UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSIDAD
POLITECNICA
DE VALENCIA

ESCUELA POLITECNICA
SUPERIOR DE GANDIA

"Definición de la metodología para la gestión y verificación de proyectos Ágiles, haciendo uso de Microsoft Team Foundation Server, para su posterior uso en una aplicación real."

TRABAJO FINAL DE GRADO

Autor/a:
Antonio Jiménez Ramírez

Tutor/a:
Dr. Narcís Cardona Marcet

GANDIA, 2013

CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN

El objetivo del presente trabajo se basa en detallar el ciclo de vida de un proyecto de desarrollo software gestionado mediante una metodología de trabajo denominada Scrum y apoyándose en diversas herramientas de desarrollo como son el Team Foundation Server, Test Manager y Sharepoint de Microsoft.

Se detallaran todas las fases por las que debe pasar un proyecto desde su planificación a su puesta en producción, pasando por las fases de desarrollo y validación, enumerando en cada fase aquellos aspectos de gestión, documentación y/o validación que sean necesarios.

El proyecto se ha dividido en varias partes para facilitar su comprensión. La primera de ellas será una exposición de los fundamentos básicos necesarios. En segundo lugar, se definirá como será la gestión del proyecto ágil que se va a desarrollar. Seguidamente, se explicará la fase de testing que se debe ejecutar paralelamente pero de forma conjunta con el punto anterior.

Para finalizar, y unificar todos los conceptos anteriores, se simulará el desarrollo de un pequeño programa software siguiendo la metodología desarrollada y utilizando las herramientas de gestión, documentación y testing necesarias.

Palabras clave: Scrum, Metodologia, Agil, Desarrollo, Software.

ABSTRACT

The aim of this work is based on detailed life cycle of a software development project managed by a methodology called Scrum and leaning in various development tools such as Team Foundation Server, Test Manager and Microsoft Sharepoint.

It detailing all phases that must pass a project from planning to put into production, through the development and validation phases, in each phase listing those aspects of management, documentation and / or validation may be required.

The project has been divided into several parts for ease of understanding. The first will be a presentation of the basic fundamentals necessary. Second, will be defined as agile project management to be developed. Next, explain the testing phase to be executed in parallel but in conjunction with the previous point.

Finally, and unify all previous concepts will simulate the development of a small program following the methodology developed software and using management tools, documentation and testing required.

Keywords: Scrum, Methodology, Agil, Development, Software.

1. INTRODUCCIÓN

En los últimos años estamos asistiendo a un cambio radical en la forma de gestionar los proyectos de desarrollo de software. Debido al gran vacio en el que se encontraban los desarrolladores de software, las metodologias ágiles aparecieron para poder dar una solución a todos esos proyectos aportando una elevada simplificación sin renunciar a las prácticas esenciales para asegurar la calidad del producto. Dentro de los métodos de ingeniería de software encontramos el Desarrollo Ágil, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupos auto-organizados y multidisciplinarios.

La idea de un Desarrollo Ágil se fundamenta en la reducción de riesgos basándose en la división de un gran problema en otros más pequeños. Actualmente existen muchos tipos de metodologias ágiles que pueden ser utilizadas (Scrum, Feature-Driven Development (FDD), Adaptive Software Development (ASD), Kamban, etc) donde cada una tiene sus características propias y hacen hincapié en algunos aspectos más específicos.

Pero de nada nos sirve una metodología, por muy buena que pueda ser, si perdemos más tiempo gestionando el proceso que programando. Por lo que nos vemos casi obligados a empezar a usar herramientas que nos ayuden con esta tarea. Una de las consideradas como más potentes es Microsoft Team Foundation Server (TFS). Este software es la herramienta que nos ayudará con la gestión del ciclo de vida de aplicaciones. Además, nos aportará una serie de utilidades que nos facilitarán la gestión de los procesos, el control de versiones del código fuente, el testing de aplicaciones, la implementación del código, entre otros.

Asimismo, el uso de forma conjunta de herramientas, como Microsoft Visual Studio, Microsoft Test Manager y Microsoft SharePoint, junto con Team Foundation Server, hacen que el equipo pueda entender mejor lo que el cliente necesita y diseñar, aplicar e implementar el código con más eficacia.

2. METODOLOGIAS ÁGILES

2.1. Ciclo de vida de un proyecto software (SDLC)

2.1.1. Definición

El término ciclo de vida de desarrollo software, en ingles SDLC (Software Development LifeCycle), describe el desarrollo de software desde la fase inicial hasta la fase final. Su propósito no es otro que definir el conjunto de etapas por las que pasa el sistema que se está desarrollando desde que nace la idea hasta que el software es retirado o remplazado.

Las fases por las que transcurre el desarrollo del software son un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Cada fase se construye agrupando tareas (actividades elementales) que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales correspondientes a la asignación de recursos (humanos, financieros o materiales).

El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costes que conllevan.

2.1.2. Procedimientos de un ciclo de vida

Por lo que hace a los procedimientos de los que debe constar un ciclo de vida básico de un software, el orden y la presencia de cada uno de ellos dependerán del tipo de modelo acordado entre el cliente y el equipo de desarrolladores. A continuación podemos ver dichos procedimientos:

- *Definición de objetivos*: definir el resultado del proyecto y su papel en la estrategia global.
- *Análisis de los requisitos y su viabilidad:*
  - User Requirements Sepecification (URS): documento donde se recopilan, examinan y formulan los requisitos del software desde el punto de vista del cliente.
  - Software Requirements Spedifications (SRS): documento basado en el URS y creado por los analistas, los cuales van al lugar del cliente para conocer las actividades del software a desarrollar y examinar cualquier restricción que se pueda aplicar.
- *Diseño general (HLD,High Level Design):* requisitos generales de la arquitectura de la aplicación donde se listan los módulos que contendrá la aplicación y una breve descripción de cada uno (funciones, relación y dependencias con otros modulos).
- *Diseño en detalle (LLD, Low Level Design):* definición precisa de cada subconjunto de la aplicación en donde se muestra: el detalle de la lógica funcional de cada módulo en pseudocódigo, las bases de datos con todos los elementos, detalles de la interfaz y la definición completa de las entradas y salidas de los distintos módulos.
- *Programación (programación e implementación):* es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.
- *Testing Levels:*
  - Prueba de unidad (Unit Testing): testeo por el propio desarrollador de cada subconjunto de la aplicación.
  - Integración (Integration Testing): tras unir todos los módulos, se realiza un testeo conjunto para garantizar que los diferentes módulos se integran entre ellos y los datos de unos a otros funcionan correctamente.
  - Prueba beta (System Testing): se testean todos los módulos, como si de una sola aplicación se tratara.
  - Aceptación (Acceptance Testing): el cliente final es el encargado de realizar el testeo de la aplicación en un uso real o simulado.
- *Documentación:* sirve para documentar información necesaria para los usuarios del software y para desarrollos futuros.

- *Release:* tras pasar el periodo de pruebas de testing, el cliente ya tiene en sus manos una versión del software para que sea probado en la vida real, con servidores reales.
- *Mantenimiento:* esta fase incluye los procedimientos correctivos (mantenimiento correctivo), las actualizaciones secundarias del software (mantenimiento continuo), mejoras del software y reparación de bugs (Bug Fixing).

2.2. Modelos de desarrollo de software

Los distintos modelos que encontramos permiten suministrar una guia para los ingenieros de software con el fin de:

- Ordenar las diversas actividades técnicas en el proyecto.
- Suministrar un marco para la administración del desarrollo y el mantenimiento (estimación de recursos, definición de puntos de control intermedios, monitorización del avance, etc).

2.2.1. Modelo en cascada

Este modelo ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la anterior.

La visión del modelo cascada del desarrollo de software es a través de una secuencia simple de fases. Cada fase tiene un conjunto de metas bien definidas, y las actividades dentro de una fase contribuyen a la satisfacción de metas de esa fase.

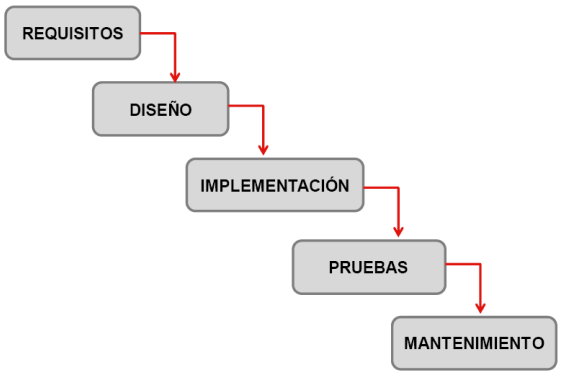

Figura 1: Esquema del modelo en cascada.

Las ventajas de este modelo son:

- Funciona bien para proyectos pequeños donde los requisitos están bien entendidos.
- Es simple y fácil de usar.
- Fácil de gestionar ya que cada fase tiene entregables específicos.

En cambio, sus principales inconvenientes son:

- En la vida real los proyectos no siguen una secuencia lineal.
- Es muy restrictivo y no permite movilizarse entre fases.
- El producto se ve cuando ya está finalizado.

2.2.2. Modelo V

Es un modelo que ilustra cómo las actividades de prueba (verificación y validación) se pueden integrar en cada fase del ciclo de vida. Dentro del modelo en V, las pruebas de validación tienen lugar especialmente durante las etapas tempranas, por ejemplo, revisando los requisitos de usuario durante las pruebas de aceptación de usuario.

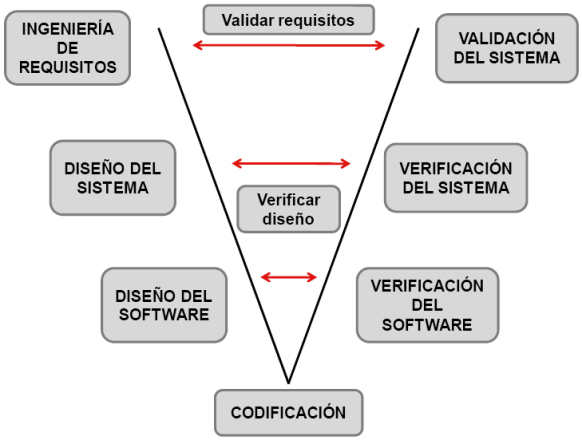

Figura 2: Esquema del modelo en V.

El lado izquierdo de la V representa la descomposición de las necesidades y la creación de las especificaciones del sistema. El lado derecho de la V representa la integración de las piezas y su verificación.

Las ventajas de este modelo son:

- Modelo simple y fácil de utilizar.
- En cada una de las fases hay entregables específicos.
- Funciona bien para proyectos pequeños con requisitos entendidos fácilmente.

En contra, sus principales inconvenientes son:

- Modelo muy rígido, con poca flexibilidad.
- No se producen prototipos del software.
- No tiene caminos claros para problemas encontrados durante las fases de pruebas.

2.2.3. Modelo iterativo

Este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos.

Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada una se le entrega al cliente una versión mejorada. El cliente será quien después de cada iteración evalúe el producto y proponga mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga las necesidades del cliente.

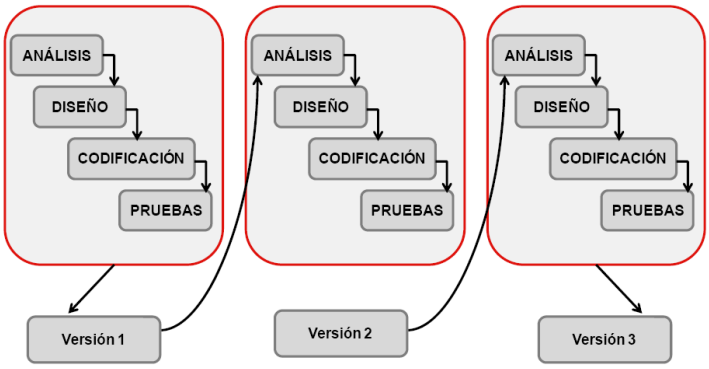

Figura 3: Esquema del modelo iterativo.

Las ventajas de este modelo son:

- No hace falta que los requisitos estén totalmente definidos al inicio del desarrollo.
- Desarrollo en pequeños ciclos lo que permite gestionar mejor los riesgos.
- Al usuario se le entrega parte del producto (versiones) para que lo pruebe.

En contra, su principal inconveniente es que el proceso de desarrollo es largo ya que en cada interación se actualizan las especificaciones.

2.2.4. Modelo de desarrollo iterativo e incremental

Este modelo se basa en construir incrementando las funcionalidades del programa a desarrollar. Éste aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario.

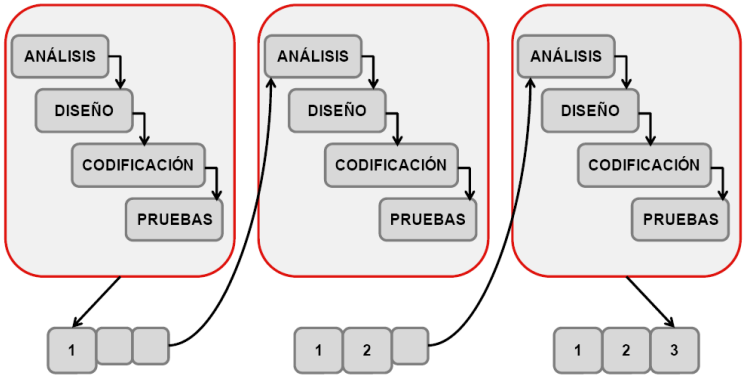

Figura 4: Esquema del modelo de desarrollo iterativo e incremental.

Este modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

Las ventajas de este modelo son:

- Es flexible, generando software operativo de forma rápida y en etapas tempranas.
- Es más fácil probar y depurar en una iteración más pequeña.

En contra, sus principales inconvenientes son:

- Se requiere de una experiencia importante para definir los incrementos y distribuir en ellos las tareas de forma proporcionada.
- Cada fase de una iteración es rígida y no se superponen con otras.

2.3. Metodologías de desarrollo de software

2.3.1. Definición

Una metodología es un conjunto de técnicas y métodos que permite abordar, de forma homogénea y abierta, cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Además, es un proceso de software detallado y completo.

