



img/LOGO-eps-converted-to.pdf

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

COMUNIC@

Un ejemplo práctico de desarrollo ágil

Realizado por

**ELENA ALEJO TEISSIÈRE
30242147-E**

**IRENE ALEJO TEISSIÈRE
28814859-E**

Dirigido por

**ANTONIO RUIZ CORTÉS
JOSÉ MARÍA GARCÍA RODRÍGUEZ**

Departamento

LENGUAJES Y SISTEMAS INFORMÁTICOS

Sevilla, noviembre de 2011

A nuestros padres que, a pesar de todas las dificultades, siempre creyeron en nosotras.

Agradecimientos

Deberíamos agradecer a tantas personas el hacer posible la finalización de este proyecto, con el que culmina una de las épocas más importantes de nuestras vidas, que los agradecimientos terminarían ocupando casi más que el resto de la memoria. Por lo tanto, vamos a intentar resumir lo máximo posible.

Queremos darle las gracias primero a nuestro tutor, quien siempre se ha mostrado disponible para nosotras, y que ha tenido que hacer su labor persiguiéndonos para que terminemos cuanto antes esta losa que llevamos arrastrando (alguna un poco más de tiempo que la otra). Gracias a su apoyo y ayuda, este proyecto está viendo la luz.

También queremos agradecer a nuestra familia y amigos, sobre todo por su comprensión por haber estado tan absorbidas durante tanto tiempo. Sabemos que nos habéis echado de menos, aunque seguro que no tanto como nosotras, que hemos pasado fines de semana enteros enclaustradas como ermitaños sin ver la luz del sol. Gracias en especial a Puri por sus mimos y galletas, y a Chiara y Rafa por entender que no estuviéramos ahí cuando nos necesitaban. A la familia, porque casi olvidan nuestras caras; y al resto por su paciencia y ánimos. Os queremos.

Gracias a nuestros compañeros de trabajo, que se han esforzado para que el desarrollo de este proyecto sea compatible con el trabajo, a pesar de ir a veces en detrimento de las exigencias laborales.

Gracias a Alberto y a Carlos del Centro Avanzado en Tecnologías Aeroespaciales, defensores a ultranza del desarrollo ágil y que, día tras día, demuestran que otra forma de hacer las cosas es posible. Encontraros ha sido toda una suerte. Gracias al equipo de ANIMO, que han enseñado lo duro y gratificante que es ser Scrum máster.

Gracias a los compañeros de Everis por aguantar la paliza diaria al hablar de las maravillas del desarrollo ágil, que generaba un debate que alegraba esos días de trabajo que parecían nunca acabar. Y al jefe de sección, por cambiar las horas extra por todos los días libres que corresponde por ley para terminar este proyecto.

A Chicho, Ichi, Misa-Misa y Ryuk, por darnos calorcito y compañía durante esas largas jornadas en casa. Tenemos suerte de tener unos gatitos como vosotros. A Ronco, por sacarnos a pasear y obligar a que la sangre volviera a circular por

las piernas; no hay perro como tú.

Y como agradecimiento especial, a Pablo y Sergio, que han aguantado nuestras jaquecas, enfermedades, malos humores, quejas, lloriqueos y, encima, han realizado las tareas del hogar en las épocas de más agobio. Sin vuestra ayuda y comprensión nunca habríamos tenido la fuerza necesaria para acabar a tiempo. Gracias por estar ahí, os queremos.

Resumen

“Lo simple puede ser más difícil que lo complejo: debes trabajar duro para mantener el pensamiento claro y hacer las cosas simples. Pero al final vale la pena porque, una vez que lo consigas, puedes mover montañas.” – Steve Jobs.

Las actuales características de dinamismo y variabilidad de la industria de software han precisado replantear los cimientos sobre los que se sustenta el desarrollo de software convencional. Un reciente estudio realizado por Boehm en [?], sobre la tendencia en ingeniería del software, indica que el mercado actual está caracterizado por el rápido desarrollo de aplicaciones y la reducción de la vida de los productos. Ante esta situación, cabe reflexionar sobre el grado de adaptación de las metodologías convencionales a estas circunstancias. La mayoría de los estudios coinciden en que el carácter normativo y la fuerte dependencia de planificaciones previas al desarrollo que definen las metodologías convencionales hacen que resulten excesivamente pesadas para cubrir las necesidades de un amplio porcentaje del mercado de software actual. Ante este panorama, las empresas se están interesando en el desarrollo con metodologías ágiles. Uno de los escollos que se deben superar es la indicación del porcentaje de beneficio que obtendrían estas empresas en cuanto a productividad si aplican estas tecnologías. Está demostrado que para pequeños proyectos es muy beneficioso, pero es difícilmente escalable a proyectos de gran envergadura, aunque grandes empresas como Google lo estén llevando a cabo ¹.

El objeto de esta investigación es estudiar la evolución de un producto de software concreto utilizando las directrices marcadas por las metodologías ágiles, en concreto por la metodología Scrum [?] y BDD [?]. Se presentan los resultados obtenidos en aspectos tales como las características del producto a lo largo de la evolución, incluidas las estimaciones de la calidad del producto obtenido, la agilidad en el desarrollo y la evaluación del esfuerzo dedicado a adoptar la metodología. Además, dado que el factor humano es fundamental en este tipo de metodologías, se presenta un análisis cualitativo del desarrollo del proyecto.

Este proyecto trata de comprobar, con un ejemplo práctico, cómo estas metodologías mejoran la calidad del software y los costes.

Se ha decidido realizar una aplicación a la que se denominará Comunic@, en la que se simulará que su objetivo final es formar parte de la intranet de una

¹ <http://video.google.com/videoplay?docid=8795214308797356840>

empresa. Con esto se pretende hacer más realista el desarrollo del producto.

Se creará esta aplicación utilizando metodologías ágiles. En concreto, se han usado las metodologías estudiadas y explicadas en este documento como Scrum, BDD y Pair Programming. Al final del proyecto, se realizará una retrospectiva final explicando las conclusiones a las que se ha llegado, los pros y contras de la utilización de estas metodologías y lo aprendido durante el desarrollo de la aplicación.

Índice general

I	Prefacio	3
1.	Introducción	5
1.1.	Motivación	7
1.2.	Objetivos del proyecto	8
1.2.1.	Objetivos orientados a la metodología	8
1.2.2.	Objetivos orientados a la técnica	9
1.2.3.	Objetivos personales	10
1.3.	Estructura del documento	10
II	Conceptos básicos	13
2.	Introducción a las metodologías ágiles	15
2.1.	El Manifiesto Ágil	19
3.	Cómo ser ágil.	23
3.1.	¿Qué es el Behaviour-Driven Development (BDD)?	23
3.2.	Programación por parejas	27
3.3.	¿Qué es el Scrum?	28
3.3.1.	Principios de Scrum	29
3.3.2.	Artefactos de Scrum	29
3.3.3.	Reuniones en Scrum	33
3.4.	Principios S.O.L.I.D.	38
3.4.1.	Principio de una sola responsabilidad	39
3.4.2.	Principio de abierto/cerrado	39
3.4.3.	Principio de substitución de Liskov	39
3.4.4.	Principio de la segregación de la interfaz	40
3.4.5.	Principio de inversión de dependencias	40
3.5.	Integración Continua	40
3.5.1.	Conceptos	41
3.5.2.	Empezando con IC	41
4.	Agilismo en Comunic@	43

III Sistema a desarrollar	47
5. Planificación temporal y costes	49
5.1. Planificación temporal y costes en Scrum	49
5.2. Planificación temporal y costes en Comunic@	55
5.2.1. Planificación inicial	55
5.2.2. Desviaciones con respecto a la planificación	57
6. Arquitectura básica	59
6.1. Patrón Modelo-Vista-Controlador	60
6.2. Patrón DAO	63
6.3. Servicios Web. Arquitectura orientada a servicios.	64
6.3.1. RPC	64
6.3.2. REST	67
7. Historias de usuario	71
8. Planificación y generación de la pila de productos	77
8.1. Sprint Inicial	77
9. Diseño e implementación del sistema	85
9.1. Desarrollo iterativo	85
9.2. Primer modulo: Preparación y gestión de usuarios	88
9.2.1. Historias realizadas en este módulo	90
9.2.2. Diagrama de Burndown	92
9.3. Segundo modulo: Operaciones con comunicados	93
9.3.1. Historias realizadas en este módulo	100
9.3.2. Diseño de Clases	106
9.3.3. Diagrama de Burndown	108
9.4. Tercer modulo: Administración de comunicados	109
9.4.1. Historias realizadas en este módulo	110
9.4.2. Diseño de Clases	115
9.4.3. Diagrama de Burndown	119
9.5. Cuarto modulo: Comunicaciones mediante Servicios Web	119
9.5.1. Historias realizadas en este módulo	120
9.5.2. Diseño de Clases	126
9.5.3. Diagrama de Burndown	127
9.6. Quinto modulo: Aplicación de Android	128
9.6.1. Historias realizadas en este módulo	129
9.6.2. Diseño de Clases	137
9.6.3. Diagrama de Burndown	138
9.7. Sexto módulo: Notificación, paginación y búsqueda de comunicados	139
9.7.1. Historias realizadas en este módulo	140
9.7.2. Diseño de Clases	143
9.7.3. Diagrama de Burndown	144
9.8. Pruebas de Integración	145

IV Conclusiones	149
10. Retrospectiva final de proyecto	151
11. Objetivos cumplidos	155
11.1. Objetivos orientados a la metodología	155
11.2. Objetivos orientados a la técnica	156
11.3. Objetivos personales	156
V Apéndices	159
A. Manual de usuario	161
A.1. Perfil Usuario	161
A.1.1. Alta de Usuarios	161
A.1.2. Menú de Inicio	164
A.1.3. Editar Perfil	165
A.1.4. Subir un fichero	166
A.1.5. Comunicados: Anuncios y Sugerencias	166
A.1.6. Salir	172
A.2. Perfil Administrador	173
A.2.1. Menú de inicio de administración	173
A.2.2. Ver lista de usuarios	174
A.2.3. Usuario Conectados	177
A.2.4. Comunicados: Anuncios y Sugerencias	178
A.2.5. Responder un comunicado	180
B. Glosario de Terminos	183
Bibliografía	185

Índice de figuras

2.1. Impacto de riesgos en desarrollo en cascada.	17
2.2. Análisis de riesgos en desarrollo iterativo.	18
2.3. Valores de las metodologías ágiles.	19
3.1. El algoritmo de BDD [?].	26
3.2. Diagrama de BurnDown del proyecto Comunic@.	32
3.3. Diagrama de BurnDown del proyecto Comunic@.	33
3.4. Iteración en Scrum. http://www.ted2.com.au/services/app-development/ 34	
5.1. Tira cómica y realista hecha por Alex Gorbachev	51
5.2. Cartas para realizar planning poker.	53
5.3. Diagrama de Gantt antes de comenzar con el proyecto.	56
6.1. Ejemplo de patrón MVC. [?]	63
6.2. Estructura de mensaje SOAP de Silver Spoon Sokpop.	65
6.3. Captura del response de una petición rss en Google News.	69
8.1. Factores que miden la prioridad de una historia.	78
8.2. División de una historia de usuario en varias tareas.	83
9.1. Ciclo de vida de una iteración. Propiedad de LIFE	88
9.2. Pantalla de acceso.	92
9.3. Diagrama de Burndown. Sprint 1.	93
9.4. Pantalla que muestra un listado de anuncios	104
9.5. Pantalla que muestra el detalle de un anuncio	105
9.6. Pantalla para la creación de un nuevo anuncio	106
9.7. Relación entre clases de las operaciones básicas con comunicados.108	
9.8. Diagrama de Burndown. Sprint 2.	109
9.9. Pantalla para responder a un comunicado	113
9.10. Pantalla para administrar un comunicado	115
9.11. Relación entre clases de las operaciones de cambio de estado de comunicados.	117
9.12. Relación entre clases de la operación de respuesta de comunicados.118	
9.13. Diagrama de Burndown. Sprint 3.	119
9.14. Wsdl de Comunic@.	124
9.15. Petición REST de anuncios	125
9.16. Relación entre clases de las operaciones de publicación por SW. .	127
9.17. Diagrama de Burndown. Sprint 4.	128

9.18. Emulador de Android.	130
9.19. Eclipse con plugin de Android.	131
9.20. Pantalla inicial de la aplicación.	133
9.21. Listado de Anuncios en Comunic@ndroide.	135
9.22. Comunic@ndroide en funcionamiento.	137
9.23. Relación entre clases de las operaciones lectura de comunicados para Android.	138
9.24. Diagrama de Burndown. Sprint 5.	139
9.25. Búsqueda de comunicados	142
9.26. Relación entre clases de las operaciones de búsqueda de comu- nicados.	144
9.27. Diagrama de Burndown. Sprint 6.	145
A.1. Pantalla de inicio	162
A.2. Alta de Usuario	164
A.3. Menú de inicio, usuario	165
A.4. Subir fichero, usuario	166
A.5. Búsqueda Comunicados, usuario	167
A.6. Listado Comunicados, usuario	169
A.7. Detalle Comunicado, usuario	171
A.8. Nuevo Comunicado, usuario	172
A.9. Nuevo Menú Inicio, administración	174
A.10. Menú de administración de usuarios	175
A.11. Alta Administrador, administración	176
A.12. Barra de Herramientas del Administrador, administración	178
A.13. Respuesta a un comunicado, administración	180

“La mayoría del software actual es muy parecido a una pirámide egipcia, con millones de ladrillos puestos unos encima de otros sin una estructura integral, simplemente realizada mediante fuerza bruta y miles de esclavos” – Alan Kay

Parte I

Prefacio

Capítulo 1. Introducción

“Para investigar la verdad es preciso dudar, en cuanto sea posible, de todas las cosas.” – René Descartes.

Hoy en día, en la industria, la globalización es un hecho. Las empresas tienen que realizar sus productos en un marco cada vez más competitivo. La aparición continua de nuevos productos y servicios, cuyo ciclo de vida se acorta, obliga a incrementar la productividad y disminuir el tiempo de reacción, adaptarse rápidamente a las variantes necesidades de los clientes y, en definitiva, aumentar la capacidad de competir en un mercado mucho más amplio.

Sobre el desarrollo de software, además, hay que tener en cuenta otros factores. Mientras algunas áreas de la ingeniería como la construcción de edificios, de puentes o de caminos son centenarias e incluso milenarias, la edad del software es de apenas medio siglo. Hasta ahora se ha aplicado una metodología similar a las de las ingenierías centenarias, heredada de la industria de la construcción y manufactura, para abordar los proyectos de software, cuya idea base es construir a partir de unos planos bien definidos. Sin embargo, el software padece una extraña enfermedad: su envejecimiento está mil veces más acelerado que el de esas ingenierías de las cuales se ha copiado el modelo de desarrollo.

Como ejemplo del fracaso de estos modelos tradicionales, de un estudio del proyecto Standish ¹ se extrae la siguiente estadística:

- ★ Porcentaje de proyectos que son cancelados: 31 %
- ★ Porcentaje de proyectos problemáticos: 53 %

¹ <http://blog.standishgroup.com/>

★ Porcentaje de proyectos exitosos: 16 %

Debido a toda esta situación, nos preguntamos si el modelo actual se adapta a lo que solicita el mercado. ¿Son las metodologías ágiles la solución?

Según Kent Beck, uno de los padres del modelo ágil, *“cada cosa en software cambia. Los requisitos cambian, el diseño cambia, el negocio cambia, la tecnología cambia, el equipo cambia y los miembros del equipo cambian. El problema no es el cambio, porque el cambio, inevitablemente, va a ocurrir. El problema, realmente, es nuestra incapacidad para hacer frente a los cambios”*[?]. Evidentemente, se puede concluir que el mercado de software está siempre en continua variación y que estamos obligados a abordar estos cambios con nuevas alternativas que nos permitan adaptarnos.

En este contexto las metodologías ágiles aparecen como una atractiva opción pues, en contraposición con las metodologías convencionales que actúan basadas en principios de estabilidad y control del contexto, las metodologías ágiles no se centran en habilidades de predicción, ni pretenden tener un sistema perfectamente definido como paso previo a su construcción, sino que perciben cada respuesta al cambio como una oportunidad para mejorar el sistema e incrementar la satisfacción del cliente considerando la gestión de cambios como un aspecto inherente al propio proceso de desarrollo de software y permitiendo, de este modo, una mejor adaptación en entornos turbulentos.

En este proyecto fin de carrera se presenta el estudio realizado sobre el comportamiento de las metodologías ágiles en la evolución de proyectos de software. Concretamente se usará el marco de trabajo Scrum y la programación orientada a comportamiento (BDD) durante el desarrollo de un producto concreto. Se explicará el proceso desde la concepción hasta la entrega final, siguiendo el marco de trabajo Scrum para la gestión de dicho proyecto, basándonos en nuestras propias experiencias profesionales y con la suerte de haber participado en un curso de certificación de la Scrum Alliance.

Concretamente, el proyecto Comunic@ pretende servir como ejemplo práctico de un desarrollo ágil, que pueda valer de iniciación en esta metodología y que ayude a entender este gran desconocido llamado “Agile”.

Se va a seguir el marco de trabajo Scrum. Para ello, se han realizado todas las reuniones necesarias con el dueño del producto, figura encarnada por el tutor del proyecto, que hará el papel de representante de la empresa ficticia

interesada. Se han utilizado historias de usuario que se han dividido en tareas, se han mostrado demos y, durante todo el proyecto, se puede asegurar que los requisitos cambiantes han estado a la orden del día.

En cuanto a la programación, esta se ha desarrollado conforme a las directrices de XP: escribiendo los test antes que el código (BDD), refactorizando y realizando, cuando era posible, programación por parejas. Se pretende probar cómo, siguiendo estas pautas, se consigue un código flexible y de mejor calidad capaz de adaptarse a los cambios.

1.1. MOTIVACIÓN

La idea de este proyecto comenzó a tomar forma cuando se empezó a trabajar en empresas dedicadas al sector de la informática. Con esta experiencia se observó que el cliente es un ente especial y caprichoso que todo lo quiere sin querer implicarse. Se descubrió que, para los gestores de proyectos, el único objetivo es vender los proyectos a toda costa aunque la vida privada de sus programadores se vieran mermadas o destruidas. No les importa lo bien hecha que esté la aplicación, quieren que el mantenimiento sea inconmensurable para ganar más dinero. Y siempre se podrá recompensar al cliente de otra manera.



Ante esta terrible visión de la programación, no dándonos por vencidas, y gracias al azar, se descubrió esta metodología que, con su lógica, intenta restaurar nuevos pilares para una programación lógica y fácil, con implicación de todas las partes, para realizar un software que funcione, de mantenimiento fácil y, por tanto, crear la figura de cliente feliz.

Porque sin clientes y usuarios felices, la informática no tiene sentido.

Puesto que la universidad forma a los futuros ingenieros informáticos como hizo con nosotras, se considera importante que se le dé dentro de esta institución, un hueco en donde enseñarse, al igual que se hace con metodologías provenientes de otras ingenierías, como lo es el modelo en cascada.

Con este proyecto, se darán a conocer estas metodologías, explicando sus pros y contras, comparándolas con los modelos actuales y haciendo una reflexión de hacia qué camino se está dirigiendo el desarrollo informático.

Se espera que, tanto si se conocen las metodologías ágiles en las que se va a profundizar como si no, disfruten de este proyecto y consigan entender o extender conocimientos sobre estas técnicas que en España son unas tremendas desconocidas, mientras que en Europa están a la orden del día.

1.2. OBJETIVOS DEL PROYECTO

El objetivo general de este proyecto es analizar la evolución de productos de software en el contexto de utilización de metodologías ágiles. Se realizará mediante un ejemplo práctico, por lo que se pretende estudiar la evolución del producto ficticio Comunic@.

Explicaremos el proceso desde la concepción hasta la entrega final, siguiendo el marco de trabajo Scrum para la gestión de dicho proyecto, basándonos en nuestras propias experiencias profesionales y lo aprendido en cursos de Scrum.

img/CircuitoMarquesina-eps-converted-to.pdf

Para hacerlo más real, Comunic@ es una aplicación web con un tablón de anuncios y sugerencias para una empresa ficticia. En este tablón virtual vamos a poder insertar tanto anuncios como sugerencias. Para ello nos reunimos con nuestro cliente, un ejecutivo de la empresa, sección de comunicación. Este papel será realizado por nuestro tutor, de modo que el ejercicio parezca más realista.

A parte de eso, hablando ya sobre el aspecto técnico, se ha aprovechado el proyecto para utilizar tecnologías recientes como Struts 2, Hibernate 3, Spring, comunicación a través de WebServices e, incluso, una aplicación desarrollada para Android que se comunica mediante Rest con el sitio web de Comunic@.

1.2.1. OBJETIVOS ORIENTADOS A LA METODOLOGÍA

Los objetivos que se tienen en mente al realizar esta aplicación con respecto a las metodologías usadas son los siguientes:

- ★ Aplicar el marco de trabajo Scrum, usando para ello los conocimientos ad-

quiridos al trabajar en empresas que utilizan dicha metodología y cursos. También se dispone del apoyo bibliográfico de libros como “Scrum and XP from the Trenches” [?].

- ★ Verificar los costes y mejoras producidos al aplicar dicho marco de trabajo. (Scrum).
- ★ Comprobar los beneficios colaterales a la realización de estas prácticas como, por ejemplo, la facilidad de añadir nuevas tareas durante el proceso de desarrollo.
- ★ Asegurarse de que el cliente quede satisfecho con el producto final y el trato recibido durante el desarrollo.

1.2.2.OBJETIVOS ORIENTADOS A LA TÉCNICA

Los objetivos que se han querido validar al realizar esta aplicación son los siguientes:

- ★ Utilizar procesos ágiles, como eXtreme Programing. Con ello se quiere comprobar que realmente la utilización de estas técnicas mejora la calidad y adaptabilidad del software producido.
- ★ Enfatizar en un aspecto de la metodología XP, BDD. Para realizar este objetivo, se implementarán primero los test y a partir de ellos se realizará la codificación.
- ★ Comprobar una máxima de la programación XP, la legibilidad del código. Para ello, los programadores intercambiarán partes de su código y se programará en parejas. Se asegurará la aplicación de este apartado usando el siguiente lema: “El código es de todos”.
- ★ Estar a la vanguardia con respecto a las nuevas tecnologías. Para ello, se utilizarán tecnologías recientes como: Struct 2, Hibernate 3, Spring y comunicación a través de RestFul.

img/Cir

1.2.3.OBJETIVOS PERSONALES

Se ha querido asegurar que se cumplen los siguientes objetivos a lo largo del desarrollo de la aplicación:

- ★ Adaptación; adecuarse a las exigencias de estas nuevas metodologías. Los desarrolladores tienen la motivación de aprender nuevas técnicas que mejoren la calidad del software. Para ello se tendrán que destruir muchos muros fuertemente contruidos y arraigados de metodologías usadas en otros proyectos. Se comprende el choque que produce esta nueva perspectiva de trabajo, pero con esfuerzo todo se logra.
- ★ Coraje o valentía; como se explica en el libro “Planning Extreme Programming” [?]. Como compromiso personal esta el de adquirir el suficiente coraje como para poder afrontar la realización de este proyecto. Y la suficiente valentía como para no ser derrotado por el camino.
- ★ Respeto; se quiere, durante todo el proyecto, mantener el respeto entre los integrantes del grupo. Esto es esencial para la buena armonía en el desarrollo del proyecto. Sin respeto, el trabajo puede llegar a ser insufrible y, por tanto, se puede perder la motivación.
- ★ Retroalimentación; se quiere aprender de los errores, para ello se realizará una reunión final para echar una mirada atrás. Se repasarán los objetivos conseguidos y fallidos. Para la realización de este objetivo, se utilizará el siguiente dicho popular: “rectificar es de sabios”.

Se tendrán en cuenta estos objetivos durante la realización del proyecto y se comprobará si han sido realizados al final de este, en el capítulo de retrospectiva.

1.3. ESTRUCTURA DEL DOCUMENTO

El presente documento se estructura en las siguientes partes:

En la Parte I se encuentra el presente capítulo, en el que se ha introducido someramente el proyecto realizado junto con los objetivos que se pretenden conseguir con su realización.

La Parte II se centra en introducir al lector en las materias relacionadas con el proyecto. Se hará una pequeña introducción de las metodologías utilizadas para facilitar la comprensión de este. Para finalizar esta parte, se ha realizado un capítulo que describe la aplicación de dichas metodologías a este proyecto en particular.

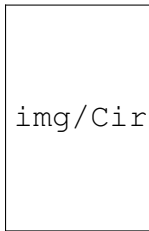
Una vez entrados en materia, la Parte III muestra los resultados de las distintas fases del desarrollo de nuestro sistema de software. Se comienza esta parte con un capítulo de costes. En este capítulo se quieren exponer los costes y la duración del proyecto. Se prosigue con la descripción de la arquitectura utilizada. Además, se hace un breve resumen de los modelos y tecnologías usados.

A continuación, se agregan dos capítulos relacionados con la manera de gestionar el proyecto con Scrum. Para el seguimiento de este marco de trabajo se describen las historias de usuario y las pilas de producto. También se explica la planificación del sprint.

El capítulo nueve trata sobre la implementación del sistema. Se meten “las manos en la masa” y se explica el desarrollo de las historias de usuario mencionadas en capítulos anteriores.

Finalmente, en el capítulo diez se hace una retrospectiva de final de proyecto, donde se comentarán las impresiones generales que han marcado el desarrollo de este proyecto fin de carrera, y se sacarán algunas conclusiones.

Como apéndice se añade un manual de usuario, el cual explica la funcionalidad del aplicativo.

img/Cir

Parte II

Conceptos básicos

Capítulo 2. Introducción a las metodologías ágiles

“En software, muy raramente partimos de requisitos con sentido. Incluso teniéndolos, la única medida del éxito que importa es si nuestra solución resuelve la cambiante idea que el cliente tiene de lo que es su problema.” – Jeff Atwood.

Tradicionalmente se ha tendido a desarrollar software a través de metodologías un tanto rígidas provocando un resultado que, día a día, demuestra su ineficacia dadas las actuales características de dinamismo y variabilidad del mercado con el que se trabaja. Como indica Boehm en la referencia [?], se tiende hacia el rápido desarrollo de aplicaciones y la vida de los productos se acorta. En este entorno inestable, que tiene como factor inherente el cambio y la evolución rápida y continua, la ventaja competitiva se encuentra en aumentar la productividad y satisfacer las variantes necesidades del cliente en el menor tiempo posible para proporcionar un mayor valor al negocio. Sin embargo, las metodologías convencionales presentan diversos problemas a la hora de abordar un amplio rango de proyectos industriales.

Entre estos problemas podemos destacar los siguientes: perciben la captura de requisitos del proyecto como una fase previa a su desarrollo que, una vez completada, debe proporcionar una fotografía exacta de qué desea el cliente. Se trata de evitar a toda costa que se produzcan cambios en el conjunto de requisitos inicial, puesto que a medida que avanza el proyecto resulta más costoso solucionar los errores detectados o introducir modificaciones [?]. Además, pretenden delegar toda responsabilidad económica en el cliente en caso de que estos cambios de requisitos se produzcan. Por este motivo, se les conoce también como “metodologías predictivas”. Sin embargo, el esfuerzo, tanto en coste como en tiempo, que supone hacer una captura detallada de todos los requisitos de un proyecto al comienzo de este es enorme, y rara vez se ve justificado con el resultado obtenido. Además, en muchas ocasiones, el cliente no conoce sus propias necesidades con la profundidad suficiente como para definir las de forma exacta a priori y, a menudo, estas necesidades y sus prioridades varían durante la vida del proyecto.

img/Cir

Establecer mecanismos de control es una de las opciones existentes para protegerse de estos cambios, aunque frecuentemente provocan la insatisfacción de los clientes, que perciben el desarrollo del proyecto como algo inflexible que no se adapta a sus necesidades y que, si lo hace, repercute negativamente en costes añadidos al presupuesto del proyecto. Por otro lado, en muchas ocasiones el proceso de desarrollo convencional está oprimido por excesiva documentación no siempre útil. Un porcentaje elevado del tiempo de desarrollo de un producto de software, desde el punto de vista de las metodologías ágiles, se malgasta en crear documentación que finalmente no se utiliza o queda obsoleta en un corto espacio de tiempo y que, por tanto, no aporta valor al negocio. Además, esta documentación innecesaria entorpece las labores de mantenimiento de la propia documentación útil, lo que provoca que en muchas ocasiones el mantenimiento de la documentación se vea agudizado.

Evidentemente, estas circunstancias no se adaptan a las restricciones de tiempo del mercado actual. Otra dificultad añadida al uso de metodologías convencionales es la lentitud del proceso de desarrollo. Es difícil para los desarrolladores entender un sistema complejo en su globalidad, lo que provoca que las diferentes etapas del ciclo de vida convencional transcurran lentamente. Dividir el trabajo en módulos abordables ayuda a minimizar los fallos y, por tanto, el coste de desarrollo. Además, permite liberar funcionalidad progresivamente, según indiquen los estudios de las necesidades del mercado que aportan mayor beneficio a la organización. En la feroz competencia del mercado vigente, en la que los productos quedan obsoletos rápidamente, básicamente, se pide rapidez, calidad y reducción de costes, pero para asumir estos retos, es necesario tener agilidad y flexibilidad.

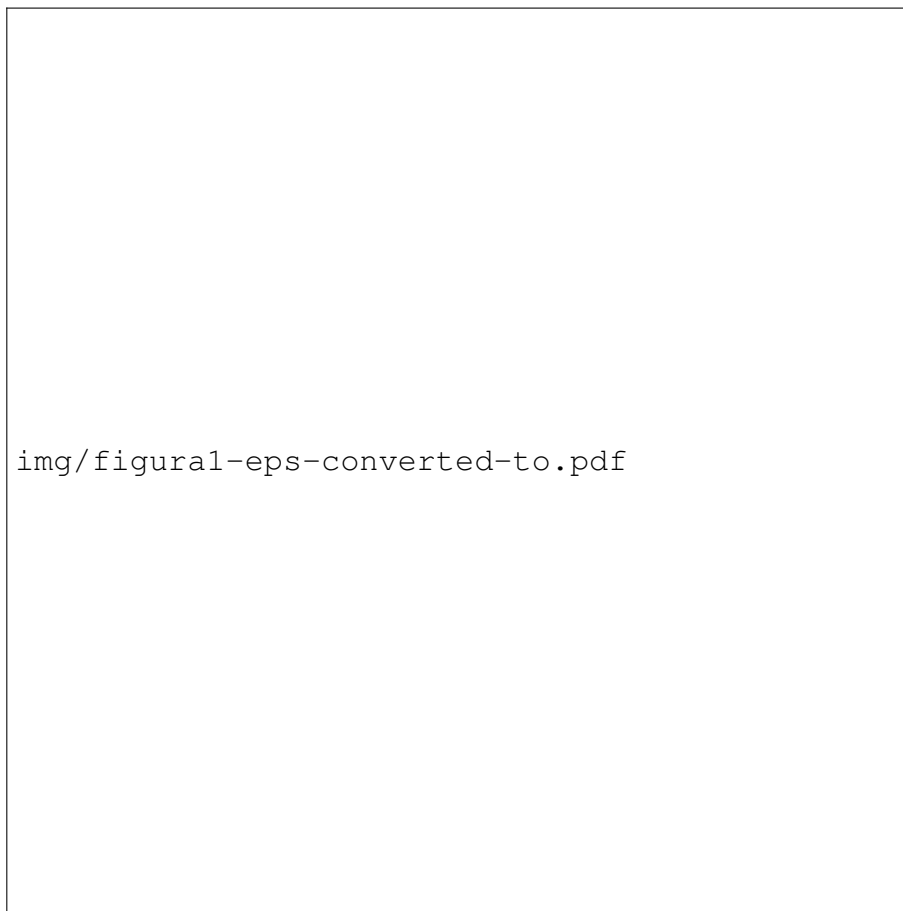
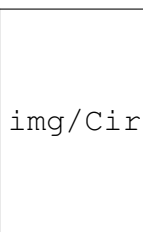


Figura 2.1: Impacto de riesgos en desarrollo en cascada.



Asimismo, las metodologías convencionales tienden a acumular los riesgos y dificultades que surgen en el desarrollo del producto al final del proyecto, como puede apreciarse en la figura 2.1, y conllevan retrasos en la entrega de productos o influyen en la incorrecta ejecución de las últimas fases del ciclo de vida. En contraposición a las metodologías convencionales, las metodologías ágiles aparecen como alternativa atractiva para adaptarse a este entorno. Tal y como se indica en [?] y [?], son apropiadas cuando los requisitos son emergentes y cambian rápidamente. De este modo, presentan diversas ventajas en el contexto actual:

- ★ Capacidad de respuesta a cambios a lo largo del desarrollo, ya que no se perciben como un lastre sino como una oportunidad para mejorar el sistema e incrementar la satisfacción del cliente, considerando la gestión de cambios como un aspecto característico del propio proceso de desarrollo software.
- ★ Entrega continua y en plazos breves de software funcional, lo que permite al

cliente verificar in situ el desarrollo del proyecto, ir disfrutando de la funcionalidad del producto progresivamente y comprobando si satisface sus necesidades, lo cual repercute en una mayor satisfacción. Además, el desarrollo en ciclos de corta duración favorece que los riesgos y dificultades se repartan a lo largo del desarrollo del producto, principalmente al comienzo del mismo, y permite ir aprendiendo de estos riesgos y dificultades (ver figura 2.2).



Figura 2.2: Análisis de riesgos en desarrollo iterativo.

