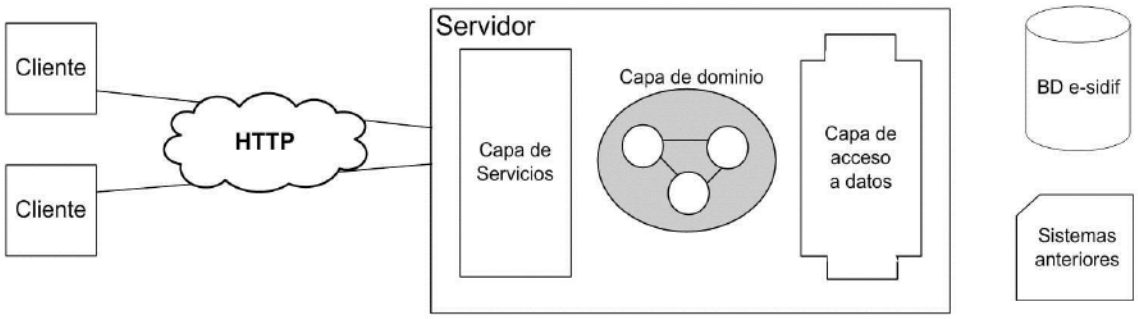


Figura 8 – Arquitectura de la aplicación

4.5 Conformación de los grupos de trabajo

Como el nombre de la tesis lo indica, uno de los objetivos principales es explicar la convivencia de dos metodologías que son Rup y Scrum, pero lo que también mencionamos es que dicha convivencia se realiza en un **proyecto de gran escala**. Con esto nos referimos a un número elevado de personas involucradas en el desarrollo del mismo, como así también a un gran volumen de codificación.

Este apartado tiene como fin explicar cómo está conformado el proyecto en cuanto a los grupos de trabajo, desde los niveles más bajos como Diseño y Desarrollo, hasta los más altos como la Coordinación General.

Para ello se presenta un organigrama (actualizado con datos de fecha correspondiente al año 2009).

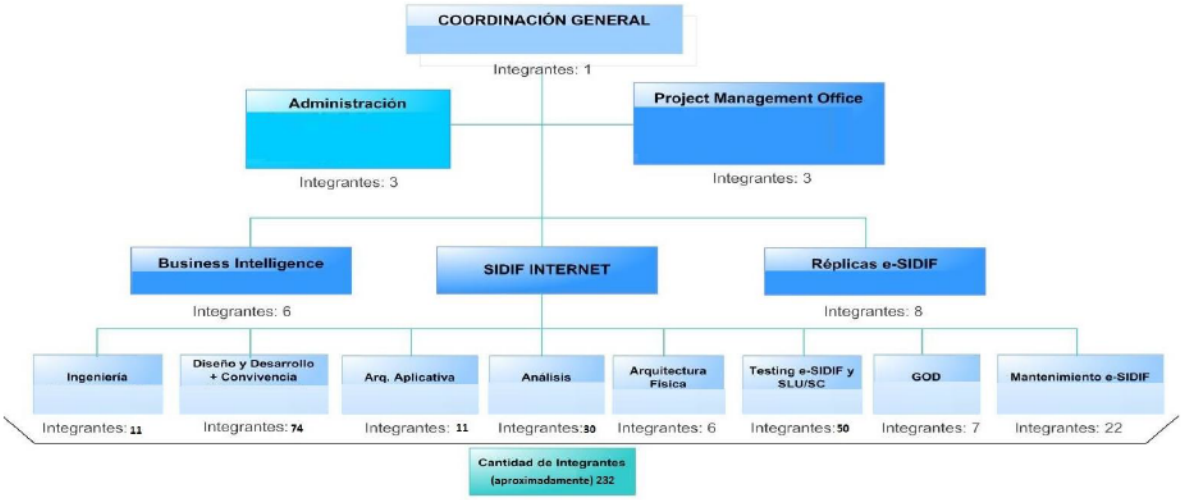


Figura 9 – Organigrama del proyecto

En la figura 9 (correspondiente al organigrama del proyecto) se puede observar que existen distintos grupos de trabajo, los cuales realizan diferentes tareas. A continuación se explicará brevemente qué tipo de tareas realiza cada uno, así como también se detallará cada área de acuerdo a sus funciones y conformación.

Comenzando por la parte inferior izquierda de la figura 9, se encuentra el área de *Ingeniería*. Entre los objetivos de esta área podemos nombrar el de definir el proceso de desarrollo del eSidif, así como también el de proveer al equipo del proyecto un ambiente estándar de herramientas que soporten el proceso de desarrollo definido (estas herramientas se tratarán en el apartado 5.6 Herramientas utilizadas en el proyecto).

Esta área está conformada por 11 personas, y entre sus actividades se destacan:

- Definición de procesos de desarrollo.
- Definición de guías de trabajo.
- Proveimiento de un ambiente estándar.
- Aseguramiento del cumplimiento de Calidad.
- Investigación de las disciplinas del proceso Rup, adaptándolas al proyecto y definiendo las actividades, roles y artefactos.
- Estandarización de cada tipo de artefacto: templates, Ejemplos, Guías de buenas prácticas, Material de capacitación, etc.
- Gestión de Control de calidad de los artefactos generados (Estos artefactos van a ser explicados en el capítulo 6, más precisamente en la sección 6.3 Disciplinas, roles, responsabilidades y artefactos generados con Rup en eSidif").
- Elaboración de guías de administración, instalación y uso de herramientas.
- Capacitación y soporte al equipo de desarrollo, a través de sus disciplinas, artefactos y herramientas de soporte.
- Investigación y adaptación de prácticas de procesos ágiles al proyecto.

A su vez, el área de Ingeniería se encuentra dividida de acuerdo a las funciones que realiza cada persona. En la figura 10 se visualiza un organigrama del área de Ingeniería, donde se indica la función y la cantidad de personas involucrada en dicha área.

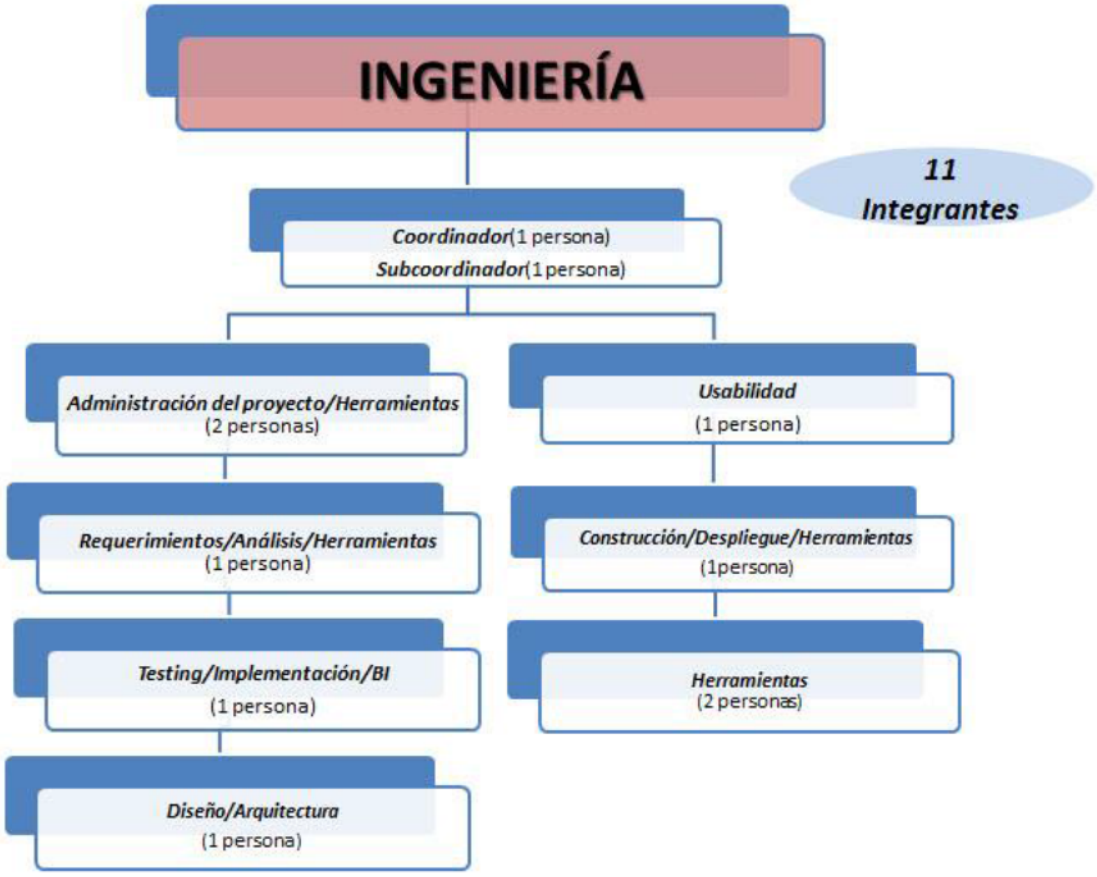


Figura 10 – Estructura y conformación del área de Ingeniería

Continuando en el mismo nivel de jerarquía (hacia la derecha de la figura 9), se encuentra el área de *Diseño y Desarrollo + Convivencia*.

Esta área (en conjunción, es decir, DyD más Convivencia) está conformada por alrededor de 74 personas y entre sus actividades se destacan:

- Diseño y Construcción de artefactos.
- Realización de ejecutables para los usuarios.

En particular, el área de Diseño y Desarrollo, tiene como misión materializar los requerimientos del usuario en artefactos visibles y ejecutables por el usuario. Entre sus funciones se destacan:

- Modelado y construcción de componentes para ejecutar la funcionalidad del sistema.

- Construcción de los algoritmos de convivencia entre sistemas legados desarrollados por el equipo de convivencia.

El área de *Diseño y Desarrollo (DyD)*, está conformada aproximadamente por 67 personas (incluyendo a los 2 coordinadores).

En la figura 11 se visualiza el organigrama de Diseño y Desarrollo con sus correspondientes equipos/negocios y cantidad de integrantes.

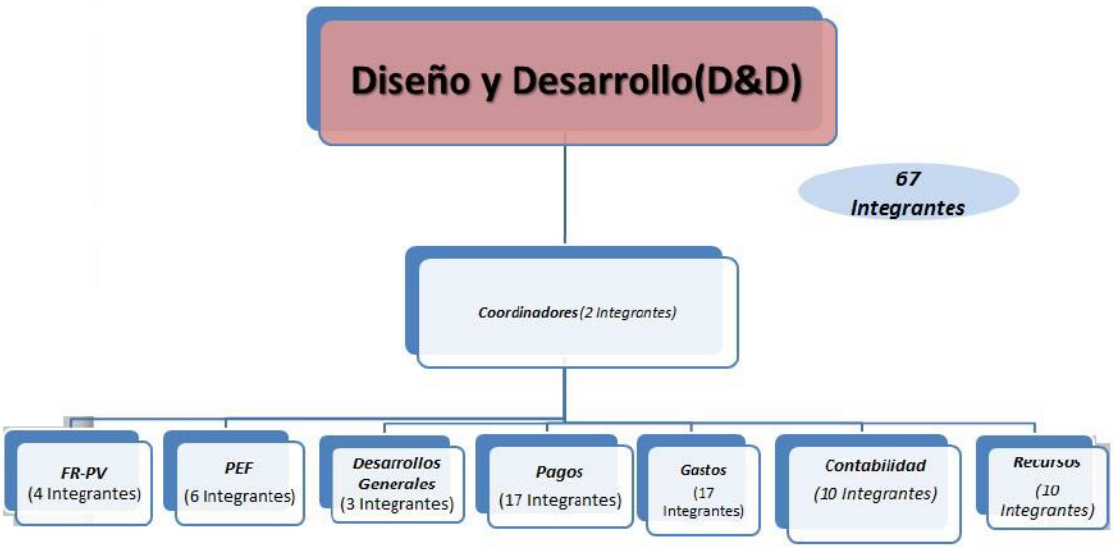


Figura 11 - Estructura y conformación del área de DyD

Como se puede observar en la figura 12, el área de *Convivencia* está constituida por 7 integrantes y tiene como funciones principales:

- Construcción de algoritmos en lenguajes PL/SQL los cuales garanticen la actualización y el control de los datos, entre todos los sistemas que conforman el eSidif.

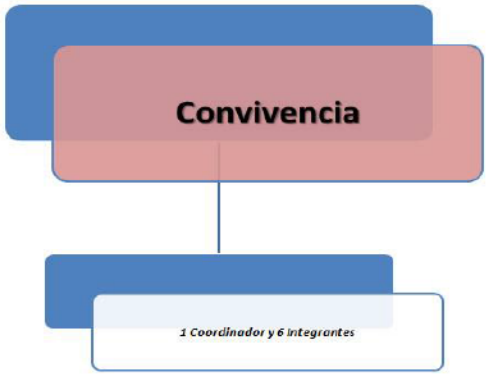


Figura 12 - Estructura y conformación del área de Convivencia

Otra área es la de *Arquitectura Aplicativa*, la cual contiene 11 integrantes y su misión es definir la arquitectura de software del eSidif y la plataforma de desarrollo sobre la cual se construye el producto. Además se encarga de proveer de actualización tecnológica de la plataforma, y de mantenimiento permanente de la misma, así como también de brindar soporte a los demás equipos.

Entre sus funciones principales se encuentran:

- Definición de la arquitectura de software.
- Gestión de la plataforma de desarrollo, que incumbe desde la definición, diseño, implementación y mantenimiento de la misma.
- Evaluación de las nuevas tecnologías que pueden ser de utilidad para la plataforma.
- Realización de manuales de uso para cada uno de los componentes de la plataforma.

Esta área, de acuerdo a sus funciones se divide en diversas sub-áreas que se visualizan en la figura 13.

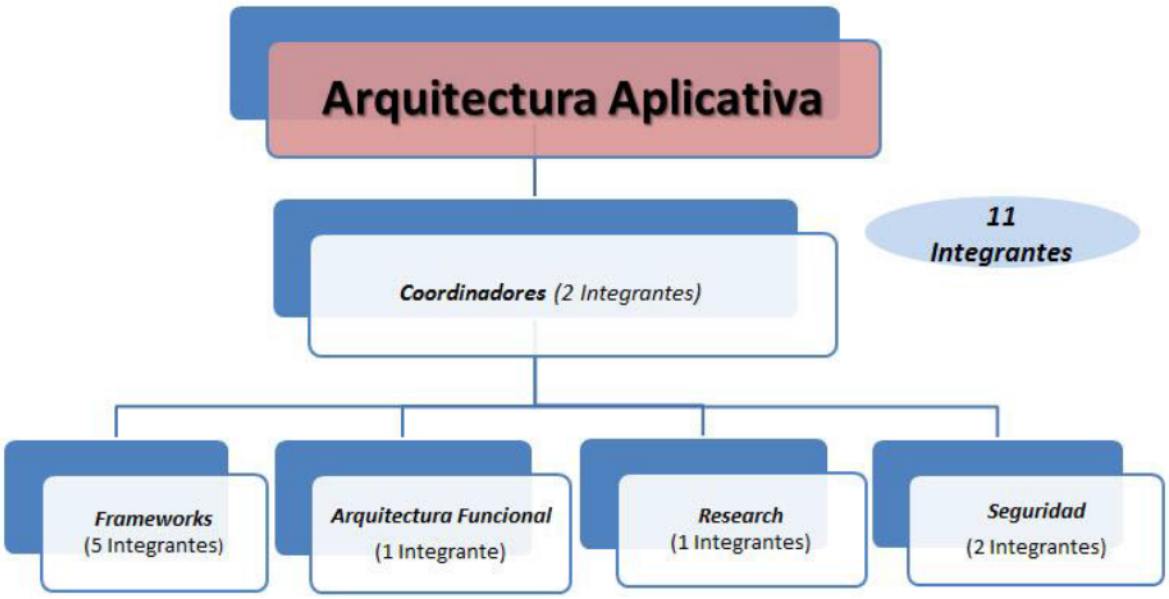


Figura 13- Estructura y conformación del área de Arquitectura Aplicativa

El área de *Análisis*, está formada por aproximadamente 30 integrantes (incluidos el coordinador y los 2 sub-coordinadores) y su misión es la de interpretar los requerimientos del usuario y realizar la concepción de la solución plasmándola en los modelos definidos en el proceso mediante el lenguaje de modelado UML. Entre sus actividades principales figuran:

- Interpretación de requerimientos del usuario.
- (A partir de los requerimientos obtenidos) el modelamiento de la solución en los diagramas de Casos de Uso correspondientes.

- Estimación del tamaño y esfuerzo de cada negocio.
- Implementación de los modelos UML definidos en la metodología, ya sean, MDR, MCU, MRN, MCA.
- Definición de la estrategia y documentación del análisis de convivencia y migración.
- Generación de la documentación del despliegue.
- Revisación y control de los casos de prueba, y modelos de bases de datos desarrollados.

Esta área se encuentra subdividida para realizar un análisis de cada equipo/negocio en particular. Esto se puede visualizar en la figura 14.

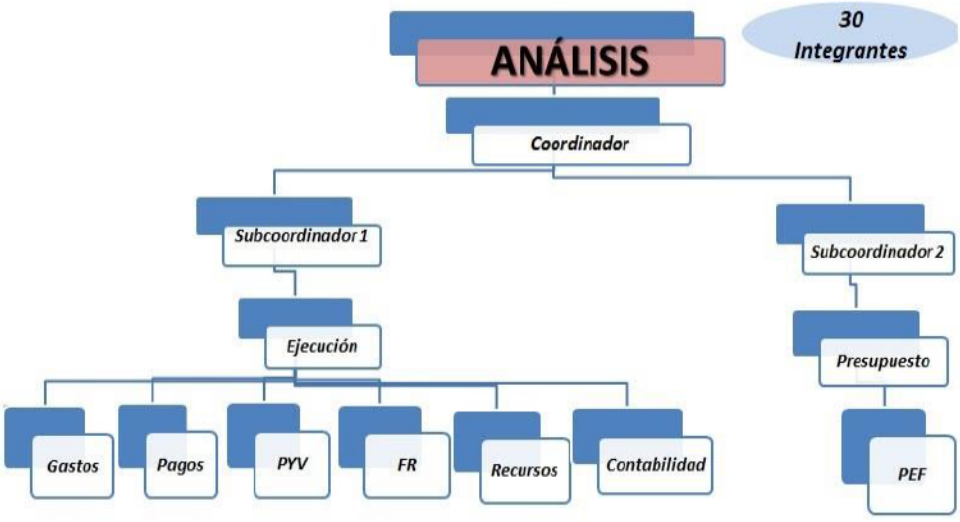


Figura 14 - Estructura y conformación del área de Análisis

El área de *Arquitectura Física*, está compuesta por 6 integrantes. Su actividad es la de monitorear la capa de aplicación para garantizar el buen funcionamiento aplicativo.

Testing eSidif y SLU-SC es un área la cual cuenta con una gran cantidad de integrantes, alrededor de 50 personas (incluidos el coordinador y los dos sub-coordinadores). Entre sus actividades se destacan:

- Control de la calidad de los ejecutables.
- Verificación del cumplimiento de los requerimientos del usuario, esto es, a través de casos de prueba.

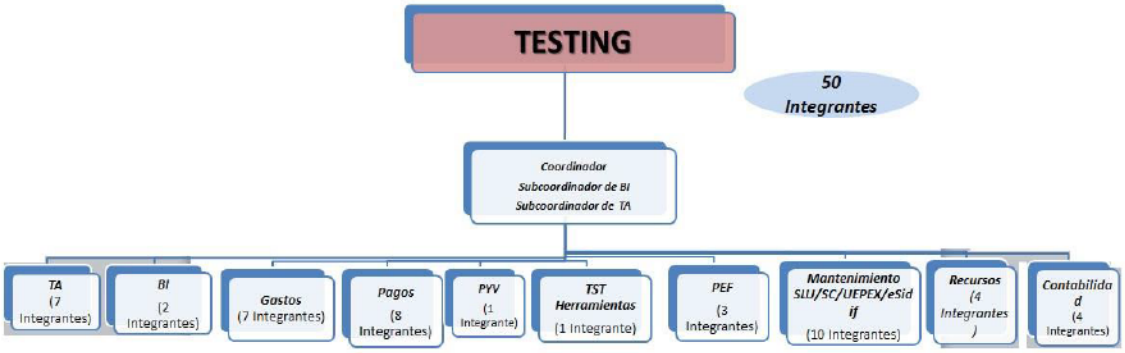


Figura 15 - Estructura y conformación del área de testing

GOD (Grupos de Orientación del diseño), es un área la cual cuenta con 7 integrantes. Es un área pequeña pero también tiene actividades interesantes. Entre ellas se destacan:

- Optimización de los productos, tanto intermedio como final.
- Promoción del reuso de código normalizado.
- Fomentación del reuso de pantallas comunes.
- Participación en los procesos de aseguramiento de calidad de modelado de Análisis y Diseño.
- Colaboración con el área de Arquitectura para implementar componentes generales.