Las metodologías se basan en una combinación de los modelos de proceso de desarrollo genéricos (cascada, incremental…) mediante las cuales se definen roles, actividades, prácticas y técnicas recomendadas. Asimismo, nos permiten optimizar el proceso y producto, hacer una guía en la planificación y en el desarrollo del software, y por último, definir qué hacer, cómo y cuándo durante el desarrollo y mantenimiento del proyecto.

Los elementos más importantes que forman parte de una metodología son:

- *Fases:* tareas a realizar en cada fase.
- *Productos:* Entradas y salidas de cada fase, documentos.
- *Procedimientos y herramientas:* apoyo a la realización de cada tarea.
- *Criterios de evaluación del proceso y del producto:* saber si se logran los objetivos.

Actualmente, podemos encontrar dos grupos de metodologías según la filosofía de desarrollo:

- *Metodologías tradicionales:* hacen énfasis en la planificación.
- *Metodologías ágiles:* se basan en la adaptabilidad del proceso y un desarrollo incremental, cooperativo y sencillo.

La filosofía de las metodologías ágiles es centrarse en el factor humano o el producto software, dando mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software, con iteraciones muy cortas, manteniendo una alta calidad.

2.3.2. Ventajas de uso de metodologías

Existen numerosas ventajas del uso de las metodologías en el campo del desarrollo de software, las cuales se pueden ver desde distinto puntos de vista:

- Desde el punto de vista de gestión:
  - ✓ Facilitar la tarea de planificación, control y seguimiento del proyecto.
  - ✓ Mejorar la relación coste/beneficio.
  - ✓ Optimizar el uso de recursos disponibles.
  - ✓ Facilitar la comunicación efectiva entre usuarios y desarrolladores.
- Desde el punto de vista de los ingenieros del software:

- ✓ Ayudar a la comprensión del problema.
- ✓ Optimizar el conjunto y cada fase del proceso de desarrollo.

➢ Desde el punto de vista del cliente o usuario:

- ✓ Garantizar un determinado nivel de calidad en el producto final.
- ✓ Dar confianza en los plazos de tiempo fijados en la definición del proyecto.

2.3.3. *Metodología tradicional*

Son denominadas como metodologías pesadas. Éstas centran su atención en llevar una focalización de la documentación, planificación y los procesos de forma exhaustiva de todo el proyecto. Además, se centran en cumplir con un plan de proyecto definido en la fase inicial del desarrollo del proyecto.

La ventaja de este modelo radica en el perfeccionamiento de dar respuesta a los requerimientos inicialmente analizados. Por el contrario, las desventajas principales son los altos costes al implementar un cambio y la falta de flexibilidad en proyectos.

2.3.4. *Metodología Ágil*

Sus principios y valores fueron formalizados en el manifiesto para el desarrollo de software ágil (Manifiesto Ágil). El enfoque de este método es la planificación adaptativa (la habilidad de responder al cambio), permitiendo potenciar aún más el desarrollo de software a gran escala.

Existen numerosos Métodos Ágiles, como por ejemplo: Adaptative Software Development (ASD) creado en el año 2000, Agile Modeling (AM) creado en el 2002, Extreme Programming (XP) del 1999, Feature-Driven Development (FDD) en el 1998 y Scrum del año 1994.

2.3.5. *Comparativa entre metodologías*

Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. Además, es importante tener en cuenta que el uso de un método ágil no vale para cualquier proyecto.

A continuación se detallan algunas de las principales diferencias entre ambas metodologías:

METODOLOGÍAS TRADICIONALES	METODOLOGÍAS ÁGILES
Dificil realizar cambios durante el proyecto.	Preparados para cambios durante el proyecto.
Existe un contrato prefijado.	Es bastante flexible.
El cliente es parte del equipo de desarrollo mediante reuniones.	El cliente interactúa con el equipo de desarrollo.
Grupos grandes y distribuidos.	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.
Más roles.	Pocos roles.

Tabla 1: Comparativa metodología tradicional y ágil

2.4. Desarrollo Ágil

2.4.1. Descripción

El desarrollo de software ágil son métodos adaptativos de ingeniería del software basados en el desarrollo iterativo e incremental que, entre otras, promueven las siguientes particularidades:

- Una filosofía fomentando trabajo en equipo, organización propia y responsabilidad.
- Proceso de gestión promoviendo la inspección y adaptación a los cambios.
- Permite la entrega rápida de software de alta calidad.

Una de las carácterísticas más importantes del desarrollo ágil es el uso de iteraciones a lo largo de proceso de creación del software. Las iteraciones son estructuras de tiempo pequeñas con una duración típica de 1 a 4 semanas que contienen algunas peculiaridades como: permitir conseguir una adaptación rápida del proyecto a los cambios; hacer las cosas en incrementos pequeños con una planificación a corto plazo y obtener versiones disponibles (con errores mínimos) al final de cada iteración.

El termino equipo (equipo ágil) y su composición es normalmente de organización propia y multidisciplinar, es decir, el conjunto de personas tienen diferentes formaciones académicas y experiencias profesionales, que operan en conjunto durante un tiempo determinado. Éstos son abocados para dar solución a un problema complejo que será su objetivo común, pero serán cada uno de ellos los que decidan como realizar las tareas durante una iteración.

En cuanto al tamaño del equipo es normalmente pequeño (5-9 personas) para ayudar a hacer la comunicación y la colaboración más fácil. Dicha relación destaca porque incluyen una comunicación de cara a cara rutinaria, diaria y formal entre sus miembros sobre los documentos escritos (por ejemplo, las especificaciones del software). Además, la mayoría de los equipos ágiles se encuentran localizados en una única ubicación abierta para facilitar esta comunicación.

Igualmente, y no menos importante, será el cargo de representante del cliente que tendrá que contener cada equipo ágil. Al final de cada iteración, el representante del cliente y las personas involucradas en el negocio revisarán el progreso y reevaluarán las prioridades con vistas a optimizar el retorno de la inversión para así asegurar las necesidades del cliente y los objetivos de la compañía.

2.4.2. El manifiesto Ágil

En marzo de 2001, 17 críticos de los modelos de mejora basados en procesos de desarrollo se reunieron en Salt Lake City (capital del estado de Utah, Estados Unidos) para discutir sobre el desarrollo de software. En la reunión se acuñó el término “Métodos Ágiles” y se creó “The Agile Alliance”, una organización que resumió en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que son las bases sobre las que se asientan estos métodos.

Finalmente, se desarrollaron los 12 principios que actualmente son considerados los definitorios de la agilidad, sin importar la metodología o marco del que se trata. Estos son:

I. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
II. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
III. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
IV. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
V. Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
VI. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
VII. El software que funciona es la principal medida del progreso.
VIII. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
IX. La atención continua a la excelencia técnica enaltece la agilidad.
X. La simplicidad como arte de maximizar la cantidad de trabajo que se hace, es esencial.
XI. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.
XII. En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

2.4.3. El método Scrum

2.4.3.1. Descripción

Es una metodología para afrontar los proyectos de creación de aplicaciones de forma iterativa, incremental, rápida y eficaz. Su característica básica es trabajar con equipos pequeños multidisciplinares, formados por un número de personas que suele ir de dos a nueve. En cuanto a las ventajas que proporciona este método es que és muy fácil de entender y requiere poco esfuerzo para comenzar a usarse.

Figura 5: Componentes del equipo Scrum.

Scrum es un esqueleto de proceso que incluye un conjunto de prácticas y roles predefinidos:

- *Product Owner (Propietario del Producto):* Es el cliente y el dueño del producto.

- *SCRUM Master (Arquitecto SCRUM):* guiará el proceso de desarrollo según los principios de la metodología. En ocasiones, esta misma persona pertenece a la vez al SCRUM Team.
- *SCRUM Team (Equipo SCRUM):* personas que forman el equipo de desarrollo.
- *Usuario final:* será el que utilizará en la vida real el producto a desarrollar.

En otro orden de cosas, Scrum se diferencia del resto de metodologias por hacer uso de las iteraciones (tambien llamados Sprints). En la siguiente figura podemos ver el proceso llevado a cabo en cada Sprint:

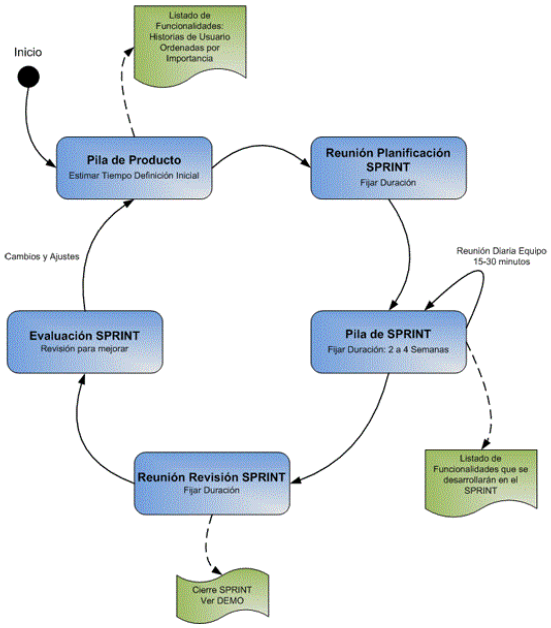

Figura 6: Esquema del proceso de un Sprint en Scrum.

El proceso se inicia con la elaboración de un listado de funcionalidades ordenadas por importancia, conocido como el Product Backlog (la Pila de Producto), compuesto por las Historias de Usuario o Casos de Uso (breves descripciones textuales de cada una de las funcionalidades que tendrá el producto).

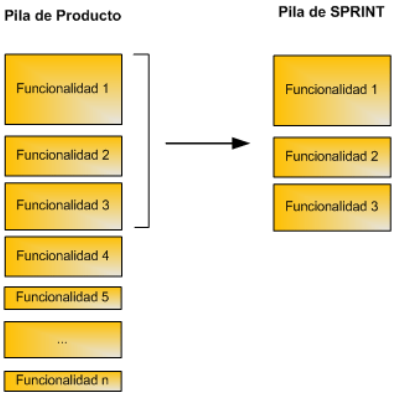

Figura 7: Pilas del producto y del Sprint en Scrum.

El conjunto de características que entra en un Sprint viene del Product Backlog (Pila de Producto), que son un conjunto priorizado de requisitos de trabajo de alto nivel que se han de

hacer. Con ello entramos en el ciclo de desarrollo que se repetirá en varias ocasiones a lo largo de todo el proyecto. Durante cada Sprint (cuya longitud es decidida por el equipo, de 2 a 4 semanas), el equipo crea un incremento de software operativo.

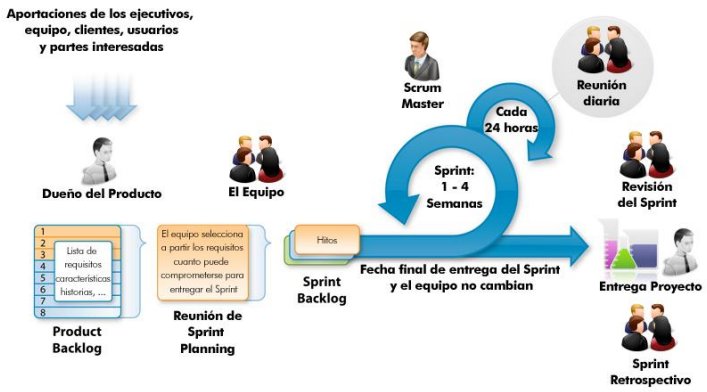

Figura 8: Esquema del proceso Scrum.

Los ítems, denominados PBI (Product Backlog Item) que entran en el Sprint se determinan durante la reunión de planificación. Durante esta reunión, el Product Owner informa al equipo de los ítems en el backlog del producto que quiere que se completen. El equipo determina entonces a cuanto de eso puede comprometerse a completar durante el siguiente Sprint. Durante un Sprint, nadie puede cambiar el backlog de la iteración.

Una vez concluye el plazo fijado para el Sprint, el equipo muestra lo que se ha hecho en el software con una demo en la reunión de revisión del Sprint.

Finalmente, se llevarán a cabo la reunión de Retrospectiva del Sprint, en la que se evaluará si se ha de mejorar algo o si se han de añadir nuevas historias a la Pila de Producto, tales como bugs, nuevas funcionalidades descubiertas, etc. De esta manera se finaliza el Sprint actual y se comienza uno nuevo, así hasta acabar con todos los Backlogs de la Pila de Producto que se tienen.

Por lo que hace a las reuniones de Scrum encontramos las siguientes:

- *Sprint Planning Meeting (Reunión de planificación del Sprint):* se lleva a cabo al principio del ciclo del Sprint y es donde se seleccionará el trabajo que se hará. En esta reunión se preparará el Sprint Backlog.
- *Daily Scrum:* es la reunión que cada día se realiza, durante la iteración, para hablar del estado del proyecto. La reunión tiene una duración fija de 15 minutos, de forma independiente del tamaño del equipo, y debe ocurrir en la misma ubicación y a la misma hora todos los días. Asimismo, cada miembro del equipo debe contestar a tres preguntas: ¿Qué has hecho desde ayer?, ¿Qué es lo que harás hasta la reunión de mañana? y ¿Has tenido algún problema que te haya impedido alcanzar tu objetivo?
- *Scrum de Scrum:* esta reunión se realiza después de la Daily Scrum cuando hay varios grupos de equipos y están formadas por una persona asignada por cada equipo.

- *Sprint Review Meeting (Reunión de revisión del Sprint):* se realiza al final del ciclo del Sprint y se hace una revisión del trabajo que fue completado y no completado. También se presentará el trabajo completado a los interesados en una demo del software.
- *Sprint Retrospective (Retrospectiva):* se realiza al final del ciclo de la iteración en la cual todos los miembros del equipo dejan sus impresiones sobre el sprint recién superado, con el propósito de la realizar una mejora continua del proceso.