- ★ Trabajo conjunto entre el cliente y el equipo de desarrollo con una comunicación directa para mitigar malentendidos, que constituyen una de las principales fuentes de errores en productos de software y de exceso de documentación improductiva.
- ★ Importancia de la simplicidad, por lo que se prescinde del trabajo innecesario que no aporta valor al negocio.
- ★ Atención continua a la excelencia técnica y al buen diseño para mantener una alta calidad de los productos.

- ★ Mejora continua de los procesos y del equipo de desarrollo, entendiendo que el éxito depende de tres factores: éxito técnico, éxito personal y éxito de la organización.

2.1. EL MANIFIESTO ÁGIL

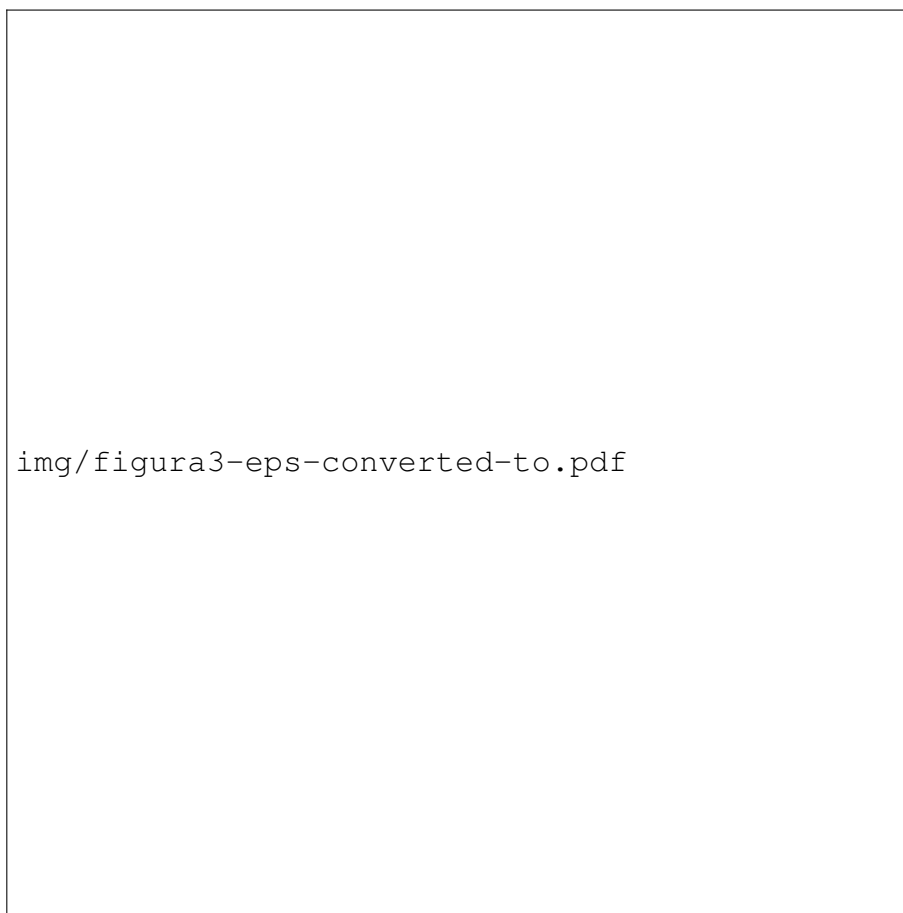


Figura 2.3: Valores de las metodologías ágiles.

Continuamente estamos utilizando el concepto de agilidad para definir estas metodologías, pero ¿qué significa ser ágil desde una perspectiva de software? Basados en el consenso de varias definiciones contemporáneas, Qumer y Henderson-Sellers ofrecieron la siguiente definición de agilidad [?]:

“La agilidad es un comportamiento persistente o habilidad, de entidad sensible, que presenta flexibilidad para adaptarse a cambios, esperados

o inesperados, rápidamente; persigue la duración más corta en tiempo; usa instrumentos económicos, simples y de calidad en un ambiente dinámico; y utiliza los conocimientos y experiencia previos para aprender tanto del entorno interno como del externo.”

Por tanto, el desarrollo ágil no especifica unos procesos o métodos que seguir, aunque bien es cierto que han aparecido algunas prácticas asociadas a este movimiento. El desarrollo ágil es más bien una filosofía de desarrollo de software. El punto de partida se establece en las ideas del Manifiesto Ágil tras la reunión de Utah [?], un documento que resume la filosofía Agile y establece cuatro valores y doce principios.

Según el Manifiesto, se valora:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proceso de software. Este primer valor expresa que es preferible utilizar un proceso indocumentado con buenas interacciones personales que un proceso documentado con interacciones hostiles. Se considera que no se debe pretender la construcción del entorno en primer lugar y esperar que el equipo se adapte automáticamente, sino lo contrario: construir primero el equipo y que este configure su propio entorno. El talento, la habilidad, la capacidad de comunicación y de tratar con personas son características fundamentales para los miembros de un equipo ágil.

Desarrollar software que funcione por encima de una completa documentación. Este valor es utilizado por muchos detractores de las metodologías ágiles que argumentan que estas son la excusa perfecta para aquellos que pretenden evitar las tareas menos gratificantes del desarrollo de software como las tareas de documentación. Sin embargo, el propósito de este valor es acentuar la supremacía del producto por encima de la documentación. El objetivo de todo desarrollador es obtener un producto que funcione y cumpla las necesidades del cliente, y la documentación es un artefacto más que utiliza para cumplir su objetivo. Por tanto, no se trata de no documentar, sino de documentar aquello que sea necesario para tomar de forma inmediata una decisión importante. Los documentos deben ser cortos y centrarse en lo fundamental. Dado que el código es el valor principal que se obtiene del desarrollo, se enfatiza el hecho de seguir ciertos estándares de programación para mantener el código legible y documentado.

La colaboración con el cliente por encima de la negociación contractual. Se propone una interacción continua entre el cliente y el equipo de desarrollo, de tal forma que el cliente forme un tándem con el equipo. Se pretende no diferenciar entre las figuras cliente y equipo de desarrollo, sino que se apuesta por un solo

equipo persiguiendo un objetivo común.

Responder a los cambios más que seguir estrictamente un plan. Planificar el trabajo a realizar es muy útil y las metodologías ágiles consideran actividades específicas de planificación a corto plazo. No obstante, adaptarse a los cambios es vital en la industria del software actual y, por tanto, también consideran mecanismos para tratar los cambios de prioridades. La regla es :

“planificar es útil. Seguir un plan es útil hasta que el plan se distancia de la situación actual. Pender de un plan desactualizado es caminar por un callejón sin salida”.

Para cumplir estos valores se siguen doce principios que establecen algunas diferencias entre un desarrollo ágil y uno convencional:

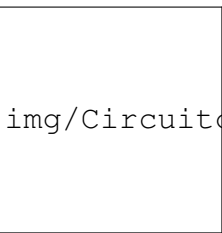
- 1.- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten valor.
- 2.- Dar la bienvenida a los cambios de requisitos. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- 3.- Liberar software que funcione frecuentemente, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
- 4.- Los miembros del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto.
- 5.- Construir el proyecto en torno a individuos motivados. Darles el entorno y apoyo que necesiten 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.
- 7.- El software que funciona es la principal medida de progreso.
- 8.- Los procesos ágiles promueven un desarrollo sostenible. Los promotores,

img/Cir

desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

- 9.- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- 10.- La simplicidad es esencial.
- 11.- Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se organizan ellos mismos.
- 12.- En intervalos regulares, el equipo debe reflexionar sobre cómo ser más efectivo y, según estas reflexiones, ajustar su comportamiento.

Estos principios marcan el ciclo de vida de un desarrollo ágil, así como las prácticas y procesos a utilizar.



img/CircuitoMarquesina-eps-converted-to.pdf

Capítulo 3. Cómo ser ágil.

“Codifica siempre como si la persona que finalmente mantendrá tu código fuera un psicópata violento que sabe dónde vives.” – Martin Golding.

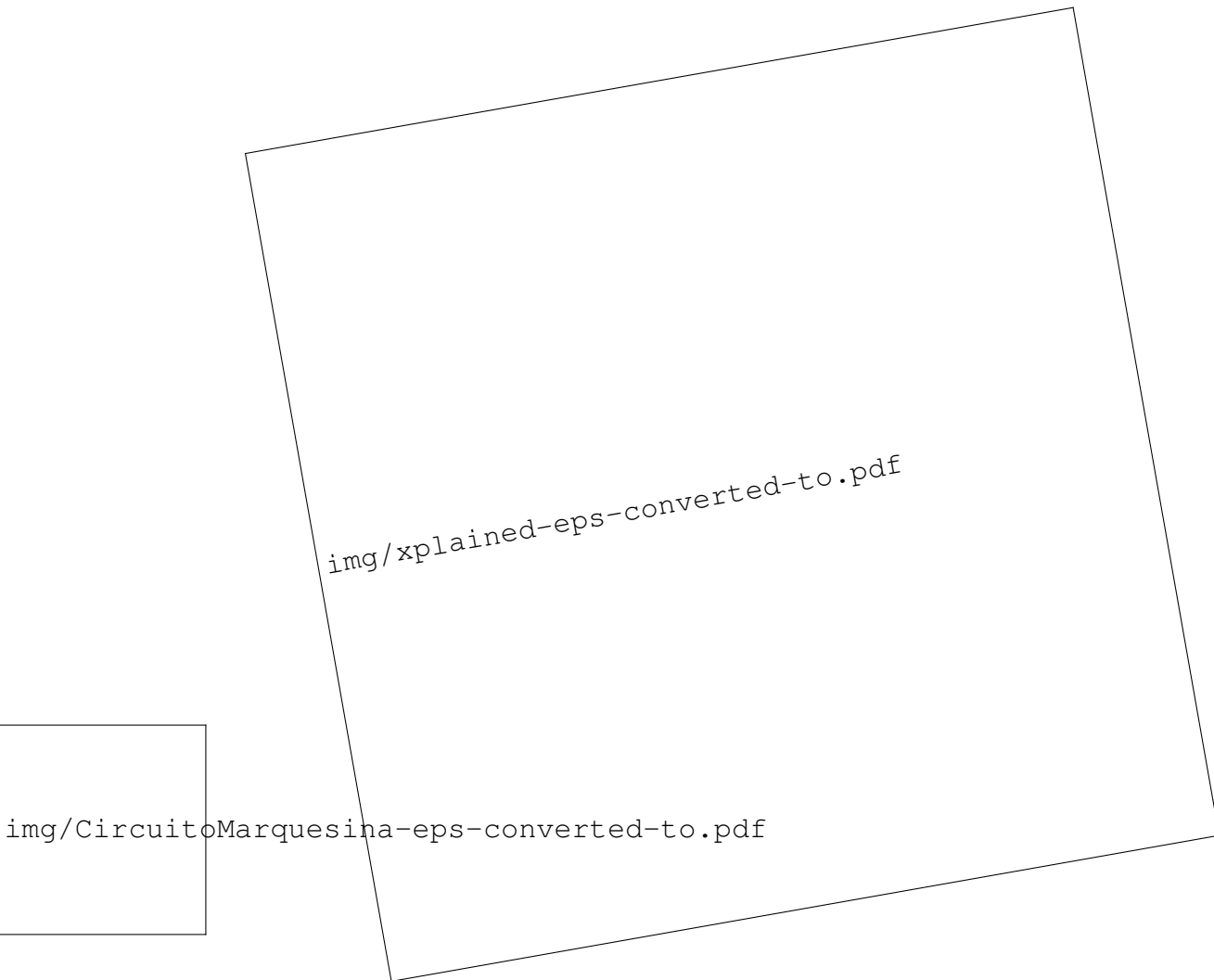
Aunque el movimiento ágil está sustentado por valores y principios para el desarrollo de productos de software, la mayoría de estas metodologías tienen asociadas un conjunto de prácticas, en muchos casos comunes, que buscan la agilidad en el desarrollo. En esta sección vamos a analizar algunas de las más relevantes: Scrum, programación en parejas, integración continua, refactorización y desarrollo dirigido por comportamiento (Behaviour-Driven Development). Comenzaremos realizando una breve descripción de las metodologías usadas en este proyecto.

img/Cir

3.1. ¿QUÉ ES EL BEHAVIOUR-DRIVEN DEVELOPMENT (BDD)?

Es una metodología de XP (eXtreme Programming) para desarrollar software. Su antecesor, y con quién comparte prácticamente todas las características, es TDD (Test-Driven Development). Las bases de esta técnica son:

- 1.- Implementar las funciones justas que el cliente necesita.
- 2.- Minimizar el número de errores que llegan a la fase de producción.
- 3.- Producir software modular para que pueda reutilizarse y modificarse.



El principio de esta técnica es convertir al programador en desarrollador, es la respuesta al cómo lo hago, por dónde empiezo, qué hay que implementar.

Esta técnica trata de diseñar las pruebas adecuadamente según los requisitos. Para empezar, los casos de uso no existen. En su lugar, con BDD, se escriben historias de usuario que se traducen en varios ejemplos llamados “pruebas de aceptación”, que eliminan la ambigüedad del lenguaje escrito.

Otras metodologías de software se preocupan en definir cómo va a ser la arquitectura y las infraestructura. En el BDD, se implementan pequeños ejemplos de comportamiento que hacen emerger la arquitectura que se necesita usar. Lógicamente, no es que haya que despreocuparse de la arquitectura; evidentemente, se tendrán que conocer algunos parámetros para saber si es una aplicación web o móvil, incluso tener una definición general a grandes rasgos de cómo va a funcionar. Simplemente, se eliminan las arquitecturas encima de esas arquitecturas que intentan que todos los proyectos se hagan siempre igual.



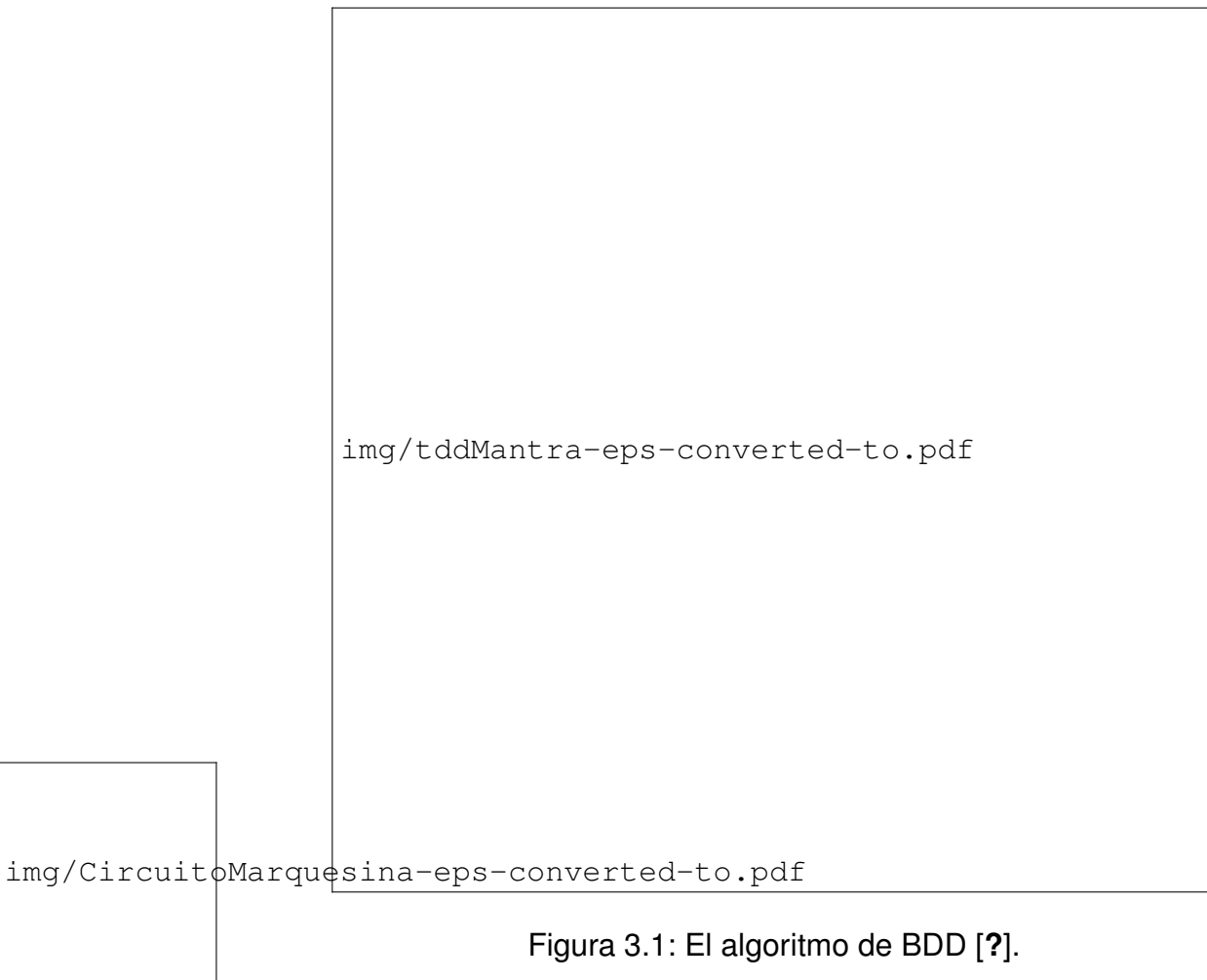
img/tddJoke-eps-converted-to.pdf



img/Cir

Para programar en BDD, el algoritmo tiene tres pasos:

- 1.- Escribir la especificación del requisito. Una vez que tengamos claro cual es el requisito, lo expresamos en forma de código en un ejemplo con XUnit. Explicamos cómo queremos que se comporte un elemento en una determinada situación.
- 2.- Implementar el código a partir del ejemplo. Codificamos lo mínimo necesario para que se cumpla, para que el test pase.
- 3.- Refactorizar para eliminar duplicidad y hacer mejoras. Rastreamos el código (el test incluido) en busca de líneas duplicadas y las eliminamos refactorizando, utilizando algún principio de diseño. Fowler escribió un libro sobre refactorización muy útil[?].



Una pregunta que se suele hacer es si es posible aplicar BDD a proyectos de gran envergadura. Ciertamente, se puede aplicar; simplemente hay que utilizar aquello del “divide y vencerás”. Para saber dividir y priorizar, se suele combinar técnicas de XP como esta con Scrum.

3.2. PROGRAMACIÓN POR PAREJAS

La programación por parejas es una técnica de XP (eXtreme Programming) que consiste en lo siguiente: dos desarrolladores trabajan juntos en el mismo equipo físico, de tal manera que, mientras uno escribe código, el otro supervisa. Esto permite ir desarrollando en conjunto. Ambos aportan ideas de diseño, ambos deciden la solución óptima y el camino a seguir. Beneficia tener cuatro ojos sobre el código, puesto que el desarrollador que no escribe está pendiente en todo momento del cómo se está haciendo y le es más fácil mantener una perspectiva global de qué se quiere hacer.

Es especialmente útil que en las parejas haya miembros con perfiles distintos para que aprendan uno del otro. Por ejemplo, parejas expertas en distintos campos o parejas compuestas por un desarrollador senior y otro junior, para que este último obtenga el conocimiento del senior y aprenda de él sobre la marcha.

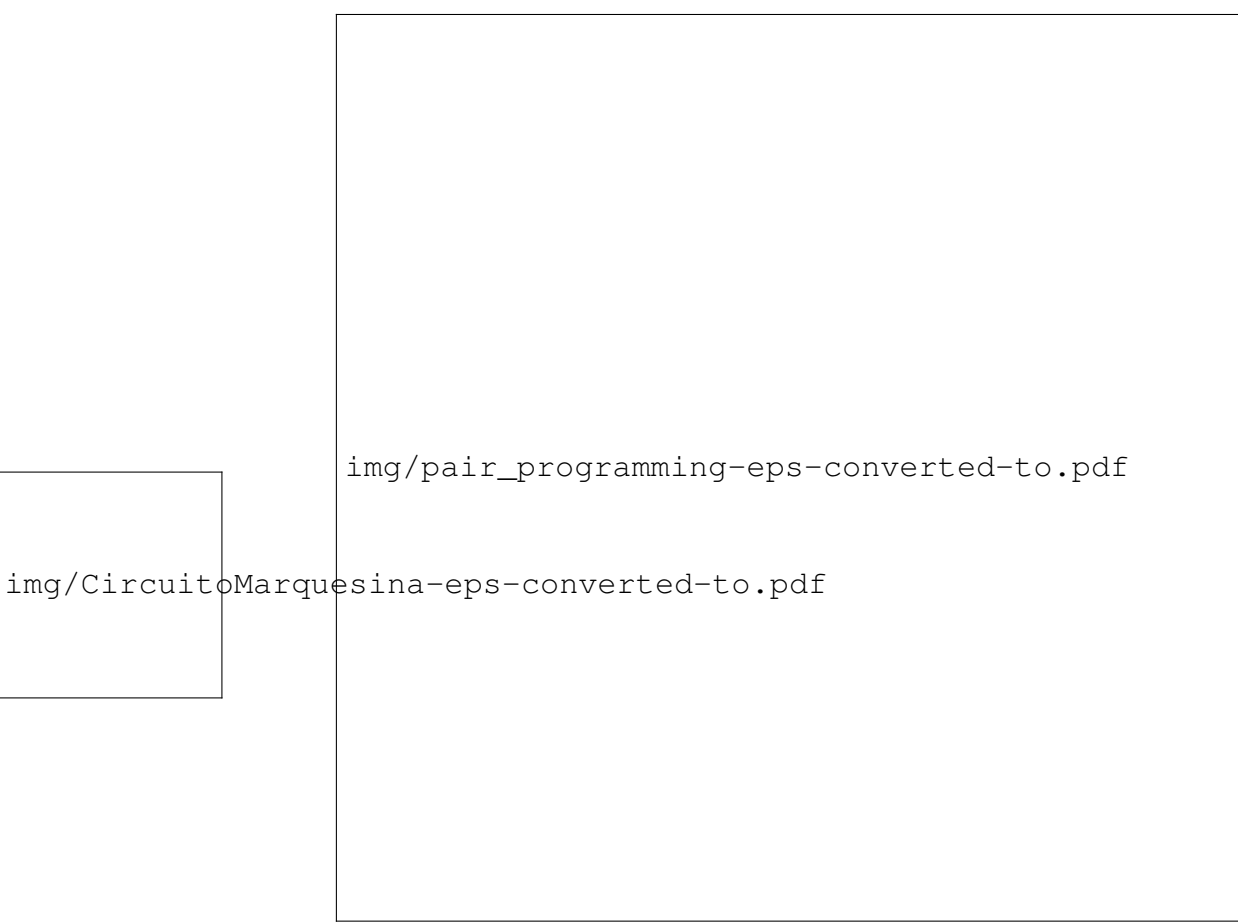
También, cuando llega alguien nuevo al equipo, es mucho más sencillo el traspaso de conocimientos de esta manera. Así, el nuevo miembro, empieza a producir cuanto antes.

Como se ha explicado antes, uno de los desarrolladores escribe el código, y el otro observa de manera activa. No obstante, es imprescindible que, cada cierto tiempo, se cambien los papeles para aumentar la sinergia entre ambos.

Los detractores de este método piensan que es una pérdida de tiempo tener a dos personas trabajando en un mismo equipo, pero se comprueba que las personas tienden a estar mucho más atentas a lo que están haciendo y se reduce el tiempo improductivo y los despistes, ya que adquieren un compromiso el uno con el otro. Aparte, mejora el trabajo en grupo y une al equipo, siempre y cuando se logren solucionar pequeños roces que suelen darse por los desacuerdos que seguro se producirán en las elecciones que deben hacerse durante el desarrollo.

img/Cir

En nuestra experiencia personal, hemos comprobado que este método es especialmente útil cuando hay que enfrentarse a un problema complejo o cuando hay que hacer transferencia de conocimientos que, de este modo, se hace de una manera muy natural, rápida y efectiva. Para codificaciones más sencillas, puede no ser el método más acertado si tratamos con desarrolladores que están dentro del mismo nivel, por lo que sería preferible en este caso el desarrollo individual.



img/pair_programming-eps-converted-to.pdf

img/CircuitoMarquesina-eps-converted-to.pdf

3.3. ¿QUÉ ES EL SCRUM?

Scrum no es una metodología, es un marco de trabajo. Por tanto, Scrum no va a indicar exactamente lo que se tiene que hacer. Scrum forma parte de las metodologías ágiles y está indicada para pequeños grupos de trabajo de hasta ocho programadores. A continuación, se va a realizar un pequeño resumen de qué es el Scrum comenzando por definir conceptos:

3.3.1. PRINCIPIOS DE SCRUM

Scrum basa sus principios en no imponer una forma fija de hacer las cosas. Es un marco de trabajo y pone herramientas que se pueden o no aplicar, en función del sentido que tengan en un determinado grupo de trabajo. Lo más importante para Scrum es el grupo de trabajo, ya que unos desarrolladores contentos e implicados contagian al resto de las partes, incluido el cliente.

A pesar de su flexibilidad, tiene como característica principal que toda actividad de Scrum está acotada en el tiempo. Cada reunión, cada iteración, todo, tiene una fecha y hora de fin. Esto es imprescindible para la buena marcha del proyecto, ya que sin una gran disciplina es imposible poder ser tan flexibles sin caer en el caos.

Scrum no está hecho para todo tipo de proyectos. Sirve para aquellos en los que las dificultades son fuertes, ya sea porque se esté en un entorno desconocido usando técnicas sobre las que no se tiene experiencia o porque la complejidad del problema a resolver sea elevada, siempre que no se sobrepase la línea del caos. Es decir, que no se tenga dominio ni experiencia, sea algo nuevo y desconocido, y, además, la dificultad sea elevada. Por eso, se dice que Scrum surfea sobre la línea del caos.

img/Cir

3.3.2. ARTEFACTOS DE SCRUM

Scrum usa un conjunto de artefactos que sirven para dar forma al marco de trabajo. En esta sección se pasa a describir cada uno de ellos.

ROLES

Los principales roles en los que se categoriza a los individuos son los siguientes:

Desarrolladores: son los que realizan el trabajo duro, los que hacen viable el nuevo sistema.

Scrum máster: es el facilitador del equipo, la figura de líder servil que vela por que los desarrolladores puedan cumplir su trabajo, organizando las reuniones de Scrum, haciendo cumplir los acuerdos adquiridos por el equipo, impidiendo que se les moleste, etc.

Dueño del producto: representa a los clientes y usuarios principalmente; es el punto único de acceso de estos con el equipo de desarrollo y el Scrum máster.

Stakeholder: son los clientes, usuarios, inversores involucrados en el proyecto. Solo participan en las revisiones de sprint (las demos).

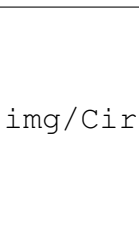
Aparte de esta división, en Scrum se dice que hay dos grupos dentro de los roles. Estos son los *involucrados* y los *comprometidos*.

Al grupo de *involucrados* se les llama también “gallinas”, en realidad no son parte del proceso Scrum, pero deben tenerse en cuenta ya que son los que “ponen los huevos”. Se encuentran en este grupo los stakeholders (usuarios, clientes ...).

Al grupo de *comprometidos* se les llama también “cerdos”, son los verdaderamente importantes y los que “van al matadero” si la cosa no sale. Se encuentran en este grupo el equipo de desarrolladores, el Scrum máster y el dueño del producto.

PILA DE PRODUCTOS O BACKLOG

Es una lista de requisitos, historias de cliente, funcionalidades, etc. Suelen estar definidos en algún documento o programa de gestión en donde se llevará el estado de cada uno de los componentes que forman la pila de productos.



HISTORIAS DE USUARIO

Es un resumen de acciones que debe realizar el programa. Generalmente una historia tiene que tener los siguientes datos:

- ★ **ID:** número identificador de la historia. Sirve también para priorizar entre historias.
- ★ **Nombre:** debe ser lo mas descriptivo posible.
- ★ **Importancia:** un número para indicar la prioridad de la historia.

- ★ **Estimación inicial:** estimación de la dificultad en la realización de una historia.
- ★ **Test de aceptación:** el test de aceptación que resuelve y prueba si la historia está correctamente implementada.

Una vez que se tengan las pilas de productos con las historias de usuarios iniciales, se planifica el primer sprint, que será siempre el primer paso dentro de cada iteración.

DIAGRAMA DE BURNDOWN

Es una gráfica de progreso, cada día vamos actualizando los puntos restantes que nos quedan por implementar. Esta gráfica nos puede ir avisando de cómo va nuestro sprint, en caso de ser muy horizontal la línea de progreso, se debería plantear para el siguiente sprint reducir las historias que se quieren abarcar.

img/CircuitoMarquesina-eps-converted-to.pdf

A continuación vamos a mostrar un diagrama, ver figura que nos indica cuando se pueden añadir más historias de usuario al sprint y cuando debemos de quitar alguna historia porque no se van a realizar todas en el tiempo propuesto.

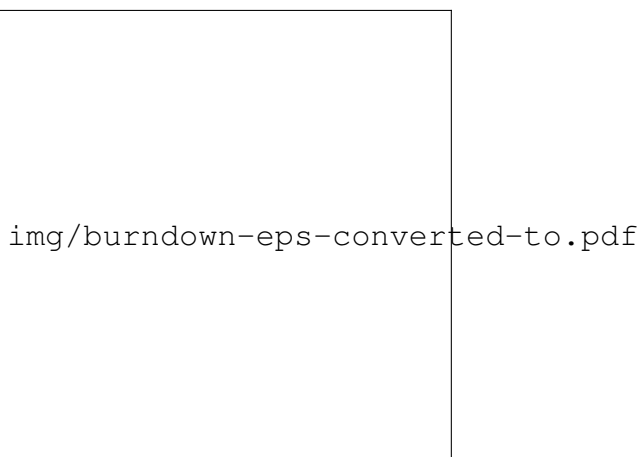


Figura 3.2: Diagrama de BurnDown del proyecto Comunic@.

Se muestra en la figura 3.3, el diagrama de BurnDown del proyecto Comunic@. Se incluye una gráfica de velocidad, en la cual se indican los puntos de

historia implementados por sprint. En la siguiente gráfica, se indican los puntos que faltan por implementarse, los puntos añadidos en los sprint correspondientes. Como se puede observar, la línea se asemeja a la ideal y por lo tanto se puede indicar que la planificación ha sido exitosa.



Figura 3.3: Diagrama de BurnDown del proyecto Comunic@.

img/Cir

3.3.3.REUNIONES EN SCRUM

En cada sprint, se determina un conjunto de reuniones, siempre acotadas en el tiempo, que cumplen con el objetivo de una mejora continua, tanto del producto como de las relaciones interpersonales de los participantes. Son importantes para tener el control de qué y cómo se están haciendo las cosas. Además, previenen desviaciones, ya que su propósito es detectar los problemas a tiempo.

PLANIFICACIÓN DEL SPRINT

Se realiza una reunión, que es crítica: la planificación del sprint. Su función es dar trabajo sin interrupciones durante unas semanas (entre dos y cuatro normalmente).

Aquí se determina una meta de sprint, una lista de miembros, una pila de sprint, una fecha para la demo, y un lugar y una hora para el Scrum diario. Es muy importante que asista el dueño del producto puesto que es él quien determina el alcance e importancia de la tarea. En caso de que no quiera o pueda asistir, el dueño asignará a un componente del grupo la función de mediador y responsable de tomar las decisiones.

¿Cuándo se termina una reunión? En función de la longitud de las iteraciones, la duración puede ser de 2 a 8 horas. Es importante que estas no se prolonguen indefinidamente, puesto que el cansancio puede provocar que se produzcan malas elecciones en el diseño de la pila de sprint. Además, aunque pueda parecer lo contrario, en Scrum se da mucha importancia a que cada actividad esté acotada en el tiempo.



Figura 3.4: Iteración en Scrum. <http://www.ted2.com.au/services/app-development/>

Duración de un sprint Durante la primera reunión de planificación, se debe decidir cuál va a ser la duración de las iteraciones para el proyecto que se va a

llevar a cabo.

Hay que buscar un tiempo medio, puesto que sprint muy cortos provocan mayor agilidad, pero se gasta más tiempo en planificación, puesto que se tienen que realizar más reuniones de sprint.

Si la duración del sprint es mayor, tendremos más tiempo para recuperarnos de posibles imprevistos y menos tiempo de planificación, pero disminuye la agilidad.

Normalmente, la duración varía de dos a cuatro semanas. Hay que recordar que, una vez elegida la duración de un proyecto determinado, se mantendrá hasta el final.

Meta de un sprint Está descrita en términos de negocio para que personas fuera del proyecto puedan entenderlo. Es muy importante definir una meta para saber si hemos tenido éxito o no en la realización del sprint.

img/Cir

Decisión de qué historias se pasan de la pila de productos a la pila de sprint

La pila de sprint estará ordenada por prioridad, el equipo debe indicar cuantas historias podrán realizarse en el tiempo que dura un sprint. Esto no es decisión del dueño del producto, pero si no le satisface la pila de sprint que se ha hecho, puede modificar la disposición de prioridad para meter una tarea que crea conveniente que se haga en un determinado sprint. El dueño de producto tiene acceso libre para modificar la pila de productos, pero no para interferir durante el sprint.

Después de elegir las historias de usuario, el equipo las divide en tareas más pequeñas con el objetivo de llegar a los pasos que hay que dar para que se cumplan los test de aceptación. Se recomienda que las tareas tengan la duración máxima de un día, aunque hasta 48 horas, puede considerarse aceptable. Si alguna tarea termina durando más, puede ser indicativo de que se debía haber dividido en varias.

El número de historias de usuario que nosotros vamos a utilizar es a “ojo”, existen estimaciones de velocidades mediante sprints realizados anteriormente, con estos datos formamos el factor de dedicación.

f. de dedicación = vel. Real (media sprints) / Dis – desarrolladores disponibles.

