Universidad de Sancti Spíritus "José Martí"
Facultad de Ingeniería
Carrera Informática

Estudio comparativo de metodologías de desarrollo para Aplicaciones Web con UML.

Trabajo de Diploma para optar por el título de Ingeniería Informática

Autor: Odainys de las Mercedes Valdés Gómez.

Tutor: Ing. Yanetsy Jiménez Hidalgo.

Consultante: Lic**.** Alain Pereira Toledo.

Sancti Spíritus, Cuba
Junio, 2014

Agradecimientos

- *A Dios Todopoderoso, porque siempre está conmigo.*
- *A mi esposo por siempre brindarme su amor incondicional y su apoyo en todo momento.*
- *A mis padres por su cariño y apoyo en los momentos más difíciles de mi vida.*
- *A mi tía Yamilet, por siempre preocuparse por mí.*
- *A la Ing. Yanetsy Jiménez Hidalgo y al Lic. Alain Pereira Toledo por su apoyo durante la realización y culminación de mi especialización.*
- *A todos mis profesores que hicieron posible mi formación como futuro profesional.*
- *A mis compañeros de grupo por su apoyo durante estos años, en especial a Yadira y Omar.*
- *A todas las personas que de una forma u otra siempre han estado a mi lado aportaron su granito de arena.*

Resumen

La presente investigación se realizó con el objetivo de comparar metodologías que utilizan UML para el desarrollo de aplicaciones web, abarcando no solo aquellas metodologías ya conocidas, como las metodologías tradicionales y las ágiles, sino también un grupo de metodologías creadas específicamente para el desarrollo de las aplicaciones web y que por su poca divulgación pocos desarrolladores conocen.

Para ello se realizó un estudio sobre las diferentes metodologías tradicionales, ágiles y web para el desarrollo de aplicaciones web que utilizan UML, se establecieron criterios para la comparación de las mismas y se realizó la comparación atendiendo a estos criterios.

Abstract

This research was performed with the aim of comparing methodologies using UML for developing web applications, including not only those already known methodologies such as traditional methodologies and agile, but also a group of methodologies created specifically for the development of web applications and their easy dissemination few developers know.

To do a study on the various traditional and agile methodologies for developing web applications using UML web was made, criteria for comparing them settled and comparison was performed in response to these criteria.

Índice

Índice de tablas

Índice de figuras

Introducción

Internet y la web han influido enormemente tanto en el mundo de la informática como en la sociedad en general. Centrándose en la web, en poco menos de diez años ha transformado los sistemas informáticos: ha roto las barreras físicas (debido a la distancia), económicas y lógicas (debido al empleo de distintos sistemas operativos, protocolos, entre otros y ha abierto todo un abanico de nuevas posibilidades. Una de las áreas que más expansión está teniendo en la web en los últimos años son las aplicaciones web.

Estas permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o el desarrollo del comercio electrónico. Además, una aplicación web permite entre otras cosas interactuar con los sistemas informáticos de gestión de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de una página web (Standing, 2010).

Aunque las aplicaciones web están creciendo rápidamente tanto en uso como en aceptación, sus resultados tienden a ser de poca calidad. La mayoría de los desarrolladores web ponen poca atención en el análisis de requisitos, así como en las metodologías y procesos de desarrollo. Además, los desarrolladores de aplicaciones confían excesivamente en el conocimiento y experticia de los desarrolladores individuales y sus prácticas de desarrollo individual más bien que en las prácticas estándar (Instituto Tecnológico de Veracruz, 2010).

Por esta razón, la complejidad del proceso de creación de aplicaciones web es netamente dependiente de la naturaleza del proyecto mismo, por lo que la metodología a utilizar estará acorde al nivel de aporte del proyecto, ya sea pequeño, mediano o de gran nivel y será en gran escala quien pueda conducir al programador a desarrollar un buen sistema de software (WikiUDO, 2012).

Es clave destacar que existen diferentes metodologías que utilizan UML (del inglés, Unified Modeling Language) como lenguaje de modelado.

UML es un lenguaje que permite modelar sistemas de información, y su objetivo es lograr modelos que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos por los clientes o usuarios de aquello que se modela (Rumbaugh, Jacobson, & Booch, 1998). Este lenguaje ha sido desarrollado por los más importantes autores en materia de análisis y diseño de sistemas y ha sido usada con éxito en sistemas hechos para toda clase de industrias alrededor del mundo: salud, bancos, comunicaciones, aeronáutica, finanzas, entre otros (Schmuller, 2011).

El modelado sirve no solamente para los grandes sistemas; aún en aplicaciones de pequeño tamaño se obtienen beneficios. Sin embargo, es un hecho que entre más grande y más complejo es el sistema, el modelado juega un papel más importante (Zamudio, Ramírez, Alvarez, & Rodríguez, 2009).

Para trabajar en la realización de software, de forma general, se crean grupos de investigación que encaminan su trabajo en líneas determinadas y desarrollan sistemas en muchos casos orientados a la web. Para la realización de estos, se utilizan metodologías de desarrollo de software, las cuales son diversas. Además de aquellas metodologías tradicionales y las ágiles, se ha elaborado una apreciable cantidad de metodologías para el desarrollo web. Esta diversidad dificulta aspectos claves en las estrategias tomadas por los desarrolladores, las cuales están relacionadas con la selección adecuada de una metodología.

En consecuencia, es necesario contar con un estudio comparativo que sirva de base para otras investigaciones sobre selección de metodologías, construcción de herramientas case basadas en ellas o para la elaboración de nuevas metodologías de desarrollo de aplicaciones.

Es por ello que se ha propuesto como ***problema de investigación:*** ¿Cómo comparar diferentes metodologías que utilicen UML para el desarrollo de aplicaciones web?

Para solucionar el problema identificado se traza como objetivo general:

Comparar diferentes metodologías que utilicen UML para el desarrollo de aplicaciones web mediante criterios de comparación***.***

El objetivo general antes enunciado da lugar a las **preguntas de investigación siguientes**:

1. ¿Cuáles son los fundamentos teóricos y metodológicos que sustentan la comparación de metodologías que utilicen UML para el desarrollo de aplicaciones web?
2. ¿Cómo comparar metodologías que utilicen UML para el desarrollo de aplicaciones web?

Para responder a estas preguntas de investigación se trazan las siguientes **tareas de investigación**:

1. Determinación de los fundamentos teóricos y metodológicos que sustentan la comparación de metodologías que utilizan UML para el desarrollo de aplicaciones web.
2. Comparación de las metodologías que utilizan UML para el desarrollo de aplicaciones web.

El presente trabajo está compuesto por dos capítulos, que contienen todo lo relacionado con el trabajo investigativo realizado. Además cuenta con introducción, conclusiones y bibliografía.

Capítulo I. Fundamentos para la comparación de metodologías de desarrollo para aplicaciones web con UML.

En este capítulo se exponen las diferentes clasificaciones de metodologías para el desarrollo de software, se realiza un estudio de las metodologías utilizadas para el desarrollo de aplicaciones web enfocándose en aquellas que utilizan UML y se exponen los criterios de comparación para realizar la comparación de las mismas.

Capítulo II. Comparación de las metodologías para el desarrollo de aplicaciones web con UML.

En este capítulo se realiza la comparación de las metodologías explicadas en el capítulo anterior para el desarrollo de aplicaciones web enfocadas al uso de UML, atendiendo a los criterios de comparación planteados.

Capítulo 1 Fundamentos para la comparación de metodologías de desarrollo para aplicaciones web con UML

1.1 Introducción

En este capítulo se exponen las diferentes clasificaciones metodologías para el desarrollo de software, y se realiza un estudio de las metodologías utilizadas para el desarrollo de aplicaciones web enfocándonos en aquellas que utilizan UML.

1.2 Metodologías para el desarrollo de software

Debido a que no existe un consenso entre los diferentes autores sobre el concepto de metodología de desarrollo de software se pudiera decir, de forma genérica, que es un conjunto de pasos que deben seguirse para el desarrollo de software.

Según (Laboratorio Nacional de Calidad del Software de INTECO, 2009) una metodología de desarrollo de software es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo.

Según (Gruzado Nuño, García de Egido, & Portugal Alonso, 2012) una metodología de desarrollo de software es un conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.

Una definición estándar de metodología puede ser: el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea.

Existen numerosas propuestas de metodologías para desarrollar software. Tradicionalmente estas metodologías se centran en el control del proceso, estableciendo rigurosamente las actividades, herramientas y notaciones al respecto, en consecuencia estas metodologías se caracterizan por ser rígidas y dirigidas por la documentación que se genera en cada una de las actividades desarrolladas.

