Universidad de las Ciencias Informáticas
Facultad 5 Entornos Virtuales

Propuesta de estrategia ágil para el desarrollo de video juegos.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autores: Reina Mercedes Bauta Gómez

Dallandy Castillo Barbosa

Tutor: Ing. Igr Alexander Fernández Saúco
Asesor: MSc. Pedro Carlos Pérez Martinto

Ciudad de la Habana, junio del 2007
"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Dallandy Castillo Barbosa

_____

Reina Mercedes Bauta Gómez

_____

Tutor: Ing. Igr Alexander Fernández Saúco

_____

A mi patria: Cuba.

Al Comandante, por materializar este sueño.

A la UCI, por mis mejores años.

A mis padres, familia y amigos; por creer en mí.

Reina.

Mami y papi, aquí tienen realizado su sueño,

a mis familiares presentes y ausentes

gracias por su apoyo y confianza.

Dallandy.

Resumen

El presente trabajo de diploma titulado “Propuesta de Estrategia Ágil para el desarrollo de video juegos”, surge por la necesidad de poner en manos del grupo de desarrollo de Juegos Virtuales de la Facultad 5 de la Universidad de las Ciencias Informáticas (UCI), una metodología que agilizara el proceso de desarrollo de video juegos.

En esta investigación fueron objetos de análisis metodologías de desarrollo de software como *Rational Unified Process* (RUP), *Agile*, *Extreme Programming (*XP), *Microsoft Solution Framework* (MSF) y *Microsoft Solution Framework Agile* (MSF ágil); siendo esta última la propuesta de solución técnica a defender en este trabajo. Paralelamente se determinó trabajar con el entorno integrado de desarrollo *Visual Studio 2005 Team System* (VSTS) haciendo uso de las prestaciones de *Visual Studio Team Foundation Server* (TFS).

MSF ágil contiene las mejores prácticas para la administración de proyectos y es adaptable a los cambios que el cliente requiera para el producto en cualquier etapa dentro del ciclo de vida del proyecto y a diferencia de las metodologías tradicionales, MSF ágil tiene exigencias razonables en cuanto a la creación de documentos y otros entregables.

Los objetivos concretos de esta investigación fueron determinar cuál metodología o modelo de desarrollo de software aplicar al proyecto de Juegos Virtuales a partir de un estudio de las ya existentes y analizar y proponer herramientas que permitan agilizar el proceso de producción y gestión de software de forma integrada.

La documentación de la metodología MSF ágil y las experiencias de su posterior aplicación, en conjunto con herramientas integradas para lograr una mejor gestión del proceso de desarrollo de software, fueron los resultados obtenidos en la presente investigación.

Palabras claves: investigación, video juego, realidad virtual, metodología, ágil, proceso, desarrollo, software, herramientas, integración, MSF ágil, proyecto.

Introducción

La informática incluye la ciencia, la práctica para el procesamiento y la ingeniería de sistemas de información. Es la encargada de estudiar la estructura, el comportamiento, y las interacciones de los sistemas naturales y artificiales que almacenan, procesan y comunican la información, además de desarrollar sus propios basamentos conceptuales y teóricos. La informática abarca desde las computadoras, los individuos y toda la información generada en los procesos de las organizaciones, hasta aspectos de cómputo, cognoscitivos y sociales, incluyendo el estudio del impacto social de las tecnologías de información.

Como herramienta tecnológica, la computadora permite a las personas, mediante la comprensión de los códigos de las nuevas tecnologías, entender el mundo en que viven, adaptarse activamente a la sociedad y ser conscientes que el conocimiento es dinamizador del crecimiento, fundamental para el cambio y la transformación social; por eso es necesario adentrarse en el mundo de la informática teniendo en cuenta desde un principio conceptos como el de software y el de cómo llevar a cabo la ingeniería de los productos de esta índole.

Para que un producto de software tenga éxito es preciso hacer una gestión correcta del proceso de producción del mismo sin descuidar elementos que incluyen desde el planeamiento del problema, la valoración de riesgos, la evasión de los errores clásicos, la participación del usuario en todo el proceso de desarrollo, hasta la aplicación de una metodología adecuada de software y la valoración *post-mortem*.

Los informáticos en el afán de automatizar otros procesos han descuidado sus propias tareas, dando lugar a un sin número de errores que han provocado el fracaso parcial o total de algunos proyectos. La poca motivación, las fricciones entre los desarrolladores y el cliente, las expectativas poco realistas, la falta de un manager efectivo para el proyecto, la pérdida de tiempo, el diseño no adecuado, la falta de supervisión y de estimaciones, los requerimientos excesivos, la sobre-estimación de las ventajas del uso de una nueva herramienta, el cambio de herramientas a mitad del proyecto y la falta de un control de código fuente automático son, por citar algunos, los errores más comunes relacionados con los procesos de desarrollo, junto a los productos y las tecnologías que se emplean en la elaboración de un software.

La no aplicación de alguna metodología y herramientas en el proceso de producción de software de realidad virtual, presupone una mala gestión del mismo. En la Universidad de las Ciencias Informáticas (UCI) no se cuenta con una metodología o modelo que en concordancia con el desarrollo de la informática en nuestro país, permita la realización de video juegos de manera eficiente; lo que trae como consecuencia que en el proceso de creación de los mismos, se realicen pasos innecesarios que retrasen la culminación del proyecto y por ende, el proceso tienda a ser más costoso, atrasando la entrega final del producto. **¿Cómo disminuir los problemas en la realización de video juegos en el campo de la informática?**

Al plantearse esta interrogante se derivaron los siguientes **objetivos**:

1. Determinar cuál metodología o modelo de desarrollo de software aplicar para el proceso de producción de software del proyecto de Juegos Virtuales de la Facultad 5 de la UCI, a partir de un estudio de las ya existentes.
2. Analizar y proponer herramientas que permitan agilizar el proceso de producción y gestión de software de forma integrada.

Para el cumplimiento de los objetivos anteriormente expuestos, se conformaron diferentes **tareas investigativas** como son:

1. Estudiar las metodologías existentes como son: *Rational Unified Process* (RUP), *Agile*, *Extreme Programming* (XP), *Microsoft Solution Framework* (MSF) y *Microsoft Solution Framework Agile* (MSF ágil).
2. Estudiar las herramientas y técnicas existentes que agilizan el proceso de producción y gestión de software: Automatización de las pruebas–*NUnit*, *Refactoring*, *Visual Assist* e ingeniería inversa, por citar algunas.
3. Estudiar y configurar el *Microsoft Visual Studio 2005 Team System* (VSTS), el *Microsoft Visual Studio Team Foundation Server* (TFS) y el *Microsoft Office SharePoint Server 2007* como propuesta final de herramientas a usar.

Los posibles resultados a obtener son:

- Documentación de la metodología a usar en el proceso de producción de software.
- Aplicación de la metodología en el proceso de producción de software de realidad virtual, específicamente en el proyecto de Juegos Virtuales.
- Uso de herramientas integradas en el proceso de producción de software que permitan la gestión del mismo.

En un proceso de desarrollo de software, la metodología define QUIEN debe hacer QUE, CUANDO y COMO debe hacerlo; por esa razón no existe una metodología de desarrollo de software universal, dado que las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable. De igual manera sucede con las herramientas a utilizar para automatizar las actividades definidas en dicho proceso de desarrollo de software.

Cuando la informática era todavía una ciencia demasiado novedosa, según plantea (Zeledón), los sistemas basados en computadora se desarrollaban usando técnicas de gestión orientadas al hardware. Los gestores del proyecto se centraban en el hardware, debido a que era el factor principal del presupuesto en el desarrollo del sistema y para controlar los costes del hardware establecieron controles formales y estándares técnicos que exigían un análisis y diseño completo antes de que fuera construida la más ínfima parte del sistema. Estos controles a su vez, medían el proceso para determinar dónde podían hacerse mejoras. En resumen, los gestores de proyecto de aquel entonces aplicaron los controles, métodos y herramientas que se reconocen como la ingeniería del software.

Hoy, la distribución de costes en el desarrollo de sistemas informáticos ha cambiado radicalmente. El software en lugar del hardware, es usualmente el elemento principal del coste, según (Zeledón).

Aunque el avance de la informática en nuestro país es notable, según plantea (Zeledón), se puede decir que se ha aprendido relativamente poco sobre la ingeniería de los sistemas computacionales; por tanto debe ser un objetivo básico para cualquier grupo de desarrollo, defender el hecho de que tener conocimiento de toda la ingeniería de un software es lo esencial sin priorizar excesivamente el costo; y si para lograrlo se utiliza una metodología desarrolladora adecuada y herramientas integradas al proceso de desarrollo de un software de realidad virtual como lo es un video juego, mejor aun.

Para hacer un video juego hay que emplear métodos informáticos evolutivos. La peculiaridad de la mayoría de los video juegos de hoy es que los equipos de desarrollo suelen preocuparse mucho por contar con la “tecnología de punta” para tener un producto que destaque por gráficos, inteligencia artificial y que requiera más prestaciones de hardware. Este es un requerimiento importante puesto que la tecnología cambia rápidamente; pero no es menos cierto también que siguiendo fielmente una metodología se terminaría teniendo un producto obsoleto en cuanto saliese al mercado, según (Zeledón).

Es importante tener presente que una metodología no es algo que haya que seguir al pie de la letra, pues en un proyecto de software influyen factores tales como tamaño del equipo de trabajo, preparación y experiencia de sus miembros, tipo de producto, etc, que son diferentes para cada grupo de desarrollo. Por tanto, lo más conveniente es adaptar la metodología a las capacidades/necesidades de tu equipo/proyecto y no al revés.

En el presente trabajo se emplearon los métodos científicos de tipo histórico-lógico y analítico-sintético con el objetivo de constatar la evolución que han sufrido las distintas metodologías de desarrollo de software y a través del análisis de documentos y definiciones teóricas, encontrar la herramienta que facilitará que el proceso de desarrollo de software sea adaptable a proyectos específicos como el de Juegos Virtuales.

El contenido de este trabajo de diploma se estructura en un total de tres capítulos. En el primero se realiza un análisis de metodologías y modelos de desarrollo de software existentes. Comparación de las mismas y selección de una como propuesta de solución técnica para su aplicación en el proyecto de Juegos Virtuales de la facultad 5 de la UCI. El contenido del siguiente capitulo es la argumentación teórica de la propuesta de solución técnica (MSF ágil) y de distintas herramientas integradas al proceso de desarrollo de software, así como la selección de la propuesta final de herramientas a utilizar. El último capítulo contiene experiencias de la aplicación de la metodología y de las herramientas seleccionadas, así como su configuración.

Nota aclaratoria importante: Casi la totalidad de la bibliografía que se tuvo en cuenta para realizar la presente investigación fueron textos e imágenes en idioma inglés, dado que la tecnología que se propone aplicar es de la creación de la corporación *Microsoft*. Como el idioma en que se defiende este trabajo de diploma es el español, se aclara que los términos que aparecen escritos en inglés no tienen una traducción literal al lenguaje que compete a estos textos y por eso su aparición se conserva en el lenguaje original.

Capítulo I. Marco Teórico Referencial

Introducción

En el presente capítulo se exponen diferentes metodologías y modelos de desarrollo de software: *Rational Unified Process* (RUP), *Agile*, *Extreme Programming* (XP), *Microsoft Solution Framework* (MSF) y *Microsoft Solution Framework Agile* (MSF ágil); así como sus características, ventajas y desventajas en dependencia de los requerimientos del proceso, y se realiza de manera conjunta la comparación de las mismas con la finalidad de seleccionar entre ellas, sobre la base teórica de esta investigación, la óptima para el proyecto de Juegos Virtuales de la Facultad 5 de la UCI.

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se lleva una metodología adaptable a las condiciones del equipo/proyecto, solo se obtienen clientes y desarrolladores insatisfechos con el resultado.

Muchas veces se realiza el diseño de un software de manera “rígida” con los requerimientos que el cliente exigió, de tal manera que cuando en la etapa final o de prueba, el cliente solicita un cambio se hace muy difícil realizarlo, pues de llevar a cabo dicho cambio, se alterarían muchas cosas que no estaban previstas. Es justamente lo antes planteado, uno de los factores que ocasiona un atraso en el proyecto y por tanto la incomodidad del desarrollador por no cumplir con el cambio solicitado y el malestar por parte del cliente por no tomar en cuenta su pedido. Obviamente, para evitar estos incidentes, se debe llegar a un acuerdo formal con el cliente al inicio del proyecto, de tal manera que cada cambio o modificación no perjudique el desarrollo del mismo.

Conceptos fundamentales:

Metodología

Desde el punto de vista informático, un proceso de software detallado y completo suele denominarse "Metodología" según plantea (P.Letelier). Las metodologías se basan en una combinación de los modelos de procesos genéricos (cascada, evolutivo, incremental, entre otros). Adicionalmente una metodología debe definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc.

Define (Miguélez 1999) que los métodos son vías que facilitan el descubrimiento de conocimientos seguros y confiables para solucionar los problemas que la vida nos plantea.

Los métodos de la ingeniería de software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas, según (P.Letelier).

Proceso

Un proceso (del latín *processus*) es un conjunto de actividades o eventos que se realizan o suceden con un determinado fin. Este término tiene significados diferentes según la rama de la ciencia o la técnica en que se utilice, define (Wikipedia 2007).

Un proceso define *quién* esta haciendo *qué*, *cuándo*, y *cómo* alcanzar un determinado objetivo. En la ingeniería del software la idea es construir un producto software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que el estado actual de la tecnología permite. En consecuencia, reduce el riesgo y hace el proyecto más predecible. El efecto global es fomento de una visión y una cultura comunes, según (Ivar Jacobson 1999).

Según (CIUP 2006), un proceso es un conjunto de actividades que realiza una organización, mediante la transformación de unos insumos, para crear, producir y entregar sus productos, de tal manera que satisfagan las necesidades de sus clientes.

Software

Se denomina software, según (Paraguay 2007), a todos los componentes intangibles de un ordenador o computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica. Esto incluye aplicaciones informáticas (tales como un procesador de textos) que permite al usuario realizar una tarea, y software de sistema (como un sistema operativo) que permite al resto de los programas funcionar adecuadamente, facilitando la interacción con los componentes físicos y el resto de las aplicaciones.

Proceso de desarrollo de software

Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software (Ivar Jacobson 1999)

Metodología de Desarrollo de Software

“Conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas informáticos.”, definió (Maddison 1983).

Sobre la base de la presente investigación, los autores consideran que las metodologías para el desarrollo de software se pueden clasificar en:

- Metodologías orientadas a objetos (OO)
- Metodologías tradicionales (no ágiles)
- Metodologías ágiles

Ingeniería de software

La Ingeniería de software es la rama de la ingeniería que crea y mantiene las aplicaciones de software desarrollando tecnologías y prácticas de las ciencias computacionales, manejo de proyectos, ingeniería, ámbito de la aplicación, y otros campos. Extraído de(Wikipedia 2007).

La (Wikimedia 2007) define que las mejoras prácticas de la ingeniería de software que proporcionan que el proceso de desarrollo de un producto sea satisfactorio son: la administración de requisitos, el modelado visual del software, el desarrollo iterativo, el control de cambios, la verificación de la calidad del producto y el uso de la arquitectura basada en componentes. No obstante, las mejoras prácticas que se apliquen a un proyecto determinado para lograr una buena ingeniería, dependen de las necesidades de cada equipo de trabajo/proyecto.

Video juegos

Un video juego o juego de vídeo es un programa informático, creado expresamente para divertir, basado en la interacción entre una persona y un aparato electrónico donde se ejecuta el video juego. Estos recrean entornos virtuales en los cuales el jugador puede controlar a un personaje o cualquier otro elemento de dicho entorno, para conseguir uno o varios objetivos por medio de unas reglas determinadas, planteó (binario 2007).

Iteración

Según plantea (Fernandez 2005) iteración es: “Cada una de las repeticiones de las acciones contenidas en un grupo de {instrucciones: cada una de las líneas que componen un programa en un ordenador que se repiten un número determinado de veces hasta que se cumple una condición de un programa (bucle)”}.

Las iteraciones son siempre puntos de control, aprendizaje y planificación. El ciclo primario del planeamiento es la iteración.(GUIDANCE 2005).

1.1 Análisis de Metodologías de Desarrollo de Software.

Si se ha estado inmerso en la confección de un software determinado o de una pieza de éste, en algún momento ha surgido la interrogante sobre ¿qué metodología debo usar?

Dentro de la presente investigación se estudian metodologías probadas, con el objeto de desarrollar soluciones informáticas para video juegos.

1.1.1 *Rational Unified Process* (RUP)

El *Rational Unified Process* (RUP), es un proceso de desarrollo de software que junto con el lenguaje de modelado *Unified Model Language* (UML), constituye la metodología estándar para el análisis, implementación y documentación de sistemas orientados a objetos (OO).

Esta metodología tiene como características principales la forma de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo) a cada uno de los miembros del proyecto. También permite la implementación de las mejoras prácticas de la ingeniería de software.

RUP es una metodología que presenta tres características distintivas como proceso de desarrollo de software. Es centrado en la arquitectura, guiado por casos de uso (CU) y además iterativo e incremental. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración, según (Ivar Jacobson 1999).

El proceso de desarrollo de RUP está dividido en fases y flujos de trabajo que unidos conforman el ciclo de vida del proyecto, el cual se desarrolla por iteraciones.

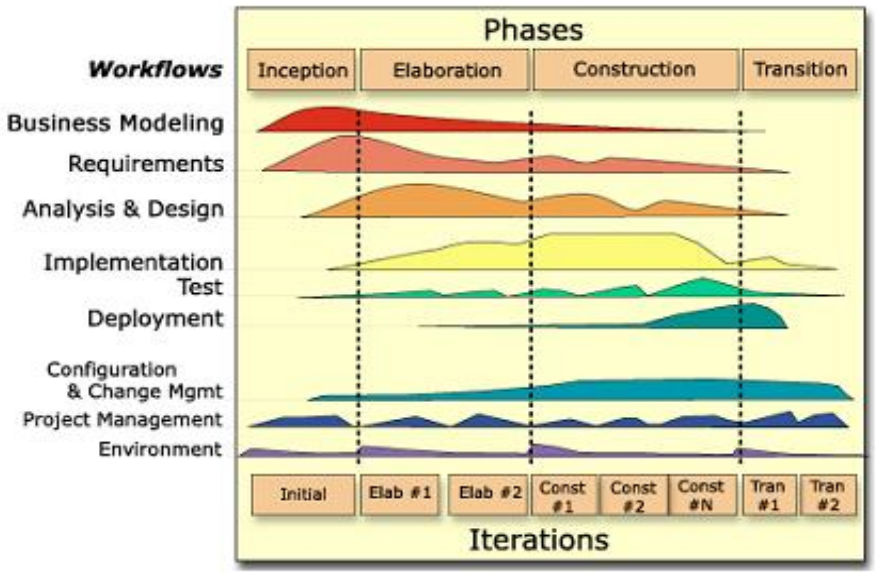

Fig. 1 Vista general de RUP (Sanchez 2004)

En la primera fase de RUP, conocida como **inicio**, conceptualización o concepción, se realiza un plan de fases y se identifican los principales casos de uso, se empiezan a percibir los riesgos que pueden existir, y se comienza a tener una visión del proyecto.

En la segunda fase, nombrada **elaboración**, se lleva a cabo un plan de proyecto y comienzan a completarse los casos de uso, se determina la arquitectura a utilizar y se eliminan los riesgos encontrados en la primera fase.

Le sigue la fase de **construcción**, que se encarga de elaborar un producto totalmente operativo y eficiente, también se realiza el manual de usuario y se llega a obtener la capacidad operacional inicial.

Y en la fase de **transición** el *release* se encuentra listo para su instalación en condiciones reales y a su vez, puede implicar reparación de errores, además se entrena a los usuarios, y como consecuencia suelen surgir nuevos requisitos a ser analizados.

Como ya se expuso, estas fases se desarrollan mediante el ciclo de iteraciones, el cual consiste en reproducir el ciclo de vida de un proyecto cada vez que se acaba una iteración.