Por otro lado, existen diferentes formatos en los que se puede usar y trabajar con Scrum:

- *Hoja de cálculo:*

SPRINT	INICIO	DURACIÓN
1	1-feb-06	12

					X
					1-feb
				Tareas pendientes	23
				Horas de trabajo pendientes	276
PILA DEL SPRINT					
Backlog	Tarea	Tipo	Estad	Responsal	
1	Descripción de la tarea 1	Análisis	Terminada	Luis	16
1	Descripción de la tarea 2	Prototipado	Terminada	Luis	12
1	Descripción de la tarea 3	Pruebas	Terminada	Luis	4
1	Descripción de la tarea 4	Codificación	Terminada	Elena	8

Figura 9: Ejemplo de uso de hoja de cálculo para un Sprint en Scrum.

- *Pizarra o pared física:*

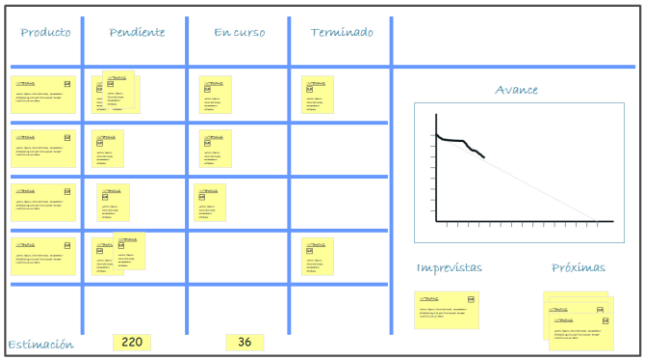

Figura 10: Ejemplo de uso de pizarra para un Sprint en Scrum.

- *Herramienta colaborativa o de gestión de proyectos: Team Foundation Server.*

Figura 11: Ejemplo de uso de TFS para un Sprint en Scrum.

Asimismo, a la hora de elegir uno u otro formato deberemos de tener en cuenta, además de elegir el más cómodo para todos, los siguientes criterios:

- Que incluya la información necesaria: lista de tareas, persona responsable de cada una, estados y tiempo de trabajo que queda para completarla.
- Que facilite la consulta y comunicación diaria y directa del equipo.

3. TESTING

3.1. Introducción

Las pruebas de Testing de software son los procesos que permiten verificar y revelar la calidad de un producto software identificando posibles fallos de implementación o usabilidad ¿Porqué el software usualmente no funciona correctamente? Debido a los errores de las personas. Si un usuario comete un error durante el uso de por ejemplo, una aplicación, esto puede desembocar en un problema, ya que la aplicación será utilizada de un modo no esperado y no tendrá el comportamiento deseado. Estos errores son los denominados defectos o bugs.

El Testing se ha ido introduciendo poco a poco en el proceso de vida del desarrollo gracias a los beneficios que aporta:

- Prevención de posibles daños en la producción, ya que los errores encontrados durante el Testing son marcados como “errores conocidos”.
- Ganar confianza en el producto.
- Facilitar la gestión del proyecto, aportando información sobre el progreso y la calidad.

3.2. Conceptos generales

3.2.1. Error, defecto y fallo (error, fails and failures)

Las definiciones según el estándar IEEE 610.12-1990 (Glosario de términos de ingeniería de software) son:

- Un *error* es una acción humana que provoca un resultado no esperado.
- Un *defecto o fail* es una imperfección, o deficiencia de un componente, que puede provocar que el sistema no se comporte según lo esperado.
- Una *fallo o failure* es un resultado observado no esperado.

Figura 12: Ejemplo de errores, defectos y fallos.

Como podemos ver en la anterior figura, un error humano puede generar un defecto interno en un programa, lo cual puede producir un fail externo del programa, provocando graves daños.

Quienes participan en el desarrollo del software, introducen los defectos. Pero los errores que se cometen pueden estar relacionados con el código fuente, la carencia e imprecisiones en las especificaciones o en el proceso de desarrollo.

3.2.2. Axioma del Testing

"La prueba de un programa sólo puede mostrar la presencia de defectos, no su ausencia."

Edsger Wybe Dijkstra

Cuando testeamos sólo conocemos los resultados que obtenemos despues de ejecutar cada prueba. Tal vez ejecutamos 10 pruebas y obtenemos 10 fallos, o ninguno, o 2, pero en ningún caso sabemos el estado interno del software y la cantidad de defectos que contiene.

Para obtener resultados valiosos de las pruebas, tenemos que seleccionar cuidadosamente las pruebas, diseñándolas para encontrar la mayor cantidad de fallos severos con los recursos y en el plazo disponible.

3.2.3. Paradoja del Testing

"Cualquier método que se use para prevenir o encontrar bugs deja como residuo los más sutiles, contra los que ese método no es efectivo."

Boris Beizer

Poniendo como ejemplo cercano el uso de un pesticida, éste mata a un gran porcentaje de los insectos, pero deja un residuo y los más fuertes sobreviven. Como consecuencia, al año siguiente es necesario hacer uso de un pesticida más fuerte.

Cuando hacemos testing pensamos pruebas, las ejecutamos, detectamos bugs, se corrigen y volvemos a ejecutar pruebas para verificar su corrección. Despues, el conjunto de casos de prueba pensados, ya no detectan bugs, pues los que detectaron en un principio ya fueron corregidos. Pero aún así pueden haber bugs en el software que no hayamos detectado (residuos), es por ello que tenemos que pensar y ejecutar nuevas pruebas para detectar esos bugs que aún persisten. Si se hace un buen testing, los errores van a ir cambiando y tenemos que cambiar los casos de prueba para mantener la efectividad.

3.3. Tipos de pruebas de Test

3.3.1. Clasificación

El proceso de testing está divido en diferentes tipos de pruebas para reflejar qué clase de defectos tienen cobertura con los distintos tipos de test. El testing puede ser clasificado desde diferentes puntos de vista:

Clasificación según conocimientos del sistema

- *Test de Caja Negra (Black Box):* el software es tratado como un sistema cerrado del que se desconoce cómo está desarrollado. La única información que tiene el tester es el documento de requerimientos y la funcionalidad del SW.

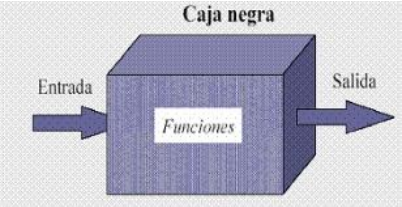

Figura 13: Esquema de test de Caja Negra.

- *Test de Caja Blanca (White Box):* está basado en el conocimiento del código interno del software. Los tests se basan en la cobertura de las condiciones y declaraciones del código.

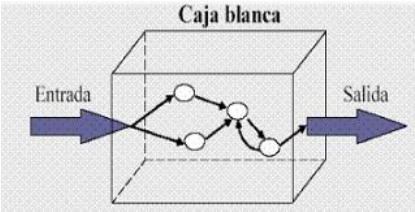

Figura 14: Esquema de test de Caja Blanca.

Clasificación según en qué momento del proceso es ejecutado el test

- *Test Unitario:* método para asegurar que los pequeños fragmentos de código funcionan correctamente aislados. Este tipo de test es ejecutado tanto por los desarrolladores como por el equipo de calidad.
- *Test de Integración:* el objetivo es asegurar una correcta funcionalidad con la interacción de dos o más componentes del software. El criterio de entrada para el test de integración es completar el test unitario (testear primero los componentes por separado).
- *Test de Sistema:* se lanza con todo el software integrado y en una réplica del entorno de producción. El test de sistema prepara al equipo para un tipo de test centrado desde el punto de vista del usuario (Test de Aceptación de Usuario).
- *Test de Aceptación de Usuario*: este tipo de test determina si el software está listo para su uso. El test deberá ser ejecutado desde el punto de vista de un usuario final. Se comenzará a ejecutar una vez finalizado el Test de Sistema.

Clasificación según su objetivo:

- *Testing funcional:* describe lo que hace el sistema. Es un proceso de control de calidad (QA) de tipo Caja Negra que basa sus casos de prueba en las

especificaciones del componente de software bajo prueba. El Testing funcional típicamente involucra cinco fases:

1) La identificación de las funciones que se espera que el software realice.
2) La creación de los datos de entrada basándose en las especificaciones.
3) La determinación del resultado basado en las especificaciones.
4) La ejecución de los casos de prueba.
5) La comparación entre los resultados reales y esperados.

Dentro del test funcional se ejecuntan los siguientes tipos de test:

- ✓ *Test de instalación:* comprobar la instalación correcta del software.
- ✓ *Test de regresión:* para asegurar que los cambios, debidos a nuevos Bugs solventados o mejoras en el software, no provoquen daños colaterales en funcionalidades ya existentes. En cuanto a el "cuando" debe ser ejecutado, siempre debe ser despues del Sanity/Smoke Test y se deberá de repetir con cada nueva release del software.
- ✓ *Test de actualización:* comprueba que las mejoras añadidas en nuevas versiones del software se realizan correctamente.
- ✓ *Test de accesibilidad:* se refiere a la capacidad de acceso a los contenidos del software por todas las personas independientemente de la discapacidad que presenten (visuales, motrices, auditivas, cognitivas) o de las que se deriven del contexto de uso (tecnológicas o ambientales).
- ✓ *Test de localización*: el objetivo es comprobar que el software se adapta a distintos lugares donde será instalado.

- *Testing no funcional:* se centra en el comportamiento y la experiencia del usuario con el software. Generalmente se lleva a cabo durante la fase de Test de Sistema. A continuación se muestran los test correspondientes que son ejecutados:
  - ✓ *Sanity/Smoke Test:* permite saber si el software está en un estado en el que ya se puede empezar el ciclo de test. Usualmente, el sanity test contiene casos de prueba muy básicos.
  - ✓ *Test de stress:* se utiliza para romper la aplicación, ya que determina su solidez en los momentos de carga extrema y ayuda a determinar su rendimiento.
  - ✓ *Test de carga:* se realiza para observar el comportamiento de la aplicación bajo una cantidad de peticiones esperada, como por ejemplo un numero enorme de usuarios (cuellos de botella).

- ✓ *Test de rendimiento:* es útil para comprobar que las funcionalidades siguen intactas bajo situaciones que podríamos considerar de estrés (reacción con un número elevado simultáneo de usuarios).
- ✓ *Test de usabilidad:* consisten en seleccionar a un grupo de usuarios de una aplicación y solicitarles que lleven a cabo las tareas para las cuales fue diseñada. Mientras, el equipo de diseño, desarrollo y otros involucrados toman nota de la interacción.
- ✓ *Test de seguridad:* el objetivo principal es identificar los posibles puntos vulnerables y eliminarlos.

Clasificación según la forma en que se ejecutan los tests:

- *Analisis estático – Verificación:* revisión por parte del tester del producto.
- *Analisis dinámico – Validación:* Es la actividad más frecuente que un tester va a llevar a cabo, independientemente de si está ejecutando caja negra, funcional, no funcional, etc. La validación está asociada a la ejecución en sí, es por ello que los test que se pueden llevar a cabo son: *Tests manuales, exploratorios, automatizados y basados en modelos.*

3.4. Testing vs Debugging

Existen dos terminos en el mundo del Testing que deben de diferenciarse, el proceso de Testing y el Debugging (depuración de programas).

TESTING	DEBUGGING
Focalizado en encontrar y localizar un Bug.	Focalizado en arreglar un Bug.
Realizado por el equipo de Testing.	Realizado por el equipo de desarrolladores.
Su intención es encontrar el máximo número de defectos posibles.	Su intención es eliminar todos los defectos encontrados y modificar parte del código que lo origina.
No es necesario conocer el código del software.	Es necesario conocer el código del software.

Tabla 2: Comparativa concepto Testing y Debugging.

3.5. El proceso de Testing

El Testing conlleva una serie de pasos básicos que a continuación se detallan:

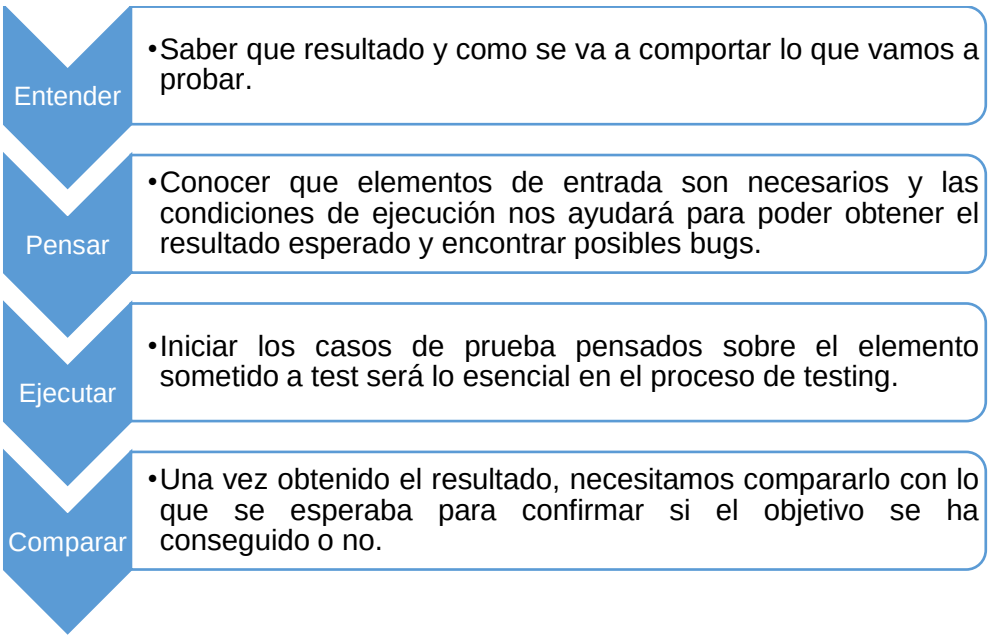

Figura 15: Proceso básico del Testing.

3.6. Ciclo de vida del test de software

3.6.1. Preparación

Es el primer punto del ciclo de vida y consta de las siguientes fases:

- Revisión de la documentación.
- Determinar las técnicas de especificación de casos de prueba a utilizar.
- Especificar la infraestructura necesaria para las pruebas.