Por último, en este nivel de jerarquía se encuentra el área de *mantenimiento*. Ya sea, mantenimiento del eSidif como mantenimiento del SLU/UI (SIDIF Local Unificado/Unidad Informática). Esta área está conformada por 22 integrantes. Tiene como objetivo, tomar los requerimientos de los usuarios de las aplicaciones que se encuentran en producción, los cuales pueden ser una nueva funcionalidad o de corrección. Entre sus funciones principales se encuentran:

- Interpretación y modelado de modificaciones de los requerimientos del usuario.
- Modificación de los ejecutables.
- Participación en las reuniones de priorización de los requerimientos.
- Participación en los emprendimientos de mejoras de los procesos.

Como dijimos anteriormente, está conformada por aproximadamente 22 integrantes (incluido el coordinador) pero se encuentra subdividida en distintos tipos de mantenimiento, dependiendo del tipo del proyecto en el cual se esté participando. Esto puede observarse más claramente en la figura 16, donde se encuentran los 3 sistemas a los cuales se provee mantenimiento.

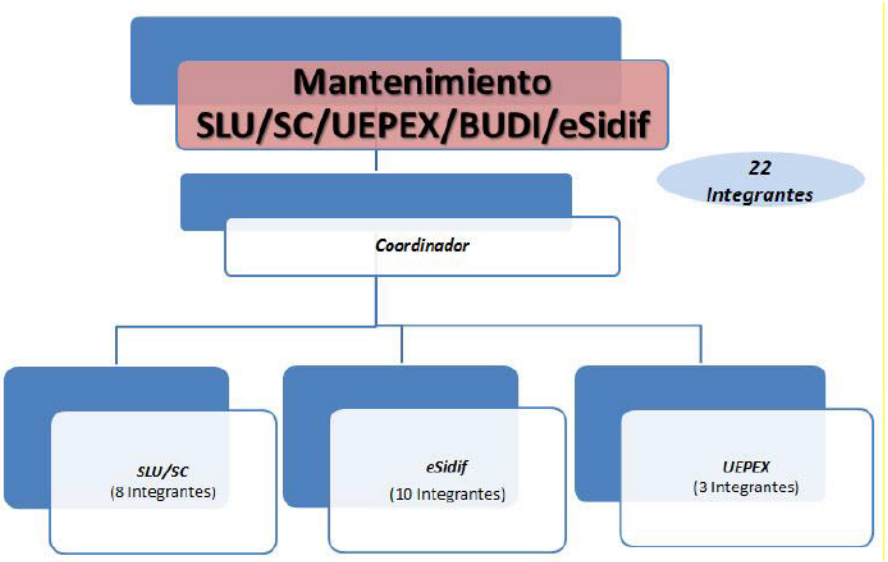


Figura 16 - Estructura y conformación del área de mantenimiento

En el segundo nivel de jerarquía de la figura 9, que como dijimos corresponde al organigrama del proyecto, se encuentran dos áreas, que son las siguiente: *Business Intelligence (BI)* y *Réplicas eSidif.*

La primera área *(BI)* está compuesta por 6 integrantes y se encarga de:

- Interpretar requerimientos de niveles ejecutivos.
- Desarrollar e implementar con herramientas gerenciales.

La segunda área, es decir, la de *réplicas*, está compuesta por 8 integrantes y tiene distintas funciones de acuerdo a las etapas en que se encuentre el proyecto.

Cuando el proyecto está en pasaje a producción, se encarga de la realización de demos y capacitaciones de los usuarios hasta que alcancen autonomía en el uso de la aplicación.

En cambio, cuando el proyecto se encuentra durante las etapas de desarrollo, realiza la participación en talleres de definición funcional y reuniones de avance y evaluación de negocios implementados y a implementar.

Prosiguiendo con la figura 9, en el tercer nivel en la jerarquía del organigrama se encuentran dos áreas: *Administración* y *PMO (Project Management Office).*

Primeramente, el área de Administración está conformada por 3 integrantes. Entre las tareas que realizan se destacan:

- Administración del personal.
- Gestión de Expedientes.
- Generación de la documentación.

Por otra parte, PMO está constituida también por 3 integrantes y su objetivo es ejecutar con los distintos coordinadores del proyecto, los aspectos específicos relacionados con la gestión, haciendo hincapié en la planificación y seguimiento del proyecto. Entre sus actividades principales encontramos:

- Gestión de la planificación.
- Evaluación y proposiciones de mitigación.
- Gestión del Seguimiento.
- Gestión de la contratación de recursos.
- Gestión y seguimiento de riesgos.
- Gestión del despliegue.

El cuarto nivel de jerarquía en el organigrama corresponde al área de *Coordinación general*. Ésta área está conformada por una persona y se encarga (entre otras tantas actividades) de elaborar planes para el cumplimiento de los objetivos de las SSP (Sub-Secretaria de Presupuesto).

4.6 Herramientas utilizadas en el proyecto

Como se dijo en uno de los apartados anteriores, el área de Ingeniería se encarga de la elaboración de guías de administración, instalación y uso de herramientas que son utilizadas en las distintas etapas del desarrollo del proyecto. En la *tabla 2* se mencionan las distintas herramientas utilizadas en el proyecto junto con una breve descripción de su utilidad.

HERRAMIENTA	DESCRIPCIÓN
Eclipse	*Es un entorno integrado de desarrollo de software.*
Enterprise Architect(EA)	*Es una herramientas CASE de modelado UML. Se documentan los artefactos(MCA, MCU, RCU, MBD, DTE,etc).*
CVS	*Concurrent Version System, es un sistema de control de versiones. Se utiliza integrado en Eclipse para sincronizar el código desarrollado por las distintas áreas.*
MOIN MOIN	*Corresponde a la wiki, que es el portal colaborativo de las personas participantes en el proyecto.*
PSI	*Es un software de mensajería instantánea que permite la comunicación interna de los integrantes del proyecto.*
MOZILLA SUNBIRD	*Calendario.*
NEA	*Es un software para generar notas de entregas automáticas.*
GLO	*Se refiere al Glosario funcional.*
Windows Tester	*Se encarga de la automatización de las pruebas.*
ITEM	*Sistema de Software desarrollado por la UNLP, que permite la administración y gestión de requerimientos de desarrollo.*
BWA Build With Ant	*Software que permite la automatización del proceso de construcción de ADD.*
Sitio BWA	*Sitio que visualiza reportes de la construcción de la aplicación.*
BAMBOO	*Integración continúa.*
Toolkit	*Automatiza las tareas recurrentes de los desarrolladores. Es un plugin integrado a Eclipse desarrollado por el área de ARQ.*
SGR	*Permite la gestión de requerimientos de todo tipo, principalmente de defectos hallados en producción.*
Agilefant	*Gestión de equipos ágiles en base a nuestra adaptación de SCRUM.*
QuickItem	*Herramienta que permite:- Acceso rápido a información administrada por el ITeM;- Generación de reportes; - Interacción con Agilefant*
WIKI	*Es un repositorio de información que se encuentra accesible para todos los equipos.*

Tabla 2 – Herramientas utilizadas

CAPÍTULO 5. Nuestra propuesta de Proceso de desarrollo combinado

En este capítulo se va a explicar la manera de cómo se trabaja en el proyecto eSidif.

En primer término, se va a explicar cómo funcionan los equipos de DyD, con respecto a SCRUM. Luego se va a explicar el funcionamiento de RUP en el proyecto.

Para explicar los aspectos de Scrum aplicados al proyecto partimos por definir y detallar distintos conceptos claves y la manera en que son utilizados los mismos a través de algunos ejemplos.

5.1 Aspectos de Esidif Relacionados con SCRUM

REUNIONES

Planning

La planning y/o planificación hace referencia a la planificación del sprint.

En esta reunión, (que se realiza el primer día hábil del sprint) el equipo/negocio se reúne con el objetivo de realizar la planificación de las tareas correspondiente a dicho sprint, es decir, las tareas que se esperan realizar.

En la fase de estimación de las tareas participan todos los integrantes del equipo, donde cada uno indica el tiempo que supone puede llevar la realización de dicha tarea y luego se saca el promedio total. A la hora de la estimación se pueden utilizar como referencia el registro de tareas similares realizadas en iteraciones previas.

Las herramientas que se utilizan para documentar la planificación, y generar el ***backlog del sprint*** fueron variando. En un principio se utilizaban hojas de cálculo (estilo Excel), con la lista de tareas correspondientes y su estimación prevista. Estas tareas se transformaban en tarjetas que luego eran colocadas en una pizarra, de donde los desarrolladores las tomaban y registraban su desarrollo y estado.

Posteriormente, el área de Ingeniería evaluó y optó por comenzar a utilizar una herramienta de gestión de proyectos, que se denomina ***Agilefant,*** y es la que se utiliza actualmente.

Dailys

Las dailys son las reuniones diarias, y se realizan todos los días, excepto los días de planificación y/o integración.

Estas reuniones se realizan por cada negocio, y en un horario acordado para todos los días.

Forman parte de estas reuniones, el grupo de DyD, el grupo de analistas y el grupo de testers.

El objetivo de la daily es facilitar la transferencia de información y la colaboración entre los miembros del equipo para aumentar su productividad, al poner de manifiesto puntos en que se pueden ayudar unos a otros.

En cuanto a la duración no deberían extenderse de 20 minutos.

Demo's

El objetivo de esta reunión está en mostrar a los analistas (que en esta instancia cumplen el rol de cliente) el producto desarrollado en el sprint, haciendo un recorrido por los caminos más relevantes de la funcionalidad planificada. La finalidad de esta reunión es la detección temprana de bug's o cambios en los requerimientos.

Se lleva a cabo en una sala, a través de un proyector donde una de las personas del equipo va mostrando lo mencionado anteriormente. Es muy interesante el feedback que se genera entre las distintas partes, donde generalmente surgen cuestiones a tener en cuenta, como por ejemplo la mirada del área de usabilidad y/o cuestiones que no se llegan a visualizar o no se consideran durante el desarrollo.

Esta reunión se realiza un día particular que es acordado por los distintos participantes involucrados.

Forman parte de la demo, el equipo de desarrollo, los testers, el área de usabilidad y los analistas.

Reunión retrospectiva

La reunión retrospectiva, conocida como "retro", son reuniones utilizadas para profundizar en temas que le interesan al equipo. Esto permite a cada integrante del equipo realizar aportes desde su punto de vista y contribuir a las soluciones de los problemas que tiene el equipo.

Es una actividad importante, que permite identificar y fortalecer las buenas prácticas realizadas, como así también encontrar las razones por las cuales un objetivo determinado no pudo ser alcanzado con éxito.

Se abarcan temas relacionados con la interacción del equipo y la comunicación entre los integrantes del mismo.

Integraciones

Las integraciones se realizan al finalizar cada sprint, donde se entrega un incremento de funcionalidad para el producto en desarrollo (Versiones).

Generalmente los días viernes, cada negocio versiona la codificación desarrollada durante el sprint. Una vez que se tienen todas las versiones de los equipos involucrados, se realiza la integración de las mismas.

Existe un rol importante en esta actividad al que llamamos integrador, que es el ***encargado*** *de* coordinar todas las tareas involucradas.

El integrador es un miembro de un equipo de desarrollo involucrado en la entrega que se define previamente y va rotando en los sprints siguientes.

Una vez finalizado el proceso, el integrador realiza la instalación de la aplicación en un entorno de pruebas y comunica a todas las áreas de DyD involucradas para que estas realicen los circuitos de prueba que crean necesarios dependiendo de lo desarrollado en el sprint. Si hubo algún error en dichas pruebas, debe ser corregido y se solicita una nueva integración.

Con respecto a la documentación, no se encuentra estandarizada la forma en que se documentan las pruebas. Es decir, depende del negocio/equipo si es que requieren u optan por realizar algún tipo de documentación. Desde nuestra experiencia, participamos en un equipo en el cual en un principio no se realizaba documentación de las pruebas aunque luego surgió la necesidad de realizar una planilla para nombrar las tareas relevantes y/o ítems que se corregían para ir indicando su estado y el responsable de la prueba. De esta manera se optimiza el tiempo de cada desarrollador y no se prueba varias veces el mismo circuito.

Un ejemplo de estas planillas de prueba se puede visualizar en la figura 17. En ella se destacan diversas tipos de pruebas como son las pruebas generales, pruebas de reportes variables, etc. Por cada prueba se puede observar lo que mencionábamos anteriormente, es decir, quién es el responsable de probar, el tiempo de prueba insumido, el resultado de la prueba y también permite indicar algún comentario relacionado a la misma.

Pruebas generales	Quien prueba?	Tiempo de Prueba	Cómo estuvo?	Comentarios / Observaciones	Quien prueba?	Tiempo de Prueba	Cómo estuvo?	Comentarios / Observaciones
	DESARROLLO				INTEGRACIÓN			
Salida de Entidad IR / CMIR / OV / CMP	Seba		OK		Seba		OK	
OV - Accion Agregar Comprobante	Juan		OK		Juan		OK	
OV - Modificar OV Autorizada	Seba		OK		Seba		OK	
IR/CMIR: probar Donaciones	Juan		OK		Juan		OK	
CMP - Búsqueda	Seba		OK		Seba		OK	
CMP - Editor (En modo consulta)	Lea		OK		Lea		OK	
Búsqueda Unificada	Seba		OK		Seba		OK	
EB Tarjeta de Credito	Lea		OK		Lea		OK	
EB Puntos de Venta	Lea		OK		Lea		OK	
EB Donaciones	Seba		OK		Seba		OK	
EB Concepto de Ingreso (grisado/desgrisado de campos)	Mario		OK		Mario		ERROR	Falla Validación
Acumulador Mensual Monotributo	Mario		OK		Mario		OK	
Monotributistas Excluidos	Kike		OK		Kike		ERROR	
EB Talonario	Seba		OK		Seba		OK	
Saldo Disponible - Editor en modo consulta y navegaciones	Mario		ERROR	Probar circuito completo	Mario		OK	
- Reportes Variables - Probar todas las columnas, con y sin leyenda	Quien prueba?	Tiempo de Prueba	Cómo estuvo?	Comentarios /	Quien	Tiempo	Cómo	Comentarios / Observaciones
	DESARROLLO				INTEGRACIÓN			
Reporte Variable IR	Juan		OK		Seba		OK	
Reporte Variable CMIR	Seba		OK		Juan		OK	
Reporte Variable CMP-REC					Juan		OK	
Reporte Variable Unificado					Lea		OK	
Reporte Variable Unificado Resumen	Juan		OK		Seba		OK	
Reporte Variable OV					Lea		OK	
Reporte Variable IR - Probar todas las plantillas	Seba		OK		Juan		OK	
Reporte Variable CMIR - Probar todas las	Juan		OK		Seba		OK	
Reporte Variable Unificado - Probar todas las plantillas	Seba		OK		Seba		OK	

D19.11.0RC1 | D19.10.0RC1 | D19.9.0RC1 | D19.8.0RC1 | D19.7.0RC1 | D19.6.0RC1 | D19.5.0RC1 | D19.4.0RC1 | D19.3.0RC1 | D18.8.3RC1 | D18.8.2RC1 | D19.2.0

Figura 17 - Planilla de pruebas

Despliegue

Esta es la etapa más crítica del proceso, la misma consiste básicamente en instalar el desarrollo realizado a lo largo de todo el período (habitualmente un año) en el entorno de producción, es decir, es el momento en que la funcionalidad comprometida empieza a ser

utilizada por los clientes del sistema.

ARTEFACTOS/ELEMENTOS DE SCRUM

En esta sección se nombran y explican los artefactos de Scrum adoptados y adaptados según las necesidades del proyecto.

Como se dijo anteriormente, la forma en que se trabaja con Scrum fue variando. Primeramente se utilizaban hojas de cálculo para documentar los artefactos de dicha metodología pero luego se decidió actualizar el proceso con la inserción de una herramienta para la gestión de las actividades de desarrollo ágil, llamada AgileFant.

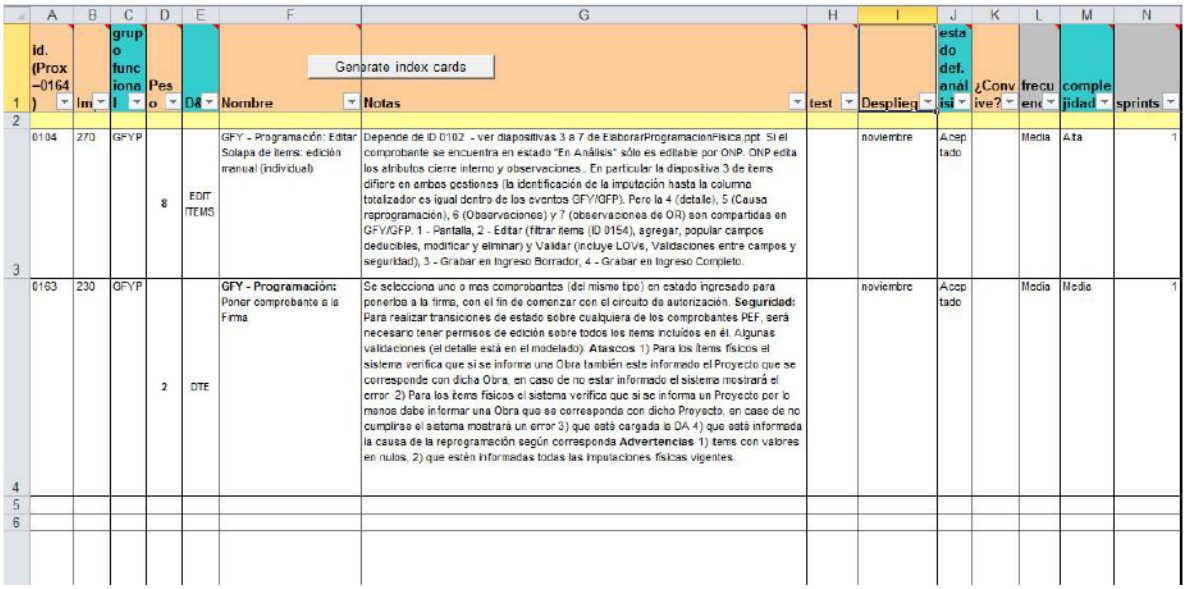
A continuación vamos a detallar como se realizaban los primeros artefactos/elementos utilizados y posteriormente daremos una descripción de la herramienta que se utiliza actualmente en el proyecto, AgileFant.

Descripción de artefactos basados en hojas de cálculo

Product backlog

Es un documento de alto nivel que posee descripciones genéricas de los requerimientos y funcionalidades de porciones del producto que va a desarrollar un equipo. Es decir, que representa lo que va a ser implementado.

A continuación se mostrarán capturas de un ejemplo de product backlog del proyecto. Cabe destacar que se van a visualizar 2 imágenes separadas pero representan a un mismo product backlog. Esto es para una visualización más clara del mismo.