La comparación y clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, la información disponible y alcance de cada

una de ellas. Si se toma como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, se pueden clasificar las metodologías en dos grupos: metodologías estructuradas y metodologías orientadas a objetos. Por otra parte, si se considera su filosofía de desarrollo se pueden diferenciar en metodologías tradicionales, ágiles y web, las cuales abarcará esta investigación.

1.2.1 Evolución histórica de las metodologías de desarrollo de software

Desde que se empezó a trabajar sobre el desarrollo de programas, se siguieron ciertos métodos que permitían llevar a producir un buen proyecto, estas metodologías aplicadas eran simples, solo se preocupaban por los procesos mas no por los datos, por lo tanto, los métodos eran desarrollados hacia los procesos. El modelo de procesos predominaba para los años 60 y consistía en *codificar y corregir* (Code-and-Fix), si al terminar se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar, este modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento. Los principales problemas del modelo de procesos son (WikiUDO, 2012):

- Los arreglos se hacen costosos, después de tantas correcciones el código tiene una mala estructura.
- El software no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
- El código es difícil de reparar por su pobre preparación para probar y modificar.

En la década de los setenta se le empezó a dar importancia a los datos, y para solucionar sistemas complejos empezó el análisis por partes o etapas, se introducen la planeación y administración. El modelo en cascada surge como respuesta al modelo de procesos, este modelo tiene más disciplina y se basa en el análisis, diseño, pruebas y mantenimientos (WikiUDO, 2012).

La década de los ochenta es la época marcada por las metodologías dirigida a datos, cuya importancia va tomando cuerpo en las organizaciones. Se empiezan a estudiar los objetos en sí como unidades de información. Para los años 90 se quiere dar respuesta al entorno siempre cambiante y en rápida evolución en que se han de desarrollar los programas informáticos, lo cual da lugar a trabajar en ciclos cortos (como mini-proyectos) que implementan una parte de las funcionalidades, pero sin perder el rumbo general del proyecto global.

Por otra parte, en los años 80 las metodologías de desarrollo comienzan a interesarse no sólo en lograr que el proyecto sea puesto en funcionamiento sino en minimizar costos durante su

desarrollo y sobre todo durante su mantenimiento. Los nuevos métodos van buscando minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo.

A continuación se muestran algunas de las metodologías utilizadas a nivel mundial en orden cronológico (WikiUDO, 2012):

(1970- 79)

- Programación Estructurada, desde 1975.

(1980-89)

- SSADM (del inglés, Structured Systems Analysis and Design Methodology), desde 1980.
- SADT (del inglés, Structured Analysis and Design Technique), desde 1980.
- Ingeniería de la Información (IE/IEM), desde 1981.

(1990-99)

- RAD (del inglés, Rapid Application Development), desde 1991.
- VFSM (del inglés, Virtual Finite State Machine), desde 1990.
- Dynamic Systems Development Method, desde 1995.
- FDD (del inglés, Feature Driven Development).
- RUP (del inglés, Rational Unified Process), desde 1999.

Año 2000 en adelante

- XP (del inglés, Extreme Programming), desde 1999.
- EUP (del inglés, Enterprise Unified Process), desde 2002.
- CDM (del inglés, Constructionist Design Methodology), desde 2004.
- AUP (del inglés, Agile Unified Process), desde 2005.

La trascendencia de las metodologías se ha hecho notoria, pasando de solo programar, luego a establecer funciones en etapas o módulos, continuando en la primera etapa el análisis de los requisitos, seguido de los objetos, y por último agilizar el desarrollo del software y minimizar los costos. Estos cambios de paradigma han logrado las mejoras para desarrollar software y principalmente buscando acortar el tiempo de solución sin reprochar los recursos disponibles.

1.3 Clasificación de las metodologías

Como se explicó anteriormente la presente investigación se va a enmarcar en las metodologías tradicionales, ágiles y las web, por esta razón se va a realizar una descripción de las mismas en este epígrafe.

1.3.1 *Metodologías tradicionales*

Las metodologías tradicionales centran su atención en llevar una documentación exhaustiva de todo el proyecto, en cumplir con un plan de proyecto, definido todo esto en la fase inicial del desarrollo del proyecto (Uñoja, 2012).

Otra de las características importantes dentro de este enfoque son los altos costos al implementar un cambio, al no ofrecer una buena solución para proyectos donde el entorno cambia.

Entre las principales metodologías tradicionales se tiene las ya tan conocidas RUP y MSF (Microsoft Solution Framework), pero existen otras como:

- Win – Win Spiral Model.
- Iconix.

1.3.2 *Metodologías ágiles*

Las metodologías ágiles están orientadas a proyectos pequeños, aportando una elevada simplificación, que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto (Amaro Calderón & Valverde Rebaza, 2009).

Algunos ejemplos de metodologías ágiles son:

- Scrum.
- XP.
- Crystal Clear.
- FDD.
- ASD (del inglés, Adaptive Software Development).
- XBreed.
- DSDM (del inglés, Dynamic Systems Development Method).

1.3.3 *Metodologías web*

Históricamente, la ingeniería de software ha intentado aplicar un enfoque planificado para el desarrollo, la operación y mantenimiento de sistemas, esto no ha incluido únicamente los aspectos técnicos de análisis, diseño, construcción y pruebas del software, si no también aspectos administrativos complementarios (planificación del proyecto, aseguramiento de calidad e implantación de sistemas). Hoy, esta disciplina impone nuevos e importantes desafíos a los actuales desarrolladores de sistemas y especialmente a las aplicaciones desarrolladas para la

web, que tienen nuevas y especiales características que hacen que los mecanismos empleados hasta el momento se deban adaptar.

De la necesidad de adecuar los procesos de la ingeniería de software tradicionales a este entorno de rápido y constante cambio, surge la ingeniería web. Esta se ha provisto de una serie de nuevas metodologías y herramientas para trabajar.

Algunos ejemplos de metodologías web son:

- OOHDM (del inglés, Object -Oriented Hypermedia Design).
- UWE (del inglés, UML Based Web Engineering).
- W2000.
- SODHM (del inglés, Scenario-based Object-oriented Hypermedya Design Methodology).

1.4 Metodologías para el desarrollo de aplicaciones web enfocadas al uso de UML

El desarrollo de aplicaciones web involucra decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo, afectando la división de tareas. Los problemas involucrados, como el diseño del modelo del dominio y la construcción de la interfaz de usuario, tienen requerimientos que deben ser tratados por separado.

El alcance de la aplicación y el tipo de usuario a los que estará dirigida son consideraciones tan importantes como las tecnologías elegidas para realizar la implementación. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también puede reducir su capacidad de usabilidad. El marco de desarrollo de este tipo de aplicaciones debe incluir un proceso general que tenga en cuenta de manera explícita las características particulares de las aplicaciones web (Silva & Mercerat, 2010).

Para todo esto se han desarrollado metodologías que utilizan UML para el modelado, estas permiten estructurar, comunicar, entender, simplificar y formalizar tanto el dominio del problema como las decisiones de diseño, así como disponer de una documentación detallada y exacta ante futuras modificaciones.

1.4.1 UML

UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y

controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios (Rumbaugh, Jacobson, & Booch, 1998).

El desarrollo de UML comenzó a finales de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software Corporation empezaron a unificar sus métodos. A finales de 1995, Ivar Jacobson y su compañía Objectory se incorporaron a Rational en su unificación, aportando el método OOSE (del inglés, Object Oriented Software Engineering) (Zamudio, Ramírez, Alvarez, & Rodríguez, 2009). En noviembre de 1997, después de pasar por el proceso de estandarización, el Object Management Group (OMG), entidad internacional creada en 1989 y dedicada al diseño orientado a objetos, publicó como estándar la versión 1.1 del Lenguaje Unificado de Modelado (Jacobson, Booch, & Rumbaugh, 2000). Posteriormente se han establecido nuevas especificaciones. Existe también una certificación para los distintos niveles profesionales de UML, ya que en su desarrollo intervienen numerosas empresas. El lenguaje UML se ha convertido en el lenguaje estándar utilizados por las industrias y las grandes empresas (OMG.org, 2012).

Las versiones de UML anteriores a la 2.0 ofrecen nueve tipos de diagramas y un conjunto de elementos de modelado para los tipos de diagramas (KOCH & KRAUS, 2002).