Días – Desarrollador * factor de dedicación = nº de días disponibles para las historias.

Hora y lugar del Scrum diario Se realizan pequeñas reuniones diarias que no durarán más de quince minutos donde se indica qué se ha hecho y qué se va a realizar. Si alguien se ha encontrado con algún problema que no ha podido resolver, es momento de pedir ayuda al equipo.

Con todos estos puntos aclarados, ya podemos terminar la reunión de sprint. Una vez terminada la reunión de sprint, el Scrum máster, que organiza el grupo de trabajo, actualiza el documento de backlog del Scrum con las historias de usuario que el equipo se ha comprometido a hacer para el sprint que se va a iniciar.

img/Circuitos que una-eps-converted-to.pdf

Comunicación del estado de sprint Es muy importante tener informado al grupo del progreso del sprint, por tanto realizamos un panel de progreso a la vista de todos los integrantes del grupo.

En nuestro trabajo lo hemos organizado de esta manera, similar a la vista en el libro “Scrum y XP desde las trincheras” [?].

Se hace una tabla de tareas y se ponen en el panel de sprint. Las tarjetas blancas representan historias. Las tarjetas amarillas representan tareas para cumplir dicha historia. Se mueven las tarjetas amarillas según se vayan finalizando las tareas de la columna “pendiente”(no se está haciendo esa tarea) a “En curso” y, cuando se terminen, pasan al estado “Terminado”. Cuando se terminan todas las tareas de una historia, se mueve la tarjeta de la historia a “Terminado”. En caso de que se terminen todas las historias, se coge una nueva historia del backlog.

SCRUM DIARIOS

Son reuniones cortas, de alrededor de quince minutos, en las cuales se actualiza el panel y cada componente del equipo indica a su Scrum máster qué tareas va a realizar ese día y qué hizo el día anterior. También es el momento en el que se pide ayuda al resto del equipo si alguien se ha atascado en una tarea particular.

DEMO DEL SPRINT

Es una reunión de importancia que sirve para obtener el feedback del dueño del producto. Se enseña los resultados del sprint al resto de los equipos y al dueño. Es en esta reunión donde se muestra a los demás el progreso que están realizando y, además, se puede reconocer el trabajo que ha realizado el grupo.

Después de esta reunión, se ejecutará una de replanificación, que está definida en la sección de “Planificación del sprint”. Para terminar, el Scrum máster actualiza el backlog o pila de productos y prepara el panel de Scrum para que pueda dar comienzo el nuevo sprint.

RETROSPECTIVAS DEL SPRINT

Se reservan de una a tres horas para meditar el desarrollo del sprint y se comentan los errores cometidos y las cosas que se quieren mantener. Se indican qué errores deben ser resueltos en el próximo sprint, y se invita a una persona fuente que difunda los errores entre otros equipos para evitar que se repitan.

Esta reunión se considera la más importante de Scrum, ya que permite ver los puntos débiles del equipo y poner en práctica métodos para solucionarlos. Dichos puntos débiles no se aplican solamente al modo de trabajar, sino también a las relaciones humanas.

Como ya hemos terminado el sprint, se pueden dejar unos días para descansar, aprender y actualizarse, u obtener algunas certificaciones, aunque esto

no siempre puede hacerse por falta de tiempo.



3.4. PRINCIPIOS S.O.L.I.D.

BDD tiene una estrecha relación con el buen diseño orientado a objetos y, por tanto, con los principios S.O.L.I.D que se comentan a continuación. En el último paso del algoritmo TDD, el de refactorizar, entra en juego la pericia en el diseño de clases y métodos.

Se trata de cinco principios fundamentales, uno por cada letra, que tratan el diseño orientado a objetos en términos de la gestión de dependencias. Las dependencias entre unas clases y otras son las que hacen el código más frágil, o más robusto y reutilizable. El problema con el modelado tradicional es que no se ocupa en profundidad de la gestión de dependencias entre clases sino de la

conceptualización. Quien decidió resaltar estos principios y dar nombre a algunos de ellos fue Robert C. Martin, por el año 2000 [?].

3.4.1. PRINCIPIO DE UNA SOLA RESPONSABILIDAD

Cada clase debería tener un único motivo para ser modificada. Por ejemplo, si estamos delante de una clase que se podría ver obligada a cambiar ante una modificación en la base de datos y, a la vez, ante un cambio en el proceso de negocio, podemos afirmar que dicha clase tiene más de una responsabilidad o más de un motivo para cambiar.

Se aplica tanto a la clase como a cada uno de sus métodos, con lo que cada método también debería tener un solo motivo para cambiar.

3.4.2. PRINCIPIO DE ABIERTO/CERRADO

Una entidad de software (una clase, módulo o función) debe estar abierta a extensiones pero cerrada a modificaciones. Puesto que el software requiere cambios y que unas entidades dependan de otras, las modificaciones en el código de una de ellas puede generar indeseables efectos colaterales en cascada. Para evitarlo, el principio dice que el comportamiento de una entidad debe poder alterarse sin tener que modificar su propio código fuente.

img/Cir

3.4.3. PRINCIPIO DE SUBSTITUCIÓN DE LISKOV

Introducido por Barbara Liskov en 1987, el principio se basa en que los objetos de un programa deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa. Es decir, si un método recibe un objeto como parámetro de tipo X y, en su lugar, le pasamos otro de tipo Y, que hereda de X, dicho método debería funcionar correctamente.

Este principio está estrechamente relacionado con el anterior en cuanto a la extensibilidad de las clases cuando esta se realiza mediante herencia o subtipos.

3.4.4. PRINCIPIO DE LA SEGREGACIÓN DE LA INTERFAZ

Defiende que no obliguemos a los clientes a depender de clases o interfaces que no necesitan usar. Tal imposición ocurre cuando una clase o interfaz tiene más métodos de los que un cliente (otra clase o entidad) necesita para sí mismo. Seguramente sirve a varios objetos cliente con responsabilidades diferentes, con lo que debería estar dividida en varias entidades. Por lo tanto, siempre se implementan todos los métodos de la interfaz y de las clases de las que se hereda.

3.4.5. PRINCIPIO DE INVERSIÓN DE DEPENDENCIAS

Este principio dice que un módulo concreto A no debe depender directamente de otro módulo concreto B, sino de una abstracción de B. Tal abstracción es una interfaz o una clase (que normalmente será una clase abstracta) que sirve de base para un conjunto de clases hijas.

img/CircuitosMarquesina-eps-converted-to.pdf

La inversión de dependencias da origen a la conocida inyección de dependencias, una de las mejores técnicas para lidiar con las colaboraciones entre clases y que produce un código reutilizable, sobrio y preparado para cambiar de manera limpia.

3.5. INTEGRACIÓN CONTINUA

Según Martin Fowler ¹, entendemos la integración continua como:

“Una práctica del desarrollo de software donde los miembros del equipo integran su trabajo con frecuencia. Normalmente, cada persona integra su trabajo al menos diariamente, lo que produce múltiples integraciones por día. Cada integración es comprobada por una construcción automática (pruebas incluidas) para detectar errores de integración tan rápido como sea posible. Muchos equipos encuentran que este enfoque conduce a

¹ <http://www.martinfowler.com/articles/continuousIntegration.html>

la reducción significativa de problemas de integración y permite a un equipo desarrollar software cohesivo más rápidamente.”

Muchos asocian la integración continua (IC) con el uso de herramientas como Cruise o Hudson. Sin embargo, la IC no se basa en estas, aunque son de bastante utilidad. IC es mucho más que la utilización de una herramienta.

IC casa a la perfección con prácticas como BDD dado que se centran en disponer de una buena batería de pruebas y en realizar pequeños cambios que se suben al control de versiones (Mercurial, Subversion. . .). Aunque en realidad, la metodología de desarrollo no es determinante, siempre y cuando se cumplan una serie de buenas prácticas, los beneficios que aporta proporcionan gran valor a las metodologías ágiles. En algunos proyectos, la integración se lleva a cabo como un evento (cada lunes integramos nuestro código...), la IC elimina esta forma de ver la integración, ya que forma parte de nuestro trabajo diario. El resultado es que siempre tenemos un entregable preparado para entregar al cliente.

3.5.1.CONCEPTOS

Construcción (Build): una construcción implica algo más que compilar; suele consistir en compilar el código, ejecutar los test, desplegar en un entorno de preproducción y ejecutar. Un build puede ser entendido como el proceso de convertir el código fuente en software que funcione en las instalaciones del cliente.

Scripts de construcción (build script): se trata de un conjunto de scripts que son utilizados para compilar, testear, inspeccionar y desplegar software. Podemos tener scripts de construcciones sin tener que implementar IC; sin embargo, para IC son vitales.

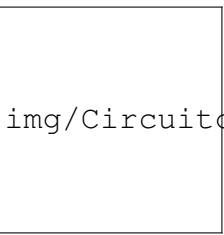
3.5.2.EMPEZANDO CON IC

Para empezar con IC, es necesario tener el código en un repositorio de versiones. Aunque es difícil pensar en proyectos que no utilicen alguna herramienta de este tipo. Se tendrá también un equipo donde se descargarán los cambios cada vez que estos se produzcan y se construirá el proyecto. Será la base para realizar las construcciones cada vez que un desarrollador suba sus cambios. Es-

img/Cir

ta máquina debe contener todo lo necesario para que nuestro proyecto pueda ser construido de forma automática, ya sean scripts, librerías de terceros, ficheros de configuración. . .

Después de construir tanto el ejecutable de las pruebas como el entregable, hay que probar que los test pasan y que el ejecutable funciona. Lo ideal es que eso se realice en una máquina de referencia idéntica a la máquina donde se va a ejecutar, lo que comúnmente se llama, en el ámbito empresarial, un entorno de preproducción.



img/CircuitoMarquesina-eps-converted-to.pdf

Capítulo 4. Agilismo en Comunic@

“Hay dos formas de escribir programas sin errores; solo la tercera funciona.” – Alan J. Perlis.

En los apartados anteriores se ha querido hacer un resumen sobre los fundamentos de las metodologías ágiles. La mayoría de ellas se han implementado en el proyecto. Para aplicar cada metodología, se ha tenido en cuenta los costes en relación al proyecto que se tenía que llevar a cabo.

Es sabido que, para proyectos de mayor envergadura, se tiene que hacer un estudio para saber si estas metodologías son las apropiadas dentro del contexto de dicho proyecto. En el caso de este proyecto, al estar formado únicamente por dos desarrolladores y ser un desarrollo web con cierto nivel de incertidumbre, se ha decidido que cumplía con todos los requisitos. El mayor coste se ha invertido en educar y disciplinar a los desarrolladores. Estos, al ser programadores con experiencia en consultoría, tenían malos hábitos que ha habido que pulir. Una vez superado este escollo, la metodología ha dado buenos resultados puesto que agilizaba la detección de errores y el traspaso de conocimiento entre los dos programadores.

Particularmente, se ha utilizado Scrum de principio a fin, tanto para calcular costes de producción, como para la división del trabajo.

Una de las características del marco de trabajo Scrum, es la existencia de determinados roles; el proyecto cuenta con todos ellos. Se dispone de dos desarrolladores, que son los integrantes que han realizado este trabajo de fin de carrera. Existe una gran diferencia entre ambos desarrolladores en cuanto a sus años de experiencia laboral. Por tanto, el desarrollador con más experiencia tomó el rol de Scrum máster. Además, este desarrollador está trabajando actualmente en una empresa que está utilizando las metodologías ágiles. En dicha empresa le han dado la oportunidad de realizar un curso de Scrum y aprobar la

img/Cir

certificación de ‘Scrum Alliance’. Todos estos detalles han propiciado la elección de este desarrollador como el Scrum máster del proyecto.

El rol de dueño del producto lo han tomado los tutores del proyecto. Han facilitado la descripción del sistema que querían realizar. Es necesario recalcar que su labor ha sido muy realista, puesto que, a medida que se iban realizando las tareas, cambiaba el alcance y la prioridad de estas e, incluso, se añadió más funcionalidad de la que se iba a realizar en un principio.

El rol de “stakeholder” lo encarnan los usuarios de la empresa ficticia y sus directivos. Como no había nadie real que pudiera encarnar este papel, el dueño del producto lo tenía más fácil.

Como se explicará en capítulos posteriores, se ha realizado la reunión de pila de productos, donde se recogieron los requisitos iniciales. También se realizó la primera reunión de sprint donde se organizó el trabajo.

Se han utilizado historias de usuario, las cuales se han dividido en tareas. Se ha realizado un documento de Excel para administrar la pila de productos de Scrum. Puesto que los integrantes del equipo suelen trabajar a distancia, todos tenían acceso a dicho documento. La idea de tener una pizarra física no se consideró productiva.

Por otro lado, hay ciertas partes del marco de trabajo que no se han podido llevar a cabo tajantemente. Al ser este proyecto un trabajo paralelo a otras actividades, no se ha dispuesto del suficiente tiempo como para realizar un Scrum diario. Además, existía la dificultad de que los desarrolladores trabajaban a distancia gran parte del tiempo. Por tanto, un sprint diario no tenía sentido. Se han intentado realizar dos reuniones de Scrum “diario” los fines de semana. Durante la semana, mediante correo o chat se hacía un resumen de las peticiones que se iban realizando. De todos modos, no hay que olvidar que Scrum es un marco de trabajo, no un conjunto de normas sólidas, y que su principal ventaja es que se adapta al equipo.

Por otro lado, los sprints han tenido una duración variable. Debido a determinados picos de trabajo externo de los desarrolladores, estos no podían ofrecer toda la disponibilidad necesaria para terminar los sprints a tiempo. Por tanto, aunque se organizaron en un principio los sprints para que se pudiesen realizar en un número determinado de semanas, al final, con el consentimiento del dueño del producto, se determinó calificar las tareas en hora-persona. Por tanto,

los desarrolladores trabajaban cuando tuvieran tiempo. Las fechas de las demos las ponían los desarrolladores en consenso con el dueño del producto según se fueran desarrollando las tareas.

Esta práctica no es usual puesto que, generalmente, el dueño del producto quiere disponibilidad total y una fecha cerrada en la cual el producto tiene que estar finalizado. Nuestro dueño del producto, insistió en que quería el proyecto para la convocatoria de septiembre, pero, debido a las adversidades ya mencionadas, se decidió pasar la entrega a la convocatoria de diciembre. El resto de las fechas podían ser flexibles.

Como se ha ido explicando en capítulos anteriores, con el uso de las metodologías ágiles viene implícito el uso de ciertas metodologías en el ámbito de la programación.

Para comenzar, se realizó un repaso a los principios S.O.L.I.D. Se educó a los desarrolladores para que realizaran una programación realmente orientada a objetos.

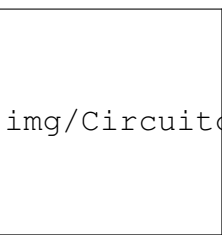
En este documento se han descrito ciertas técnicas de programación como “eXtreme Programing”[?] y “Behaviour-Driven Development”[?]. Durante el desarrollo del proyecto, se ha utilizado BDD. Primero se realiza el test donde se especifica el comportamiento y después se implementa el código. Finalmente, al terminar de realizar el código, este se refactorizaba para que el programa resultante fuese de gran calidad. Esta parte de la programación y el repaso del código se hacía preferentemente por parejas, aunque no se pudo programar en parejas todo lo que se quiso. Los desarrolladores trabajaban la mayor parte del tiempo desde sus casas para ahorrarse el desplazamiento. Para paliar esta situación, se dispusieron de todos los recursos tecnológicos posibles, como teléfono, vídeo conferencia y chat, pero no es tan provechoso programar por parejas a distancia. Por tanto, se optó por trabajar al menos un día a la semana estando físicamente en el mismo lugar y dedicarlo a la programación por parejas.

Aparte, explicamos la utilidad de un entorno de integración continua. Aunque la ayuda que supone implementar este sistema es evidente; por desgracia, para el desarrollo de este proyecto, no se ha podido poner en práctica. Requiere de una infraestructura a la que no se tiene acceso (un servidor donde alojar alguna aplicación del tipo Hudson o CruiseControl y una máquina de compilación/referencia al menos). También, teniendo en cuenta que el proyecto no va a instalarse en un entorno de producción, sino que se trata, simplemente, de un proyecto fin de carrera que debe ejecutarse en un ordenador personal, se ha



determinado que tampoco era imprescindible para este caso. Aun así, se ha considerado importante exponer, al menos, un resumen de esta buena práctica que debiera ser imprescindible en el desarrollo de aplicaciones que van a tener como destino un entorno de producción.

Para finalizar esta sección, se quiere resaltar que se han aplicado todos los recursos ágiles que han sido posibles y hemos descartado aquellos que no se han podido implementar por falta de infraestructura. Aun así, con los medios y tiempo de los que se disponían, podemos decir que Comunic@ es un buen ejemplo de cómo desarrollar una aplicación con una metodología ágil.



img/CircuitoMarquesina-eps-converted-to.pdf

Parte III

Sistema a desarrollar

Capítulo 5. Planificación temporal y costes

5.1. PLANIFICACIÓN TEMPORAL Y COSTES EN SCRUM

En este apartado se va a resumir cómo se realiza una planificación temporal con Scrum. Se tiene en cuenta que todo cliente quiere saber el coste de su aplicación y cuánto tiempo se va a necesitar para llevar a cabo un proyecto conforme a los recursos de los que se dispone.

En otros modelos, como el modelo en cascada¹, se hace una toma de requisitos previa con la cual se realiza la planificación. Con esta planificación se estiman los costes y el número de recursos requeridos para entregar en una fecha fijada. Se firma todo en un contrato cerrado, y el cliente espera a la fecha elegida para ver el producto finalizado.

Scrum no es tan rígido como este modelo, aunque existe una planificación temporal: se describen los costes, puesto que es necesario, pero se abre la posibilidad de negociar con el cliente nuevas fechas y añadir nuevas tareas.

En el modelo en cascada, desde el inicio del producto hasta el comienzo de su codificación, deben darse las siguientes etapas:

- 1.- Análisis de requisitos.
- 2.- Diseño del sistema.

¹ Ingeniería del software, un enfoque práctico[?], capítulo 3.2.

3.- Diseño del programa.

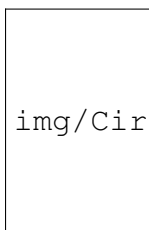
4.- ...

Por tanto, se tienen que finalizar tres etapas de arduo diseño para comenzar la codificación. Por la experiencia que hemos adquirido, sabemos que, hasta que uno no empieza a “ensuciarse”, no sabe realmente hasta qué punto quedará “hundido en la suciedad”. Esta metáfora quiere decir que hasta que no se empieza a construir el proyecto, no se puede saber el alcance que este tendrá, especialmente, si no hay retroalimentación por parte del cliente.

Además, en estos modelos surge la figura del analista. Esta persona indica lo que quiere hacer y cómo pero no se preocupa en desarrollarlo. Son los programadores los que comprueban si se puede realizar o no una tarea mientras están codificando, por tanto, pueden surgir problemas que no se habían previsto o tenido en cuenta. En ese momento es cuando se empiezan a gastar los “colchones” que utilizan los analistas en la estimación para usar en caso de imprevisto. Si no se ha realizado bien la estimación puede acarrear graves consecuencias. Un síntoma de que se ha realizado una mala estimación es no tener la aplicación desarrollada en fecha de entrega, generalmente los programadores en esta situación deben realizar horas extras, lo que suele desembocar en frustración por parte del equipo. En definitiva, se deja sobre el analista la labor de adivino y como se sabe, no es una ciencia exacta y por eso se comenten errores.



Figura 5.1: Tira cómica y realista hecha por Alex Gorbachev



La filosofía de Scrum es totalmente distinta, en primer lugar se requiere implicación del cliente, debe estar presente en la toma de requisitos, lo cual también se realiza en el modelo en cascada, como es lógico. Además, también debe estar en el análisis de requisitos, lo cual ya no es habitual en otros modelos. En este momento la pregunta sería: ¿pero no es el trabajo del analista analizar los requisitos?, ¿qué tiene que ver el cliente?.

Para responder a esta pregunta comenzaremos a recordar algún aspecto mencionado en el capítulo 4. Un equipo en Scrum no está estructurado de forma piramidal, se trata de un equipo de trabajo horizontal. Aunque existe la figura de Scrum máster es solo un guía que encarna la figura de “líder servil”. Se entiende que todos los trabajadores tienen la misma voz y voto. En este caso, todos los componentes participan y por eso se realiza una reunión inicial de sprint en la cual está presente el representante del cliente, para que pueda aclarar las ideas que surjan del producto. En la reunión inicial es donde se recogen las historias de usuario (Reunión backlog), se hace la denominada comúnmente “toma de requisitos”. La reunión inicial de sprint es donde se analizan los requisitos y debe

estar presente el cliente, puesto que es el quién guiará al equipo indicando lo que tiene más prioridad para la consecución de su aplicación.

Las otras dos etapas del modelo en cascada también están presentes en Scrum, en cambio, no están tan diferenciadas. Se van realizando a la vez que se implementa el sistema. Es cierto que el primer sprint, el denominado sprint 0, se encarga de preparar el entorno. En cambio, la meta de este sprint no es la de instalar todas las tecnologías que se van a usar o se supone que se van a usar a lo largo de todo el proyecto, sino que se ocupa de las necesarias para poder realizar el sprint siguiente. Con esta serie de acciones se empieza a trabajar rápidamente, más adelante se seguirá ajustando el sistema cuando el desarrollo del proyecto lo requiera.

Para poner un ejemplo específico de nuestro proyecto, cuando se modificó el alcance del proyecto para realizar la aplicación android, el equipo no sabía realizar aplicaciones android, así que, aún menos se tenía el entorno instalado. En otros modelos, el jefe de proyecto podría estallar en cólera dando lugar a un mal ambiente y problemas con el cliente. Estos pensamientos pueden surgir si se sigue una metodología predictiva:

“Si hubiera sabido que se iba a realizar una aplicación android hubiese contratado a un desarrollador con esa experiencia. Encima ahora tenemos que volver a generar la documentación, volver a realizar una toma de requisitos, y volver a perder tiempo instalando el entorno para realizar aplicaciones android. Por lo menos esto le va a salir caro al cliente.”

Incluso se puede dar el caso de buscar un nuevo recurso que supiese realizar esta tarea ya que, como es el cliente quién ha violado el contrato que tenían previsto ambas partes, no se había contratado al personal adecuado. El resultado es siempre el mismo: modificación del contrato y más gastos para cliente.

Dado que no hay flexibilidad ante cambios en la planificación, la metodologías antiguas caen bajo su propio peso. En cambio, en el caso de comunic@, se vió con buenos ojos esta nueva historia de usuario e, incluso, uno de los desarrolladores se motivó, puesto que llevaba tiempo queriendo hacer una aplicación android y esta era una oportunidad excelente. Es cierto que el coste aumentó, puesto que se cambio la fecha y aumentaron las horas de trabajo, pero el cliente pudo entenderlo, porque no vió que se gastase tiempo en una documentación excesiva o que aumentasen el número de horas de forma exagerada, simplemente, anuló otras historias de usuario.

Una vez resaltadas las principales características de Scrum y sus diferencias con otros modelos procedemos a explicar como se planifica en Scrum.

La planificación se realiza en la reunión inicial de sprint, en capítulos posteriores explicaremos el transcurso de esa reunión. Tras la toma de requisitos en la reunión de backlog, se pone fecha para la reunión inicial de sprint. En ella estarán presentes tanto el cliente como el equipo de desarrollo.

Para realizar una planificación en primer lugar se tienen que definir la duración en la realización de las tareas. Para ello contamos con todos los integrantes del desarrollo. En nuestro caso se utilizará la siguiente técnica: planning poker[?].

Para realizar esta técnica cada miembro del equipo cuenta con una baraja de trece cartas, como las que se muestran en la imagen 5.2[?].



img/Cir

Figura 5.2: Cartas para realizar planning poker.

A continuación se definen los pasos que se siguen cada vez que se quiere estimar una historia:

- 1.- Se lee en voz alta la historia que se quiere estimar.
- 2.- Cada miembro del equipo selecciona una carta que representa su estimación de tiempo (en puntos de historia) y la coloca bocabajo en la mesa.
- 3.- Cuando todos los miembros del equipo han preparado sus cartas, se les da la vuelta al mismo tiempo. Con esto se obliga a que cada miembro no se vea influenciado por la estimación de otra persona.
- 4.- Si hay mucha discrepancia en las opiniones, se escuchan las justificaciones del equipo y se llega a un consenso. Se pueden volver a usar las cartas hasta alcanzarlo.
- 5.- En este momento se puede hacer una división en historias más pequeñas y volver a estimarla si se necesita.

img/CircuitoMarquesina-eps-converted-to.pdf

Además del conjunto de cartas numeradas, existe un conjunto de cartas especiales:

- ★ 0 = Con esta carta se quiere indicar que esta historia ya está hecha o que no implica esfuerzo.
- ★ ¿ = No se tiene idea de cómo abordar esta historia de usuario.
- ★ Taza de café = El miembro del equipo que saque esta carta quiere poner en evidencia que necesita un descanso para poder continuar con las estimaciones.

Una vez estimadas las tareas, se procede a darles según su puntuación una relación en tiempo. En este caso se hace una relación horas-hombre, porque no se van a poder dedicar días enteros en el desarrollo, por tanto no tiene mucho sentido el uso de días-hombre.

Con esto se da por concluida la estimación, a continuación indicamos los valores específicos de esta estimación para Comunic@.

5.2. PLANIFICACIÓN TEMPORAL Y COSTES EN COMUNIC@

img/Cir

5.2.1. PLANIFICACIÓN INICIAL

Como se trata de un proyecto con plazo fijo, se hizo un breve análisis que se documentó con un diagrama de Gantt en donde las tareas tenían un alcance muy amplio. Esto sirvió para contabilizar el tiempo del que se disponía. Las historias de usuario que se iban a acometer aún no estaban definidas de manera concreta, aunque se tenía una propuesta de lo que el proyecto iba a abarcar. Por este motivo, era imposible realizar un análisis de grado fino, aunque tampoco era necesario en ese momento del desarrollo del proyecto.

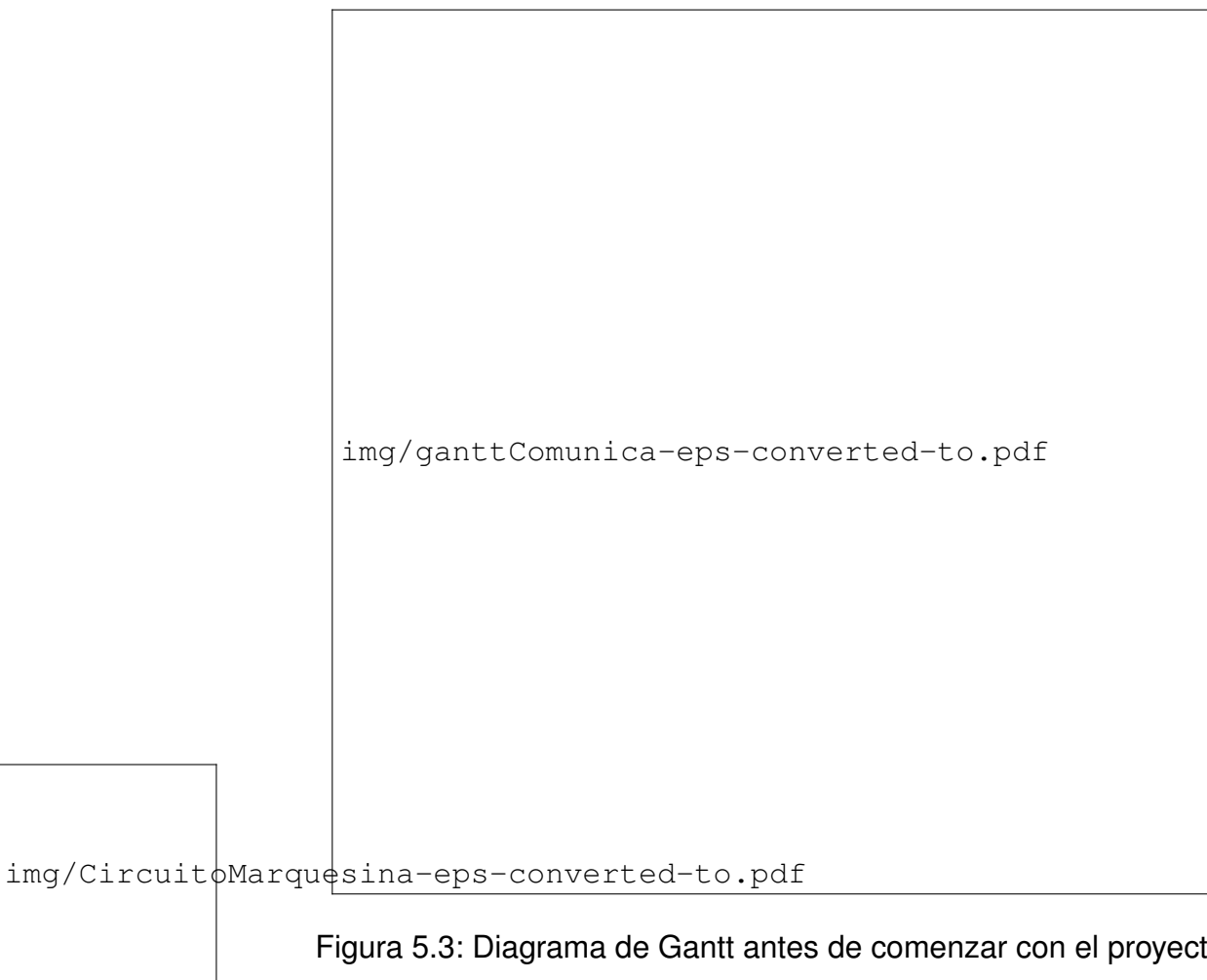


Figura 5.3: Diagrama de Gantt antes de comenzar con el proyecto.

Contabilizando el tiempo y la disponibilidad de los desarrolladores, se planificaron las semanas de manera que se pudiera trabajar unas dieciocho horas: seis horas entre semana a dividir en dos días, y doce durante el fin de semana. El cálculo de costes de personas se hizo de la siguiente manera:

18 horas X semana X 2 desarrolladores = 36 horas/semana de trabajo efectivo.

Esto da un total de 656 horas dedicadas al proyecto.

Si se tratara de un proyecto real, habría que calcular los costes. A los costes del salario que se paga al desarrollador, hay que sumarle los costes indirectos basados en la utilización de la infraestructura y en el pago de los puestos no productivos de la empresa, que suelen subir un 50 %-60 % con respecto a los costes de la persona.



img/Planificacion-eps-converted-to.pdf



img/Cir

Puede estimarse que el sueldo medio de un desarrollador está en torno a 15€la hora, suponiendo costes indirectos en torno al 50 %. El resultado quedaría como sigue:

(costes personales + costes indirectos) X horas de trabajo
(15+7) x 656 = 14432 euros en total.

Como se observará en la siguiente sección, esta planificación fue bastante optimista y tuvimos que retrasar la entrega del proyecto, tanto por motivos profesionales y personales, como por la ampliación de alcance.

5.2.2.DESVIACIONES CON RESPECTO A LA PLANIFICACIÓN

Se puede concluir en este capítulo que no ha habido importantes desviaciones en este proyecto. El incremento en horas de desarrollo ha sido debido al

aumento de la funcionalidad de la aplicación.

En un principio, se quería implementar la aplicación para septiembre. Como se ha explicado anteriormente, los programadores no disponían de total disponibilidad para este proyecto, por tanto se eliminaron las fechas de final de sprint y se decidió realizar entregas a medida que finalizaran las historias comprometidas para dicho sprint.

Se tenía una previsión aproximada de la disponibilidad de los desarrolladores, pero debido a una mala planificación en otros proyectos relacionados con el trabajo personal que desembocaron en horas extra² y enfermedades, esta disponibilidad se vio mermada. Además, en una demo de sprint, se decidió añadir servicios web y una pequeña aplicación para Android. Aunque, se reestimaron o desecharon otras tareas, el número de horas necesarias para la realización del proyecto aumentó.

En conclusión, para el cliente eran muy importantes las nuevas tareas, por lo que se decidió modificar la fecha de entrega final a la convocatoria de diciembre.

img/CircuitoMarquesina-eps-converted-to.pdf

Además de este cambio en fechas y tareas de usuario, se pudo entregar correctamente el presente proyecto para la convocatoria que estaba estimada. Con ello concluimos que la planificación fue un éxito. Al final, solo se descartaron dos historias de usuario que no eran importantes para el cliente.

²como es de interés estadístico para este proyecto fin de carrera, se especifica que la metodología usada en el desarrollo del otro proyecto fue CMMI.

Capítulo 6. Arquitectura básica

“Hay solo dos clases de lenguajes de programación: aquellos de los que la gente está siempre quejándose y aquellos que nadie usa.” – Bjarne Stroustrup.

En el proyecto Comunic@ se han utilizado diversas tecnologías que conforman la arquitectura básica del aplicativo. Como se van a utilizar metodologías ágiles como BDD, no tiene mucho sentido un análisis demasiado detallado de esta parte, lo que no quiere decir que se desestime un análisis previo de las tecnologías que requieren nuestro proyecto.

Al ser una aplicación web, se ha elegido usar el patrón de diseño MVC (Modelo-Vista-Controlador), que hace una división de las capas muy utilizada y útil para este tipo de aplicativos. Se explicará más adelante en este mismo capítulo.

Como lenguaje de programación se ha decidido usar Java en su versión 1.5. Java es uno de los lenguajes de programación más usados para aplicaciones web, sobre todo cuando tienen una complejidad algo elevada, y posee muchas herramientas que nos facilita la tarea de programar en este entorno.

