

Planificación y **G**estión de **P**royectos

ShadowTester

Yolanda Alonso Barbero
Diego Güemes Peña
José Vicente Ortega Bartolomé
Jorge Valdivielso Báscones

ÍNDICE

PLANIFICACIÓN INICIAL.....	3
OBJETIVO.....	3
CONCEPTOS.....	3
JOB SHADOWING	3
PARTICIPANTES	3
REQUISITOS INICIALES.....	3
PROTOTIPO DEL DISEÑO DE LA INTERFAZ DE USUARIO	3
PLAN DEL PROYECTO SOFTWARE.....	5
PLANIFICACIÓN TEMPORAL	5
DESCRIPCIÓN DE LAS TAREAS	6
SCRUM.....	8
PRODUCT BACKLOG	8
SPRINT BACKLOGS.....	9
DIVISIÓN DE LAS HISTORIAS DE USUARIO EN TAREAS.....	11
ISSUE TRACKER.....	12
XP-DEV.....	12
MANTIS.....	13
SISTEMA DE CONTROL DE VERSIONES.....	14
SUBVERSION.....	14
GIT	16
PLASTIC SCM	17
INTEGRACIÓN CONTINUA	18
TESTS AUTOMATIZADOS	19

PLANIFICACIÓN INICIAL

OBJETIVO

El objetivo principal del **proyecto pgp2011** es la creación de un programa que permita la observación indirecta de cómo los usuarios utilizan ciertas aplicaciones de las que se quiere probar si están bien diseñadas para su manejo.

CONCEPTOS

JOB SHADOWING

Con esta aplicación se pretende implementar la técnica del “job shadowing” que aparece en el libro *Designed For Use* de Lukas Mathis.

El “job shadowing” puede traducirse en español como aprendizaje por observación del trabajo y consiste en visitar a los usuarios para observarlos en el momento en el que van a utilizar el programa que estamos poniendo a prueba.

El objetivo de esta técnica es observar las necesidades del usuario para saber qué tipo de producto debemos crear o qué tipo de mejoras se pueden realizar en función de la observación realizada.

El **proyecto pgp2011** consiste en crear un programa que permita este proceso sin necesidad de acudir al lugar de trabajo de los usuarios.

PARTICIPANTES

Desarrolladores	Clientes
Yolanda Alonso Barbero Diego Güemes Peña José Vicente Ortega Bartolomé Jorge Valdivielso Báscones	Pablo Santos Luaces

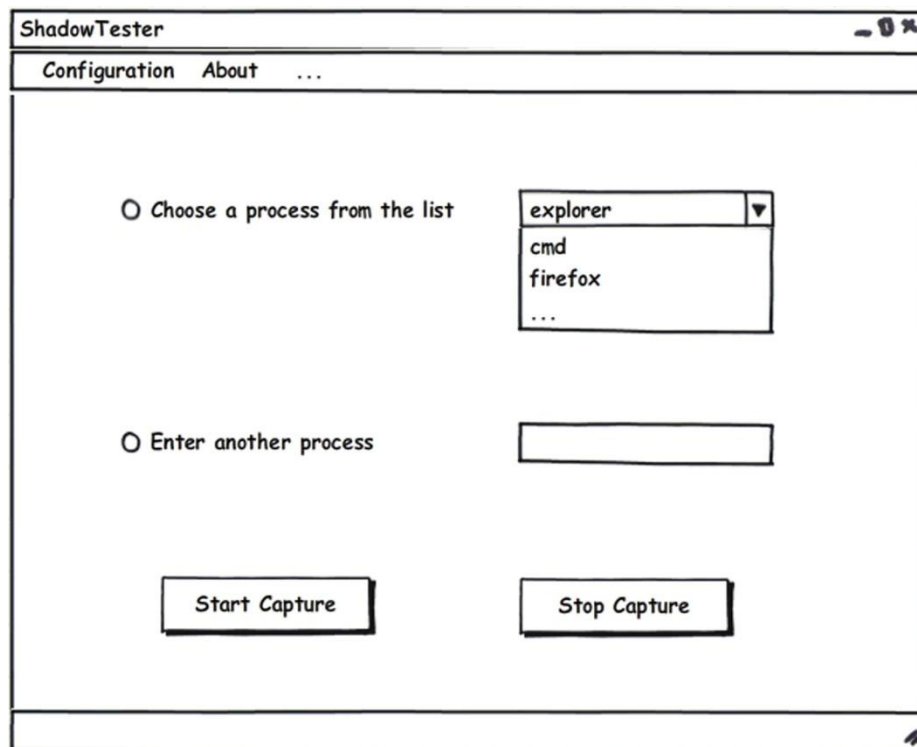
REQUISITOS INICIALES

Los requisitos iniciales del programa se trataron en la primera reunión celebrada con el cliente y son los siguientes:

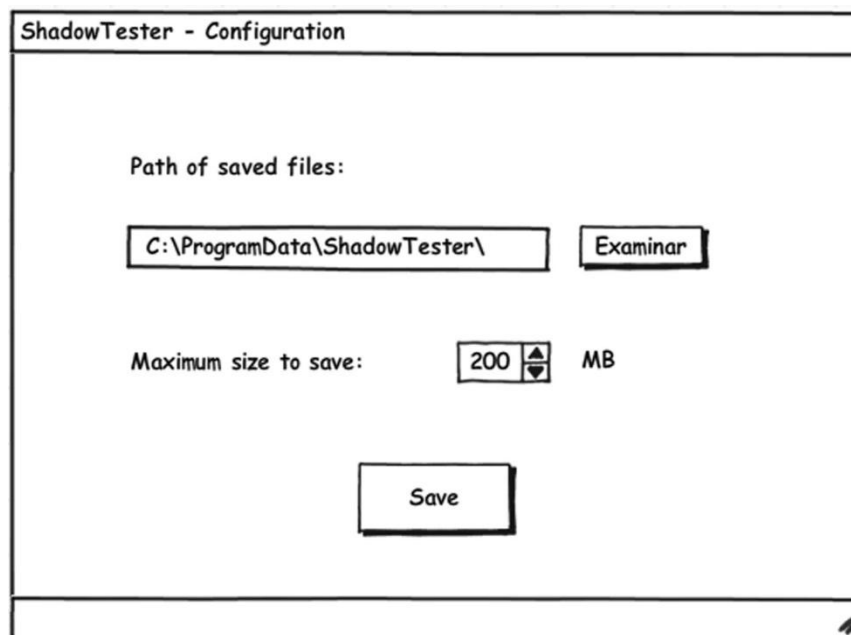
1. Ser capaz de realizar capturas de pantalla.
2. Las capturas de pantalla se deben realizar repetidamente con una frecuencia de tiempo determinada.
3. Solo se realizan capturas cuando la aplicación que estamos probando se encuentra en la ventana que está en primer plano.
4. Con las capturas realizadas generar un vídeo.

PROTOTIPO DEL DISEÑO DE LA INTERFAZ DE USUARIO

Pantalla principal del programa:



Configuración:



Sobre el programa:



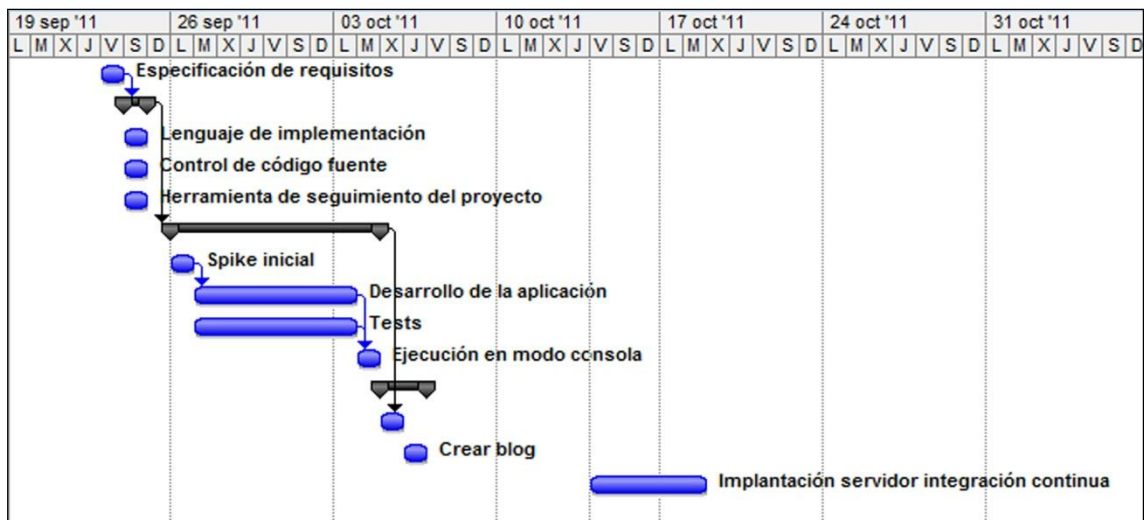
PLAN DEL PROYECTO SOFTWARE

Para poder llevar a cabo el proyecto con éxito es necesario la realización de un plan que nos permita organizarnos desde un principio, haciendo un análisis de las posibles tareas y del tiempo que nos va a costar llevar a cabo cada una de ellas.

PLANIFICACIÓN TEMPORAL

Es de fundamental importancia dividir el proyecto en diversas tareas para poder hacer una estimación del tiempo que ocupará cada una de ellas. Además es necesario organizar el orden en que dichas tareas se deben realizar ya que puede haber algunas que no puedan comenzar hasta que otra haya terminado.

Para organizar toda esta información de una forma gráfica que facilite su comprensión en un solo vistazo se va a utilizar un Diagrama de Gantt.



	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Especificación de requisitos	1 día	vie 23/09/11	vie 23/09/11	
2	<input checked="" type="checkbox"/> Toma de decisiones iniciales	1 día	sáb 24/09/11	sáb 24/09/11	1
3	Lenguaje de implementación	1 día	sáb 24/09/11	sáb 24/09/11	
4	Control de código fuente	1 día	sáb 24/09/11	sáb 24/09/11	
5	Herramienta de seguimiento del proyecto	1 día	sáb 24/09/11	sáb 24/09/11	
6	<input checked="" type="checkbox"/> Desarrollo primera versión software	25,25 días	lun 26/09/11	mar 04/10/11	2
7	Spike inicial	1 día	lun 26/09/11	lun 26/09/11	
8	Desarrollo de la aplicación	5 días	mar 27/09/11	lun 03/10/11	7
9	Tests	5 días	mar 27/09/11	lun 03/10/11	
10	Ejecución en modo consola	1 día	mar 04/10/11	mar 04/10/11	8;9
11	<input checked="" type="checkbox"/> Documentación	2 días	mié 05/10/11	jue 06/10/11	
12	Manual de usuario	1 día	mié 05/10/11	mié 05/10/11	6
13	Crear blog	1 día	jue 06/10/11	jue 06/10/11	
14	Implantación servidor integración continua	3 días	vie 14/10/11	mar 18/10/11	

DESCRIPCIÓN DE LAS TAREAS

- Especificación de requisitos:

Tras la primera reunión se crea la primera lista de requisitos especificados por el cliente.