1.4.1.1 Objetivos de UML

Algunos de los objetivos que posee UML son los siguientes (Hans & Magnus, 2000):

- Proporcionar una notación y semánticas suficientes para poder alcanzar una gran cantidad de aspectos del modelado contemporáneo de una forma directa y económica.
- Proporcionar las semánticas suficientes para alcanzar aspectos del modelado que son de esperar en un futuro, como por ejemplo: aspectos relacionados con la tecnología de componentes, el cómputo distribuido, entre otros.
- Proporcionar mecanismos de extensión de forma que proyectos concretos puedan extender el metamodelo a un coste bajo.
- Proporcionar mecanismos de extensión de forma que aproximaciones de modelado futuras podrían desarrollarse encima de UML.
- Permitir el intercambio de los modelos entre una gran variedad de herramientas.
- Proporcionar semánticas suficientes para especificar las interfaces a bibliotecas para la comparación y el almacenamiento de componentes del modelo.

1.4.2 Metodologías tradicionales

En este epígrafe se abordarán algunas de las metodologías tradicionales que utilizan UML como lenguaje de modelado.

1.4.2.1 RUP

El proceso unificado utiliza UML como lenguaje de modelado, para preparar todos los esquemas de un sistema de software, de hecho, UML es una parte esencial del proceso unificado (Jacobson, Booch, & Rumbaugh, 2000).

1.4.2.1.1 Características de RUP

- **Iterativo e incremental**. Resulta muy práctico dividir el trabajo en piezas o mini-proyectos. El desarrollo de un producto software supone un gran esfuerzo que puede durar entre varios meses y en ocasiones hasta años, por lo que es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto (Jacobson, Booch, & Rumbaugh, 2000).
- **Centrado en la arquitectura**. La arquitectura es una vista del diseño con las características más importantes resaltadas, dejando los detalles de lado. Ofrece la forma del sistema y debe diseñarse de forma que este pueda evolucionar no únicamente de su desarrollo inicial, sino en futuras generaciones (Jacobson, Booch, & Rumbaugh, 2000).
- **Dirigido por casos de uso**. Los casos de uso representan los requerimientos funcionales, todos los casos de uso constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema, constituyen el punto de partida para las tareas de análisis y diseño y son la fuente para que el equipo de pruebas construya los casos de pruebas (Jacobson, Booch, & Rumbaugh, 2000).

1.4.2.1.2 Fases de RUP

El proceso unificado consta de ciclos que puede repetir a lo largo del ciclo de vida de un sistema. Un ciclo consiste en cuatro fases: inicio, elaboración, construcción y transición. Un ciclo concluye con una liberación, también hay versiones dentro de un ciclo (WikiUDO, 2012).

- **Fase de inicio:** durante la fase inicial se concibe la idea central del producto, se elabora el documento de visión. En esta fase, se revisan y confirma el entendimiento sobre los objetivos centrales del negocio. Se quiere entender los argumentos comerciales en favor

de porqué el proyecto debe intentarse. La fase de inicio establece la viabilidad del producto y delimita el alcance del proyecto. Se describe el producto final. Se responde a las preguntas:

- ¿Cuáles son las principales funciones del sistema para sus usuarios más importantes? La respuesta está en el modelo de casos de uso simplificado.
- ¿Cómo podría ser la arquitectura del sistema?
- ¿Cuál es el plan del proyecto y cuánto costará desarrollar el producto?

- **Fase de elaboración:** durante la fase de elaboración la mayoría de los casos de uso son especificados en detalle y la arquitectura del sistema es diseñada. Se identifican los riesgos significativos y se preparan el calendario, el equipo de trabajo y el costo del proyecto.
- **Fase de construcción:** durante la fase de construcción, el foco del producto se mueve de la arquitectura de base a un sistema lo suficientemente completo como para llevarlo al usuario. La línea base de arquitectura crece en complejidad y se convierte en un sistema completo, de la misma manera, se refina el diseño para llevarlo a código fuente. Se construye el producto, se utiliza la mayor parte de los recursos, y al finalizar se cubren todos los casos de uso. La pregunta que se realiza es: ¿Cubre el producto las necesidades de los usuarios como para hacer una primera entrega?
- **Fase de transición:** en la fase de transición el objetivo es garantizar que los requisitos se han cumplido, con la satisfacción de las partes interesadas. Esta fase a menudo se inicia con una versión beta de la aplicación. Otras actividades incluyen la preparación del ambiente, se completan, se identifican y corrigen defectos. La fase de transición termina con un cierre dedicado al aprendizaje de lecciones, las cuales quedan para futuros ciclos. El producto existe en versión beta.

1.4.2.1.3 Descripción de los diagramas de UML

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones. En concreto, un diagrama ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas. A continuación se exponen los nueves diagramas utilizados por RUP:

El diagrama de casos de usos que representa gráficamente los casos de uso que tiene un sistema. Se define un caso de uso como cada interacción supuesta con el sistema a desarrollar,

donde se representan los requisitos funcionales. Es decir, se está diciendo lo que tiene que hacer un sistema y cómo (Gomaa, 2011).

Figura 1.Diagrama de casos de uso

El diagrama de clases muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos.

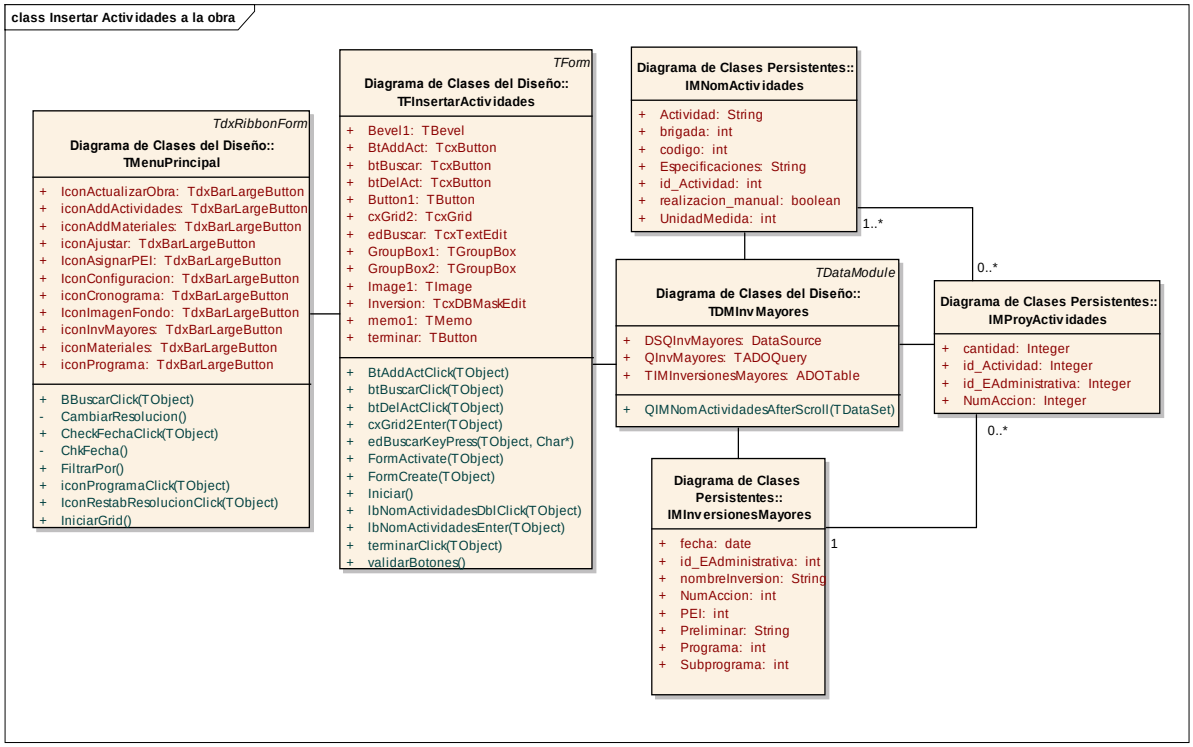


Figura 2. Diagrama de clases.

En el diagrama de secuencia se muestra la forma en que los objetos se comunican entre sí al transcurrir el tiempo. Consta de objetos que se representan del modo usual: rectángulos con

nombre (subrayado), mensajes representados por líneas continuas con una punta de flecha y el tiempo representado como una progresión vertical (Schmuller, 2011).

Un diagrama de secuencia UML muestra la interacción de objetos dispuestos en orden temporal. Presenta los objetos que participan en la interacción y la secuencia de los mensajes enviados entre ellos (KOCH & KRAUS, 2002).