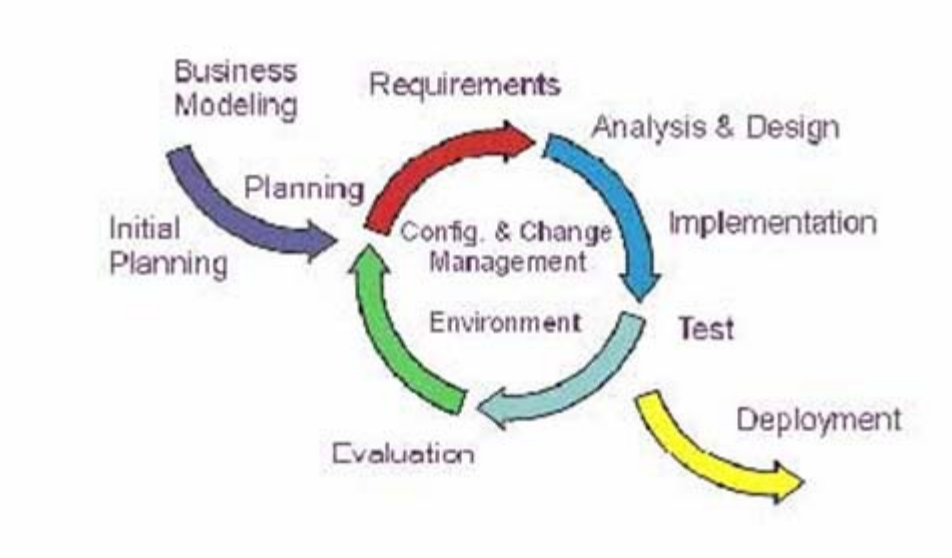

Fig. 2 Flujos de Trabajo (Molpeceres 2003)

El proceso de flujos de trabajo define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y los resultados (artefactos) que se espera de ellos.(Molpeceres 2003)

Entre los flujos de trabajo se encuentra el **modelo de negocios** que, como su nombre lo indica, es el que se encarga de entender las necesidades y hacer la ingeniería del negocio; en este flujo se hallan los requerimientos, que se ocupan de llevar las necesidades del negocio de un sistema tradicional a un sistema automatizado.

Le siguen los flujos de trabajo de **análisis y diseño**, que se hacen cargo de llevar los requerimientos dentro de la arquitectura de software.

Luego está el flujo de **implementación**, cuya misión consiste en crear un software que se ajuste a la arquitectura y que funcione de la forma deseada.

Por último se tiene la etapa de **prueba**, en la que debe asegurarse que el comportamiento de la versión del producto es el correcto y que todo lo que se solicitó se encuentra en el software.

La metodología RUP también está compuesta por elementos que nos ayudan en la realización del proyecto como son las **actividades** (procesos que se determinan en cada iteración); los **trabajadores** (vienen siendo las personas u objetos involucrados en cada proceso) y los **artefactos** (pueden ser un documento, un modelo o un elemento de un modelo).

Una particularidad de esta metodología, plantea (Sanchez 2004), es que en cada ciclo de iteración, se hace exigente el uso de determinados artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

La metodología RUP está concebida para el desarrollo de sistemas basados en programación (OO). Esta se puede emplear en proyectos que sean de diferentes lenguajes de programación. Por ser RUP un proceso de desarrollo de software genérico que tiene varias aplicaciones adaptables al proyecto que se esté realizando según su naturaleza.

También se conoce que RUP fue creada para proyectos de grandes dimensiones en cuanto a tamaño y duración, así como para grandes equipos de trabajo. A la hora de obtener requisitos, RUP crea como base los CU y el historial de usuarios o *User Stories* en inglés, los cuales describen los requerimientos de la aplicación desde el punto de vista del usuario y definen los requisitos técnicos sin inmiscuirse en los detalles de la implementación.

En cuanto a la carga de trabajo se puede decir que RUP es un proceso “pesado”, que depende mucho de la documentación. Existen diferentes elementos de planificación (plan de desarrollo, plan de iteración, plan de calidad, etc.) con los que se controla el desarrollo del software.

A través de un predefinido esquema de escalabilidad y gestión de riesgos, se pueden reconocer, prevenir y corregir problemas tempranamente, así como fallos. RUP define en cada momento del ciclo de vida del proyecto, qué artefactos, con qué nivel de detalle, y por cuál rol se deben crear. Según (Molpeceres 2003), se definirán qué artefactos son necesarios para poder realizar una actividad y cuales se deberán crear durante dicha actividad.

RUP le presentará al cliente los artefactos del final de una fase y se valorarán las precondiciones para la siguiente (definición de riesgo, aceptación del plan de iteraciones, prototipos, etc.) y sólo después que el

cliente acepte los artefactos generados, se pasará a la siguiente fase. La calidad de estos artefactos será probada durante la totalidad del ciclo de vida del proyecto a través de distintas medidas de calidad como convenciones, revisiones y auditorías periodísticas, pruebas, etc.

También RUP intentará reducir la complejidad del software a producir a través de una planificación intensiva en cuanto a conocimiento sobre la arquitectura. Así intentará evitar que por la desaparición de alguna pieza clave del equipo, se pierda el conocimiento sobre la aplicación. En cuanto a la evaluación del estado del proyecto esta metodología es tan grande y compleja como en el resto de las características previamente reflejadas, por lo que para manejar el volumen de información se requiere mucho tiempo, lo que constituye una desventaja de esta metodología.

Otra desventaja de RUP lo compone el hecho de que para el desarrollo de software con equipos pequeños (hasta unas diez personas), es RUP definitivamente muy grande y prácticamente inalcanzable; esto supone que si se ha decidido implantarla, antes se debe adaptar hasta el punto de hacerla parecer otro proceso, lo que también requiere de tiempo y coste.

1.1.2 *Microsoft Solution Framework* (MSF)

Plantea (Spaces 2005) que *Microsoft Solution Framework* (MSF) no es como tal una metodología, sino prácticas que, ajustándose al contexto de proyecto (tamaño del equipo, frecuencia de entregas, etc.) serán más o menos recomendables de aplicar. Para el proyecto se podrán seleccionar aquellas prácticas que realmente agreguen valor al proceso.

Cada práctica se compone de una secuencia de actividades y las mismas se describen en **ETVX**, un modelo de documentación de procesos introducido en los '80 que sirve para representar criterio de entrada/salida, tareas, verificaciones y validaciones.

Fig. 3 Actividades de MSF (Sanchez 2004)

Las prácticas generan resultados tangibles en forma de productos de trabajo. Estos resultados aunque análogos a los artefactos de RUP, no son necesariamente los mismos. Todo proyecto se separa en cinco fases principales como queda representado en la siguiente figura:

Fig. 4 Fases de MSF análogas a RUP

Según (Consultores 2006), las fases de MSF son las siguientes:

Visión y Alcance. Trata de unir el proyecto/producto a las experiencias de otros trabajos, el equipo debe saber lo que el cliente desea y cómo lo desea, en esta fase se definen los líderes del proyecto, se hace la primera evaluación de riesgo del proyecto, se plantean cuáles son los objetivos que se persiguen con el

trabajo y hasta dónde se quiere llegar con el proyecto, además se realiza la evaluación inicial de riesgos del proyecto.

Planificación. Aquí se termina la planificación del proyecto, se realiza el proceso de diseño de la solución, se crea el plan de trabajo y se estima el costo. En fin, se planifica todo con respecto al proyecto.

Desarrollo. Se empieza a construir el proyecto tanto en la documentación como en el código y también se desarrolla la infraestructura del proyecto.

Estabilización. En esta fase se empiezan hacer las primeras pruebas, que enfatizan el uso y manipulación bajo condiciones reales; se empiezan a detectar los errores y a darle solución a los mismos.

Implantación. Se decide la tecnología base y sus componentes, por último se obtiene la aprobación del cliente.

Para compensar algunas de las desventajas de las estructuras jerárquicas impuestas por los equipos de proyecto se desarrolló un modelo de roles, que no es más que la organización de los equipos de proyecto en modelos más pequeños y multidisciplinarios, para que así sus miembros puedan conformar una visión común del proyecto y guiar su enfoque de trabajo hacia la implementación de soluciones, ya que tienen que compartir responsabilidades con los demás miembros del equipo y así lograr un alto estándar de calidad y un mayor deseo de aprender en cuanto al tema en que se desenvuelve el proyecto.

Estos modelos de MSF tienen roles que corresponden a las metas de un proyecto y estos son responsables de que las mismas sean cumplidas. Todos los roles de un proyecto tienen igual importancia en su aporte al mismo, aunque estos pueden tener diferentes niveles de actividades durante las etapas de desarrollo y ninguno puede ser omitido.

MSF cuenta además con modelos que se encargan de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión de Riesgos, Modelo de Diseño de Soluciones, Modelo de Infraestructura, Modelo de Costo Total de Propiedad y finalmente el Modelo de Aplicación.

El éxito de aplicar estos modelos MSF se basa en la existencia de algunos factores como los que muestran en la siguiente imagen:

Fig.5 Factores para el éxito de los modelos de MSF

El punto de visión se necesita para proveer la guía requerida para tomar decisiones técnicas. Luego con un conjunto de puntos de referencia se puede realizar un seguimiento efectivo de la marcha de los procesos o proyectos, con énfasis en el manejo de los riesgos durante todo el ciclo de vida. La capacidad de reutilización es importante para tomar ventaja del conocimiento previo en forma estructurada y consistente en un ambiente tecnológico flexible.

1.1.3 *Agile*

Las metodologías de desarrollo de software han tenido que seguir progresando consecuentemente con el avance de las tecnologías. En los últimos seis años ha surgido un nuevo grupo de metodologías entre las que se encuentra la llamada Metodología ágil, la cual surge para acabar con el “burocratismo” de aquellas metodologías que como RUP necesitan mucha documentación. Es por eso que la “ágil” es más aceptada, ya que el trabajador ve mejor el fruto de su desempeño y no se siente cansado en la medida que va

evolucionando el proyecto; por eso se dice que esta metodología da más valor al individuo y lo hace sentir más a gusto trabajando con ella.

El término "ágil" surge en febrero de 2001 en una reunión celebrada en Utah-EEUU cuyo objetivo era hacer un esbozo de los valores y principios que permitieran a los equipos desarrollar productos que fueran capaces de responder con mayor rapidez a los cambios que pudieran surgir con la creación del proyecto y así poder desarrollar un software en la menor cantidad de tiempo posible.

Esta metodología no necesita tanta documentación para el desarrollo del proyecto, sino que exige una cantidad más pequeña para cada tarea y no es tan "rígida" como las demás metodologías.

La filosofía de esta metodología se encuentra resumida en el "Manifiesto ágil" creado en la reunión efectuada en el año 2001 y según (Jose H. Canós 2005) está basado en valores y principios.

En el manifiesto se valora:

- Al individuo y las interacciones del equipo de desarrollo más que al proceso y las herramientas. Las personas son el principal factor de éxito de un proyecto de software. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente al mismo; pero es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- Desarrollar software funciona más que conseguir una buena documentación. La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo elemental.
- La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Principios del Manifiesto ágil:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que agreguen valor al proceso.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre las entregas.

IV. Trabajar juntos desarrolladores y personal del negocio implicado a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Proporcionarles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para transmitir información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los organizadores, desarrolladores y usuarios deberán ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares de tiempo, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y basado en esto ajusta su comportamiento.

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

Tabla 1. Diferencias entre metodologías ágiles y no ágiles (Jose H. Canós 2005)

1.1.4 *Extreme Programming* (XP)

Un ejemplo de metodología ágil es la metodología *Extreme Programming* (XP), que ha sido creada para eliminar las actividades improductivas y a su vez reducir costos y frustraciones. El éxito de la misma consiste en enfatizar la satisfacción del cliente y promover el trabajo en equipo para solucionar el eterno problema del desarrollo de software por encargo y entregar a tiempo el resultado que el cliente necesita. También presenta una relación con el cliente muy diferente a la que presentan las metodologías tradicionales que se basan en la fase de captura de requisitos que se realiza antes de comenzar a realizar el software y una fase de validación que se hace posterior al desarrollo del mismo.

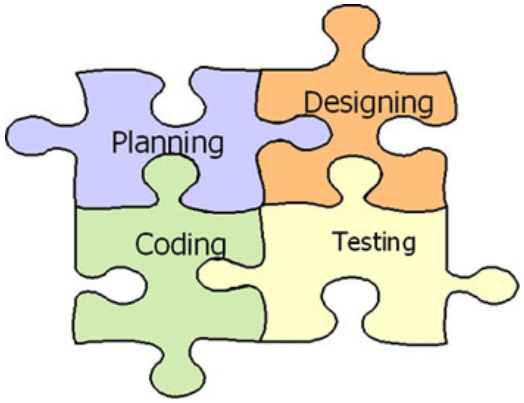

Fig. 6 Procesos generales de XP (Sanchez 2004)

Plantea (Sanchez 2004) que la metodología XP tiene como principio para su fundamentación acortar el ciclo de desarrollo de software y la participación del cliente desde el primer momento en que comienza el proyecto y hasta el final del mismo, esto permite que en el momento que surja el problema se le pueda dar solución de manera inmediata y así no tener que arrastrarlo hasta el final del proyecto. También responde de manera rápida a las necesidades del cliente y los cambios que en el transcurso del proyecto éstos puedan hacer a su pedido. La XP es útil para fortalecer la unidad del equipo de trabajo y lograr una mejor comunicación, de manera que se puedan explotar las capacidades de cada uno de los integrantes.

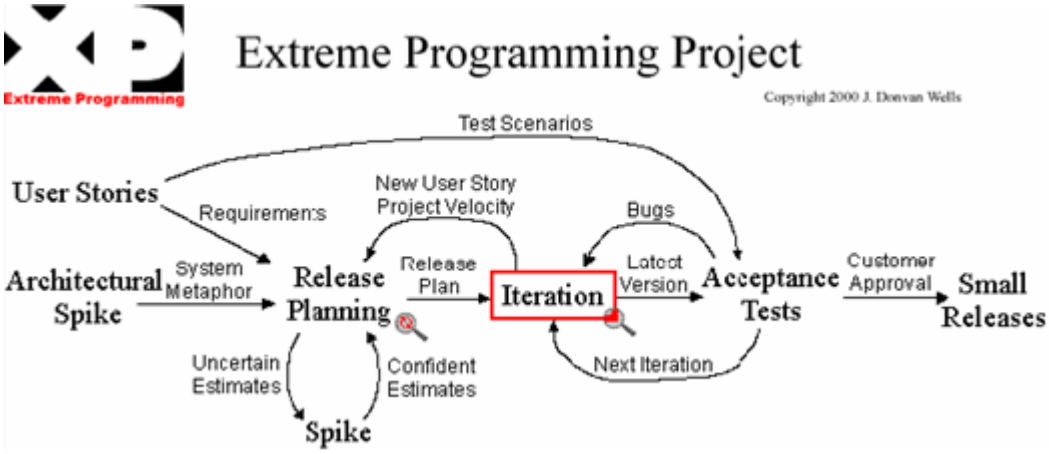

Fig. 7 Proceso de desarrollo con la metodología XP (Molpeceres 2003)

Esta metodología tiene las características fundamentales siguientes:

- Existe una estrecha comunicación entre los diseñadores y programadores, los integrantes del equipo y el cliente, y entre los mismos integrantes del equipo.
- El diseño del software es sencillo.
- El grupo de prueba permite obtener una mejor comunicación entre los usuarios y el cliente y así tomar sus experiencias.
- A medida que se proponen los cambios en el producto se van realizando lo más pronto posible y se reajustan el tiempo y costo en función de la evolución real del proyecto.

Estas características demuestran que la metodología XP es la más adecuada para un proyecto que se encuentre en constante cambio y evolución a causa de los avances en la tecnología y en los negocios.

Los principales valores que posee la metodología XP es la comunicación entre sus integrantes, la simplicidad, el valor y respeto tanto entre sus miembros como entre los jefes del proyecto y el cliente. La unión y correcta aplicación de estos valores conlleva al éxito del proyecto y al de la propia metodología.

Por lo general todas las demás metodologías tienen por norma que el coste del cambio en el desarrollo de un proyecto va aumentando a medida que va pasando el tiempo, sin embargo XP es innovadora en este sentido.

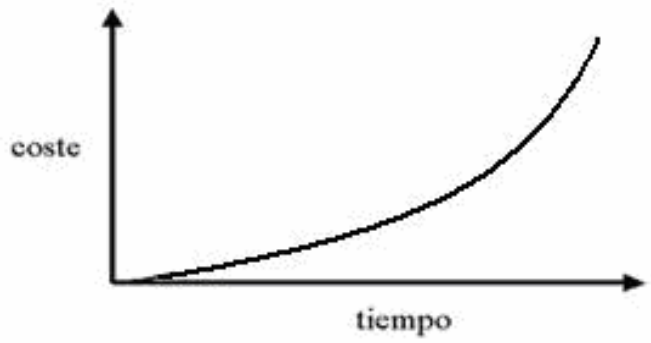

Fig. 8 Curva de los métodos tradicionales

La metodología XP plantea que la curva anteriormente mostrada ya ha perdido validez y que al combinar buenas prácticas de programación y tecnología es posible lograr que la curva sea la contraria, como la figura que sigue, y esto es lo que se pretende conseguir con esta tecnología.

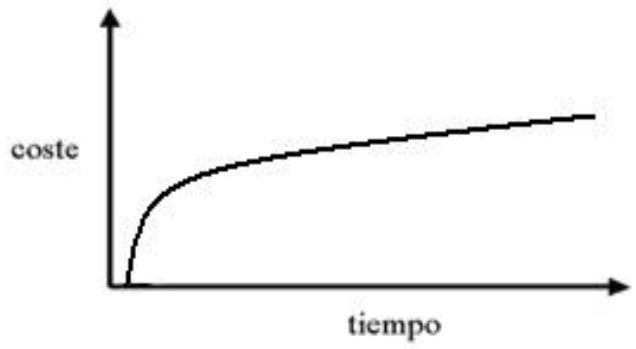

Fig. 9 Curva planteada por la metodología XP

Lo planteado por XP en esta gráfica se lograría si en vez de diseñar para el cambio, se diseñara lo más sencillo posible, haciendo en cada momento solo lo que es imprescindible hacer, pues la propia simplicidad del código, y sobre todo las pruebas y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario.

1.1.5 MSF ágil

La versión más reciente [según el periodo en que se enmarca esta investigación] de *Microsoft* para el entorno integrado de desarrollo *Visual Studio* conocido por sus siglas en inglés (IDE) *es el Visual Studio 2005 Team System* (VSTS), el cual permite definir procesos MSF y crear proyectos para el trabajo en equipos (*Team Projects)*. En esta versión vienen por defecto dos plantillas para los procesos de desarrollo: uno ágil y el otro formal; no obstante el VSTS permite modificar estas plantillas para crear versiones personalizadas.

RUP	MSF ágil
Proceso iterativo e incremental.	Proceso iterativo que incrementalmente va aproximando entregables de mayor fidelidad.
Pensado para grandes equipos de trabajo.	Pensado para equipos reducidos.
Dirigido solamente por casos de uso.	Dirigido por escenarios que cubren cierta información específica del actor o persona *y* por requerimientos de calidad de servicio (como seguridad).
Centrado en la arquitectura.	Asigna mayor relevancia al análisis de riesgos.

Tabla 2. Comparación entre RUP y MSF ágil.

MSF ágil es un conjunto de procesos que consta de roles. Estos roles no tienen que ser realizados por diferentes personas sino que según lo que estime el proyecto quizás una persona pueda desempeñar más de un rol.

Las grandes actividades de este proceso se le dan a los *work items*. Existen diferentes tipos de *work items* predefinidos que son: escenario*,* requerimiento de calidad de servicio*,* riesgo*,* tarea y *bug.* En otros procesos más elaborados se pueden encontrar éstos y/u otros *work items*, e incluso crear nuevos.

Es interesante conocer que el estupendo soporte que trae la herramienta de desarrollo *Visual Studio 2005*, permite que MSF ágil no se quede sólo en una brillante idea, sino que se lleve a cabo un desarrollo de software exitoso ya que, entre otras cosas, permite extender tanto las guías de proceso como las plantillas de los mismos, alterar los *work items*, las actividades y/o los flujos de trabajo.

De ese modo, es sencillísimo lograr adaptar un proceso existente a uno más acorde al contexto conveniente, y que la herramienta garantice su adecuado funcionamiento para evitar tener que cumplir con un procedimiento en forma involuntaria.

Consideración del capítulo

Posteriormente al estudio del contenido expuesto en este capítulo y de otras fuentes bibliográficas consultadas, se concluye que la metodología MSF ágil es la que más se ajusta a las características del grupo de desarrollo de Juegos Virtuales de la Facultad 5 de la UCI y que es, por ende, la metodología que debe aplicarse a este equipo de desarrollo, por lo que se propone como solución técnica al problema científico planteado en este trabajo, ya que parece ser el modelo que va a arrojar del modo requerido, el producto de software deseado según los requisitos del usuario que se interese por los juegos virtuales.