Requisitos	Completado
Realizar capturas de pantalla.	SÍ
Con una frecuencia de tiempo determinada.	SÍ
Solo cuando la aplicación esté en primer plano.	SÍ
Generar un vídeo.	NO

- Lenguaje de implementación:

Es fundamental como primer paso elegir el lenguaje que se va a utilizar para el desarrollo.

Se ha elegido C#, lenguaje de alto nivel orientado a objetos que facilita el desarrollo al disponer de una amplia biblioteca de clases. Es multiplataforma y solamente requiere la instalación de .NET Framework.

- Control de código fuente:

Hemos decidido utilizar un sistema de control de versiones para que todos los integrantes del grupo podamos trabajar de forma conjunta, siendo capaces de modificar el mismo conjunto de archivos desde distintas ubicaciones y además poder mantener un control de las versiones que se van generando tanto del código como de la documentación.

El sistema de control de versiones elegido ha sido Subversion y vamos a trabajar con un repositorio alojado en xp-dev en el que hay que crear las carpetas para estructurar el contenido: trunk, branches y tags.

- Herramienta de seguimiento del proyecto:

XP-Dev, página elegida para trabajar con el sistema de control de versiones es además una herramienta para poder realizar un seguimiento del proyecto, permite la creación de historias de usuario, tareas de dichas historias y la asignación de éstas a los distintos miembros del equipo de desarrollo, además de almacenar estadísticas sobre la evolución del proyecto.

Stories: 5 (2 remaining) Tasks: 9 (3 remaining) Bugs: 0 Estimated Hours: 8 (7.5 added, 0 remaining) 100 %							
Type	Id	Description	Hours			Assigned To	Completed
			Total	Added	Remaining		
	#1	Realizar capturas de pantalla.	2	2	0	 pgp2011	 100 %
	#4	Detectar la ventana activa.	2	2	0	 pgp2011	 100 %
	#7	Sacar capturas de pantalla con X frecuencia.	2	1	0	 pgp2011	 100 %
	#11	Interfaz Modo Consola	2	2.5	0	 pgp2011	 125 %
	#14	Generar un vídeo a partir de las imágenes previamente generadas	0	0	0	 pgp2011	 0 %

- Spike inicial:

Debido a que los desarrolladores no están muy familiarizados con las labores de realizar capturas de pantalla y detectar procesos activos, se dedicará un tiempo inicial a tener un primer contacto con bibliotecas que permitan realizar estas funcionalidades en un proyecto aparte.

- Desarrollo de la aplicación:

Durante el desarrollo de la aplicación se realiza todo el código que dará funcionalidad a la primera versión del producto software.

- Tests:

Con el fin de aumentar la calidad del código, se realizarán tests que servirán principalmente como herramienta de diseño, aunque también proporcionan una garantía del correcto funcionamiento de la aplicación.

- Manual de usuario:

Una vez desarrollada la primera versión del producto es necesario la creación de un manual de usuario que proporcione al cliente una guía útil sobre cómo manejar el programa. El manual debe ser sencillo de entender para que cualquier tipo de cliente, incluso usuarios no expertos, puedan manejar de forma correcta el programa.

- Crear un blog:

Para poder dar a conocer el producto al mayor número de clientes potenciales posible se creará un blog público en Internet, describiendo la funcionalidad del programa, el estado del desarrollo del mismo y proporcionando la descarga de una versión del producto a quienes quieran probarlo.

- Implantación de un servidor de integración continua:

En el desarrollo de software, se destina mucho tiempo a la integración de los distintos componentes que han realizado los miembros del equipo, compilación, ejecución de las pruebas, etc.

Para automatizar este proceso, se implantará un servidor de integración continua, permitiendo detectar los problemas comunes de la integración y disponer de la última build.

SCRUM

PRODUCT BACKLOG

ID	IMP	HISTORIAS DE USUARIO	ESTIMACIÓN	SPRINT	HECHO
1	140	Sacar capturas de pantalla del proceso activo.	5	1	Sí
2	130	Poder especificar una lista de procesos.	2	1	Sí
3	125	Interfaz consola.	5	1	Sí
4	120	Generar vídeo a partir de las capturas de pantalla.	8	2	Sí
5	110	Poder especificar la ruta donde se guarda el vídeo.	1	2	Sí
6	100	Reducir el área de captura a la ventana activa.	5	3	Sí
7	80	Poder especificar la frecuencia de las capturas de pantalla.	1	3	Sí
8	75	Mostrar información del vídeo en el primer frame del vídeo.	5	4	Sí
9	65	Poder añadir procesos en medio de una grabación.	2	4	Sí
10	60	Interfaz gráfica	13		No
11	55	Incluir ayuda del programa.	3		No
12	50	Tener opción de desechar una grabación.	2		No
13	40	Poder configurar el formato de vídeo.	3		No
14	35	Proteger el programa con contraseña.	2		No
15	30	Reproducir vídeo desde el propio programa.	3		No

ID	IMP	HISTORIAS TÉCNICAS	ESTIMACIÓN	SPRINT	HECHO
16	115	Control de versiones con Plastic.	1	2	Sí
17	105	Instalar Issue Tracker.	2	2	Sí
18	90	Crear script de compilación.	5	3	Sí
19	85	Instalar servidor de Integración Continua.	5	4	Sí

El campo de importancia (IMP) está realizado con un rango de 10 a 150, con el fin de que haya suficiente holgura entre las distintas historias de usuario. El hecho de que una historia tenga asignada una importancia de 50 y otra 10, significa simplemente que la primera es más importante, es decir, la importancia no es proporcional.

La estimación en puntos de historia se ha realizado con una serie similar a la de Fibonacci: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100 para evitar la sensación de exactitud, no tiene sentido discutir si una historia debería tener asignada 19, 20 o 21 puntos, sino que con 20, sabemos la amplitud de esa historia.