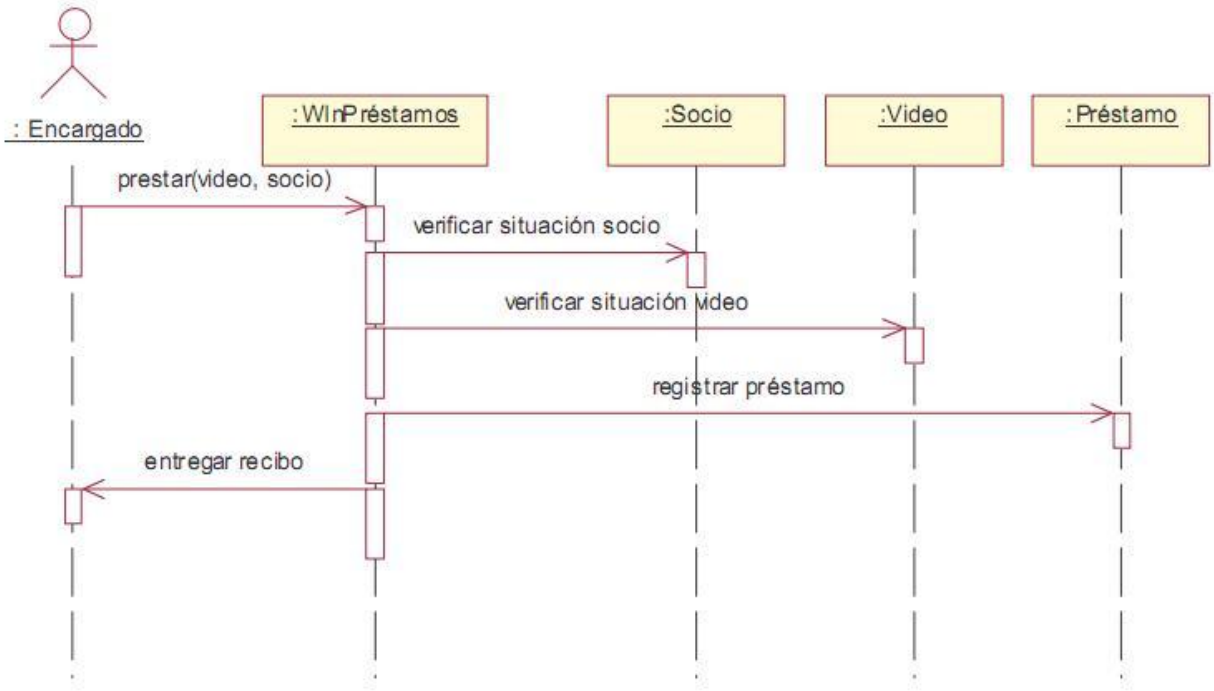


Figura 3. Diagrama de secuencia.

El diagrama de colaboración presenta una alternativa al diagrama de secuencia para modelar interacciones entre objetos en el sistema. Mientras que el diagrama de secuencia se centra en la secuencia cronológica del escenario que se está modelando, el diagrama de colaboración se centra en estudiar todos los efectos de un objeto dado durante un escenario. Los objetos se conectan por medio de enlaces, cada enlace representa una instancia de una asociación entre las clases implicadas. El enlace muestra los mensajes enviados entre los objetos, el tipo de mensaje (sincrónico, asincrónico, simple, blanking y 'time-out'), y la visibilidad de un objeto con respecto a los otros.

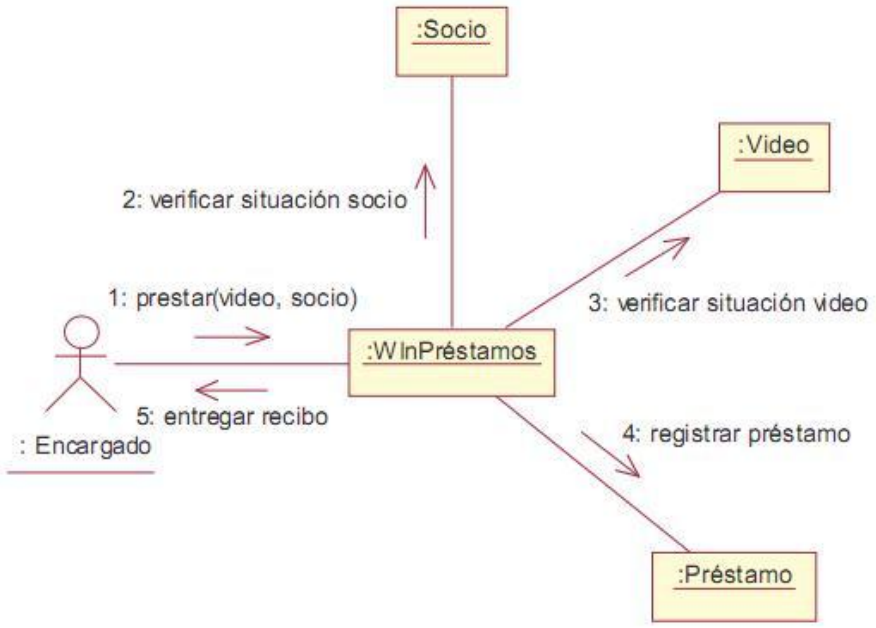


Figura 4. Diagrama de colaboración.

Los diagramas de estado presentan los estados en los que puede encontrarse un objeto junto con las transiciones entre los estados, y muestra los puntos inicial y final de una secuencia de cambios de estado (Schmuller, 2011). He aquí algunos ejemplos rápidos de cuando un objeto cambia su estado:

- Cuando acciona el interruptor, la fuente de luz cambia su estado de apagada a encendida.
- Cuando presiona un botón de un control remoto, una televisión cambia su estado para mostrarle un canal u otro.
- Luego de un lapso adecuado, una lavadora cambia su estado de lavar a enjuagar.

Los diagramas de actividades muestran el orden en el que se van realizando las tareas dentro de un sistema, agrega la dimensión de visualizar responsabilidades. Este tipo de diagrama es útil para representar las operaciones de un objeto y los procesos de negocio. El diagrama de actividades es una extensión del diagrama de estados (Schmuller, 2011).

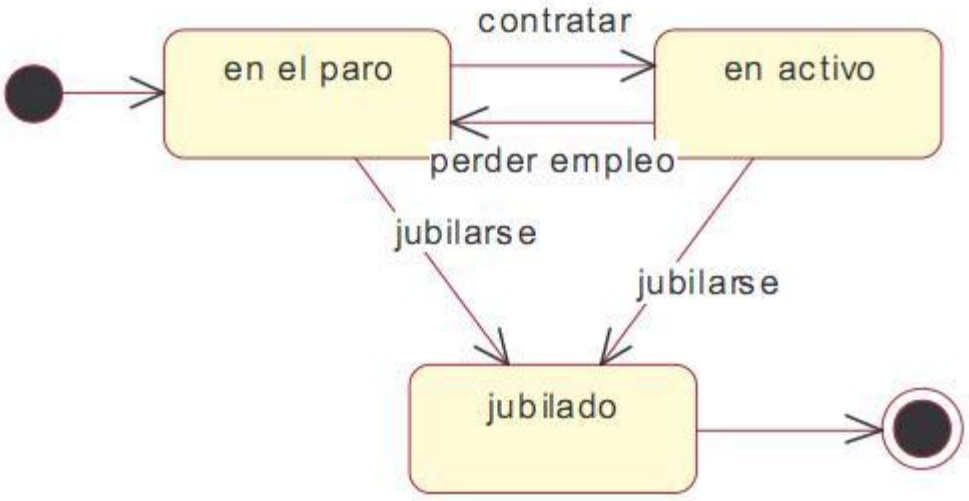


Figura 5. Diagrama de estados.

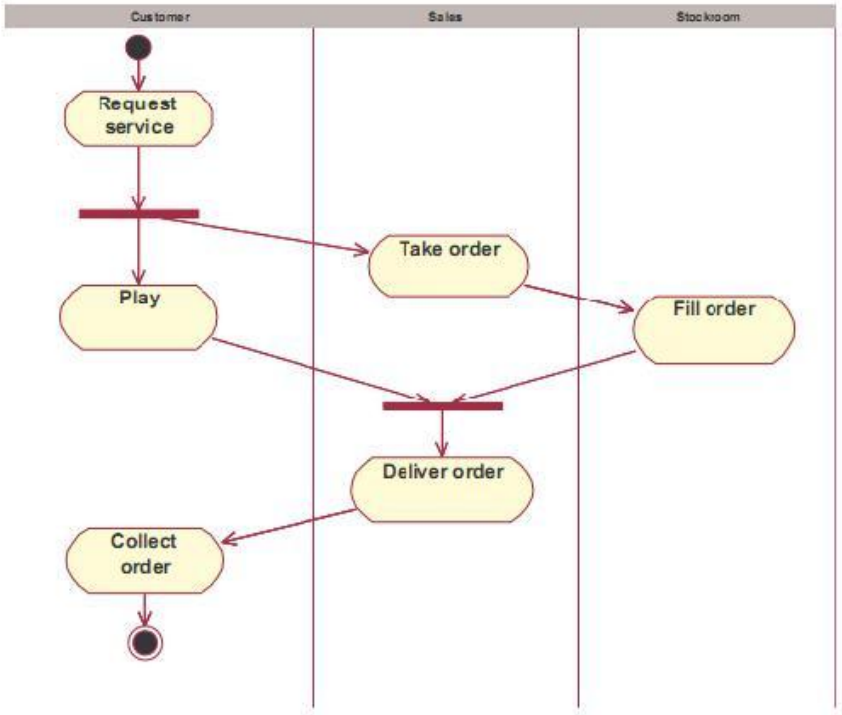


Figura 6. Diagrama de actividades.