Como framework para la aplicación del patrón MVC se va a usar Strut 2 junto con Spring, ya que soluciona de manera eficaz dicho patrón, configurando las interacciones entre el modelo y la vista con unos simples ficheros de configuración en xml. Hay más frameworks de este tipo, como JSF, pero por conocimiento previo y buena experiencia, se decidió usar Strut 2.

Para el acceso a datos se va a usar un framework llamado Hibernate, en su versión 3. En la actualidad no es común que alguien implemente la capa de acceso a datos a mano, teniendo herramientas como Hibernate o Ibatis que solucionan parte de esta complejidad. Se elige Hibernate ya que para el uso de



img/Cir

metodologías como BDD, donde el diseño de la base de datos se basa en las clases y no al contrario, facilita mucho el uso de anotaciones dentro de las clases que luego se parsearán de manera automática a tablas.

La base de datos elegida ha sido una Oracle 10g XE, aunque se podría haber usado otra como MySQL sin problemas, ya que, aunque en un principio se eligió por la potencia que tiene la escritura de PL/SQL, se decidió que al usar BDD no era buena idea hacer que parte de la lógica de negocio la desarrollara la base de datos, mejor que fuera totalmente independiente de esta.

Debido al avance en el desarrollo para dispositivos móviles como los smartphones, se decidió añadir un pequeño aplicativo para Android 2.2, que mostrara los últimos comunicados publicados. Para ello se decidió hacer uso de REST como protocolo para comunicar ambos dispositivos. Así se implementó tanto en el cliente (aplicación Android) como el servidor (aplicación web) esta capa de comunicación entre ellos. Se ha añadido también la posibilidad de consumir por servicios web por si en un futuro se necesitara, ya que, una vez implementado REST, publicar por RPC resultó trivial.

Para el despliegue y construcción de la aplicación web se ha usando Maven, también hemos usado un arquetipo como esqueleto del proyecto llamado AppFuse. Como entorno de desarrollo tanto de la aplicación web como de la de Android, Eclipse Indigo. Como contenedor de Servlet, la elección ha sido Tomcat 1.5.

Como repositorio, estamos usando Subversión, aunque al descubrir Mercurial se estuvo sopesando migrar a este sistema puesto que es bastante más potente que SVN.

6.1. PATRÓN MODELO-VISTA-CONTROLADOR

El patrón MVC es un patrón de arquitectura de software que separa el acceso a los datos de una aplicación de la lógica de negocios y de la interfaz de usuario [?].

En aplicaciones web, que modelo sería el acceso a los datos y la lógica de negocios, la vista sería las jsp que dan lugar al código html visto por el cliente



img/Cir

(interfaz gráfica) y el controlador es el responsable de coordinar los eventos de entrada de la interfaz gráfica hacia la lógica de negocios y viceversa.

Este patrón fue escrito por primera vez por Trygve Reenskaug ¹ cuando estaba trabajando en Smalltalk en 1979.

Se detallan a continuación los tres componentes que conforman este patrón:

Modelo: Esta es la representación específica de la información con la cual el sistema opera. Se compone por el Sistema de Gestión de Base de Datos y la lógica de negocio (aunque se podrían usar otros artefactos para almacenar datos como xml o ficheros csv). La lógica de negocio asegura la integridad de estos y permite derivar nuevos datos. El Sistema de Gestión de Base de Datos (SGBD) será el encargado de almacenar los cambios en los datos (agregarlos, editarlos o borrarlos) producidos por la lógica de negocio. Ejemplos de SGBD son MySQL, Oracle...

¹ <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

Es recomendable una capa de abstracción extra denominada Data Access Object (DAO), que es un componente de software que suministra una interfaz común entre la lógica de negocio y el SGBD.

Vista: esta capa presenta el modelo en un formato adecuado para interactuar, en este caso la interfaz de usuario. Por lo tanto, la vista es la encargada de presentar los datos al usuario y la interfaz necesaria para modificarlos. Un ejemplo de tecnología podría ser las JSP que, mediante el servidor, genera HTML que interpreta el navegador del usuario mostrándole los datos y los formularios que constituyen la vista para que pueda interactuar con la aplicación.

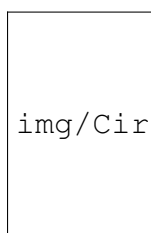
Controlador: esta capa responde a eventos, es decir, acciones del usuario, que invoca cambios en el modelo y probablemente en la vista. Por lo general, el controlador sería la unidad central que comunica la vista con el modelo y viceversa, asociando los eventos del usuario con los cambios que se producirán en el modelo y devolviendo los datos resultantes que genere el modelo a la vista que corresponda.

El flujo básico podría dividirse en los siguientes pasos:

- ★ El usuario realiza una acción en la interfaz.
- ★ El controlador trata el evento de entrada.
- ★ El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo o ser simplemente una consulta de algunos datos.
- ★ Se genera una nueva vista. La vista toma los datos del modelo.
- ★ La interfaz de usuario espera otra interacción del usuario que implicará volver al primer punto.



Figura 6.1: Ejemplo de patrón MVC. [?]



6.2. PATRÓN DAO

El patrón “Data Access Object” surge de la necesidad de diferenciar el manejo de los datos con respecto a cómo se realiza el acceso a los datos y cómo se almacenan. Es bastante normal hacer aplicaciones que almacenan y recogen datos de una base de datos. También puede ser interesante hacer que la aplicación sea lo más independiente posible de una base de datos concreta, de cómo se accede a los datos o incluso de si hay o no base de datos detrás. La aplicación debe conseguir los datos o ser capaz de guardarlos en algún sitio, pero no tiene por qué saber de dónde los está sacando o dónde se guardan.

La ventaja de usar este patrón es que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula.

El uso de ambos patrones es imprescindible para la buena marcha del proyecto, ya que encajan perfectamente dentro de las metodologías ágiles, facilitando la modularización de sus partes.

6.3. SERVICIOS WEB. ARQUITECTURA ORIENTADA A SERVICIOS.

Un servicio web es un método de comunicación entre dos dispositivos usando la red como canal de comunicación. Define un conjunto de protocolos y estándares que sirven para orquestar el intercambio de datos entre aplicaciones. Hace posible que distintas aplicaciones de software puedan utilizar los servicios web para intercambiar datos a través de la red sin que importe el lenguaje de programación o la plataforma en la que están funcionando. La interoperabilidad se consigue mediante la adopción de estándares abiertos que se han de respetar a la hora de implementarse. Las organizaciones OASIS² y W3C³ son los responsables de la arquitectura y regulación de los estándares de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web nació el organismo WS-I⁴ que se encarga de definir de manera más exhaustiva estos estándares.

Durante el desarrollo del proyecto se tienen que estudiar dos protocolos de servicios web, estos son RPC (SOAP) y REST. En las siguientes secciones se aprovecha para explicar lo investigado con respecto a ambos ya que terminarán siendo una de las elecciones importantes dentro de la arquitectura.

6.3.1.RPC

RPC son las siglas de “Remote Procedure Call”, se generó el primer protocolo XML-RPC en 1998 por David Winer, que evolucionó al protocolo SOAP (Simple Object Access Protocol) creado por Microsoft y que está amparado por la W3C.

²<http://www.oasis-open.org/home/index.php>

³<http://www.w3c.es/Consortio/>

⁴<http://www.ws-i.org/>

SOAP se basa en XML (Extensible Markup Language) para su formato de mensaje, y por lo general se basa en otros protocolos a nivel de capa de aplicación, en concreto el de transferencia de hipertexto (HTTP) y el Simple Mail Transfer Protocol (SMTP), que definen la negociación y transmisión de mensajes. SOAP forma la capa base de un conjunto de protocolos de servicios web, proporciona un marco básico de mensajería a partir del cual se construyen los servicios web.

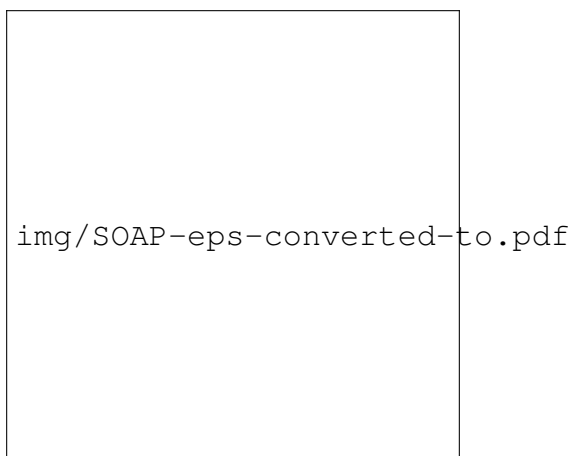


Figura 6.2: Estructura de mensaje SOAP de Silver Spoon Sokpop.

Como se observa en la figura 6.2, un mensaje SOAP es un documento XML que contiene ordinaria los siguientes elementos:

- ★ Un elemento “sobre” que identifica el documento XML como un mensaje SOAP.
- ★ Un elemento “encabezado” que contiene la información de cabecera y que puede venir vacío.
- ★ Un elemento “cuerpo” que contiene la información de llamadas y respuesta con sus datos.
- ★ Un elemento “error” que contiene errores y la información de estado

Todos los elementos anteriores se declaran en el espacio de nombres por defecto para el elemento “sobre” de SOAP ⁵.

⁵<http://www.w3.org/2001/12/soap-envelope>

El espacio de nombres predeterminado para la codificación de SOAP y tipos de datos está definido en <http://www.w3.org/2001/12/soap-encoding>

SOAP tiene tres características principales: extensibilidad (seguridad y enrutamiento), neutralidad (SOAP se puede utilizar en cualquier protocolo de transporte tales como HTTP, SMTP o TCP), y la independencia (SOAP permite que cualquier lenguaje de programación y plataforma).

Para finalizar con SOAP, se muestra un ejemplo de mensaje. Para más información, en la web de w3schools hay un completísimo manual al respecto⁶.

```
SOAP request:

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

```
SOAP response:

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Los servicios web cuenta entre sus herramientas con una forma de publicar un servicio de páginas amarillas, o catálogo de procedimientos remotos llamado WSDL. Especifica la interfaz abstracta a través de la cual un cliente puede

⁶<http://www.w3schools.com/soap>

acceder al servicio y los detalles de cómo se debe utilizar. WSDL combina perfectamente con SOAP y suelen ir de la mano.

WSDL describe la interfaz pública de los servicios Web. Está basado en XML y explica cómo interactuar con los servicios listados en su catálogo. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar qué métodos están disponibles en el servidor y cómo puede acceder a ellos. Los tipos de datos especiales se incluyen en el archivo WSDL definidos con un formato especial. El cliente usaría SOAP para hacer la llamada a uno de los métodos listados en el WSDL y así realizar las operaciones en el servidor.

Se observa como todo lo referente a servicios web se basa en una arquitectura orientada a servicios.

6.3.2.REST

REST (Representational state transfer) es un estilo de arquitectura de software para sistemas hipermedia distribuidos como la World Wide Web. El término REST fue introducido y definido en 2000 por Roy Fielding en su disertación doctoral⁷. Fielding es uno de los autores principales del Protocolo de transferencia de hipertexto (HTTP) versiones 1.0 y 1.1 de su especificación⁸.

En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y disecionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional, como hace SOAP. Estos dos significados pueden chocar o incluso solaparse. Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en cambio seguir un modelo RPC. Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST no es un estándar, está basado en estándares: HTTP, URL, Representación de los recursos (XML/HTML/GIF/JPEG/...), Tipos MIME (text/xml, text/html).

⁷<http://www.ics.uci.edu/~fielding/>

⁸<http://tools.ietf.org/html/rfc1945> y <http://tools.ietf.org/html/rfc2616>

Un servicio web REST (también llamado REST Web API) es un servicio web simple implementado utilizando HTTP y los principios de REST. Es una colección de recursos, con cuatro aspectos definidos:

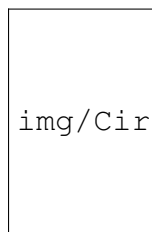
- ★ Una base de URI para el servicio web, como `http://example.com/resources/`.
- ★ Un tipo de representar los datos compatibles con el servicio web. Esto a menudo son del tipo JSON, XML o YAML.
- ★ Un conjunto de las operaciones soportadas por el servicio web usando para ello los métodos HTTP (por ejemplo, POST, GET, PUT o DELETE).
- ★ El API debe ser transmitido mediante hipertexto.

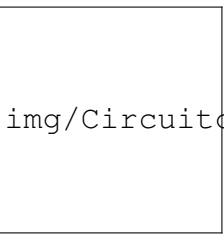
El acceso a los datos por parte del cliente se realiza mediante URIS específicas, de tal manera que, si necesita un listado de objetos de un determinado tipo, esta tendrá su propia URI, que devolverá al llamarla, un objeto response con los datos, ya sea un XML o una estructura JSON. Esta estructura será un simple XML con los objetos.

Un ejemplo de cómo son los mensajes en REST se puede observar al suscribirse a un canal de RSS o Atom, ya que trabajan con servicios web REST. La figura 6.3 es una muestra de la respuesta de una petición RSS a Google News.



Figura 6.3: Captura del response de una petición rss en Google News.





img/CircuitoMarquesina-eps-converted-to.pdf

Capítulo 7. Historias de usuario

“Knuckle deep inside the borderline. This may hurt a little but it’s something you’ll get used to. Relax. Slip away.” – Tool. Aenima.

Las historias de usuario se definen en la primera reunión que se realiza con el cliente, y se genera el backlog. Esta reunión es esencial y una de las más importante puesto que el objetivo es detallar las historias de usuario tal y como se explica en el capítulo 6.2.

En esta reunión se tiene que obtener del cliente toda la información necesaria para comenzar a trabajar. Se escuchan sus peticiones y se recogen usando el lenguaje más sencillo posible, escribiendo frases simples y evitando ambigüedades.

La idea del cliente para el proyecto Comunic@ es la siguiente: se pretende hacer un tablón de anuncios y sugerencias para la intranet de una gran empresa. Cada usuario del sistema (que son los trabajadores de la empresa) puede poner sus anuncios o sugerencias para que puedan verse a través de una página web que forma parte de la intranet de la empresa.

Se aprovecha para definir los roles que puede tener el proyecto, así surge el rol de administrador, de usuario y de dueño del producto. También se designará una persona que adquirirá el rol de dueño del producto durante el desarrollo del proyecto. Normalmente, será algún contacto dentro de la parte interesada. En el caso de Comunic@, este papel lo interpretará el tutor del proyecto.

La visión global del proyecto tras la primera reunión se puede resumir en lo siguiente:

img/Cir

El cliente indica que necesita un sistema de autenticación básico para acceder a la página web. Nuestro cliente quiere que se puedan insertar anuncios. Asimismo, quiere que los usuarios también tengan la posibilidad de hacer sugerencias para la empresa, pero, tanto los anuncios como las sugerencias, deben ser previamente aprobados o rechazados por un usuario administrador.

Una vez que se tiene una idea generalizada de la propuesta del cliente y terminada la reunión, se escriben las historias de usuario con sus test de aceptación de la información recogida en esta reunión y se contrastan con las necesidades del dueño del producto (representante del cliente) para garantizar que definan la funcionalidad de la aplicación. Como nuestro modelo tiene como máxima ser flexible, el cliente podrá añadir más historias de usuario o eliminar alguna según se vaya desarrollando el producto, y priorizarlas como crea conveniente durante el desarrollo del proyecto, eso sí, sin interrumpir al equipo durante el transcurso del sprint. Esta reunión sería la primera toma de contacto y nos debería de dar la suficiente información como para poder trabajar en varios sprints.

En el proyecto Comunic@, se generan y contrastan 26 historias de usuario en total, es decir, las que se generaron al iniciar el proyecto y las que se han ido añadiendo después. Algunas no se han realizado, ya que se eliminaron durante la ejecución del proyecto, mientras que otras entraron más tarde. Todo esto se explica en capítulos posteriores, pero de momento se pueden resumir en las siguientes:

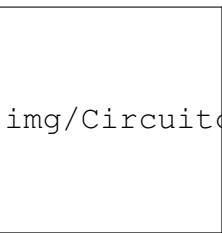
- 1.- **Despliegue de aplicativo:** el dueño de la aplicación quiere que sea una aplicación web fácilmente portable e independiente.
- 2.- **Autenticación en el sistema:** el dueño de la aplicación quiere que todo usuario tenga que autenticarse en el sistema.
- 3.- **Insertar Comunicado:** el usuario quiere tener la opción de insertar un comunicado.
- 4.- **Leer los comunicados:** el usuario quiere tener la opción de leer los comunicados.
- 5.- **Filtro de comunicados publicados para usuarios:** El administrador quiere que solo puedan ser vistos por el usuario los anuncios y sugerencias publicados.

- 6.- **Ordenar por fecha descendente los comunicados en el listado de administración:** el administrador quiere ver su listado de comunicados por orden descendente.
- 7.- **Responder comunicados:** el administrador puede responder un comunicado.
- 8.- **Estados de los comunicados:** el administrador quiere tener la opción de publicar o rechazar un comunicado.
- 9.- **Borrado de las respuestas:** el administrador puede borrar las respuestas.
- 10.- **Borrado de los comunicados:** el administrador puede borrar los comunicados.
- 11.- **Publicación de Web Services:** el dueño de la aplicación quiere que se pueda operar fácilmente desde otras aplicaciones con Comunic@.
- 12.- **Aplicación lectora de comunicados Android:** se puede ver el listado de anuncios y sugerencias en un móvil Android.
- 13.- **Borrar comunicado por email:** el usuario quiere contar con la opción de pedir al administrador que borre un comunicado por email.
- 14.- **Notificación de nuevos comunicados:** el administrador quiere que le lleguen por email los nuevos comunicados.
- 15.- **Notificación de cambio de estado:** el usuario quiere que se le avise por email del cambio de estado de su comunicado.
- 16.- **Búsqueda de comunicados:** el usuario quiere la opción de buscar comunicados.
- 17.- **Paginar comunicados:** los usuarios quieren ver los comunicados paginados.

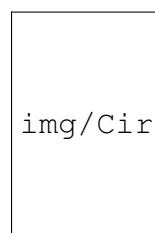
img/Cir

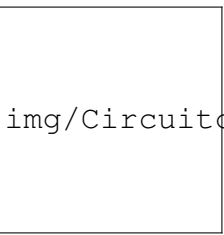
- 18.- **Ordenar comunicados:** el usuario quiere ordenar por fecha de publicación.
- 19.- **Firma de comunicados:** el administrador quiere que los comunicados vayan firmados.
- 20.- **Notificación del motivo de rechazo de un comunicado:** el usuario quiere que le notifiquen porque se ha rechazado su comunicado.
- 21.- **Notificación de la respuesta a un comunicado:** el usuario quiere que se le notifique la respuesta a su sugerencia.
- 22.- **Borrado de anuncios por parte del usuario que los creó:** el usuario quiere borrar sus anuncios.

Con estos datos se generan las historias de usuario, recogidas por el Scrum máster. Ya se dispone de material suficiente para poder realizar nuestra primera reunión de sprint, tal y como se especifica en el apartado siguiente.



img/CircuitoMarquesina-eps-converted-to.pdf





img/CircuitoMarquesina-eps-converted-to.pdf

Capítulo 8. Planificación y generación de la pila de productos

8.1. SPRINT INICIAL

img/Cir

“El optimismo es un riesgo laboral de la programación: el feedback es el tratamiento” - Kent Beck.

En el capítulo anterior se define la recogida de datos y se generan a partir de estos las historias de usuario. En este capítulo se procede a elegir las historias que se van a realizar en los siguientes sprints, asignándoles un peso para indicar qué tarea tiene mayor prioridad. Este trabajo se realiza en una reunión llamada planificación de sprint, en la cual debería estar presente un representante del cliente en calidad de dueño del producto. Sabemos que es complicado que asista la figura de cliente, puesto que el dueño una vez hecha la pila de backlog, no entiende porque tiene que “perder el tiempo” en una reunión de planificación. La importancia de la presencia del dueño de la aplicación, se debe a estos tres factores:

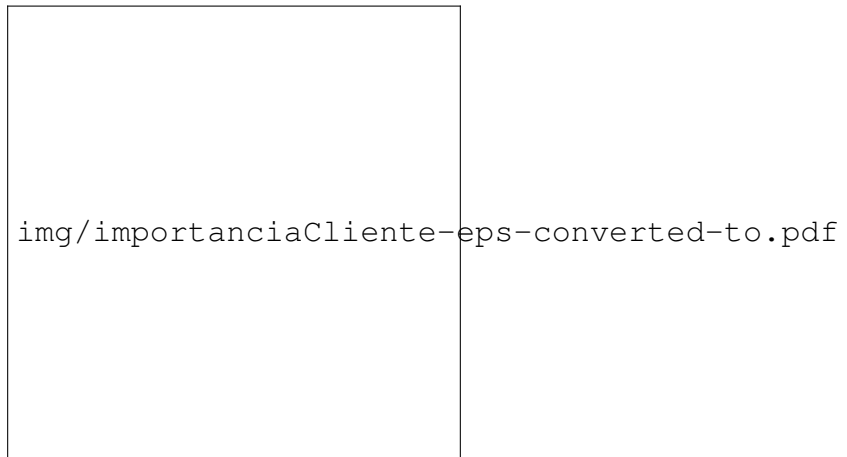


Figura 8.1: Factores que miden la prioridad de una historia.

El dueño de la aplicación o, para ser más exactos, el representante de este, decide la importancia y la prioridad, siempre aconsejado por el equipo; la estimación la proporciona el equipo. Generalmente el dueño indica cuál es la meta del próximo sprint y las historias más importantes. A continuación, el equipo le asigna una estimación. Muchas veces, esa estimación no suele ser la esperada por el cliente, lo que puede provocar una re-estimación y que cambie la prioridad de alguna historia de usuario. Por lo tanto, es importante que esté presente el dueño del producto ya que, en este caso, podría proponer un cambio de meta del sprint al conocer el verdadero alcance de las historias.

En Comunic@, hemos decidido clasificar nuestras tareas utilizando los siguientes conceptos para ayudarnos a clasificar la dificultad y prioridad de las tareas:

Desconocida importante, DI : son tareas en las que, a priori, se desconoce cómo se van a implementar, pero son importantes para el cliente. En este caso, se les da la prioridad más alta puesto que, si se desconoce el cómo se van a realizar, es posible cometer errores o prolongar la investigación de la resolución de la historia. En el proceso de implementación de un programa en la etapa inicial siempre es más fácil solventar problemas que en etapas más avanzadas; por tanto, se deciden que estas historias son las que mayor prioridad tienen.

Conocida importante, CI : son historias de cliente cuya implementación es conocida y, por tanto, fáciles de programar. Como son importantes para nuestro cliente, su prioridad también es alta.

Conocida poco importante, CP : son historias que no repercuten en la funcionalidad de la aplicación, pero son extras que mejoran la aplicación. Por tanto, son poco importantes para el cliente pero, dado que son conocidas, su implementación no debería ser muy costosa y, por tanto, su prioridad es baja. En todo caso se deberían de poder realizar dado que su coste es bajo.

Desconocidas poco importantes, DP : como se ha descrito en el caso anterior, son funcionalidades poco importantes para el cliente, pero que el equipo desconoce. Por consiguiente, pueden llegar a ser muy costosas. Generalmente, si se cumplen las historias de usuario anteriores, el cliente debería quedar lo suficientemente satisfecho. Por tanto, en caso de ir justos en los plazos, estas historias se podrían no realizar y, aun así, la aplicación cubriría las expectativas del cliente.

Una vez descritas las categorías en la reunión, procedemos a catalogar las historias de usuarios, teniendo en cuenta los aspectos descritos anteriormente.

Para ello se utilizan tarjetas. El motivo de utilizar tarjetas en vez de una hoja de cálculo es que, gracias a las tarjetas, se consigue que el grupo se involucre más; no hay una sola persona modificando la hoja de cálculo mientras todos miran. Las tarjetas se pueden poner encima de la mesa y el grupo puede modificar la prioridad de la tarea con solo moverla de lugar. Además, se pueden pegar en un papel y tener ya preparada la pizarra de Scrum para empezar a trabajar.

A rectangular box with a thin black border. Inside the box, the text 'img/Cir' is written in a standard font.

Después, para tenerlo en formato digital, el Scrum máster puede pasar las historias de usuario a una hoja de cálculo indicando qué tareas van en el primer sprint.

En el proyecto se escribieron las tarjetas en papel, pero no se usó la pizarra porque la mayor parte del tiempo se trabajaba a distancia. Por ello, se decidió pasar las tarjetas a un archivo de Excel y compartirlo a modo de pizarra para su modificación via internet.

Una vez que se tienen las historias de usuario ordenadas, se procede a darles un número de prioridad. Este número nos indica qué historia se realizará en primer lugar a juicio de los integrantes del grupo, ayudados por el dueño del producto, que también estimará la prioridad de las historias para su aplicación.

Cuando el dueño del producto haya indicado la prioridad de las tareas, el



img/CircuitoMarquesina-eps-converted-to.pdf

equipo decide cuáles entran dentro del sprint inicial. Si el dueño no está de acuerdo, puede volver a estimar la prioridad asignada a una tarea.

En nuestro caso, ordenamos las historias de usuario de la siguiente forma (se han incluido todas, incluso las que entraron durante el desarrollo y que se explican en el siguiente capítulo):

Desconocida importante, DI :

9000 - **Borrar comunicado por email:** el usuario quiere contar con la opción de pedir al administrador que borre un comunicado por email.

8750 - **Notificación de nuevos comunicados:** el administrador quiere que le lleguen por email los nuevos comunicados.

8500 - **Notificación de cambio de estado:** el usuario quiere que se le avise por email del cambio de estado de su comunicado.

8400 - **Publicación de Web Services:** el dueño de la aplicación quiere que se pueda operar fácilmente desde otras aplicaciones con Comunic@.

8350 - **Aplicación lectora de comunicados Android:** el usuario quiere ver el listado de anuncios y sugerencias en su móvil Android.

Conocida importante, CI :

8000 - **Despliegue de aplicativo:** el dueño de la aplicación quiere que sea una aplicación web fácilmente portable e independiente.

7900 - **Autenticación en el sistema:** el dueño de la aplicación quiere que todo usuario tenga que autenticarse en el sistema.

7800 - **Insertar comunicado:** el usuario quiere poder insertar un comunicado.

7700 - **Leer comunicados:** el usuario quiere leer los comunicados que se han publicado.

7500 - **Filtro de comunicados publicados para usuarios:** El administrador quiere que solo el usuario pueda ver los anuncios y sugerencias publicados.

7400 - **Responder comunicados:** el administrador puede responder a un comunicado.

7300 - **Estados de comunicados:** el administrador quiere tener la opción de publicar o rechazar un comunicado.

7000 - **Búsqueda de comunicados:** el usuario quiere buscar comunicados.

7100 - **Ordenar por fecha descendente los comunicados en el listado de administración:** el administrador quiere ver el listado de comunicados ordenado por fechas.

6900 - **Borrado de respuestas:** el administrador quiere borrar sus respuestas.

img/Cir

6800 - **Borrado de comunicados:** el administrador quiere borrar comunicados.

6500 - **Paginar comunicados:** los usuarios quieren ver los comunicados paginados.

Conocida poco importante, CP :

5000 - **Ordenar comunicados:** el usuario quiere tener a opción de ordenar por fecha de publicación.

Desconocidas poco importantes, DP :

3000 - **Firma de comunicados:** el administrador quiere que los comunicados vayan firmados.

2900 - **Notificación del motivo de rechazo de un comunicado:** el usuario quiere que le notifiquen por qué se ha rechazado su comunicado.

2800 - **Notificación de la respuesta a un comunicado:** el usuario quiere que se le notifique la respuesta a su sugerencia.

2700 - **Borrado de anuncios por parte del usuario que los creó:** el usuario quiere tener la opción de borrar sus anuncios.

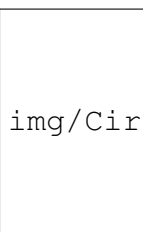
Una vez ordenadas las historias de usuario se plantea una duración del sprint. En nuestro caso, el sprint 0 de preparación, será de dos semanas, puesto que disponemos de la semana santa para realizar el proyecto, el resto serán de dos o tres semanas según el tiempo del que se disponga.

Cuando se tienen las historias de usuario con su prioridad, se eligen las que se van a realizar en el segundo sprint usando la prioridad y la lógica de la aplicación.

Una vez que se tengan las historias que se van a realizar, se dividen en tareas para comenzar a programar.

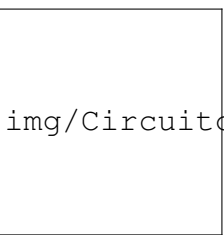


Figura 8.2: División de una historia de usuario en varias tareas.

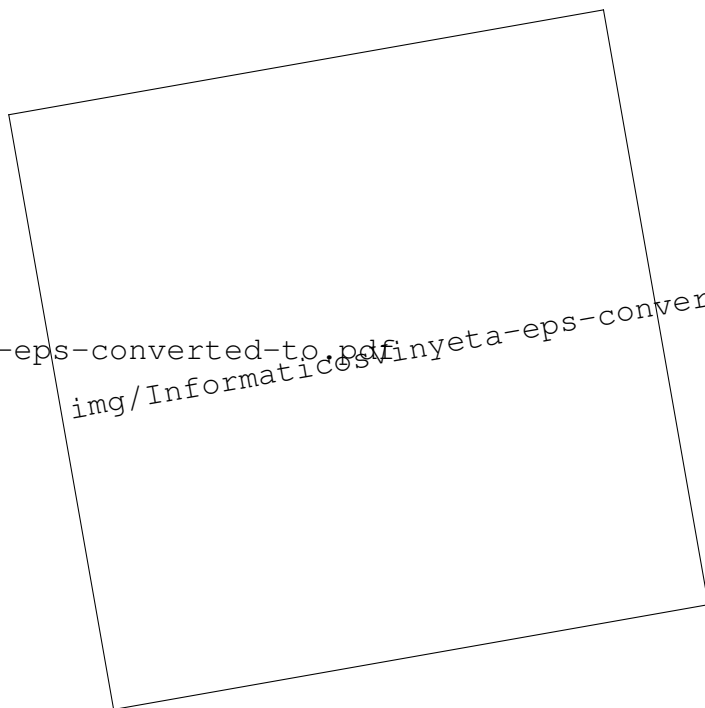


Con esta práctica, se revela el trabajo adicional que tiene una historia, que en un principio no se había tenido en cuenta. También se consiguen simplificar los sprint diarios.

Una vez que se tienen las historias de usuario divididas en tareas, la meta de sprint definida y el lugar y hora del sprint diario damos por concluida la reunión.



img/CircuitoMarquesina-eps-converted-to.pdf
img/InformaticesVineta-eps-converted-to.pdf



Capítulo 9. Diseño e implementación del sistema

9.1. DESARROLLO ITERATIVO

“La programación es una carrera entre los desarrolladores, intentando construir mayores y mejores programas a prueba de idiotas, y el universo, intentando producir mayores y mejores idiotas. Por ahora va ganando el Universo” – Rich Cook

img/Cir

Como se ha estado comentando desde el principio del documento, se va a seguir un desarrollo iterativo, mediante el uso de Scrum.

La finalidad es la de realizar entregables que funcionen al cliente en cada iteración, por lo tanto se van a describir los paquetes que han compuesto estas entregas.

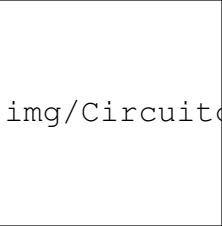
Para entender el concepto de “entregables que funcionen” se podría comparar el aplicativo entero como una pieza de sushi. Si un cliente va a un restaurante de sushi podemos suponer que, si en vez de darle la pieza entera la dividimos de la siguiente manera: el arroz por un lado, las algas por otro y el pescado al final, comérselo por separado no tiene mucho sentido. Es mejor darle pequeñas porciones de las tres capas antes, poco a poco, ya que así podrá ir saboreando el conjunto desde el principio.

Así pues, al iniciar el desarrollo se ejecutará un pre-sprint o sprint 0 en donde se prepara todo el entorno de trabajo y, después, podemos empezar a entregar funcionalidad en los siguientes sprints que podríamos dividir en las siguientes entregas:



img/sushi-eps-converted-to.pdf

- ★ Primer módulo: Preparación y gestión de usuarios.
- ★ Segundo módulo: Operaciones con comunicados básicos (ver e insertar comunicados).
- ★ Tercer módulo: Operaciones de administración sobre comunicados (borrar, cambiar estado, notificar, responder comunicados).
- ★ Cuarto módulo: Publicación mediante Web Services.
- ★ Quinto módulo: Aplicación de Android de lectura de comunicados.
- ★ Sexto módulo: Paginación y búsqueda de comunicados.



img/CircuitoMarquesina-eps-converted-to.pdf

Al realizar un desarrollo iterativo durante el desarrollo del proyecto surgieron cambios en los requisitos, lo que se tradujo en nuevas historias de usuario.

Las tarea de “el usuario quiere poder buscar comunicados” y la aplicación de Android entraron durante el desarrollo del proyecto a formar parte del “backlog” del producto. Así mismo se cambió la de “el usuario quiere poder borrar sus anuncios” por otra más sencilla que era “el usuario quiere poder pedir al administrador que borre una sugerencia por email” y que entró a formar parte del backlog al finalizar el tercer sprint.

Durante la demo del tercer sprint, que coincide con el módulo de mismo

número, surgió la necesidad, por parte del cliente, de buscar los comunicados, ya que, para tareas de administración o, incluso para facilitar a los usuarios normales encontrar cierto anuncio o sugerencia, podía ser de gran utilidad. Después de sopesar las consecuencias y, al tener una fecha fija para la entrega del producto, se decidió sustituir esto por el cambio de la tarea de “el usuario quiere poder borrar sus anuncios” por la de “el usuario quiere poder pedir al administrador que borre una sugerencia por email”.