Para realizar las estimaciones, decidimos utilizar “Wideband Delphi”, una técnica de estimación que tiene como objetivo el consenso. Existen múltiples variaciones de esta técnica, una de las que vimos en clase es “Planning Poker”, pero debido a que no disponíamos de una baraja adecuada para aplicarlo y para simplificarlo decidimos utilizar una variante muy similar denominada “Flying Fingers” que consiste en:

- Esconder la mano debajo de la mesa.
- En el momento del consenso todos los miembros ponen la mano encima de la mesa con los dedos levantados, indicando cuánto creemos que puede durar una tarea.
- Si no hay acuerdo, los miembros del grupo discuten por qué y se repite el proceso hasta que hay acuerdo en la duración de la tarea.

SPRINT BACKLOGS

<i>SPRINT BACKLOG 1</i>	ESTIMACIÓN
Sacar capturas de pantalla del proceso activo	5
Poder especificar una lista de procesos	2
Interfaz consola	5
<i>TOTAL</i>	12

<i>SPRINT BACKLOG 2</i>	ESTIMACIÓN
Generar vídeo a partir de las capturas de pantalla.	8
Control de versiones con Plastic.	1
Poder especificar la ruta donde se guarda el vídeo.	1
Instalar Issue Tracker	2
<i>TOTAL</i>	12

<i>SPRINT BACKLOG 3</i>	ESTIMACIÓN
Reducir el área de captura a la ventana activa.	5
Crear script de compilación	5
Poder especificar la frecuencia de las capturas de pantalla.	1
<i>TOTAL</i>	11

<i>SPRINT BACKLOG 4</i>	ESTIMACIÓN
Instalar servidor de Integración Continua.	5
Mostrar información del vídeo en el primer frame del vídeo.	5
Poder añadir procesos en medio de una grabación.	2
<i>TOTAL</i>	12

DIVISIÓN DE LAS HISTORIAS DE USUARIO EN TAREAS

Debido a que aprendimos a utilizar Scrum después de iniciar el proyecto, únicamente están divididas en tareas las historias a partir de tercer Sprint.

Para la estimación de tareas hemos usado el método PERT, calculando que dada una tarea:

$$\text{Estimación} = \frac{\text{Mejor Caso} + 4 * \text{Caso Más Probable} + \text{Peor Caso}}{6}$$

HISTORIA DE USUARIO	TAREAS	Estimación	Mejor	Probable	Peor	MRE	Real
Reducir el área de captura a la ventana activa.	Obtener tamaño de la ventana activa.	3,67	2	3,5	6	8%	4
	Obtener coordenadas de la ventana activa.	1,67	1	1,5	3	17%	2
	Asignar dichos parámetros a la captura.	3,00	2	3	4	20%	2,5
Crear script de compilación	Aprender la sintaxis de Nant.	3,00	2	3	4	20%	2,5
	Crear archivo de build.	2,00	1	2	3	0%	2
	Ejecutar los tests desde el propio script.	1,17	1	1	2	17%	1
Poder especificar la frecuencia de las capturas de pantalla.	Implementar funcionalidad.	1,50	1	1,5	2	50%	1
	Crear tests.	0,58	0,5	0,5	1	17%	0,5
Instalar servidor de Integración continua	Aprender la sintaxis de CCNet.	3,33	2	3,5	4	17%	4
	Integrar con el control de versiones.	2,00	1	2	3	0%	2
Mostrar información del vídeo en el primer frame del vídeo.	Obtener los datos del sistema.	1,17	1	1	2	22%	1,5
	Asignar dichos datos al primer frame del vídeo.	4,17	3	4	6	17%	5
	Crear tests.	0,58	0,5	0,5	1	42%	1
Poder añadir procesos en medio de una grabación.	Implementar funcionalidad.	1,17	1	1	2	17%	1
	Crear tests.	0,58	0,5	0,5	1	17%	0,5

ISSUE TRACKER

Los issue tracker son programas que permiten administrar una lista de incidencias a la hora de desarrollar software. Esta lista permite mantener una organización de todas las tareas del proyecto, estableciendo en cada una un valor de prioridad o asignándoselas a los desarrolladores del proyecto. De esta forma, en cualquier momento, todos los miembros del equipo pueden tener acceso a la información de qué es lo que ya se ha hecho, lo que se está haciendo y lo que queda por hacer.

Una funcionalidad muy útil de los issue tracker es la posibilidad que tienen de conectarlos con un sistema de control de versiones, de manera que cada una de las incidencias del issue tracker se corresponda con determinados cambios realizados en el código y recogidos por el control de versiones.

Durante la realización del proyecto ShadowTester hemos utilizado dos sistemas diferentes para llevar a cabo el seguimiento de incidencias:

XP-DEV

Nada más comenzar el desarrollo de ShadowTester decidimos utilizar el sitio xp-dev.com porque lo conocíamos de anteriores proyectos. En XP-Dev, además de trabajar con un repositorio de Subversion como se explica en el apartado del control de versiones, podíamos organizar la lista de todas las tareas, incluir su estimación en número de horas, asignarlas a un desarrollador e ir indicando lo que se había trabajado en cada una o cuando una tarea había sido completada. Eso fue nuestro Issue Tracker inicialmente.