Capítulo II. MSF para la dirección ágil

Introducción

Se ha estado buscando un método para que las prácticas probadas que se aplican internamente en los ciclos de vida de un producto, así como la entrega de la solución a los clientes; pueda contribuir a la comunidad del desarrollo del software. Si se sigue el "patrón ágil" de definir el trabajo por iteraciones y se programa "prácticamente en función del tiempo", se puede decir que MSF para el desarrollo ágil será realmente ágil.

El desarrollo requiere las prácticas más comunes. Se tienen algunas técnicas innovadoras a las barreras comunes del proceso de desarrollo que proporcionan alternativas a usar en un proceso ágil. Esto hace a MSF ágil asequible para muchos proyectos, por eso uno de los primeros pasos a dar por el equipo de desarrollo ante la disyuntiva de cuál metodología usar, debe ser analizar si el proyecto se adapta al enfoque ágil.

MSF para el desarrollo ágil del software contiene la dirección para la organización de los analistas del negocio, de los managers del proyecto y del release del producto, de los arquitectos, de los liberadores y de los *testers*.

MSF cabe en el marco de las soluciones de *Microsoft* como un sistema integrado de la dirección de procesos que abraza metodologías ágiles y formales y proporciona un espacio para poner una solución en ejecución, modificada para requisitos particulares de una amplia variedad de proyectos.

No puede haber proceso de desarrollo de software para todos los proyectos de desarrollo de software. Ésta es la filosofía central detrás del marco de las soluciones de *Microsoft*.

MSF para el desarrollo ágil del software es un proceso que utiliza muchas de las ideas incorporadas al *Visual Studio Team System* (VSTS). El proceso MSF incorpora prácticas probadas desarrolladas en *Microsoft* alrededor de requisitos, de diseño, de seguridad, de funcionamiento, y de la prueba. MSF ágil

para el desarrollo del software es el primer proceso ágil para que se determine el ciclo de vida de desarrollo del software.

El VSTS es una plataforma productiva, integrada, y herramientas extensibles del ciclo de vida del desarrollo del software que ayuda a los equipos de trabajo a ir mejorando la comunicación y la colaboración a través del proceso de desarrollo del software.

2.1 MSF para el desarrollo ágil del software

Algo que ocurre frecuentemente por la necesidad de poner en orden las versiones del producto que se distribuyen a los usuarios, es que las empresas opten en algún momento por imponer a sus desarrolladores la aplicación de una metodología rigurosa para dar frente a los cambios bruscos y frecuentes en el producto, donde es casi imposible determinar la ocurrencia de estos cambios dado los vaivenes del negocio; pero por todos estos factores influyentes en la liberación de un producto final, es que las empresas se ven obligadas a permitir en algún momento que sus desarrolladores dejen de aplicar rigurosamente la metodología, dada la necesidad de no entorpecer la productividad de las versiones.

El primer problema con el proceso de desarrollo de aplicaciones es lograr que sea comprendido por el equipo de trabajo. Luego viene el problema siguiente: conseguir que sea aceptado.

Pueden surgir interrogantes como las plateadas por (Spaces 2005): ¿Por qué ese “malestar” con las metodologías? ¿Cuáles son las razones por las cuales las necesidades de cierta rigidez en los procesos terminan endureciendo la productividad? ¿Acaso existe un punto de equilibrio entre formalismo y agilidad?

Para dar respuesta a las preguntas anteriores, (Spaces 2005) dice que a principios de los '90 *Microsoft* comenzó a recopilar a nivel interno las mejores prácticas en términos de procesos de desarrollo de software, no con la intención de establecer una metodología, sino con la idea de tener una colección de prácticas individuales de “lo que funciona”, aplicables dentro de determinados contextos.

Microsoft Solution Framework (MSF) es un conjunto de procesos, principios y probadas prácticas de desarrollo de software, altamente personalizables, escalables, totalmente integrados y pensados para brindar el tipo de guía que el usuario desea a la hora de satisfacer sus necesidades.

MSF no es más que la unión de las mejores prácticas para la administración de proyectos. Esta no es una metodología “rígida” que nos sirve para administrar el mismo, sino una serie de modelos que son adaptables a cualquier tipo de proyecto de tecnología de información.

Como concepto suele parecer interesante lo planteado sobre MSF en los dos párrafos anteriores, pero si encima se aplica con las herramientas adecuadas, integradas a los mismos mecanismos por los cuales se generan los productos de trabajo, los resultados son impresionantes. Sin embargo, la aplicación de una metodología obviando el uso de herramientas apropiadas e integradas, abre las puertas a la posibilidad de comenzar a incumplir con las formalidades del proceso de desarrollo.

2.1.1 MSF ágil para la gestión de proyectos informáticos

Tanto o más importante que la tecnología que se usa en un proyecto informático es el proceso o método que se usa para controlar el proyecto. Sin duda, es una receta para el desastre no usar ninguna metodología pero, por otro lado, los años han demostrado que usar una metodología muy "protocolaria" (en el sentido de que exige una serie de pasos y documentos largos y detallados) puede resultar improductivo, causar retrasos y desgastes. Paradójicamente, una metodología de proyectos detallista y demandante puede ser tan peligroso como no usar ninguna metodología. MSF ágil se coloca justamente en el centro de estos dos extremos y acata los diferentes aspectos que se deben manipular en un proyecto informático, que son:

- La constitución y responsabilidades del equipo
- Las fases y entregables del proceso de desarrollo
- La vigilancia y mitigación continua de los riesgos
- La administración distribuida de responsabilidades y la escalabilidad del modelo.

Obviamente, MSF ágil establece la necesidad de realizar reuniones y crear documentos y otros entregables, pero tiene exigencias razonables y livianas, lo que aumenta las posibilidades de que el equipo informático promedio no deserte del uso de la metodología.

La utilización de la metodología de gestión de proyectos MSF ágil permite introducir los conceptos necesarios para mejorar el resultado y garantizar el éxito del proyecto.

MSF ágil ayuda a implantar soluciones de tecnología según (colombia 2000):

- Manteniendo siempre el enfoque al usuario. Es decir, ayudándolo a asegurar que la solución implantada realmente es lo que el usuario necesita, que mejorará el desempeño del usuario, y no una que va a ser desechada y olvidada al momento de su liberación.
- Proporcionando elementos valiosos a las inevitables preguntas: ¿Cuáles elementos van en el cliente?, ¿Cuáles en el servidor?
- Permitiendo desarrollar siguiendo una filosofía de reutilización de múltiples componentes, lo cual reduce el tiempo necesario para desarrollar nuevas aplicaciones, garantiza la uniformidad e interoperabilidad entre las mismas, y las hace mucho más flexibles para incorporar los cambios que sean necesarios en el futuro.
- Estableciendo un equipo de trabajo balanceado, con tareas y objetivos claramente definidos, que permitan no solo desarrollar buenos sistemas, sino también saber en todo momento cuál es el grado real de avance del proyecto, y cuáles son los riesgos que se corren si se decide introducir modificaciones al mismo una vez que el desarrollo se ha iniciado.

Conceptos fundamentales:

Roles

Se denomina rol, según (Microsoft 2007), al papel que ejerce un actor en una actividad o proyecto.

La (GUIDANCE 2005) plantea que en MSF ágil, los roles se agrupan por “equipo de pares” para representar los componentes del sistema implicados con la producción, el uso y el mantenimiento del producto. Cada rol es responsable de representar las necesidades específicas de su trabajo y ninguno es más importante que otro, lo que explica el uso del término “pares”, pudiendo nombrarse también “equipo de iguales” para una mejor comprensión. Estos criterios proporcionan el equilibrio necesario para asegurar que el equipo produce la solución correcta.

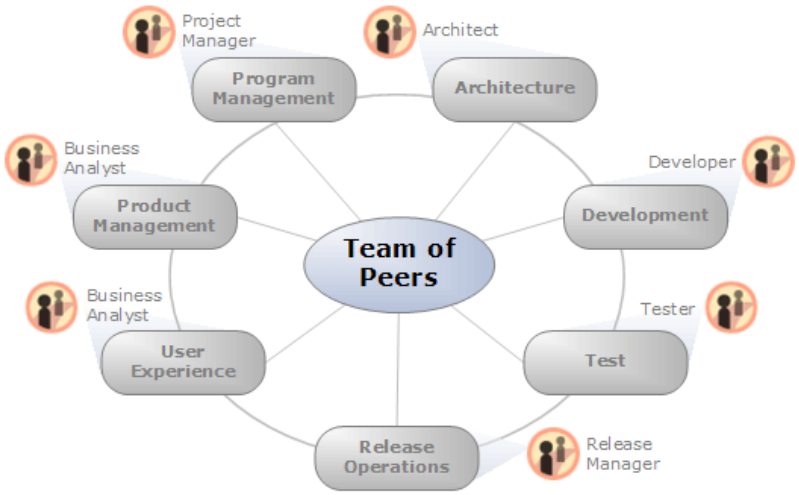

Fig. 10 Roles de MSF (GUIDANCE 2005)

Los roles que intervienen dentro del proceso MSF ágil son seis fundamentalmente: analista de negocio, jefe de proyecto, arquitecto, desarrollador, *tester* y administrador de *releases*.

No necesariamente esos roles deben ser asignados a personas diferentes. Si bien es cierto que determinados roles pueden ser desempeñados por una misma persona, hay otras combinaciones en las que eso no es lo deseable.

Work items

Las actividades atómicas en MFS ágil se asignan mediante el concepto de *work item,* que se puede describir como un registro de la base de datos que *Visual Studio Team Foundation Server* utiliza para rastrear trabajos asignados y el estado del mismo. El proceso MSF para el desarrollo ágil del software define *work items* para asignar y seguir el trabajo. La base de datos común del *work item* y el almacén de métrica hacen posible dar respuesta a preguntas sobre la “salud” del proyecto en tiempo real.

También según (GUIDANCE 2005), en el proceso MSF ágil los *work items* que por lo general se encuentran son los que siguen:

- Escenario

Tipo de *work item*, que va registrando una sola trayectoria de la interacción del usuario con el sistema. Por ejemplo, mientras el usuario procura alcanzar una meta, el escenario registra las medidas específicas que se tomarán para alcanzar dicha meta. Algunos escenarios registrarán una trayectoria acertada y otros no. Al escribir escenarios en un proyecto determinado, se debe ser muy específico, ya que existen muchas trayectorias posibles.

- Requerimiento de Calidad de Servicio

Este *work item* documenta las características del sistema tales como: funcionamiento, carga, disponibilidad, tensión, accesibilidad, utilidad y capacidad de mantenimiento. Estos requisitos toman generalmente la forma de alertas en cuanto a cómo el sistema debe funcionar.

- Riesgo

Un aspecto esencial de la administración de proyecto es identificar y manejar los riesgos inherentes al mismo. Un riesgo es cualquier acontecimiento o condición probable que pueda tener un resultado potencialmente negativo en el futuro del proyecto. El *work item* de tipo riesgo, documenta y sigue las debilidades técnicas o de organización de un proyecto. Cuando se requiere una acción concreta, estos riesgos pueden traducirse a tareas para atenuar el riesgo. El ambiente debe ser tal que los individuos que identifican riesgos puedan hacerlo sin “miedo” a la consecuencia por dejar expuestas visiones tentativas o polémicas. Los equipos que crean un ambiente positivo en la administración de riesgos aciertan más en identificar los problemas que puedan afectar el desarrollo futuro del proyecto que aquellos que no lo instauran.

- Tarea

Un *work item* de tipo tarea comunica la necesidad de hacer un trabajo determinado. Cada rol tiene sus propios requisitos para una tarea. Por ejemplo, un *tester* utiliza tareas de los casos corrientes de prueba. Una tarea se puede utilizar también para señalar regresiones o para sugerir que la prueba de exploración

sea realizada. También se puede usar genéricamente una tarea para asignar el trabajo por separado dentro del proyecto.

- Bug

Un *bug* es un *work item* que comunica un problema potencial existente o que ha existido en el sistema. La meta de abrir un *bug* es divulgar exactamente los *bugs* de una manera que permita que el lector entienda el impacto completo que puede acarrear el problema. La descripción en el informe de un *bug* debe hacer fácil la tarea cuando éste es encontrado, permitiendo así que sea reproducido fácilmente. Los resultados de la prueba que se realice deben demostrar claramente el problema. La poca claridad y la legibilidad de esta descripción afecta a menudo la probabilidad de que el *bug* sea fijo.

2.2 Principios de MSF ágil

Los principios para MSF ágil que define (GUIDANCE 2005) son:

- ***Aprender de todas las experiencias -*** la validación del cliente es a menudo la diferencia entre el valor de negocio verdadero y ficticio. Entender el asunto del valor y comunicarlo con eficacia es un factor dominante del éxito.
- ***Alentar comunicaciones abiertas -*** para maximizar la eficacia individual de los miembros y optimizar la misma en el trabajo, la información tiene que estar fácilmente disponible y compartida.
- ***Trabajar hacia una visión compartida -*** la visión compartida asegura que todos los miembros del equipo negocien mientras se avanza en la construcción del producto. Se mejora cuando las decisiones no son arbitrarias.
- ***Invertir en calidad -*** la calidad requiere la prevención del error y la verificación de la solución. Se utilizan prácticas tales como análisis del código y revisiones por pares para prevenir errores así como maximizar pruebas para encontrarlos. Todos los roles son responsables de la prevención y verificación de los errores.
- ***Permanecer ágil, esperar el cambio -*** mientras más veces una organización intente maximizar el impacto del negocio de una inversión tecnológica, más se aventura en nuevos ámbitos

intrínsecamente inciertos y se adapta a los cambios como resultado de la exploración y experimentación de nuevos métodos. Exigir certeza en un ambiente que cambia es poco realista y conduce a resultados funestos.

- ***Otorgar poder a los miembros del equipo -*** se debe confiar en el equipo al crear un producto de alta calidad mientras el mismo evoluciona. Cada cambio se debe hacer en el contexto de la creencia que el producto se encuentre listo en cualquier momento. Los liberadores deben poner a funcionar constantemente el producto y la compañía debe ensayar nuevos *releases* del mismo.
- ***Concentrarse en agregar valor al negocio -*** planificar, ejecutar y medir el progreso y la velocidad basándose en el aumento de la entrega de valor al cliente. Se deben reducir al mínimo esas actividades que no agreguen valor al cliente porque son inútiles. Se utilizan las iteraciones para mantener la cadencia de los productos del trabajo que su cliente puede evaluar. Se debe delegar trabajo partiendo de un miembro del equipo a otro, tan eficiente como esto sea posible.

2.3 Ciclos e Iteraciones

En el portal del (GUIDANCE 2005) son abordados además los ciclos e iteraciones de MSF ágil.

Los ciclos describen la frecuencia con que se realizan las actividades o trabajan los productos producidos y puestos al día. Los ciclos están sobre la ejecución del proyecto y de sus tareas.

La iteración de proyecto es una jerarquía de proceso de los eventos del ciclo de vida del proyecto. Cuando se crea por primera vez el proyecto de equipo o *Team Project*, su ciclo de vida se expresa en términos de iteraciones. MSF ágil define las iteraciones como un periodo fijo de tiempo en el cual programar tareas. Las iteraciones generalmente son numeradas consecutivamente y siguen una a otra de manera continua. El desarrollo iterativo se presentó como antídoto al desarrollo secuencial en "cascada".

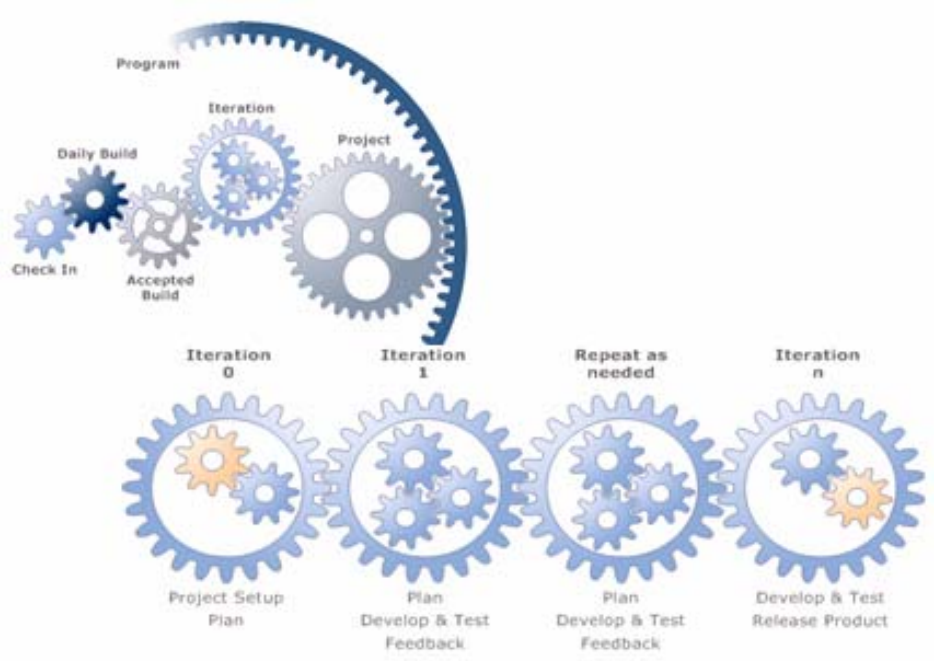

Fig. 11 Ciclos e iteraciones en MSF. (Medela 2006)

Con independencia de la metodología utilizada, lo que todas las metodologías populares tienen en común, es que al principio de una iteración se planifica el trabajo que se va a abordar a corto plazo y al final de una iteración se sacan conclusiones de la iteración que recién se terminó. Este proceso de “mirar atrás” se debe hacer siempre con el afán de que sirva para avanzar positivamente. El final de una iteración siempre coincide con el comienzo de otra, de manera que en ese punto lo que se ve son los resultados de una iteración y estos resultados son los que deben guiar a la hora de planificar la siguiente iteración.

Evidentemente, el proceso descrito, se puede aplicar de manera anidada. Una iteración puede contener a otras. Esto es así porque al principio de un proyecto es habitual no tener un alto nivel de detalle sobre cuáles serán las iteraciones, y por lo tanto se pueden definir 3 ó 4 iteraciones que luego se irán refinando en sub-iteraciones. Otro motivo para anidar iteraciones es la posibilidad de tener, en desarrollos grandes, diferentes equipos en diferentes iteraciones, pero esto no es lo usual.

MSF es un proceso iterativo e incremental. El desarrollo iterativo se ha definido generalmente como "la técnica en la cual la definición, el diseño, la puesta en práctica y la prueba de los requisitos ocurren solapadamente dando por resultado la terminación incremental del producto de software total."

¿Por qué iterar?

Hay muchas discusiones, según (Sam Guckenheimer 2006) que obligan a poner en marcha el desarrollo iterativo, por ejemplo:

- **Administración de riesgos:** El resultado deseado es incognoscible por adelantado. Para manejar riesgos, se deben probar o refutar los requisitos y diseñar incrementalmente poniendo elementos del sistema en ejecución, comenzando con los elementos de alto riesgo.
- **Economía:** En un clima de negocio incierto, es importante repasar prioridades con frecuencia y tratar las inversiones como si fueran opciones financieras. Mientras más flexibilidad se gane a través de la rentabilidad temprana y de puntos de comprobación frecuentes, más valiosas llegan a ser las opciones.
- **Foco:** Tratando el trabajo por lotes en iteraciones pequeñas, los miembros del equipo centralizan su labor, y así los analistas pueden hacer un mejor trabajo con los requisitos, los arquitectos con el diseño y los liberadores con el código por solos citar algunos ejemplos.
- **Motivación:** La ocurrencia de la motivación en un equipo de software propicia que los lanzamientos tempranos del producto (demos) se realicen de manera efectiva. No hay una cifra numérica de revisión de especificaciones que pueda sustituir el valor del trabajo iterado.
- **Teoría de control:** Las iteraciones pequeñas permiten reducir el margen de error en las estimaciones y proporcionar la regeneración rápida sobre la exactitud de los planes del proyecto.
- **Implicación del cliente:** Los *stakeholders* ven resultados rápidos y se enganchan más al proyecto, ofreciendo más de su tiempo, compenetración y financiamiento.
- **El aprendizaje constante:** El equipo completo aprende de cada iteración, mejorando la exactitud, la calidad y la eficacia del producto.

No obstante, el desarrollo iterativo debe ser adoptado por las organizaciones ya que en la práctica, requiere que el equipo y los managers del proyecto tengan una vista total del trabajo que harán, de la capacidad de supervisar y de ceder prioridad con frecuencia en los límites cortos de la iteración. Ésta puesta al día frecuente requiere una reserva fácilmente visible, de preferencia con la colección de datos automatizada, tal como la base de datos de *work items* que VSTS proporciona.