Durante el cuarto módulo, se sabía que se necesitaba publicar ciertas funcionalidades por la red, es decir, el cliente deseaba que otras aplicaciones independientes pudieran operar con los comunicados. Por eso se eligió tanto REST como RPC (Web Services). Durante este mismo sprint, el cliente se encaprichó con la posibilidad de poder ver los comunicados desde su Samsung Galaxy S, ya que, bajo su punto de vista, hacía del producto algo tecnológicamente a la vanguardia. Llegó hasta el punto de asignarle la mayor prioridad. Por ese motivo está antes que la paginación y la búsqueda de comunicados. A cambio, el cliente desestimó la tarea de “el usuario quiere que le notifiquen por qué se ha rechazado su comunicado”.

Esto es un ejemplo de cómo historias que podían parecer importantes al principio, luego son descartadas; y de cómo durante las iteraciones, el cliente se puede dar cuenta de ciertos requisitos importantes que podían haber pasado desapercibidos al comienzo del desarrollo, saliendo a la luz solo después de ver la evolución del producto.

Por supuesto, el cliente quedó contento con el producto, ya que se abordó lo que él consideraba que le daba mayor valor de negocio y, además, con la sensación de que se ajustó el desarrollo debidamente a sus exigencias, sin la desagradable repercusión que comúnmente se da si se hubiera realizado el proyecto a la antigua usanza, donde no hubiera tardado en salir, con el contrato en la mano, el jefe del proyecto pidiendo explicaciones por los cambios que necesita realizar y las repercusiones negativas que pueden llegar a tener.



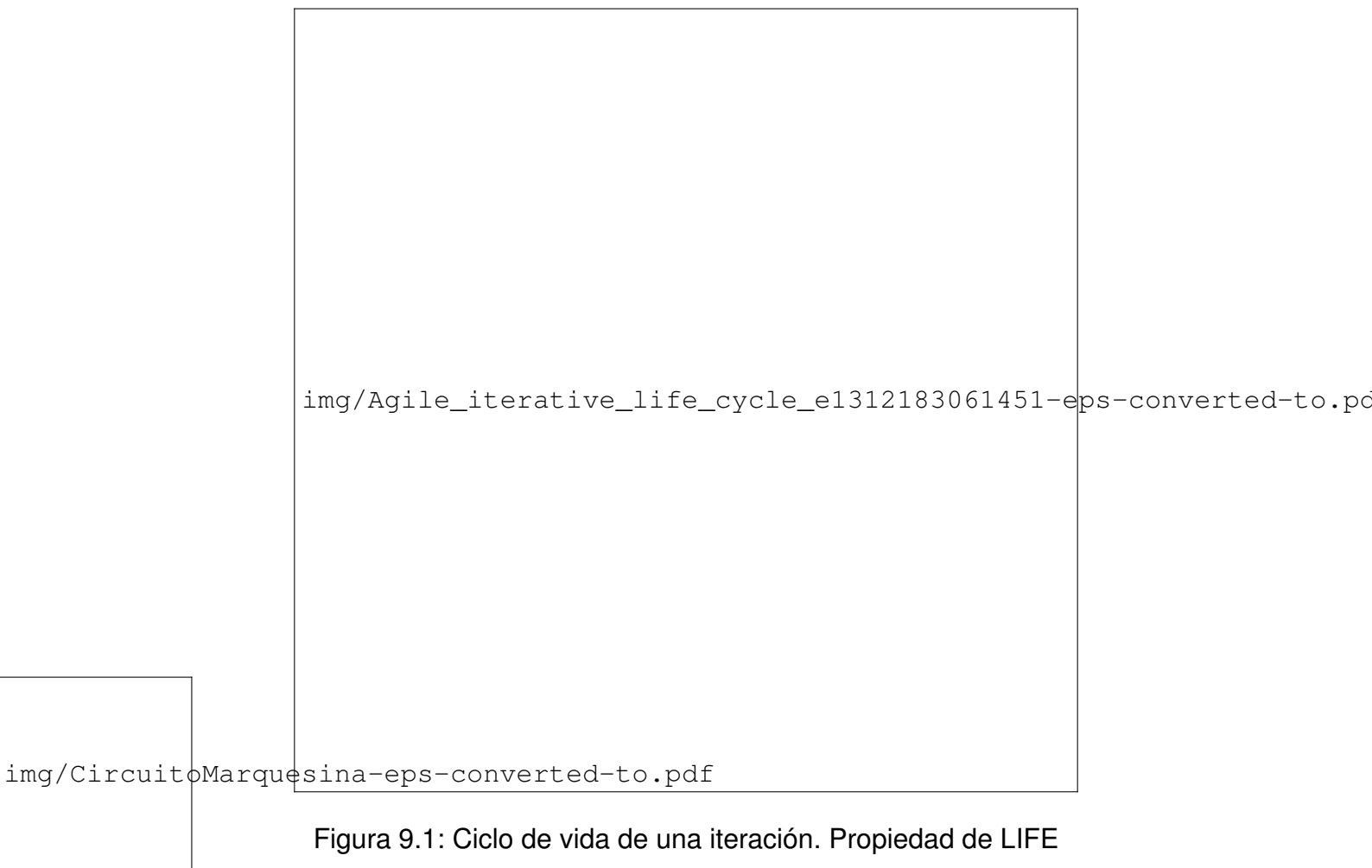


Figura 9.1: Ciclo de vida de una iteración. Propiedad de LIFE

De cara a explicar un poco como se va a abordar el resto de la documentación, conviene aclarar en este punto que, debido a que se ha usado BDD durante el desarrollo del proyecto, no se van a describir los tests en un apartado distinto a la explicación del código relevante del que se va a hablar en el siguiente capítulo, si no que formará parte importante en la explicación de este. Por eso no cuenta con un capítulo exclusivo para él.

9.2. PRIMER MODULO: PREPARACIÓN Y GESTIÓN DE USUARIOS

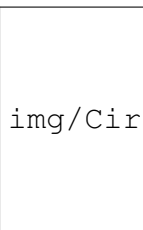
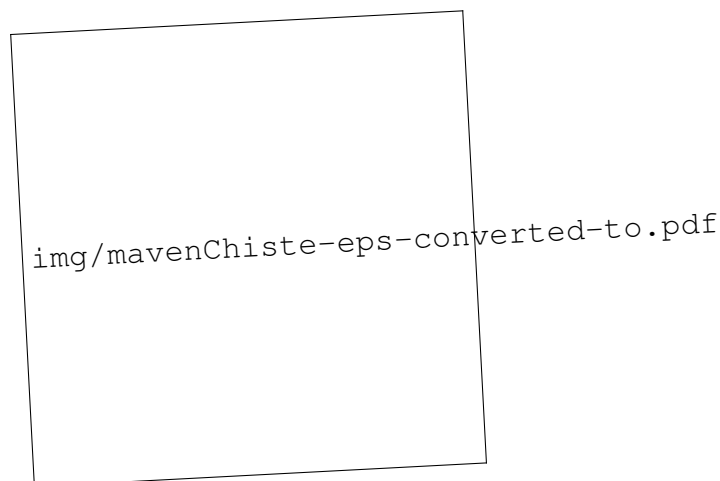
El primer módulo entregado al cliente coincide con el desarrollo realizado durante el sprint 0 y el primer sprint. Corresponde con la preparación del entorno, la familiarización con las herramientas y la gestión de usuarios. Durante la plani-

ficación de este sprint se decide, junto con el cliente, empezar con las historias de usuario que estén relacionadas con el sistema operativo y autenticación del sistema, por lo que las historias 8000 y 7900 pasan a tener la mayor prioridad.

8000 - El dueño de la aplicación quiere que sea una aplicación web fácil de usar, portable e independiente.

7900 - El dueño de la aplicación quiere que todo usuario tenga que autenticarse en el sistema.

Como lema de este sprint, se decide la siguiente frase: "El usuario puede acceder al sistema."



La parte más ardua fue investigar y aprender cómo funciona el arquetipo de Maven Appfuse, que el equipo eligió después de estudiar varias maneras de iniciar el proyecto por que encajaba con los objetivos que se quieren desarrollar y permitía empezar de manera más rápida al tratarse de un arquetipo. Este arquetipo viene con versiones modernas de Strut, Spring y se puede configurar para que use Hibernate y Oracle, además ofrece la estructura básica de proyecto para empezar a trabajar directamente. De todas formas, y como no es oro todo lo que reluce, hacer que funcionase con las especificaciones que se requerían trajo bastantes complicaciones. Destacar sobre todo la integración con Oracle, que fue bastante complicada, desde encontrar la librería, la cual no estaba en ningún repositorio de Maven, hasta conseguir instalarla en el repositorio local. Dicha dificultad también estaba causada por la inexperiencia en el uso de Maven a ese nivel.

9.2.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

HISTORIA 8000 - DESPLIEGUE DE APLICATIVO

Para acometer con la tarea 8000, es decir, hacer la aplicación independiente del sistema operativo y fácilmente portable, se decide usar J2SE, que es independiente del sistema operativo. Como contenedor de servlets Tomcat 5.5, también independiente. La facilidad de instalación la proporciona maven que, una vez configurado su repositorio local, es capaz de descargar todas las dependencias que el sitio necesita, pasar una batería de test e instalar el sistema. Solo se necesitaría instalar la base de datos Oracle y Maven en el servidor que fuera a albergar la web. Si se hubiera dispuesto de un entorno de preproducción y se hubiera usado integración continua, simplemente habría que pasar el war generado para su despliegue en el entorno de producción, pero esto está fuera del alcance de este proyecto.

img/CircuitoMarquesina-eps-converted-to.pdf

7900 - AUTENTICACIÓN EN EL SISTEMA

A la hora de seleccionar esta historia de usuario surgen varias preguntas que se realizan directamente al dueño del producto: ¿Cómo se registra un nuevo usuario? ¿Qué datos se quieren almacenar de este? ¿Se autentica por nombre de usuario y contraseña?

Tras contestar a estas preguntas, se generan las siguientes tareas:

- 1.- El usuario se autentica con nombre de usuario y contraseña.
- 2.- Si se le olvidó la contraseña, algún mecanismo para que pudiera recordarla.
- 3.- El usuario puede darse de alta añadiendo sus datos al sistema.
- 4.- No puede haber más de un usuario con el mismo email.

5.- No puede haber más de un usuario con el mismo nombre de usuario.

A pesar de las complicaciones a la hora de configurar el proyecto, gracias a usar el arquetipo de Maven Appfuse, el acceso de usuarios viene ya resuelto por defecto. La agradable sorpresa fue que con pocas modificaciones por nuestra parte pudimos implementar lo que el dueño del producto pedía.

En el proyecto se utiliza Spring para garantizar el acceso a los recursos de manera correcta, indicando cuales son los recursos que necesitan autenticación y qué roles son los permitidos. En el archivo “security.xml” es donde se configuran estos accesos. Se tiene que indicar en qué rutas se necesita un determinado rol y cuales son de acceso anónimo.

```
...
<http auto-config="true" lowercase-comparisons="false">
  <intercept-url pattern="/admin/*" access="ROLE_ADMIN"/>
  <intercept-url pattern="/passwordHint.html*" access="ROLE_ANONYMOUS,ROLE_ADMIN,
    ROLE_USER"/>
  <intercept-url pattern="/signup.html*" access="ROLE_ANONYMOUS,ROLE_ADMIN,
    ROLE_USER"/>
  <intercept-url pattern="/**/*.*.html*" access="ROLE_ADMIN,ROLE_USER"/>
  <form-login login-page="/login.jsp"
    authentication-failure-url="/login.jsp?error=true"
    login-processing-url="/j_security_check"/>
  <remember-me user-service-ref="userDao" key="e37f4b31-0c45-11dd-bd0b-0800200
    c9a66"/>
</http>

<authentication-manager>
  <authentication-provider user-service-ref="userDao">
    <password-encoder ref="passwordEncoder"/>
  </authentication-provider>
</authentication-manager>
...
```

img/Cir

Los roles son configurables también. En este aplicativo se necesitan dos: el rol de usuario y el de administrador, que ya vienen definidos por defecto.

Se realizaron algunos toques de estilo en la vista de acceso y en la de registro, pero poco más. Ver figura 9.2

Por otro lado, se generó una clase “Usuario” para abstraer la parte de Appfuse con respecto al resto del aplicativo. No se quiere hacer la funcionalidad del aplicativo dependiente de las funcionalidades que trae por defecto Appfuse, por lo que se generó también la clase “UsuarioDao” para mapear los datos de un lado a otro. Durante el resto de la aplicación, siempre que se necesite un objeto del tipo usuario, se hará usando el propio.

Tras las dificultades iniciales, se entrega en la demo lo acordado con el cliente y se finaliza el primer sprint. Durante la reunión de replanificación, se observa que el siguiente tema a abordar de una manera lógica y que tuviese valor para el cliente era las operaciones a nivel de usuario con los comunicados. Es decir, ver los comunicados y añadirlos, pasando las historias de usuario implicadas a tener mayor prioridad.

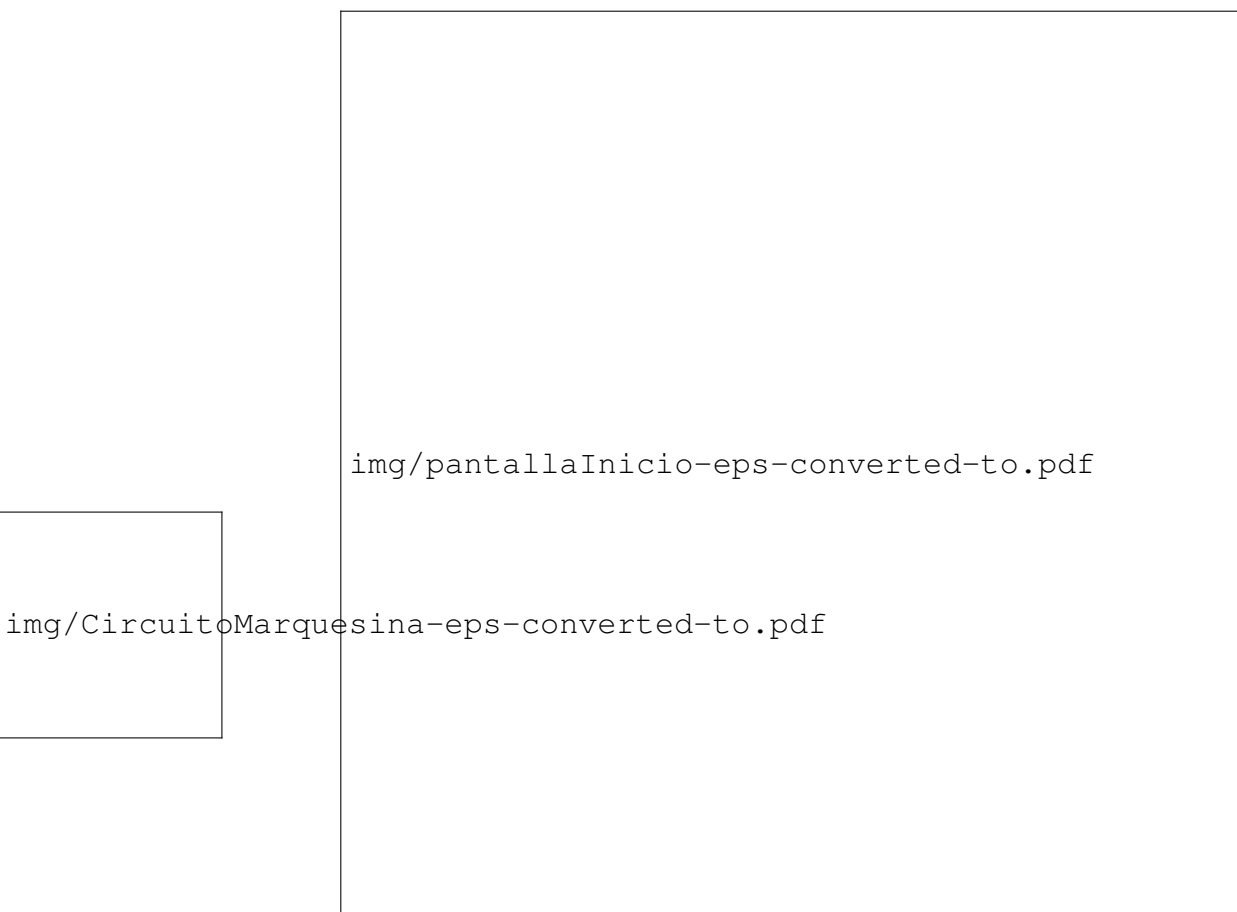


Figura 9.2: Pantalla de acceso.

9.2.2. DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 17 puntos de historia.

Como se puede observar en la figura 9.3 el ritmo es bastante bueno, si los dividimos 60 puntos totales de historia en 6 sprints la media sería 10 puntos de

historia por sprint, por tanto, esto nos avisa que el ritmo es muy elevado o que se han sobrestimado la puntuación de las historias de usuario, lo que se podría concluir que se pueden abarcar menos puntos de historia en el siguiente sprint o meter más puntos de historias y por lo tanto más funcionalidad.



Figura 9.3: Diagrama de Burndown. Sprint 1.

9.3. SEGUNDO MODULO: OPERACIONES CON COMUNICADOS

En este módulo, realizamos las operaciones básicas con los comunicados. Esta parte de la implementación corresponde al sprint 2. Como frase que definiera la meta para este sprint, se acordó junto con el dueño del producto, la siguiente: “Un usuario debe poder ver los comunicados e insertarlos”.

Para conseguir cumplir los objetivos, Se tomó la decisión de realizar las

tareas de operación primero, antes de otras tareas con mayor prioridad y así, al finalizar el sprint, se pudo mostrar un producto donde el cliente puede realizar las primeras operaciones completas, es decir, ver los comunicados y generarlos.

Siguiendo con BDD, en primera instancia se realizaron los test. Se puede observar un test unitario básico que nos sirve tanto para probar el correcto funcionamiento del comunicado como la API de esta clase. Siempre, gracias a los test unitarios, se añade un extra a parte de probar el comportamiento del sujeto bajo pruebas, y es que tenemos comentada la API de la clase con ejemplos prácticos.

Se mostrará como ejemplo el código completo de los test de comportamiento de “Comunicado” solo en este caso, para los casos posteriores se señalarán nada más que los aspectos que se entienden más importantes.

```
public class BComunicado extends TestCase {

    private IComunicado comunicadoUT;
    private SimpleDateFormat format;

    private String titulo = "Titulo UT";
    private String descripcion = "Descripcion UT";
    private Date fecha;

    @Before
    public void setUp() {
        format = new SimpleDateFormat("dd/MM/yyyy HH:mm");
        fecha = new Date();
    }

    @Test
    public void testGeneracionDeUnComunicadoDeTipoSugerenciaCorrecto()
        throws ParametrosErroneosComunicadoException {

        comunicadoUT = new Comunicado(titulo, descripcion,
            IComunicado.Tipo.sugerencia);

        comunicadoUT.guardar();

        assertEquals(titulo, comunicadoUT.getTitulo());
        assertEquals(descripcion, comunicadoUT.getDescripcion());
        assertEquals(format.format(fecha), format.format(comunicadoUT
            .getFechaInserccion()));
    }

    @Test
    public void testGeneracionDeUnComunicadoDeTipoAnuncioCorrecto()
        throws ParametrosErroneosComunicadoException {

        comunicadoUT = new Comunicado(titulo, descripcion,
            IComunicado.Tipo.sugerencia);

        comunicadoUT.guardar();

        assertEquals("Titulo UT", comunicadoUT.getTitulo());
        assertEquals("Descripcion UT", comunicadoUT.getDescripcion());
        assertEquals(format.format(fecha), format.format(comunicadoUT
            .getFechaInserccion()));
    }
}
```

```
}

@Test
public void testGeneracionDeUnComunicadoConTituloNuloFalla() {

    String titulo = null;
    boolean res = false;
    comunicadoUT = new Comunicado(titulo, descripcion,
        IComunicado.Tipo.sugerencia);

    try {
        comunicadoUT.guardar();
    } catch (ParametrosErroneosComunicadoException e) {
        res = true;
    }
    if (!res)
        fail("No Hemos capturado excepción");
}

@Test
public void testGeneracionDeUnComunicadoConTituloVacioFalla() {

    String titulo = "";
    boolean res = false;
    comunicadoUT = new Comunicado(titulo, descripcion,
        IComunicado.Tipo.sugerencia);

    try {
        comunicadoUT.guardar();
    } catch (ParametrosErroneosComunicadoException e) {
        res = true;
    }
    if (!res)
        fail("No Hemos capturado excepción");
}

@Test
public void testGeneracionDeUnComunicadoConDescripcionNulaFalla() {

    String descripcion = null;
    boolean res = false;
    comunicadoUT = new Comunicado(titulo, descripcion,
        IComunicado.Tipo.sugerencia);

    try {
        comunicadoUT.guardar();
    } catch (ParametrosErroneosComunicadoException e) {
        res = true;
    }
    if (!res)
        fail("No Hemos capturado excepción");
}

@Test
public void testGeneracionDeUnComunicadoConDescripcionVacíaFalla() {

    String descripcion = "";
    boolean res = false;
    comunicadoUT = new Comunicado(titulo, descripcion,
        IComunicado.Tipo.sugerencia);

    try {
        comunicadoUT.guardar();
    } catch (ParametrosErroneosComunicadoException e) {
        res = true;
    }
    if (!res)
```

img/Circui

```

        fail("No Hemos capturado excepción");
    }

    @Test
    public void testValidarAnuncios()
        throws ParametrosErroneosComunicadoException {
        // creamos un comunicado
        IComunicado comunicadoUT = new Comunicado(IComunicado.Tipo.sugerencia);
        comunicadoUT.setTitulo("Titulo UT");
        comunicadoUT.setDescripcion("Descripcion UT");
        comunicadoUT.guardar();

        Usuario userUT1 = new Usuario(1L, "JJ", "Juanito", "Perez",
            Usuario.Rol.administrador, "mail");

        try {
            comunicadoUT.modificarEstado(IComunicado.Estado.publicado,
                userUT1);
        } catch (UsuarioExcepcion e) {
            fail("Ha saltado excepción");
        }
    }

    @Test
    public void testValidarAnunciosUsuarioFalla()
        throws ParametrosErroneosComunicadoException {

        // creamos un comunicado
        boolean res = false;
        IComunicado comunicadoUT = new Comunicado(IComunicado.Tipo.sugerencia);
        comunicadoUT.setTitulo("Titulo UT");
        comunicadoUT.setDescripcion("Descripcion UT");
        comunicadoUT.guardar();
        Usuario userUT2 = new Usuario(1L, "JJ", "Juanito", "Perez", Usuario.Rol.
            usuario, "mail");

        try {
            comunicadoUT.modificarEstado(IComunicado.Estado.publicado,
                userUT2);
        } catch (UsuarioExcepcion e) {
            res = true;
        }
        if (!res)
            fail("No Hemos capturado excepción");
    }
}

```

Llegado el momento, surgió la necesidad de usar como ayuda los útiles objetos Mock. Estos son objetos ficticios que abstraen la generación de clases que no están involucradas en los objetivos de ese momento, ya que, a partir de una interfaz, es capaz de generar en tiempo de ejecución un objeto que la implementa y que se comporta como el desarrollador necesita para sus test. A continuación ponemos un ejemplo del código.

```

Mockery mock;
IComunicado comunicadoMock;
IComunicadoManager managerUT;

```

```

@Before
public void setUp() {
    mock = new Mockery();
    comunicadoMock = mock.mock(IComunicado.class);
    managerUT = new ComunicadoManager();

    mock.checking(new Expectations() {
        {
            allowing(comunicadoMock).getEstado();
            will(returnValue(IComunicado.Estado.publicado));
        }
    });
    mock.checking(new Expectations() {
        {
            allowing(comunicadoMock).getTipo();
            will(returnValue(IComunicado.Tipo.anuncio));
        }
    });
    mock.checking(new Expectations() {
        {
            allowing(comunicadoMock).getCodigo();
            will(returnValue("0000"));
        }
    });
    mock.checking(new Expectations() {
        {
            allowing(comunicadoMock).getTitulo();
            will(returnValue("Soy un mock"));
        }
    });
}

@Test
public void testAnyadirComunicadoAlManagerDeComunicadosDeManeraCorrecto() {
    managerUT.inserta(comunicadoMock);
    Assert.assertEquals(comunicadoMock, managerUT.obtener(0));
}

@Test
public void testAnyadirComunicadoRepetidoEsIncorrectoYNoSeAnyade() {
    managerUT.inserta(comunicadoMock);
    managerUT.inserta(comunicadoMock);
    Assert.assertEquals(1, managerUT.getListaComunicados().size());
}

@Test
public void testObtenerComunicadoPorIndiceSiNoExisteDebeDevolverObjetoNulo() {
    managerUT.inserta(comunicadoMock);
    Assert.assertNull(managerUT.obtener(managerUT.getListaComunicados().size()
        ()+1));
}
}

```

img/Cir

En este código de ejemplo se añade un nuevo comunicado, en este caso de tipo anuncio, aunque podría haber sido de cualquier otro tipo ya que no afectaría a la implementación. Para comprobar que la clase tipo “IComunicadoManager” funciona correctamente y como se quiere, se abstrae del resto de la implementación creando un mock de la interfaz de “IComunicado”. Esto es lo que se define como test unitario dentro de BDD, se prueba solo la funcionalidad de la clase que

esta bajo pruebas. Siempre se sabrá cual es ya que el objeto de prueba lleva en su nombre el sufijo “UT” que viene de “under test”. En caso de fallar este sujeto de prueba en algún momento, se localizaría el error dentro de su test.

Como podemos observar es en el método setUp() donde se define el comunicado como un objeto mock que implementa la interfaz “IComunicado”. Se redefine mock.cheking para indicar qué se espera recibir cuando se realice la comprobación, es decir, se define específicamente el comportamiento del mock.

Se quiere recalcar la utilidad de los objetos Mocks en la realización de los test. El programador se puede centrar en una parte del desarrollo y la parte que es necesaria pero ajena a lo que queremos validar en el test, se pueden utilizar estos objetos e implementar más tarde su funcionalidad. Con ello podemos realizar tests sin pararnos a implementar clases que no se van a realizar en este sprint.

Al realizar los test, se observó la necesidad de hacer en este sprint la base de datos para así poder realizar las comprobaciones pertinentes en la capa de acceso a datos. En esta metodología de programación, el código no se adapta a la base de datos. Es usual en otras metodologías que sea lo primero que se implementa, sin embargo, en el desarrollo ágil, se abstrae la implementación del modelado de la base de datos. Si durante el desarrollo se necesita realizar cambios que terminan afectando al diseño de la base de datos, no supondrá un gran esfuerzo, por lo que no deriva en un fracaso en la realización de la arquitectura.

En este caso, se decide implementar el acceso a los datos de la entidad “comunicado”, por tanto solo se tiene que añadir lo referente a la obtención e inserción de comunicados en la base de datos.

Para la generación del modelo de datos, se delega en Hibernate, haciendo uso de las notaciones. Por lo tanto, las tablas se generarán o modificarán al arrancar el aplicativo de manera automática.

En el siguiente código de ejemplo se muestra la definición de la clase “ComunicadoPojo” que será mediadora entre la información en base de datos y Java.

```
@Embeddable
@Entity
@Table(name = "t_comunicado")
public class ComunicadoPojo extends BaseObject {

    private static final long serialVersionUID = 1L;
```



```

/** \brief El id del comunicado. Es la clave primaria. */
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

/**\brief El codigo del comunicado. */
@Column(name = "codigo", length = 254, nullable = false)
private String codigo;

/**\brief El estado del comunicado Usa la clase #GenericEnumUserType para mapear el
 * enum.
 * */
@Column(name = "estado", columnDefinition = "integer", nullable = true)
@Type(type = "us.es.comunica.webapp.model.GenericEnumUserType", parameters = {
    @Parameter(name = "enumClass", value = "us.es.comunica.webapp.negocio.
        IComunicado$Estado"),
    @Parameter(name = "identifierMethod", value = "toInt"),
    @Parameter(name = "valueOfMethod", value = "fromInt") })
private Estado estado;

/**
 * \brief El tipo del comunicado. Usa la clase #GenericEnumUserType para mapear el
 * enum
 */
@Column(name = "tipo", columnDefinition = "integer", nullable = true)
@Type(type = "us.es.comunica.webapp.model.GenericEnumUserType", parameters = {
    @Parameter(name = "enumClass", value = "us.es.comunica.webapp.negocio.
        IComunicado$Tipo"),
    @Parameter(name = "identifierMethod", value = "toInt"),
    @Parameter(name = "valueOfMethod", value = "fromInt") })
private Tipo tipo;

/**\brief El título del comunicado. */
@Column(name = "titulo", length = 254, nullable = false)
private String titulo;

/**\brief La descripción del comunicado. */
@Column(name = "descripcion", length = 2000, nullable = true)
private String descripcion;

/** \brief La fecha de insercción del comunicado. */
@Column(name = "fecha_inserccion")
private Date fechaInserccion;

/**\brief La dirección de la imagen del comunicado. */
@Column(name = "ruta_fichero", length = 255, nullable = true)
private String rutaFichero;

/** \brief Si el comunicado está o no borrado. */
@Column(name = "borrado", nullable=false, columnDefinition="number(1) default 0")
private boolean borrado;

@OneToMany(mappedBy="comunicado", fetch=FetchType.EAGER, targetEntity=RespuestaPojo.
    class )
private List<RespuestaPojo> respuestas;
...
}

```

img/Cir

Por experiencias previas en otros desarrollos, se aprovecha la posibilidad de utilizar una base de datos limpia cada vez que se inicia el aplicativo mientras se está en la etapa de desarrollo del código, así se asegura que las pruebas que se realizan sobre la base de datos son inocuas, es decir, la ejecución de los test,

no modificará la base de datos original. Maven nos permite volver a cargar los datos de prueba, limpiándola cada vez que se inicia el servicio.

9.3.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

HISTORIA 7800 - INSERTAR COMUNICADO.

La historia 7800 es realmente dos historias de usuario que se han unido, estas hacían diferencia entre sugerencia y anuncio, se decidió unirlos semánticamente usando la palabra comunicado. Para ello se utilizará una nomenclatura llamada “tipo” con los posibles valores, sugerencia y anuncio, para diferenciar cada caso.

Para realizar esta historia de usuario la dividimos en las siguientes tareas:

- 1.- Rellenar datos de una sugerencia o anuncio (título, descripción e imagen asociada).
- 2.- Salvar una nueva sugerencia o anuncio.

Para realizar la primera tarea, se construyó la estructura de comunicado, creando un test de tipo “comunicado behaviour” como se ha explicado en el apartado anterior. Para ello se crea el comunicado con los datos necesarios. Primero se implementaría un atributo Tipo, de tipo enumerado que describe si el comunicado es un anuncio o una sugerencia. Se escribe la interfaz y la clase a medida que se van escribiendo las pruebas, implementando lo imprescindible para que pasen.

Para la segunda tarea, era necesario comprobar si se han salvado correctamente los datos, para ello se tiene que comprobar la base de datos, usando para esta labor Hibernate, que abstrae el acceso a los datos. Se escribe la clase ComunicadoPojo, la cual se utilizará como pasarela entre los datos en Java y en la base de datos. En el apartado anterior se ha explicado el funcionamiento de esta clase.

Para la capa de acceso a datos, se escribe la clase `ComunicadoDaoHibernate`, que hereda de `GenericDaoHibernate`, clase que proporciona Hibernate. En ella se definen consultas que son necesarias y que no las proporciona Hibernate por defecto.

```
@Repository("ComunicadoDao")
public class ComunicadoDaoHibernate extends
    GenericDaoHibernate<ComunicadoPojo, Long> implements IComunicadoDao {

    ...
    /**
     * (non-Javadoc)
     *
     * @see
     * us.es.comunica.webapp.dao.ComunicadoDao#getByEstado(us.es.comunica.webapp
     * .negocio.IComunicado.Estado)
     */
    public List<ComunicadoPojo> getByEstado(Estado estado) {
        return getHibernateTemplate().find(
            " from ComunicadoPojo where estado = ? and borrado =
            false", estado);
    }

    /**
     * (non-Javadoc)
     *
     * @see us.es.comunica.webapp.dao.ComunicadoDao#getById(java.lang.Long)
     */
    public ComunicadoPojo getById(Long id) {
        return this.get(id);
    }

    /**
     * (non-Javadoc)
     *
     * @see
     * us.es.comunica.webapp.dao.ComunicadoDao#getByTipo(us.es.comunica.webapp
     * .negocio.IComunicado.Tipo)
     */
    public List<ComunicadoPojo> getByTipo(Tipo tipo) {
        return getHibernateTemplate().find(
            " from ComunicadoPojo where tipo = ? and borrado = false
            ", tipo);
    }

    /**
     * (non-Javadoc)
     *
     * @see
     * us.es.comunica.webapp.dao.ComunicadoDao#getByTipoYEstado(us.es.comunica
     * .webapp.negocio.IComunicado.Tipo,
     * us.es.comunica.webapp.negocio.IComunicado.Estado)
     */
    public List<ComunicadoPojo> getByTipoYEstado(Tipo tipo, Estado estado) {
        return getHibernateTemplate().find(
            " from ComunicadoPojo where tipo = ? and estado = ? and
            borrado = false", tipo,
            estado);
    }
}
```

img/Circui

El test `BComunicadoDAO` comprueba que, efectivamente, el comunicado se ha guardado correctamente.

```
...
@Test
public void testSaveComunicado() {
    ComunicadoPojo expected = new ComunicadoPojo();
    expected.setCodigo("Bas-01");
    expected.setDescripcion("Generado desde los test");
    expected.setEstado(Estado.publicado);
    expected.setFechaInserccion(new Date());
    expected.setTitulo("Básico");
    expected.setTipo(Tipo.anuncio);

    expected = comunicadoDao.save(expected);
    ComunicadoPojo actual = comunicadoDao.getById(expected.getId());

    assertTrue(expected.equals(actual));

    comunicadoDao.remove(expected.getId());
}
...
```

HISTORIA 7700 - LEER COMUNICADOS

img/CircuitoMarquesina-eps-converted-to.pdf

El detalle de la historia de usuario dice lo siguiente:

7700 - El usuario quiere leer los comunicados que se han publicado..