The screenshot shows the 'Project Activity' page on the XP-Dev website. The page has a header with the title 'Planificación y Gestión de Proyectos' and a user profile for 'pgp2011' with links to 'projects', 'account', and 'logout'. Below the header is a navigation bar with tabs: 'Dashboard', 'Project Tracking', 'Wiki (1)', 'Forums (0)', 'Blog (0)', 'Repository', 'Activity (40)', 'Search', and 'Settings'. The 'Activity (40)' tab is selected. The main content area is titled 'Project Activity' and shows a list of activities. On the right side of the activity list, there is a pagination control showing 'Page: 1 2 3 4 5', with '3' being the active page. The activities listed are:

- 3 months ago
- pgp2011 updated Task #6
- pgp2011 created 1.0 hours in Task #6
- pgp2011 updated Task #5
- pgp2011 created 1.0 hours in Task #5
- pgp2011 created Task #6
- pgp2011 created Task #5
- pgp2011 created Story #4
- pgp2011 deleted TaskHour #49724
- pgp2011 created 1.0 hours in Task #3
- pgp2011 created Task #3

MANTIS

Después de una sesión de clase en la que se nos recomendaron diversos issue trackers o bug trackers, entre ellos: Bugzilla, Mantis, FogBugz o Jira; decidimos cambiar y trasladar el issue tracking que hasta entonces habíamos llevado en XP-Dev a Mantis.

Entre todos los bug trackers decidimos decantarnos por Mantis por varios motivos. El primero de ellos es que es gratuito, al contrario que por ejemplo Jira, lo cual era una ventaja importante para nosotros. Motivos económicos a parte, nos pareció sencillo de utilizar, permite entre otras cosas asignar la prioridad de cada tarea, un identificador único para cada una y establecer el desarrollador que la va a realizar. Además, uno de los factores más decisivos fue que es posible conectar este bug tracker con el sistema de control de versiones Plastic SCM, con el que terminamos trabajando.



Logged in as: *administrator*
(administrator)

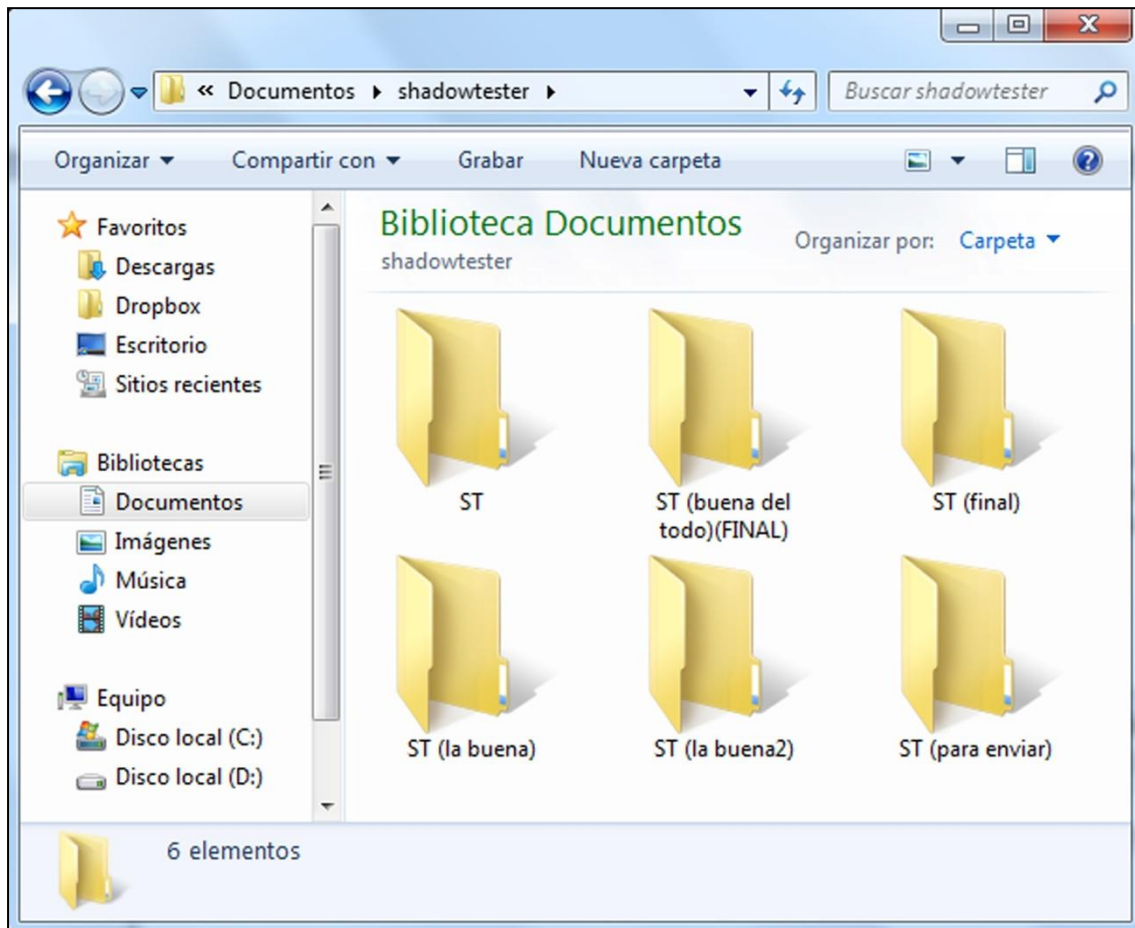
[Main](#) | [My View](#) | [View Issues](#) | [Report Issue](#) | [Change Log](#) | [Roadmap](#) | [Summary](#) | [Manage](#) | [My Account](#) | [Logout](#)

Issue #

	P	ID▲	#	Category	Severity	Status	Updated	Summary
<input type="checkbox"/>			0000001	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Command-line interface
<input type="checkbox"/>			0000002	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Take screenshots of active process.
<input type="checkbox"/>			0000003	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Specify a list of processes
<input type="checkbox"/>			0000004	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Generate video from screenshots.
<input type="checkbox"/>			0000005	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Specify a path to save the video.
<input type="checkbox"/>			0000006	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Include only the active window in the screenshot.
<input type="checkbox"/>			0000007	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Specify the frame rate.
<input type="checkbox"/>			0000008	[ShadowTester] User stories	feature	assigned (shadowtester)	2011-11-17	Specify new processes while the program is running.