En ella deberemos de crear el plan de pruebas y las bases de los test que serán ejecutados. El plan de pruebas principal será el *Master Test Plan (MTP*) el cual recogerá todos los test creados a lo largo del desarrollo del software. Es muy importante que todos los casos de pruebas creados y ejecutados en el software se documenten y archiven para poder llevar un control. El encargado de realizarlo es el Test Manager (director de pruebas), que en consulta con el cliente y otras partes interesadas, hace una descripción total de la distribución a través de los niveles de prueba en cuanto a lo que debe ser probado, cuándo y con qué rigor.

El siguiente plan de pruebas a crear será el *Detailed Test Plan (DPT)*, el cual contendrá los escenarios de test conocidos como “condiciones de test” (Test Conditions) donde se responderá a la pregunta: ¿Qué vamos a testear?. Y como respuesta, tendremos que escribir en detalle el plan de test sobre la configuración necesaria del software, los requerimientos de funcionalidad y el entorno en el que se va a usar.

Por último, crearemos los *Detailed Test Cases (DTC),* que serán los propios Testers los que deberán escribirlos con detalle para evitar fallar a la hora de obtener el resultado esperado. Para ello, la pregunta que deberemos contestar será: ¿Cómo vamos a testear?

3.6.2. Especificación de las pruebas

Un Caso de prueba (Test Case) es una lista de comprobación (unas pautas paso a paso fáciles de entender) de lo que debe y no debe hacer el sistema. Lo ideal es que los casos de prueba estén basados en los requisitos especificados por el cliente. Estos requisitos o especificaciones son escritos en un documento que se le suele conocer como “Documento de Requisitos Funcionales” o “Especificaciones Funcionales”.

Por lo que hace a la manera de escribir un caso de prueba, existen muchas maneras de hacerlo. Todas ellas pueden ser perfectamente válidas pero debemos de tener en cuenta que se deberá de definir los pasos a seguir y el resultado esperado, así como comprobar si hace o no lo que debe (casos positivos y negativos) tanto en entradas válidas como en inválidas.

3.6.2.1. Partes de la especificación de pruebas

- *Historia o Revisión:* Se indicará quien creó la especificación de los test, encargados de documentación, fecha de creación y actualización.
- *Descripción de las características:* descripción del área que va a ser probada.
- *¿Qué se va a probar?:* mínima descripción general de los escenarios a probar.
- *¿Qué no se va a cubrir?:* zonas que serán cubiertas por otras especificaciones.
- *Casos de prueba nocturnos:* pruebas que serán ejecutadas cada noche.
- *Propagación de las áreas de pruebas:* los testers organizan las pruebas según lo que están probando (pruebas de funcionalidad específica, tests de seguridad, test de accesibilidad, test de stress, test de rendimiento…).

3.6.2.2. Prioridades de los Casos de Prueba (Test Cases)

Es necesario priorizar los test para ordenar su ejecución según esta clasificación. Se puede intuir claramente que los casos de prueba más prioritarios, normalmente, serán los que están relacionados con las partes más importantes o críticas y las susceptibles a tener más defectos.

La clasificación es: Prioridad Alta (High), Prioridad media (Medium) y Prioridad baja (Low).

3.6.3. Ejecución

En esta fase encontramos las tareas que conciernen a la ejecución de los casos de prueba según el plan y las especificaciones. Asimismo, se deberá de registrar el resultado de la prueba y los bugs encontrados, actualizando los tests en caso de que se actualicen las especificaciones.

3.6.4. Finalización

En esta etapa es donde se deberá de realizar el informe final que contendrá: el resumen de actividades, el análisis de los resultados, el análisis de los Bugs encontrados, la evaluación del producto y del proyecto de pruebas.

Además, deberemos de consolidar y guardar en el Master Test Plan todos los casos de prueba que hemos ejecutado.

3.6.5. Control

En esta fase es donde se realizará el seguimiento del progreso del proyecto. Asimismo, deberemos de identificar los problemas y tomar las acciones preventivas o correctivas necesarias. Todo ello quedará plasmado en el informe de progreso, que deberá de contener el estado de las actividades previstas para la semana (pruebas ejecutadas contra las planificadas) y la planificación de las actividades de la siguiente semana. Igualmente, se indicarán el número de incidencias encontradas y los riesgos y acciones preventivas y correctivas.

3.6.6. Tabla resumen del ciclo de vida

A continuación se muestra las entradas y salidas que contienen cada una de las fases del ciclo de vida del test de software, en la que podremos ver la relación existente entre ellas.

	ENTRADAS	SALIDAS
PREPARACIÓN	La base de test	La actualización del Plan de Pruebas
	El Plan de Pruebas	El catálogo de situaciones de prueba y asignación de técnicas de especificación.
ESPECIFICACION DE LAS PRUEBAS	Especificaciones	Casos de pruebas
EJECUCION	Plan de pruebas y de ejecución	Resultados de pruebas
	Especificaciones de los casos de prueba	Reporte de Bugs
	Entorno de pruebas preparado	Actualización de casos de prueba
FINALIZACION	Test Plan	Informe final
CONTROL	Test Plan	Informe de progreso

Tabla 3: Resumen del ciclo de vida del Testing.

3.7. Gestión de Bugs

3.7.1. Ciclo de vida de un Bug

En el momento que se detecta un Bug del software se tiene que realizar el reporte del error encontrado para que realicen su corrección. A partir de este momento el Bug inicia un proceso de seguimiento hasta que es corregido y cumple con los requerimientos y expectativas del cliente.

Existen unos estados principales por los que un Bug debe pasar:

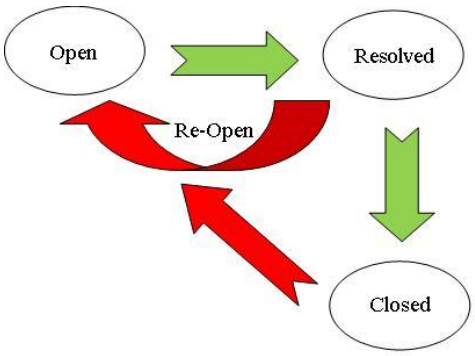

Figura 16: Estados principales del ciclo de vida de un Bug.

- *Open:* cuando se encuentra por primera vez un Bug y se abre su seguimiento.
- *Resolved:* cuando el desarrollador corrige o implementa la tarea del Bug que se le fue asignada, la tarea entra en estado de resolved y puede ser por los siguientes puntos que llega a ese estado: arreglado, resuelto por diseño, no reproducible o no arreglado.
- *Closed:* la tarea entra en estado cerrado después de que sea revisada por el responsable de la calidad del software.
- *Re-open:* en el caso de que la tarea del Bug no esté funcionando correctamente, el estado vuelve a reabrirse y comienza el mismo proceso hasta llegar a cerrarlo.

En la siguiente figura podemos encontrar un ejemplo real y más completo:

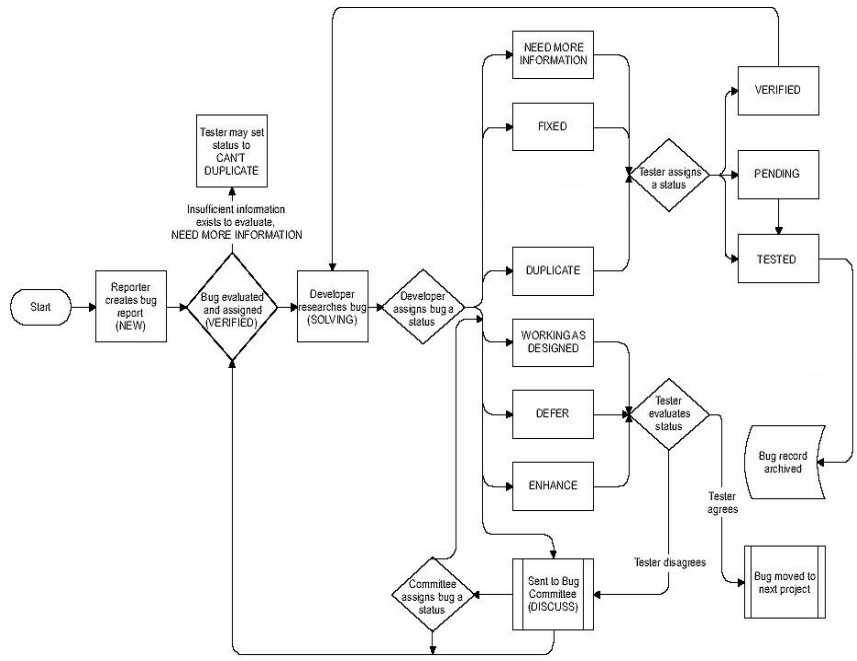

Figura 17: Proceso completo de un Bug.

1) *NEW (Nuevo):* el Tester encuentra un nuevo Bug.
2) *VERIFIED (Verificado y Asignado):* se verifica la existencia del Bug y se le asigna a un desarrollador. En el caso de que falte información, se pasará al estado “Need

more information" donde la persona que ha encontrado el Bug deberá de proporcionar más datos.

3) *SOLVING:* el desarrollador asignado comienza a buscar una solución al Bug. Éste puede asignar al Bug uno de los siguientes estados según aplique:
   a) *Need more information:* se necesita más información por parte del desarrollador para continuar investigando el defecto. En este caso el propietario del Bug pasa a ser el Tester, el cual deberá buscar la información que el desarrollador necesita y volver a poner el estado del Bug en "Asignado".
   b) *Fixed (Arreglado):* el desarrollador ha resuelto el problema. El Tester deberá intentar reproducir el Bug con la release de software que el desarrollador haya indicado, donde estará resuelto el Bug. Dependiendo del resultado, el estado del Bug puede pasar a "Verificado" o "Reabierto".
   c) *Duplicate:* el Bug está duplicado ya que se ha encontrado en anteriores ocasiones y no es nuevo. Si el Tester está en desacuerdo y cree que el defecto reportado puede tener alguna diferencia con el que ya existía puede pasar el estado a "Reopened".
   d) *Rejected o Working as designed (Rechazado):* si el desarrollador cree que el Bug reportado no es un defecto de acuerdo a los requerimientos, su estado pasará a "Rejected". Si el Tester no está de acuerdo con la decisión del desarrollador, y cree que realmente existe un defecto, puede poner el Bug en el estado "Reopened".
   e) *Deferred (Pospuesto):* el equipo de desarrollo considera que el Bug será revisado en las siguientes releases. Los motivos para asignar este estado al Bug pueden ser una baja severidad o que sea poco reproducible.
   f) *Enhance (Improve):* es cuando los Bugs no se consideran defectos, sino posibles mejoras.
4) El Tester será, de nuevo, el que revisará el Bug. Los posibles estados serán:
   a) *Verified:* verificación de que el Bug ha desaparecido en la release indicada.
   b) *Pending:* el Bug todavía está pendiente de testear por el Tester.
   c) *Tested:* el Bug ha sido testeado correctamente (pasa a ser archivado).
5) *Re-Open (Reabierto):* si el Bug está en este estado, debe volver a atravesar todo el flujo de nuevo.

3.7.2. Informes

Para el correcto flujo entre desarrolladores y testers, es muy importante reportar correctamente los defectos que se detectan. Estos reportes dependen de la herramienta que se esté utilizando, pero hay que seguir una determinada línea de trabajo para evitar la pérdida de información o la confusión entre el equipo.

Previo al reporte del bug, el Tester deberá de asegurarse que realmente es un defecto nuevo y que está usando la última versión del software bajo testeo.

Si finalmente se reporta el bug, se deberán detallar los puntos importantes a la hora de reportar un defecto: pasos a reproducir, causa del defecto, describir el entorno donde ha sido encontrado, título del Bug claro y preciso, proporcionar datos de contacto y asignarle la correcta severidad.

3.7.3. Revisión

Debido al gran número de Bugs que se pueden encontrar en un producto de software, es necesaria la creación de un equipo para la revisión de defectos. Este equipo es el llamado el DRT (Defects Review Team), el cual es esencial en cualquier equipo de Testing.

El DRT proporciona una rápida y eficiente respuesta a un defecto en cualquiera de nuestros entornos y garantiza una oportuna solución. El equipo estará integrado por los siguientes miembros:

- Responsable del equipo de desarrollo (Development Team Leader).
- Jefe de proyecto (Project Manager).
- Responsable del equipo de pruebas (QA Team Lead).
- Cliente (Opcional). Es importante que el cliente forme parte de este equipo.

En cuanto al plan de acción frente a Bugs que debe de tener el DRT, principalmente tiene que establecer como norma, una reunión semanal y después, cada miembro del equipo se debe comprometer a revisar diariamente los bugs. De este modo, no solo mantendremos un control de los Bugs que han aparecido y como han sido tratados, sino que además intentaremos evitar que vuelvan a aparecer (aunque realmente esto nunca se podrá asegurar).

3.8. Métricas en el entorno de pruebas

Las métricas son estándares para medición que indican la efectividad y la eficacia de una actividad en particular dentro de un proyecto. El uso regular de métricas durante el transcurso del proyecto nos puede aportar la información necesaria para conocer el estado del mismo, y por lo tanto ver la mejora. El uso de ellas nos puede proporcionar la información necesaria para tomar decisiones críticas (por ejemplo, tener que pasar más del 50 % de los casos de prueba en un tiempo reducido para poder entregar a tiempo una release interna del software).

Existen diferentes tipos de métricas: # de test cases, # de test cases Ejecutados, # de test cases “Passed”, # de test cases “Failed”, # de test cases “Blocked”, Tiempo de ejecución de los test cases, Total de errores, Total de Ejecuciones o Total de Bugs.

De este modo, con ellas podremos realizar los cálculos obteniendo información más útil, sobre todo para el Jefe de Proyecto. Algunos de los cálculos que podemos obtener son: % Completado, % Defectos corregidos, % Test Cases Passed o % Cobertura de Test.

En resumen, las métricas y la información obtenida de ellas nos ayudan a mejorar significativamente la calidad de nuestro proyecto. Asimismo, nos permiten ahorrar tiempo y dinero haciendo que el equipo avance con más rapidez.