Esta historia de usuario está dividida en las siguientes tareas:

- 1.- Obtener comunicados ordenados por fecha descendiente
- 2.- Mostrar el listado de comunicados
- 3.- Mostrar el detalle del comunicado.

Al realizar la primera tarea, se observó que también se podía incluir la historia 7100. Esto es un ejemplo, tal y como se ha comentado anteriormente, de que, al subdividir en tareas, se puede modificar el alcance de las mismas. En este caso se observa, incluso, la existencia de tareas comunes para varias historias. La historia de usuario 7100 dice así:

7100 - El administrador quiere ver el listado de comunicados ordenados por fechas.

Esto hizo que fuera necesario definir dos roles: usuario y administrador, que se aplicarán a las personas autenticadas.

Al realizar una lectura de las historias de usuario, se pudo observar que existen conceptos que aún se estaban por definir y que era necesario concretar en esta historia, como por ejemplo, la figura de “publicación”.

Por tanto se realiza un repaso de las historias de usuario y se comprueba que en la siguiente historia de usuario 7500, introduce el concepto de estado publicado. Esta historia dice así:

7500 - El administrador quiere que solo puedan ser vistos por el usuario los anuncios y sugerencias publicados.

Se podrían haber unido las historias de usuario 7500 y 7700 para que expresase que los usuarios quieren ver los comunicados publicados, pero no se consideró relevante para la buena marcha del proyecto.

img/Cir

La historia de usuario 7500 es dividida en las siguientes tareas y se decide implementar primero esta historia de usuario para poder implementar la prioritaria tal y como se tenía previsto.

- 1.- Filtrar los comunicados por tipo, anuncio o sugerencia
- 2.- Filtrar los comunicados por Estado, publicado o rechazado

Por tanto esta tarea tendrá una clase de tipo filtro cuya función es recoger de la base de datos los comunicados según tipo y estado.

Después de realizar esta planificación, se decide añadir todas estas historias para ejecutar durante el sprint.

Para imprimir los resultados se realiza la primera página .jsp del proyecto. Su estructura es simple, se le pasa un parámetro por la url indicando de que tipo de comunicado se trata, después internamente se comprueba si el usuario actual es administrador o no. Finalmente se filtra según los datos de entrada y se visualiza el resultado por pantalla. Ver figura 9.4.

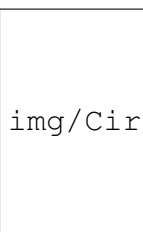


Figura 9.4: Pantalla que muestra un listado de anuncios

También se genera una vista para mostrar el detalle del comunicado. Simplemente el usuario tendrá que pinchar en el título del comunicado que desea ver cuando se encuentre dentro de la pantalla de listado.



Figura 9.5: Pantalla que muestra el detalle de un anuncio



Se añade la interfaz de inserción de un nuevo comunicado. Para ello se crean las acciones correspondientes para poder guardar un comunicado, asimismo se crea una pantalla para la inserción de los datos.

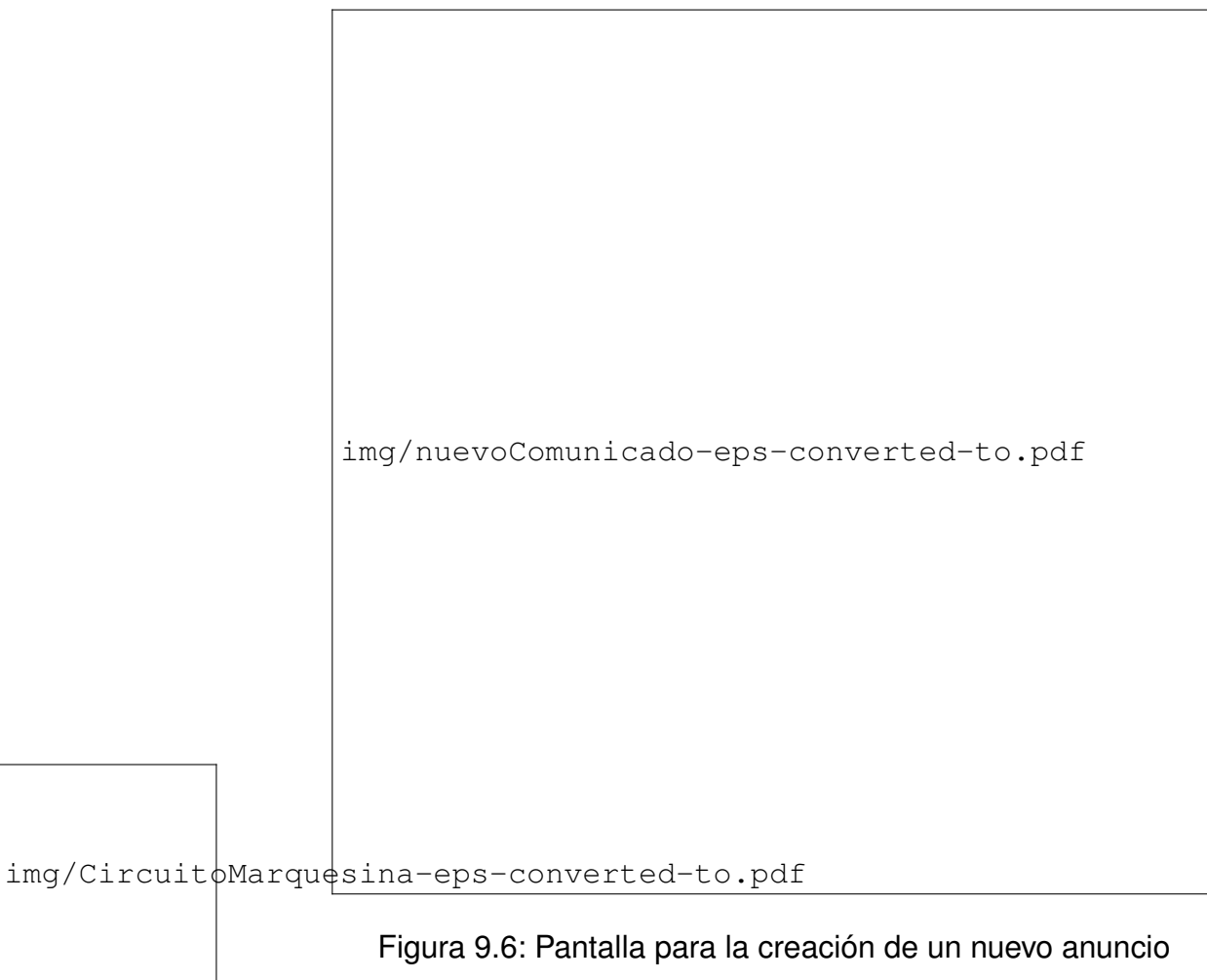


Figura 9.6: Pantalla para la creación de un nuevo anuncio

Con la realización de estas nuevas funcionalidades damos por concluido el desarrollo de esta historia de usuario y finalizada la meta del sprint 2.

Como se observa, se ha acometido una parte de cada capa de abstracción, que va desde el acceso a datos hasta la vista. Así el usuario puede tener una visión completa de las funcionalidades implementadas. Esto no hubiera sido posible utilizando otras metodologías como el modelo en cascada.

9.3.2.DISEÑO DE CLASES

En este apartado se muestra como quedó el diseño emergente al realizar este módulo. Estas imágenes se pueden observar con más definición en la API entregada junto con el proyecto.

Se puede observar en la figura 9.7 la interacción en el patrón MVC y DAO, de tal manera que la división quedaría como sigue:

- 1.- La clase “BaseAction” es la clase de la que heredan los demás Actions de Struts y que se comunica directamente con el controlador que se gestiona mediante un fichero de configuración del framework Struts denominado “struts.xml”.
- 2.- La clase “ComunicadoAction” tiene implementada la lógica de negocio de este módulo.
- 3.- La clase “ComunicadoBean” es el nexo de unión entre la vista y la lógica de negocio, ya que la instancia de esta clase que se le pasa a “ComunicadoAction” contiene los datos de entrada del usuario cuando es necesario.
- 4.- La interfaz “IComunicadoDao” forma parte del patrón DAO y abstrae a la lógica de negocios de obtener los datos, en este caso, de la base de datos.

img/Cir

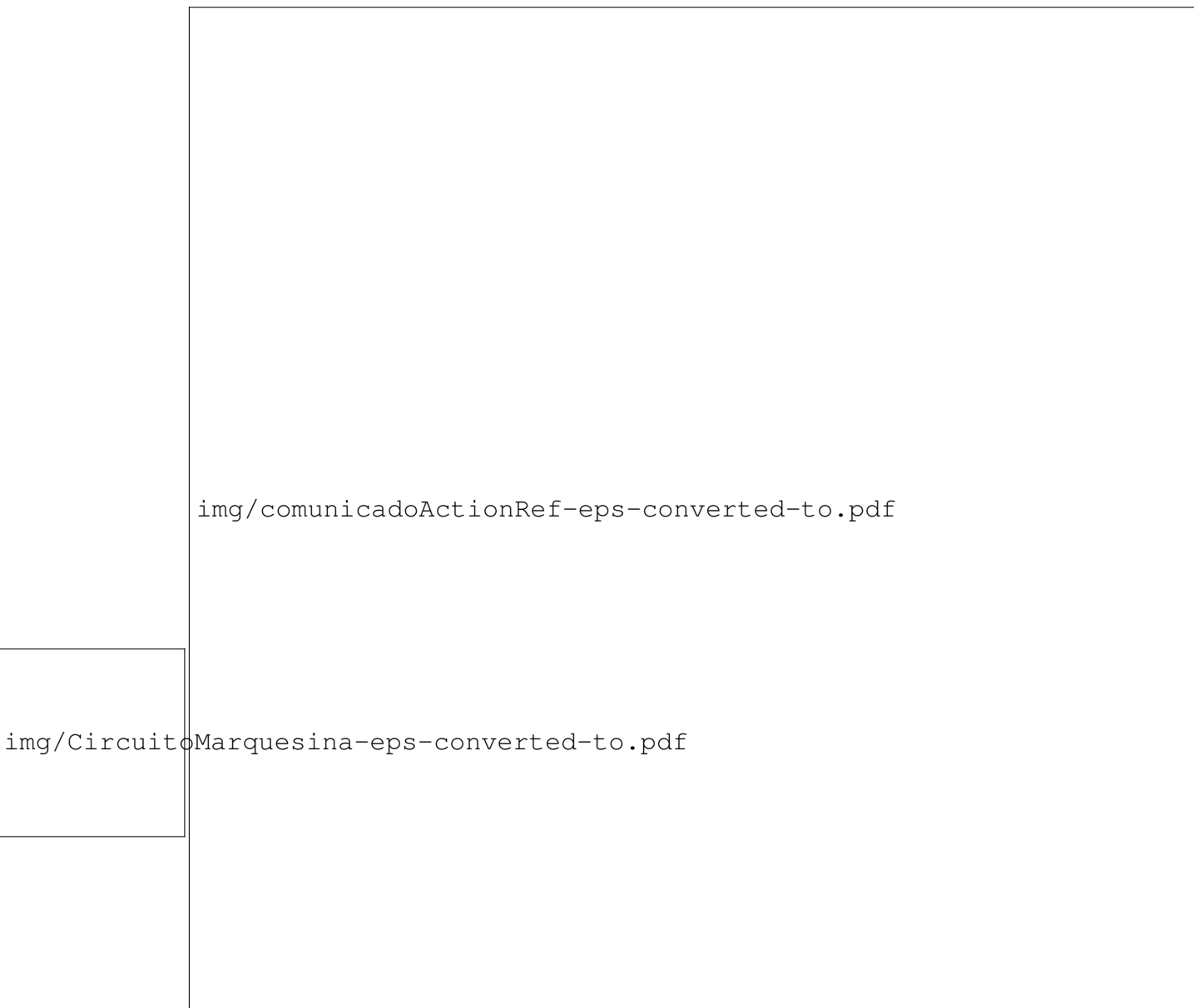


Figura 9.7: Relación entre clases de las operaciones básicas con comunicados.

9.3.3. DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 11 puntos de historia.

Como se puede observar en la figura 9.8, el ritmo está por encima de la

media, por tanto se puede decir la evolución del proyecto sigue la estimación propuesta.

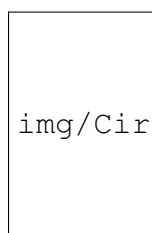


Figura 9.8: Diagrama de Burndown. Sprint 2.

9.4. TERCER MODULO: ADMINISTRACIÓN DE COMUNICADOS

En este módulo, se realizan las operaciones con los comunicados. Esta parte de la implementación corresponde al sprint 3. Se tomó la decisión de realizar las tareas de administración primero, antes que otras tareas con mayor prioridad, para que al finalizar la meta del sprint se pueda mostrar un producto donde el cliente pueda realizar las primeras operaciones de administración del sistema. El dueño del producto estuvo de acuerdo, por lo que se cambiaron las prioridades de las historias de usuario de las historias de usuario que empezaron siendo las 9000, 8750 y 8500.

Se puede observar que, de todas maneras, estas tareas, aunque tienen una prioridad importante por ser desconocidas para el equipo de desarrollo, no son las que más valor de negocio dan al cliente. Es el motivo por el que, a medida que se ha ido desarrollando el proyecto, han bajado de prioridad.

El lema que se va a utilizar como meta del sprint es “los usuarios administradores pueden administrar los comunicados.”

9.4.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

HISTORIA 7400 - RESPONDER COMUNICADOS

Para comenzar el desarrollo de esta historia de usuario, se crea un test llamado “RespuestaBehaviour”, a partir de este test se implementa una nueva clase llamada Respuesta, que estará relacionada con los comunicados y con los usuarios.

```
public void testRespuestaAdminSePublicaDirectamente() throws
    ParametrosErroneosComunicadoException
{
    Mockery context = new Mockery();

    final IComunicado comunicado = context.mock(IComunicado.class);
    context.checking(new Expectations() {{
        atLeast(1).of (comunicado).getId(); will(returnValue(new Long(1)));
    }});

    String respuesta = "Esto es una respuesta";
    //Respuesta 1
    Usuario userUT1 = new Usuario(1L,"JJ", "Juanito", "Perez", Usuario.Rol.
        administrador, "mail");
    IRespuesta respuestaUT1 = new Respuesta (userUT1, comunicado.getId(),
        respuesta);
    //Respuesta 2
    Usuario userUT2 = new Usuario(2L,"JJ2", "Juanito", "Perez", Usuario.Rol.
        usuario, "mail");
    IRespuesta respuestaUT2 = new Respuesta (userUT2, comunicado.getId(),
        respuesta);

    assertEquals(IRespuesta.Estado.aceptado, respuestaUT1.getEstado());
    assertEquals(IRespuesta.Estado.borrador, respuestaUT2.getEstado());
}
```

Como se puede observar se genera un objeto de tipo Mock para comunicado, puesto que el objetivo del test es comprobar que se crean correctamente las

respuestas.

También se generará una clase para manejar las respuestas de manera simple y controlada: RespuestasManager.

```
Mockery mock;
IComunicado comunicado;
IRespuestaManager manager;

@Before
public void setUp() {
    mock = new Mockery();
    comunicado = mock.mock(IComunicado.class);
    manager = new RespuestaManager(comunicado);
}

@Test
public void testAnyadirRespuestaOk()
    throws RespuestaActivaDuplicadaException {

    final IRespuesta respuesta = mock.mock(IRespuesta.class);
    mock.checking(new Expectations() {
        {
            oneOf(respuesta).getEstado();
            will(returnValue(IRespuesta.Estado.aceptado));
        }
    });

    mock.checking(new Expectations() {
        {
            oneOf(comunicado).getId();
            will(returnValue(new Long(1)));
        }
    });

    mock.checking(new Expectations() {
        {
            oneOf(respuesta).setIdComunicado(1L);
        }
    });

    manager.responder(respuesta);
    Assert.assertEquals(1, manager.getRespuestas().size());
}
```

img/Circui

Se genera la capa de acceso a datos con la implementación de la clase “RespuestaPojo”, en donde se define el paso de los datos de Java a la base de datos y viceversa.

```
@Entity
@Table(name = "t_respuesta")

public class RespuestaPojo extends BaseObject {

    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = 1L;

    /** The id. */
    @Id
```

```

@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

/** The respuesta. */
@Column(name = "respuesta", length = 2000, nullable = false)
private String respuesta;

/** The id usuario. */
@ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE}, targetEntity=
    User.class )
@JoinColumn(name="id_usuario")
private User usuario;

/** The id comunicado. */
@ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE}, targetEntity=
    ComunicadoPojo.class )
@JoinColumn(name="id_comunicado")
private ComunicadoPojo comunicado;

/**\brief El estado del comunicado Usa la clase #GenericEnumUserType para
    mapear el
    * enum.
    * */
@Column(name = "estado", columnDefinition = "integer", nullable = true)
@Type(type = "us.es.comunica.webapp.model.GenericEnumUserType", parameters = {
    @Parameter(name = "enumClass", value = "us.es.comunica.webapp.negocio.
        IRespuesta$Estado"),
    @Parameter(name = "identifierMethod", value = "toInt"),
    @Parameter(name = "valueOfMethod", value = "fromInt") })
private Estado estado;

/** The borrado. */
@Column(name = "borrado", nullable=false, columnDefinition="number(1) default 0")
private boolean borrado;
...

```

img/Circui

Se añade en la pantalla de visualización del detalle de un comunicado la posibilidad de responder un comunicado en caso de ser administrador.



Figura 9.9: Pantalla para responder a un comunicado

Una respuesta puede ser borrada o aprobada. Para que las respuestas sean visibles se deben de pasar del estado “borrador” al “publicado”. Por lo tanto se realizó la implementación a la par que esta historia de usuario. Esta acción corresponde con la historia usuario 6900.

6900 - El administrador quiere borrar sus respuestas.

HISTORIA 7300 - ESTADOS DE COMUNICADOS.

En esta historia, se añade el estado “Publicado” o “Rechazado”. En las tareas anteriores se realizó un filtro para que el usuario solo pueda ver los comuni-

cados publicados, en este caso, si el rol es administrador, la lista de comunicados no se filtra por estado publicado, sino que se muestran todas.

Dependiendo del estado en el que se encuentre un comunicado el administrador podrá rechazarlo o publicarlo.

Como se está gestionando la administración de los comunicados, se tomó la decisión de implementar la siguiente tarea:

6800 - El administrador quiere borrar comunicados.

Se realizó la historia de usuario añadiendo un botón de borrado en la vista. A parte se modificó en “ComunicadoHibernateDAO” la manera de borrar un comunicado ya que no se desea que se borre físicamente de la base de datos, si no que se necesita un borrado lógico para mantener un histórico de comunicados en la base de datos.

img/Circui

```
...
/* (non-Javadoc)
 * @see us.es.comunica.webapp.dao.IComunicadoDao#delete(java.lang.Long)
 */
public boolean delete(Long id) {
    try {
        ComunicadoPojo com = this.get(id);
        com.setBorrado(true);
        this.save(com);

    } catch (Exception e) {
        log.error("A ocurrido un error al borrar un comunicado:"
            + e.getMessage());
        return false;
    }
    return true;
}
...
```

Así pues a la finalización de esta historia de usuario, el administrador podría publicar, rechazar y borrar comunicados. Ver figura 9.10.

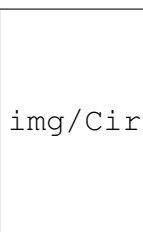
Con la realización de estas nuevas funcionalidades damos por concluido el desarrollo de esta historia de usuario y finalizada la meta del sprint 3.

En este caso, en la demo con el dueño del producto, se pudo probar todo el módulo de gestión de comunicados de inicio a fin, pudiendo realizar los

comentarios pertinentes sobre el trabajo en ese momento.



Figura 9.10: Pantalla para administrar un comunicado



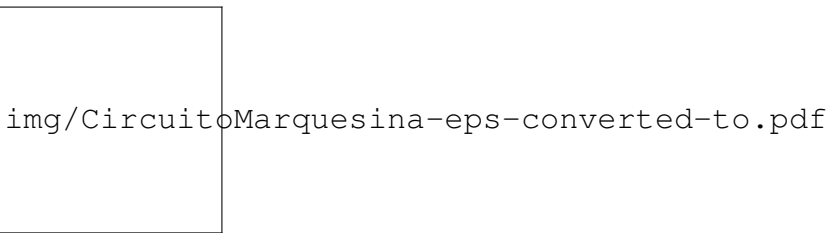
9.4.2.DISEÑO DE CLASES

En este apartado se muestra como quedó el diseño emergente al realizar este módulo. Estas imágenes se pueden observar con más definición en la API entregada junto con el proyecto.

Se puede observar en las figuras la interacción en el patrón MVC y DAO, de tal manera que la división quedaría como se procede a explicar en este apartado.

En la funcionalidad de administración de cambio de estado de un comunicado tenemos la figura9.11 que muestra la interrelación.

- 1.- La clase “BaseAction” es la clase de la que heredan los demás Actions de Struts y que se comunica directamente con el controlador que se gestiona mediante un fichero de configuración del framework Struts denominado “struts.xml”.
- 2.- La clase “ComunicadoWorkflowAction” tiene implementada la lógica de negocio de este módulo, es decir, el cambio de estado del comunicado.
- 3.- La interfaz “IComunicadoDao” forma parte del patrón DAO y abstrae a la lógica de negocios de obtener los datos, en este caso, de la base de datos.



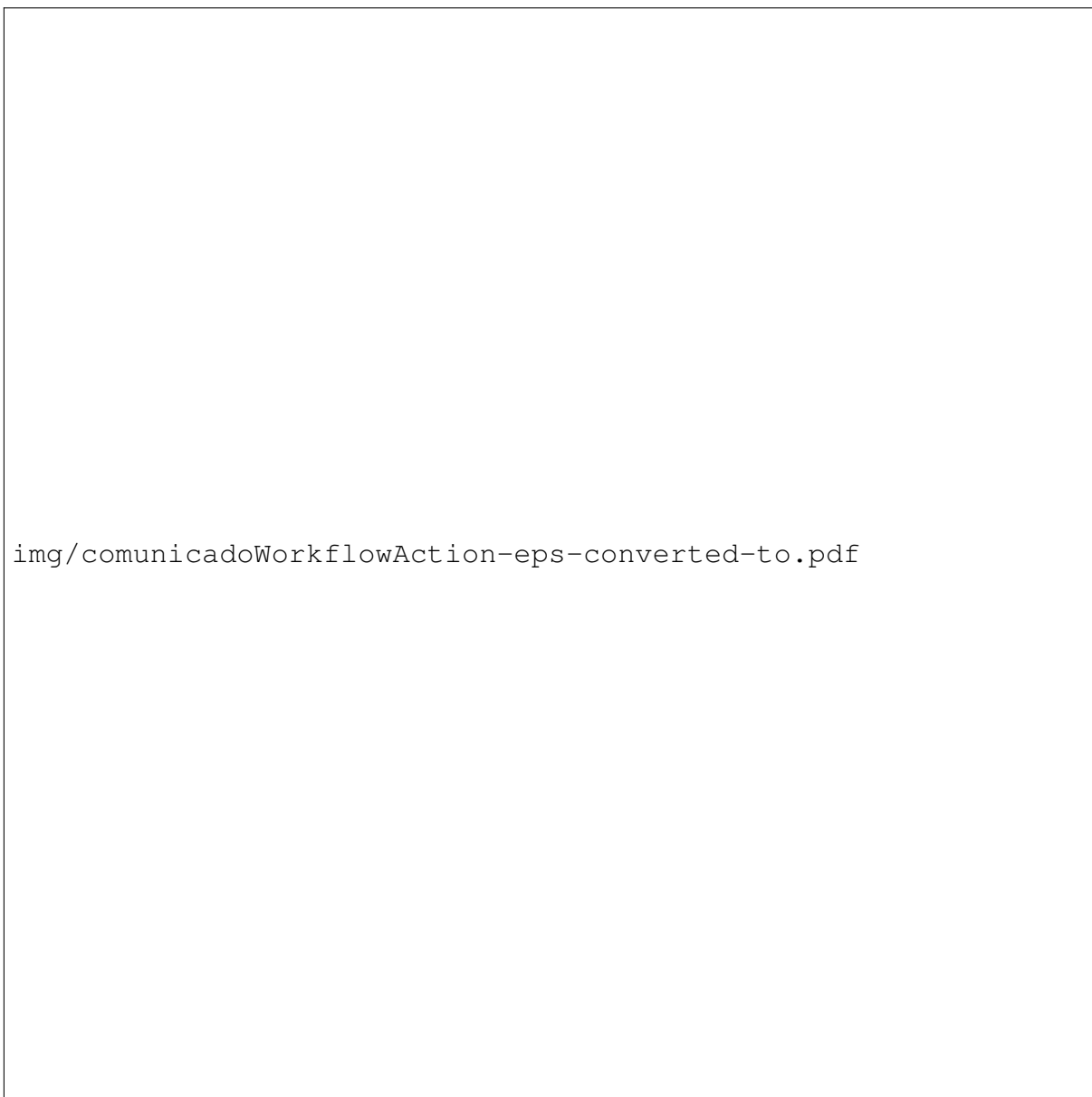


Figura 9.11: Relación entre clases de las operaciones de cambio de estado de comunicados.

En la funcionalidad de administración de respuesta a un comunicado tenemos la figura 9.12 que muestra la interrelación.

- 1.- La clase “BaseAction” es la clase de la que heredan los demás Actions de Struts y que se comunica directamente con el controlador que se gestiona mediante un fichero de configuración del framework Struts denominado “struts.xml”.

- 2.- La clase “RespuestaAction” tiene implementada la lógica de negocio de este módulo, es decir, responde a un comunicado, publicarlo o pasarlo a borrador.
- 3.- La interfaz “IComunicadoDao” y “IRespuestaDao” forma parte del patrón DAO y abstrae a la lógica de negocios de obtener los datos, en este caso, de la base de datos.
- 4.- La clase “RespuestaBean” genera una instancia de ella con los datos de entrada del usuarios provenientes de la vista. Es el nexa entre la vista y la lógica de negocio.



Figura 9.12: Relación entre clases de la operación de respuesta de comunicados.

9.4.3. DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 10 puntos de historia.



Figura 9.13: Diagrama de Burndown. Sprint 3.

9.5. CUARTO MODULO: COMUNICACIONES MEDIANTE SERVICIOS WEB

Durante la reunión de replanificación del sprint 4, se decidió, junto con el dueño del producto, acometer la funcionalidad que gestiona la comunicación con otros aplicativos. Al estar hablando de tecnología web, surgió necesariamente el concepto de servicios web. Se determinó que bastaba con un servicio de publi-

cación de procedimientos remotos (RPC) bajo el protocolo SOAP (Simple Object Access Protocol). También se planteó el uso de REST, quizás más interesante al estar orientado a los datos, y no a llamadas a procedimientos. Como no se llegó a una conclusión en principio de cual de las dos tecnologías usar, se realizó una tarea del tipo “spike” consistente en investigar ambas tecnologías y proponer el uso de una, otra o las dos.

9.5.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

HISTORIA 8400 - PUBLICACIÓN WEB SERVICES

En el sprint número cuatro solo se abordó esta historia de usuario debido a las necesidades de investigación que iba a requerir.

img/CircuitoMarquesina-eps-converted-to.pdf

La historia de usuario 8400 dice así:

8400 - El dueño de la aplicación quiere que se pueda operar fácilmente con otras aplicaciones con Comunic@.

Por lo tanto se generaron las siguientes tareas:

- 1.- Spike: investigar pros y contras de RPC-SOAP y REST.
- 2.- Spike: investigar cómo implementar la opción elegida (RPC o REST).
- 3.- Publicar listado de anuncios.
- 4.- Publicar listado de sugerencias.

El lema elegido como meta de este sprint fue: “Publicar de manera simple datos u operaciones para su uso por parte de otras aplicaciones”.

Estudiando durante el sprint qué alternativa era la mejor, el equipo se decantó por usar REST, ya que lo que se quería publicar era nada más que datos y ediciones o inserciones de estos datos, a parte de la sencillez para implementarlo tanto por la parte del servidor como por la del cliente. En concreto, se necesita devolver listados de comunicados, el detalle de un comunicado específico y, en un futuro, tener la posibilidad de editar un resultado.

Mientras se estudiaba qué se necesitaba para configurar REST en el aplicativo y cómo se utilizaba, se observó que, gracias a las anotaciones y a la modificación de un simple xml, se tenía resuelto el misterio, siendo muy parecida la solución tanto para REST como para RPC, por lo que se decidió que, si se disponía un poco de tiempo, una vez implementado REST se añadiría RPC. Y así se hizo.

Appfuse viene con las librerías CXF¹ que abstrae la parte de servicios web (tanto RPC-SOAP como REST), permitiendo el uso de anotaciones, lo que facilita mucho la implementación.

Entrando en detalle, se generó una clase “BComunicadoManager” que corresponde a las pruebas unitarias que detallan el comportamiento de la clase “ComunicadoManager”. En ella se dice qué se espera al llamar a los métodos “getListadoSugerencias” y “getListadoAnuncios”. A continuación ponemos un ejemplo de una de las pruebas.

```
...
@Test
public void testGetListadoAnuncios() {

    comunicadoManager = new ComunicadoManagerImpl(comunicadoDao);
    List<Comunicado> listadoAnunciosPublicados = comunicadoManager.
        getComunicadosAnuncios();
    assertTrue(listadoAnunciosPublicados.size()>0);
}
...
```

Con el uso de anotaciones, en la interfaz “ComunicadoManager” (que es la encargada de definir los métodos a los que se van a llamar para la publicación por servicios web) se hace tan sencilla la publicación de servicios como añadir una notación al inicio de la interfaz y otra en cada método que va a publicar datos. Se puede observar la simpleza en el siguiente código.

```
@WebService
@Produces({"text/xml","text/plain","application/json"})
@Path("/")
public interface ComunicadoManager extends GenericManager<ComunicadoPojo, Long>{
```

¹<http://cxf.apache.org/>

```

        @Path("/anuncios")
        @GET
        public List<Comunicado> getComunicadosAnuncios();
        @Path("/sugerencias")
        @GET
        public List<Comunicado> getComunicadosSugerencias();
    }

```

La clase “ComunicadoManagerImpl” implementa dicha interfaz y proporciona los datos haciendo llamadas a la capa de acceso a datos usando un objeto del tipo “IComunicadoDao”. En el siguiente código se puede observar la implementación del método “getComunicadosAnuncios”

```

...
public List<Comunicado> getComunicadosAnuncios() {
    List<Comunicado> res = getComunicados(Tipo.anuncio);
    return res;
}
...
private List<Comunicado> getComunicados(Tipo tipo) {
    IComunicadoDao aux = comunicadoDao;
    List<ComunicadoPojo> comunicadosPojo = aux.getByTipoYEstado(
        tipo, Estado.publicado);
    List res = ComunicadoPojo2IComunicado
        .listComunicadoPojo2Comunicado(comunicadosPojo)
        .getListComunicados();
    return (List<Comunicado>)res;
}
...

```

Para que todo funcione correctamente, hay que configurar el xml de CXF (cxf-servlet.xml), indicando en que rutas se van a publicar los servicios webs y que beans (en este caso, “ComunicadoManager”) se van a hacer cargo de la obtención de los datos.

```

...
<jaxws:endpoint id="userService" implementor="#userManager" address="/UserService"/>
<jaxws:endpoint id="ComunicadoService" implementor="#comunicadoManager" address="/
    ComunicadoService"/>
<!-- Add new endpoints for additional services you'd like to expose -->
<jaxrs:server address="/rest">
    <jaxrs:features>
        <cxfr:logging/>
    </jaxrs:features>
    <jaxrs:serviceBeans>
        <ref bean="comunicadoManager"/>
        <ref bean="userManager"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="jsonProvider"/>
    </jaxrs:providers>
</jaxrs:server>
<jaxrs:extensionMappings>

```



```
<entry key="json" value="application/json"/>
<entry key="xml" value="application/xml"/>
<entry key="feed" value="application/atom+xml"/>
</jaxrs:extensionMappings>
</jaxrs:server>

...
```

En el xml “applicationContext.xml” hay que definir el bean “comunicadoManager”, que debe ser la clase que contiene la implementación específica para la obtención de los comunicados y sugerencias.

```
<bean id="comunicadoManager" class="us.es.comunica.webapp.rest.ComunicadoManagerImpl">
    <constructor-arg ref="comunicadoDao" />
</bean>
```

img/Cir

Después de estos pasos, al inicial la aplicación, se publican tanto la url de la wsdl de RPC como las urls que devuelven tanto los anuncios como las sugerencias mediante REST (como estructuras json).

En la figura 9.14 y 9.15, podemos observar la manera en la que se resuelve por parte del servidor las peticiones a estos servicios web.



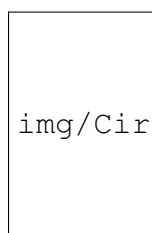
img/wsdComunica-eps-converted-to.pdf

img/CircuitoMarquesina-eps-converted-to.pdf

Figura 9.14: Wsdl de Comunic@.