SISTEMA DE CONTROL DE VERSIONES

Un sistema de control de versiones es una herramienta fundamental a la hora de desarrollar cualquier proyecto software. Gracias al control de versiones podemos tener organizados los cambios que se van realizando a lo largo del proyecto evitando situaciones como la de la imagen a continuación:



Cualquier sistema de control de versiones podrá ayudarnos a mejorar la organización de nuestro proyecto. Existen diversos sistemas actualmente, de entre los cuales hemos utilizado Subversion, Git y Plastic SCM durante el desarrollo de ShadowTester como explicamos a continuación.

SUBVERSION

Inicialmente decidimos utilizar XP-Dev para organizar el desarrollo de ShadowTester. Éste es el motivo de que al principio comenzásemos a trabajar con el sistema de control de versiones Subversion, ya que XP-Dev te permite tener los proyectos en repositorios de Subversion.

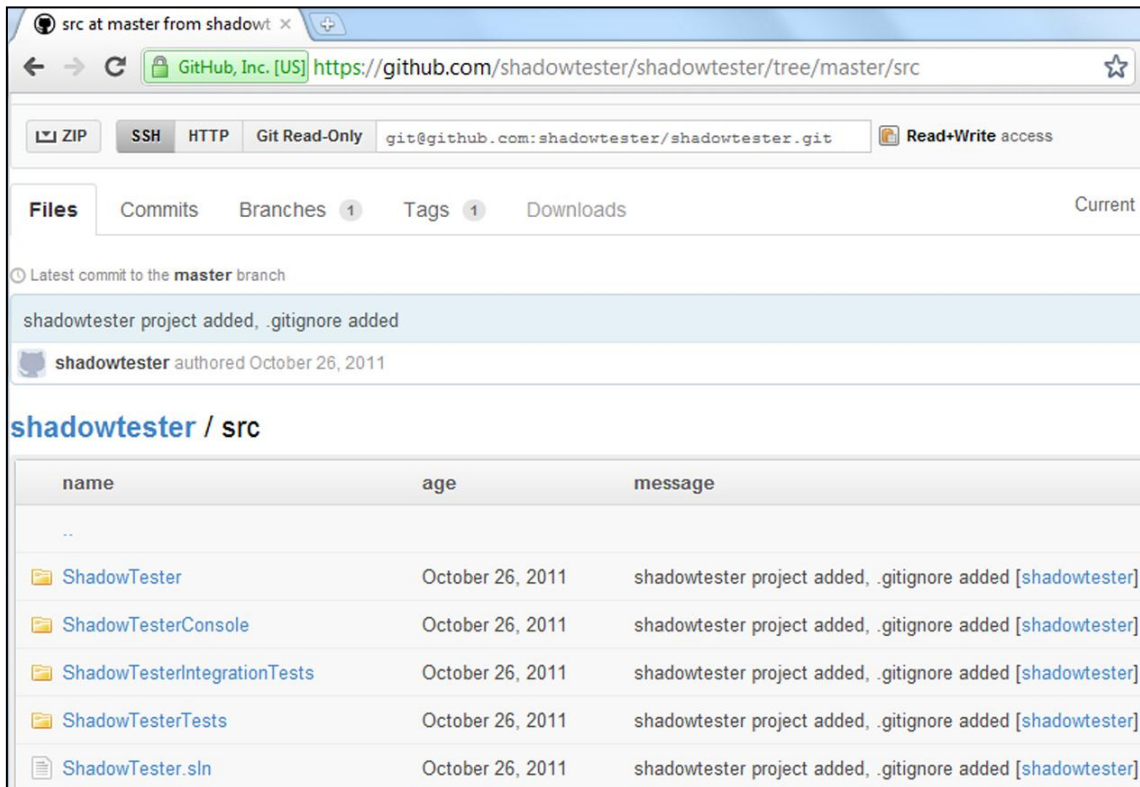
De esta forma comenzamos a desarrollar ShadowTester utilizando Subversion como control de versiones a través de XP-Dev y utilizando el cliente Tortoise SVN.

Pero el control de versiones Subversion tenía algunos inconvenientes: era necesario estar conectado a Internet para poder incorporar los cambios realizados al proyecto, aparecían

conflictos fácilmente y se hacía imposible el trabajar con ramas. Por estos motivos decidimos cambiar y utilizar Git, que presentaba ventajas con respecto a Subversion.