Según plantea el paradigma *value-up* del desarrollo iterativo, hay muchos ciclos en los cuales las actividades se solapan en paralelo. Una iteración es el número fijo de semanas, a veces llamado "caja del tiempo", utilizado para programar un sistema de tareas. Usando iteraciones, el intervalo en el cual se pensaron escenarios mide el flujo del valor, determinan el proceso real, examinan embotellamientos y hacen ajustes.

Cambio de paradigma

La inherente incertidumbre en los procesos de desarrollo de software hace difícil estimar las tareas correctamente, las cuales se crean con una alta variación de exactitud en las estimaciones. Una idea falsa común es pensar que la variación es aceptable porque las variaciones positivas y negativas hacen un promedio; sin embargo, debido a que los proyectos de software son cadenas largas de eventos que dependen entre sí, la variación por sí misma acumula un retraso.

A menudo, el desarrollo está en una pequeña parte del proyecto concerniente al análisis de negocio, a la gerencia de proyecto y a las pruebas. Típicamente, los proyectos antes mencionados tienen una variabilidad más baja que los nuevos proyectos del desarrollo, así que la sabiduría de trabajos basada en "caminos y puentes" funciona mejor para ellos que para el nuevo desarrollo.

Desde 1992, ha habido un desafío cada vez mayor al paradigma *work-down* sobre el proceso del software. Ningún término ha capturado el paradigma que emergía y como sucede con nuevos prototipos, la opinión del *value-up* ha aparecido ya en ajustes y comienzos de proyectos.

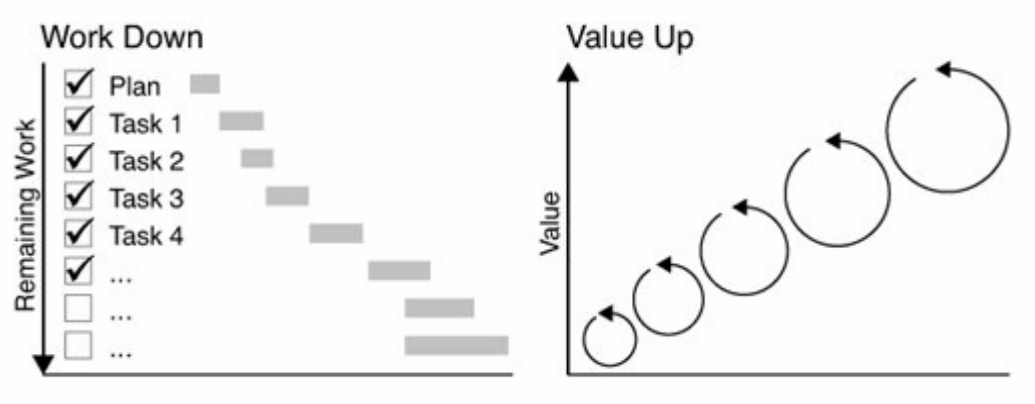

Fig. 12 Paradigmas *work-down* y *value-up* (Sam Guckenheimer 2006)

La diferencia entre *work-down* y *value-up* está en la medida primaria. *Work-down* trata al proyecto como acción fija de las tareas que necesitan la terminación para un cierto costo y las medidas de gasto contra esas tareas. El *value-up* mide el valor entregado en tiempo en cada punto y trata las entradas como flujos variables o más bien como una acción fija.

Se pueden apreciar más en detalle las diferencias entre ambos paradigmas a través de la siguiente tabla:

Bases	Work-down	Value-up
Planes y cambios de procesos	Planificar y diseñar son unas de las actividades más importantes a realizar correctamente. Se necesita hacer esto inicialmente para prever cambios.	Aceptar los cambios. Se debe invertir bastante en planeamiento y diseño para entender los riesgos y manejar su incremento.
Medida primaria	Tareas terminadas. Al saber los pasos para alcanzar la meta final, se puede medir cada entregable intermedio y computar el valor ganado.	Sólo entregables (software de trabajo, documentación terminada, etc.) que el cliente estime. Se necesita medir el flujo de las corrientes del trabajo que agreguen valor al cliente.
Definición de calidad	Se necesitan especificaciones precisas desde el principio que brinden conformidad.	No existe valor del cliente hasta que no se le entrega un software inicialmente. Las opciones de subsistencia se abren, se optimizan para la entrega continua, y no se especifican demasiado.

Aceptación a la variación	Tareas que pueden ser identificadas y estimadas en un determinado momento. No hay que pagar atención a la variación.	La variación es parte de todo el flujo de procesos. Para predecir el alcance se necesita entender y reducir la variación.
Work products intermedios	Documentos, modelos, y otros artefactos intermedios son necesarios para descomponer las tareas del diseño y del plan, y proporcionar la manera necesaria de medir el progreso intermedio.	La documentación intermedia debe reducir al mínimo la incertidumbre y la variación para mejorar el flujo.
Localizar y vigilar de cerca los problemas	Los apremios del tiempo, de recursos, de funcionalidad, y de calidad determinan lo que se puede alcanzar. Si se ajusta uno se deben ajustar los demás.	Los apremios pueden o no relacionarse con tiempo, recursos, funcionalidad o calidad. Identifique el embotellamiento primario para asegurar un flujo más liso.
Aproximación a la realidad	La gente necesita de supervisión y de ser comparada con los estándares. La gerencia debe utilizar incentivos para recompensar a los individuos para su funcionamiento de acuerdo a los planes previstos.	El orgullo de la ejecución y el trabajo en equipo son motivadores más eficaces que los incentivos individuales. Transparencia digna de confianza, donde todos los miembros del equipo pueden apreciar los datos del funcionamiento.

Tabla 3. Comparación entre paradigmas (Sam Guckenheimer 2006)

A veces se tiende a definir iteraciones en función de hitos y no de un horizonte temporal. Esto puede resultar siendo una mala práctica. Es conveniente realizar periódicamente actividades explícitas de control, aprendizaje y planificación detallada. A la hora de construir iteraciones es conveniente establecer una meta. La duración de las iteraciones es fija en algunas metodologías, pero en MSF ágil por ejemplo, se decide como parte de la planificación de la misma.

Sin perder de vista lo que se logró en la iteración anterior, en la planificación de una iteración se debe:

- Decidir cuánto durará
- Establecer qué trabajos se realizarán
- Estimar en detalle estos trabajos

- Asignar quién realizará los mismos
- Establecer una meta para la iteración

2.4 Autoridad en MSF ágil

La autoridad en MSF ágil, enunciada en (GUIDANCE 2005), se refiere al control del tiempo y el dinero concerniente al flujo del valor. MSF para el desarrollo ágil de software define cinco puntos de comprobación de autoridad, cada uno de los cuales se centra en una pregunta específica que debe ser contestada.

1. **Objetivos repasados** (¿Conoceremos el mínimo nivel de aceptación de funcionalidad?).
2. **Progreso determinado** (¿Estamos haciendo el progreso necesario para resolver nuestro plazo?).
3. **Pruebas evaluadas** (¿Mantendremos un nivel apropiado de calidad?).
4. **Riesgos identificados y mitigados** (¿Direccionaremos el proyecto y las técnicas de riesgo?).
5. **Despliegue listo** (¿Hemos diseñado y creado una solución lista para el despliegue?).

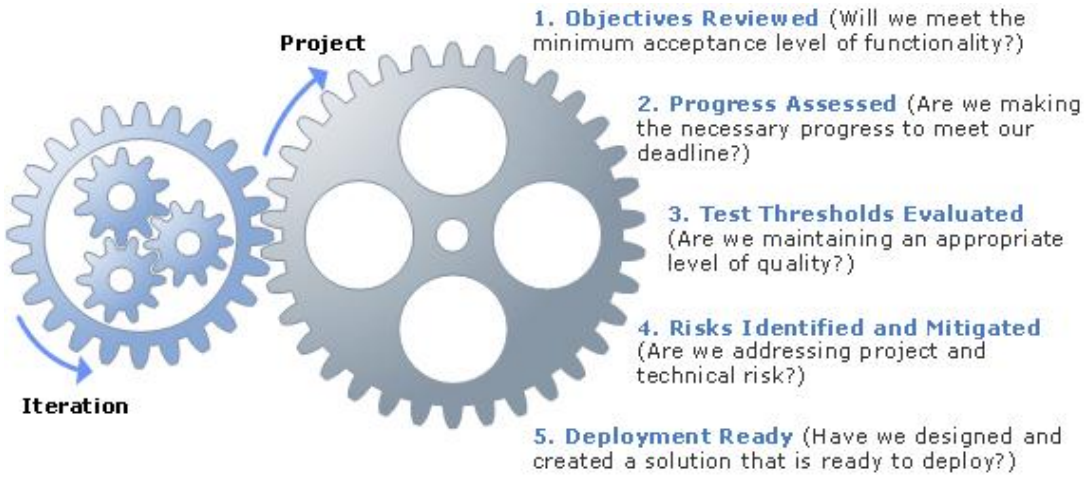

Fig. 13 Puntos de control de autoridad en MSF ágil (GUIDANCE 2005)

Las actividades y flujos de trabajo que conducen a los puntos de comprobación se llaman “pistas”. Es importante no confundir las pistas para la autoridad con la organización del trabajo que se agrupa en iteraciones y dentro los ciclos más finos.

Las pistas siguen las actividades que conducen a los puntos de comprobación de la autoridad dando tratamiento a las preguntas, que se refieren al gasto del tiempo y del dinero. Las pistas no están necesariamente sobre la ejecución de las tareas o las interrupciones del trabajo que se manejan en ciclos, sino que pueden trasladarse considerablemente y hay una regeneración continua entre las actividades en diferentes pistas.

2.5 Modelos de MSF ágil

En el capitulo anterior fueron enunciados los diferentes modelos de MSF (que pueden ser usados individualmente o combinados). Estos modelos son adoptados por la metodología ágil y es por ello que son aplicables a MSF ágil. En el presente capítulo solo serán objeto de profundización los modelos de equipos y de procesos.

2.5.1 Modelo de equipos

El modelo de equipos de trabajo de MSF ágil facilita la estructuración de equipos para construir o implantar soluciones eficientes y de manera oportuna proyectadas hacia el mejoramiento continuo.

Este modelo se define como un equipo no jerarquizado cuyos integrantes trabajan en los seis roles interdependientes y cooperativos abordados al inicio del capítulo. Por ejemplo, los líderes de cada equipo son responsables de la administración, guía y coordinación mientras que los miembros del equipo se centran en llevar a cabo sus misiones.

La meta principal del equipo de trabajo es entregar un sistema o solución de calidad, o sea, realizar un software y que el cliente quede satisfecho con este, entregarlo en tiempo, gastando así la menor cantidad de recursos y coste, preparar al usuario para que sepa cómo usar el software, lo que implica que para que se cumplan todos estos objetivos el equipo de trabajo tiene que lograr un mejor entendimiento y trabajar

en una misma línea al tiempo que se responsabilizan con la(s) tarea(s) asignada(s). Por todas estas ventajas es que se aplica este nuevo modelo de equipos de MSF ágil.

En el modelo de equipos, las características de los miembros del equipo, ya sean la visión compartida, la comunicación, la delegación, la responsabilidad o el entendimiento, permiten la obtención de un producto con calidad en conjunto con un cliente satisfecho con el trabajo realizado. Las metas de calidad sobre las cuales se concentran los esfuerzos del equipo de desarrollo son:

- Cumplir con las expectativas del usuario.
- Entregar el sistema o solución dentro de las restricciones del proyecto (tiempo, recursos, costos).
- Identificar todos los problemas o riesgos de importancia para el usuario y manejarlos de forma oportuna.
- Asegurar que el usuario final sepa cómo usar el sistema.
- Asegurar una implantación/replicación del sistema sin contratiempos.

Este modelo de MSF ágil puede escalarse de dos maneras principales, plantea (Microsoft 2002):

1. Abstrayendo las funciones del equipo como un grupo de responsabilidades funcionales, en lugar de hacerlo como descripciones de trabajos específicos. De esta manera, las responsabilidades de cada función no están sujetas a los límites de una sola persona. Una función puede ampliarse a grupos de funciones y cada uno de los miembros se especializa en un grupo más específico de responsabilidades. Una o más personas pueden llevar a cabo estas funciones más especializadas.

2. Usando equipos de producto y calidad y equipos funcionales en diversas combinaciones para crear cualquier número de posibles estructuras de grandes equipos.

Este modelo además, propicia la agilidad para hacer frente a nuevos cambios involucrando a todo el equipo en las decisiones fundamentales, asegurándose así que se exploran y revisan los elementos de juicio desde todas las perspectivas críticas.

El modelo de equipos se basa en la premisa de que cada función presenta una perspectiva única en el proyecto y que ninguna persona por sí misma puede representar de una manera satisfactoria todos los objetivos de calidad de todas las funciones. Sin embargo, el cliente necesita una sola fuente de

información que cuente con autoridad sobre el estado del proyecto, las acciones y los problemas actuales. Para resolver este dilema, el equipo de trabajo debe combinar una línea clara de responsabilidades hacia el cliente con la responsabilidad compartida con el propósito de lograr el éxito general en el proceso de desarrollo.

En resumen las funciones del equipo de MSF ágil comparten responsabilidades en muchos aspectos de la administración del proyecto como por ejemplo, la administración del riesgo, del tiempo, de la calidad, del planeamiento, de la programación, de la selección de los miembros del equipo y de los recursos humanos.

2.5.2 Modelo de procesos

MSF ágil también tiene su modelo de procesos, que se puede utilizar en la realización del proyecto y que hace posible avizorar el cambio y controlarlo, por ende, un modelo de procesos dirige el orden de las actividades del proyecto y representa el ciclo de vida de dicho proyecto.

Algunos modelos de procesos cuyo surgimiento fue anterior a este que propone MFS ágil eran estáticos, y otros no permitían puntos de comprobación. El modelo de “cascada” y el de “espiral” son ejemplos de esos arcaicos modelos.

Como modelo de procesos superior, el de MSF ágil a través de su estrategia iterativa en la construcción de productos del proyecto, suministra una imagen más clara del estado de dichos productos en cada etapa sucesiva. El equipo puede identificar con mayor facilidad el impacto de cualquier cambio y lidiar con él de manera efectiva, minimizando los efectos colateralmente negativos y optimizando los beneficios.

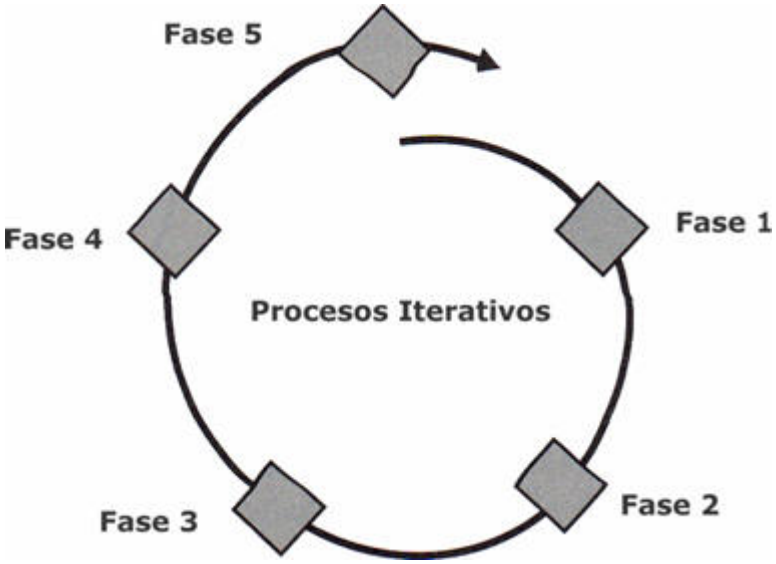

Fig. 14 Procesos de MSF ágil por fases (Ávila 2005)

Este modelo de MSF ágil se encarga de planificar y controlar el proyecto dirigiendo sus objetivos a resultados específicos, es iterativo y se basa en puntos de revisión.

Para profundizar más en este modelo se debe conocer que:

- Consta de cinco fases distintas (ver MSF Capítulo I)
- Se basa en obtener una visión de los próximos proyectos o sistemas, prioriza el análisis de riesgos e implementa incrementalmente con puntos de revisiones frecuentes que relaciona el trabajo del equipo con las expectativas de los usuarios a lo largo de todo el proyecto. (Ivar Jacobson 1999)
- Los cuatro puntos de revisión principales están precedidos por una fase principal dentro de la cual ocurren puntos de revisión internos y se pueden generar productos entregables.(Ivar Jacobson 1999)

También fue planteado por (Ivar Jacobson 1999) que los puntos de revisión no necesariamente son puntos de congelación. En cada uno de ellos los entregables pueden colocarse bajo un proceso de control de cambios. Cada punto de revisión establece un punto de partida que permite en forma confiable que el equipo planee y continúe hacia el siguiente punto de revisión.

En resumen, el modelo de procesos de MSF ágil se aplica para encontrar un balance en cuanto a alcance, tiempo y recursos de un proyecto y a su vez puede ser aplicado a una amplia variedad de proyectos sin pérdida del flujo del proceso.

2.6 Aspectos principales del desarrollo a estar en la mente de cada miembro (Mindsets)

Según se enuncia en (GUIDANCE 2005), se puede decir también que MSF ágil es mucho más que un sistema de actividades a seguir ya que intenta crear una cultura que fomente proyectos acertados. Los aspectos que deben tener en mente cada uno de los miembros de un equipo no son más que una colección de valores que determinan cómo los individuos interpretarán y responderán a las situaciones.

Estos aspectos permiten proyectarse, a cada miembro del equipo, en la toma de decisiones y a darle prioridad al trabajo. Enseñan a los miembros del equipo a obrar recíprocamente con los demás, así como con otros *stakeholders*.

Fig. 15 *Mindsets* y principios de MSF ágil (GUIDANCE 2005)

Los *mindsets* mostrados en la figura anterior se describen a continuación:

1. ***La calidad es definida por el consumidor* -** los clientes satisfechos son la prioridad número uno para cualquier buen equipo de desarrollo. Esta satisfacción incluye a clientes internos y externos. La implicación del cliente se debe maximizar al mayor grado posible asegurándose que las expectativas del cliente estén alineadas con el proyecto. La aplicación de técnicas que sustenten la localización y organización de las expectativas del cliente incluyen la divulgación sobre la reserva, estructurar un pequeño grupo de medidas, y entregar productos de calidad.
2. ***Orgullo de ejecución* -** tomar orgullo en contribuir a una solución es una parte importante de crear un producto de calidad. La motivación y sentido de la responsabilidad son resultados de este orgullo. Crear el orgullo de ejecución es una responsabilidad individual y organizacional. Las técnicas que ayudan a mantener este sentido de propiedad incluyen dar nombres del código para identificar claramente el proyecto y el equipo.
3. ***Equipo de pares* -** el *mindset* de "equipo de pares" pone igual valor en cada miembro. Este *mindset* requiere la comunicación sin restricción entre los roles, transparencia, y una sola reserva visible. El resultado es el incremento de la responsabilidad del equipo y una comunicación eficaz.
4. ***Entrega frecuente* -** nada establece más credibilidad como la entrega frecuente. Con entregas frecuentes se prueba y se mejora el proceso y la infraestructura. Los riesgos, los errores y los requerimientos que faltan se detectan temprano. La regeneración puede ser proporcionada cuando puede marcar diferencia. El éxito de una entrega frecuente está en mantener pequeños grupos de medidas, trabajar en entregas que se efectúen en tiempo y eliminar opciones por decisiones prematuras. La entrega frecuente es un modo de inculcar disciplina a un equipo de desarrollo de software.
5. ***Buena voluntad de aprender* -** en cada proyecto, el ambiente y el equipo son únicos, pues cada proyecto y sus iteraciones crean una oportunidad única para aprender. Sin embargo, no es posible aprender sin la regeneración y la reflexión honesta. A menos que haya un ambiente de apoyo que fomente valor y seguridad personal, la regeneración será limitada.

6. ***Obtención de especificaciones tempranas -*** muchos proyectos pierden el tiempo en vez de abordar problemas solubles. Este *mindset* obliga a tomar una medida a la vez, aprendiendo de lo específico más que de lo extracto. Las técnicas que apoyan este *mindset* incluyen personajes, escenarios, diseño para el despliegue y casos de prueba.
7. ***Calidad del servicio -*** este *mindset* desarrolla los planes basados en la experiencia del cliente. La idea no es que las calidades del servicio (funcionamiento y seguridad) sean consideradas atrasadas al final del proyecto sino a través de él. Con este *mindset* se pone fin a exaltaciones implícitas en la calidad explícita de los requisitos del servicio.
8. ***Ciudadanía -*** el *mindset* de ciudadanía se centra en la administración de los recursos corporativos, del proyecto y el automatizar. Se manifiesta de muchas maneras para conducir el proyecto de forma eficiente con vistas a optimizar el uso de los servicios web. Las técnicas que apoyan la ciudadanía incluyen reasignar *bugs* con toda la información necesaria para abastecerse de buenas estimaciones de la cantidad de trabajo que una tarea o prueba llevará.