4. HERRAMIENTAS DE DESARROLLO

4.1. Introducción

Las herramientas de desarrollo de software son aplicaciones informáticas que utilizan los programadores para poder crear, depurar, gestionar o mantener un programa. Existen múltiples empresas que ofrecen estos servicios, pero nosotros nos centraremos en los que ofrece Microsoft, ya que la principal ventaja que tiene es que su sistema operativo Windows está implantado en la mayoría de los ordenadores.

A continuación tenemos la figura que resume todas las herramientas que haremos uso, las cuales las explicaremos a continuación:

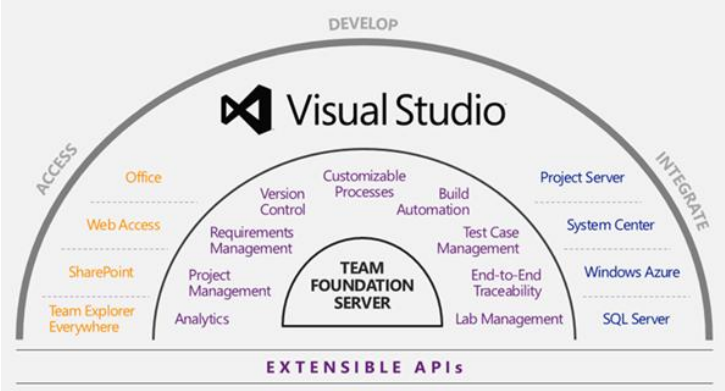

Figura 18: Resumen de herramientas de Microsoft en desarrollo de software.

4.2. Microsoft Visual Studio (VS)

4.2.1. Descripción

Una definición básica de Visual Studio podría ser “Son los recursos para ayudarle a compilar, administrar y aprender sobre sus aplicaciones”. En otras palabras, es un Entorno de Desarrollo Integrado (IDE) para sistemas operativos de Windows que permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte varios lenguajes y la plataforma .NET. De este modo, se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Existen diferentes versiones del producto, pero una de las más utilizadas para el desarrollo de software es Visual Studio Team Foundation Server. Esta versión permite usar una plataforma de colaboración en la gestión del ciclo de vida de una aplicación automatizando el proceso de entrega del software.

4.2.2. Team Explorer

Team Explorer es una ventana de herramientas ampliable dentro de Visual Studio que agrupa herramientas y artefactos por proyecto de equipo. Al inicio, Team Explorer está vacío y

se debe conectar a Team Foundation Server. A continuación, se puede seleccionar los proyectos de equipo que desea que aparezcan en TFS. Team Explorer se conecta sólo a un servidor Team Foundation Server y, por lo tanto, sólo muestra los proyectos de equipo de un servidor.

4.3. Microsoft Team Foundation Server (TFS)

4.3.1. Descripción

Team Foundation Server (TFS) proporciona un conjunto de herramientas que permiten la colaboración, comunicación y coordinación de un equipo. Además, TFS admite prácticas ágiles de desarrollo y proporciona las herramientas necesarias para administrar, de manera eficaz, los proyectos de desarrollo de software a lo largo del ciclo de vida.

Hacer uso de Team Foundation Server nos permitirá la colaboración del equipo a través de la comunicación mejorada, funciones multidisciplinarias, seguimiento del estado del trabajo, el proceso del equipo y la integración de las herramientas entre ellos. De este modo conseguiremos evitar la pérdida de información cuando se transfieren tareas de un miembro del equipo a otro.

4.3.2. Los proyectos de equipo

El concepto central cuando se trabaja con Team Foundation es el proyecto de equipo. Los proyectos de software pueden implicar la generación y publicación de un producto de software, así como una actualización o la publicación de una versión secundaria.

De este modo, el proyecto de equipo se almacena en Team Foundation Server y tiene un nombre que todo el equipo puede identificar fácilmente. Del mismo modo, este proyecto proporcionará una ubicación central para que los usuarios puedan coordinar su trabajo.

4.3.2.1. Características del proyecto

Dentro de cada proyecto tendremos una serie de propiedades que nos permitirán obtener el mayor rendimiento de nuestro equipo. A continuación se muestran las más importantes:

TIPO	DESCRIPCION
Portal del proyecto	Cada proyecto de equipo tiene un portal Web asociado de Microsoft SharePoint. Los miembros del equipo pueden utilizar el portal del proyecto para almacenar documentos, buscar informes y utilizar otras características de colaboración como las agendas y las listas.
Team Explorer	Es utilizado para tener acceso a los proyectos de equipo con los que se trabaja y está situado en Visual Studio.
Alertas	Son las alertas que TFS envía a los miembros del equipo, a través del correo electrónico, cuando se realiza algún cambio en el proyecto.

Control de código fuente	Permite al equipo administrar los archivos de código fuente del proyecto.
Generaciones	Team Foundation permite al equipo crear y administrar regularmente generaciones del producto, proporcionando informes sobre el estado y la calidad de cada uno.
Seguimiento del estado del trabajo	El propósito es vigilar la salud del proyecto, conocer a quién se le ha asignado trabajo y cuál es su estado.
Integración con otras aplicaciones	Es posible importar y hacer un seguimiento de copias de los elementos de las bases de datos del proyecto en Microsoft Project, Microsoft Excel, entre otras.
Integración en el entorno de desarrollo integrado	Muchas herramientas de Team Foundation se integran con el entorno de desarrollo integrado (IDE) de Visual Studio. Team Explorer es la ventana principal para trabajar con los proyectos de equipo.

Tabla 4: Características principales de un proyecto en Team Foundation Server.

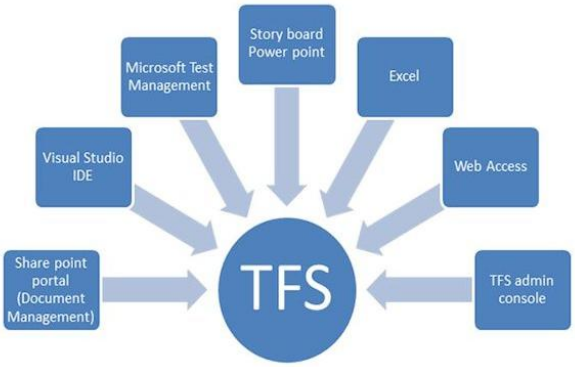

Figura 19: Integración de TFS con otras aplicaciones.

4.3.3. Proceso del Proyecto de Equipo (Team Project) con TFS

Por lo que hace al Team Project, podemos encontrar diferentes miembros dentro de él:

- *Administradores de proyecto:* administran las operaciones diarias de los servidores Team Foundation (configuraciones, creación de proyectos, copias de seguridad).
- *Responsables de proyecto:* se encargan de uno o varios proyectos de equipo. Supervisan la actividad diaria en el proyecto realizando consultas y generando informes, además de asignar los elementos de trabajo para asegurarse de que el proyecto sigue avanzando.
- *Miembros del Proyecto:* es cualquier persona que trabaja en uno o varios proyectos de equipo, los cuales deben saber cómo buscar y resolver los elementos de trabajo que tienen asignados y cómo trabajar con el control de código fuente.

Para la creación de un proyecto, lo primero será que los administradores instalen y configuren los servidores Team Foundation concediendo los permisos necesarios. Después, será el Project Manager el que deberá de crear nuestro Team Project desde el Team Explorer y comenzar el trabajo de seguimiento. Finalmente, los miembros del proyecto se unirán a los proyectos de equipo y comenzarán a crear elementos de trabajo o trabajar en los que tienen asignados.

4.3.4. Proyectos de equipo con TFS

Un proyecto de equipo es una infraestructura única que engloba todas las herramientas y los elementos independientes utilizados en el ciclo de vida del desarrollo de una aplicación de software.

Cada Team Project en desarrollo se agrupa virtualmente en un espacio de nombres propio destinado en exclusiva al proyecto de equipo. Por lo tanto, es simplemente un contenedor que aísla todas las herramientas y los artefactos asociados a una determinada aplicación de software en desarrollo, de tal forma que el resto de proyectos de equipo no tenga acceso a ellos.

Asimismo, un único servidor Team Foundation Server puede contener varios proyectos de equipo, cada uno de los cuales se crea en un espacio de nombres independiente.

4.4. Microsoft Test Manager (MTM)

4.4.1. Descripción

Microsoft Test Manager (MTM) permite realizar la fase de Testing a la que se debe someter cualquier aplicación en su ciclo de vida de desarrollo. Es importante indicar que para hacer uso de él, es necesario disponer de productos vinculados a él como el Team Foundation Server (ya que todos los casos de test se almacenan como Work Items de TFS), y una herramienta de cliente como es el Visual Studio Test Professional.

MTM puede grabar imágenes, comentarios, archivos adjuntos, narración de audio y video de la pantalla mientras estamos ejecutando los diferentes casos de prueba. La grabación hace que sea más fácil localizar cualquier bug que pueda aparecer en la aplicación bajo Testing. Asimismo, también se pueden almacenar las acciones como un caso de prueba, lo que facilita la réplica de la prueba después de que la aplicación bajo Testing sea actualizada a una nueva versión.

El Testing exploratorio (es decir, probar una aplicación sin un conjunto de pruebas definidas de antemano y sin un guión de pasos predeterminados) en MTM se puede realizar de dos formas: completamente libre o relacionado con algún Work Item que tengamos en el proyecto.

De este modo, con MTM podemos ayudarnos a conseguir que todo el ciclo de pruebas sea parte integral del proceso de desarrollo en equipo. Además, conseguiremos que los test cases queden accesibles a todos los miembros del equipo y alineados con los objetivos de las historias de usuario y nuestros planes de release.

4.4.2. El proceso de Testing con MTM

Los pasos a seguir en el Testing mediante MTM son sencillos, pero es necesario que se lleven en un orden establecido:

1) *Conexión con el servidor TFS y el proyecto.*
2) *Creación del plan:*

   Deberemos crear un plan de pruebas en el que agruparemos todos los casos de pruebas que creemos y ejecutemos. Para un proyecto concreto de TFS podremos tener todos los casos de pruebas que necesitemos y los que podrán estar asignados a varios planes a la vez.
3) *Creación de suites de pruebas:*

   Dentro de un plan de pruebas, organizaremos los casos de prueba en tres posibles tipos de suites:

   - *En base a un requerimiento o PBI:* se definen qué pruebas son necesarias para cada uno de los Work Items (o PBI) a entregar en una iteración.
   - *Manuales*: Este es el tipo de suite más simple que podemos crear. En ella agregaremos manualmente todos los casos de prueba que queramos que formen parte de la suite.
   - *En base a una consulta de Work Items:* En este tipo nos basamos en una consulta de Work Items creada a medida por nosotros.
4) *Agregar los casos de prueba.*

   Una vez que ya tenemos nuestro plan de pruebas definido, es el momento de agregar casos de prueba. La única excepción aquí son las suites basadas en una consulta de Work Items ya que no podemos añadir o crear nuevos casos de pruebas; simplemente, cuando se cree un nuevo test que cumpla las condiciones correspondientes, éste se agregará automáticamente.

4.5. Microsoft SharePoint (SP)

4.5.1. Descripción

Microsoft SharePoint (SP) es una plataforma de colaboración empresarial y comunicación, formada por productos y elementos de software que incluyen funciones de colaboración basados en el Explorador web, módulos de administración de proceso, módulos de búsqueda y una plataforma de administración de documentos. Asimismo, permite que los desarrolladores puedan crear soluciones rápidamente mediante estas herramientas y entornos conocidos sacando provecho de sus funcionalidades y características.

Desde el punto de vista funcional, SharePoint puede verse desde tres perspectivas:

- *Como manejador de información:* para almacenamiento de información.
- *Como sistema de representación de informació*n: publicación de sitios en Intranet, Extranet e Internet.

- *Como plataforma de desarrollo:* por medios de sus marcos de Modelos de Objetos y Servicios Web se puede modificar el sistema para adaptarlo a las necesidades de cada usuario.

4.5.2. Arquitectura de SharePoint

SharePoint se puede ver como un contenedor de información que está compuesto a su vez por otros contenedores, comenzando con las Bases de Datos (contenedores físicos) de contenidos hasta llegar a los documentos y elementos.

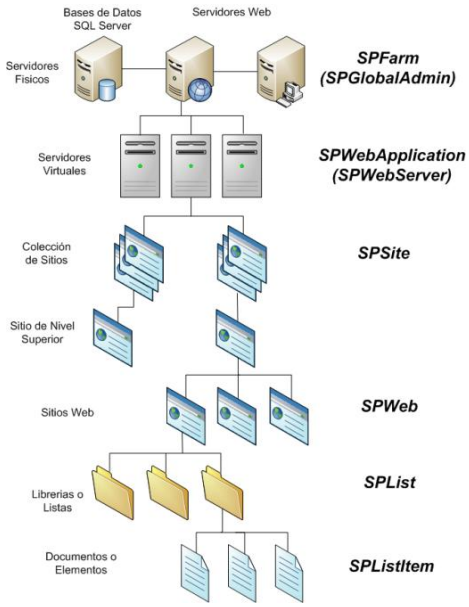

Figura 20: Arquitectura de SharePoint.

En la anterior figura podemos ver los niveles en los que se divide la arquitectura de SharePoint. El primer nivel representa la capa física de los servidores, también conocida como granja o conjunto de servidores.

A continuación, en el segundo nivel, se encuentra el primer contenedor de información que son las Aplicaciones Web. Cada una de ellas puede ser vista como un "servidor virtual" de SP, y a su vez, cada Aplicación Web es un contenedor de Colecciones de Sitios. Esto es necesario para que las Aplicaciones Web tengan una interfaz que los usuarios puedan utilizar.

La primera Colección es la "Raíz" (que contiene un Sitio creado por defecto denominado "Sitio de Nivel Superior") y tiene la misma dirección URL que la Aplicación Web (http://SitioPrincipal). Cada Colección de sitio subsiguiente tendrá una ruta administrada en su URL para poder diferenciarla de la Colección "Raiz"(http://SitioPrincipal/RutaAdministrada/Sitio).