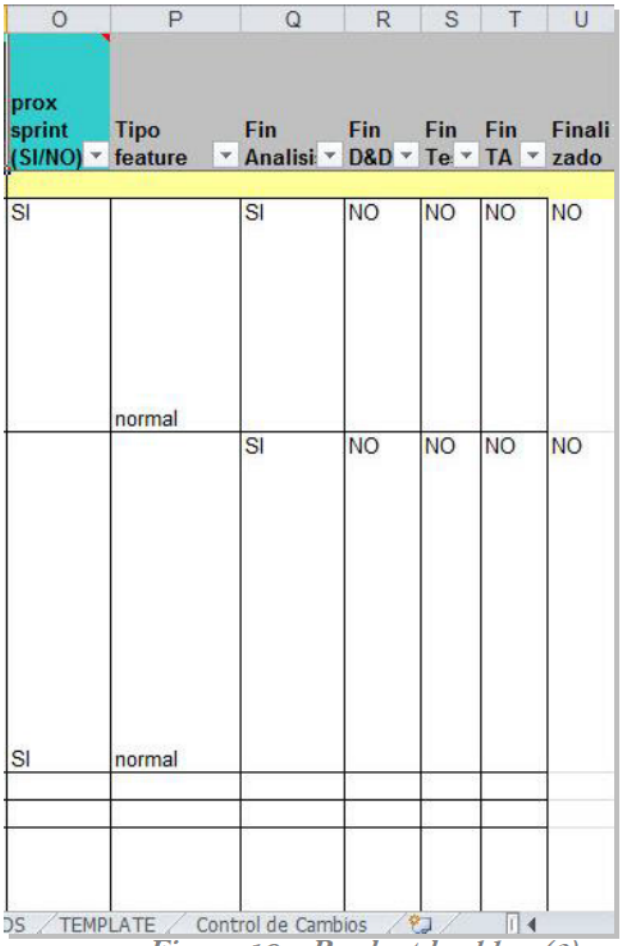
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	id. (Prox –0164)	Im	grupo funcional	Peso	DA	Nombre	Generate index cards Notas	test	Despliegue	estado def. análisi	¿Conv ive?	frecu enc	comple jidad	sprints
2														
3	0104	270	GFYP	8	EDIT ITEMS	GFY - Programación: Editar Solapa de Items: edición manual (individual)	Depende de ID 0102 - ver diapositivas 3 a 7 de ElaborarProgramacionFisica.ppt. Si el comprobante se encuentra en estado "En Análisis" sólo es editable por ONP. ONP edita los atributos cierre interno y observaciones. En particular la diapositiva 3 de ítems difiere en ambas gestiones (la identificación de la imputación hasta la columna totalizador es igual dentro de los eventos GFY/GFP). Pero la 4 (detalle), 5 (Causa reprogramación), 6 (Observaciones) y 7 (observaciones de OR) son compartidas en GFY/GFP. 1 - Pantalla, 2 - Editar (filtrar ítems (ID 0154), agregar, popular campos deducibles, modificar y eliminar) y Validar (incluye LOVs, Validaciones entre campos y seguridad), 3 - Grabar en Ingreso Borrador, 4 - Grabar en Ingreso Completo.		noviembre	Aceptado		Media	Alta	1
4	0163	230	GFYP	2	DTE	**GFY - Programación:** Poner comprobante a la Firma	Se seleccione uno o mas comprobantes (del mismo tipo) en estado ingresado para ponerlos a la firma, con el fin de comenzar con el circuito de autorización. **Seguridad:** Para realizar transiciones de estado sobre cualquiera de los comprobantes PEF, será necesario tener permisos de edición sobre todos los ítems incluidos en él. Algunas validaciones (el detalle está en el modelado): **Atascos** 1) Para los ítems físicos el sistema verifica que si se informa una Obra también este informado el Proyecto que se corresponde con dicha Obra, en caso de no estar informado el sistema mostrará el error 2) Para los ítems físicos el sistema verifica que si se informa un Proyecto por lo menos debe informar una Obra que se corresponda con dicho Proyecto, en caso de no cumplirse el sistema mostrará un error 3) que esté cargada la DA 4) que esté informada la causa de la reprogramación según corresponda **Advertencias** 1) ítems con valores en nulos, 2) que estén informadas todas las imputaciones físicas vigentes.		noviembre	Aceptado		Media	Media	1
5														
6														

Figura 18- Product Backlog(1)

En la *figura 18*, que representa la imagen 1, se visualizan distintas propiedades del product backlog. Entre ellas se encuentran (en orden de izquierda a derecha según las columnas):

- ID del feature[xiv]: esta propiedad sugiere un número a utilizar que representa la identificación del respectivo feature.
- Importancia o valor: es un número que se utiliza sólo para establecer un orden de prioridad. Un mayor número representa una mayor prioridad. Este valor lo establece el product owner.
- Grupo funcional: representa una clasificación funcional, la cual puede ser un código alfabético que representa a un comprobante afectado por el feature, o un código que representa la funcionalidad.
- Peso: representa el tamaño relativo del feature. Este valor lo establece el equipo.
- Clasificación de DyD: puede representar al componente de arquitectura en el que se basa el feature u otra clasificación relevante para DyD.
- Nombre: representa el nombre significativo para feature.
- Notas: propiedad que realiza una descripción que incluye al alcance del feature, notas que pueden ser de utilidad para diseño y además las aclaraciones que se consideren necesarias.
- Test: son consideraciones a tener en cuenta para la prueba. Es decir, cómo mostrar que funciona correctamente.
- Despliegue: representa la fecha de despliegue en la cual se aspira a que esté finalizado el feature.
- Estado def. análisis: representa al estado de la definición de análisis. Este estado puede ser: borrador, propuesto, aceptado.

- Frecuencia: propiedad que se refiere a la frecuencia de uso de la funcionalidad por parte de los usuarios. Es uno de los criterios para determinar la importancia del feature.
- Complejidad: representa a la complejidad del desarrollo del feature. La complejidad puede ser: alta, media y/o baja.
- Sprints: indica en qué sprints se incluye/n el feature.



O	P	Q	R	S	T	U
prox sprint (SI/NO)	Tipo feature	Fin Analisi	Fin D&D	Fin Te	Fin TA	Finali zado
SI	normal	SI	NO	NO	NO	NO
SI	normal	SI	NO	NO	NO	NO

DS / TEMPLATE / Control de Cambios

Figura 19 – Product backlog (2)

Por otra parte, en la *figura 19*, la cual representa a la imagen 2 del product backlog, se visualizan las siguientes propiedades:

- Próximo Sprint: Indica si el feature va a estar incluido o no, en el próximo sprint.
- Tipo de feature: propiedad que representa al tipo de feature. Este puede ser: épica y/o normal.
- Fin Análisis: Indica si el feature está finalizado por análisis.
- Fin DyD: Indica si el feature está finalizado por Diseño y Desarrollo.
- Fin Test: Indica si el feature está finalizado por el área de Testing.
- Fin TA: Indica si el feature está finalizado por área de Testing Automático.
- Finalizado: Indica si el feature está finalizado por todas las áreas.

Sprint backlog

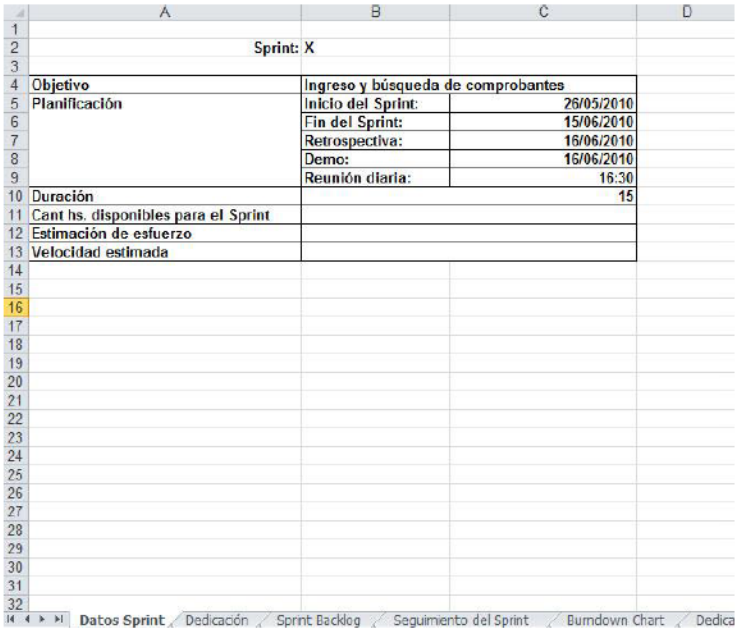
Es un documento detallado en donde se describe cómo el equipo va a implementar los requerimientos durante el sprint. Este documento contiene la estimación detallada en horas que cada equipo realiza durante la planificación inicial del sprint.

Se poseía una hoja de cálculo distinta para cada objetivo. Estos objetivos eran:

- Datos del sprint
- Dedicación
- Sprint Backlog propiamente dicho
- Seguimiento del sprint
- Burndown Chart
- Dedicación chart
- Estadísticas
- Tarjetas
- Control de Cambios

Datos del Sprint

La *figura 20* visualiza los "***datos principales***" del sprint.



	A	B	C	D
1				
2	Sprint: X			
3				
4	Objetivo	Ingreso y búsqueda de comprobantes		
5	Planificación	Inicio del Sprint:	26/05/2010	
6		Fin del Sprint:	15/06/2010	
7		Retrospectiva:	16/06/2010	
8		Demo:	16/06/2010	
9		Reunión diaria:	16:30	
10	Duración		15	
11	Cant hs. disponibles para el Sprint			
12	Estimación de esfuerzo			
13	Velocidad estimada			
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				
32				

Datos Sprint | Dedicación | Sprint Backlog | Seguimiento del Sprint | Burndown Chart | Dedica

Figura 20 – Datos principales del Sprint

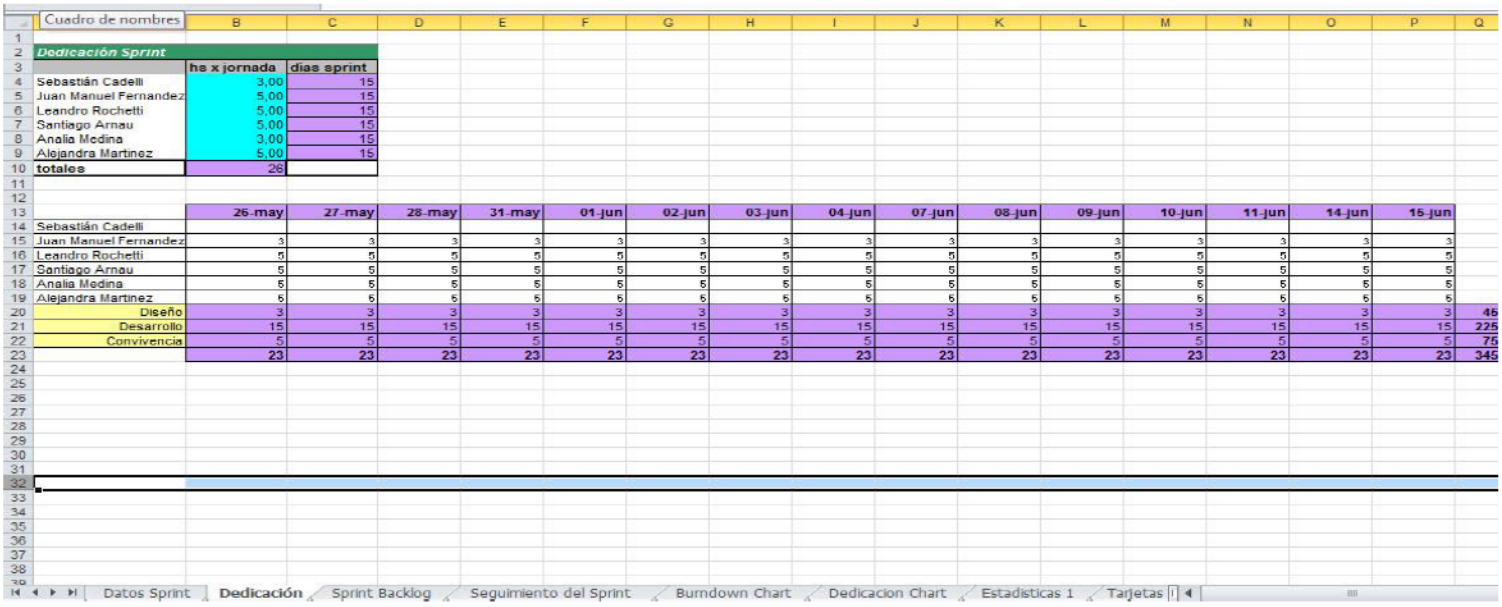
Esta hoja posee propiedades básicas con respecto al sprint. Ellas son las siguientes:

- Número del sprint: indica el número correspondiente al sprint.
- Objetivo: es una descripción del objetivo a cumplir para dicho sprint.
- Planificación: se proponen diversas cuestiones relacionadas con las fechas. Desde la fecha de inicio de sprint, la fecha de finalización, la fecha de la reunión diaria(daily), la fecha de la retrospectiva.
- Duración: esta propiedad indica la duración en horas del sprint.
- Cantidad de Hs. disponibles para el Sprint: indica la cantidad de horas que se disponen para llevar a cabo el sprint.
- Estimación del esfuerzo
- Velocidad estimada

Dedicación

La dedicación se refiere al tiempo (en horas) que un integrante del equipo le dedica a una determinada tarea.

En la *figura 21*, se visualiza por cada integrante del equipo la dedicación por día que realiza en el correspondiente sprint.



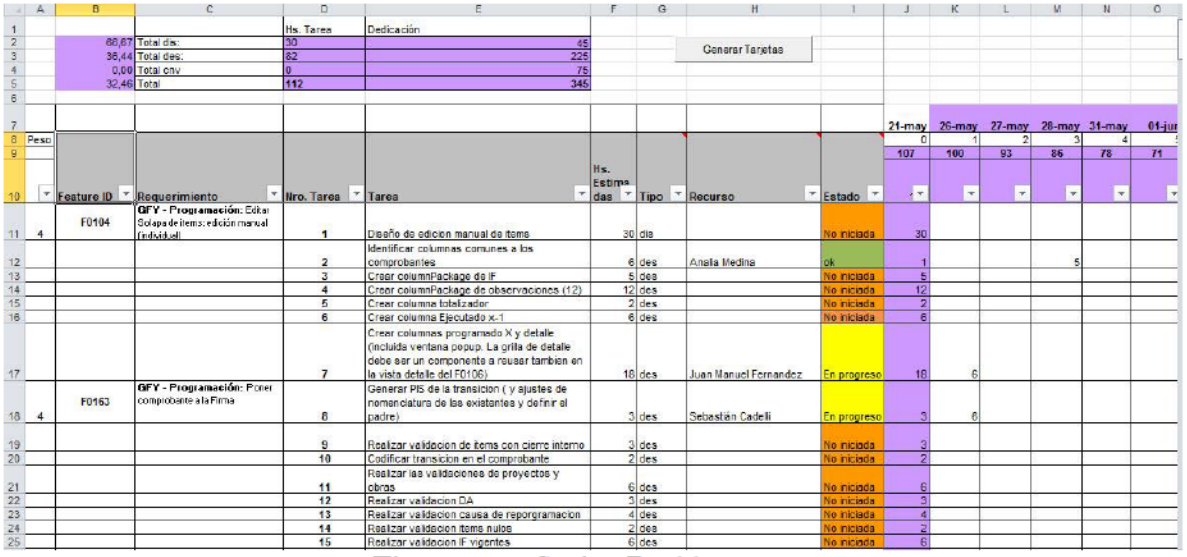
Dedicación Sprint	hs x jornada	dias sprint
Sebastián Cadelli	3,00	15
Juan Manuel Fernandez	5,00	15
Leandro Rochetti	5,00	15
Santiago Arnau	5,00	15
Analia Medina	3,00	15
Alejandra Martinez	5,00	15
totales	26	

	26-may	27-may	28-may	31-may	01-jun	02-jun	03-jun	04-jun	07-jun	08-jun	09-jun	10-jun	11-jun	14-jun	15-jun	
Sebastián Cadelli																
Juan Manuel Fernandez	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
Leandro Rochetti	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
Santiago Arnau	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
Analia Medina	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
Alejandra Martinez	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
Diseño	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	45
Desarrollo	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	225
Convivencia	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	75
	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	345

Datos Sprint | Dedicación | Sprint Backlog | Seguimiento del Sprint | Burndown Chart | Dedicacion Chart | Estadisticas 1 | Tarjetas

Figura 21 – Dedicación del equipo

La *figura 22* visualiza el "***sprint backlog***" con sus propiedades.



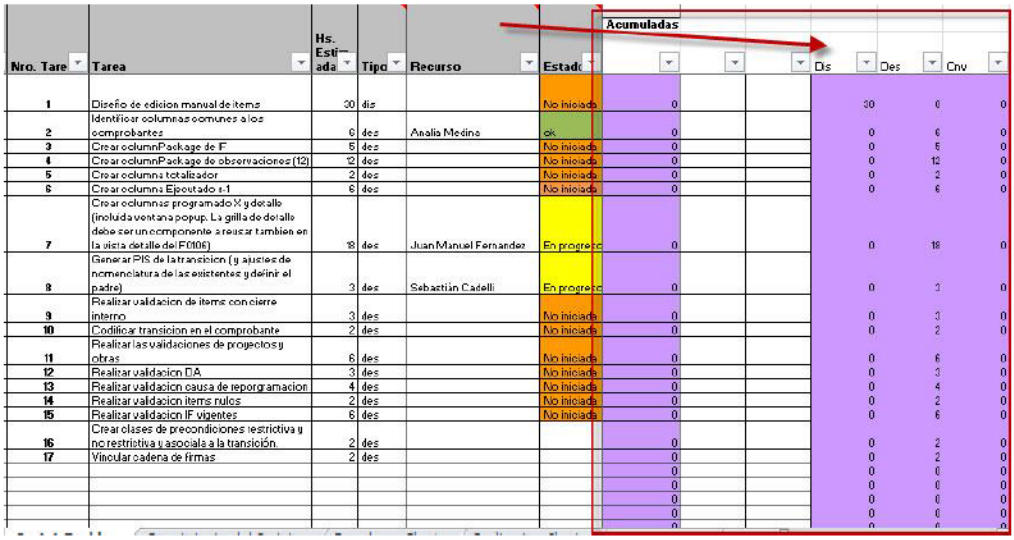
				Hs. Tarea	Dedicación							21-may	26-may	27-may	28-may	31-may	01-jun
	66,67	Total dis:		30	45			Generar Tarjetas				0	1	2	3	4	5
	36,44	Total des:		82	225							107	100	93	86	78	71
	0,00	Total cnv		0	75												
	32,46	Total		112	345												
Peso		Feature ID	Requerimiento	Nro. Tarea	Tarea	Hs. Estimadas	Tipo	Recurso	Estado								
4		F0104	GFY - Programación: Edicion Solapa de Items: edición manual (individual)	1	Diseño de edicion manual de items	30	dia		No iniciada	30							
				2	Identificar columnas comunes a los comprobantes	6	des	Analia Medina	ok	1					5		
				3	Crear columnPackage de IF	5	des		No iniciada	5							
				4	Crear columnPackage de observaciones (12)	12	des		No iniciada	12							
				5	Crear columna totalizador	2	des		No iniciada	2							
				6	Crear columna Ejecutado x-1	6	des		No iniciada	6							
				7	Crear columnas programado X y detalle (incluida ventana popup. La grilla de detalle debe ser un componente a reusar tambien en la vista detalle del F0106)	18	des	Juan Manuel Fernandez	En progreso	18	6						
4		F0163	GFY - Programación: Poner comprobante a la Firma	8	Generar PIS de la transicion (y ajustes de nomenclatura de las existantes y definir al padre)	3	des	Sebastián Cadelli	En progreso	3	6						
				9	Realizar validacion de items con cierre interno	3	des		No iniciada	3							
				10	Codificar transicion en el comprobante	2	des		No iniciada	2							
				11	Realizar las validaciones de proyectos y obras	6	des		No iniciada	6							
				12	Realizar validacion DA	3	des		No iniciada	3							
				13	Realizar validacion causa de reporgramacion	4	des		No iniciada	4							
				14	Realizar validacion items nulos	2	des		No iniciada	2							
				15	Realizar validacion IF vigentes	6	des		No iniciada	6							

Figura 22 – Sprint Backlog

El sprint backlog contiene diversas propiedades, que son las siguientes. Comenzando por las columnas desde izquierda a derecha se presentan:

- *Peso*: es la importancia que posee una tarea.
- *Feature ID*: es una identificación unívoca que se le asigna a la tarea.
- *Requerimiento*: Indica el requerimiento que está asociado para que se tenga que realizar el feature.
- *Nro. de tarea*: representa el número de tarea a realizar.
- *Tarea*: es una descripción de lo que se requiere realizar en dicha tarea.
- *Hs. Estimadas*: propiedad que hace referencia a la cantidad de horas que se le asignó al feature durante la planificación.
- *Tipo*: indica el tipo de tarea. En este caso "des" representa a una tarea de desarrollo. Pero puede ser de "conv" que es el caso de un tipo de tarea de convivencia y/o "dis" que representa a una tarea de diseño.
- *Recurso*: representa a la persona que está a cargo de realizar dicha tarea.
- *Estado*: se refiere al estado en que se encuentra la tarea. Esta puede ser: No iniciada, en progreso, ok (que ya se finalizó), suspendida, y/o Completada en revisión.
- Las columnas que continúan hacia la derecha de la columna "estado" representan los días incluidos en el sprint. En cada día se van registrando los esfuerzos que cada recurso realiza para realizar cada tarea, es decir, se indica la cantidad de horas insumidas para realizar dicha tarea por cada integrante del equipo.
- Al finalizar la tabla se encuentran la cantidad de horas acumuladas por dicha tarea. Esto se visualiza en la figura 23

Además se puede indicar la cantidad de horas acumuladas de acuerdo al tipo de tarea que representa, o si esta tarea requiere de más de un tipo, se indica la cantidad de horas acumuladas por cada tipo de tarea.