2.7 Visual Studio Team System (VSTS)

El VSTS integra la dirección de proceso probada, las arquitecturas formales, y las herramientas del ciclo de vida para desplegar con éxito soluciones en la plataforma de *Windows*. El VSTS incluye además los aceleradores de solución que ayudan a organizaciones a mejorar la previsión y confiabilidad de entregar soluciones. El *Visual Studio 2005 Team System* proporciona las herramientas para apoyar a todo el equipo de desarrollo del software.

Las herramientas de dirección de proyectos del VSTS permiten una mejor planeación, programación, colaboración, comunicación, divulgación y control del proceso. Estas herramientas se integran con el ambiente de desarrollo integrado del *Visual Studio* (IDE), *Office*, los servicios de *SharePoint*, y los servicios de reportes del *SQL Server 2005*.

Estos servicios ofrecen una personalización y extensibilidad casi ilimitada. Además incluye plantillas de procesos que brindan la posibilidad de configurar los nuevos proyectos y documentar lo básico sin escribir siquiera una línea de código o HTML. Estos archivos se basan en XML.

VSTS incorpora funcionalidades básicas e intuitivas a todas las áreas de gestión del ciclo de vida de las aplicaciones para ayudar a resolver problemáticas de negocio y retos de desarrollo. Desde el modelado inicial y el diseño mediante el desarrollo, la gestión de cambios, las pruebas y la implantación, VSTS incorpora conectividad, ya que ha sido construido desde cero para trabajar con él como un sistema integrado.

Los datos cuantitativos por parte de VSTS incorporan información sobre el control de calidad, código y los tiempos de ciclo, y ofrecen métricas para evaluar el éxito informático de forma cualitativa. Así, las organizaciones pueden determinar qué recursos deben proporcionar para un soporte más eficaz de los proyectos de desarrollo.

Los reportes de proyectos que se integran en VSTS son visibles a través de *Windows SharePoint Services* (WSS); *Microsoft Excel* y *Microsoft Project* generan visibilidad del ciclo de vida y el proceso de desarrollo. Utilizando estas capacidades, los equipos pueden tomar decisiones adecuadas sobre la elección de los recursos a emplear para un proyecto específico, y decidir además sobre el alcance y las políticas que tendrá el mismo.

2.8 Visual Studio Team Foundation Server (TFS).

El TFS es una colección de las tecnologías de colaboración que apoyan un esfuerzo del equipo para entregar un producto determinado.

Aunque las tecnologías del TFS son empleadas generalmente por un equipo del software para construir un producto, pueden también ser utilizadas en otros tipos de proyectos.

Si se es novato en la utilización del TFS, se puede comenzar a aprender por los conceptos y la ayuda de la biblioteca MSDN de *Microsoft*, conseguir una descripción de la arquitectura del sistema al que se quiere aplicar esta herramienta, así como características dominantes de dicha arquitectura.

Visual Studio Team Foundation Server proporciona una plataforma cohesiva para el desarrollo de software y la colaboración al incorporar elementos para la gestión de procesos, seguimiento de *work items*, control de código fuente, automatización de versiones, pruebas y controles de equipo con la generación de reportes de modo centralizado.

Las vistas sobre el trabajo de desarrollo se generan en función de los roles de los miembros involucrados en el proceso de desarrollo (arquitecto, desarrollador y *tester*). Todos los *work items* se localizan en una única base de datos para permitir una coordinación más precisa.

Personalizar y configurar el TFS son tareas a cumplimentar seguidamente que se instale el/los servidor/res de *Team Foundation* para adecuarlo al entorno de trabajo que el equipo/proyecto requiera. Esta configuración resulta básica para poner en práctica todas las características del TFS y de esta forma conocer resultados inmediatos del trabajo diario.

El TFS, ya sea a nivel de pantallas de configuración o manejando las APIs del sistema, permite una total personalización de su entorno, lo cual constituye una ventaja de esta herramienta.

También se puede considerar una ventaja el hecho de cómo se trata la seguridad en *Team Foundation*, ya que está basada en usuarios y grupos que se pueden manipular para lograr implementar un modelo de seguridad que permita a los usuarios acceder a los datos y la funcionalidad que ellos requieran, mientras se protege la información principal del proyecto.

El modelo de seguridad de TFS es un poco complejo ya que aparte de los permisos asignados dentro del *Team System*, esta la asignación de privilegios y usuarios en otros aplicativos. Existen tres grandes grupos para otorgar permisos y estos son:

1. Grupo de TFS
2. Grupo de Servicios de WSS
3. Grupo de Servicios de Reportes

Aunque parezca trivial, esta tarea puede ser propensa a errores, para evitar este problema se puede utilizar el aplicativo llamado Herramienta para la Administración de TFS (*Team Foundation Server Administration Tool)* para toda la comunidad de desarrollo.

Elementos del TFS

Visual Studio Team Foundation Server habilita el marco de trabajo para el VSTS. Entre los componentes se incluyen el seguimiento de los *work items*, un portal de proyectos (basado en WSS), un almacén de datos de métricas, el control de código fuente, la automatización de versiones, recomendaciones sobre procesos MSF y servicios de integración. Entre estos procesos se incluyen MSF para el desarrollo ágil del software (para estrategias más ligeras e iterativas) y MSF para el Proceso de mejoras de CMMI (aproximaciones más estructuradas y rigurosas). Estos procesos son personalizables a través de campos, formularios, estados y reglas utilizando el asistente de Dirección de Plantillas de Procesos y la plantilla CMMI mencionada anteriormente, es un súper conjunto de la plantilla *Agile*. El asistente de creación de proyectos ayuda a lanzar proyectos y el "editor de procesos" permite a los usuarios enmarcar los proyectos en el contexto de las mejores prácticas.

Interactividad entre el VSTS y TFS a través de los procesos de MSF ágil.

El VSTS proporciona una plantilla de la metodología para fijar el proceso. Esta plantilla contiene los *work item*s: escenario, requisitos de calidad del servicio, tareas del desarrollo, tareas de prueba, riesgos, ajustes de seguridad, dirección de proceso, reportes de proceso, políticas de los registros de la fuente, entre otros. A su vez, *Microsoft* proyecta las plantillas y los servicios del proyecto sobre un portal/*Windows* y *SharePoint* localiza la plantilla configurada para apoyar los procesos de MSF.

MSF ágil proporciona la dirección integrada al plan, los requisitos de estructura, arquitectura, diseño, desarrollo, y prueba el proyecto. Esta dirección se inserta totalmente de modo que aparezca justamente diseñada como una extensión integrada al *Visual Studio.*

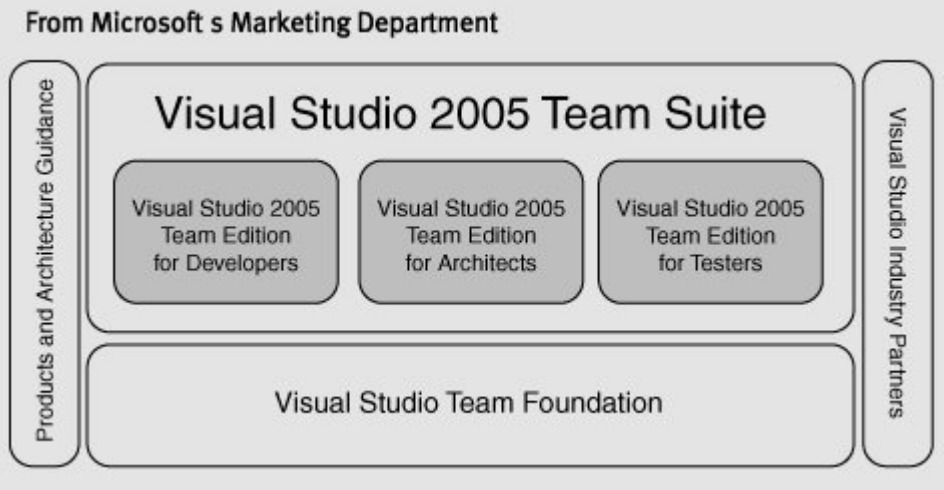

Fig. 16 ***Visual Studio2005 Team Suite*** (Sam Guckenheimer 2006)

En el caso del proyecto de Juegos Virtuales, la balanza se inclinó por la puesta en práctica de la versión *Suite* del *Visual Studio* por la oportunidad que brinda al equipo de desarrollo que compete en este caso, de establecer la comunicación y colaboración ideal. No obstante, es importante conocer el contenido de cada edición, que plantea (Sam Guckenheimer 2006) para tener un mejor dominio de estos productos.

Edición para desarrolladores de software: Proporciona las herramientas de desarrollo avanzado que permiten a los equipos construir servicios y usos confiables.

Edición para arquitectos de software: proporciona a los diseñadores visuales permitir a arquitectos, jefes de explotación y liberadores diseñar las soluciones servicio-orientadas que se pueden validar contra los ambientes operacionales.

Edición para *Testers* de software: proporciona las herramientas de prueba de carga avanzada que permiten a los equipos verificar el funcionamiento antes del despliegue.

Edición *Suite*: combina cada uno de los productos anteriores en una herramienta de gerencia asequible al ciclo de vida, que trata las necesidades de que existan roles de múltiples funciones en una organización.

Todos estos productos son soportados por *Visual Studio Team Foundation Server*, un servidor extensible de colaboración que permite a todos los miembros del equipo manejar y seguir el progreso y la "salud" de los proyectos.

Ventajas que ofrece VSTS

Con la buena puesta en marcha del VSTS en el proyecto de Juegos Virtuales de la UCI, los videos juegos pueden:

- Reducir la complejidad de entregar las soluciones servicio-orientadas modernas que se diseñan para las operaciones.
- Facilitar la colaboración entre todos los miembros de un equipo del software, dada la necesidad de cumplir con el tiempo de desarrollo establecido para cada proyecto y de asegurar la pre-visibilidad y la confiabilidad del proceso.
- Modificar y extender el *Team System* para requisitos particulares, en conjunto con las herramientas y procesos del mismo.
- Permitir a socios y clientes extender el VSTS con sus propias herramientas internas o comerciales y estructuras de proceso, mientras que la plataforma en la cual se construye el *Team System* y los servicios de base del TFS permite a terceros extender el sistema y compartir reportes, eventos, grupos y permisos, administración y capacidades para la navegación.
- Permitir que cada una de las herramientas del VSTS pueda ser extendida con APIs o servicios propios del equipo de desarrollo involucrado.

2.9 Visual SharePoint Server 2007

SharePoint Server 2007 es un producto de oficina creado para satisfacer las necesidades críticas de una entidad como la administración de procesos, la simplificación del modo de colaboración entre usuarios y la habilidad de tomar las mejores decisiones. Dentro de una plataforma integrada, *SharePoint Server 2007* no depende de sistemas fragmentados y sí admite todas las intranets, extranets y aplicaciones *Web* de la empresa. También incluye nuevas características de colaboración actualmente inasequibles a VSTS, incluyendo las *wikis*, los *blogs* y otros. Aunque VSTS no puede tomar la ventaja directa de estas nuevas

características, la integración con *SharePoint* está en el VSTS para una versión futura, que ampliará sus capacidades de colaboración.

2.10 Herramientas y técnicas existentes integradas a VSTS

Si bien es cierto que implantar VSTS es una buena opción para organizaciones o proyectos que corren sobre plataformas *Windows* y que requieran de eficaces procesos de mejoras durante el desarrollo de un producto, no es menos cierto que el uso incorrecto de las herramientas y técnicas aplicables a este entorno integrado puede provocar grandes afectaciones o hacer colapsar un proyecto que se ha comenzado a desarrollar. Sin dudas, las consecuencias de aplicar malas prácticas en la elaboración de productos de software, trae la insatisfacción del cliente, la pérdida de oportunidad del éxito en el mercado por parte del producto y un equipo de trabajo con baja autoestima. En resumen, se debe tener presente que las herramientas y técnicas que se apliquen a un proyecto de desarrollo son para agilizar el proceso de producción, no para convertirlo en un problema difícil de solucionar o sin solución.

2.10.1 Automatización de pruebas-*NUnit*

Aunque el VSTS trae de forma integrada un *framework* de prueba, muchos desarrolladores optan por usar *NUnit* en lugar del propuesto por *Microsoft*. *NUnit* surge bajo la copia fiel del *framework* de prueba *JUnit* que existía para aplicaciones Java.

Según (Francia 2007), las pruebas son diseñadas por el desarrollador y por los expertos del dominio. Estas pruebas se pueden automatizar usando como herramienta a *NUnit*. Una ventaja de esta perspectiva es la posibilidad de lanzar las pruebas varias veces. Con *NUnit* estas pruebas se llevarían a cabo rápidamente.

NUnit es un *framework* de pruebas para .NET que tiene dos características importantes:

1. *NUnit* usa "atributos" para identificar las pruebas en .NET. Las pruebas son identificadas por el atributo [*TestFixture*] en la clase y el atributo [*Test*] para indicar las pruebas individuales.

2. *NUnit* permite realizar pruebas en cualquier lenguaje de .NET. Se pueden escribir pruebas en Visual Basic .NET o C++. La interoperabilidad de lenguajes es una característica muy importante en .NET.

Las pruebas dan la confianza del código que se escribe, el equipo trabaja con menos presión y no duda a la hora de realizar cambios.

Algunas características de las pruebas unitarias planteadas por (Desarrolladores 2003) son:

- El programador prueba el código que desarrolla.
- Combate la regresión de errores y la introducción de nuevos errores al modificar el código.
- Las clases tienen sus propias pruebas.
- Aumentan la confianza en el código.

Ventajas de *NUnit*

- Un marco para desarrollar pruebas unitarias.
- Las pruebas están automatizadas.
- Permite que el código se compile y que las pruebas se ejecuten frecuentemente.

La automatización de pruebas tiene puntos fuertes pero también débiles que se tienen que conocer para evitar que las herramientas acaben quedando en el olvido.

2.10.2 Técnica del refactoring: calidad del código

Refactorizar (del inglés *refactoring*) es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo, lo que se conoce informalmente por “limpiar el código”.

La acción de refactorizar se realiza a menudo como parte del proceso de desarrollo del software: los desarrolladores alternan la inserción de nuevas funcionalidades y casos de prueba en el código para mejorar su consistencia interna y su claridad. Las pruebas aseguran que la acción de refactorizar no cambie el comportamiento del código.

Una de las opciones que ofrece esta técnica es la opción *rename*; esta opción permite renombrar el nombre de la función y reemplazar este nombre en todos los lugares donde se utiliza, por tanto, es un caso de *refactoring*.

Refactorizar es la acción que se ejecuta para el mantenimiento del código. No arregla errores ni añade funcionalidad sino que el objetivo es mejorar la facilidad de comprensión del código o cambiar su estructura y diseño y eliminar código muerto, para facilitar el mantenimiento del mismo en el futuro. Añadir nuevo comportamiento a un programa puede ser difícil con una estructura dada, así que un desarrollador puede refactorizar primero para facilitar esta tarea y luego añadir el nuevo comportamiento.

Esta técnica debe ser realizada como un paso separado, para poder comprobar con mayor facilidad que no se han introducido errores al llevarla a cabo. Al final de refactorizar, cualquier cambio en el comportamiento es claramente un *bug* y puede ser arreglado de manera separada a la depuración de la nueva funcionalidad. La acción de refactorizar es un aspecto importante de la programación extrema.

2.10.3 Visual Assist

Escrito por los liberadores para los liberadores, V*isual Assist* está llena de características imprescindibles que permiten leer, escribir y navegar. Por ejemplo:

- Sugerencias, siglas y plantillas del código
- Refactorizar en todos los lenguajes
- Resaltar colores en el código
- Destacar el *bug*
- Facilitar acceso a los métodos
- Buscador de la próxima clase

Para los proyectos programados en C++, Visual Studio no permite la acción de refactorizar el código y es por este motivo que se utiliza el *Visual Assist* en el proyecto de Juegos Virtuales, ya que esta herramienta permite refactorizar en cualquier lenguaje de programación.

2.10.4 Ingeniería Inversa

El objetivo de la ingeniería inversa es obtener información técnica a partir de un producto puesto a disposición del público, con el fin de determinar de qué está hecho, qué lo hace funcionar y cómo fue pensado. Los productos más comunes que son sometidos a la ingeniería inversa son los programas de computadoras y los componentes electrónicos.

Este método es denominado ingeniería inversa porque avanza en dirección opuesta a las tareas habituales de ingeniería, que consisten en utilizar datos técnicos para elaborar un producto determinado. En general si el producto (u otro material) que fue sometido a la ingeniería inversa se obtuvo en forma apropiada, entonces el proceso es legítimo y legal.

La ingeniería inversa es un método de resolución. Aplicarla a algo que supone profundizar en el estudio de su funcionamiento, hasta el punto de que se pueda llegar a entender, modificar y mejorar esta manera de proceder.

Pero este término no sólo se aplica al software como producto, pues se considera ingeniería inversa también al estudio de todo tipo de elementos, por ejemplo equipos electrónicos, microcontroladores, etc., siempre y cuando el resultado de dicho estudio repercuta en el entendimiento de su funcionamiento.

Otra forma de dar una definición es decir que la ingeniería inversa es la actividad que se ocupa de descubrir cómo funciona un programa (función o característica de cuyo código fuente no se dispone) hasta el punto de poder modificar ese código. Por poner un ejemplo se puede decir que la gran mayoría del software de pago incluye en su licencia una prohibición expresa de aplicar la ingeniería inversa a su código, con el intento de evitar que se pueda modificar y así los usuarios tengan que pagar por él.

Propuesta de herramientas y técnicas a usar. Justificación.

Si se ha determinado trabajar con el entorno integrado de desarrollo del *Team System* haciendo uso de las prestaciones del TFS, es válido aclarar que la correcta puesta en práctica de herramientas y técnicas integradas a este entorno (*Visual Assist*, *refactoring*, ingeniería inversa, etc) es determinante también en el éxito rotundo de cualquier proyecto. En el caso del grupo de desarrollo de Juegos Virtuales de la Facultad 5 de la UCI, ésta ha sido la estrategia a seguir para tener un mejor control sobre las versiones y dirección del proyecto; pero es posible que en el futuro sea estratégico para la UCI y/o el país migrar para el mundo del software libre.

Consideraciones del capítulo

1. MSF y las metodologías ágiles se encuentran estrechamente relacionadas tanto en los principios como en las prácticas para el desarrollo de software en ambientes que requieren un alto grado de adaptabilidad.
2. Algunas de las buenas prácticas del proceso MSF ágil aplicadas al proyecto de Juegos Virtuales de la Facultad 5 de la UCI, han corroborado la decisión de adoptarla como metodología, dada su integración con la herramienta VSTS 2005 y su extensibilidad con el TFS, y la necesidad de agilizar el proceso de producción de video juegos.
3. Por la necesidad actual que tiene el proyecto de Juegos Virtuales de obtener demos frecuentes de video juegos a corto plazo, se concluye que deben realizarse continuas pruebas a los mismos, lo cual facilita MSF ágil.

Capítulo III. Aplicación de la estrategia ágil.

Introducción

Aún son muchas las personas y organizaciones que desconocen los beneficios que traen consigo las prácticas ágiles de desarrollo de software en la actualidad. Sin embargo, esa realidad está cambiando a medida que aumentan los casos de éxito de proyectos que implementan dichos enfoques ágiles.

Un factor que influye en la toma de decisiones a la hora de seleccionar una metodología o marco de desarrollo de software, es la constante disputa que se establece entre quienes promueven el enfoque ágil versus aquellos que defienden la idea de que solo a través de metodologías formales como RUP es que puede lograrse el éxito de un proyecto de desarrollo de software. Existe también una tendencia que se ha encargado de fusionar lo mejor de ambos mundos, consolidando ciertas prácticas ágiles en procesos o marcos de trabajo considerados como formales.

A pesar de lo anterior, lo más importante que toda organización o persona debe tener en cuenta, es que no existe una metodología ideal para cualquier escenario en la que se aplique. La metodología de desarrollo que se seleccione, bien sea ágil o no, siempre dependerá directamente del equipo de trabajo, la cultura organizacional, lo cambiante del medio ambiente y la aceptación del usuario final.