A continuación, los Sitios Web constituyen la organización visible para los usuarios de SharePoint. Pueden organizarse en estructura de árbol, de tal forma que un sitio este constituido por sub-sitios que a su vez pueden tener más sub-sitios.

Finalmente, cada Sitio contiene Listas y Bibliotecas de Documentos, que serán los contenedores finales de información de SP. Asimismo, los documentos y elementos de Listas

constituyen el contenedor final de la información propiamente dicha (es decir, los metadatos que describen esa información).

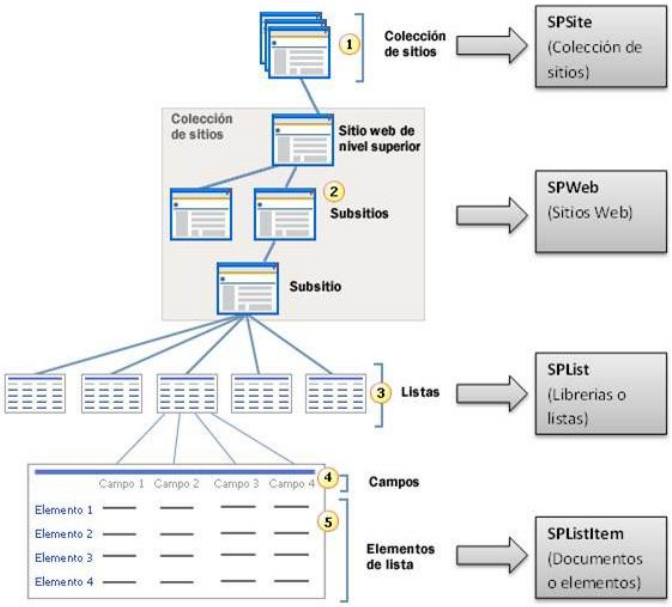

Figura 21: Modelo de objetos y arquitectura del Sitio en SharePoint.

4.5.3. Definiciones

4.5.3.1. Concepto de Aplicación Web

En SharePoint una Aplicación Web se define como un sitio web de Internet Information Services (IIS) al que se le aplican las extensiones de SharePoint, siendo contenedor de una o más Colecciones de Sitios que suponen los puntos de entrada para que los usuarios interactúen con el sistema y la información que contiene. Finalmente, las Aplicaciones Web se apoyan en Bases de Datos de contenidos de cualquier tipo de información.

4.5.3.2. Colecciones de Sitio

Proporcionan una forma de agrupar los sitios web de manera que compartan una serie de características y configuraciones comunes desde el Sitio “Raíz” (el más alto en la jerarquía) hasta todos los Sitios que forman parte de la Colección.

En cuanto a su creación, se hace bajo una aplicación Web que proporciona el mecanismo para comunicar las peticiones web con el contenido de SharePoint. Además, las Colecciones de Sitio guardan toda la información en bases de datos de SQL Server a las que se les denomina “Bases de Datos de Contenidos” (encargadas de almacenar la información de todos los sitios web de la Colección).

4.5.3.3. Sitios

Se define como una herramienta de colaboración, comunicación y almacenamiento de contenido que facilita el trabajo en equipo y simplifica el acceso a la información (documentos, enlaces web, contactos, listas de tareas…). Además, siempre se define en un nivel jerárquico por debajo del Sitio de nivel superior que define una Colección de Sitios automáticamente.

4.5.3.4. Listas y Bibliotecas

Ambos están diseñados para almacenar los documentos y elementos de información, que a su vez son los componentes que contienen la información que los usuarios desean preservar. Generalmente, las Listas son contenedores de elementos (de forma similar a una tabla en una base de datos donde guardar la meta-información) y las Bibliotecas contenedores de documentos.

4.5.3.5. Interfaz de Usuario

SharePoint dispone de una interfaz de usuario por defecto constituida por diferentes elementos, pudiendo ser modificada para personalizarla. La interfaz se configura a partir de la interacción de varios componentes: “La Página Maestra” (para definir las líneas generales del esqueleto de la interfaz que son comunes a todas las páginas de la implementación), “Las Hojas de Estilos” (es el aspecto gráfico de las páginas, como el color y fuentes de letras) y “El Contenido” (la información almacenada en bases de datos y que se mostrará).

4.5.3.6. WebParts

Una WebPart es una pieza de un Sitio de SharePoint que puede utilizarse en distintas partes del mismo, a la vez que puede personalizarse. Además, las WebParts son una parte importante en la apariencia de un Sitio ya que exponen funcionalidades y contenidos a través de la interfaz.

4.5.4. La Administración Central de SharePoint

Mediante ella es donde podremos revisar y parametrizar los componentes que necesitan ser configurados para hacer uso de SharePoint (administración de aplicaciones, configuración del sistema, copias de seguridad, asistentes de configuración, seguridad…).

5. PROCESO DE LA GESTIÓN DEL PROYECTO

El proceso que vamos a definir es solo un pequeño ejemplo de lo que se puede llegar a implementar. Existen muchas versiones dependiendo del tipo de tamaño del proyecto y el número de equipos y miembros que contengan.

Lo primero de todo será definir que metodología ágil haremos uso, en nuestro caso *Scrum*.

En segundo lugar, definiremos los roles de cada uno de los miembros que estarán involucrados en el desarrollo del producto:

- Product Owner: el cliente final.
- Scrum Master: jefe del proyecto.
- Scrum Team: desarrolladores y testers.

La siguiente tarea será crear el proyecto en TFS, y después ir añadiendo los Sprints conforme se vayan haciendo cada uno de ellos en el proceso de desarrollo. Cada Sprint abarcará una serie de objetivos a alcanzar desglosados en especificaciones o requisitos escritos en cada PBI (Product Backlog Item). Estos PBI los introducirá el Scrum Master en la herramienta TFS, bien mediante dicha herramienta o por Visual Studio, asignándolos a cada uno del Scrum Team para su desarrollo. Una simple búsqueda en la herramienta TFS con Visual Studio nos mostrará los PBIs que contiene el Sprint actual.

De forma paralela a lo anterior, el equipo QA formado por los Testers deberá crear los Sprints del proyecto y el Master Test Plan en Microsoft Test Manager. Los Test Cases creados en el proceso de desarrollo nunca se borrarán y será en el Master Test Plan donde deberán de copiarse y organizarse. Además deberá tener una estructura que sea sencilla para localizar un caso de prueba concreto:

ESTRUCTURA GENÉRICA

- Máster Test Plan
  - Tipo de App
    - Secciones dentro de la APP
      - Functional
      - No Functional
      - Visual
      - ….

Ahora es el turno de empezar los Sprint, donde al inicio de cada uno se realizará la reunión de planificación en la que acordará cuáles de estos PBIs entran, a quien se les asignará y cuales llevarán un Testing asociado.

Una vez finalizada esta reunión se procederá a dar paso al inicio del desarrollo y testing del Sprint. Las dos partes tendrán diferentes tareas, pero estarán muy ligadas entre ellas:

- Desarrollo: cada miembro del equipo desarrollará el PBI asignado, informando en todo caso cuando está acabado y preparado para la fase de Testing.
- Testing: los Tester deberán de realizar las tareas mostradas a continuación por cada PBI que deba ser testeado, mientras los desarrolladores trabajan en el desarrollo de éstos:
  1. *Creación de las tareas* de creación y ejecución de cada PBI.
  2. *Static Testing:* revisión de requisitos de cada PBI para entenderlo.
  3. *Test Creation:* creación de los casos de prueba de cada PBI.
  4. *Test Execution:* En el momento que los desarrolladores pongan a Testing los PBIs, se empezarán a ejecutar los casos de pruebas escritos. Cuando el desarrollador ya ha realizado el PBI que se le ha asignado, en el TFS pondrá el estado “Testing”, el cual indicará que el Tester ya puede ejecutar los Test Cases correspondientes.

5. *Reporte de Bugs:* en el caso de encontrar Bugs, se reportarán de forma formal y por email al Scrum Master para que sea conocedor de ellos y se prioricen.
6. *Bug State y Bug Resolution*: se crearan y asignarán los bugs específicos en el TFS para que cada desarrollador lo solucione.
7. *Priorización de Test Cases:* una vez acabado el Sprint, a los Test Cases creados se les asignará una prioridad, siendo numérica con orden de importancia decreciente (Nivel 1: Muy alto; Nivel 2: Alto; Nivel 3: Medio; Nivel 4: bajo).

Antes de acabar el periodo del Sprint y con los PBI cerrados (siempre en la medida de lo posible) se llevará a cabo la Revisión del Sprint y, posteriormente, la Demo con los todos los componentes del rol de Scrum. En ella se mostrarán los avances y/o modificaciones realizados durante el Sprint.

Para dar por finalizado el Sprint, se hará la reunión de Retrospectiva con el Scrum Master y Scrum Team, donde se pondrán en común lo positivo y negativo del Sprint finalizado. De este modo, dará comienzo otro nuevo Sprint empezando todo este proceso.

En todo este proceso del Sprint, las consultas a cada PBI las podremos realizar mediante el TFS, Visual Studio o SharePoint. Esta última herramienta nos permitirá compartir desde el portal todos los documentos del desarrollo, de información del equipo, consultas de estados de PBI, fechas de entregas de Demos, fechas de inicio y fin de Sprint, o cualquier cosa necesaria para el buen funcionamiento del proceso.

6. EJEMPLO DE APLICACIÓN

El ejemplo que vamos a realizar será el desarrollo de una aplicación que realiza los cálculos de Sumar y Restar de dos números introducidos. La aplicación podrá dar el resultado de la operación o borrar los datos. El diseño de la interfaz de la aplicación se tendrá especificada de antemano por el propio cliente.

A continuación mostramos los pasos seguidos en el desarrollo del software:

1) Instalaremos en el servidor las aplicaciones Microsoft SharePoint 2010 y Team Foundation Server, en dicho orden. De esta manera podremos realizar la integración de ambas aplicaciones fácilmente.
2) Instalaremos en el PC la aplicación Microsoft Visual Studio 2010 Ultimate.
3) Con Visual Studio nos conectaremos al servidor donde hemos instalado el TFS:

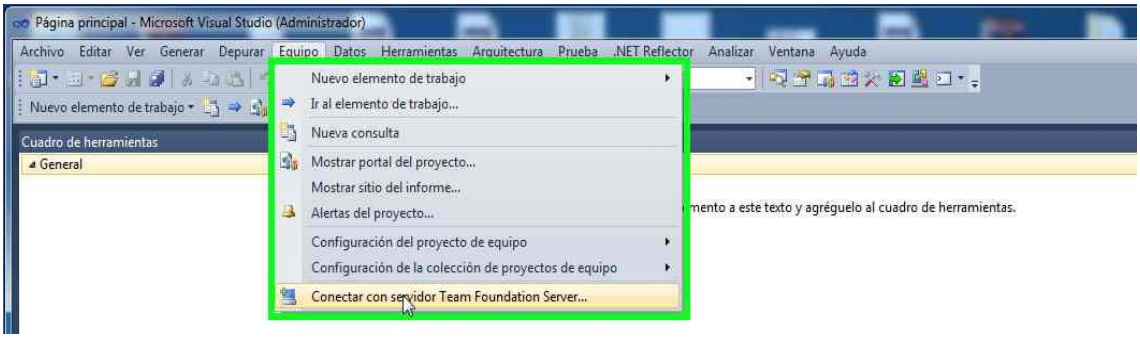

Figura 22: Conexión al servidor TFS.

4) Creamos el proyecto de equipo desde Visual Studio:

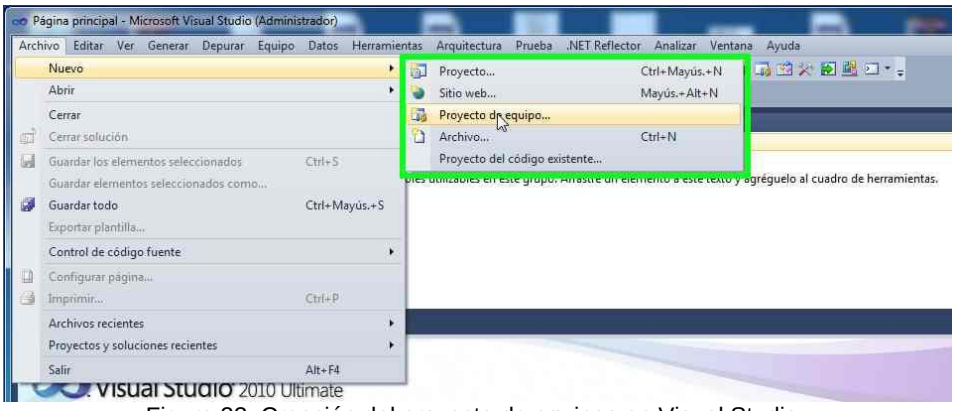

Figura 23: Creación del proyecto de equipos en Visual Studio.

5) Realizamos las configuraciones necesarias del proyecto de equipo (nombre, descripción) y plantilla a utilizar (en nuestro caso será la de Agile).

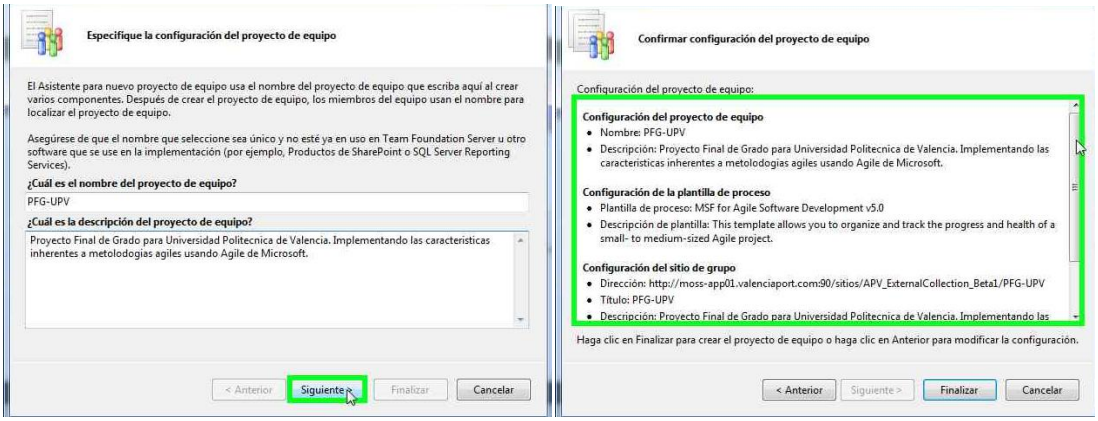

Figura 24: Pantallas de configuración del proyecto de equipo.