Nro. Tare	Tarea	Hs. Estimada	Tipo	Recurso	Estado	Acumuladas			Dis	Des	Cnv
1	Diseño de edicion manual de items	30	dis		No iniciado	0			30	0	0
2	Identificar columnas comunes a los comprobantes	6	des	Analia Medina	ok	0			0	6	0
3	Crear columnPackage de IF	6	des		No iniciado	0			0	6	0
4	Crear columnPackage de observaciones (12)	12	des		No iniciado	0			0	12	0
5	Crear columna totalizador	2	des		No iniciado	0			0	2	0
6	Crear columna Ejecutado s-1	6	des		No iniciado	0			0	6	0
7	Crear columnas programado X y detalle (incluida ventana popup. La grilla de detalle debe ser un componente a reusar tambien en la vista detalle del F0106)	18	des	Juan Manuel Fernandez	En progreso	0			0	18	0
8	Generar PIS de la transicion (y ajustes de nomenclatura de las existentes y definir el padre)	3	des	Sebastián Cadelli	En progreso	0			0	3	0
9	Realizar validacion de items con cierre interno	3	des		No iniciado	0			0	3	0
10	Codificar transicion en el comprobante	2	des		No iniciado	0			0	2	0
11	Realizar las validaciones de proyectos y obras	6	des		No iniciado	0			0	6	0
12	Realizar validacion DA	3	des		No iniciado	0			0	3	0
13	Realizar validacion causa de reporgramacion	4	des		No iniciado	0			0	4	0
14	Realizar validacion items rulos	2	des		No iniciado	0			0	2	0
15	Realizar validacion IF vigentes	6	des		No iniciado	0			0	6	0
16	Crear clases de precondiciones restrictiva y no restrictiva y asociala a la transición	2	des			0			0	2	0
17	Vincular cadena de firmas	2	des			0			0	2	0
						0			0	0	0
						0			0	0	0
						0			0	0	0
						0			0	0	0

Figura 23- Sprint Backlog(continuación)

Burndown Chart

Es un gráfico que da una visión "general" del progreso del proyecto. Permite al equipo visualizar la velocidad de avance del proyecto. Este gráfico se va actualizando a medida que se van cumpliendo y realizando las tareas correspondientes al product backlog. Se puede visualizar en la *figura 24*.

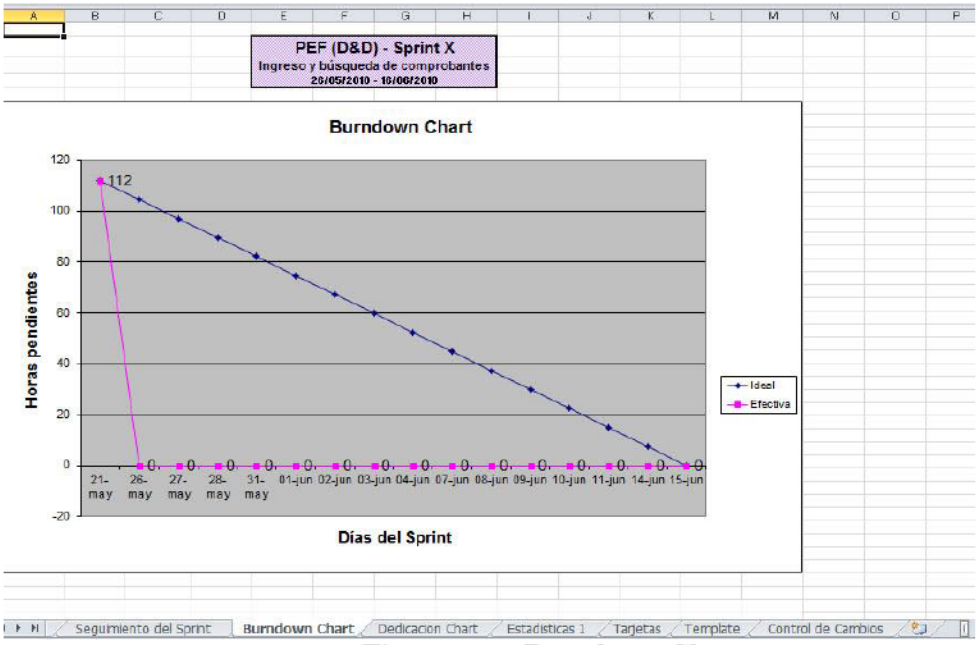


Figura 24 – Burndown Chart

Dedicación chart

La *figura 25* visualiza el gráfico correspondiente a la "***dedicación chart***". El mismo realiza una gráfica de la dedicación realizada por el equipo desde el aspecto en cuanto fue efectiva, cuanto se avanzó, cuanto queda disponible y cuanto se requiere.

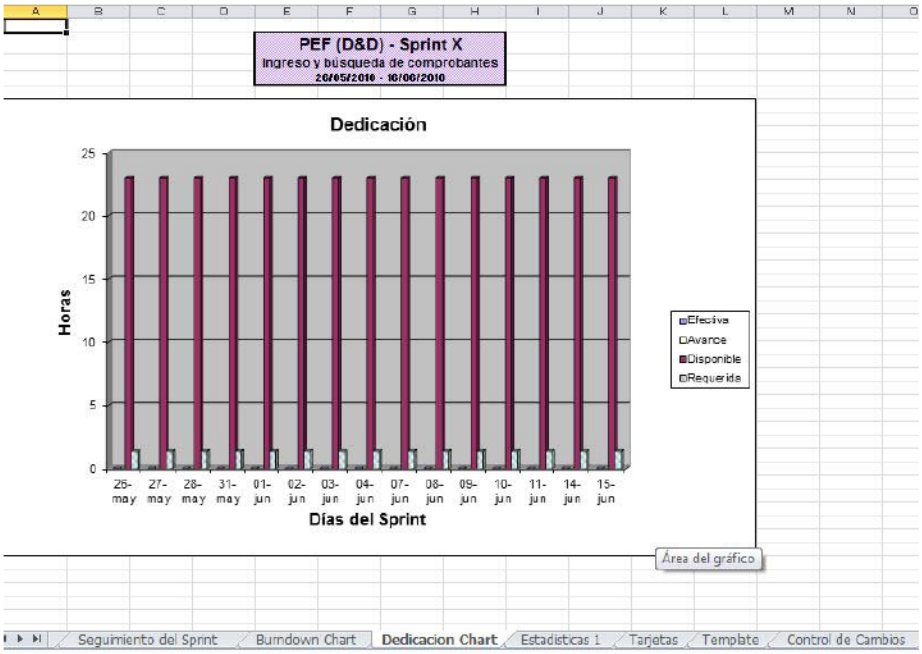
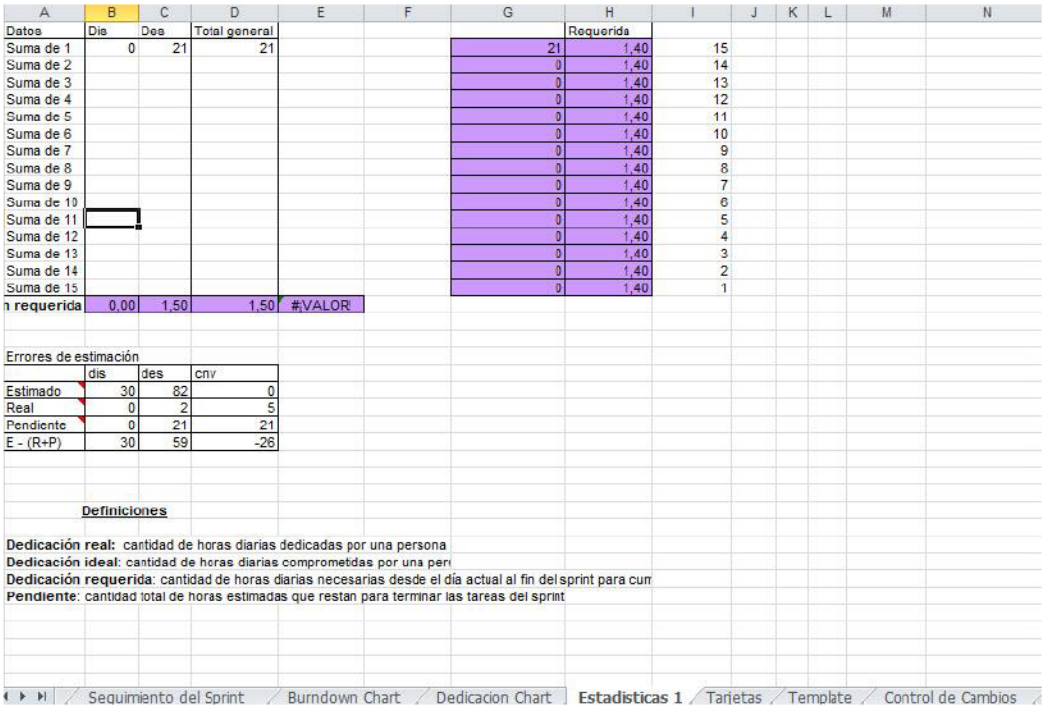


Figura 21 – Dedicación chart

La *figura 26* visualiza "***estadísticas***" correspondientes a las tareas y también los errores de estimación producidos.



Datos	Dia	Dea	Total general
Suma de 1	0	21	21
Suma de 2			
Suma de 3			
Suma de 4			
Suma de 5			
Suma de 6			
Suma de 7			
Suma de 8			
Suma de 9			
Suma de 10			
Suma de 11			
Suma de 12			
Suma de 13			
Suma de 14			
Suma de 15			
n requerida	0,00	1,50	1,50

	Requerida	
21	1,40	15
0	1,40	14
0	1,40	13
0	1,40	12
0	1,40	11
0	1,40	10
0	1,40	9
0	1,40	8
0	1,40	7
0	1,40	6
0	1,40	5
0	1,40	4
0	1,40	3
0	1,40	2
0	1,40	1

Errores de estimación

	dis	des	cnv
Estimado	30	82	0
Real	0	2	5
Pendiente	0	21	21
E - (R+P)	30	59	-26

Definiciones

Dedicación real: cantidad de horas diarias dedicadas por una persona
Dedicación ideal: cantidad de horas diarias comprometidas por una per
Dedicación requerida: cantidad de horas diarias necesarias desde el día actual al fin del sprint para cum
Pendiente: cantidad total de horas estimadas que restan para terminar las tareas del sprint

Seguimiento del Sprint | Burndown Chart | Dedicacion Chart | Estadisticas 1 | Tarjetas | Template | Control de Cambios

Figura 22 – Estadísticas

La *figura 27*, visualiza algunas "***tarjetas***". Estas tarjetas representan a las tareas que se encuentran en el sprint backlog. Poseen su número de tarea, así como también sus horas estimadas, una descripción de lo que se va a realizar en la misma, y el tipo de tarea.

Las tarjetas se imprimen y se colocaban en una pizarra en la cual se dividen zonas que corresponden al estado de la tarea, como iniciada, terminada, realizando, o Pendiente.

Una vez que alguien toma una tarea se le debea asignar el nombre de la persona que la eligió para poder llevar un registro de la misma.

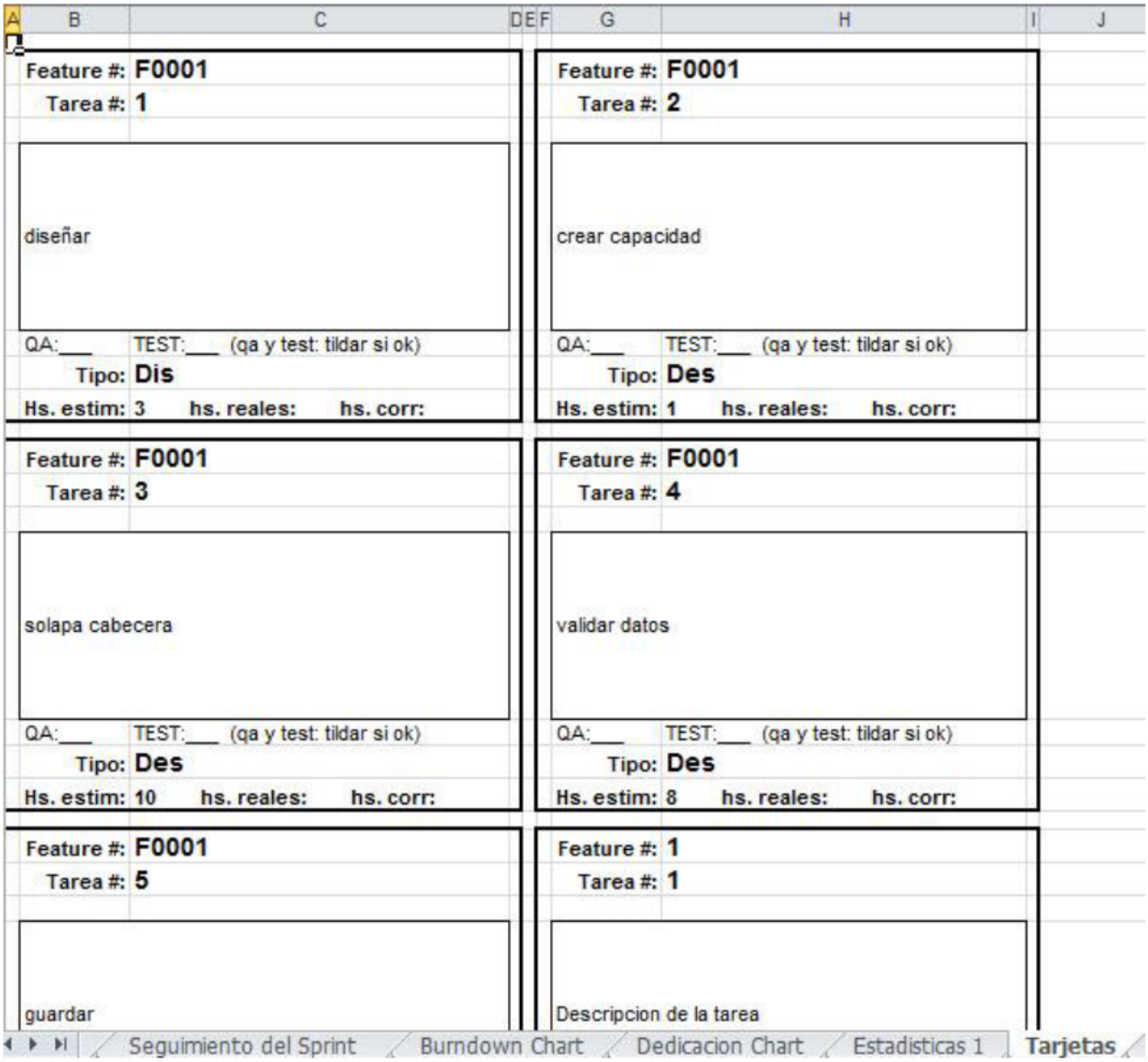
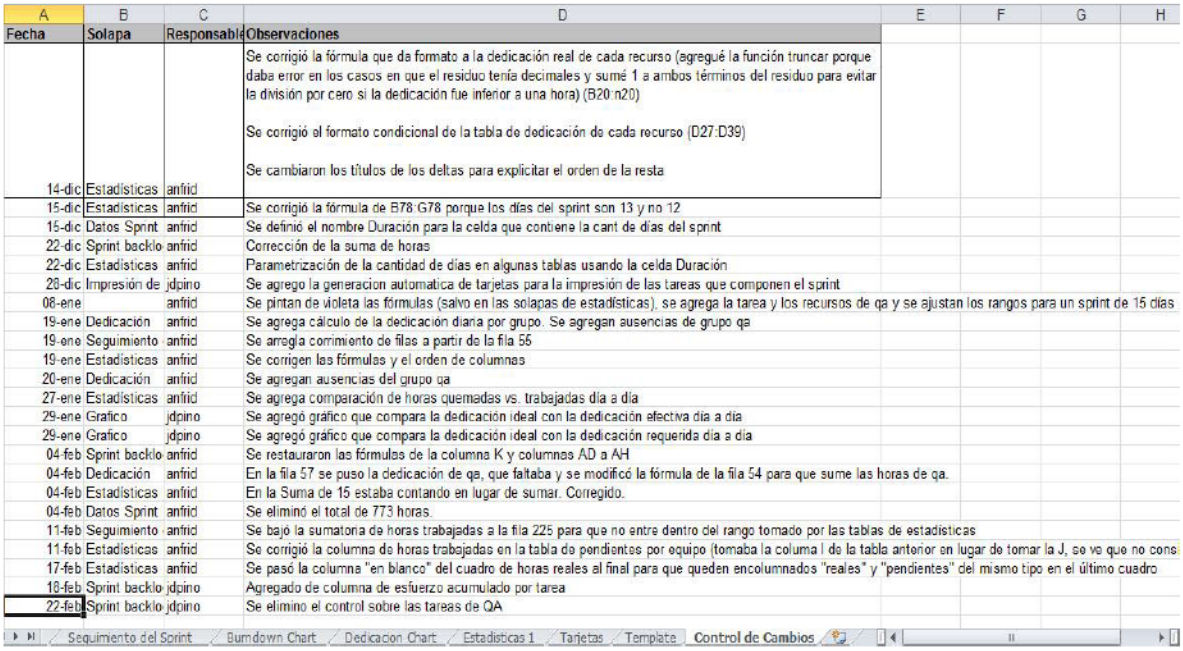


Figura 27 – Tarjetas

Por último el "***control de cambios***", (el cual se visualiza en la *figura 28)* se utiliza para llevar un control de cada cambio que se realiza en las hojas Excel. Esto es a través de una fecha de cambio, la solapa que participa del mismo, es decir, la solapa se refiere a alguna de las figuras anteriormente mencionadas (product backlog, dedicación, Gráficos, Sprint backlog, etc), se indica un responsable del cambio, y por último las observaciones que son una descripción del cambio realizado.



Fecha	Solapa	Responsable	Observaciones
14-dic	Estadísticas	anfrid	Se corrigió la fórmula que da formato a la dedicación real de cada recurso (agregué la función truncar porque daba error en los casos en que el residuo tenía decimales y sumé 1 a ambos términos del residuo para evitar la división por cero si la dedicación fue inferior a una hora) (B20:n20) Se corrigió el formato condicional de la tabla de dedicación de cada recurso (D27:D39) Se cambiaron los títulos de los deltas para explicitar el orden de la resta
15-dic	Estadísticas	anfrid	Se corrigió la fórmula de B78:G78 porque los días del sprint son 13 y no 12
15-dic	Datos Sprint	anfrid	Se definió el nombre Duración para la celda que contiene la cant de días del sprint
22-dic	Sprint backlo	anfrid	Corrección de la suma de horas
22-dic	Estadísticas	anfrid	Parametrización de la cantidad de días en algunas tablas usando la celda Duración
28-dic	Impresión de	jdpino	Se agrego la generacion automatica de tarjetas para la impresión de las tareas que componen el sprint
08-ene		anfrid	Se pintan de violeta las fórmulas (salvo en las solapas de estadísticas), se agrega la tarea y los recursos de qa y se ajustan los rangos para un sprint de 15 días
19-ene	Dedicación	anfrid	Se agrega cálculo de la dedicación diaria por grupo. Se agregan ausencias de grupo qa
19-ene	Seguimiento	anfrid	Se arregla corrimiento de filas a partir de la fila 55
19-ene	Estadísticas	anfrid	Se corrigen las fórmulas y el orden de columnas
20-ene	Dedicación	anfrid	Se agregan ausencias del grupo qa
27-ene	Estadísticas	anfrid	Se agrega comparación de horas quemadas vs. trabajadas día a día
29-ene	Grafico	jdpino	Se agregó gráfico que compara la dedicación ideal con la dedicación efectiva día a día
29-ene	Grafico	jdpino	Se agregó gráfico que compara la dedicación ideal con la dedicación requerida día a día
04-feb	Sprint backlo	anfrid	Se restauraron las fórmulas de la columna K y columnas AD a AH
04-feb	Dedicación	anfrid	En la fila 57 se puso la dedicación de qa, que faltaba y se modificó la fórmula de la fila 54 para que sume las horas de qa.
04-feb	Estadísticas	anfrid	En la Suma de 15 estaba contando en lugar de sumar. Corregido.
04-feb	Datos Sprint	anfrid	Se eliminó el total de 773 horas.
11-feb	Seguimiento	anfrid	Se bajó la sumatoria de horas trabajadas a la fila 225 para que no entre dentro del rango tomado por las tablas de estadísticas
11-feb	Estadísticas	anfrid	Se corrigió la columna de horas trabajadas en la tabla de pendientes por equipo (tomaba la columa I de la tabla anterior en lugar de tomar la J, se ve que no cons
17-feb	Estadísticas	anfrid	Se pasó la columna "en blanco" del cuadro de horas reales al final para que queden encolumnados "reales" y "pendientes" del mismo tipo en el último cuadro
18-feb	Sprint backlo	jdpino	Agregado de columna de esfuerzo acumulado por tarea
22-feb	Sprint backlo	jdpino	Se elimino el control sobre las tareas de QA

Seguimiento del Sprint | Burndown Chart | Dedicacion Chart | Estadisticas 1 | Tarjetas | Template | Control de Cambios

Figura 23-Control de cambios

Como se mencionó previamente, la forma en la cual se trabajaba con las hojas de cálculo se abandonó y actualmente se utiliza una herramienta para el manejo del proyecto. Esta herramienta es el *Agilefant*, la cual detallaremos a continuación.

5.1.1 Agilefant

El agilefant es una herramienta openSource [xv] para la gestión de actividades y/o tareas en lo que respecta al desarrollo de software utilizando una metodología ágil.