3.1 Enfoque ágil

Algunos de los más ilustres autores de destacadas publicaciones sobre el tema ágil, según plantea (Microsoft 2007), tales como *Kent Beck, Ken Schwaber, Robert C. Martin, Martin Fowler*, entre muchos otros, describen el enfoque ágil de desarrollo de software como un marco teórico de mejores prácticas que busca minimizar los riesgos de un proyecto a través de iteraciones de desarrollo muy cortas, que generalmente no exceden de 4 semanas. Estos autores también destacan la figura de prototipos de aplicación que son desarrollados literalmente frente al usuario final para obtener su apreciación lo antes posible.

Otras características que se destacan dentro de un enfoque de desarrollo ágil son:

- Documentar sólo lo estrictamente necesario y centrarse en lo más crítico del producto, que es el código fuente.
- Las metodologías ágiles de desarrollo de software de hecho emplean una gran cantidad de mejores prácticas que van desde el diseño del espacio de trabajo del equipo del proyecto, hasta la forma en que se deben establecer reuniones de avance y la recopilación de requerimientos.

MSF ágil es la nueva propuesta de Microsoft en el mundo de procesos y prácticas ágiles de desarrollo de software. "*Microsoft Agile*" renueva al ya exitoso MSF V3.0, que es un marco de trabajo en cascada y espiral, tiene como principales características el ser altamente insistente, de planificación adaptable a los cambios y enfocado a las personas, implementando las mejores prácticas de desarrollo ágil de software.

Uno de los aspectos más interesantes acerca de las metodologías MSF ágil es el hecho de que ya viene integrada a la plataforma de desarrollo de VSTS**.**

Por medio de VSTS es posible contar con una serie de plantillas y guías adaptadas a cada metodología y orientadas a los roles definidos en cada una al alinear tanto la herramienta de desarrollo de VSTS como los procesos ágiles de MSF.

VSTS centra su aplicación en proyectos de software complejos construidos por un equipo grande y tiene todos los ingredientes necesarios para un equipo distribuido de arquitectos, desarrolladores y *testers*. Dicho esto, puede surgir la siguiente interrogante: ¿Las ventajas de VSTS se aplican igualmente a equipos más pequeños?

¿Es VSTS apropiado para su equipo?

Según (Ferrill 2006), una de las mejores maneras de determinar si VSTS es correcto para su equipo es que debe entender lo que hace. *Microsoft* tiene una abundante información del VSTS en su sitio Web, incluyendo los *white papers*, casos de estudio, y cómo documentar. El centro de desarrollo del VSTS incluye las secciones específicas para los distintos roles tratados por VSTS junto con las versiones de evaluación del software.

VSTS define roles por funciones específicas dentro de un equipo, incluyendo roles predefinidos como: profesional de la base de datos, arquitecto del software, el liberador, el *tester* y el manager de proyecto.

Por si fuera poco lo anteriormente expuesto, se adiciona que VSTS adapta la experiencia del usuario y las diversas capacidades que posee a cada rol en particular. Tener todos estos roles parece implicar que es probable que su equipo tenga unos o más individuos para ocupar cada rol. Esto es probablemente una especulación justa para los equipos grandes, pero para equipos más pequeños, los individuos completan a menudo múltiples roles. ¿Cómo saber si su equipo pequeño realmente beneficia de una herramienta que fue diseñada para organizaciones mucho más grandes? Una forma es mirar las ventajas definidas y posibles beneficios de VSTS para los equipos pequeños; también las características y ayudas que no necesitarán dichos proyectos. Luego se deberán conocer las necesidades de su equipo para ver cómo necesita de las ventajas mencionadas.

Definiendo beneficios

VSTS beneficiará definitivamente, como se enuncia en (Ferrill 2006), cualquier equipo con la auditoria estricta, seguimiento de *bugs* y los reportes, etc. Usar las herramientas de reportes automatizados hace posible proveer a sus clientes un historial completo de los cambios del código y de las pruebas terminadas.

VSTS también proporciona ventajas enormes para cualquier equipo que tenga que apoyar liberaciones *(releases)* múltiples de un producto de software. El manejo de liberaciones múltiples de un proyecto del software es una tarea que mejor se manipula de forma automática. VSTS tiene un conjunto de herramientas sofisticadas de gestión de construcción/liberación *(build/release management toolset)* que simplifica grandemente el proceso de construcción de versiones específicas del programa.

También permite iniciar una construcción a través de todo el equipo. Toda actividad registrada se envía a un almacén de datos mantenido en una base de datos del servidor de SQL.

Las compañías que mantienen una base grande del código fuente son también candidatas fuertes a la adopción de VSTS. El control del código fuente en VSTS está unido a los *work items*. El nuevo código

puede asociarse en los registros a un *work item*, y cualquier modificación del código existente se adiciona automáticamente.

Reportando el uso de las herramientas, estos acoplamientos proporcionan reportes detallados, tales como mostrar todos los módulos cambiados atados a un *work item* específico o a un *bug*.

Ventajas Posibles

La automatización es una de las ventajas primarias de VSTS para equipos de cualquier tamaño, dice (Ferrill 2006). Para los equipos pequeños esto es especialmente útil en el área de prueba. VSTS genera automáticamente el código para que las pruebas unitarias hagan el proceso total de la aceptación y de la prueba más fácil. Producir calidad en el código está directamente relacionado con la cantidad y la minuciosidad de la prueba realizada contra ese código. VSTS hace que las tareas sean más fáciles e integradas con el proceso entero de desarrollo.

Otro factor a considerar son los roles dentro de un equipo. Los miembros de un proyecto pequeño cambian con frecuencia funciones de trabajo en una manera ágil; un miembro de un equipo podría ser codificador un día y diseñador el siguiente. VSTS proporciona menús constantes y las interfaces de usuario ayudan a hacer mucho más fáciles tales transiciones entre los roles. El portal del proyecto de VSTS permite a los miembros y/o administradores del equipo visualizar, directamente desde un buscador estándar de la Web, cualquier reporte. Además esta información puede ser revisada directamente desde *Microsoft Excel* y *Project* con el propósito de análisis y planificación.

La colaboración es algo que los equipos más pequeños ignoran a menudo, pero que llega a ser crítica para los equipos distribuidos, especialmente cuando a los miembros del equipo los separan grandes distancias. VSTS proporciona un número de capacidades únicas, incluyendo la capacidad de tener acceso a cosas tales como acontecimientos programados.

Los miembros del equipo pueden tener una visión fácil de acontecimientos tales como registros del código, estructura, y resultados de la prueba en sus lectores de noticias a través de mecanismos que brinda el VSTS. Incluso para los miembros del equipo en proximidad cercana, la capacidad de recibir tales

notificaciones y de tener un expediente de tales acontecimientos—puede proporcionar ventajas significativas.

¿Cuando es innecesario VSTS?

Según (Ferrill 2006), migrar viejos proyectos para VSTS no es una tarea trivial, por lo que proyectos que estén en su etapa de terminación, que no tendrán un cambio significativo en su desarrollo no ameritan tales esfuerzos. Por otra parte, aquellos pequeños proyectos de software construidos para una pequeña audiencia probablemente tampoco sean dignos de ser guiados por la herramienta. No obstante hay que tener en cuenta que algunos proyectos pueden en ocasiones crecer velozmente y tomar dimensiones insospechadas.

¿Cuándo puede ser necesario VSTS?

Según (Ferrill 2006) si al hacer las interrogantes siguientes:

- ¿A los miembros del equipo que desempeñan roles específicos les serian cómodas las definiciones de roles que VSTS define?
- ¿El equipo trabaja en proyectos grandes sobre una buena base del código fuente?
- ¿El equipo debe dar soporte para múltiples versiones del proyecto?
- ¿Está el equipo cómodo con la construcción/liberación del proyecto de manera automática?
- ¿El equipo necesita darle un seguimiento robusto a los *bugs*, *work item*s, reportes y al código?
- ¿Se puede decir que su equipo es distribuido? ¿Alternativamente, podría el equipo beneficiarse de las características de colaboración de VSTS?

La respuesta a una o más de estas preguntas es “sí”, el equipo es un candidato potencial a VSTS. Si la respuesta a todas las preguntas es “no”, entonces ese equipo puede no necesitar VSTS ahora; sin embargo, si están trabajando en un proyecto que es probable que continúe creciendo, se deben retomar las preguntas en el futuro, dado que hasta cierto punto, los proyectos pequeños se convierten en proyectos grandes, y usar técnicas de los pequeños equipos llega a ser poco manejable. Decidir si mover su equipo a VSTS no es un problema trivial. La diferencia de costes entre una versión

estándar del *Visual Studio* y una verdadera versión es significativa. Se deben tomar en cuenta lo planteado en este capitulo y no olvidar el factor tiempo de desarrollo potencial y el esfuerzo reducido que VSTS puede entregar antes de tomar una decisión final. Esta

3.2 Aplicación en el desarrollo de video juegos

El proyecto de Juego Virtuales de la Facultad 5 fue fundado desde el año 2005 con el objetivo de desarrollar video juegos para parques de diversiones como parte de un plan del país para la reanimación de los mismos.

La primera etapa del proyecto consistió en realizar diferentes módulos para la confección de video juegos. Estos módulos fueron programados en C++ bajo el entorno de desarrollo *Microsoft Visual Studio 2003*, auxiliado por el *Microsoft Visual Source Safe 6.0* para realizar el control de versiones.

Para la planificación del proyecto se utilizaron diferentes herramientas entre las que pueden enunciarse *Microsoft Exchang*e para asignación y seguimiento de las tareas, y para la planificación de encuentros y *Microsoft Visio* para la confección de diagramas relacionados con la ingeniería de software.

A medida que avanzaba el tiempo las dimensiones del proyecto fueron creciendo considerablemente lo que dificultaba la administración y control de las tareas asignadas al equipo de trabajo. Entonces, la dirección del proyecto decidió migrar para un entorno integrado de desarrollo, que permitiera tener un control sobre las versiones del producto, que facilitara la colaboración entre todos los miembros del equipo, que se pudieran realizar cambios rápidos si los requisitos del cliente variaban, en fin, que permitiera una mejor planeación, programación, comunicación y control sobre el proyecto.

En este caso se decidió por *Visual Studio 2005 Team System Suite* para poder hacer uso de las prestaciones del *Microsoft Visual Studio 2005 Team Foundation Server.*

Las condiciones recomendadas para instalar un servidor de *Team Foundation* son las siguientes:

- Memoria RAM de 1GB.
- Espacio disponible en disco duro de 8 GB.
- Velocidad del micro-procesador de 2.2 GHz Pentium IV.

El grupo de desarrollo de Juegos Virtuales ha visto limitado el desarrollo del proyecto por no contar del todo con los requerimientos de hardware antes mencionados, aunque es válido destacar que estos obstáculos no han impedido que se continúe avanzando satisfactoriamente en el mismo.

3.3 Experiencias con el enfoque integrado de desarrollo.

Partiendo del código ya implementado, el grupo de desarrollo de Juegos Virtuales empleó la ingeniería inversa para realizar la documentación del proyecto referente a la ingeniería del software.

El trabajo con la herramienta *Visual Assist* permitió la puesta en práctica de la técnica de *refactoring* para mejorar el código programado, ya que el *Visual Studio* no permite utilizar esta técnica para proyectos en C++.

Existen otras herramientas utilitarias puestas en marcha ya en el proyecto de Juegos Virtuales, que permiten a los usuarios (equipo de trabajo, administradores, etc.) acceder y trabajar con el entorno del VSTS. Algunas de ellas son:

- Team Look
- Team Explorer
- Team Plain

Team Look

El diseño *Team Look* extiende a *Microsoft Outlook* de modo que todos los *stakeholders* en un proyecto del software puedan comunicarse con más eficacia y ganar una visibilidad exacta del estado del proyecto. *Team Look* se conecta con el *Team Foundation Server* y permite a los usuarios ver *work items*, reportes, y otros artefactos almacenados. Si se lleva una sincronización constante, *Team Look* funciona de una manera similar a otras características del *Outlook*, reduciendo los costos y permitiendo a organizaciones realizar aumentos inmediatos de la productividad.

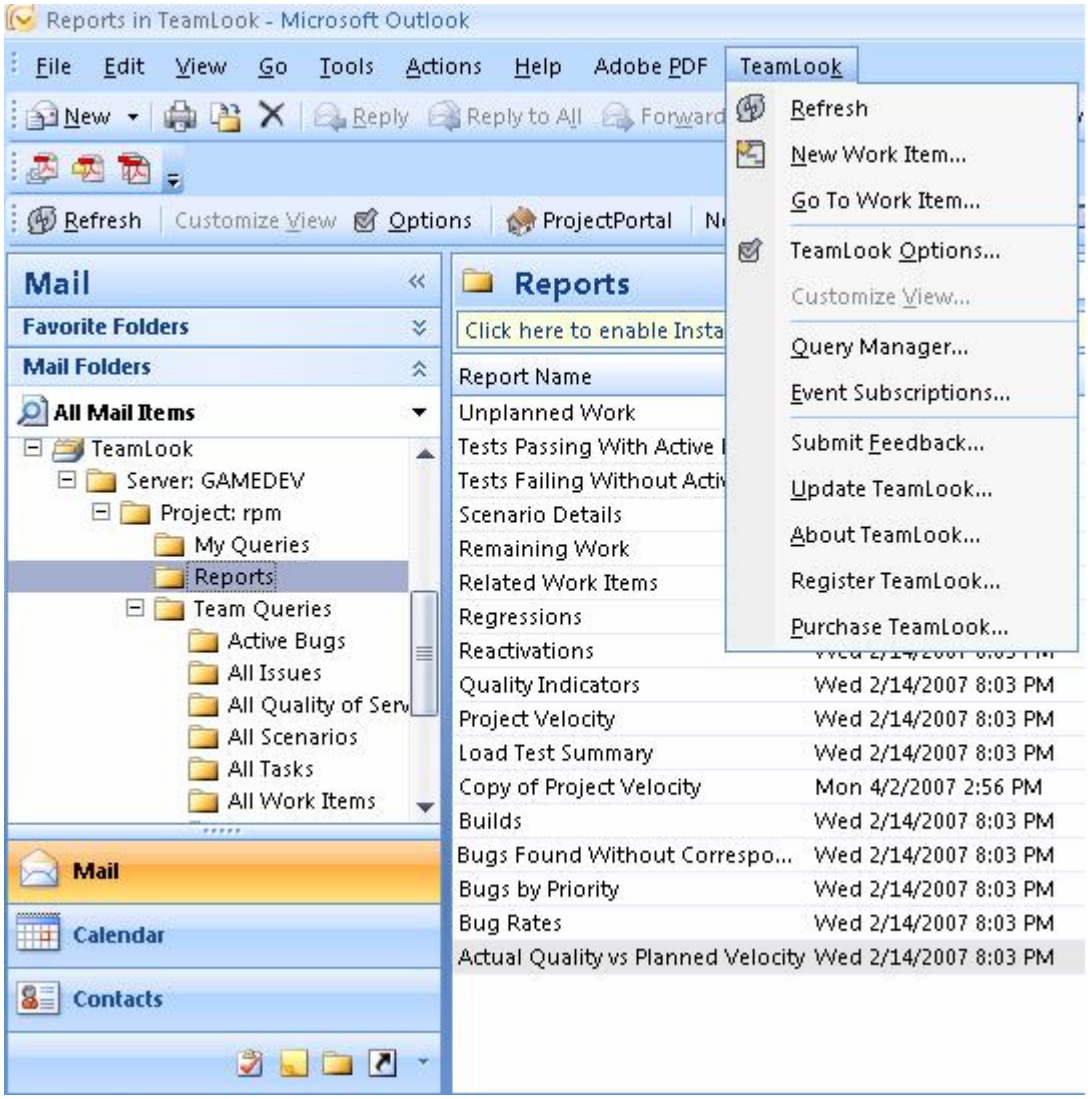

Fig.17 *Team Look* extendido a *Microsoft Outlook*

La parte izquierda de la figura muestra el explorador del *Team Look* similar al *Team Explorer* del *Visual Studio*, de esta forma se facilita la navegación rápida por los diferentes elementos que componen el proyecto.

Trabajar con TFS mediante el cliente de correo a través del *Team Look* resulta más cómodo para revisar el estado del proyecto sin necesidad de tener instalado el *Visual Studio*. Los usuarios con privilegios

insuficientes pueden crear *work items*, ver los reportes, ver las tareas, etc pero no tienen acceso a la columna vertebral del proyecto.

Team Explorer

El *Team Explorer* es la herramienta que nos permite interactuar desde *Visual Studio* 2005 con el TFS. Desde capturar w*ork items*, acceder a las bibliotecas de documentos del equipo de trabajo, generar reportes, hasta labores de administración como definir los permisos de acceso de los usuarios o administrar las plantillas de proceso existentes en el TFS.

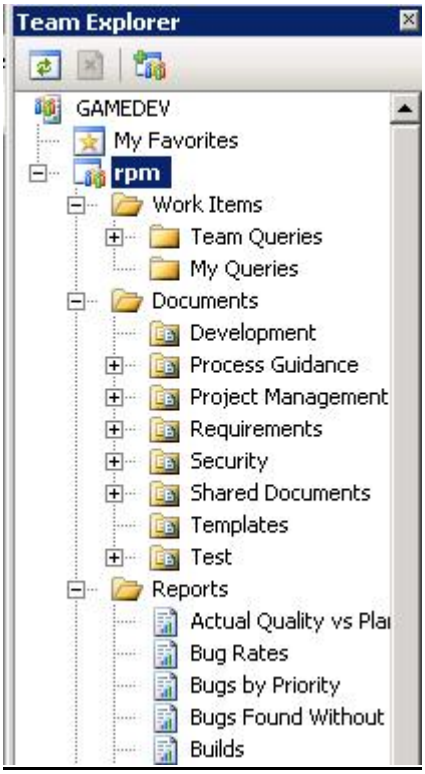

Fig.18 Vista del *Visual Studio Team System Team Explorer*

Luego de haber realizado la conexión con TFS, se carga el proyecto deseado. La ventana del *Team Explorer* muestra todo el contenido del proyecto en modo de explorador para una mejor navegabilidad.

Team Plain: Interfaz Web para TFS

Microsoft adquirió *Team Plain* para desarrollar una interfaz de acceso a TFS a través del navegador Web. Dice (Corral 2007) que *Team Plain* es una herramienta para acceder a TFS que también facilita la vida a aquellos miembros del equipo de desarrollo que colaboren ocasionalmente con el proyecto. Esta interfaz permite manipular *work items* y código fuente.

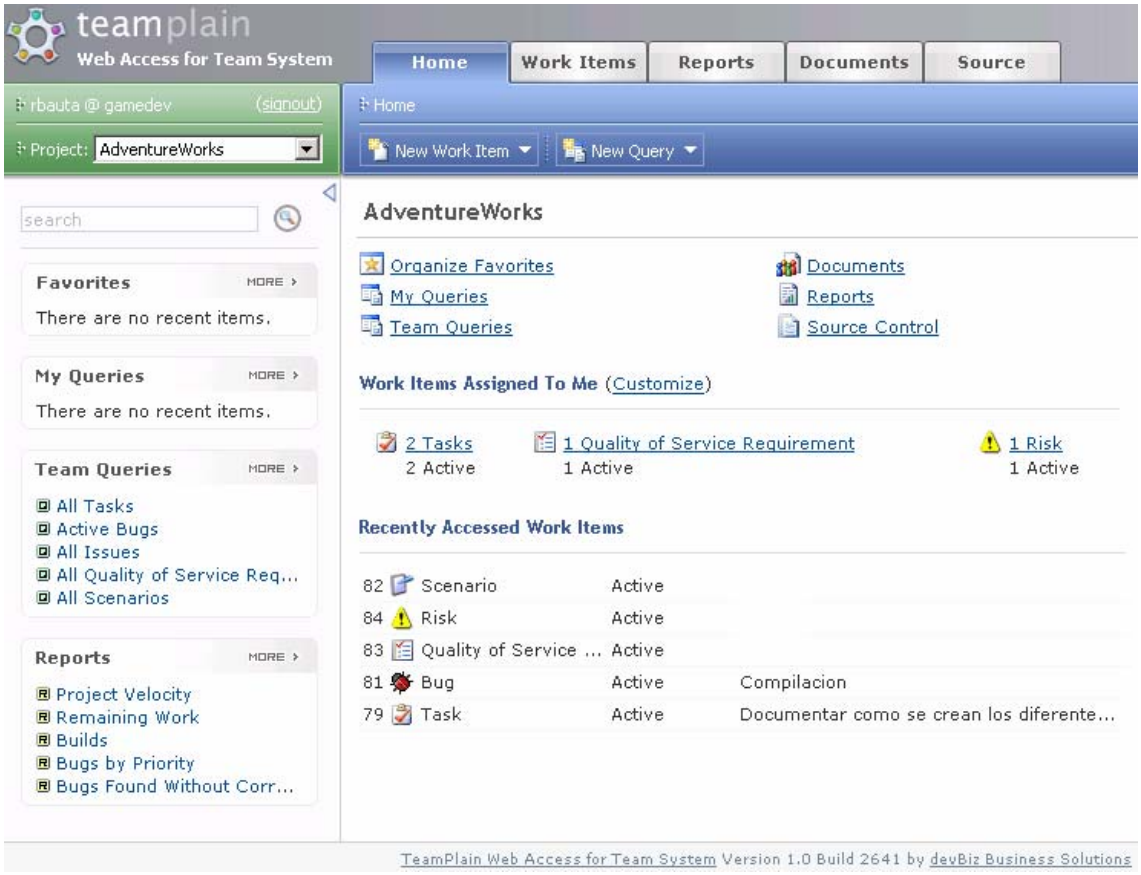

Fig. 19 *Team Plain* para TFS

3.4 Vinculación de TFS con el proyecto de Juegos Virtuales.

Después de la adopción de TFS, el grupo de desarrollo de Juegos Virtuales ha tenido que irse adaptando a esta nueva forma de trabajar, dada la gran cantidad de funcionalidades que brinda su enfoque integrado.