GIT

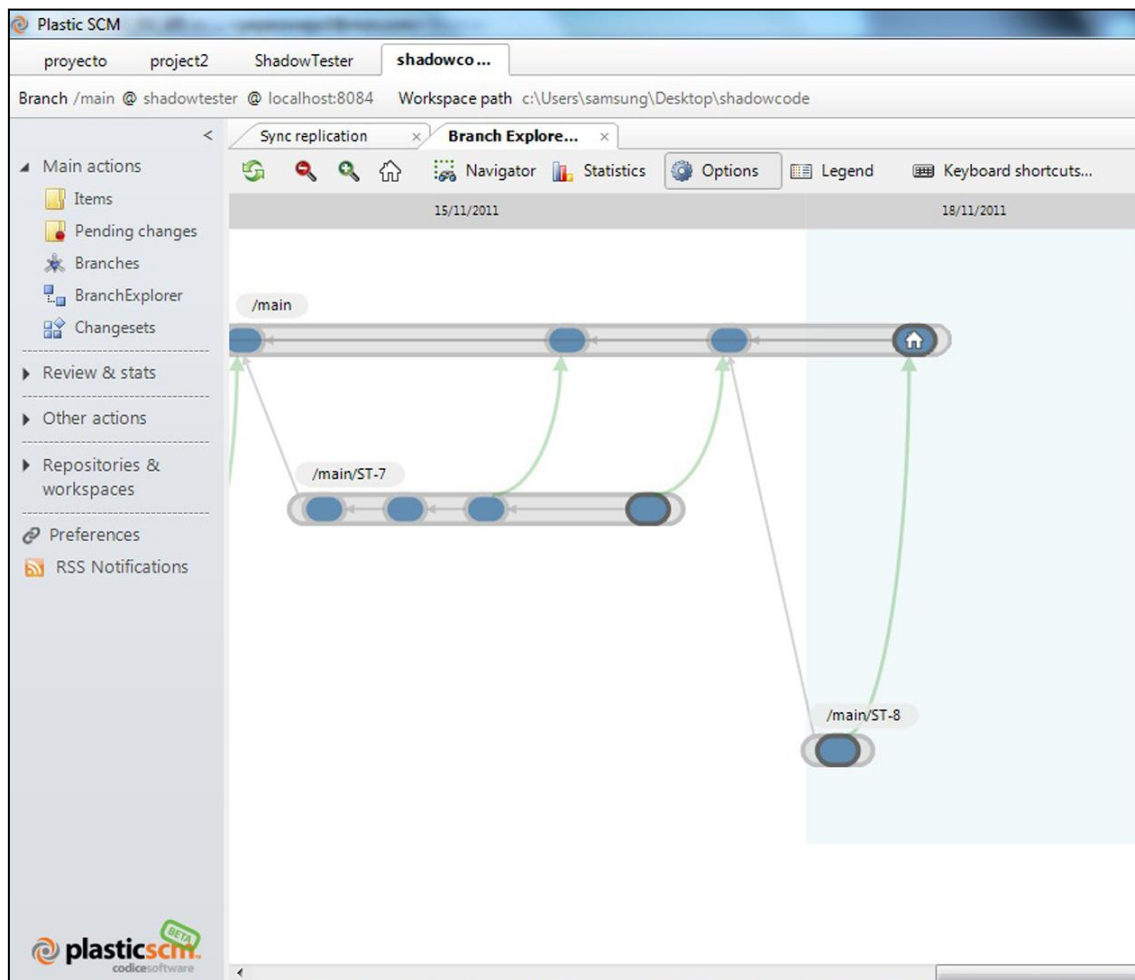


Como se muestra en la imagen trasladamos el código fuente de ShadowTester a un repositorio de Git. Para trabajar con este sistema de control de versiones decidimos utilizar la página de GitHub, sitio en el que hay multitud de proyectos software.

Sin embargo, y a pesar de aportar ventajas frente a Subversion, encontramos un inconveniente que hizo que volviésemos a pensar en cambios; y es que el repositorio de GitHub era público y, por lo tanto, accesible para todo el mundo, algo que no nos interesaba en ese momento.

Por esta razón y por multitud de ventajas que vimos en clase tras una sesión práctica con el sistema de control de versiones Plastic SCM, decidimos volver a cambiar.

PLASTIC SCM



Por último, después de haber pasado por Subversion y por Git, comenzamos a trabajar con Plastic SCM. Este sistema nos solucionaba todos los inconvenientes que habíamos encontrado en los anteriores además de proporcionarnos nuevas ventajas.

La facilidad que permite Plastic a la hora de trabajar con ramas nos brindaba la ocasión de poner en práctica una técnica aprendida en clase, "Branch per task". Esto consiste en crear una rama en el control de versiones para cada una de las tareas del proyecto, que es lo que finalmente decidimos hacer en el desarrollo de ShadowTester como se puede observar en la imagen anterior.

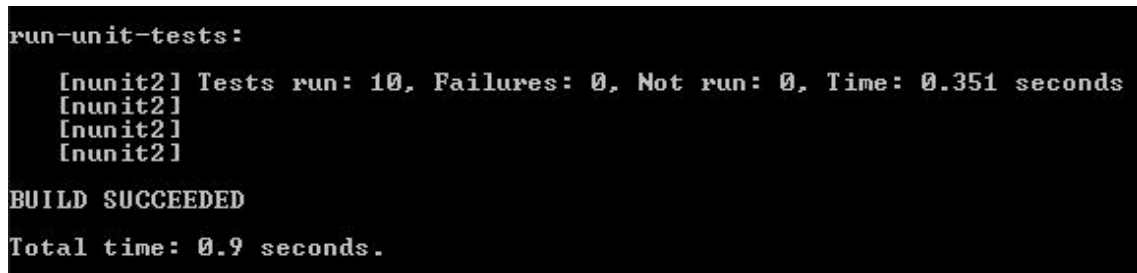
INTEGRACIÓN CONTINUA

Al comienzo del proyecto, no teníamos ningún tipo de proceso de construcción de software. Compilábamos con Visual Studio y sus opciones por defecto.

A pesar de que los IDEs son herramientas que incrementan nuestra productividad, no podemos depender de ellos completamente. Una de las claves del desarrollo ágil, es construir el software e integrar nuestro trabajo con el de los demás miembros del equipo de desarrollo con frecuencia.

Por lo tanto, para automatizar el proceso de compilación y romper la dependencia con Visual Studio decidimos crear un script con NAnt. De esta forma también, evitamos el típico “En mi Visual Studio / ordenador funcionaba”.