Permite:

- manejo de proyectos
- manejo de grupos
- manejo de product backlog
- manejo de recursos
- manejo de iteraciones y retrasos

Facilita la planificación, la organización interna y permite llevar un seguimiento mucho más detallado y optimizado del proyecto. Anteriormente cada miembro de un equipo tenía que indicar a su coordinador de equipo el esfuerzo que se realizaba diariamente al finalizar el día. Esto, además de generar una pérdida de tiempo para el coordinador, generaba una espera para cada integrante, es decir, una espera innecesaria la cual se podría utilizar más eficientemente. Tanto para el coordinador como para el desarrollador.

Con el AgileFant cada desarrollador tiene mayor autonomía durante el Sprint y a su vez el coordinador del equipo puede llevar un seguimiento más detallado del mismo. Así como también de una manera más automática que la de generar las tarjetas, imprimirlas y colocarlas en la pizarra.

Funcionamiento del Agilefant

Primeramente para entrar en la herramienta se necesita realizar un inicio de sesión a través de un nombre de usuario y una contraseña, el cual permitirá llevar el registro de la persona.

Luego de iniciar la sesión, el usuario se encuentra en condiciones de poder acceder a la herramienta y visualizar el/los product backlog del proyecto en los cuales se encuentra incluido.

A continuación se visualizarán algunas capturas a modo de ejemplo, correspondientes a un product backlog del proyecto.

La figura 29 visualiza la pantalla inicial luego del inicio de sesión en el agilefant. Entre otras cosas se visualiza un gráfico de barras que genera la herramienta, y que significa la relación entre el esfuerzo realizado por el usuario y en cada día. Por otra parte en la parte superior izquierda se visualizan los product backlog a los cuales está asignado el usuario. Y por último se visualizan las stories a las que el usuario está asignado y se encuentran en estado “started”.

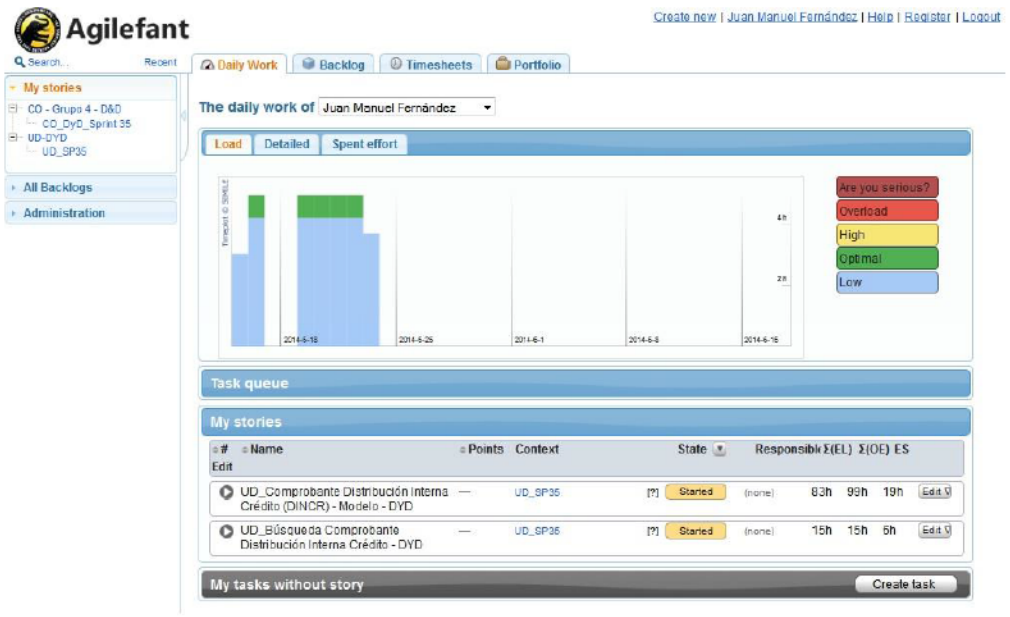


Figura 29 – Agilefant

En la esquina superior izquierda de la figura 29 aparecen dos menús que son:

- My Stories
- All Backlog

Para visualizarlo mejor se puede observar la figura 30.

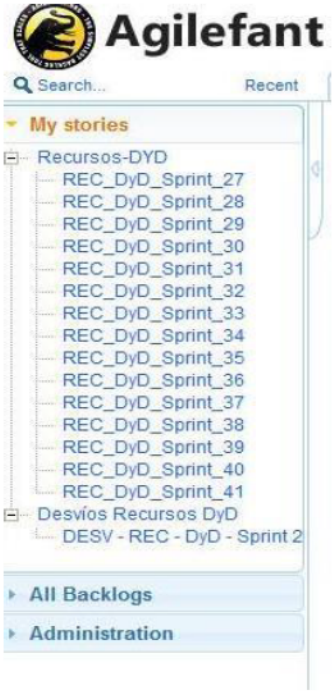


Figura 30- Menús

El primer menú denominado "My Stories" indica en todas las stories en las cuales un usuario del agilefant se encuentra asignado.

El segundo menú, denominado "All Backlog", muestra una lista desplegable de todos los product backlog que se encuentran creados en el proyecto.

La *figura 31* visualiza las características generales del product backlog seleccionado (en el caso del ejemplo el product backlog seleccionado es "Ecom DyD Sprint 2")

Figura 31 – Características product backlog

En la misma se visualizan aspectos generales del sprint, como son:

- El nombre.
- Un Id de referencia.
- Una fecha de inicio.
- Una fecha de finalización.
- La cantidad de horas totales de la planificación de dicho sprint.
- Las personas asignadas al sprint.
- Una descripción con las tareas más relevantes a realizar.
- Al lado derecho de la figura 31 también se visualizan estadísticas tales como:
  - Cantidad de días que llevara el sprint.
  - Los días que faltan para finalizar el sprint.
  - Desvíos, que hacen referencia al tiempo en que una persona puede tener algún inconveniente con la realización de una tarea. Este parámetro en particular, hace referencia a la suma de todos esos tiempos en los cuales hubo algún percance.
  - Horas de esfuerzo dedicadas hasta el momento.
  - Estimación original del sprint en horas (suma de todas las horas de todas las tareas).
  - Cantidad de Stories en estado Done.
  - Cantidad de Stories en estado Ready.
  - Cantidad de Tareas en estado Done.
  - Cantidad de Tareas en estado Ready.
  - También se muestran los Story points, es decir, la suma total de los puntos de todas las stories.

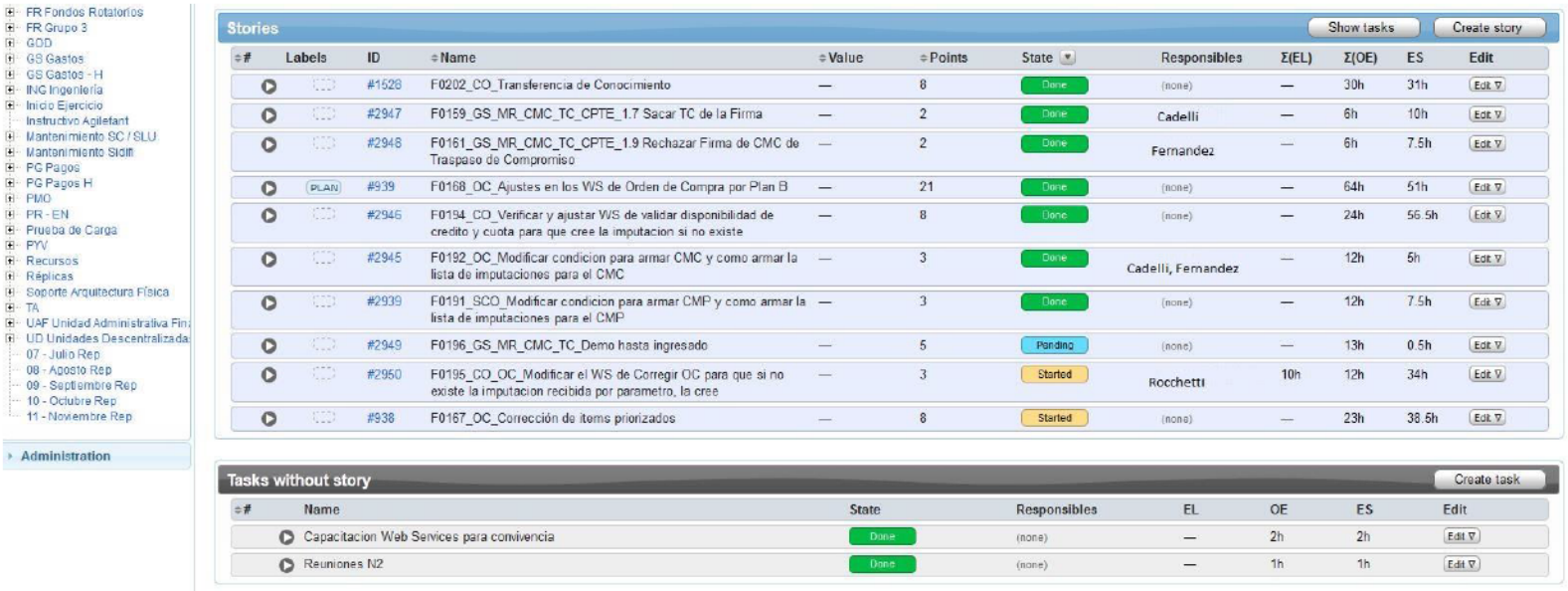


Figura 32 – Ejemplo de task y stories en product backlog

El product backlog tiene todas las tareas a realizar en el sprint. Las tareas son agrupadas en Stories dependiendo su clasificación.

Por ejemplo, un Story con el nombre "Wizard Inicial Comprobante" y dentro de este se encuentran todas las tareas (task) a desarrollar para completar dicha story.

El estado de las Stories depende de sus tareas, es decir, cuando alguna de estas se encuentra en estado **Started**, entonces el estado de la Story que la contiene también es **Started**. Una vez que todas las tareas de una Story se encuentran en estado **Done**, esta toma el mismo estado y se da por finalizada.

Cada Story contiene:

- ID: es un identificador unívoco de la story.
- Name: es una descripción que representa un nombre representativo de la story a realizar.
- Value: es un valor que se le da a una story indicando una prioridad dentro del sprint. El valor más bajo (1), indica que todas las tareas que componen esta story deben ser desarrolladas con mayor prioridad que las tareas que se encuentran en stories con valores mayores a 1.
- Points: los story points indican el tamaño relativo y la complejidad de cada story con relación a otro story (se estima el tamaño y no la duración). Determinar el valor de cada story es subjetivo, pero permiten estimar el esfuerzo sin tratar de estimar el tiempo que consumirá. De esta manera un story podría definirse como:
  Story = esfuerzo + complejidad + riesgo

Por otra parte para calcular el valor del story pueden elegirse distintas herramientas, como por ejemplo, la serie de Fibonacci.

- State: es un estado en el cual se encuentra la story. Depende del estado de las tareas que componen dicha story.
  El estado puede ser:
  - Not started: no iniciada.
  - Started: en progreso.
  - Pending: suspendida provisoriamente por causas internas.
  - Deferred: aplazada, significa que no se finalizará en el sprint en curso.
  - Done: terminada.
  - Ready: lista.
  - Blocked: impedida.

- Reponsibles: se refiere al/los responsable/s de llevar a cabo la tarea.
- EL (EFFORT LEFT): Es el tiempo en el cual se calcula la diferencia entre las horas dedicadas a la tarea (EL) y la estimación original (OE) de la misma. Es decir, es un indicador del esfuerzo que faltaría para completar la tarea.
- OE (ORIGINAL ESTIMATE): Es la estimación original en horas que se le asigna a la tarea, es decir, el que se estima durante la planning.
- ES (SPENT EFFORT): Es el esfuerzo que cada responsable de una tarea le asigna a la misma. Esto es, la cantidad de horas que una persona realiza para una determinada tarea. Cada persona tiene que indicar por día, el esfuerzo que le requiere la/las tarea/s de la/las cual/les es responsable.

El AgileFant también permite la posibilidad de generar un gráfico de Burndown como se visualiza en la *figura 33*.

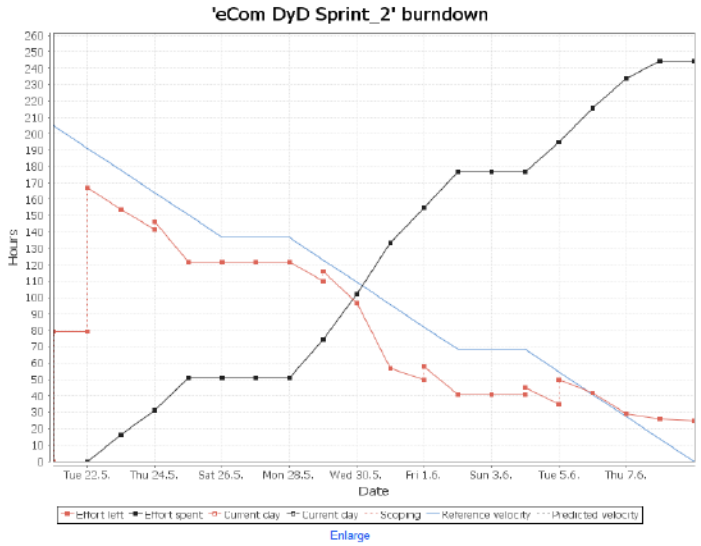


Figura 33 – Gráfico Burndown generado por agilefant

5.1.2 ¿Por qué se eligió Agilefant?

A continuación se presentan una serie de tablas comparativas de 4 herramientas para la gestión de Scrum. Estas tablas comparan diversas características relacionadas a distintos aspectos a tener en cuenta a la hora de elegir y utilizar una herramienta. Estas herramientas son:

- ✓ Xplanner
- ✓ Kunagi
- ✓ Versionone
- ✓ Agilefant

Y los aspectos a evaluar son:

- Product Backlog, Sprint Planning, Comunicación, Seguridad, Auditoría, Reportes, Seguimiento, Integración con herramientas, Precio, Plataforma/Tecnología, Estabilidad, Usabilidad, Interacción con otras herramientas, Documentación, Personalización.

A tener cuenta:

- √: Indica un SI
- **x:** Indica un NO

CARACTERISTICAS PRODUCT BACKLOG	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Multiproducto	**X**	√	√	√
Fácil de editar y priorizar	**X**	√	**X**	√ *(inline,Drag and Drop y tradicional)*
Despliegues	**X**	**X**	**X**	√ *(Por producto)*
Jerarquía de Producto – Despliegue- Sprint	**X**	**X**	**X**	√

Tabla 3 – Comparación de características del product backlog

CARACTERISTICAS SPRINT PLANNING	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Facilidad de (re)asignar stories a sprints	**X**	**X**	**X**	√
Planning poker	**X**	√	**X**	X
Facilidad de editar y priorizar	**X**	**X**	**X**	√
Jerarquía user stories	**X**	**X**	**X**	√

user story ➔tarea	X	X	X	√
user story splitting	X	√	X	√
Manejo de dependencias	X	X	√	X
Relación user story➔bugs, impedimentos	X	X	√	X
Relación task con código	X	X	X	X
Hs.Pendientes	X	X	X	√
Hs.Insumidas	X	X	X	√
Estimación Original	X	X	X	√
Info. adicional Story	X	X	X	√ *(Hipervinculo)*
Carga hacia atrás	X	X	X	√ *(sólo de effort, no de pending)*
Manejo de tiempo estándar dedicado a reuniones grupales (baseline load)	X	X	X	√ *(sí, pero igual para todos los equipos)*
Manejo de feriados	X	X	X	√ *(workaround: task sin stories)*

Tabla 4 – Comparación de características de la planificación

CARACTERISTICAS COMUNICACIÓN	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Facilidades comunicación grupal	**X**	**X**	**X**	**X**
Registro de retrospectivas	**X**	*Comments a las stories*	**X**	**X**

Tabla 5 – Comparación de características de comunicación

CARACTERISTICAS SEGURIDAD	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Acceso diferenciado a la funcionalidad	**X**	**X**	**X**	**X**
Acceso diferenciado a datos	**X**	**X**	**X**	**X**

Tabla 6 – Comparación de características de seguridad

CARACTERISTICAS AUDITORIA	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Historial de cambios	**X**	**X**	**X**	√ *(Sólo de creación y de borrado)*

Tabla 7 – Comparación de características de auditoría

CARACTERISTICAS REPORTES	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Burndown chart	**X**	**X**	**X**	√
Product Burndown chart	**X**	**X**	**X**	√ *(Burn up)*
Reportes de testing	**X**	**X**	**X**	**X**

Tabla 8 - Comparación de características de reportes

CARACTERISTICAS SEGUIMIENTO	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Seguimiento de impedimentos	**X**	√	**X**	**X**

Tabla 9- Comparación de características de seguimiento

CARACTERISTICAS INTEGRACIÓN CON HERRAMIENTAS	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Posibilidad de integrar con ITEM	**X**	**X**	**X**	√
Posibilidad de integrar con cvs/toolkit	**X**	**X**	**X**	√

Tabla 10 - Comparación de características de integración

CARACTERISTICAS PRECIO	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Precio/licencia	*Free(libre)*	*Free(libre)*	*Cara*	*Free(libre)*

Tabla 11 - Comparación de precios

CARACTERISTICAS *PLATAFORMA/TECNOLOGÍA*	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Lenguaje				*java*
Persistencia		*xml*		*mySql*
Otros				*tomcat*

Tabla 12 - Comparación de características de compatibilidad

CARACTERISTICAS *ESTABILIDAD*	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Estabilidad	*mala*			√

Tabla 13 - Comparación de estabilidad

CARACTERISTICAS *USABILIDAD*	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Usabilidad	*Deficiencia en la traducción*			√

Tabla 14 - Comparación de usabilidad

CARACTERISTICAS *INTERACCIÓN CON OTRAS HERRAMIENTAS*	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Interacción excell	**X**	**X**	**X**	√ *(spent effort)*
Interacción project	**X**	**X**	**X**	√
Interacción eclipse	**X**	**X**	**X**	√

Tabla 15 – Comparación de interacción con herramientas

CARACTERISTICAS DOCUMENTACIÓN	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Guía del usuario	*Mínima*			*Mínima*
Soporte				*Rápido y actualizado*
Código fuente				*Completo*

Tabla 16 – Comparación de documentación

CARACTERISTICAS PERSONALIZACIÓN	HERRAMIENTAS			
	Xplanner	***Kunagi***	***Versionone***	***Agilefant***
Flexibilidad				*labels*
Se adapta a nuestros requerimientos				√

Tabla 17 - Comparación de personalización

A partir de las comparaciones relacionadas con las distintas características soportadas o no por las herramientas, y de las necesidades del proyecto eSidif, se tomó la decisión (mediante el área de Ingeniería) que para la gestión de Scrum correspondiente al mismo, la mejor alternativa es la del AgileFant.

5.2 Convivencia de Scrum con Rup

Como dijimos anteriormente en otros párrafos, la metodología que se utiliza en el proyecto eSidif es una convivencia de metodología ágil, en este caso Scrum, con una metodología tradicional como Rup.

Por un lado, Scrum se utiliza para gestionar los procesos del proyecto. Esta metodología se utiliza para definir un marco de trabajo para la gestión y el desarrollo de software basándose en un proceso iterativo e incremental. En nuestro caso, la experiencia a partir del uso de esta metodología de desarrollo produjo óptimos resultados, ya que, en proyectos de gran tamaño como es el eSidif, los cambios de requerimientos son bastante frecuentes, y de esta forma adoptando esta metodología ágil, adaptarse a los cambios y/o prioridades es relativamente accesible. Otros aspectos ventajosos de la utilización de Scrum, es la participación de todo el equipo en la toma de las decisiones más significativas del proceso, por ejemplo, en la estimación de las tareas a desarrollar durante el sprint.

Otra de las ventajas es la posibilidad que brinda a los desarrolladores para que puedan adquirir un amplio y diverso conocimiento a través de la rotación de tareas.

En el apartado anterior se mencionaron aspectos claves de cómo se utiliza la metodología Scrum en el proyecto. Ahora nos vamos a ocupar de cómo se aplica la metodología Rup.

Rup se utiliza como metodología de la cual se toman los principios para definir sus artefactos.

Para toda la etapa de análisis del proyecto eSidif se utiliza una metodología tradicional, más específicamente RUP.

El equipo del área de análisis se reúne con el representante de los usuarios del sistema (por ejemplo la TGN, Tesorería General de la Nación), en encuentros denominados “Talleres”. En estos talleres el usuario realiza una presentación de la funcionalidad deseada. Esta funcionalidad habitualmente es un pedido que consiste en la funcionalidad que utilizan en el sistema anterior (SLU), más algunas mejoras que puedan surgir.

Los requerimientos son plasmados en distintos artefactos generados por los analistas, como por ejemplo: prototipos de interfaz de usuario (PIUs), validaciones, ejemplos de uso, etc. Estos artefactos son presentados al usuario en un taller posterior y en el cual dicho usuario puede realizar algún tipo de modificación que es ajustada por los analistas. Esto permite un buen feedBack a la hora de realizar la solicitud y la propuesta de los requerimientos.