Los diagramas de componentes expresan la organización lógica de la implementación de un sistema, contiene, obviamente, componentes, interfaces y relaciones. Estos diagramas representan a un elemento real: un componente de software (Schmuller, 2011).

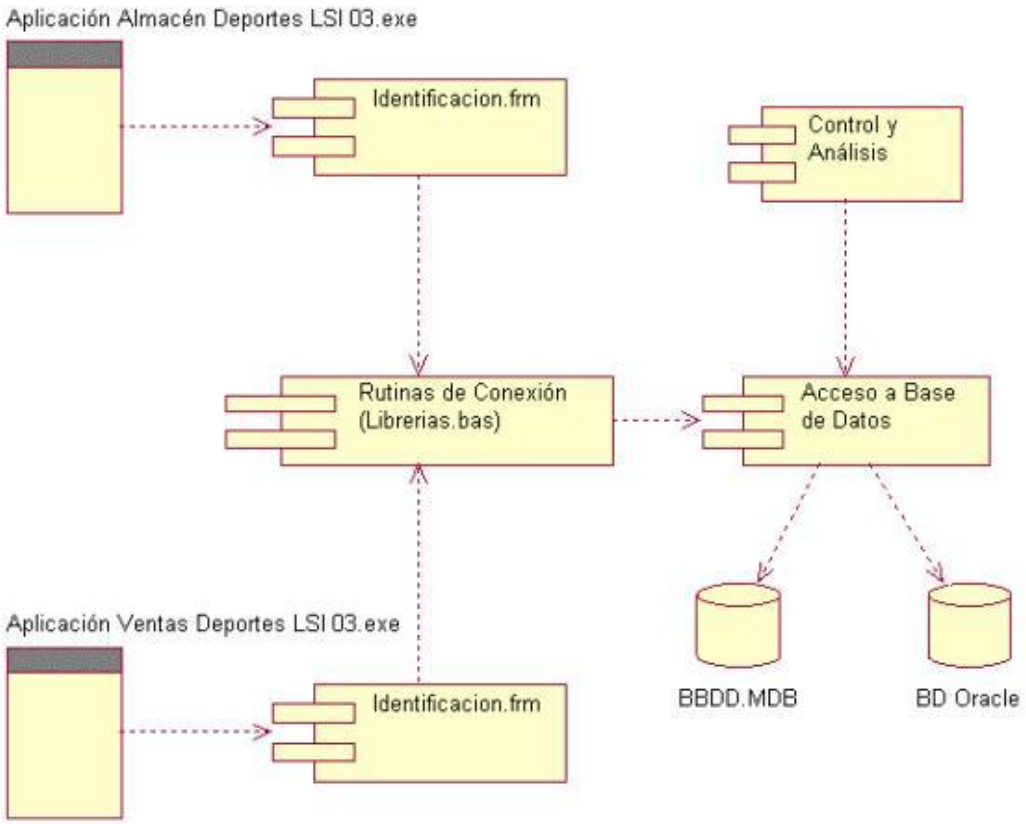


Figura 7. Diagrama de componentes.

Los diagramas de objetos están vinculados con los diagramas de clases. Un objeto es una instancia de una clase, por lo que un diagrama de objetos puede ser visto como una instancia de un diagrama de clases. Los diagramas de objetos describen la estructura estática de un sistema en un momento particular y son usados para probar la precisión de los diagramas de clases.

Los diagramas de despliegue expresan la configuración del sistema en tiempo de ejecución. La disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación (Zamuriano, 2010).

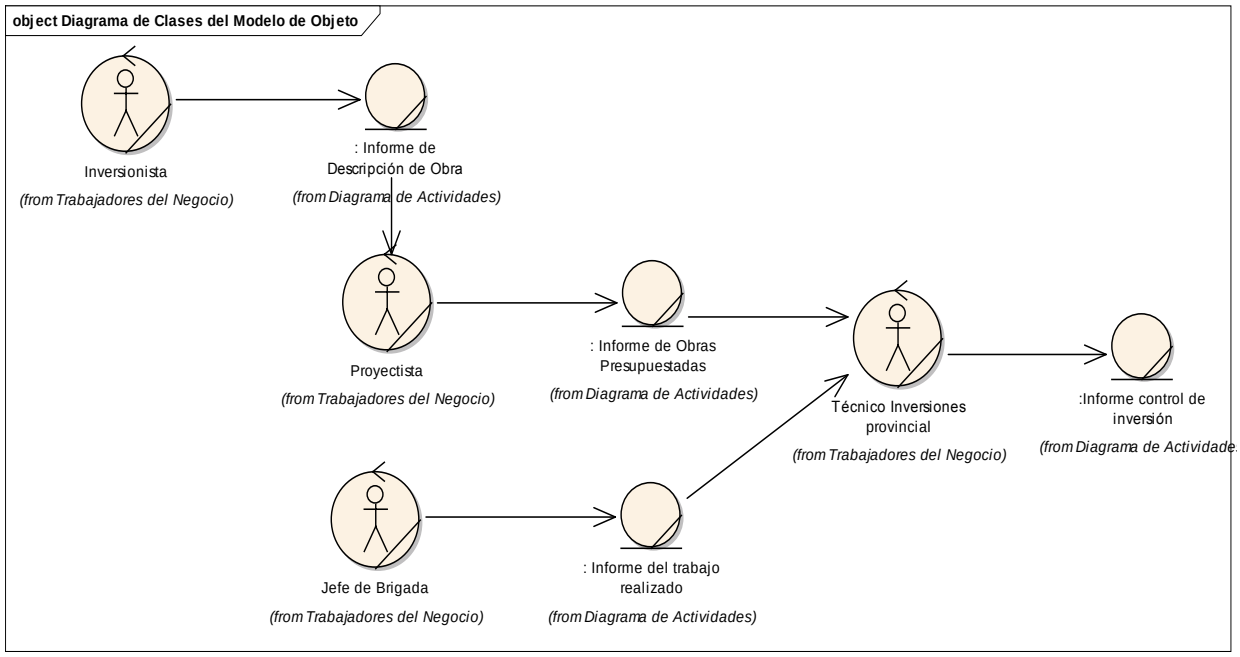


Figura 8. Diagrama de objetos.

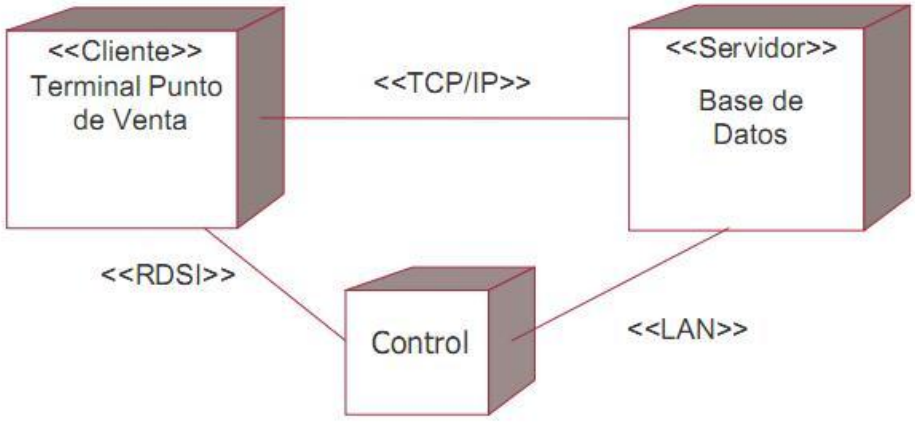


Figura 9. Diagrama de despliegue.