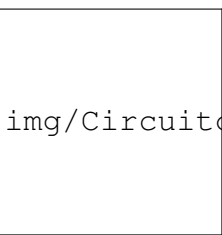
Figura 9.15: Petición REST de anuncios



En la demo de sprint se enseñó cómo se publicaban los datos, mostrando las urls de los servicios al dueño del producto. Al observarlo, el dueño del producto pidió, para poder comprobar el potencial de los servicios web generados y, de paso, tener un aplicativo que funcionara en su dispositivo móvil (con sistema operativo Android), la inclusión de una nueva historia de usuario que se especificó en la reunión de replanificación y que, debido al desconocimiento de la tecnología por parte de los desarrolladores y a la importancia que tenía para el cliente, obtuvo la máxima prioridad.

8350 - El usuario quiere poder ver el listado de anuncios y sugerencias en su móvil Android.

9.5.2.DISEÑO DE CLASES



img/CircuitoMarquesina-eps-converted-to.pdf

En este apartado se muestra como quedó el diseño emergente al realizar este módulo. Estas imágenes se pueden observar con más definición en la API entregada junto con el proyecto.

Se puede observar en las figuras la interacción en el patrón MVC y DAO, de tal manera que la división quedaría como se procede a explicar en este apartado.

En la funcionalidad de comunicación mediante servicios web tenemos la figura9.16 que muestra la interrelación entre las capas.

- 1.- La clase “ComunicadoManagerImpl” es la clase que genera los datos para su envío mediante servicios web. Implementa la interfaz “ComunicadoManager” que, a su vez, extiende de la interfaz “GenericManager” de Appfuse, y extiende de “GenericManagerImpl”, también de Appfuse.
- 2.- La interfaz “IComunicadoDao” forma parte del patrón DAO y abstrae a la lógica de negocios de obtener los datos, en este caso, de la base de datos. Esta clase hereda de “GenericDao” que pertenece a Appfuse y que permite la abstracción, en este caso, sobre Hibernate.

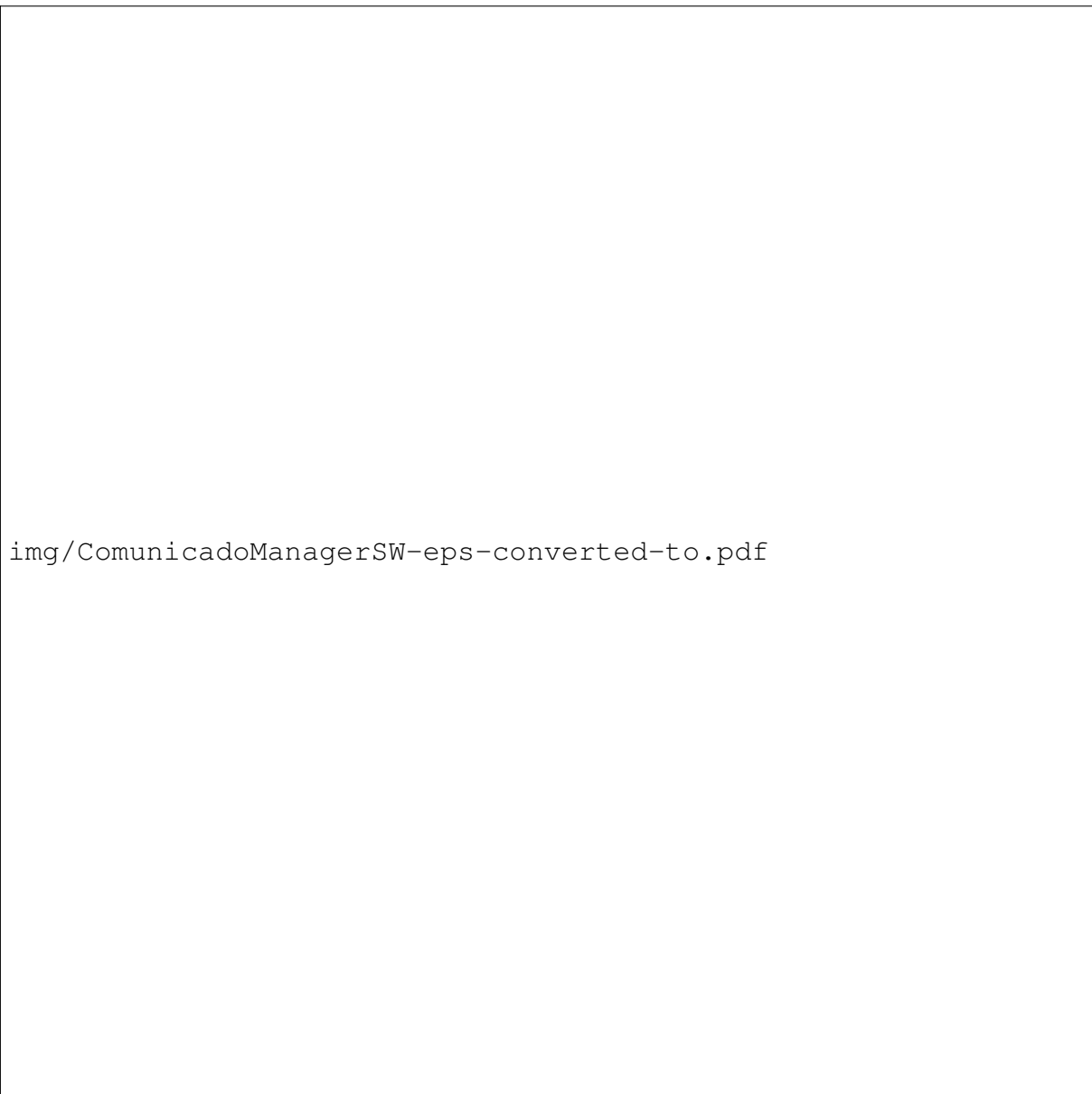


Figura 9.16: Relación entre clases de las operaciones de publicación por SW.

9.5.3.DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 8 puntos de historia.

Se puede observar la primera desviación significativa del proyecto, la media

por sprint debería ser entorno a los 10 puntos de historia realizados por cada sprint, en este caso se han realizado menos, aunque nuestra gráfica difiera de la ideal, en estos momentos la marcha del proyecto es buena puesto que de forma global se ha realizado mayor porcentaje de producto que el esperado.



Figura 9.17: Diagrama de Burndown. Sprint 4.

9.6. QUINTO MODULO: APLICACIÓN DE ANDROID

Durante la reunión de planificación anterior, surgió la necesidad por parte del cliente de ver el listado de comunicados en un dispositivo móvil, en concreto, en dispositivos que tuvieran Android como su sistema operativo.

El dueño del producto le dio a esta historia la máxima prioridad, debido a que era algo muy vistoso que enseñar al cliente y usuarios finales. El equipo estuvo en consonancia con esta decisión, ya que la dificultad de enfrentarse a

una tecnología desconocida hacía que fuera necesario abordarla cuanto antes y comprobar que, con los servicios web implementados en el sprint anterior, fuera suficiente para la comunicación con otros aplicativos.

Como la principal dificultad era el desconocimiento de la tecnología y toda nueva tecnología para el equipo requiere de un tiempo de aprendizaje, solo se comprometió para ese sprint la entrega de dicha historia de usuario.

Como lema utilizado para definir la meta de este sprint, se escogió “El usuario puede ver los comunicados publicados en su móvil Android.”

A la aplicación se le bautizó como “Comunic@ndroide”.

9.6.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

8350 - APLICACIÓN LECTORA DE COMUNICADOS ANDROID

img/Cir

Se realizó las siguiente división de tareas para conseguir cumplir los objetivos:

- 1.- Spike: Instalación de SDK para programación Android.
- 2.- Spike: Hola mundo para Android.
- 3.- Spike: Servicios web bajo Android.
- 4.- Implementación de cliente de servicio web
- 5.- pantalla de listado de anuncios y sugerencias.
- 6.- pantalla de inicio.

Durante parte del tiempo se investigó el entorno de desarrollo de Android. La web de desarrollo de Android ² es bastante completa, viene con todo lo necesario para empezar, incluyendo útiles tutoriales. Se necesita descargar el paquete Android SDK de la web (en este caso para Android 2.2, que es el que utiliza móviles como el Samsung Galaxy S, en donde se ha comprobado el correcto funcionamiento). Este paquete contiene todo lo necesario para empezar, incluyendo un emulador (que se puede observar en la figura 9.18) donde se puede ir probando el funcionamiento de la aplicación que se está desarrollando.



Figura 9.18: Emulador de Android.

Se comprobó que el lenguaje de programación que se utiliza es Java, la SDK tiene unas librerías específicas para este lenguaje. Se puede integrar gracias a un plugin³ con Eclipse, que se puede encontrar en la misma web, lo que facilita enormemente el trabajo al usar un IDE conocido. Se muestra en la figura 9.19 el entorno de desarrollo.

²<http://developer.android.com/sdk/index.html>

³ADT plugin



Figura 9.19: Eclipse con plugin de Android.

img/Circui

Después se estudió la manera en la que se gestiona e implementa la vista, muy intuitiva y sencilla ya que es un simple xml. También se aprendió el manejo de eventos tan importante en aplicaciones con estos dispositivos.

Una vez realizado la primera aplicación, el “hola mundo” de Android, se comenzó a investigar la manera de integrar servicios web a esta plataforma. Se encontraron varios artículos en donde otras personas ya habían utilizado REST y se tomaron como base a la implementación. Aunque ninguna resolvía plenamente el problema, cumplió con su objetivo, que no era otro que saber si se podía realizar y cómo se debía comenzar para alcanzarla meta.

Por lo tanto, se generó una clase “ClienteRest”, que es la encargada de obtener los comunicados mediante el protocolo REST.

Se muestra el código asociado a la petición GET del protocolo HTTP1.1:

```

...

/**
 * Genera peticiones HttpGet\WebInvoke a Rest.
 *
 * @param nombreMetodo
 *         el nombre del método.
 * @param parametros
 *         los parámetros.
 * @return el response de la llamada httpGet.
 * @throws Exception
 */
public String webGet(String nombreMetodo, Map<String, String> parametros)
    throws Exception {
    String getUrl = urlServidorRest + nombreMetodo;

    getUrl = generarURLConParametros(parametros, getUrl);

    try {
        httpGet = new HttpGet(getUrl);
        Log.e("WebGetURL: ", getUrl);

        response = clienteHttp.execute(httpGet);
        datos = EntityUtils.toString(response.getEntity());
    } catch (IOException e) {
        Log.e("WebGet IOException Error:", e.getMessage());
        throw e;
    } catch (Exception e) {
        Log.e("WebGet Error:", e.getMessage());
        throw e;
    }

    return datos;
}

...

```

La clase “LectorComunicados” tiene la lógica de negocio que hace posible la petición de comunicados. En un principio se realizó solo para sugerencias, luego, refactorizando código, se amplió para que pudiera leer cualquier tipo de comunicado con muy pocos cambios. Se muestra abajo el resultado.

```

...

/**
 * Obtiene los comunicados del servidor através de REST.
 *
 * @param tipo
 *         el tipo de comunicado. Debe ser una de las constantes de tipo de
 *         la clase
 * @return el listado de comunicados.
 * @throws Exception
 */
public List<Comunicado> getComunicados(String tipo)
    throws Exception {

    validarTipo(tipo);
    List<Comunicado> alrt = null;

    try {

```

```
ClienteRest webService = new ClienteRest(urlRestServer + "/" +
    tipo);

// Obtiene el response con forma JSON response del servidor.
String response = webService.webGet("", params);

// Se transforma la estructura JSON del RESPONSE a nuestros
    objetos
Type collectionType = new TypeToken<List<Comunicado>>() {
}.getType();
alrt = new Gson().fromJson(response, collectionType);
} catch (Exception e) {
    Log.d("Error: ", e.getMessage());
    throw e;
}
return alrt;
}
...

```

En la parte de la vista se generaron dos pantallas, una como pantalla de entrada en la que se puede elegir si se desea acceder al listado de sugerencias o al de anuncios, y otra que muestra dicho listado. En la figura 9.20 se puede observar la pantalla de inicio de Comunic@ndroide.



img/Cir

Figura 9.20: Pantalla inicial de la aplicación.

Resulta interesante la facilidad con la que se genera lo que se denomina “layout”, que no es otra cosa que la forma de disponer los elementos en la pantalla para una vista determinada. Se describe cómo se ha realizado la pantalla de entrada al aplicativo como ejemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LayoutComunicados"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical" android:weightSum="1"
    android:background="#000033">
    <TextView android:text="Comunicados" android:textAppearance="?android:attr/
        textAppearanceLarge"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:id="@+id/titulo" android:textColor="#ff9900"></TextView>
    <ListView android:id="@+id/vistaListaComunicados"
        android:layout_width="wrap_content" android:layout_height="285dp"
        android:scrollbarAlwaysDrawVerticalTrack="true"></ListView>
    <LinearLayout android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:orientation="horizontal"
        android:weightSum="1">
    <ImageView android:id="@+id/imageView1"
        android:layout_height="wrap_content" android:src="@drawable/
            logocomunicadroid3"
        android:layout_width="wrap_content" android:layout_gravity="right" />
    </LinearLayout>
</LinearLayout>
```

img/CircuitoMarquesina-eps-converted-to.pdf

Se puede observar que dentro de una etiqueta principal que define la forma del “layout” denominada en este caso “LinearLayout” se siguen los elementos que lo componen, es decir, un texto con el título, un listado de elementos donde se verán los comunicados, y una imagen con el logotipo de Comunic@ndroide. En la figura 9.21 se puede observar el resultado.



Figura 9.21: Listado de Anuncios en Comunic@ndroide.

También es interesante remarcar la manera de manejar eventos. La clase “ComunicaClientActivity” es el punto de entrada al sistema. Extiende obligatoriamente de “Activity”, que nos la proporciona el SDK de Android, y debe implementar el método “onCreate”.

A continuación se mostrara un ejemplo de cómo se utilizan los eventos en el proyecto, en este caso, cómo se genera el evento de listado de anuncios al pulsar el botón apropiado que se dibujó en la vista de la pantalla inicial.

```
...  
  
/**  
 * Se generan los eventos básicos de la pantalla inicial.  
 */  
private void generateBasicsEvents() {  
    setContentView(R.layout.main);  
    // Captura los botones de layout  
    Button buttonAnuncios = (Button) findViewById(R.id.botonAnuncios);  
    Button buttonSugerencias = (Button) findViewById(R.id.botonSugerencias);  
    // Registra el listener de onClick listener  
    buttonAnuncios.setOnClickListener(mAnuncioListener);  
}
```

```

        buttonSugerencias.setOnClickListener(mSugerenciasListener);

        vistaActual = Vista.INICIO;
    }

    /** El listener del botón de anuncio de la pantalla inicial. */
    private OnClickListener mAnuncioListener = new OnClickListener() {
        public void onClick(View v) {
            try {
                loadData(v, LectorComunicados.Anuncios);
            } catch (Exception e) {
                printErrorView(v);
            }
        }
    };

    ...

    /**
     * Se encarga de cargar los datos de comunicados según el tipo e
     * imprimirlos en la vista de la lista.
     *
     * @param vista
     *         la vista desde la que se llama.
     * @param tipo
     *         el tipo de comunicado.
     * @throws Exception
     */
    private void loadData(View vista, String tipo) throws Exception {
        try {
            setContentView(R.layout.comunicados);
            TextView title = (TextView) findViewById(R.id.titulo);
            title.setText(tipo.toUpperCase());

            vistaDeLista = (ListView) findViewById(R.id.vistaListaComunicados);

            List<Comunicado> comunicados = reader.getComunicados(tipo);

            Comunicado[] comuArray = new Comunicado[comunicados.size()];
            comunicados.toArray(comuArray);
            AdaptadorComunicados adaptador = new AdaptadorComunicados(
                vista.getContext(), comuArray);

            vistaDeLista.setAdapter(adaptador);
            vistaActual = Vista.COMUNICADO;
        } catch (Exception e) {
            Log.e("Unexpected Error:", e.getMessage());
            throw e;
        }
    }

    ...

```

img/Circui

Una vez implementado el aplicativo, se probó durante la demo con el usuario instalándolo en su dispositivo móvil, logrando el objetivo deseado. Aunque es una aplicación simple, se demostró que la comunicación mediante servicios web funcionaba de manera correcta y dejó el campo abierto a nuevas y futuras implementaciones, como generar nuevos comunicados desde Comunic@ndroide. Estas ampliaciones de funcionalidad se consideraron fuera del ámbito del proyecto final, ya que había otras historias de usuario que hacer de más prioridad.

Se concluyó, junto con el dueño del producto, que era más importante avanzar en las historias que quedaban pendientes que continuar dotando de funcionalidad a Comunic@ndroide.

Para finalizar, en la figura 9.22, se muestra la aplicación funcionando en un Samsung Galaxy mini.

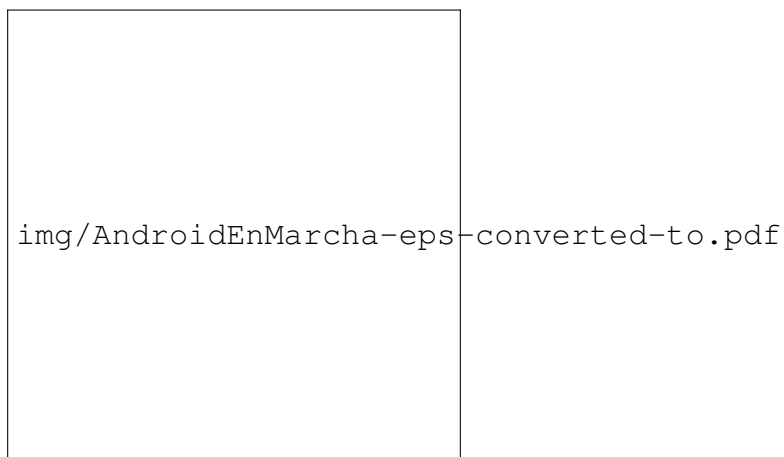
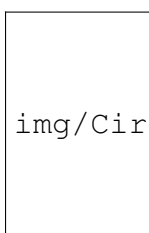


Figura 9.22: Comunic@ndroide en funcionamiento.



9.6.2.DISEÑO DE CLASES

En este apartado se muestra como quedó el diseño emergente al realizar este módulo. Estas imágenes se pueden observar con más definición en la API entregada junto con el proyecto.

En este caso también tenemos un patrón MVC, aplicado por el entorno de desarrollo Android, como podemos ver en la figura 9.23.

- 1.- La clase “ComunicaClientActivity” es la clase de la que hereda de “Activity” y proporciona el punto de acceso de la vista mediante eventos sobre la interfaz del usuario.
- 2.- La clase “LectorComunicados” es la clase que contiene la lógica de negocio principal para la obtención de los datos.

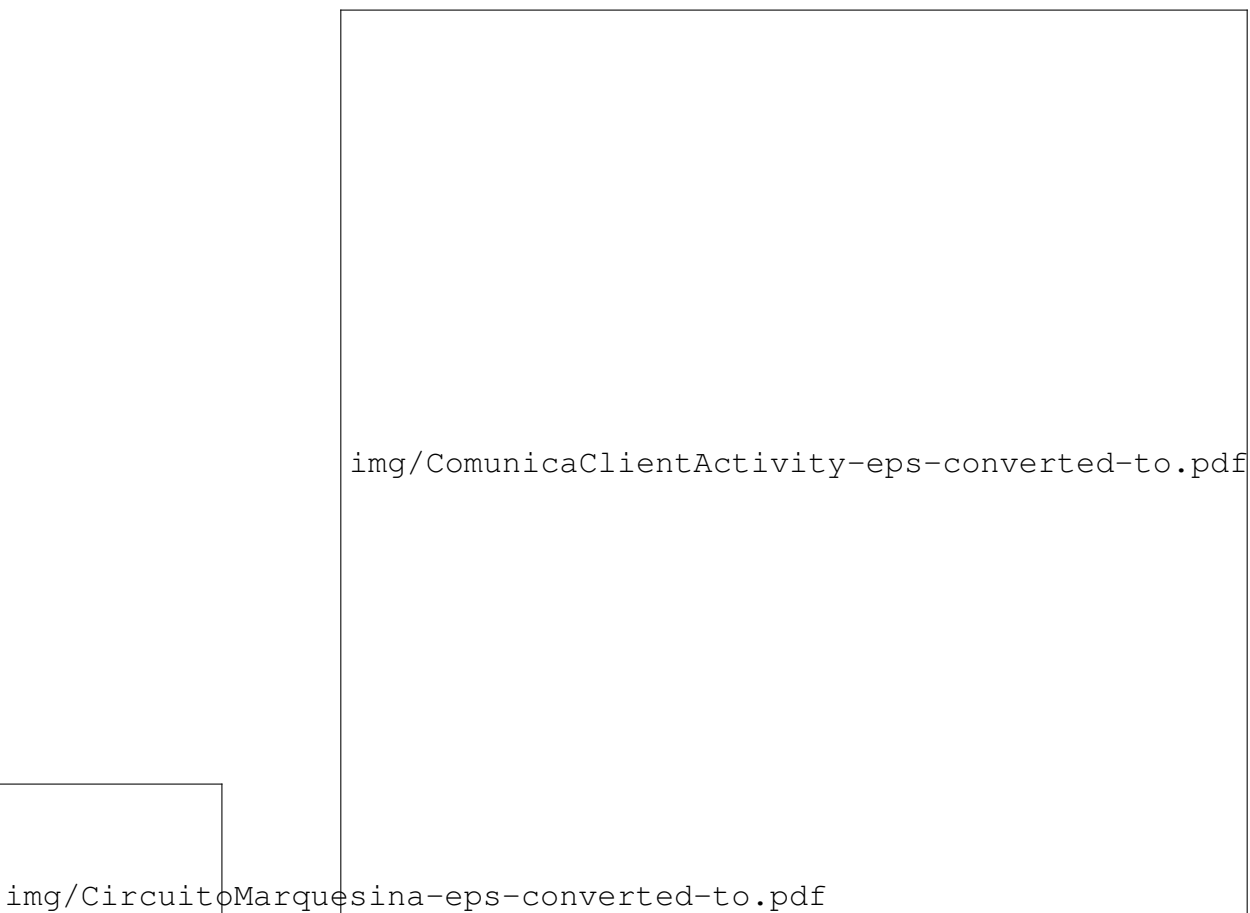


Figura 9.23: Relación entre clases de las operaciones lectura de comunicados para Android.

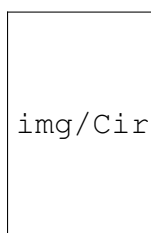
9.6.3. DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 8 puntos de historia.

En este sprint y en anteriores se ha aumentado el número de puntos de historia de usuario, por tanto se tiene un mayor número de puntos por implementar. En consecuencia si solo se han realizado 8 puntos historias, debido a la dificultad del desarrollo de una aplicación en un entorno desconocido, esto puede provocar un desvío en la entrega del proyecto.



Figura 9.24: Diagrama de Burndown. Sprint 5.



9.7. SEXTO MÓDULO: NOTIFICACIÓN, PAGINACIÓN Y BÚSQUEDA DE COMUNICADOS

En este módulo, la aplicación tiene la mayoría de la funcionalidad hecha. En este sprint se van a realizar las tareas que se han quedado bloqueadas hasta la implementación del resto de las historias de usuario. Además se van a hacer tareas de mejora como la búsqueda de comunicados.

Cuando se mostró la aplicación al cliente en la demo del primer sprint, este sugirió que se paginaran los resultados, añadiendo la tarea, Paginación. Esta parte de la implementación corresponde al sprint 7.

También se realizó una replanificación de tareas como se comenta en el apartado dos, en donde las tareas 9000, 8750 y 8500 bajaron de prioridad (y siguieron bajando durante el transcurso del resto de sprints). Como aún se dispone de tiempo, es durante este sprint cuando se van a llevar a cabo.

Como lema para determinar la meta de este sprint se eligió “Entorno más amistoso para el acceso a la información.”

9.7.1.HISTORIAS REALIZADAS EN ESTE MÓDULO

HISTORIA 9000 - BORRAR COMUNICADO POR EMAIL

Para realizar esta petición se escribe un test llamado “BEnvioCorreo”, para el cual se genera una clase llamada Correo y EnvioCorreo.

El envío del correo utiliza una cuenta específica creada para el sistema. Esta cuenta es de Gmail y desde ella se enviarán los email a los usuarios cuando sea necesario.

Para llevar a cabo la historia 9000 e implementar el resto de las historias de usuario que hacen envíos de mails, se utilizarán las mismas clases.

Durante la implementación de las anteriores historias de usuario, no se ha necesitado acceder a los datos del usuario que genera los comunicados. Pero para esta historia se necesita saber qué usuario ha generado determinado comunicado para permitirle enviar la petición de borrado.

También es necesario saber, por parte del administrador, quién a generado qué comunicado para poder enviarle por email el cambio de estado.

Se modificó la aplicación para asociar los usuarios a los comunicados, por ejemplo, en ComunicadoDaoHibernate se tuvo que añadir la relación:

```
/**\brief La id del usuario asociado al comunicado. */
@ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE}, targetEntity=
    User.class )
```

```
@JoinColumn(name="id_usuario")  
private User usuario;
```

Una vez realizada esta historia de usuario, para realizar el resto de peticiones simplemente se tenía que añadir el mismo código con mensajes de email distintos.

Por tanto al realizar las historia 9000 también se implementaron las historias:

8750 - El administrador quiere que le llegue por mail los nuevos comunicados

8500 - El usuario quiere que le lleguen por mail el cambio del estado de su comunicado.

El dueño de la aplicación al mostrarle que se podía hacer una petición de borrado de sugerencias, indicó que veía adecuado realizar la misma operación con los anuncios. En la historia de usuario 2700, se indicó que el usuario podía borrar sus anuncios, pero para una mejor administración prefirió que todo se hiciera por peticiones y que fuera el administrador el único que pudiera borrar del sistema tanto anuncios como sugerencias.



HISTORIA 7000 - BÚSQUEDA DE COMUNICADOS

La historia de usuario tiene la siguiente tarea:

7000 - El usuario puede buscar en el título, en la descripción y en un intervalo de fechas.

Se genera una nueva clase "BusquedaComunicado" para implementar la historia de usuario, que realizará búsquedas por título, fecha desde - hasta y descripción.

Se añade el buscador a la pantalla de listado de comunicados. Para facilitar la inserción de fechas se añade un calendario en javascript. Ver figura 9.25.

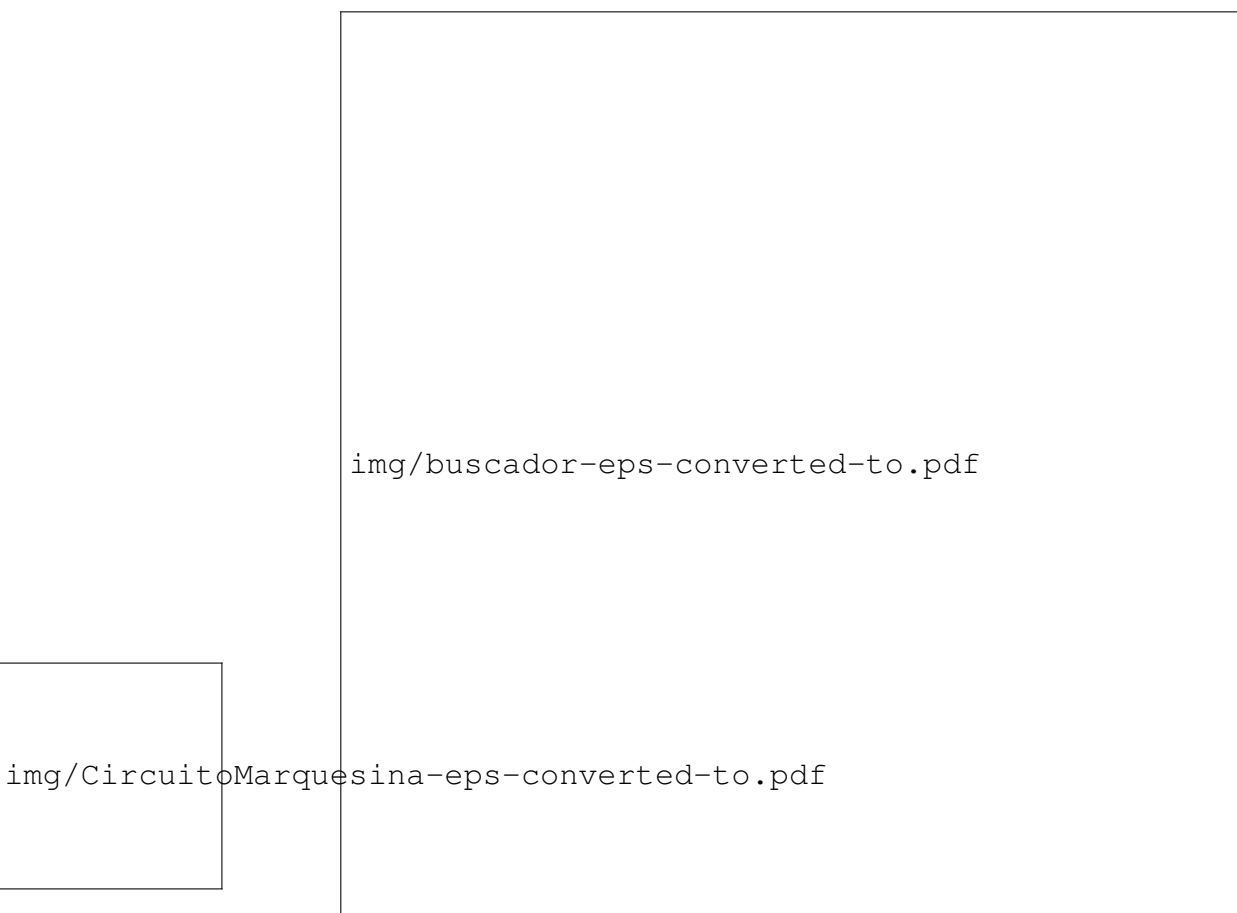


Figura 9.25: Búsqueda de comunicados

HISTORIA 6500 - PAGINAR COMUNICADOS

Para mejorar la visualización de los comunicados, el dueño indicó que quería poder paginar los resultados, ya que al aumentar el número de estos se volvía totalmente tediosa la navegación. Para ello se añade dicha funcionalidad en el listado de comunicados. Se realiza un paginador simple donde se calculan los comunicados que se van a ver pasando los parámetros de página actual y número de comunicados por página. Se permite al usuario cambiar el número de comunicados que puede ver en cada página gracias a un combo desplegable situado al lado de la paginación.

Con la realización de esta tarea se finaliza la meta de sprint y con esto, se

genera la primera versión terminada del aplicativo.

El cliente queda satisfecho en la demo como para integrarlo en su organización, ya solo quedaría un periodo de pruebas en donde podrían surgir algunos cambios menores.

9.7.2.DISEÑO DE CLASES

En este apartado se muestra como quedó el diseño emergente al realizar este módulo. Estas imágenes se pueden observar con más definición en la API entregada junto con el proyecto.

Se puede observar en las figuras la interacción en el patrón MVC y DAO, de tal manera que la división quedaría como se procede a explicar en este apartado.

En la funcionalidad de búsqueda de un comunicado, que es la que vamos a describir aquí, tenemos la figura 9.26 que muestra la interrelación.

img/Cir

- 1.- La clase “BaseAction” es la clase de la que heredan los demás Actions de Struts y que se comunica directamente con el controlador que se gestiona mediante un fichero de configuración del framework Struts denominado “struts.xml”.
- 2.- La clase “BusquedaComunicadoAction” tiene implementada la lógica de negocio de este módulo, es decir, la búsqueda de un comunicado. Para ello se apoya en “IComunicadoManager”, que define la interfaz del manejo de comunicados y de la cual se hace uso dentro de este Action.
- 3.- La interfaz “IComunicadoDao” forma parte del patrón DAO y abstrae a la lógica de negocios de obtener los datos, en este caso, de la base de datos.
- 4.- La clase “ComunicadoBusquedaBean” es el nexo de unión con la vista, permitiendo obtener de ella un objeto con los datos de entrada de la búsqueda rellenos por el usuario.



Figura 9.26: Relación entre clases de las operaciones de búsqueda de comunicados.

9.7.3. DIAGRAMA DE BURNDOWN

En esta sección se puede observar el diagrama de Burndown, en este sprint se puede comprobar la velocidad y número de puntos de historias realizados. En esta ocasión se han realizado 14 puntos de historia.

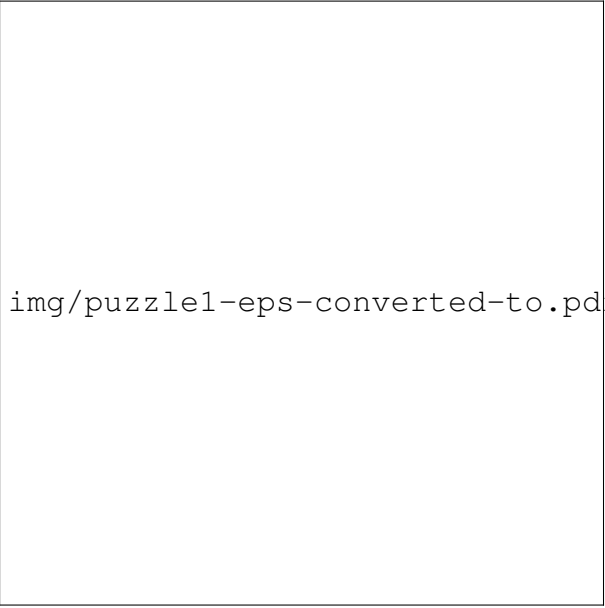
Como se puede observar en la imagen 9.27, la velocidad ha sido entre los 8 puntos de historia a 14 realizados en cada sprint. Como comentario general, cabe resaltar que la línea se asemeja a la línea ideal, que sería una línea recta que uniese los puntos máximo y mínimo de historias de usuario. La desviación que se ha obtenido, esta por encima de la recta ideal. En este caso, indica que se ha llevado un ritmo más lento de lo esperado. Como se ve reflejado en el sprint final, se tuvieron que realizar más puntos de historia para poder finalizarlos todos. Esto ha sido posible puesto que los puntos de historia a realizar tenían baja dificultad para los desarrolladores. Para la reunión de retrospectiva se debería pensar en como mejorar esta gráfica para parecerse cada vez más al diagrama ideal.



Figura 9.27: Diagrama de Burndown. Sprint 6.

9.8. PRUEBAS DE INTEGRACIÓN

Después de las pruebas unitarias es conveniente realizar pruebas de integración. Estas tratan de unir la piezas y ver que todo funciona bien estando



img/puzzle1-eps-converted-to.pdf

unido. En este caso, al estar usando el arquetipo de Maven Appfuse, ya viene con sus propias y completas pruebas que corrobora que el esqueleto de nuestra aplicación funciona correctamente antes de dejar que se inicie, es decir, que el aplicativo conecta a la base de datos, que está bien configurada, que los usuarios pueden acceder, etc. . . si por algún caso alguna de estas pruebas preliminares fallara, ni siquiera se iniciaría el servicio.

A parte, es interesante añadir las propias, en especial para comprobar que la unión de la lógica de negocios con el acceso a los datos de manera real se hace correctamente. Las pruebas de integración son, necesariamente, menores que las unitarias, primero por que la mayor parte de las cosas ya las prueban las unitarias, y, segundo, por que son mucho más lentas en su ejecución.

Como ejemplo de las pruebas de integración propias tenemos la clase de “BComunicadoDAO”, en donde se pueden observar cómo se prueba que las operaciones de comunicados sobre la base de datos funcionan como deberían.