Por otra parte nos permite ejecutar toda la batería de tests desde la línea de comandos como se muestra en la imagen siguiente:



```
run-unit-tests:
  [nunit2] Tests run: 10, Failures: 0, Not run: 0, Time: 0.351 seconds
  [nunit2]
  [nunit2]
  [nunit2]
BUILD SUCCEEDED
Total time: 0.9 seconds.
```

Una de las mejores prácticas que hemos incorporado en el script de compilación, es definir distintas partes, de tal forma que el desarrollador no tiene que compilar el proyecto entero, sino únicamente la parte que le interesa. Por ejemplo también podría ejecutar únicamente los tests unitarios y no los de integración.

Pero la parte que nos faltaba es la de automatizar el proceso de obtener la última versión del software, compilarla y pasar los tests.

Aquí es donde entra en juego un servidor de integración continua, en nuestro caso seleccionamos Cruise Control .NET. Permite gestionar eventos cada cierto tiempo, bajarnos la última versión del código fuente y ejecutar los scripts de compilación.

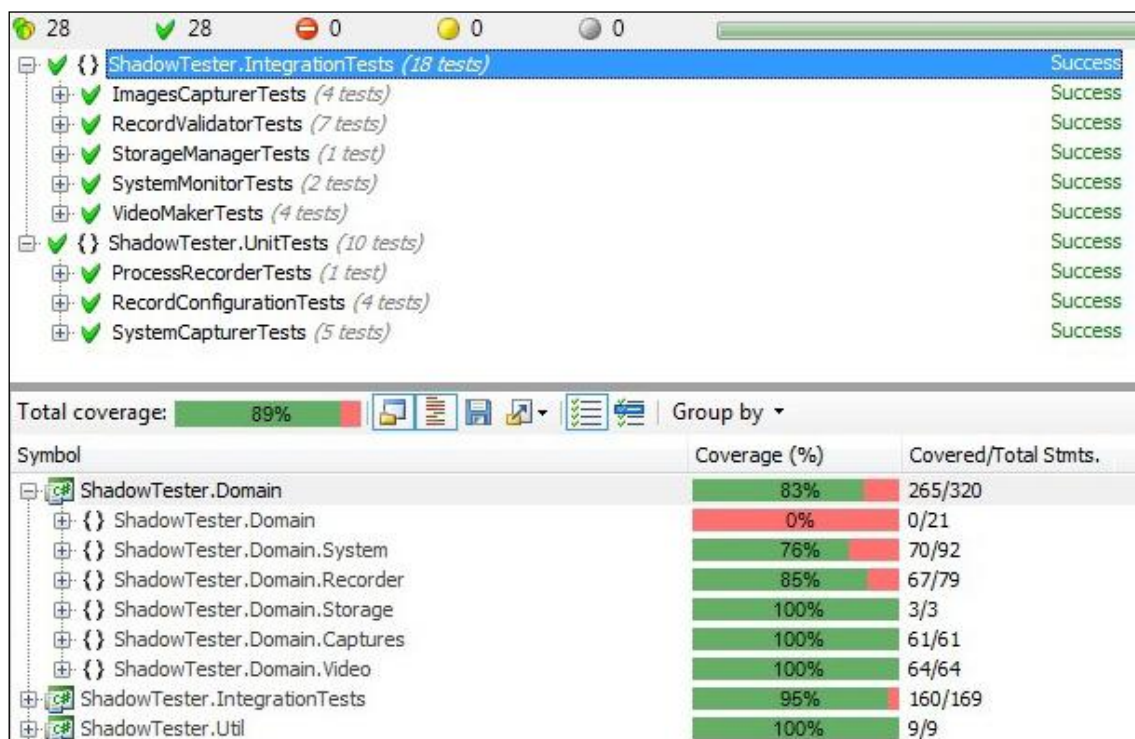
Mediante este proceso, disponemos siempre de una versión estable del producto y nos apoyamos en los tests para comprobar que no se ha roto nada.

TESTS AUTOMATIZADOS

Gracias a prácticas como TDD y revisiones continuas del código, construimos software de calidad y el mantenimiento del mismo es más simple al agregar características.

Dado que en el ciclo de TDD se deben pasar los tests cada muy poco tiempo, debemos intentar que sean rápidos. Nuestro software tiene que acceder a partes del sistema, que a menudo ralentizan los tests, es por ello por lo que hemos aislado estas partes a través de un framework de mocking: Rhino Mocks.

Sin embargo, nuestro primer objetivo siempre ha sido realizar la funcionalidad y una vez que estaba funcionando aplicar técnicas para dejar el código lo más claro posible. Los tests también son necesarios para llevar a cabo estas modificaciones sin romper nada y finalmente conseguimos código limpio que funciona como vemos en la siguiente imagen:



Pero la razón principal por la que hemos añadido un apartado de tests automatizados, es que todo lo explicado hasta el momento, estrategias de branching, integración continua... no tiene ningún sentido sin tests:

En el Issue Tracker definimos todas las tareas y de alguna manera las características que hacen a nuestro software valioso, que tienen sentido para su dominio. Los tests automatizados son una manera de documentación, podemos saber en todo momento que cosas tenemos completadas y cuáles no gracias a los tests, de no ser por ellos no tendríamos la certeza de que las cosas están funcionando como deberían.

En el SCM, da igual si se utiliza una estrategia de "Main line development" o "Branch per Task" si no existen los tests automatizados. No podrás saber si las cosas dejaron de funcionar al hacer "merge" o si hay un bug que se está propagando.

De la misma manera, en la integración continua... ¿cómo averiguaríamos que la release que acabamos de sacar funciona? Sin tests no hay manera, salvo probándolo a mano,

totalmente inviable dado que la finalidad de la integración continua es sacar releases con frecuencia.

En resumen, los tests automatizados son las piezas fundamentales para el resto del proceso ágil.