1.4.2.2 MS F

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas (Medín, 2009).

Figura 10. Metodología MSF.

1.4.2.2.1 Características de MSF

Algunas de las características de MSF son las siguientes (Medín, 2009):

• **Adaptable:** se puede utilizar para proyectos de cualquier magnitud.

• **Escalable:** puede organizar equipos tan pequeños entre tres o cuatro personas, así como también, proyectos que requieren 50 personas o más..

• **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.

•**Tecnología agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.4.2.2.2 Principios de MSF

Algunos de los principios de MSF son las siguientes (Microsoft Developer Network, 2013):

- **Promover comunicaciones abiertas:** para que el equipo sea eficaz y eficiente, debe compartir niveles adecuados de información entre los miembros del equipo. El equipo debe entender la naturaleza de lo que deben hacer y cómo se deben comunicar los miembros del equipo y a la vez estos con las personas externas.
- **Trabajar para una visión compartida:** tener una visión compartida capacita a los miembros del equipo y hace posible la agilidad de modo que los miembros del equipo puedan tomar decisiones informadas rápidamente en el contexto de lograr el objetivo de una perspectiva.
- **Establecer responsabilidades claras y compartidas:** los miembros del equipo autorizados se sienten más responsables por sus decisiones y están dispuestos a tener responsabilidad conjunta por el proyecto. Más responsabilidad del miembro del equipo conduce a una mayor calidad. Esto motiva a los miembros más fuertes de un equipo para que se ayuden entre ellos a llegar a ser tan buenos como pueden ser.

- **Permanecer ágil, y esperar los cambios:** dado que el cambio puede suceder a menudo en el peor de los momentos, si se dispone de una forma ágil de controlar el cambio, se ayuda a minimizar las interrupciones comunes producidas por el cambio. Mantenerse ágil significa que una organización está lista para el cambio y puede adaptarse y ajustarse fácilmente a él.
- **Invertir en calidad:** muchas organizaciones defienden la calidad, a menudo un término vagamente definido, pero carecen de los conocimientos para cuantificar la calidad. La calidad es algo que se debe incorporarse proactivamente en el ciclo de vida de la entrega de la solución, simplemente no sucede de forma espontánea.
- **Aprender de todas las experiencias:** si todos los niveles de una organización no aprenden de aquello que función anteriormente, ¿cómo pueden esperar mejorar la próxima vez? Los miembros del equipo deben entender y apreciar que el aprendizaje se da en todos los niveles:
    - A nivel de proyecto, por ejemplo, refinando un proceso para todo el proyecto.
    - A nivel individual, por ejemplo, cómo interactuar mejor con otros miembros del equipo.
    - A nivel de organización, como ajustar qué métricas de calidad se recopilan para cada proyecto.

1.4.2.2.3 Disciplinas de MSF

A continuación se exponen las diferentes disciplinas de MSF (Medín, 2009):

• **Gestión de proyectos**: describe el rol de la gestión del proyecto dentro del modelo de equipo de MSF, y cómo permitir mayor escalabilidad, desde proyectos pequeños a proyectos largos y complejos. Se basa en:

- Planificar sobre entregas cortas.
- Incorporar nuevas características sucesivamente.
- Identificar cambios ajustando el cronograma.

• **Control de riesgos**: diseñada para controlar los riesgos que puedan aparecer y ayudar al equipo a solucionarlo.

• **Control de cambios**: diseñada para que el equipo sea proactivo en lugar de reactivo. Los cambios deben considerarse riesgos inherentes y además deben registrarse y hacerse evidentes.

1.4.2.2.4 Modelos de MSF

MSF se compone de varios modelos encargados 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 riesgo, modelo de diseño de proceso y finalmente el modelo de aplicación (Medín, 2009).

• **Modelo de arquitectura del proyecto**: diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

• **Modelo de equipo**: este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

Figura 11. Modelo de equipo de trabajo de MSF.

El modelo de equipos de MSF divide actividades y responsabilidades de entrega de soluciones típicas en seis grupos de apoyo. Estos grupos son interdependientes y multidisciplinarios. Como se representa en la tabla siguiente, para ayudar a tener un enfoque equilibrado, estos roles incorporan cada uno una perspectiva única sobre lo que se necesita, lo que se debe recomendar y cuáles deben ser los objetivos asociados a la entrega de una solución. Estos

roles se pueden combinar para escenarios de equipos pequeños y expandir para escenarios de equipos grandes.

Roles	Tareas
Administrador de Producto	• Debe asegurarse de que la solución ofrezca un valor empresarial. • Definir la solución dentro de las restricciones del proyecto. • Debe asegurarse de que las necesidades y expectativas de los clientes se satisfagan.
Administrador de Programa	• Entregar la solución dentro de las restricciones del proyecto. • Configurar los medios que permiten cumplir las necesidades y expectativas del patrocinador.
Diseñador	• Diseñar una solución para satisfacer los objetivos empresariales en las restricciones del proyecto.
Desarrollador	• Compilar la solución según especificación.
Probador	• Aprueba la solución para su lanzamiento solo después de asegurarse de que todos los aspectos de la solución cumplen sus niveles de calidad respectivos que se hayan definido.

Tabla 1. Tareas y roles del modelo de equipos de MSF.

• **Modelo de proceso**: diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad, acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el modelo de equipo.

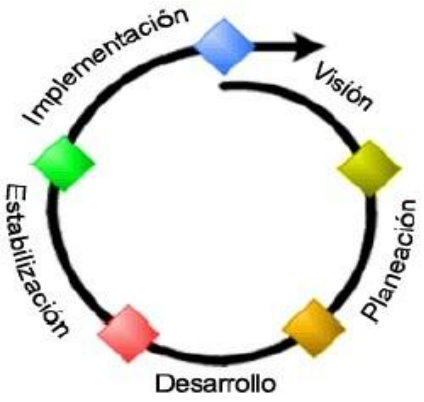


Figura 12. Modelo de proceso de MSF.

• **Modelo de gestión de riesgo**: diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones, valorando los riesgos que puedan provocar.

• **Modelo de diseño de proceso**: diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo.

• **Modelo de aplicación**: diseñado para mejorar el desarrollo, el mantenimiento y el soporte. Proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones de software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

1.4.2.2.5 Fases de MSF

MSF posee cuatro fases, las cuales se explicarán a continuación (Peña Pérez, Jiménez Ruiz, Valverde Pérez, Aramayo Cuellar, & Salazar Zorrilla, 2012):

• **Visión y alcance**: la fase de visión y alcance trata la unificación del equipo detrás de una visión común. Se definen los líderes y responsables del proyecto, adicionalmente se identifican las metas y objetivos a alcanzar; estas últimas se deben respetar durante la ejecución del proyecto en su totalidad, y se realiza la evaluación inicial de riesgos del proyecto.

Documentos a entregar:

- Documento visión.
- Documento detalle de la visión.

- Documento de requerimientos funcionales.
- Documento matriz de riesgos.
- Acta de aprobación de visión.

• **Planificación**: es en esta fase cuando la mayor parte de la planeación para el proyecto es terminada. El equipo prepara las especificaciones funcionales, realiza el proceso de diseño de la solución, y prepara los planes de trabajo, estimaciones de costos y cronogramas de los diferentes entregables del proyecto.

Documentos a entregar:

- Documento de cronograma.
- Acta de aprobación de cronograma.

• **Desarrollo**: durante esta fase el equipo realiza la mayor parte de la construcción de los componentes (tanto documentación como código).

Documentos a entregar:

- Documentos, manuales, entre otros.
- Acta de finalización de desarrollo.

• **Estabilización**: en esta fase se conducen pruebas sobre la solución, las pruebas de esta etapa enfatizan el uso y operación bajo condiciones realistas.

Documentos a entregar:

- Documento registro de pruebas.
- Acta de aprobación de versión aprobada.

• **Implantación**: durante esta fase el equipo implanta la tecnología base y los componentes relacionados, estabiliza la instalación, traspasa el proyecto al personal de soporte y operaciones, y obtiene la aprobación final del cliente.

Documentos a entregar:

- Conjunto de archivos (ejecutables, directorios, base de datos, scripts, instaladores, manuales, licencias, entre otros) que permitan su instalación y correcto funcionamiento.
- Acta de entrega y finalización de proyecto.

1.4.2.2.6 Ventajas de MSF

Algunas de las ventajas que ofrece MSF son (Peña Pérez, Jiménez Ruiz, Valverde Pérez, Aramayo Cuellar, & Salazar Zorrilla, 2012):

- Crea una disciplina de control de riesgos que ayuda y evoluciona con el proyecto.