```
@Test
public void testGetComunicados() {
    //Para que pase el test, debe haberse ejecutado el tomcat.
    List<ComunicadoPojo> listadoComunicados = comunicadoDao.getAll();
    assertTrue(listadoComunicados.size() > 0);
    List<ComunicadoPojo> listadoComunicadosRechazados = comunicadoDao.getByEstado(
        Estado.rechazado);
    assertTrue(listadoComunicadosRechazados.size() == 2);
    List<ComunicadoPojo> listadoAnuncios = comunicadoDao.getByTipo(Tipo.anuncio);
    assertTrue(listadoAnuncios.size() == 2);
    List<ComunicadoPojo> listadoAnunciosPublicados = comunicadoDao.getByTipoYEstado(
        Tipo.anuncio, Estado.publicado);
    assertTrue(listadoAnunciosPublicados.size() == 1);
}

@Test
```



```
public void testSaveComunicado() {
    ComunicadoPojo expected = new ComunicadoPojo();
    expected.setCodigo("Bas-01");
    expected.setDescripcion("Generado desde los test");
    expected.setEstado(Estado.publicado);
    expected.setFechaInserccion(new Date());
    expected.setTitulo("Básico");
    expected.setTipo(Tipo.anuncio);

    expected = comunicadoDao.save(expected);
    ComunicadoPojo actual = comunicadoDao.getById(expected.getId());

    assertTrue(expected.equals(actual));

    comunicadoDao.remove(expected.getId());
}

@Test
public void testEditComunicado() {
    ComunicadoPojo comunicado = new ComunicadoPojo();
    comunicado.setCodigo("Bas-01");
    comunicado.setDescripcion("Generado desde los test");
    comunicado.setEstado(Estado.publicado);
    comunicado.setFechaInserccion(new Date());
    comunicado.setTitulo("Básico");
    comunicado.setTipo(Tipo.anuncio);

    comunicado = comunicadoDao.save(comunicado);
    ComunicadoPojo actual = comunicadoDao.getById(comunicado.getId());
    actual.setTitulo("Modificado");
    actual = comunicadoDao.save(actual);

    ComunicadoPojo expected = comunicadoDao.getById(actual.getId());
    Assert.assertEquals("Modificado", expected.getTitulo());
    Assert.assertEquals("Modificado", comunicado.getTitulo());

    comunicadoDao.remove(comunicado.getId());
}

@Test
public void testBorrarComunicado() {
    ComunicadoPojo comunicado = new ComunicadoPojo();
    comunicado.setCodigo("Bas-01");
    comunicado.setDescripcion("Generado desde los test");
    comunicado.setEstado(Estado.publicado);
    comunicado.setFechaInserccion(new Date());
    comunicado.setTitulo("Anuncio para borrar");
    comunicado.setTipo(Tipo.anuncio);

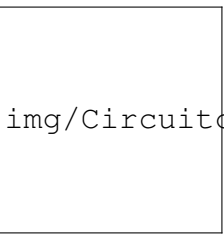
    ComunicadoPojo comunicado2 = new ComunicadoPojo();
    comunicado2.setCodigo("Bas-02");
    comunicado2.setDescripcion("Generado desde los test");
    comunicado2.setEstado(Estado.publicado);
    comunicado2.setFechaInserccion(new Date());
    comunicado2.setTitulo("Anuncio que no se borra");
    comunicado2.setTipo(Tipo.anuncio);

    comunicado = comunicadoDao.save(comunicado);
    comunicado2 = comunicadoDao.save(comunicado2);
    assertTrue(comunicadoDao.delete(comunicado.getId()));
    ComunicadoPojo comunicadoActual = comunicadoDao.get(comunicado.getId());
    ComunicadoPojo comunicadoActual2 = comunicadoDao.get(comunicado2.getId());
    assertTrue(comunicadoActual.isBorrado());
    assertTrue(!comunicadoActual2.isBorrado());
    comunicadoDao.remove(comunicado.getId());
    comunicadoDao.remove(comunicado2.getId());
}
```

img/Cir

Obviamente, para pasar estos test, la aplicación debe estar en ejecución, pues así aseguraremos que se ha iniciado la base de datos y cargado los datos de prueba.

Dichos datos de prueba se definen en el fichero “sample-data.xml” dentro de la carpeta “src/test/resources” y se cargan al iniciar el aplicativo en modo prueba mediante Maven.



img/CircuitoMarquesina-eps-converted-to.pdf

Parte IV

Conclusiones

Capítulo 10. Retrospectiva final de proyecto

Al finalizar el proyecto se hace la última retrospectiva. Esta es especial, ya que se hace al final del proyecto como revisión de qué cosas han ido bien y cuáles se pueden mejorar viendo todo el proceso en conjunto.

Al ser la retrospectiva final de un proyecto fin de carrera, no solo se habla del resultado del aplicativo y su funcionamiento, si no que también se centra en la documentación entregada. Por lo tanto, este capítulo se ha hecho a posteriori de la entrega del borrador de la documentación que nuestro tutor ha revisado antes de dicha retrospectiva.

Para ello, se ha hecho una pequeña actividad, en donde se han dividido dos columnas en un documento de texto que se han rellenado por los integrantes del equipo, incluidos el tutor (que ha desempeñado el papel de dueño de producto durante el desarrollo). Dichas columnas se dividían en “a mejorar” y en “a mantener”, y el resultado fue el siguiente:

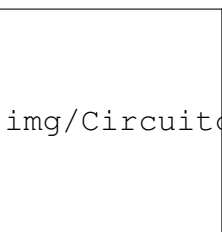
Una vez escritas las aportaciones se discutió sobre ellas, y se llegó a las siguientes conclusiones.

- 1.- El uso de metodologías ágiles se adapta muy bien al desarrollo de un proyecto fin de carrera. En este caso particular, ha ayudado a los desarrolladores que, por motivos laborales, no han podido llevar el desarrollo de una manera continuada todos los días. A conseguido que se centraran en lo importante y no perder el hilo de lo que se iba haciendo debido a los obligatorios parones ocurridos durante el desarrollo.
- 2.- Las metodologías ágiles aumentan el compromiso de los desarrolladores al ser ellos los que se comprometen a tener terminadas las tareas para cada





img/AMejorarAMantener-eps-converted-to.pdf



sprint.

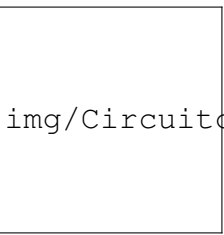
img/CircuitoMarquesina-eps-converted-to.pdf

- 3.- El equipo ha demostrado gran adaptabilidad a los cambios y motivación por el uso de nuevas tecnologías desconocidas para ellos.
- 4.- El equipo ha aprendido por su cuenta ingeniería del software que, al ser de la rama de sistemas, no habían cursado durante sus años de carrera, llenando esa falta de conocimiento.
- 5.- El proyecto ha estado muy orientado a la vida real y empresarial.
- 6.- El dueño de producto y tutor ha quedado muy convencido de la utilidad de las metodologías ágiles.
- 7.- El proyecto ha conseguido cumplir con sus objetivos, incluso aumentando el alcance de este.
- 8.- El uso específico de la técnica de programación por parejas es viable solo si se trabaja en el mismo lugar. Es especialmente útil para traspaso de

conocimientos y para resolver problemas complejos.

Con todo esto, el resultado ha sido satisfactorio y se considera que el trabajo realizado cumple con los objetivos establecidos.

img/Cir



img/CircuitoMarquesina-eps-converted-to.pdf

Capítulo 11. Objetivos cumplidos

Con las conclusiones obtenidas del capítulo anterior, se pasa ahora a explicar los objetivos cumplidos durante el desarrollo del proyecto, comparándolos con los que se deseaba abarcar al inicio de este y que vienen explicados en el prefacio.

11.1. OBJETIVOS ORIENTADOS A LA METODOLOGÍA

img/Cir

A pesar de las dificultades añadidas debido a las circunstancias personales de los implicados, ya que se encuentran trabajando durante el desarrollo del proyecto, se puede concluir que, a pesar de no haber podido usar todos los artefactos con los que cuenta Scrum, se ha adaptado favorablemente este marco de trabajo, consiguiendo cumplir con su objetivo.

Los requisitos cambiantes han estado a la orden del día, y no han supuesto mayor complicación a la hora de llevarlos a cabo gracias al BDD, y se ha conseguido no perder nunca de vista el objetivo principal de los proyectos: tener un aplicativo que cubra las necesidades del cliente. Aunque este caso es algo especial al tratarse de un cliente ficticio, a efectos prácticos se ha tenido como dueño del producto al tutor. El resultado final ha sido el de adaptarse a los requisitos que son necesarios para hacer un proyecto fin de carrera de calidad.

El tutor (dueño del producto), ha quedado satisfecho con lo entregado, por lo que concluimos en que se ha conseguido cumplir con las expectativas comprometidas.

11.2. OBJETIVOS ORIENTADOS A LA TÉCNICA

El hecho de entregar al final de cada iteración un aplicativo que funciona y que cumple con parte de los requisitos hace que la aplicación de las metodologías ágiles haya sido un éxito y que se haya adaptado perfectamente a la situación de los desarrolladores y a las demandas del dueño del producto.

Además, la implicación de los desarrolladores por aprender nuevas técnicas y su uso durante el proyecto cumple con el objetivo de estar siempre a la vanguardia de las nuevas tecnologías, como lo muestra la inclusión del módulo de Comunic@ndroide para Android.

Los desarrolladores rara vez se han dedicado a una parte específica del proyecto, por lo que ambos conocen bien todo el desarrollo y no solo una parte aislada. Gracias a la programación por parejas, el traspaso de conocimientos se ha hecho de una manera natural y sin sentir que se está perdiendo el tiempo. Esto cumple con el objetivo de ver al conjunto de desarrolladores como un equipo y que el código no tenga un solo propietario, sino que es de todos.

Por lo tanto, también se considera que se ha conseguido cumplir con los objetivos orientados a la técnica.

11.3. OBJETIVOS PERSONALES

Durante el desarrollo, se han generado diversas situaciones que han requerido de un esfuerzo por parte de los desarrolladores para cambiar su manera de hacer las cosas. Empezar con las metodologías ágiles no es fácil, pues tener que olvidar ciertos hábitos aprendidos y profundamente arraigados es muy duro, sobre todo cuando ya se ha trabajado durante un tiempo de una determinada manera.

Es duro iniciar la arquitectura de una aplicación de este tipo olvidándose de hacer primero el modelo relacional de las tablas. Es muy duro empezar a realizar el test antes que el código que lo cumpla, y cuando uno se quiere dar cuenta está escribiendo más funcionalidad de la que debiera.

El cambio de concepción sobre el desarrollo de software es radical, no admite medias tintas; ya no hay excusa para seguir haciendo las cosas como antes. A pesar de todo, cuando el interruptor cerebral hace “click” y se empieza a vislumbrar el proceso, ya no hay marcha atrás: el cambio ha comenzado y los viejos y manidos hábitos ya no tienen cabida. Es increíble la sensación de desesperación e inseguridad que produce tener una funcionalidad que no está cubierta con sus test, hasta el punto de dejar todo lo demás y otorgarle la máxima prioridad. Más increíble es el sentimiento de seguridad y paz que da el tener el código cubierto por test. Ya no hay excusas para hacer una refactorización de código o añadir nuevas funcionalidades.

La experiencia es dura y difícil pero, sobre todo, muy grata. Lo único que falta para conseguir que los objetivos personales sean un éxito es hacer que los lectores de este documento lo lean con la mente abierta y les empiece a “picar el gusanillo” por aplicar metodologías ágiles en su día a día.

El equipo espera realmente que los lectores hayan disfrutado con la lectura del presente proyecto y que le den una oportunidad a este enfoque del diseño de software como nosotras hemos hecho.



img/Cir

Parte V

Apéndices

Apéndice A. Manual de usuario

“Una antigua historia decía que cierta persona quería que su ordenador fuese tan fácil de utilizar como su teléfono. Estos deseos se han hecho realidad, ya no sé cómo usar mi teléfono. - Bjarne Stroustrup”

EN este capítulo se detallará el funcionamiento de la aplicación para facilitar su uso. Esta guía se dividirá por roles de usuario, Administrador o Usuario.

Se comenzará con la creación de un usuario, posteriormente se indicarán las funciones a las que puede acceder. A continuación se explicarán las funciones del administrador.

La intención de esta guía es realizar un ciclo entero de la aplicación paso a paso para facilitar su uso.

A.1. PERFIL USUARIO

A.1.1. ALTA DE USUARIOS

La pantalla de inicio de la aplicación, mostrada en la figura A.1, dispone de una opción para dar de alta un usuario.



Figura A.1: Pantalla de inicio

En la siguiente pantalla, mostrada en la figura A.2, se escriben los datos del usuario. Se tienen que rellenar obligatoriamente los campos señalados con un asterisco. A continuación, detallamos el significado de cada elemento del formulario:

- 1.- **Usuario:** nombre del usuario para la aplicación, este nombre se utilizará para entrar en la aplicación desde la pantalla de inicio.
- 2.- **Contraseña:** se escribe una palabra para verificar la identidad del usuario.
- 3.- **Confirme contraseña:** se vuelve a insertar la palabra escrita en el apartado “contraseña”, para verificar que se ha escrito correctamente
- 4.- **Olvido contraseña:** una frase que le sea significativa para que, en caso de

olvido de contraseña, pueda ayudarle a recordar su contraseña.

- 5.- **Nombre:** aquí debe escribir su nombre, para poder identificarlo en el sistema
- 6.- **Apellidos:** aquí debe escribir sus apellidos, para poder identificarlo en el sistema
- 7.- **E-Mail:** aquí debe escribir la dirección de correo en la cual quiere que la aplicación se comunique con usted.
- 8.- **Número de teléfono:** este dato no es obligatorio, si lo desea puede facilitar a la aplicación un número de contacto.
- 9.- **Web:** este dato no es obligatorio, puede facilitar una web de contacto.
- 10.- **Dirección:** este dato no es obligatorio, puede facilitar una dirección de contacto.
- 11.- **Ciudad:** nombre de la ciudad en la que reside habitualmente
- 12.- **Provincia:** provincia de la ciudad en la que reside habitualmente
- 13.- **Código Postal:** número identificativo de la zona de su ciudad donde reside habitualmente.
- 14.- **País:** país al que pertenece su ciudad de residencia habitual.

Una vez rellenados los datos pulsamos el botón de registrar.

Aparecerá un mensaje indicando que se ha efectuado el alta correctamente, y desde ese mismo instante se podrá acceder a la aplicación

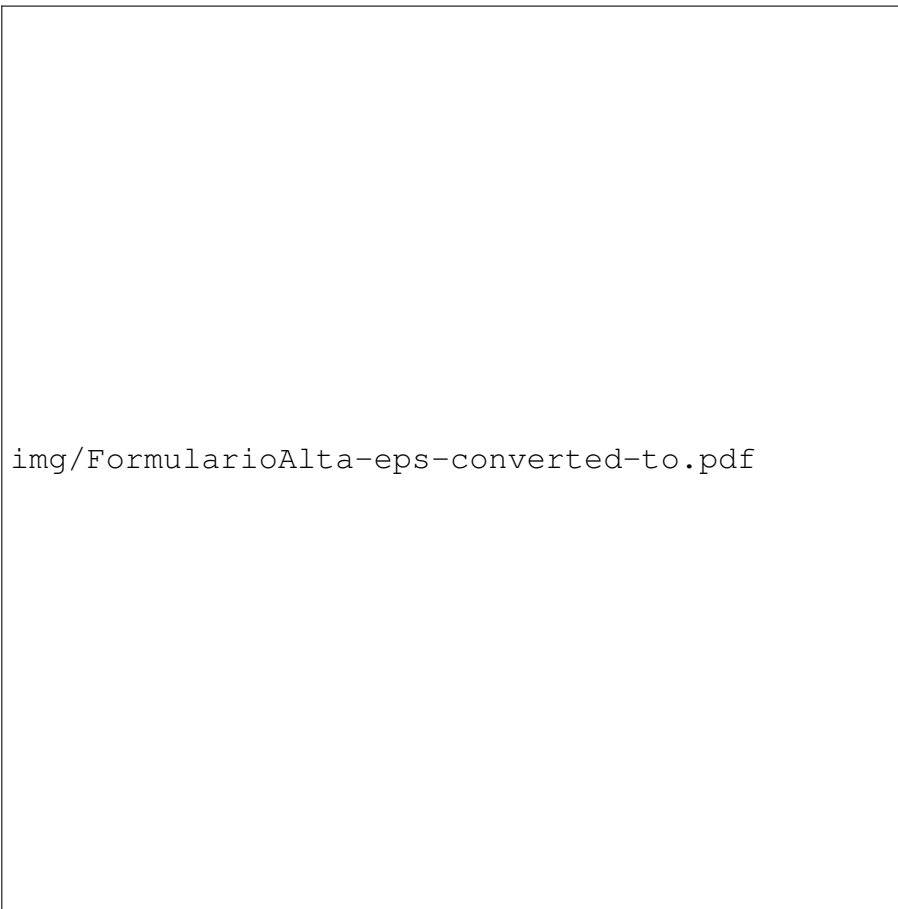


Figura A.2: Alta de Usuario

A.1.2.MENÚ DE INICIO

Una vez dado de alta el usuario, se puede acceder a la aplicación poniendo el nombre de usuario, que se ha indicado en el formulario de alta y su contraseña. Se puede observar en la figura A.1, que tenemos un botón con el nombre “Entrar”. Si se pulsa, se puede acceder a la pantalla de menú de inicio.



Figura A.3: Menú de inicio, usuario

En la pantalla de menú de inicio, mostrada en la figura A.3, existen una serie de enlaces, estos se han enumerado para poder describir su funcionalidad en los siguientes apartados.

A.1.3. EDITAR PERFIL

Para acceder a esta opción debemos pulsar en el enlace número uno del menú de inicio, ver figura A.3. Se accede al formulario de alta pero, en este caso, con los datos actuales del usuario. Se pueden modificar los datos cuando sea necesario. Para que las modificaciones sean permanentes debemos pulsar el botón de Guardar.

A.1.4. SUBIR UN FICHERO

Esta opción, la número dos de la pantalla de inicio, ver figura A.3. permite la subida de un fichero. Pulsando en este enlace nos aparece una pantalla, ver figura A.4, en la cual, primero se tiene que escribir un nombre identificativo al fichero. A continuación se pulsa el botón de examinar para buscar el archivo a subir. Finalmente, se pulsa el botón Subir. La siguiente pantalla indicará, mediante un mensaje, si se ha subido correctamente el archivo.



Figura A.4: Subir fichero, usuario

A.1.5. COMUNICADOS: ANUNCIOS Y SUGERENCIAS

Pulsando sobre los enlaces número tres y cuatro de la pantalla de inicio, ver figura A.3, se accede a la pantalla de gestión de comunicados, ver figuras A.5 y A.6. Se procede a dividir esta pantalla en tres partes para facilitar la explicación:

Búsqueda, visualización y añadir un nuevo comunicado.

Se realizará la explicación de las pantallas de anuncio solamente, las pantallas de sugerencias tienen la misma funcionalidad que los anuncios, pero con la diferencia del tipo de comunicado.

Por tanto a partir de este momento, se usará el termino comunicado que engloba tanto sugerencias como anuncios.

BÚSQUEDA DE COMUNICADOS



Figura A.5: Búsqueda Comunicados, usuario

Como se observa en la figura A.5, se puede buscar un comunicado utilizando los siguientes criterios:

- ★ **Fecha desde:** se mostrarán todos los comunicados cuya fecha de creación sea superior que la fecha indicada por el usuario. Para facilitar la inserción de fechas, puede pulsar el icono de calendario, al pulsarlo aparecerá el calendario que se muestra en la imagen A.5.
- ★ **Fecha hasta:** se mostrarán todos los comunicados cuya fecha de creación sea inferior que la fecha indicada por el usuario. Para facilitar la inserción de fechas, puede pulsar el icono de calendario, al pulsarlo aparecerá el calendario que se muestra en la imagen A.5.
- ★ **Título:** se mostrarán todos los comunicados cuyo título contenga la palabra o palabras escritas por el usuario.
- ★ **Descripción:** se mostrarán todos los comunicados cuya descripción contengan la palabra o palabras escritas por el usuario.

Para realizar la búsqueda se tiene que pulsar sobre el botón buscar.

VISUALIZACIÓN DEL LISTADO DE COMUNICADOS

Al pulsar sobre los enlaces tres y cuatro, del menú de inicio, ver figura A.3, se accede a un listado de comunicados. En este caso, solo podrán ser vistos los comunicados en estado publicado.



Figura A.6: Listado Comunicados, usuario

Como se muestra en la figura A.6, se enumera el contenido de un comunicado:

- 1.- **Título de un comunicado:** si se pulsa en el título de un comunicado, se accederá a su detalle, en el cual se pueden visualizar las respuestas.
- 2.- **Descripción del comunicado:** se puede leer la descripción de un comunicado.
- 3.- **Imagen de un comunicado:** se visualiza la imagen en miniatura del comunicado.
- 4.- **Petición de borrado:** si se pulsa sobre este enlace se enviará al administrador un email indicando que se quiere borrar el anuncio, el administrador

atenderá la petición en cuanto estime oportuno. Si no se produce ninguna incidencia al enviar la petición, aparecerá el siguiente mensaje: “Se ha creado satisfactoriamente la petición”. Esta opción solo aparece en los anuncios cuyo usuario asociado sea el mismo que el usuario autenticado.

- 5.- **Paginación:** pulsando sobre los números, se accede a la página, en la cual pueden visualizarse los comunicados que le corresponde. Si se pulsan las flechas, se accede a la primera página o última respectivamente.
- 6.- **Número de comunicados por página:** se visualizarán por página, el número de comunicados que se hayan indicado en el combo, ver figura A.6, existe la opción de “todas”, que mostrará todos los comunicados.
- 7.- **Nuevo comunicado:** pulsando en este enlace se podrá crear un nuevo Comunicado.

VISUALIZACIÓN DEL DETALLE DE UN COMUNICADO



Figura A.7: Detalle Comunicado, usuario

Para acceder al detalle del comunicado, se pulsa en la pantalla de listado de comunicados, ver figura A.6, sobre el título de un comunicado.

Al pulsar sobre el detalle, se visualiza la pantalla representada en la figura A.7. Como se puede observar, la foto tiene tamaño mayor y se pueden ver las respuestas que ha escrito el administrador en caso de que existan.

NUEVO COMUNICADO

Si se pulsa en la pantalla de listado de comunicados sobre el enlace de nuevo comunicado, ver figura A.6, se puede añadir un nuevo comunicado.

Para ello aparecerá la siguiente pantalla:



Figura A.8: Nuevo Comunicado, usuario

En esta pantalla se añade el título, descripción e imagen si se tiene. A continuación se pulsa en guardar. Si se ha guardado correctamente aparecerá el siguiente mensaje “Se ha guardado el anuncio - sugerencia.”. No se podrá visualizar en el acto el anuncio, puesto que lo tiene que validar el administrador. Se le enviará un correo y este aprobará o rechazará el comunicado según estime oportuno. En cuanto se modifique el estado del anuncio, se le enviará al usuario un email informativo.

A.1.6.SALIR

Se accede a esta opción pulsando el enlace número cinco, mostrado en la figura A.3.

Al pulsar en salir, se procede a cerrar la sesión y salir de la aplicación.

A.2. PERFIL ADMINISTRADOR

En esta sección explicaremos las opciones extras de las que dispone este usuario, se suponen explicadas en el apartado anterior, la funcionalidades que tienen en común con el usuario sin privilegios.

A.2.1.MENÚ DE INICIO DE ADMINISTRACIÓN

En el menú de inicio de administración, se puede observar que existe un nuevo menú llamado administración.



Figura A.9: Nuevo Menú Inicio, administración

Como se puede observar en la figura A.9, se han enumerado los enlaces para describir sus funciones en los siguientes apartados.

A.2.2. VER LISTA DE USUARIOS

Se accede a esta pantalla pulsando sobre el enlace número uno de la figura A.9. En esta pantalla, ver figura A.10, se realizan las tareas de gestión de usuario. Se pueden crear nuevos usuarios, modificarlos y añadirles privilegios.



Figura A.10: Menú de administración de usuarios

A continuación vamos a detallar la funcionalidad de la figura A.10:

AÑADIR UN NUEVO USUARIO

Si pulsamos sobre el botón añadir, se accede a la pantalla descrita en el apartado anterior, alta de usuario, ver figura A.2. Pero en el caso del usuario administrador, se habilitan opciones extras. Estas se muestran en la siguiente figura:



Figura A.11: Alta Administrador, administración

Los nuevos apartados son los siguientes:

- ★ **Configuración de la cuenta:** indica el estado de la cuenta, se puede elegir una de las siguientes opciones:
 - 1.- **Habilitada:** el usuario de la cuenta puede acceder con su cuenta a la aplicación.
 - 2.- **Caducada:** el usuario tiene la cuenta caducada, por tanto no puede acceder a la aplicación. Para reactivarla se tiene que poner en contacto con el administrador de cuentas de usuario.
 - 3.- **Bloqueada:** el administrador bloquea la cuenta de usuario. Para reactivarla se tiene que poner en contacto con el administrador de cuentas de usuario.

4.- Contraseña Caducada: el usuario tiene la contraseña caducada. Para reactivarla tiene que modificar su contraseña.

- ★ **Asignar roles:** el administrador puede darle permisos al usuario que quiere crear. Se le puede asignar tantos roles como existan. Cada uno de los roles tendrá privilegios distintos.

Una vez rellenados todos los datos se puede guardar el usuario. En caso de no querer dar de alta un usuario, se puede cancelar. Estas opciones están disponibles si pulsa su respectivo botón de la pantalla de alta de usuarios, ver figura A.11.

MODIFICAR UN USUARIO

Se accede a esta opción si pinchamos sobre el nombre de usuario, ver figura A.10. La pantalla será la misma que en el apartado de alta Administrador, ver figuras A.2 y A.11. Se pueden comprobar los datos del usuario y modificar si es necesario pulsando el botón de Guardar.

También se puede borrar el usuario pulsando sobre el botón de borrar.

Si no se desea hacer ninguna modificación se puede pulsar el botón de cancelar.

Una vez terminada la administración de usuarios podemos pulsar sobre el botón de hecho, ver figura A.10.

A.2.3.USUARIO CONECTADOS

Se accede a esta pantalla pulsando sobre el menú de inicio, administración, usuarios conectados, ver figura A.9.

En esta pantalla se muestra una lista de los usuarios que están en este

momento conectados al aplicativo.

A.2.4.COMUNICADOS: ANUNCIOS Y SUGERENCIAS

En esta sección el administrador podrá aprobar, rechazar, eliminar y responder a los comunicados.

BARRA DE HERRAMIENTAS DEL ADMINISTRADOR

A continuación describimos las funcionalidades del administrador:



Figura A.12: Barra de Herramientas del Administrador, administración

Como se puede observar en la lista de comunicados, ver figura A.12, cuando se entra como administrador se tienen opciones diferentes. Aparecen con fondo de color carne una barra de administración, además, debajo de la descripción de cada comunicado se muestra el estado del mismo.

Se ha enumerado, como se puede ver en la figura A.12, las nuevas funcionalidades para la administración de comunicados y a continuación se procede a explicar su funcionamiento.

- 1.- **Estado del comunicado:** informa del estado en el que se encuentra actualmente el comunicado.
- 2.- **Editar:** al pulsar sobre el icono, aparecerá la pantalla de nuevo comunicado, ver figura A.8, pero con los datos del comunicado. Estos se pueden modificar si se estima oportuno y para que se guarden los cambios se debe pulsar el botón de guardar.
- 3.- **Eliminar:** al pulsar sobre este icono, se eliminará el comunicado.
- 4.- **Publicar:** al pulsar sobre este icono, el estado del comunicado pasará a publicado, por tanto, cualquier usuario podrá visualizar el contenido del mismo.
- 5.- **Borrador:** al pulsar sobre este icono, el estado del comunicado será borrador, en este estado el comunicado solo puede ser visto por el administrador. Esta opción no implica que se este rechazado el comunicado, sino que, no ha sido revisado aún por el administrador.
- 6.- **Rechazar:** al pulsar sobre este icono, el anuncio queda rechazado. Por tanto, no podrá ser visualizado por el resto de usuarios no administradores.

Si se produce un cambio de estado, este siempre se puede modificar en cuanto el administrador estime oportuno.

A.2.5. RESPONDER UN COMUNICADO

Para acceder a esta opción se tiene que pinchar sobre el título del comunicado en la pantalla de lista de comunicados, ver figura A.12. En el detalle, se tiene la posibilidad de responder a un comunicado, como se puede observar en la figura A.13

Se escribe la respuesta y se pulsa guardar.



Figura A.13: Respuesta a un comunicado, administración

Una vez guardada la respuesta, esta se encuentra en estado borrador. Para que sea visible por todos los usuarios, se debe cambiar su estado, para ello se tiene que pulsar en la opción aceptar.

Si se quiere borrar una respuesta, pulsaremos en la opción borrar.

Con esta explicación se da por concluido el ciclo de la aplicación y el manual de usuario.

Apéndice B. Glosario de Terminos

- 1.- **Agile** - Marco de trabajo conceptual de la ingeniería de software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto.
- 2.- **Android** - Sistema operativo destinado principalmente a dispositivos móviles.
- 3.- **Artefactos, de Scrum** - Son herramientas que permiten mantener organizados los proyectos. Estos artefactos, ayudan a planificar y revisar cada uno de los sprints.
- 4.- **BDD** - Programación orientada a comportamiento, forma de programar donde primero se realizan los test que definen el comportamiento y, después, el código para que cumpla el comportamiento.
- 5.- **Cascada, modelo** - En Ingeniería de software, el desarrollo en cascada (también llamado modelo en cascada), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior.
- 6.- **Demo** - Presentación de una aplicación, el objetivo de cada sprint es tener una demo presentable al cliente.
- 7.- **Desarrollador** - Persona que mediante un lenguaje de programación construye un sistema.

- 8.- **Hibernate** - Herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.
- 9.- **Historia, de usuario** - Es la explicación de una función del sistema, redactada en un lenguaje natural, claro y conciso sin ambigüedad.
- 10.- **Meta, de sprint** - Objetivo que se marca en cada sprint, suelen usarse lemas para animar al equipo para que realice dicho objetivo.
- 11.- **MVC** - Modelo Vista Controlador. MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.
- 12.- **Pila, de productos** - Un conjunto de historias de usuario que definen un sistema.
- 13.- **Planning poker** -Planning poker es una técnica para calcular una estimación basado en el consenso, en su mayoría utilizado para estimar el esfuerzo o el tamaño relativo de las tareas de desarrollo de software.
- 14.- **Retrospectiva** - Al finalizar el desarrollo de un proyecto, se realiza una reunión llamada retrospectiva cuya finalidad es la de exponer lo positivo y negativo que ha surgido durante el desarrollo del proyecto y qué se ha aprendido.
- 15.- **Scrum** - Es un marco de trabajo para la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.
- 16.- **Scrum máster** - Es el facilitador del equipo, la figura de líder servil que vela por que los desarrolladores puedan cumplir su trabajo.
- 17.- **S.O.L.I.D.** - Cinco principios fundamentales, uno por cada letra, que hablan del diseño orientado a objetos en términos de la gestión de dependencias.

- 18.- **Spring** Es un marco de trabajo de código abierto de desarrollo de aplicaciones para la plataforma Java o .NET .
- 19.- **Sprint** - Intervalo de tiempo en el cual se tiene que realizar una meta de Spring y tener una demo que enseñar al cliente.
- 20.- **Stakeholder** - Son los clientes, usuarios, inversores involucrados en el proyecto. Solo participan en las revisiones de sprint (las demos).
- 21.- **Strut** - Herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma Java EE.
- 22.- **UML** - Es un lenguaje de modelado gráfico, para visualizar, especificar, construir y/o documentar un sistema.
- 23.- **XP, eXtreme Programming** - Es un enfoque de la ingeniería de software formulado por Kent Beck, que pertenece al desarrollo ágil del software.