Una vez que la presentación es aceptada por el usuario, los analistas pueden pasar a la siguiente etapa donde se plasman los requerimientos obtenidos y ajustados, en los artefactos que posteriormente van a consumir desde el área de diseño y desarrollo (DyD).

Con los artefactos nos referimos a documentar en el EA (Enterprise Architecture), los requerimientos obtenidos en los talleres, mediante el Modelo de Casos de Uso (MCU),

Diagramas de Transición de Estado (DTE), Modelo de Clases de Análisis (MCA), Modelo de Reglas de Negocio (MRN), Prototipos de Interfaz de Usuario (PIU), etc.

El área de Análisis genera una especificación de requerimientos (como dijimos anteriormente en los encuentros denominados "talleres") a través de la creación de una serie de artefactos que son tomados por el área de Diseño. Estos artefactos son los siguientes:

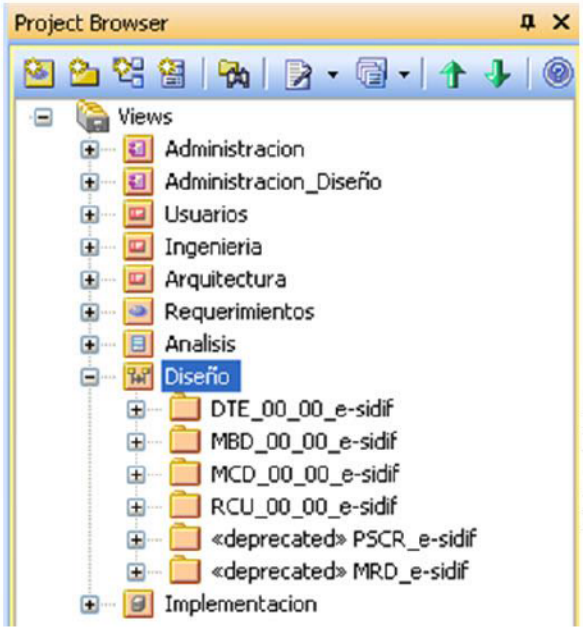
- Modelo de Casos de Uso (MCU)
- Modelo de Clases de Análisis (MCA)
- Modelo de Reglas de Negocio (MRN)
- Prototipos de Interfaz de Usuario (PIU)
- Diagrama de Transición de Estados (DTE)

A partir de los artefactos anteriormente mencionados, se analiza e interpreta esa documentación y se toman las decisiones de diseño y de implementación a nivel de equipo.

Aquí es el punto donde se observa claramente la convivencia de ambas metodologías. Por un lado desde el área de análisis, los cuales se reúnen con los clientes en los denominados talleres, con la finalidad de generar la documentación para plasmarla en los artefactos correspondientes (esto es siguiendo los principios de Rup). Los artefactos luego son consumidos por los desarrolladores, pero que realizan la planificación, tienen su comunicación propia (a través de las diversas reuniones que nombramos a lo largo del trabajo) y se organizan a su manera en sus respectivos equipos; esto ocurre bajo la línea de Scrum. Lo mismo sucede en el área de Testing con su propia planificación y comunicación como en el caso de los desarrolladores.

En la figura 34 se visualizan algunos de los artefactos mencionados anteriormente.

Modelos generados (en el EA)



Diagramas de Transición de Estados (DTE)
Modelo de Base de Datos (MBD)
Modelo de Clases de diseño (MCD)
Realización de Casos de Uso de Diseño (RCU)

Figura 34 – Artefactos documentados en el EA(Enterprise Architect)

Los artefactos de la figura 34 se encuentran contenidos en los denominados ***subsistemas*** (VER ANEXO A: SUBSISTEMAS).

5.3 Disciplinas, roles, responsabilidades y artefactos generados con Rup en eSidif

En esta sección se van a explicar los roles que cumplen las personas en el proyecto eSidif, de acuerdo a las fases del ciclo de vida correspondientes al proceso de desarrollo RUP.

Durante cada una de las fases del ciclo de desarrollo de Rup (adpatadas al proyecto eSidif) se realizan y se destacan las siguientes disciplinas:

- *Modelado de Negocios*
- *Requerimientos*
- *Análisis*

Modelado de negocios

El propósito de esta disciplina es lograr una comprensión de los procesos del negocio que el sistema debe soportar; capturar un vocabulario del negocio común a todos

los interesados; ubicar en contexto los problemas que los usuarios identifican actualmente y las propuestas de solución o mejora de los mismos.

El modelado del Negocio es la base para obtener los requerimientos del sistema necesarios para soportar los procesos de la organización.

Las principales actividades del modelado de negocio son:

- Definir el alcance del sistema.
- Modelar los procesos del negocio vigentes en la organización para los cuales exista una intención de mejora o reingeniería.
- Capturar los problemas actuales e identificar potenciales mejoras a los procesos de negocio.
- Modelar los nuevos procesos del negocio.
- Modelar las entidades u objetos esenciales del dominio que participan en los procesos del negocio.
- Elaborar un glosario de términos del negocio.
- Capturar los nuevos requerimientos del sistema.
- Lograr la aceptación del Modelo Conceptual por todos los interesados: OR, Comité.

Cabe mencionar que se destacan dos tipos de roles en esta disciplina, que son:

1. Equipo multidisciplinario funcional(EMF)
2. Analistas

Responsabilidades de Equipo multidisciplinario funcional

Entre las actividades del EMF durante el modelado del negocio se destacan las de:

- Identificar problemas en los procesos de negocio y en los sistemas que dan soporte a los mismos.
- Proponer soluciones o mejoras a los mismos.
- Definir términos del Glosario de Negocio.
- Participar en la validación y aprobación de los productos de la disciplina.

Responsabilidades de los Analistas

Entre las actividades de los analistas durante esta etapa se destacan las de:

- Modelar los procesos de negocio y las entidades esenciales del dominio.
- Capturar los requerimientos y reglas del negocio.
- Participar en la validación técnica de los productos de la disciplina.
- Registrar los términos del negocio en el Glosario de Negocio.
- Validar la consistencia y completitud de los términos contenidos en el Glosario de Negocio.
- Administrar el Modelo de Objetos del Dominio.

Insumos

Los insumos de los cuales hace uso la etapa de modelado de negocios son las descripciones de procesos del negocio, documentación de la normativa a seguir, en general cualquier fuente de información sobre los productos de la Administración Financiera que sirva como antecedente.

Productos generados

Los productos generados durante la etapa de modelado de negocios son los que se explican a continuación:

- DF (Descripción Funcional): es una descripción textual del proceso de negocio.

- DA (Diagramas de Actividad): es una representación gráfica de la DF. Describe para cada proceso de negocio las actividades que realizan, los roles que participan, los objetos y/o actividades de negocio que son insumos de cada actividad y de las que se producen como consecuencias de las mismas. Junto con DF conforman lo que se conoce como MCN.

- MCN (Modelo Conceptual de Negocio): es la formalización, mediante Diagramas de Actividad (DA), de la descripción de los procesos de negocio relevada durante los talleres funcionales. Cada DA describe las actividades que se realizan en el contexto del proceso, los roles que las llevan adelante, los objetos o entidades del negocio que son insumo de cada actividad y las que se producen como consecuencia de las mismas.

  Los objetos o entidades del negocio se organizan en el Modelo de Dominio (MOD).
  El MCN sirve como base para el planteo de los escenarios básicos y alternativos de los Casos de Uso.

- MPN (Modelo de Procesos de Negocio): El MPN, muestra cómo un conjunto de actividades distribuidas entre los roles participantes y a través de una adecuada asignación de responsabilidades satisfacen los objetivos del negocio obteniendo un conjunto de productos verificables.

- MOD (Modelo de Objetos del Dominio): es una vista de las entidades identificadas en el relevamiento, análisis y modelado del negocio y sus principales atributos y responsabilidades desde el punto de vista del negocio, y de las relaciones que vinculan las entidades entre sí y las restricciones y/o condiciones que se identifican.

- GLO (Glosario del Negocio): Catálogo alfabetizado de siglas, palabras y expresiones utilizadas en la Administración Pública Nacional.

- MDR (Minuta de Requerimientos): es el artefacto donde se registran las conclusiones de cada uno de los talleres funcionales desarrollados durante la primera etapa del Proyecto. Incluye la descripción funcional del proceso de negocio, los problemas identificados, los requerimientos funcionales propuestos, las reglas del negocio y el vocabulario o glosario utilizado durante el taller.

Requerimientos

Etapa de la captura de Requerimientos:

Figura 35 – Etapas de Requerimientos

Etapa 1: Modelo Conceptual del Negocio

Los objetivos principales del modelado conceptual son:

- Definir el alcance del sistema.
- Lograr la aceptación del Modelo Conceptual por todos los interesados: OR, Comité.

Etapa 2: Talleres Funcionales

Los objetivos principales de los talleres funcionales son:

- Refinar y completar la definición del proceso de negocio obtenida durante la ETAPA 1 de la Captura de Requerimientos.

Etapa 3: Talleres de Detalle

Los objetivos principales de los talleres de detalle son:

- Refinar y completar la definición del proceso de negocio obtenida durante la ETAPA 2 de la Captura de Requerimientos.

- Definir las transacciones válidas que permitan la gestión del negocio que se está analizando.
- Capturar las reglas que definen la ejecución de las transacciones y la estructura y controles específicos de los objetos o entidades del negocio: validaciones requeridas, impactos, restricciones de seguridad, etc.
- Definir los prototipos de pantallas (PIU) y los diseños de listados.

El propósito de la disciplina de requerimientos es desarrollar un modelo funcional del sistema que se va a construir, con un énfasis especial en el valor añadido para cada usuario individual o para cada sistema externo.

El modelo de casos de uso ayuda a pensar el sistema en términos de quiénes son sus usuarios, y qué necesidades u objetivos de la organización pueden cumplir.

Las principales actividades de la disciplina son:

- Delimitación del sistema y de su entorno, identificando qué actores interactuarán con el sistema y qué funcionalidad se espera del sistema.
- Alimentación del glosario de términos comunes.
- Identificación de los Casos de Uso y descripción breve de los mismos.
- Priorización de los Casos de Uso, inicialmente, aquellos significativos para la arquitectura.
- Descripción en detalle de los Casos de Uso prioritarios especificando el escenario principal, los alternativos y de excepción.
- Estructuración del modelo de Casos de Uso, incluyendo las relaciones de inclusión, extensión y generalización entre actores y casos de uso.
- Identificación de patrones y de otras abstracciones y evolución de los escenarios de los casos de uso en función de la estabilidad de los requerimientos.
- Especificación las reglas del negocio y los cambios o impactos que la ejecución del Caso de uso produce, y realización del trazado con los Casos de Uso.
- Elaboración de los prototipos de Interfaz de Usuario (PIU) siguiendo los lineamientos de las guías de usabilidad.
- Especificación de los Mensajes al Usuario (MSJ) y realización de trazado con los Casos de Uso.
- Especificación de los requerimientos no funcionales.

Existen tres roles:

1- Analistas
2- Equipo Multidisciplinario Funcional (EMF)
3- Réplicas

Responsabilidades de los analistas

Dentro de las responsabilidades del grupo de analistas las más relevantes son las de:

- Detectar requerimientos funcionales y no funcionales.
- Participar en la priorización de requerimientos.
- Revisar la consistencia de modelos y artefactos en general.
- Coordinar las instancias de validación de los requerimientos capturados.
- Modelar los procesos de negocio.
- Obtener una primera versión de los requerimientos.

Responsabilidades de Equipo multidisciplinario funcional

Dentro de las responsabilidades del grupo EMF se destacan las de:

- Identificar problemas y posibles soluciones.
- Proponer requerimientos del software.
- Priorizar requerimientos.
- Validar requerimientos y modelos.

Responsabilidades de réplicas

La principal responsabilidad de réplicas durante la etapa de requerimientos es la de participar en las actividades aportando la visión del usuario final.

Productos generados

- LBR (Línea Base de Requerimientos Funcionales y No funcionales): es el repositorio de los requerimientos de "alto nivel del sistema". La LBR consolida los nuevos requerimientos, identificados durante los talleres funcionales y técnicos, y los requerimientos que están contemplados en los sistemas actuales. La LBR sirve como base para el planteo de los Casos de Uso del sistema. Disponer de un repositorio centralizado de requerimientos, facilita la administración y control de cambios a lo largo del ciclo de vida del proyecto.

- VIS (Visión del Sistema): El propósito de este documento es recopilar y definir las necesidades de alto nivel y rasgos del sistema. Está enfocado en las capacidades requeridas por los interesados y los usuarios finales, y por qué existen estas necesidades.

- MRN (Modelo de Reglas del Negocio): Las reglas del negocio enuncian características que definen (CÓMO, QUIÉN, CUÁNDO, DÓNDE) la ejecución de las transacciones y la estructura y controles específicos de los objetos o entidades del negocio. Ya sean, validaciones requeridas, impactos, algoritmos a utilizar, restricciones de seguridad,

información requerida para cada objeto de negocio, valores permitidos, obligatoriedad, relaciones entre objetos, etc. El MRN consolida las reglas del negocio, identificadas durante los talleres funcionales y técnicos desarrolladas durante la primer etapa del proyecto. Este modelo debe completarse con las reglas que están presentes en los sistemas actuales.

- MCU (Modelo de Casos de Uso): Un caso de uso describe, en lenguaje natural, la secuencia de interacciones necesarias entre un actor y el sistema para lograr algún resultado de valor para un usuario del sistema.
  Los CU dirigen el proceso de desarrollo y por lo tanto constituyen el insumo esencial para las actividades de planificación, análisis, diseño y testing. A partir de la LBR y el MCN se construye una primera versión del modelo de Casos de uso. Los mismos deben especificar en detalle los requerimientos funcionales del sistema según la visión externa del usuario.

- PIU (Prototipo de Interfaz de Usuario): especifica las características de la interfaz del usuario para cada Caso de Uso: la organización general del escritorio, los tipos de ventana, la información a mostrar para cada entidad y/o herramienta del sistema y las alternativas de navegación. En el PIU se indican aquellos Componentes Gráficos Reusables (CGR) que se usan en la pantalla.
  Un CGR es una porción de pantalla, publicada en una biblioteca, que es utilizada por distintas pantallas de la aplicación.

- MSJ (Mensajes al usuario): Los mensajes definen la comunicación con el usuario de la aplicación. Lo orientan en la resolución de problemas, lo informan sobre eventos ocurridos que impiden o podrían impedir la ejecución de ciertas acciones a través del sistema.

  Los MSJ se especifican en distintos momentos a lo largo del ciclo de vida: los vinculados con la lógica del negocio durante la especificación de requerimientos y trazados con el MCU; los vinculados a la ejecución de la aplicación, durante la especificación del diseño y los vinculados con la lógica de presentación y temas de interfaz se terminan de definir durante el desarrollo.

Análisis

El propósito del modelo del análisis es que los desarrolladores adquieran un conocimiento profundo de los requerimientos especificados en el MCU.

Utiliza un lenguaje más formal (Orientado a Objetos) que el utilizado para especificar los casos de uso y comienza a definir una vista "interna" del sistema mostrando las entidades del negocio involucradas en los Casos de Uso , sus atributos, operaciones y la forma en que se relacionan.

Es un modelo conceptual, ya que, es independiente de la plataforma seleccionada, y no contiene decisiones basadas en la tecnología sobre la que se va a implementar el sistema.

El modelo de análisis se ve reflejado en un modelo de clases conceptuales (del negocio) necesarias para realizar o llevar a cabo la funcionalidad especificada en los Casos de Uso.

Las principales actividades relacionadas con el análisis son las que se presentan a continuación:

- Análisis del modelo de Casos de Uso para descubrir las clases esenciales y sus atributos conceptuales reconocibles en el dominio del problema.

- Modelamiento de las relaciones entre clases (asociación, composición, y generalización).

- Especificación de operaciones.

- Elaboración de los diagramas de transición de estados para cada entidad(objeto), identificando sus estados y las transiciones o acciones que realiza el usuario (modeladas en CU) y que provocan un cambio de estado o bien un retorno al mismo estado de la entidad/objeto al que corresponde el DTE. En su mayoría serán CU del mismo objeto, pero puede haber un CU de otro objeto/entidad que impacten en el estado del objeto/entidad al que corresponde el DTE.

- Definición de una Arquitectura Funcional Candidata; identificando los casos de uso arquitecturalmente significativos y realizando una subdivisión inicial del sistema en subsistemas del análisis.

En esta disciplina se distinguen 3 roles (al igual que la disciplina anterior):

1- Analistas
2- (EMF)Equipo Multidisciplinario Funcional Réplicas

Responsabilidades de los analistas

Entre las responsabilidades de los analistas en esta etapa de Rup, se destacan las de:

- Priorizar Casos de Uso.
- Definir los paquetes del análisis y sus vinculaciones.
- Detectar posibilidades de reuso.
- Revisar la consistencia de modelos y artefactos en general, productos de la disciplina.
- Especificar los Casos de Uso.
- Derivar Clases Esenciales
- Diseñar Interfaces de Usuario.
- Coordinar las validaciones de modelos y artefactos.

Responsabilidades de EMF

Dentro de las responsabilidades más importantes del equipo multidisciplinario Funcional correspondiente a la etapa de análisis se encuentran las de:

- Participar en la definición del Modelo de Casos de Uso
- Participar en la validación de los modelos y artefactos.

Responsabilidades de réplicas

Por último, la responsabilidad más resaltante de réplicas durante el análisis es la de participar en las actividades aportando la visión del usuario final.

Insumos

Los insumos de los que hace uso análisis son los siguientes:

- LBR (Línea Base de Requerimientos)
- VIS (Visión del Sistema)
- GLO (Glosario del Sistema)
- MPN (Modelo de Procesos de Negocio)
- MCU (Modelo de Casos de Uso.)
- MRN (Modelo de Reglas de Negocio)
- PIU (Prototipo de Interfaz de Usuario)

Productos generados

Los productos generados como consecuencia y resultado de la etapa de análisis son:

- Paquetes del análisis y sus vinculaciones.

- MCA (Modelo de Clases del Análisis): es una representación de clases conceptuales del mundo real extraídas de un dominio de interés. El modelo puede considerarse un diccionario visual del contenido y vocabulario del dominio y servirá para facilitar la comprensión de los elementos y relaciones definidos en lenguaje natural en los casos de uso.

- DTE (Diagrama de Transición de Estados): este artefacto describe los estados por los que puede pasar un objeto (comprobante o entidad del dominio) durante su ciclo de vida y el comportamiento en esos estados junto con los eventos que causan los cambios de estado.

5.4 Scrum de Scrum

5.4.1 Introducción

Se ha dicho que los métodos de desarrollo de software ágil como Scrum, son ideales para proyectos desarrollados por equipos multidisciplinarios de pocas personas y localizados en la misma oficina. Sin embargo, cada vez se observan más ejemplos de grandes compañías utilizando Scrum en proyectos de gran escala, con equipos de proyectos integrados por decenas y hasta cientos de desarrolladores.

La técnica de las reuniones Scrum de Scrum es la que permite escalar el enfoque Scrum para grandes equipos de proyecto a escala corporativa. Esta técnica consiste en dividir un equipo de muchas personas en diferentes equipos Scrum, para luego utilizar reuniones Scrum de Scrum para coordinarlos. A cada reunión de Scrum de Scrum asiste uno o dos integrantes de cada equipo.

¿Cuantos equipos Scrum se necesitan?

Dado que el tamaño ideal de un equipo Scrum es de entre 5 y 9 participantes, cuando se tienen 10 o más integrantes, la solución es dividir a un gran equipo en sub-equipos, manteniendo la regla de entre 5 y 9 participantes. Cada equipo deberá tener un Scrum Master, su propia lista de objetivos y características (Backlog) y ser atendido por un dueño de producto (Product Owner).

¿Qué roles de desarrollo de software integran cada equipo de desarrollo Scrum?

No es una buena práctica dividir los equipos por componentes o especializados por tecnología, por ejemplo desarrollo de Base de datos, desarrollo de aplicaciones, equipo de pruebas, etc. En su lugar, la buena práctica es crear equipos multifuncionales, cada uno capaz de desarrollar y probar una característica (Feature) del producto, en todos los componentes y capas involucrados. Por ende, el equipo deberá incluir desarrolladores de todas las capas involucradas, analistas funcionales, analistas de pruebas, etc.

Deben distribuirse por igual los niveles de experiencia y conocimiento del negocio (Know-How), con la intención que cada equipo sea capaz de tomar una historia del Backlog y transformarla en un producto.

¿Quienes asisten a la reunión Scrum de Scrum?

Cada equipo designa a uno de sus integrantes para asistir a la reunión Scrum de Scrum. Es buena práctica que sea un participante técnico (Desarrollador o Tester), y no el dueño de producto o Scrum Master.