- Orientado al trabajo en equipo.
- Tiene facilidad de soporte y mantenimiento.
- Es adaptable, se puede utilizar para proyectos de cualquier magnitud.
- Aplica mucho e incentiva el trabajo en equipo y la colaboración.
- Permite la reutilización de componentes ya desarrollados en ciclos anteriores.
- Es un modelo enfocado a los requerimientos del usuario.
- Es una metodología que se puede ajustar a equipos de trabajo compuestos por tres o más personas.

1.4.2.2.7 Desventajas de MSF

Algunas de las desventajas que ofrece MSF son (Peña Pérez, Jiménez Ruiz, Valverde Pérez, Aramayo Cuellar, & Salazar Zorrilla, 2012):

- Al estar basado en tecnología Microsoft, trata de obligar a usar sus propias herramientas.
- Solicita demasiada documentación en sus fases.
- Si el control de riesgos se hace muy exhaustivo puede retrasar el proyecto.
- Los precios de licencias, capacitación y soporte de Microsoft son caros.
- Alto grado de dependencias de tecnologías propietarias.

1.4.2.3 Iconix

La metodología Iconix se define como un proceso de desarrollo de software práctico. Está entre la complejidad de RUP y la simplicidad y pragmatismo de XP, sin eliminar las tareas de análisis y diseño que XP no contempla (Brito Acuña, 2009).

En esta metodología se busca tener una retroactividad con el cliente en la mitad de los procedimientos, comenzando con un prototipo en donde el analista y el cliente definirán pantallas y funcionalidades, en sí lo que se espera obtener del programa. Además se definirán los modelos de casos de uso, de secuencia y de robustez, con la finalidad de conseguir un buen sistema.

Esta metodología es un proceso simplificado en comparación con otros procesos más tradicionales, que unifica un conjunto de métodos de orientación a objetos con el objetivo de abarcar todo el ciclo de vida de un proyecto. Iconix presenta claramente las actividades de cada fase y exhibe una secuencia de pasos que deben ser seguidos. Además, está adaptado a

patrones, ofrece el soporte a UML, es dirigido por casos de uso y es un proceso iterativo e incremental.

1.4.2.3.1 Características de Iconix

A continuación se abordan algunas de las características de Iconix (Brito Acuña, 2009):

• **Iterativo e incremental:** varias interacciones ocurren entre el modelo del dominio y la identificación de los casos de uso.

• **Trazabilidad:** cada paso está referenciado por algún requisito. Se define la trazabilidad como la capacidad de seguir una relación entre los diferentes artefactos producidos

• **Dinámica del UML:** la metodología ofrece un uso dinámico del UML, como los diagramas del caso de uso, diagramas de secuencia y de colaboración.

1.4.2.3.2 Fases de Iconix

Iconix posee cuatro fases, las cuales se explican a continuación (Bona, 2010):

• **Análisis de requisitos:** en esta fase se identifican los objetos y todas las relaciones de agregación y generalización entre ellos. Se deben analizar todos los requisitos que formarán parte del sistema y con estos construir el diagrama de clases, que representa las agrupaciones funcionales que estructurarán el sistema en desarrollo.

Para esta fase se utilizan tres modelos:

- ***Modelo de dominio*:** esto se refiere a identificar objetos y cosas del mundo real que intervienen en el sistema.
- ***Modelo de casos de uso*:** este modelo es usado para representar las exigencias de los usuarios, describiendo las acciones que este realiza dentro del sistema (Anízio Maia, 2013).
- ***Prototipo de interfaz de usuario*:** implica la creación de un modelo o modelos operativos del trabajo de un sistema, en el que analistas y clientes deben estar de acuerdo. Los usuarios se hacen participantes activos en el desarrollo.

• **Análisis y diseño preliminar:** en esta fase a partir de cada caso de uso se obtendrán una ficha de caso de uso, (la cual no pertenece a UML), está formada por un nombre, una descripción, una precondición que debe cumplir antes de iniciarse, una post-condición que debe cumplir al terminar si termina correctamente.

En esta fase se realizan los siguientes diagramas: el diagrama de robustez y el diagrama de clases.

Diagrama de robustez: un diagrama de robustez es un híbrido entre un diagrama de clases y un diagrama de actividades. Es una herramienta que nos permite capturar el Qué hacer y a partir de eso él cómo hacerlo. Facilita el reconocimiento de objetos y hace más sencilla la lectura del sistema. Ayuda a identificar los objetos que participan en cada caso de uso.

El diagrama de robustez se divide en:

- **Objetos fronterizos:** usado por los actores para comunicarse con el sistema.
- **Objetos entidad:** son objetos del modelo del dominio.
- **Objetos de control:** es la unión entre la interfaz y los objetos de entidad.

Diagrama de clases*:* describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos

• Diseño: en esta fase se reconocen todos los elementos que forman parte de nuestro sistema.

Diagramas de secuencia*:* muestra los métodos que llevaran las clases de nuestro sistema. Muestra todos los cursos alternos que pueden tomar todos nuestros casos de uso. Se debe terminar el modelo estático, añadiendo los detalles del diseño en el diagrama de clases.

• Implementación: en esta fase a partir del buen diseño logrado se creará el software; que posteriormente se entregará.

1.4.3 Metodologías ágiles

En marzo de 2001, 17 críticos de los modelos de producción basados en procesos, convocados por Kent Beck se reunieron en Salt Lake City para discutir sobre el desarrollo de software, en esta reunión se planteó el término “Métodos Ágiles” para definir a aquellos que estaban surgiendo como alternativa a las metodologías pesadas y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo. Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que son los valores sobre los que se asientan estos métodos (Body of Knowledge, 2014).

Según el Manifiesto Ágil se valora (Body of Knowledge, 2014):

- **Los individuos y su interacción son más importantes los procesos y las herramientas:** este es el valor más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Las herramientas mejoran la eficiencia, pero hay tareas que requieren talento y necesitan personas que lo aporten y trabajen con una actitud adecuada.
- **Valoramos más el software que funciona que la documentación exhaustiva:** poder anticipar cómo será el funcionamiento del producto final, observando prototipos previos,

o partes ya elaboradas ofrece una retroalimentación estimulante y enriquecedora, que genera ideas imposibles de concebir en un primer momento, y difícilmente se podrían incluir al redactar un documento de requisitos detallado en el comienzo del proyecto. El manifiesto ágil no da por inútil la documentación, sólo la de la documentación innecesaria. Los documentos son soporte de hechos, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero su relevancia debe ser mucho menor que el producto final.

- **La colaboración con el cliente más que la negociación de un contrato:** las prácticas ágiles están indicadas para productos cuyo detalle resulta difícil prever al principio del proyecto; y si se detallara al comenzar, el resultado final tendría menos valor que si se mejoran y precisan con retroinformación continua durante él. También son apropiadas cuando se prevén requisitos inestables por la velocidad de cambio en el entorno de negocio del cliente. El objetivo de un proyecto ágil no es controlar la ejecución conforme a procesos y cumplimiento de planes, sino proporcionar el mayor valor posible al producto. Resulta por tanto más adecuada una relación de implicación y colaboración continua con el cliente, más que un contrato que delimita las responsabilidades.
- **La respuesta al cambio antes que el seguimiento de un plan:** para desarrollar productos de requisitos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que el de seguimiento y aseguramiento de planes. Los principales valores de la gestión ágil son la anticipación y la adaptación.

Los valores anteriores inspiran los 12 principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto a metas a seguir y organización del mismo.

Los 12 principios del manifiesto ágil son (Body of Knowledge, 2014):

- La prioridad principal es satisfacer al cliente a través de la entrega temprana y continua de software que aporten un valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo: los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia de software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.

- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y aportándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido: los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

1.4.3.1 XP

A mediados de la década de 1980, Kent Beck y Ward Cunningham (dos programadores) trabajaban en un grupo de sistemas de prueba de semiconductores del Tektronix Computer Research Laboratory; allí idearon las tarjetas CRC3 y sentaron las bases de lo que después serían los patrones de diseño y XP. En su economía sintáctica y en su abstracción, prefiguran a lo que más tarde serían las tarjetas de historias de XP. Beck y Cunningham prohibían escribir en las tarjetas más de lo que en ellas cabía. XP se funda en cuatro valores: comunicación, simplicidad, retroalimentación y coraje. Pero tan conocidos como sus valores son sus prácticas (Pollice, Augustine, Lowe, & Madh, 2004).

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (Bautista, 2010).