6) En el Team Explorer del VS podremos verificar que el proyecto ha sido creado:

Figura 25: Ventana de Team Explorer con el proyecto creado.

7) Para acceder al sitio de SharePoint del proyecto, haremos clic con el botón derecho del ratón en el nombre del proyecto que nos aparece en el Team Explorer:

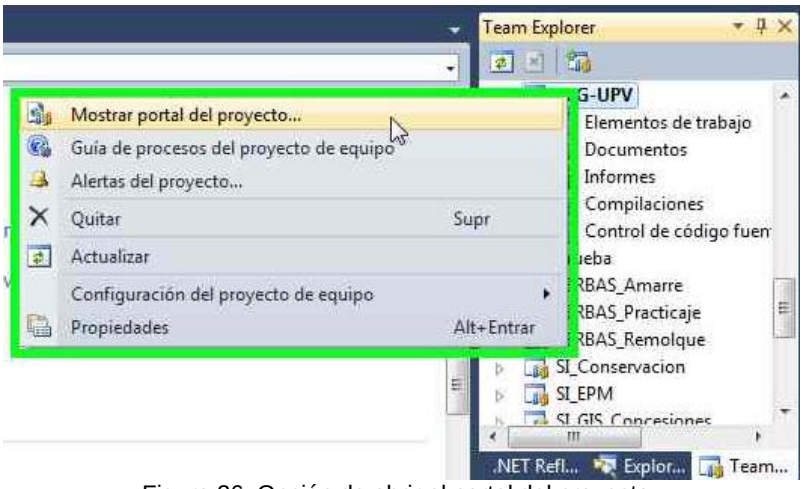

Figura 26: Opción de abrir el portal del proyecto.

8) Tras introducir el nombre de usuario y contraseña con los permisos suficientes para poder editar, se nos mostrará nuestro sitio principal del proyecto en SharePoint mediante el navegador:

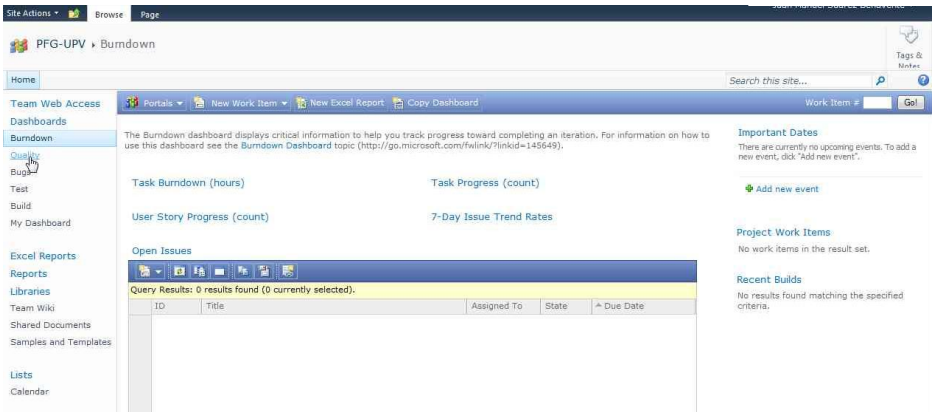

Figura 27: Sitio principal del proyecto en SharePoint.

9) Realizamos la primera reunión de planificación del proyecto, en el cual se detallarán los PBI del Sprint 1 y la asignación de cada uno de ellos a cada miembro del grupo.

10) Creamos el Sprint 1 en el TFS mediante Visual Studio, donde añadiremos los PBIs que contendrán las especificaciones detalladas, así como sus tareas de testing.

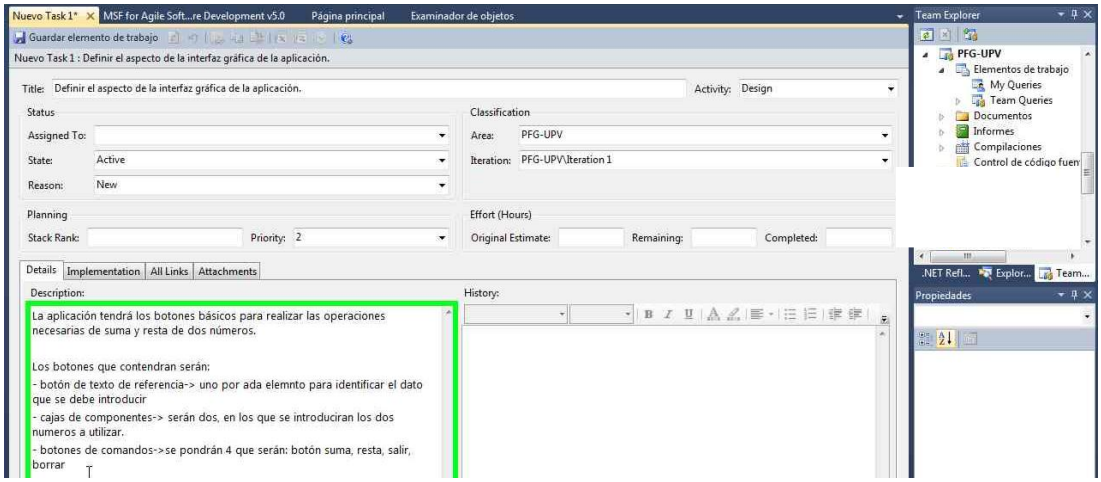

Figura 28: Creación de PBIs en Visual Studio.

Una simple búsqueda con una Query del actual Sprint nos mostrará todos los PBI creados:

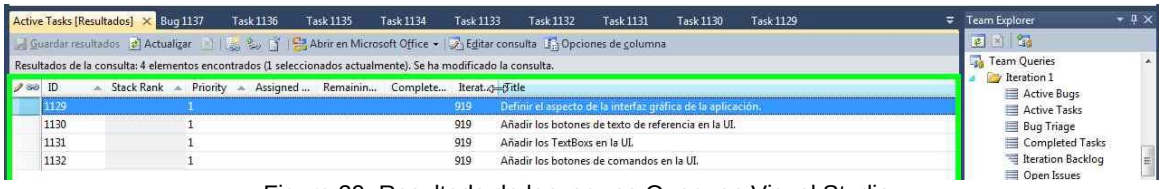

Figura 29: Resultado de lanzar una Query en Visual Studio.

Ahora, el Scrum Master pondrá a Approved el PBI y será, a continuación, cuando el desarrollador comience su tarea asignada. El PBI pasará al estado de Commited.

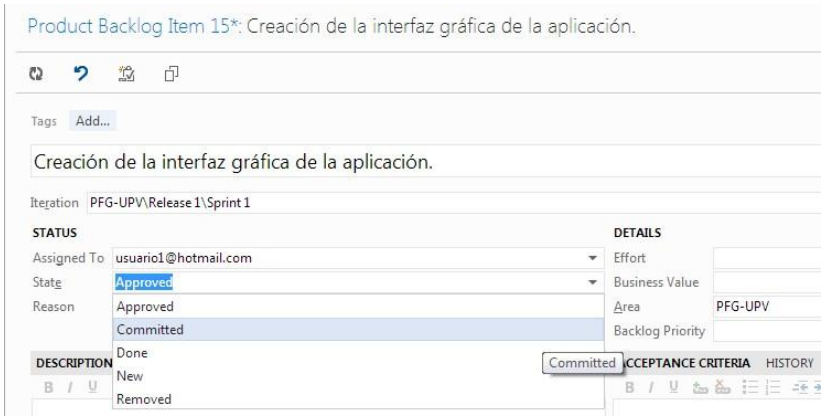

Figura 30: Cambios de estados de los PBIs.

11) Mientras tanto, en la aplicación Microsoft Test Manager, el Tester creará la estructura del Master Test Plan y el Sprint 1. Dentro del Sprint 1 añadiremos sus PBIs asociados.

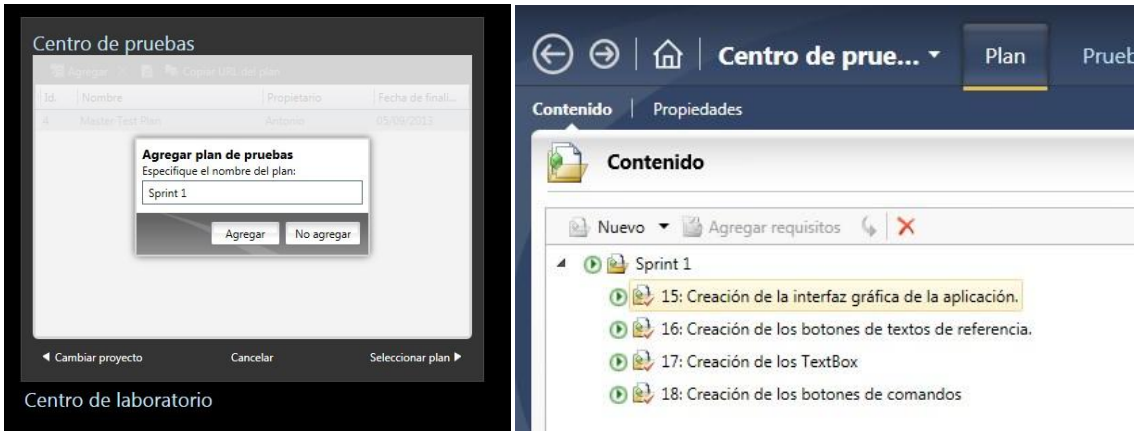

Figura 31: Creación del Sprint y adición de PBIs en Microsoft Test Manager.

12) Creamos los Casos de Test en el MTM para cada uno de los PBIs del actual Sprint, y a la vez pondremos la tarea Test Creation del TFS en modo “In Progress”.

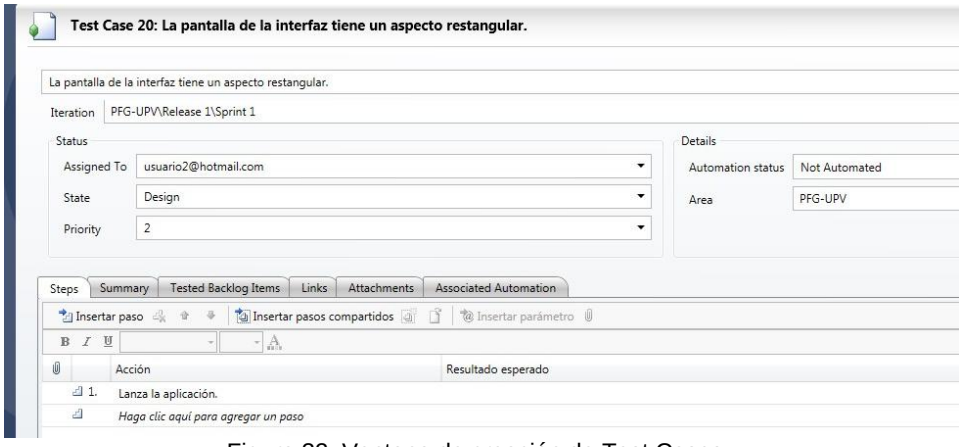

Figura 32: Ventana de creación de Test Cases.

13) Una vez finalizada la creación de los casos de prueba en el MTM, pondremos la tarea Test Creation a “Done”.

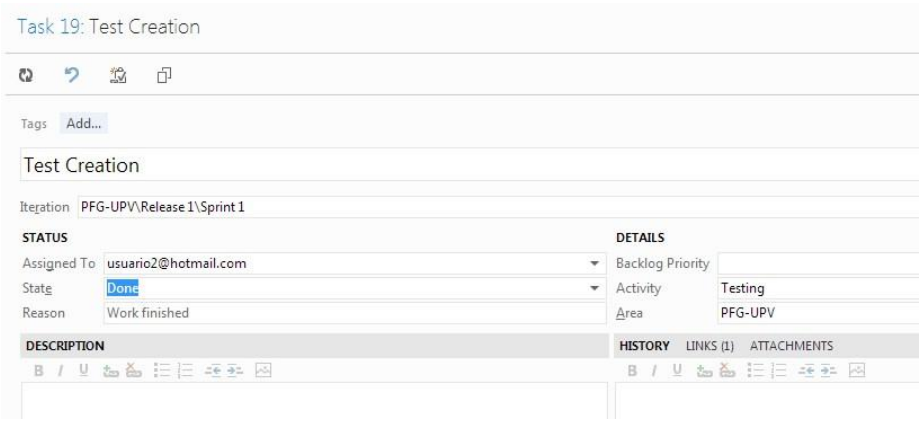

Figura 33: Tarea Test Creation en estado a Done.

14) Cuando el desarrollador haya finalizado su tarea, pondrá el PBI al estado de Testing.
15) Ahora es turno del Tester, el cual ejecutará los casos de prueba escritos y deberá de poner la tarea Test Execution a “In Progress”.

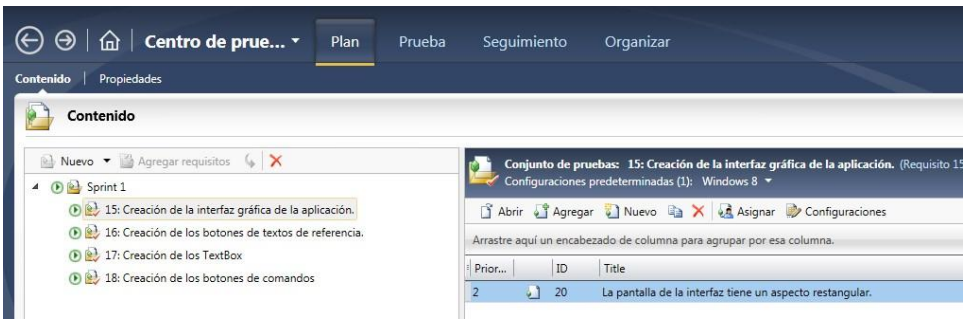

Figura 34: Conjunto de pruebas de cada PBI que compone el Sprint.