No necesariamente tiene que ser siempre la misma persona, sino que puede rotar según la etapa del proyecto y quien esté en mejor posición para contribuir con la reunión. Al principio del proyecto deberían enviarse a personal técnico o diseño de usuario, mientras que al final debería enviarse a personal que está ejecutando las pruebas.

Si el número de equipos Scrum es pequeño, pueden asistir hasta dos personas por equipo, por ejemplo un participante técnico y el Scrum Master.

Escalamiento de las reuniones Scrum de Scrum

Cuando los proyectos son grandes, las reuniones Scrum de Scrum se pueden escalar en múltiples niveles. Por ejemplo, supongamos que se tienen 49 equipos Scrum de 7 personas cada uno. Por cada equipo se selecciona a una persona para asistir a reuniones Scrum de Scrum. De esta forma, se conforman 7 reuniones con 7 participantes cada una. Por cada uno de estos equipos se puede escalar a 3 equipos y estos a su vez a un equipo directivo. Bajo este ejemplo se tendrían 4 niveles.

Frecuencia de las reuniones Scrum de Scrum

Las reuniones Scrum de Scrum deben ser frecuentes, pudiendo considerar inclusive la frecuencia diaria o tres veces a la semana (por ejemplo los lunes, miércoles y jueves). Se recomienda que tenga la misma duración de la reunión diaria (15 minutos).

Tal como indica Mike Cohn (miembro de la junta directiva del Scrum Alliance) es aconsejable reservar 30 minutos adicionales para resolución de problemas y consultas de información que surjan de la reunión, dado que si bien a nivel de equipo Scrum se pueden dejar asuntos por resolver en reuniones sucesivas, en las Scrum de Scrum debe buscarse resolución inmediata y así evitar programar muchas reuniones con las mismas personas.

Agenda

La agenda de las reuniones Scrum de Scrum debería ser similar a la de los Scrums diarios, realizando algunos ajustes, dado que en la reunión no están los equipos completos sino representantes de cada equipo. A cada integrante del Scrum de Scrum, que a su vez representa cada uno a un equipo, se le pregunta lo siguiente:

- ¿Qué actividades ha ejecutado tu equipo desde la última reunión?
- ¿Qué actividades realizará tu equipo antes de la próxima reunión?
- ¿Existen impedimentos en tu equipo?
- ¿Existe alguna actividad a ejecutar próximamente por tu equipo que interfiera o afecte de alguna forma el trabajo de otro equipo?

Al igual que las reuniones diarias (Daily Scrum), la intención es que la exposición de cada integrante sea breve e ir al punto central. Los problemas se pueden mencionar, no se buscan soluciones hasta que cada integrante ha hecho su exposición.

Una vez finalizada se realiza la segunda parte de la reunión, en la cual los participantes discutirán los problemas o situaciones planteadas, o puntos tratados en reuniones previas que aún no se han cerrado.

El equipo mantendrá un Backlog de Scrum de Scrum, el cual es análogo a lo que se llamaría una lista de issues en proyectos tradicionales. Una lista de issues es un simple listado con asuntos sobre los cuales los participantes necesitan tomar acción sobre ellos o hacer algún seguimiento por alguna razón. Pueden incluirse issues de otros grupos de Scrum de Scrum pero que este equipo necesita ser notificado de la resolución.

A diferencia de cada equipo Scrum, el grupo Scrum de Scrum no realiza planificaciones de iteración ni revisiones de Backlogs sino que cada miembro del grupo es simplemente un participante del grupo al que pertenece, con la intención de coordinar actividades entre grupos Scrum. Cada iteración puede traer consigo un conjunto de participantes diferentes en las reuniones Scrum de Scrum. Quien asume los compromisos y la responsabilidad que el proyecto avance es cada equipo Scrum individual, el grupo Scrum de Scrum no tiene responsabilidades directivas.

5.4.2 "Scrum de Scrum" aplicado al Esidif

Como ya se explicó anteriormente, nuestro proyecto de referencia tiene una magnitud muy importante, tanto en desarrollo como en cantidad de participantes. La alternativa de metodología que se pensó para llevar a cabo el sistema, fue la adaptación de "Scrum de Scrum", que permite mantener el concepto de desarrollo ágil sin perder de vista la envergadura del proyecto.

En dicho proyecto trabajan más de 10 equipos, cada uno de estos formados por las respectivas áreas de Análisis (ANA), Diseño/Desarrollo (DyD) y Testing (TST).

La primera gran planificación que se realiza es a nivel gerencial, donde se acuerdan todos los objetivos esperados al momento de realizar el despliegue a producción. Habitualmente esta planificación se realiza a principio del año calendario.

Desde una visión global se realiza una segunda planificación a nivel de los objetivos que posee cada equipo para una determinada fecha de entrega (habitualmente cada 3 semanas), teniendo en cuenta la cantidad de recursos que posee el equipo, las horas laborales, la complejidad de las tareas, etc. Este es un Scrum a gran escala del proyecto.

Por otro lado, cada equipo tiene sus "áreas" (DyD, ANA, TST), y cada una realiza la planificación de un Sprint por separado. Es decir, el área de DyD, planifica un sprint con los objetivos pensados en la "planificación superior", en forma de tareas a través del product backlog y donde las mismas son estimadas en horas y luego son tomadas por los recursos del equipo los cuales registran sus horas de desarrollo en cada una. Lo mismo realiza el área de TST. Esto representa el Scrum interno de cada área.

En la figura 36 podemos visualizar la estructura y organización de los equipos utilizando Scrum de scrum.

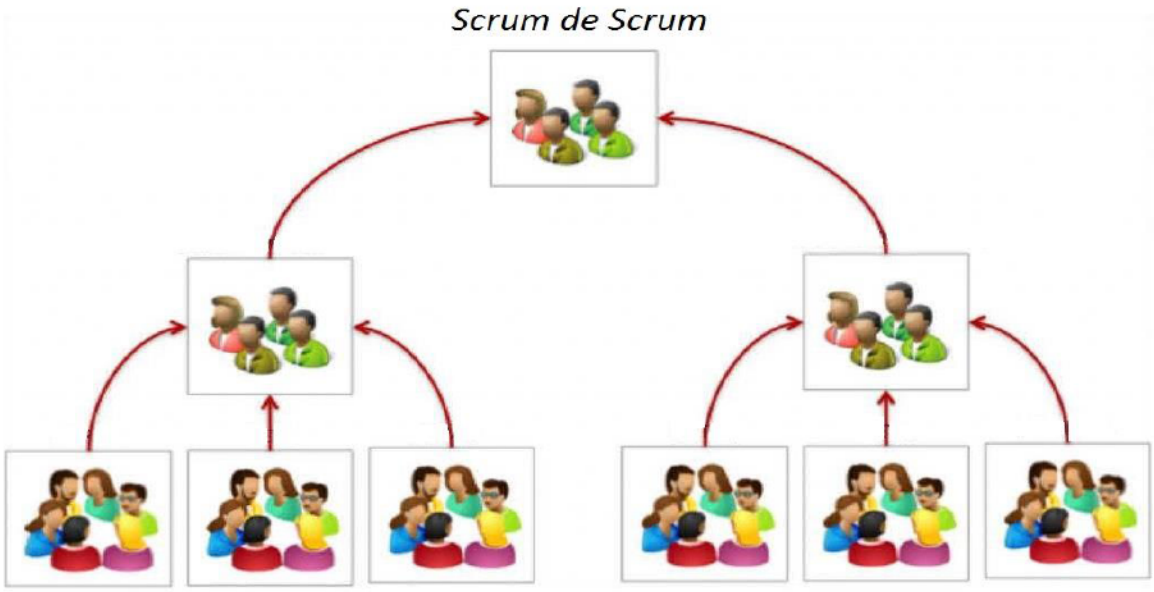


Figura 36 – Organización y estructura de Scrum de Scrum

La coordinación del proyecto se lleva a cabo mediante distintos niveles de reuniones, cada una de las cuales tiene distintos objetivos, alcances y están conformadas por diferentes participantes.

A continuación se detallan dichas reuniones:

Reuniones N1 (de priorización):

Objetivos

- Las reuniones conocidas como N1 se utilizan para realizar la priorización de defectos.
- En forma periódica (semanalmente por defecto) se llevan adelante las reuniones N1, en las cuales se presentan los defectos no priorizados. Si como parte de la priorización se detectan defectos de alta criticidad se podría tener que actualizar el sprint backlog para poder desarrollar la solución a dicho defecto de la forma más inmediata posible.
- Otros de los objetivos de este nivel es plantear las dudas funcionales que pudieron surgir en el Sprint.

Precondiciones

- Contar con la lista de defectos a priorizar.
- Contar con un detalle de todos los incidentes e impedimentos que ponen en riesgo el sprint.

Participantes

- Referentes por área, es decir, Diseño y Desarrollo, Testing, Análisis.
- Réplicas, que hacen de intermediario representando al usuario.

Reuniones N2

Objetivos

- El objetivo de las reuniones N2 es reportar el avance del negocio al PMO a cargo.
- También se realiza la sincronización necesaria entre las diferentes áreas.
- Tiene una frecuencia semanal.
- Se tratan también temas sobre recursos, es decir, vacaciones de los mismos, ingresos, egresos.

Precondiciones

Para esta reunión se deben completar y actualizar una serie de artefactos que comprenden:

- Informe de Avance.
- Burndown Chart del Sprint en curso.
- Product Burndown Chart.
- Defectos.
- Presentación de estado de Bugs a cargo del equipo de Testing.
- Cambios de alcance o redefinición de requerimientos.
- Ítems pendientes de Arquitectura.
- Alertas N1.
- Incidentes.
- Riesgos.
- Novedades de Recursos.

Participantes

- PMO encargado del seguimiento externo.
- Analistas.
- Testing ➔ reporta datos estadísticos, por ejemplo cantidad de ítems reingresados, cantidad de ítems corregidos, etc.

Reuniones N1 de PMO:

Objetivos

El objetivo de las reuniones N1 de PMO es poner al tanto el estado de avance de los distintos equipos.

Precondiciones

- Se sincronizan los objetivos comunes.
- Dependencia funcional con otro equipo.

Participantes

- PMO de cada equipo

Cabe aclarar que se realizan reuniones de N3 pero en este caso los PMO participan de forma no presencial a través de un avance que envían en un formulario estándar donde se visualiza el avance del proyecto.

CAPÍTULO 6. Trabajos relacionados

A continuación se mencionan algunos documentos que están relacionados con esta tesis:

- ***RUP versus Scrum: una comparación empírica en un ámbito académico [3]***: Santiago D'Andre, Miguel Martinez Soler, Gabriela Robiolo Universidad Austral. Este documento realiza una comparación empírica entre Rup y Scrum. Esta se basa en un experimento formal que se desarrolló en el contexto de un taller de diseño de 4to año, que forma parte de la carrera de Ingeniería en Informática correspondiente a la Facultad de Ingeniería de la Universidad Austral. En este experimento, se divide al curso de alumnos en dos grupos: un grupo el cual trabajará con Scrum y un grupo que trabajará con Rup. Ambos grupos trabajarán en forma independiente en el desarrollo de un juego de estrategia por turnos partiendo de una misma definición de requerimientos. A partir de esta división ambos grupos trabajan en su objetivo y se realiza el experimento en base a los resultados que obtienen ambos grupos, realizando así la comparación entre Rup y Scrum.

- ***Tesis de grado PDSM: Proceso de Desarrollo de Software Mixto, combinando Rup y Scrum [9]***. En esta tesis se realiza una combinación de metodologías para generar un nuevo proceso de desarrollo de software que se adapta a proyectos pequeños como medianos y proyectos nuevos y de mejoras y mantenimientos.

- ***Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise (IBM Press) Autores: Scott W. Ambler (Author), Mark Lines (Author) [11].*** Es ampliamente conocido que pasar de enfoques tradicionales a enfoques ágiles para crear soluciones de software representa una fuente crítica de ventaja competitiva. Los enfoques ágiles convencionales, que resultan de hecho ser apropiados para proyectos menos complejos, requieren ser diseñados a medida de forma significativa para proyectos empresariales de mayor tamaño y complejidad. En Disciplined Agile Delivery, Scott W. Ambler y Mark Lines presentan un marco de procesos de entrega ágil disciplinada (DAD), un avance de IBM, que describe la manera en que se debe realizar el diseño a medida. DAD aplica un enfoque más disciplinario para el desarrollo ágil al reconocer y tratar con las realidades y complejidades de una cartera de iniciativas de programas interdependientes.
  Ambler y Lines muestran cómo extender Scrum con estrategias suplementarias, ágiles y de tendencia para Modelos Agiles (MA), XP (Extreme Programming), Kanban, UP (Unified Process), y demás métodos demostrados para proveer un enfoque híbrido que se adapte a sus necesidades únicas de organización. Los autores francamente describen cuáles son las prácticas que funcionan de mejor manera, por qué lo hacen, cuáles son las compensaciones, y cuándo es necesario considerar alternativas, todo dentro del contexto de su situación.

Disciplined Agile Delivery aborda prácticas ágiles a lo largo del ciclo de vida completo, que van desde requerimientos, diseño y desarrollo hasta entrega y manejo. Los autores muestran cómo estas técnicas de prácticas efectivas encajan en un proceso de principio a fin para entregar grandes sistemas complejos de manera exitosa: desde la etapa de inicio del proyecto hasta su entrega.

- ***Gestión de Proyectos: ¿formal o ágil? [12].*** Es un artículo que presenta distintas clasificaciones de la gestión de proyectos así como también sus características. Por otra parte, presenta premisas que indican cuándo y cómo aplicar un estilo de gestión para determinado proyecto, dependiendo de las características del mismo y de la organización subyacente.

- ***CoMakeIT : White Paper on the Agile process: Rup y Scrum[13]***: Es un documento que explica las metodologías Rup y Scrum así como también como pueden acoplarse y convivir ambas en un proyecto.

Los documentos mencionados y explicados anteriormente están relacionados con nuestra tesis en cuanto a los temas centrales de los mismos que son las metodologías de desarrollo Scrum y Rup.
Por otra parte, nuestro artículo realiza una ampliación de estos documentos explicando, además de las metodologías por separado, la posibilidad de convivencia de ambas, centrándonos en la aplicación de dicha convivencia en un proyecto de gran escala y aplicando un concepto importante como es el de scrum de scrum.

CAPÍTULO 7. Conclusiones y trabajos futuros

7.1 Conclusiones

Como conclusión general podemos decir que no existen recetas para aplicar metodologías de desarrollo de software a los proyectos. Cada metodología posee sus principios, y por ende sus ventajas y/o desventajas. Por eso, es importante tener en cuenta qué aspectos se van a necesitar de una metodología y de esta forma aplicarla a un proyecto tomando las cosas que sean necesarias para el mismo y adaptándolas.

Por otra parte, es importante resaltar que dos metodologías distintas y con distintos principios pueden convivir, como es el caso del proyecto que presentamos, donde fusionamos las características más convenientes de cada una para sacar el mayor provecho de las mismas y optimizar un nuevo proceso de desarrollo de software combinado.

Con respecto a la las metodologías podemos mencionar varias ventajas y utilidades que encontramos con su adaptación y utilización:

- Scrum de Scrum nos facilitó la organización del proyecto teniendo en cuenta la gran cantidad de participantes. Nos permitió una fluidez en la comunicación entre los integrantes del equipo, y con otros equipos, que es muy necesario para mantener un nivel de desarrollo estable.
- Diversidad de aprendizaje a la hora del desarrollo, teniendo en cuenta que la metodología permite que los distintos recursos puedan ir tomando distintas tareas con distintas responsabilidades en un mismo proyecto.
- Encontramos la posibilidad de ir rotando en los roles que cumple un integrante dentro del proyecto. Por ejemplo, este integrante puede cumplir tareas de desarrollador, luego formar parte del equipo de análisis, involucrarse como responsable técnico del equipo, etc. Esto también tiene como finalidad el crecimiento personal en la adaptación a distintos grupos, con diversas personalidades bajo diferentes responsabilidades.
- Se produce un importante feedback entre los participantes a través de las distintas reuniones que fueron explicadas en detalle con anterioridad (retrospectivas, N1, N2, planning, daily). Lo que permite resolver problemas rápidamente, tener distintos puntos de vista o visualizar aspectos que uno no tuvo en cuenta para determinada tarea.
- La planificación a corto, mediano y largo plazo. Esto es un gran aprendizaje, y una experiencia muy necesaria, ya que por la magnitud del sistema es fundamental realizar distintos niveles de planificación, siendo una de las tareas más complicadas a la hr de llevar a cabo un proyecto.

Como desventajas o aspectos a tener en cuenta con la utilización de las metodologías podemos mencionar:

- Sincronización entre los equipos con el fin de llegar a un hito previamente comprometido. Esta es una tarea muy importante a tener en cuenta, ya que la magnitud del proyecto obliga a minimizar las distintas dificultades que pueden tener un impacto directo en el tiempo de entrega de una funcionalidad previamente acordada.
- Sobre-estimación /bajo-estimación de tiempo de tareas. Es otra dificultad en la que se debe prestar mayor atención, ya que el recambio de participantes es superior en relación a un proyecto pequeño, por lo tanto la estimación de las tareas debe adaptarse tanto a los recursos

con mayor experiencia como a aquellos recursos nuevos, teniendo en cuenta que dicha estimación tiene un impacto directo en la fecha de entrega acordada.

Por otra parte podemos decir que a pesar de que RUP es una metodología tradicional la cual centra su importancia en la documentación exhaustiva y a veces excesiva, en esta convivencia de metodologías resulta útil poder documentar determinadas actividades. En nuestro caso, (en el proyecto que presentamos y explicamos durante la tesis) la metodología documenta artefactos que son necesarios y útiles para distintos roles aplicados en diferentes etapas.

Por ejemplo, el MCA resulta útil para que el desarrollador pueda comprender los requerimientos que los analistas plasman en dicho modelo y de esta manera poder llevar a cabo la implementación de lo deseado. Esto resulta importante ya que, por un lado, el desarrollador puede interpretar los requerimientos descriptos por el analista de una forma más abstracta y posteriormente volcarlo, primeramente al diagrama de objetos o de clases, y luego al código. Por otra parte, el desarrollador tiene la libertad y el poder de decisión de ver cómo va a implementar lo propuesto por análisis, es decir, que hay una cierta flexibilidad para cumplir con los requerimientos presentados en el Modelo de Clases de Análisis.

De esta forma se puede concluir que además de que no existen recetas al momento de elegir una metodología para un determinado proyecto, nos resultó muy importante la comunicación entre todos los participantes, permitiendo la organización de los equipos de forma distribuida a través de las distintas reuniones, obteniendo varios puntos de comunicación con diferentes enfoques.

Otro punto destacable es que en algunos casos no es necesario abrumar con tanta documentación, la cual es difícil de mantener actualizada en un proyecto de tal magnitud, sino contar con la documentación más útil para los integrantes del sistema.

7.2 Trabajos Futuros

A partir de nuestro análisis y comparación de metodologías con distintos principios, como son Scrum y Rup, de la convivencia de las mismas y de la aplicación de Scrum de Scrum en el proyecto, proponemos una tarea similar pero con respecto a otra técnica que nos resultó interesante en la actualidad, y que se postula para ser de gran utilidad. Esta técnica es la denominada "Kanban".

Kanban: "El término kanban aplicado a la gestión ágil de proyectos se refiere a técnicas de representación visual de la información para mejorar la eficiencia en la ejecución de las tareas de un proyecto."[7]

Entonces planteamos como posibles trabajos futuros; análisis y desarrollo de proyectos que utilicen las nuevas alternativas de gestión ágiles como por ejemplo Kanban y su fusión con Scrum denominada "ScrumBan". Definiendo ventajas y desventajas, detallando qué aspectos de Scrum son comunes con Kanban y que otros aspectos particulares agrega Kanban a la gestión.