Sin embargo aprender a trabajar con las prestaciones de TFS y paralelamente ir desarrollando el producto ha sido un reto para todos.

Se pretende en este epígrafe exponer como se ha ido trabajando en el proyecto y como se han podido constatar errores, atrasos, incumplimientos de tareas con la modificación de *work items* en las fases de desarrollo, sin olvidar que los roles desempeñados por el equipo de trabajo en dichas fases influyen determinantemente en el resultado final. En resumen, este epígrafe contiene algunas de las principales experiencias que se han tenido en el proyecto de Juegos Virtuales con la puesta en práctica de las propuestas realizadas.

Para comenzar a trabajar hacia la adopción de este enfoque integrado, el primer paso que tuvo que dar el equipo de trabajo fue migrar el proyecto ya creado de *Visual SourceSafe 6.0* a *Team Foundation Server,* aunque fue necesario instalar el *Visual SourceSafe 2005* porque era preciso emplear una herramienta de conversión de esta versión más avanzada, el *VSSConverter.exe*. Esta herramienta usa *SQL Express*.

Al dar este paso los miembros del equipo ya tenían definido ciertos requisitos como por ejemplo la plantilla del proceso “MFS para Ágil Desarrollo de Software - v4.0”, no obstante este proceso de migrar el código no fue una tarea sencilla, a pesar que a primera vista lo puede parecer dado que no hay que empezar de cero sino sobre la base de un código ya implementado; pero lo cierto es que el proceso se realizó a través de líneas de comandos con el convertidor antes mencionado dado que aún no se cuentan con herramientas de administración con interfaces graficas para realizar estos procedimientos.

Otro de los pasos que se dio fue la creación del *Team Project* al proyecto ya migrado, para que los miembros del equipo de trabajo pudieran manipular las disímiles herramientas del TFS.

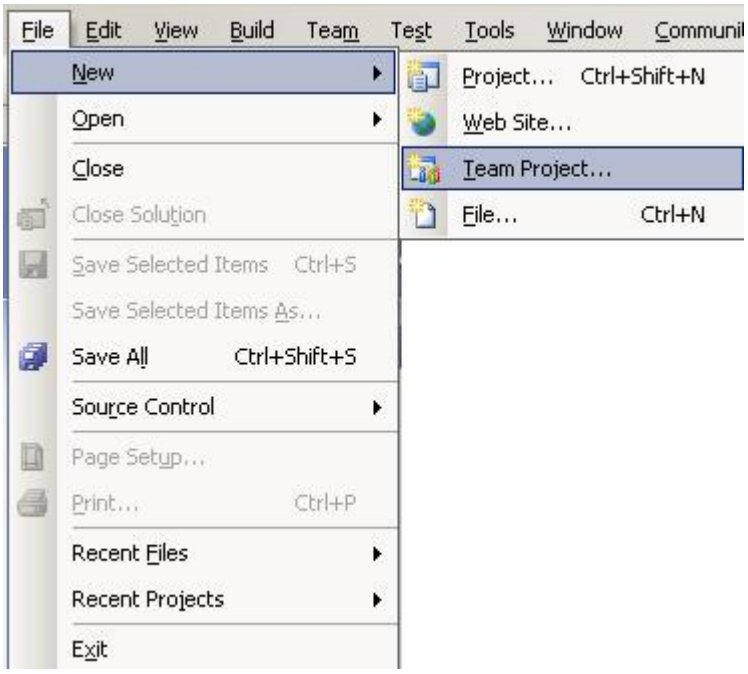

Fig.20 Creando un Team Project

El asistente para *Team Project* se utiliza cuando se necesitan crear varios componentes de este tipo en V*isual Studio.* Después que el *Team Project* es creado los miembros del equipo usan su nombre para ubicarlo en el *Team System*. Es importante asegurarse que el nombre del *Team Project* sea único y que ese nombre no esté realmente en uso en TFS, ni en los servicios de *SharePoint* ni en los servicios de reportes del *SQL Server.*

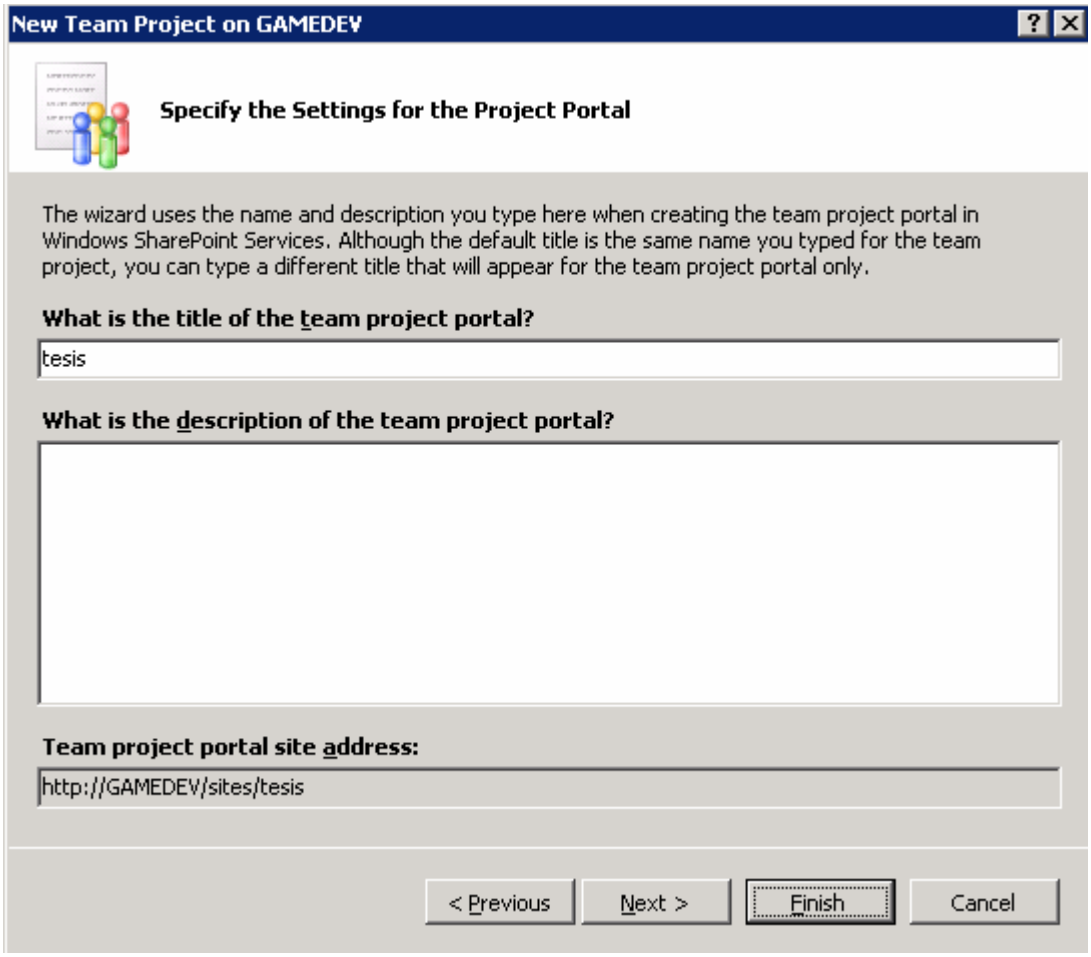

Fig. 21 Asistente para Team Project

Todos los proyectos del equipo de trabajo se almacenan y manejan en los servidores de TFS, el de Juegos Virtuales no es la excepción. Para ello fue necesario establecer una conexión con el servidor apropiado de TFS, cuyo nombre hasta el momento de esta investigación es GAMEDEV, y se entró en contacto con el administrador del servidor/proyecto, que resulta ser la misma persona en este caso, para asegurarse de que se tuvieran privilegios suficientes para llevar a cabo la conexión.

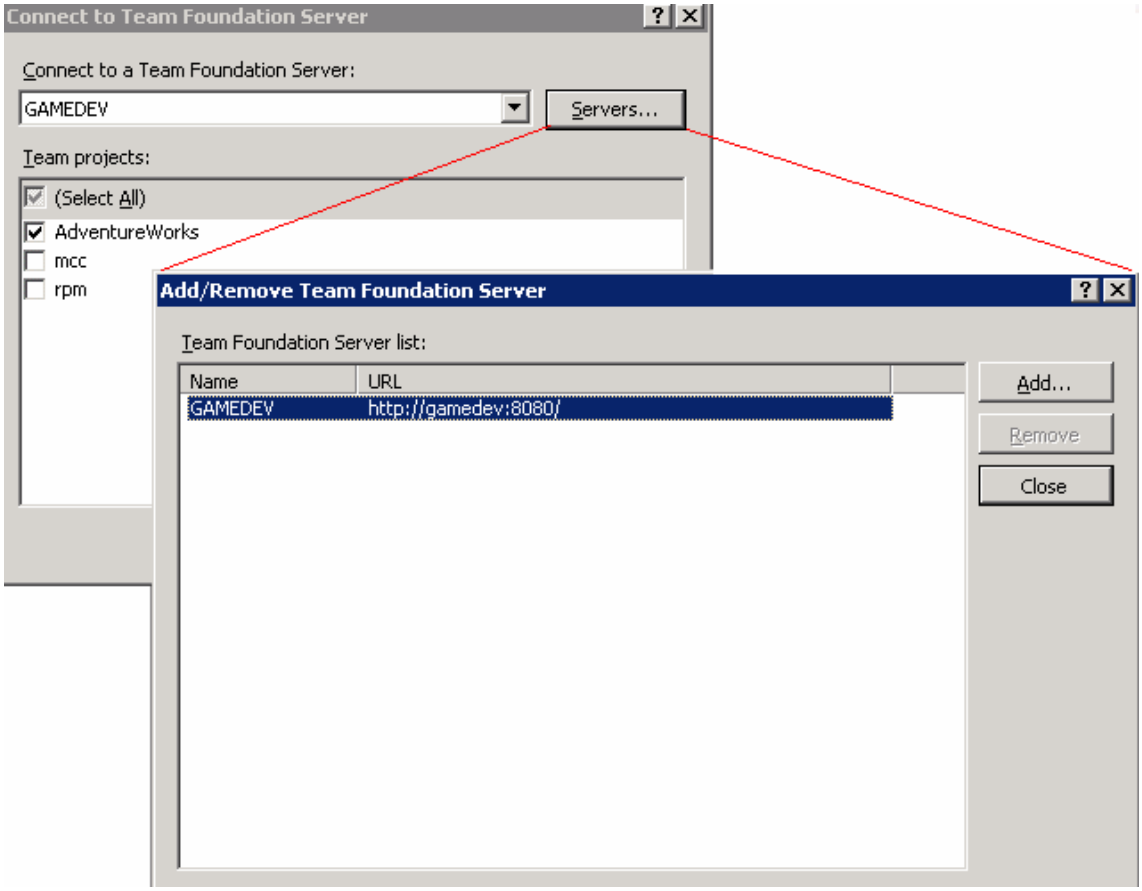

Fig.22 Conexión a TFS desde *Visual Studio*

El administrador del proyecto de Juegos Virtuales fue agregando diversos *work items* para darle seguimiento a los trabajos asignados a cada miembro. MSF para el proceso ágil del desarrollo del software (plantilla que se utiliza en este proyecto) incluye por ejemplo, un *work item* tipo *bug* que se utiliza para seguir de cerca los defectos del producto.

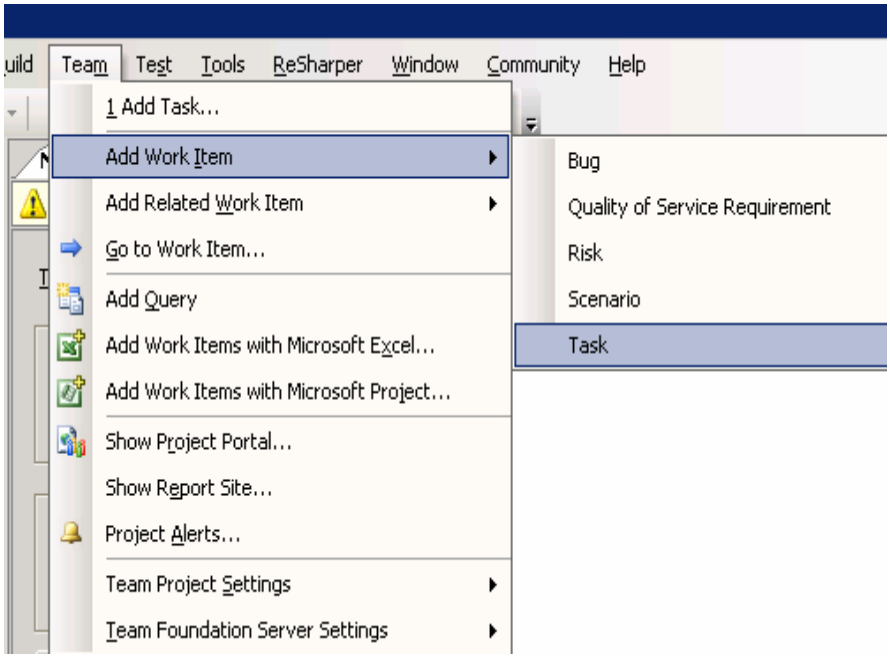

Fig.23 Creación de un work item.

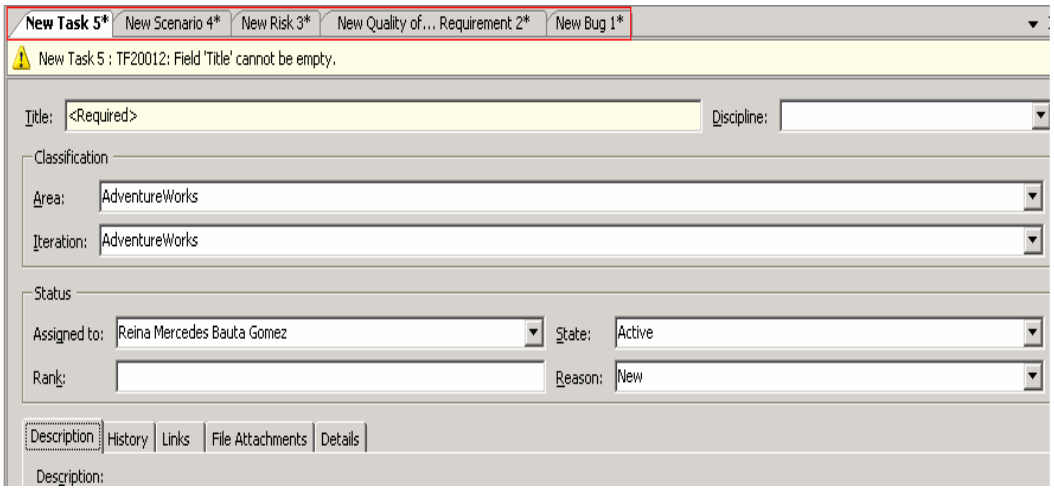

Fig. 24 Ventana estándar para los *work items*.

El proyecto de Juegos Virtuales de la Facultad 5 tiende a seguir aumentando sus dimensiones y para el administrador ya se está haciendo engorrosa la manipulación de los *work items* que se tienen almacenados. Una forma de acceder con mayor rapidez a uno de ellos es conociendo su ID.

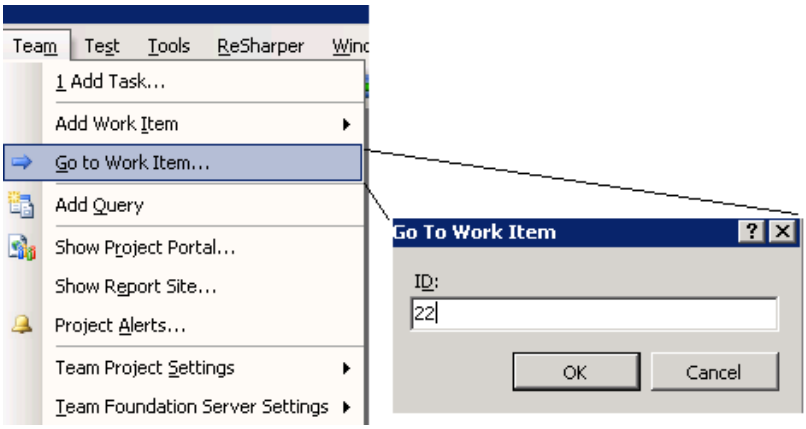

Fig.25 Acceso rápido a un *work item.*

También el proyecto cuenta con varios reportes, que se pueden manipular y visualizar a través de las herramientas vinculadas al TFS si se tienen los privilegios suficientes. Los reportes ofrecen al usuario una visión general del estado actual del proyecto para ejercer un mejor control sobre el mismo, por lo que contribuyen al proceso de mejoras continuas.

Estos reportes constan de una serie de filtros (en forma de *listbox* generalmente) que permiten seleccionar los parámetros deseados para su emisión como se ejemplifica en la siguiente figura:

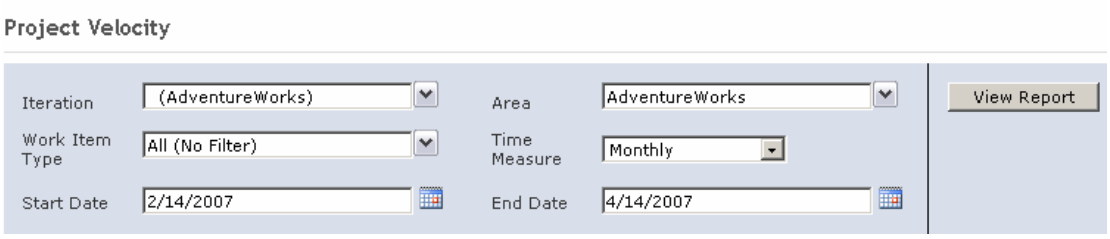

Fig.26 Filtros del reporte para conocer la velocidad de avance del proyecto.

Un ejemplo de reporte emitido por el proyecto de Juegos Virtuales es la gráfica de velocidad del proyecto que se muestra a continuación:

Fig.27 Gráfica del reporte *Project Velocity*

En la gráfica se expresan la cantidad de transiciones del proyecto en función del tiempo (desde febrero hasta abril del 2007), en correspondencia con las actividades que en ese intervalo se encontraban cerradas, resueltas y una línea promedio de estas últimas.