16) Los Test Cases se irán poniendo a Pass si todo sale como se esperaba en las especificaciones, que son los pasos descritos en los propio Test Cases.

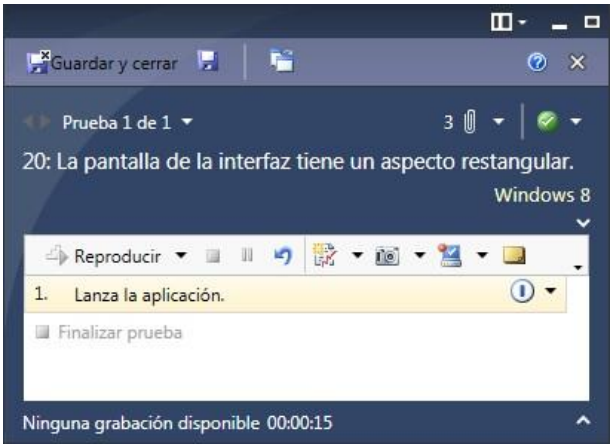

Figura 35: Ejecución de los pasos a reproducir en un Test Case.

17) Tras la aparición de un Bug, deberemos de poner el caso de prueba a Fail y reportarlo.

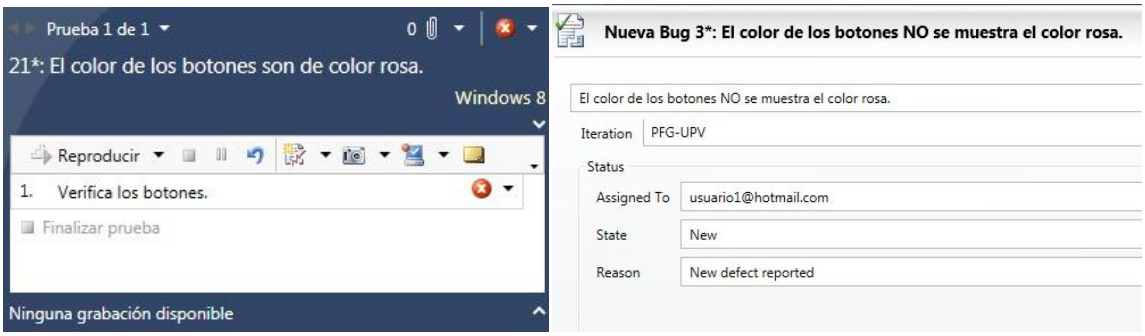

Figura 36: Test Case a Fail y creación del Bug.

18) Después de pasar todos los casos de prueba, pondremos los PBI que han sido satisfactorios a Done en el TFS. En cambio, los que no han salido según lo esperado, los dejaremos abiertos para el siguiente Sprint.

Con SharePoint podremos ir viendo mediante gráficos como se va pasando el desarrollo de la aplicación en función del tiempo, así como poder visualizar los posibles retrasos del Sprint.

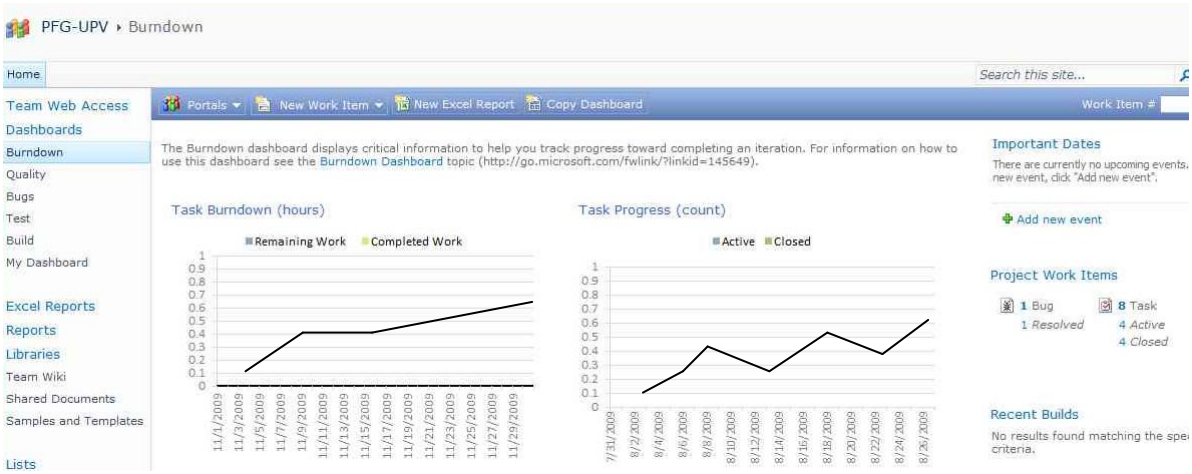

Figura 37: Gráficos de proceso del Sprint en SharePoint.

19) En el MTM, el Tester priorizará los casos de prueba ejecutados y se pasarán los casos de prueba del Sprint 1 al Master Test Plan.

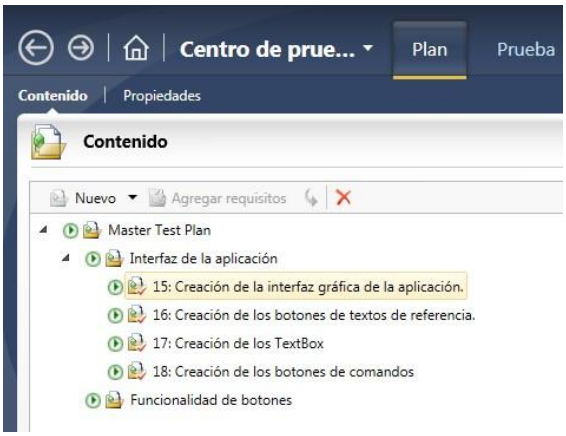

Figura 38: Master Test Plan con los PBI introducidos.

20) Realizaremos la Revisión del Sprint viendo lo desarrollado y la Retrospectiva indicando posibles mejoras. Asimismo, en la Demo mostraremos los avances de la aplicación.
21) Volvemos a realizar una nueva reunión de Planificación del nuevo Sprint Final, donde elegiremos los nuevos PBIs y el Bug que hemos dejado abierto en el anterior Sprint. Todos ellos los añadiremos a un nuevo Sprint dentro del proyecto con VS.

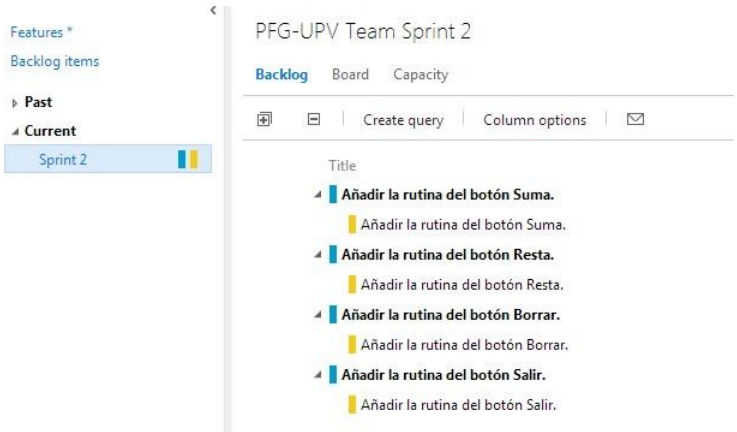

Figura 39: Creación del nuevo Sprint.

22) Volveremos a realizar de nuevo los pasos 9) al 20), pero en este caso (ya que estamos en un ejemplo corto) tendremos que dejar todos los PBI cerrados y a Done, así como todos los Bugs corregidos.

En la siguiente imagen podemos ver cómo se van pasando los PBIs a los diferentes estados del Sprint y sus tareas asignadas a cada componente del equipo según la metodología Scrum.

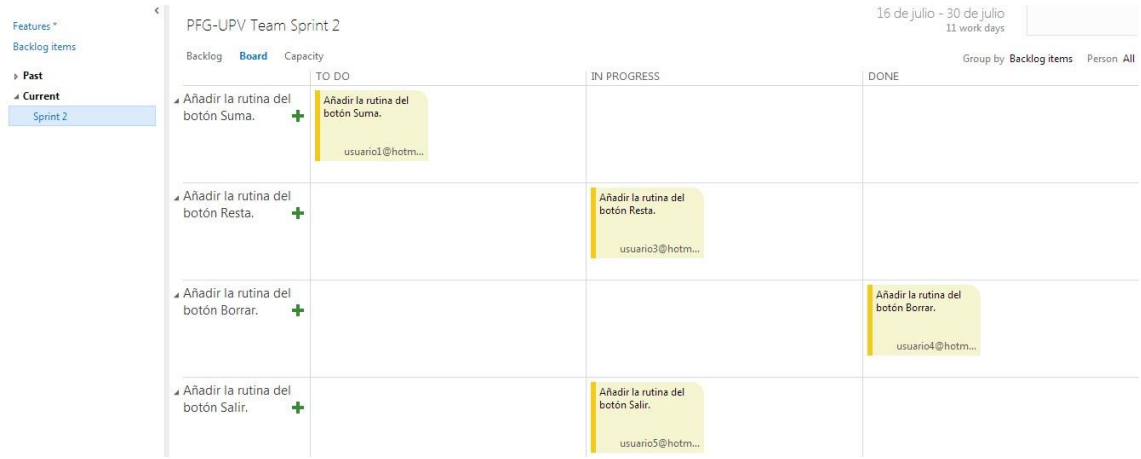

Figura 40: Dashboard donde se muestran los estados de cada Sprint.

23) Una vez que tenemos finalizado todo el desarrollo de la aplicación, deberemos de ejecutar todos los casos de prueba del actual Sprint y de los anteriores que tenemos recogidos en el Master Test Plan. De esta manera nos aseguraremos que la aplicación funciona como se espera y reduciremos los posibles Bugs.

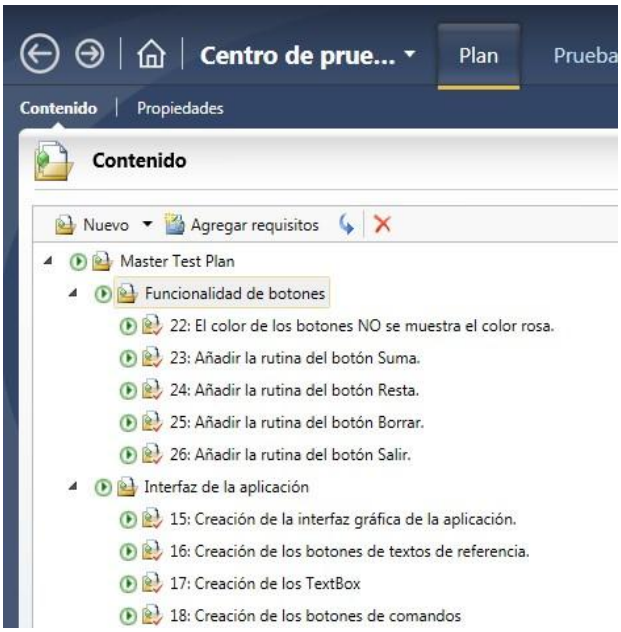

Figura 41: Master Test Plan con todos los PBIs y Test Cases del proyecto.

24) Finalizamos con la última Revisión del Sprint y la Demo final de la aplicación, en la que al cliente se le dará la versión final de la aplicación.

7. BIBLIOGRAFIA

[1] Muentes Molina, Verónica Estefanía. *Exposición sobre Team Foundation Server*. Manabí: Universidad laica “Eloy Alfaro” de Manabí, 2012.

[2] Arboleda Jiménez, Hugo F. *Modelos de Ciclo de Vida de Desarrollo de Software en el Contexto de la Industria Colombiana de Software*. Bogotá-Colombia: Universidad de San Buenaventura Cali, 2002.

[3] José H. Canós; Patricio Letelier; Mª Carmen Penadés. *Métodologías Ágiles en el Desarrollo de Software.* Valencia: Universidad Politécnica de Valencia.

[4] Roberth G. Figueroa; Camilo J. Solís; Armando A. Cabrera. *Metodologías tradicionales vs. Metodologías ágiles*. Ecuador: Escuela de Ciencias en Computación de la Universidad Técnica Particular de Loja,

[5] Cruzado Paredes, Naim Jhon. *Instalación y configuración de team foundation server 2010*. Lima-Peru, 2010.

[6] Palacio, Juan. *Flexibilidad con Scrum.* España: Safe Creative, 2007

[7] Laboratorio Nacional de Calidad del Software de INTECO. *GUIA DE INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA*. España: Instituno nacional de tecnologías de la comunicacion, 2009.

[8] N.Satheesh Kumar; S.Subashni. *Software Testing Using Visual Studio 2010.* Editorial Packt Publishing Enterprise Ltd, 2010.

[9] Glenford J.Myers; Corey Sandler. *The Art of Software Testing*. Second Edition. John Wiley & Sons, Inc.2004

[10] Microsoft Developer Network, Julio 2013: http://msdn.microsoft.com

[11] Microsoft Sharepoint, : http://sharepoint.microsoft.com

[12] El Manifiesto Ágil: http://agilemanifesto.org/iso/es/

[13] Testing en español, 2013: http://pruebassw.blogspot.com.es/

[14] Sinfonia del testeo, 2013: http://sinfoniadetesteo.blogspot.com.es

[15] Manual de testing, 2013: http://www.guru99.com/software-testing.html

[16] Desarrollo Ágil con SCRUM, Mayo 2013:http://www.icons.es/software-1/37-ingenieria-software/65-desarrollo-agil-scrum