Realizar una comparación detallada de Kanban con las diferentes alternativas de gestión ágiles más utilizadas en la actualidad, como pueden ser, Scrum, XP, Lean; entre otras

Referencias Bibliográficas

- [1]José H. Canós, Patricio Letelier y Mª Carmen Penadés, *Metodologías Ágiles en el Desarrollo de Software* DSIC. Universidad Politécnica de Valencia,Camino de Vera s/n, 46022 Valencia.
- [2]Juan Palacio, *Flexibilidad con Scrum.*, Edición Octubre - Noviembre 2007.
- [3]Santiago D'Andre, Miguel Martinez Soler, Gabriela Robiolo, *RUP versus Scrum: una comparación empírica en un ámbito académico.* Universidad Austral (39JAIIO- ASSE 2010).
- [4]Ian Sommervile, *Ingeniería del software*, Séptima edición, Edición Año 2005.
- [5] Philippe Kruchten, *The Rational Unified Process: An Introduction* - Addison Wesley (2004).
- [6] RSC02 - *Rational Software Corporation, Product: Rational Software Corporation, Edition 2002*
- [7] Gestión Ágil de Proyectos Software- Javier Garzás, Juan Enríquez S., Emmanuel Irrazábal – Edición año 2012. URL de acceso: http://www.javiergarzas.com/2011/11/kanban.html. Fecha de acceso: 10/04/2014
- [8] Jacaboson, I., Booch, G., Rumbaugh J., El Proceso Unificado de Desarrollo de Software. Edicion 2000. Addison Wesley.
- [9] PDSM: Desarrollo de Software Mixto. Autores: Mariani Maria Florencia, Okabe Evangelina- Tesis de grado en Licenciatura en Informática, Facultad de Informática, UNLP (Universidad Nacional de La Plata). Directora: Dra. Claudia Pons. Noviembre 2010.
- [10] Portal Web del Laboratorio de Investigación y Formación en Informática Avanzada. URL:http://lifia.info.unlp.edu.ar/es/hacienda.htm. Fecha de acceso: 15/04/2014.
- [11] Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. Autores: Scott W. Ambler, Mark Line (IBM Press). Edición año 2012.
- [12] Gestión de Proyectos: ¿formal o ágil?- Juan Palacio. Edición año 2006. URL de acceso: http://www.navegapolis.net/files/s/NST-004_01.pdf. Fecha de acceso: 02/03/2014.

- [13] White Paper on the Agile Process: Rup and Scrum, CoMakeIT, Publication Date: October 10, 2007. URL de acceso :http://www.comakeit.com/download/agile_process_rup_scrum.pdf. Fecha de acceso: 30/03/2014.

- [14] Agile Project Management with Scrum - Ken Schaner Edited by Microsoft. Publication Date: February 21, 2004 | ISBN-10: 073561993X.

- [15] Diseño de una Metodología Ágil de Desarrollo de Software. Autor: Schenone Marcelo Hernán. Año 2004. Tesis de Grado en Ingeniería en Informática, Facultad de Ingeniería, UBA (Universidad de Buenos Aires). URL de acceso: http://materias.fi.uba.ar/7500/schenone-tesisdegradoingenieriainformatica.pdf. Fecha de acceso: 28/03/2014.

- [16] Scrum and XP from the Trenches - Henrik Kniberg. Edición año 2007.

ANEXO A: SUBSISTEMAS

Un subsistema es un conjunto de elementos de software que colaboran para proveer un comportamiento bien definido.

Este comportamiento es expuesto a otros subsistemas mediante una interfaz, de forma tal que los clientes de un subsistema (los consumidores de tal comportamiento) sólo pueden ver lo que éste ofrece en su interfaz. Pero no pueden ver su composición interna ni la forma en que lleva a cabo su funcionalidad.

¿Qué representa un subsistema?

Un subsistema es un recorte conceptual de un sistema con base en lo funcional, con aplicación en el diseño y la construcción.

Representa una forma de agrupar clases de diseño de un módulo o negocio en unidades más manejables.

Características de un subsistema

- ✓ Un subsistema debe ser cohesivo
  - sus componentes deben estar estrechamente vinculados.
- ✓ Un subsistema debe poseer bajo acoplamiento
  - las dependencias de unos con otros deben ser mínimas.
  - *la herramienta que permite lograr el bajo acoplamiento es la interfaz que todo subsistema deberá exponer*

¿Cómo se representa un subsistema?

- ✓ Se representa en el agrupamiento de clases de diseño en unidades manejables e independientes.
- ✓ En el e-SIDIF se ha determinado el uso de diagramas de subsistemas que representan su definición y evolución desde la disciplina de Análisis hasta su despliegue en un ambiente productivo.

Beneficios de los subsistemas

- ***Aislamientos de cambios***

  Es una consecuencia del encapsulamiento. Encapsular el comportamiento de un subsistema implica que los ***subsistemas clientes*** no se verán afectados por los cambios internos de un ***subsistema proveedor***, siempre y cuando la ***interfaz*** no tenga cambios.

- ***Implementación reemplazable***

  Si ningún subsistema cliente depende directamente de las partes internas de un subsistema, entonces es posible y sencillo reemplazar estas partes internas de un subsistema, o bien reemplazar la implementación completa si la interface se mantiene sin cambios.

  Esto puede ser especialmente útil cuando se quieran actualizar subsistemas, cambiar algoritmos (por ejemplo por cuestiones de performance), responder a cambios externos a un subsistema, etc.

- ***Abstracción***

  Los subsistemas sirven para elevar el nivel de abstracción del sistema en construcción y por lo tanto ayudan a tener una mejor comprensión del mismo.

  El subsistema esconde los detalles internos de su implementación.

  Esto significa que los clientes del subsistema no deben preocuparse por cómo están implementados los servicios que éste expone, sino sólo por saber qué hacen esos servicios.

- ***Reuso***

  El diseño con subsistemas favorece el reuso ya que se abstraen los detalles internos de un subsistema.

  La organización en subsistemas ayuda a que se tenga una visión más integrada del sistema ayudando a reducir la cantidad de código duplicado.

  Una estructura de subsistemas bien definida permite saber qué clases existen o que funcionalidad se encuentra diseñada o implementada, evitando la duplicación de código.

- ***Desarrollo paralelo***

  Una vez que los subsistemas fueron definidos pueden ser asignados a diferentes grupos de trabajo favoreciendo el desarrollo en paralelo.

  Cada grupo sólo debe respetar las interfaces definidas previamente.

- ***Demora en la especificación de detalles***

  Diseñar con subsistemas nos permite usar una aproximación top-down, partiendo de lo más general y entrando luego en los detalles específicos.

- ***Facilidad de construcción***

  Se puede comenzar a construir un subsistema (A) aunque otro del que depende (B) no esté construido, siempre que la interfaz de (B) este establecida.

- ***Vista global del sistema***

  Poseer una vista de alto nivel de la arquitectura funcional permite evaluar fácilmente los impactos que puede tener un cambio en algún subsistema.

- ***Mejora en la utilización de recursos***

  Las interfaces sirven como contratos entre equipos de trabajo.

- ***Favorece la tercerización de trabajo***

  El bajo acoplamiento permite asignar el desarrollo a terceros minimizando el costo de integración y comunicación que tendría en forma tradicional.

  Esta independencia favorece el desarrollo en lugares físicos diferentes.

- ***Facilita la automatización del test***

  Permite reemplazar la implementación de un subsistema por un objeto que simule su comportamiento, que implementa la interfaz del mismo.

  Esto se utiliza para test de integración donde se quiere probar de forma independiente del resto de los subsistemas.

- ***Ayuda en las estimaciones***

  El esfuerzo requerido para el desarrollo de un subsistema se puede calcular por ***comparación de subsistemas similares*** en cuanto a cantidad de servicios expuestos o cantidad de componentes internos.

DESCRIPCIÓN GENERAL DE LA DOCUMENTACIÓN

DIAGRAMAS DE ANÁLISIS

- **DION** - Diagrama de interacción con otros negocios: se modelan exclusivamente los requerimientos que un subsistema le hace a los negocios de los cuales necesita colaboración. Es un refinamiento de los DRE de ARQS.

- **DIFZ** - Diagrama de interfaz: se exponen reglas y componentes reusables que el subsistema modelado expone a los otros subsistemas.

DIAGRAMAS DE DyD

- **DAS** - Diagrama de Arquitectura de Subsistemas de diseño (**vista alto nivel**): este diagrama como se puede visualizar en la figura 37, muestra las dependencias de los subsistemas del e-SIDIF, con el subsistema que se está modelando.

  Esta vista será proporcionada por el Arquitecto de subsistemas, quien tiene la visión global de la aplicación.

  *Los **diseñadores** podrán usarlo para ver las dependencias generales hacia ese subsistema.*

  Es tarea del **arquitecto de subsistema** realizar o autorizar la modificación del mismo.

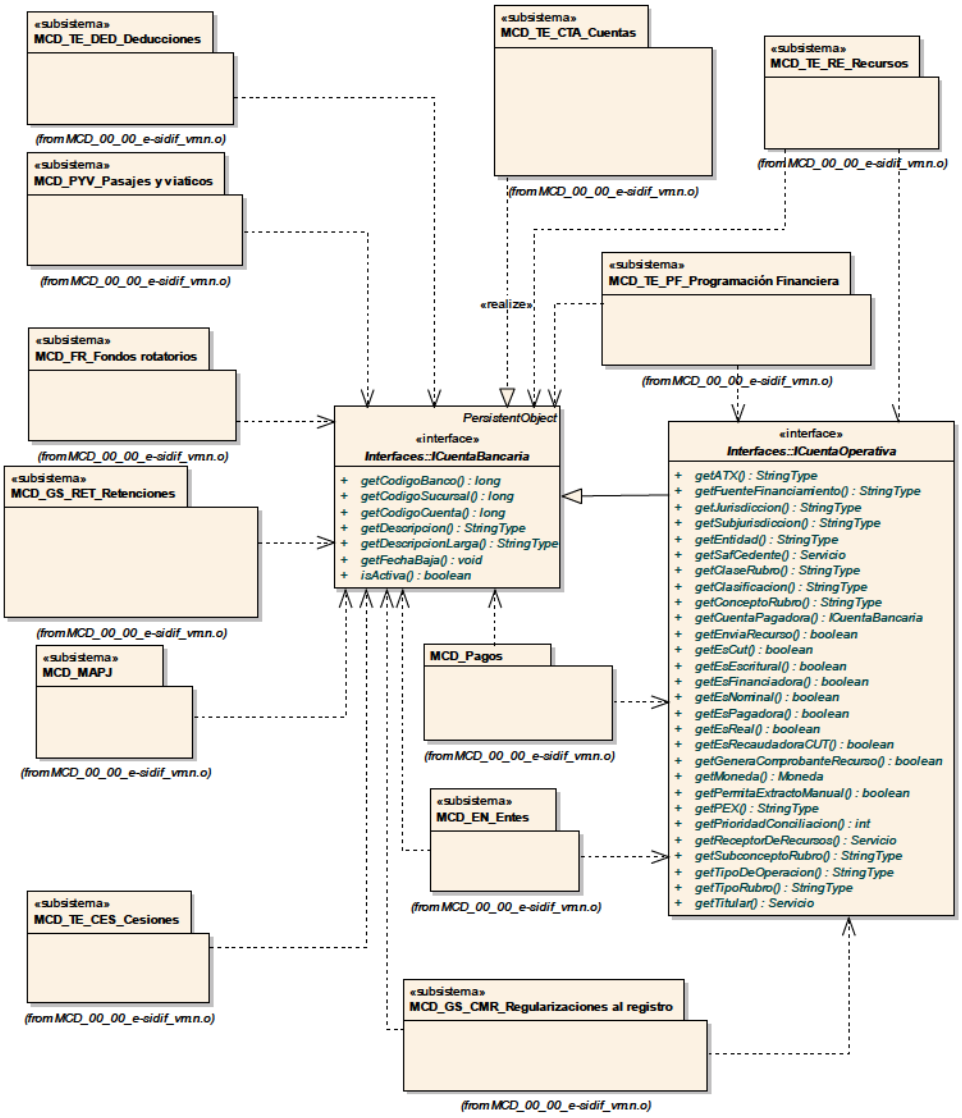


Figura 37- Ejemplo de un DAS

- **DDS -** Diagrama de Dependencias de Subsistemas de diseño (**vista externa**): muestra las dependencias de un subsistema con otros subsistemas y paquetes.

  Sólo muestra la relación del subsistema que se está modelando con otros subsistemas de los que "consume" servicios.

  Este diagrama debe ser mantenido por el diseñador encargado del subsistema en cuestión. En la figura 38, se visualiza un DDS a modo de ejemplo.

Las dependencias entre subsistemas deben darse siempre desde ese subsistema hacia una de las interfaces de otro subsistema del que consume servicios.

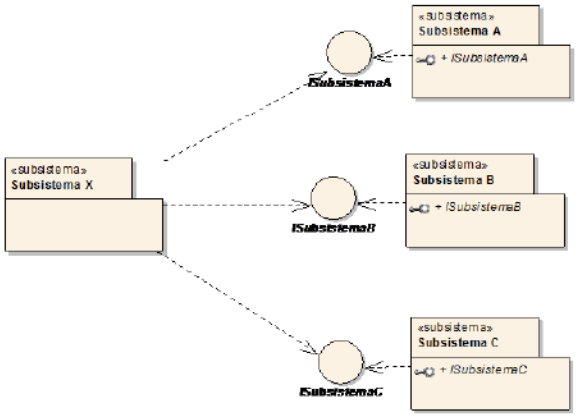


Figura 38- Ejemplo de un DDS:

- **DCS** - Diagrama de Composición de Subsistemas (**vista interna**): muestra los subsistemas y paquetes por los que está compuesto un subsistema, y la forma en que los mismos se relacionan entre sí.

  Este diagrama es opcional ya que solo tendrá sentido para subsistemas complejos que se componen internamente de otros subsistemas o módulos. En la figura 39 se visualiza un ejemplo de un DCS.

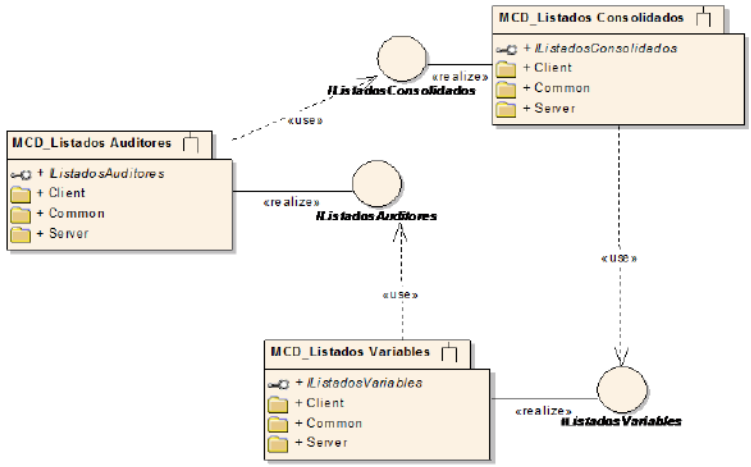


Figura 39- Ejemplo de un DCS:

DISEÑO DE UN SUBSISTEMA

Para diseñar un sub-sistema se pueden seguir distintos principios. A continuación se mencionan algunos ítems a seguir:

- ***Identificar Subsistemas***
  - Agrupa clases
  - Cohesivo
  - Bajo acoplamiento
- ***Definir las interfaces***
  - separan especificación de funcionalidad
  - define la forma en que los sistemas interactúan
  - contrato entre grupos de desarrollo
- ***Encontrar dependencias***
  - dependencia de interfaces
  - minimizar dependencias

NOTAS

[i] *CASE (acrónimo de Computer Aided Software Engineering, traducido al español Ingeniería de Software Asistida por Computadora)* se refiere a distintas aplicaciones, las cuales permiten aumentar la productividad en el desarrollo de SW y a la vez reducir en tiempo y costos del mismo. Estas herramientas están involucradas en todas las partes del ciclo de vida de desarrollo del SW, desde el diseño, cálculo de tiempo y costos, implementación automática de código, test, compilación automática, detección de errores e incluso documentación. Existen diversas herramientas de este tipo, ya sea para determinado lenguaje de programación, y/o para diversos fines. Por mencionar algunos tenemos, Magic Draw UML, Microsoft Vicio, IBM Rhapsody, Snap, System Architect, Power Designer, Enterprise Architecture (una de las utilizadas en el proyecto eSidif).

[ii] *Del acrónimo Microsoft Solutions Software*, es un tipo de metodología de software tradicional desarrollada por Microsoft Consulting Services que define un marco de trabajo de referencia para desarrollar e implementar sistemas empresariales distribuidos. Es una metodología flexible y que su principio fundamental es la de dejar en segundo plano la elección tecnológica. Características principales: adaptabilidad, escalabilidad, flexibilidad y tecnología agnóstica

[iii] *XP (eXtreme Programming, traducido al español Programación Extrema)*, es una metodología ágil centrada en potenciar a las relaciones interpersonales como clave para el éxito en el desarrollo de SW. Su nombre fue acuñado por Beck, debido a que el enfoque fue desarrollado utilizando buenas prácticas reconocidas, como el desarrollo iterativo y con la participación del cliente en niveles "extremos". En XP, todos los requerimientos se expresan como escenarios, denominados historias del usuario, los cuales se implementan directamente como una serie de tareas. Los programadores trabajan en parejas y desarrollan pruebas para cada tarea antes de escribir el código (TDD, Test Driven Development). Todas las pruebas se deben ejecutar satisfactoriamente cuando el código nuevo se integre al sistema. Existe un pequeño espacio de tiempo entre las entregas del sistema. Esta metodología ágil, se considera especialmente adecuada para proyectos donde existen requisitos imprecisos y cambiantes, y donde existe un alto riesgo técnico. En XP existen distintos roles: Programador, Cliente, Tester (encargado de pruebas), Tracker (encargado de seguimiento), Coach (entrenador), consultor, Gestor (Big boss).

[iv] *Crystal Methodologies*, traducido al español *Metodologías Crystal*, se refiere más que a una metodología de desarrollo de SW, se refiere a una FAMILIA de metodologías, ya que, se las subdivide en distintas metodologías dependiendo la cantidad de personas que van a participar en el proyecto. Esta familia de metodologías fue creada por Alistair Cockburn, el cual la creo en base a sus experiencias y análisis de proyectos desarrollo de sw. Además de estar caracterizadas porque la cantidad de personas, se centran en producir la mínima cantidad de artefactos. El desarrollo de sw se considera un juego cooperativo de invención y comunicación,

limitado por los recursos a utilizar. Como dijimos existe una clasificación de acuerdo a la cantidad de personas involucradas en el equipo. Esta clasificación es de acuerdo a colores, y es la siguiente: Crystal Clear: equipos conformados de hasta 8 personas o menos; Crystal Yellow: equipos conformados entre 10 y 20 personas; Crystal Orange: equipos conformados entre 20 y 50 personas; Crystal Red: equipos conformados entre 50 y 100 personas; Crystal Blue: equipos conformados entre 100 y 200 personas.

[v] *DSDM (de su acrónimo en inglés Dynamic System Development Method)*, es una metodología ágil surgida en un Consorcio formado por 17 miembros durante Enero del año 1994. DSDM define 5 fases en la construcción de un sistema: 1) Estudio de factibilidad; 2) Estudio del negocio; 3) Iteración del modelo funcional; 4) Iteración del diseño; 5) Construcción/Implementación. Esta metodología ágil es lo más cercano a los métodos tradicionales, de hecho la implementación de un modelo de DSDM en una organización permite alcanzar lo que en CMM se consideraría un nivel 2 de madurez.

[vi] *ASD (de su acrónimo en inglés Adaptive Software Development*). Su impulsor fue Jim Highsmith el cual en su libro impulsaba un cambio en la filosofía de las organizaciones pasando de la transición de modelo Comando-Control al modelo Liderazgo-Colaboración. Basado en los conceptos de los Sistemas Adaptativos Complejos relacionado con la IA(Inteligencia Artificial), Highsmith lleva los mismos al campo de la IS en general. Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje.

[vii] *FDD (de su acrónimo en inglés Feature -Driven Development)* es también una metodología ágil que surge de la colaboración de distintas personas: por un lado, Peter Coad (uno de los referentes más importantes dentro de la IS), por otro Ed Yourdon (uno de los creadores del Análisis Estructurado), y por último Jeff de Luca y otros, son los que dan origen a esta metodología alrededor del año 1998. FDD, define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas) y se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software.

[viii] *LD (de su acrónimo en inglés,Lean Development)* fue definida por Bob Charette's a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

[ix] *Jeff Sutherland* desarrolló la metodología conocida como Scrum en el año 1993 y junto con Ken Schwaber formalizaron esta metodología en OOPSLA'95. Ambos extendieron y mejoraron Scrum en muchas empresas de software y ayudaron a escribir Manifiesto ágil.

Tiene su blog en el cual revisa los orígenes y los procedimientos recomendados de Scrum. La dirección URL del blog es http://scrum.jeffsutherland.com.

[x] *Ken Schwaber* es uno de los desarrolladores de Scrum, se basó más en la parte formal de la metodología.

[xi] Suceso o acontecimiento que sirve como punto de referencia.

[xii] *JEE*, es el acrónimo de Java Enterprise Edition que traducido informalmente al español es Java Empresarial. Anteriormente conocido como Java 2 Platform Enterprise Edition(J2EE) hasta la versión 1.4. JEE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java.

[xiii] *J2SE*, es el acrónimo de Java 2 Standart Edition, que traducido al español es Java Estándar. J2SE fue nombrado de esa forma hasta la versión 5.0 Y luego se pasó a nombrar JSE.

[xiv] *Feature (característica)* hace referencia a una característica propiamente dicha del producto.

[xv] Es decir, que es de código abierto, y por ende, es una herramienta gratuita que se pude descargar y utilizar sin ningún tipo de licencia, y de esta forma adecuarla para la utilización en el proyecto que se requiera.