Otro ejemplo de reporte emitido por el proyecto es también el que se emite para conocer el trabajo que queda por hacer o *remaining work,* mostrado en la siguiente gráfica:

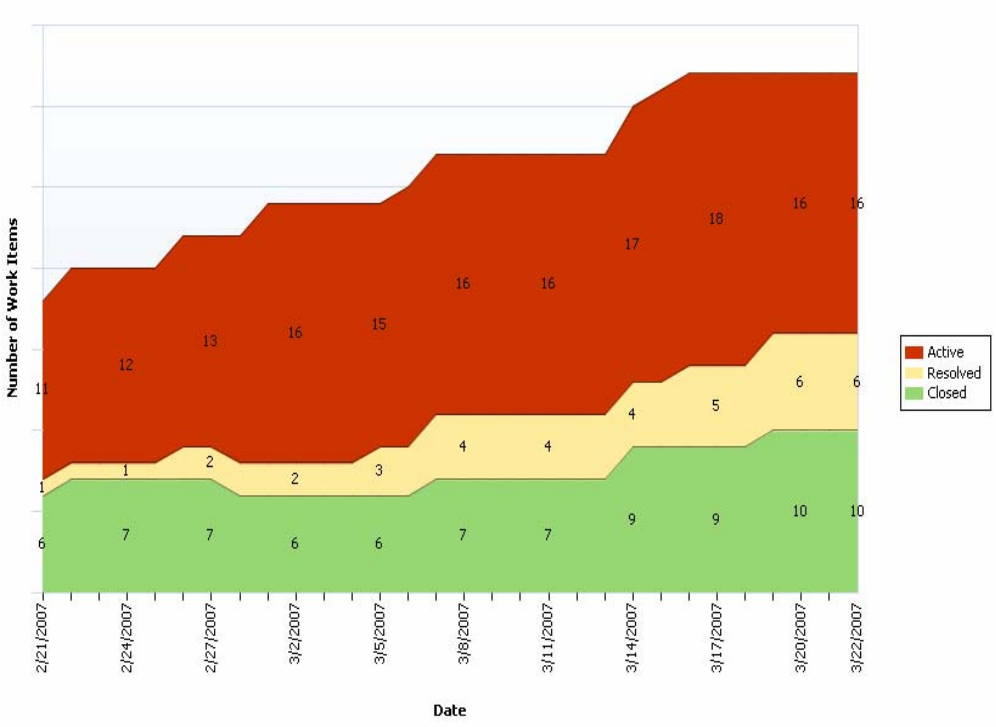

Fig. 28 Vista del estado de las tareas en el tiempo

Este reporte muestra el estado diario del avance que tuvieron las actividades del proyecto en los meses de febrero a marzo del 2007. Se pueden observar la cantidad de *work items* que fueron resueltos, los que están cerrados y los que faltan por solucionarse.

El TFS requiere de Microsoft SQL Server 2005, lo que permite realizar consultas de los *works items* en forma eficiente.

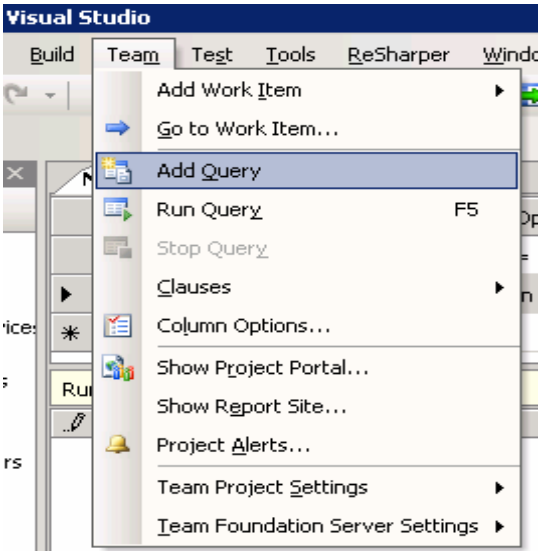

Fig. 29 Creación de una consulta.

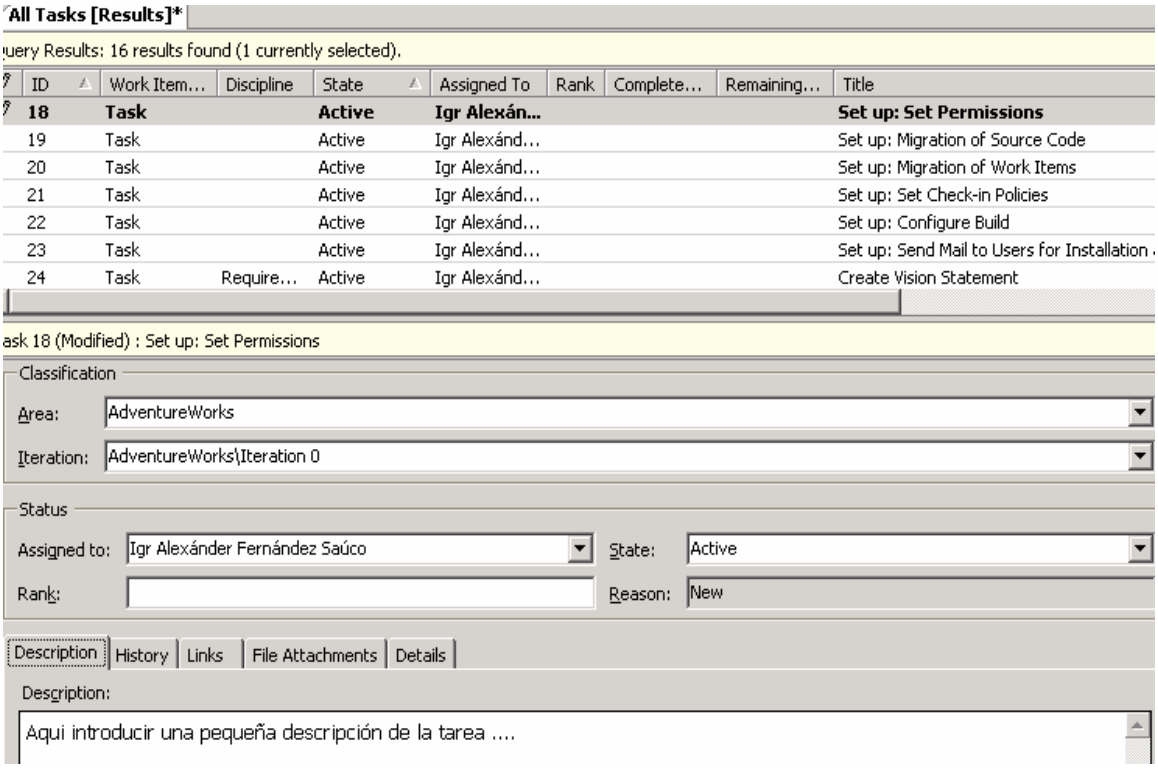

Fig. 30 Ejemplo de una consulta realizada al proyecto.

A través del *Team Explorer* el administrador del proyecto y demás miembros del equipo han podido visualizar y editar de forma directa la documentación del proyecto (inaccesible para algunos miembros del equipo). Como se había expuesto en el capitulo anterior, TFS puede integrarse además con *Microsoft Office Project 2003* y *Office Excel 2003* para ampliar sus capacidades de elaboración de reportes y seguimiento. En la siguiente imagen de ejemplo se muestran todos los *work items* tipo tareas que se estaban llevando a cabo por un miembro del equipo el 15 de mayo de 2007, entre otros datos.

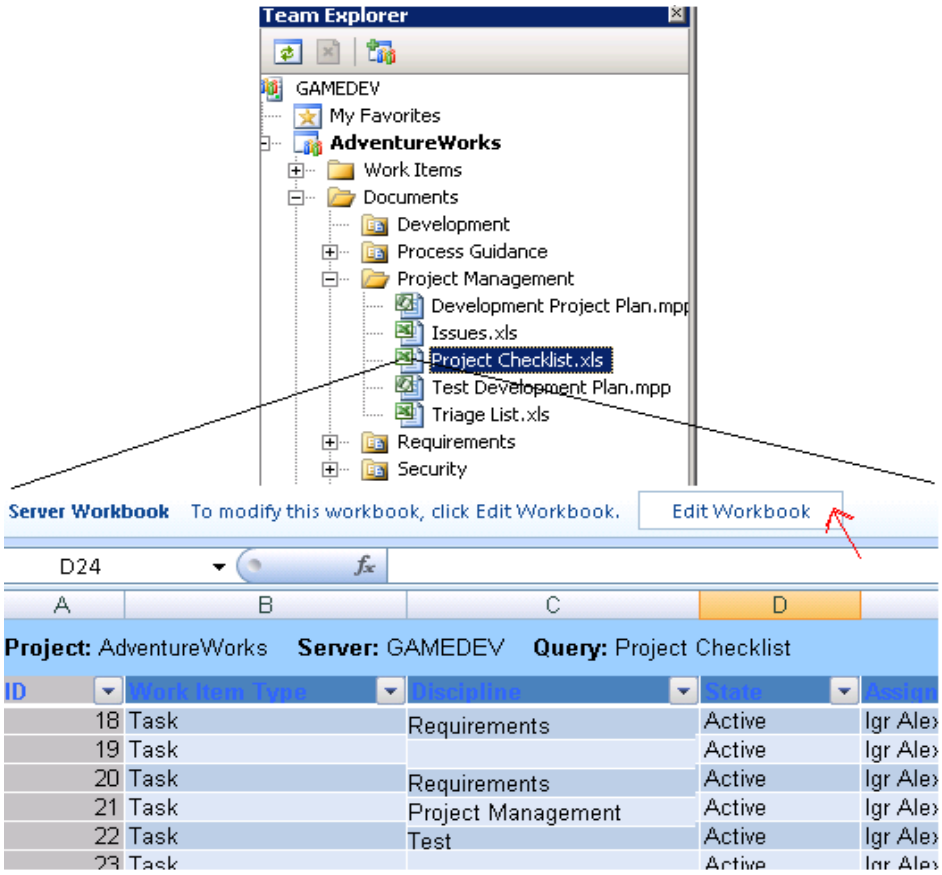

Fig. 31 Documentación del proyecto.

Consideraciones del capítulo

1. *Visual Studio 2005 Team System* unido a *Team Foundation Server* es de gran ayuda en la creación de un software exitoso a cada uno de los integrantes de un equipo de desarrollo independientemente del rol que se desempeñe.
2. Si el equipo de trabajo tiene la habilidad de seguir la metodología y proceso soportado por el conjunto de herramientas expuestas, entonces es casi evidente el éxito del proyecto.
3. El enfoque integrado permite al equipo de trabajo cumplir con las tareas asignadas en un ambiente de alta colaboración, facilitando la información que se demanda para mantener el rumbo y garantizar que el proyecto cumpla con los entregables en cada paso de su desarrollo.
4. El entorno integrado de desarrollo permite a la dirección del proyecto tener un mayor control sobre el desarrollo del producto y el desempeño de los miembros del equipo, pudiendo medir la eficiencia, asignar de manera eficaz las tareas a cada uno de ellos y ver su cumplimiento a través de los reportes emitidos por la herramienta.

Conclusiones

1. Sin la puesta en práctica de una metodología adecuada, teniendo en cuenta que cada proyecto tiene sus características propias, no es posible obtener un software con calidad, pues esta es el hilo conductor del desarrollo de un producto en cualquiera de sus fases.
2. Las metodologías RUP, MSF, AGILE, XP y MSF Ágil tienen sus ventajas y desventajas pero se decidió aplicar MSF Ágil por la necesidad de utilizar una metodología desarrolladora que arrojara resultados en tiempo récord y que fuera cómoda para los miembros del *proyecto de juegos de consola.*
3. *Visual Studio 2005 Team Foundation Server* desde su implantación en el proyecto ha logrado mejorar la gestión del control del código fuente, ha centralizado el almacenamiento de datos, le ha dado seguimiento a los *work items*, y como consecuencia ha proporcionado una mayor colaboración entre los miembros del equipo.

Recomendaciones

1. Profundizar en el estudio para aplicación de otras técnicas y herramientas como por ejemplo la automatización de pruebas unitarias, pruebas de rendimientos, y otras funcionalidades no contempladas en este documento.
2. Extender el uso de entornos integrados de desarrollo en todos los proyectos de la UCI.
3. Desarrollar en la UCI un conjunto de proyectos para la realización de herramientas de gestión integradas al proceso de desarrollo que permitan en el futuro el empleo de estrategias ágiles fuera del entorno del software propietario.

Referencias Bibliográficas

Ávila, S. J. V. (2005). "Introducción a Microsoft Solutions Framework." Retrieved 12, 2006, from http://www.mentores.net/articulos/intro_microsoft_sol_frame.htm.

binario, L. (2007). "Definición de Video Juego." Retrieved 5, 2007, from http://www.lenguajebinario.com.ar/foro/image-vp4433.html?sid=ddf63f05d0310ae972d9927200ecdc09.

CIUP. (2006). "GLOSARIO." from http://controlinterno.udea.edu.co/ciup/glosario.htm.

colombia, M. c. (2000). "Microsoft Solutions Framework (MSF): Disciplinas y buenas prácticas para el desarrollo e implantación de proyectos

Un servicio de transferencia de conocimientos de Microsoft Consulting Services " Retrieved 12, 2006, from http://www.microsoft.com/colombia/portafolio/msf.htm.

Consultores, G. (2006). "Disciplina de administración del proyecto - M.S.F." Retrieved 11, 2006, from http://www.gpicr.com/msf.aspx.

Corral, R. (2007). " Varios temas interesantes sobre Team Foundation Server y Visual Studio Team System." 2007, from http://geeks.ms/blogs/rcorral/archive/2007/03/27/varios-temas-interesantes-sobre-team-foundation-server-y-visual-studio-team-system.aspx.

Desarrolladores, F. d. (2003). "CICLO DE VIDA DEL DESARROLLO DE SOFTWARE CON .NET." Retrieved 1, 2007, from http://www.slideshare.net/vsanchez/ciclo-de-vida-del-desarrollo-de-software-con-net.

Fernandez, L. (2005). "Herramientas y métricas para guiar inspecciones y pruebas de software." from http://web.iti.upv.es/~squac/JTS/JTS2005/contenido.html.

Ferrill, P. (2006). "A Small-team Analysis of Visual Studio Team System." from http://www.devx.com/dotnet/Article/33443.

Francia, J. (2007). "Desarrollo usando pruebas en .NET." 2007, from http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art150.asp.

GUIDANCE, P. (2005). "MSF for agile software development ", 2007, from http://www.microsoft.com/downloads/details.aspx?familyid=9F3EA426-C2B2-4264-BA0F-35A021D85234&displaylang=en.

Ivar Jacobson, G. B., James Rumbaugh (1999). Proceso Unificado de Desarrollo de Software.

Jose H. Canós, P. L. y. M. C. P. (2005). "Métodologías Ágiles en el Desarrollo de Software." Retrieved 12, 2006, from http://www.willydev.net/descargas/prev/TodoAgil.Pdf.

Maddison. (1983). "INGENIERIA DE SOFTWARE EDUCATIVO." Retrieved 12, 2006, from http://www.fi.uba.ar/laboratorios/lsi/c-icie99-ingenieriasoftwareeducativo.pdf.

Medela, S. P. S. S. L. (2006). "Y con el UML qué hacemos?" Retrieved 12, 2006, from http://www.mug.org.ar/Descargas/Jornadas/2481.aspx.

Microsoft. (2002). "De la Disciplina para la administración de proyectos MSF v. 1.1." Retrieved 12, 2006, from http://www.willydev.net/descargas/DisMSF.pdf.

Microsoft. (2007). "Metodologías Ágiles en el Desarrollo de Software y la propuesta de Microsoft -Ser o no ser ágil, he ahí el dilema." 2007, from http://www.microsoft.com/colombia/empresas/clientepreferencial/actualizacion-gerencial/febrero/innovaciones2.mspx.

Miguélez, M. M. (1999). "Criterios para la Superación del Debate Metodológico "Cuantitativo/Cualitativo"." from http://prof.usb.ve/miguelm/superaciondebate.html.

Molpeceres, A. (2003). "Procesos de desarrollo:RUP, XP y FDD." Retrieved 11/11, 2006, from http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.

P.Letelier. "Proceso de desarrollo de software" Retrieved 11, 2006, from https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20Proceso%20de%20Desarrollo%20de%20SW.doc.

Paraguay, a.-V. (2007). "TECNOLOGIAS : COMENZÓ A TOMAR IMPULSO LA INDUSTRIA DEL SOFTWARE EN CORRIENTES." Retrieved 4, 2007, from http://www.vivaparaguay.com/modules/news/article.php?storyid=62355.

Sam Guckenheimer, J. J. P. (2006). Software Engineering with Microsoft Visual Studio Team System, Addison Wesley Professional.

Sanchez, M. A. M. (2004). "Metodologías De Desarrollo De Software." Retrieved 12, 2006, from http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf.

Spaces, W. L. (2005). "Qué hay para decir de MSF (Microsoft Solution Framework)." Retrieved 11, 2006, from http://diegumzone.spaces.live.com/Blog/cns!1pHxrrKG6RzuZjEIZgyJyg0A!119.entry.

Wikimedia. (2007). "Proceso Unificado de Rational." Retrieved 1, 2007, from http://es.wikipedia.org/wiki/Rational_Unified_Process.

Wikipedia. (2007). "Proceso." from es.wikipedia.org/wiki/Proceso.

Zeledón, J. "Conceptos y Evolución de la Ingeniería del Software." from http://html.rincondelvago.com/conceptos-y-evolucion-de-la-ingenieria-del-software.html.

Bibliografía

ACTUAL., P. *Desarrollo Web Profesional*

Calidad en el desarrollo web, 2006. [2007]. Disponible en: http://www.pymeactual.com/desarrolloweb/calidad-desarrollo-web.php

CORPORATION, M. *Disciplina para la administración de proyectos MSF v. 1.1*, 2007. [2007]. Disponible en: http://www.microsoft.com/latam/technet/articulos/200304/art01/

CORPORATION, M. *Process Templates*, 2007. [2007]. Disponible en: http://msdn2.microsoft.com/en-us/teamsystem/aa718801.aspx

Desarrollo de Software., [PDF]. 2004. [2006]. Disponible en: http://www.fi.uba.ar/materias/7500/schenone-tesisdegradoingenieriainformatica.pdf

Documentación de Team Foundation 2007. [2007]. Disponible en: http://msdn2.microsoft.com/es-es/library/ms181232(VS.80).aspx

ESCRIBANO., G. F. *eXtreme Programming / Programación Extrema*, [PDF]. XP Overview, 2002. [2006]. Disponible en: http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf

FOWLER, M. *La Nueva Metodología*, programacionextrema.org, 2003. [Disponible en: http://www.programacionextrema.org/articulos/newMethodology.es.html

GREGORIO ROBLE, J. F. *Programación Extrema y Software Libre*, 2002. [2006]. Disponible en: http://www.ultimaorbita.com/raciel/x-ezine/x2/2x010-XP.html

HERNÁN., S. M. *Diseño de una Metodología Ágil de*

JFERRER. *Principios Ágiles*, 2005. [2006]. Disponible en: http://www.agile-spain.com/agilev2/principios_agiles

LOZADA'S, J. C. *La nueva estrategia de Control de Versiones de Microsoft: SourceSafe y parte de Visual 2005 Team System*, 2004. [2007]. Disponible en: http://blogs.msdn.com/juanlozv/

RENACIMIENTO. *DESARROLLO DE SOLUCIONES A MEDIDA*, 2007. [2007]. Disponible en:

http://www.renacimiento.com/sitios/idiomas/ESP/Paginas/desarrollo.aspx

S.A, I. P. G. *Presentación de Metodología MSF (Microsoft Solutions Framework)*, 2007]. Disponible en: http://www.e-gattaca.com/eContent/library/documents/DocNewsNo50DocumentNo6.PDF

WIKIMEDIA FOUNDATION, I. *Extreme Programming*, Wikimedia Foundation, Inc., 2007. [2006]. Disponible en: http://en.wikipedia.org/wiki/Extreme_Programming

WIKIMEDIA FOUNDATION, I. *Proceso Unificado de Rational*, Wikimedia Foundation, Inc., 2007. [2007]. Disponible en: http://es.wikipedia.org/wiki/Rational_Unified_Process

---. *Visual Studio Enterprise Tools Visual Studio 2005 Team System: Microsoft Solutions Framework* 2007 [2007]. Disponible en: http://msdn2.microsoft.com/en-us/library/aa302179.aspx

---. *Visual Studio Team System*, 2007 [2007]. Disponible en: http://www.microsoft.com/events/series/Msdnvsts.aspx#TeamFoundationServer

---. *Visual Studio Team System*

---. *Visual Studio Team System Microsoft Solutions Framework Team Model y Team System* 2007. [Disponible en: http://msdn2.microsoft.com/es-es/library/ms195024(VS.80).aspx

Glosario de Términos

***Métricas*:** sirven para recopilar información sobre el proyecto de manera tal que permita una mejor toma de decisiones.

Mindsets: colección de valores que determinan cómo los individuos interpretarán y responderán a las situaciones.

Release: lanzamiento de la versión definitiva de un producto final a menos que aparezcan errores que lo impidan

Stakeholders: partes involucradas en el proceso de desarrollo de un software que esperan recibir un beneficio del mismo. (usuarios, clientes, *managers*).

***Suite*:** en términos informáticos significa versión integral de un producto de software, en este caso del VSTS.

Work items: registros de la base de datos para asignar y seguir el trabajo de los miembros del proyecto.