1.4.3.1.1 Prácticas XP

Las prácticas de XP se muestran a continuación (Amaro Calderón & Valverde Rebaza, 2009):

- **El juego de la planificación:** es un espacio frecuente de comunicación entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario, los clientes deciden sobre el ámbito, el tiempo de las entregas y de cada iteración.
- **Entregas pequeñas:** producir rápidamente versiones del sistema que sean operativas, aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema, pero sí que constituyan un resultado de valor para el negocio. Una entrega no debería tardar más tres meses.
- **Metáfora:** el sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.
- **Diseño simple:** se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas:** la producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema.
- **Refactorización (*refactoring*):** la refactorización es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. La refactorización mejora la estructura interna del código sin alterar su comportamiento externo.
- **Programación en parejas:** la programación se realiza en parejas. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, entre otros).
- **Propiedad colectiva del código:** cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- **Integración constante:** cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

- **40 horas por semana:** se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los plazos, suelen al final, ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.
- **Cliente in-situ:** el cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio, y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita, ya que esta última toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada.
- **Estándares de programación:** XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada, puesto que se apoyan unas en otras. Esto se ilustra en la figura 13, donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí.

La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica para un análisis histórico de ideas y prácticas, que sirven como antecedentes a las utilizadas por las metodologías ágiles. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

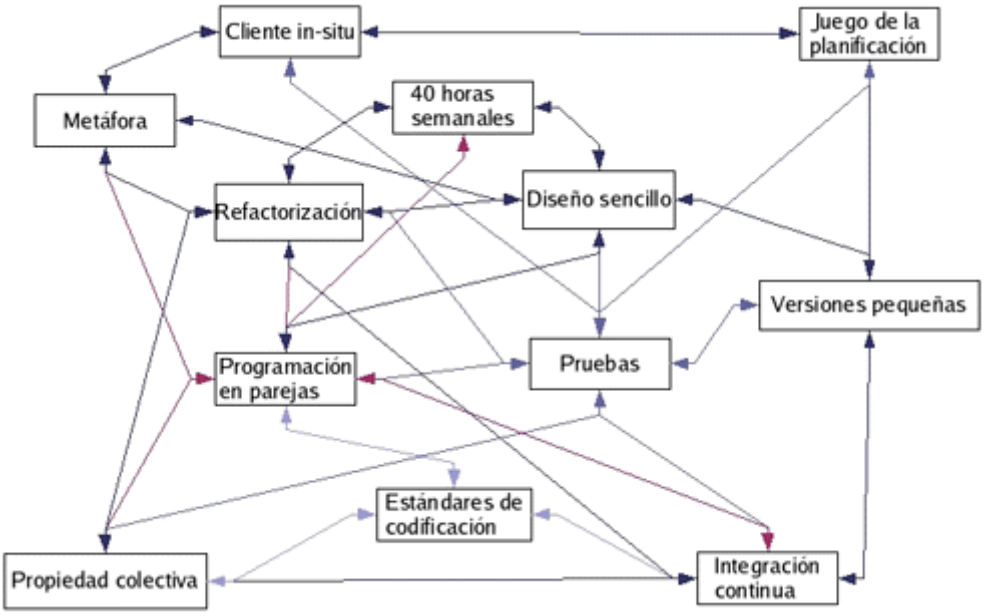
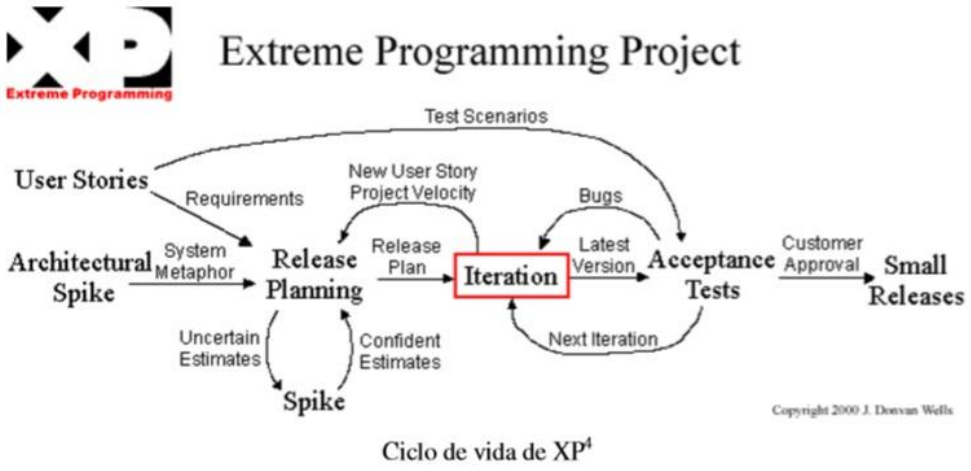


Figura 13. Las prácticas XP se refuerzan entre sí.

1.4.3.1.2 Fases de XP

El ciclo de vida es iterativo. El siguiente diagrama describe su cuerpo principal (Letelier & Penadés, 2010):



Ciclo de vida de XP[4]

Figura 14. Ciclo de vida.

El ciclo de vida ideal de XP consiste de seis fases: exploración, planificación de la entrega, iteraciones, producción, mantenimiento y muerte del proyecto.

- **Exploración**: en esta fase los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración dura pocas semanas o pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.
- **Planificación de la entrega**: en esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de uno a tres puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar el alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

- **Iteraciones:** esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. En ella se realizan dos planes: el plan de entrega y el plan de la iteración:

El plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del plan de la iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

- **Producción:** la fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).
- **Mantenimiento:** mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.
- **Muerte del proyecto:** es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

1.4.3.1.3 Procesos XP

El ciclo de desarrollo de XP consiste en (Medín, 2009):

- El cliente define el valor del negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente seleccionará que construir, de acuerdo con sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al principio.

Entre los artefactos que se utilizan en XP vale la pena mencionar las historias de usuario, estas, describen la funcionalidad de un sistema de software que aporta valor al usuario y/o al cliente. Es una técnica utilizada para especificar los requisitos del software. Son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe tener, ya sean éstos requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que el programador la pueda implementar en unas semanas, además estas tarjetas son dinámicas y flexibles de tratar.

1.4.3.1.4 Roles XP

Según la propuesta original de Beck son:

- **Programado**r: escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración, centrándose en aportar mayor valor al negocio.
- **Encargado de pruebas**: ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento**: proporciona retroalimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador:** es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las practicas XP y se siga el proceso correctamente.
- **Consultor**: es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- **Gestor:** es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

1.4.3.1.5 Ventajas y desventajas de XP

A continuación se presentan las ventajas y desventajas de XP (Bautista, 2010):

Ventajas:

- Programación organizada.

- Menor taza de errores.
- Satisfacción del programador.
- Solución de errores de programas
- Versiones nuevas.
- Implementa una forma de trabajo que se adapta fácilmente a las circunstancias

Desventajas:

- Es recomendable emplearlo solo en proyectos a corto plazo.
- Altas comisiones en caso de fallar.
- Imposible prever todo antes de programar.
- Demasiado costoso.

1.4.3.2 Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle en los años 90. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante varios años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto. Estas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración (Letelier & Penadés, 2010).

Scrum, más que una metodología de desarrollo software, es una forma de auto-gestión de los equipos de programadores. En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales (Proyectosagiles.org, 2013).

1.4.3.2.1 Roles de Scrum

Los roles en Scrum son seis (Rivadeneira Molina, 2012):

- **Scrum master (o Facilitador):** debe interactuar con el equipo, el cliente y los gestores. Garantiza el funcionamiento de los procesos y la metodología. Debe ser miembro del equipo y trabajar a la par. Coordina los encuentros diarios y se encarga de eliminar obstáculos.
- **Propietario del producto:** es el responsable oficial del proyecto, gestión, control y visibilidad del product backlog. Es elegido por el scrum master, el cliente y los ejecutivos.
- **Equipo de desarrollo:** tiene autoridad para reorganizarse y definir acciones necesarias o sugerir eliminar problemas para cumplir con los objetivos del sprint.
- **Cliente:** participa en la creación del product backlog. Se refiere a las personas para quien el proyecto producirá el beneficio acordado que lo justifica.
- **El gestor:** toma las decisiones finales, participa en la elección de objetivos y requisitos. Selecciona el propietario del producto.
- **El usuario:** es el destinatario final del producto. Como bien lo dice la paradoja, el árbol cae en el bosque cuando no hay nadie ¿Hace ruido? Aquí la definición sería si el software no es usado ¿fue alguna vez escrito?

1.4.3.2.2 Proceso de Scrum

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento del producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite (Proyectosagiles.org, 2013).

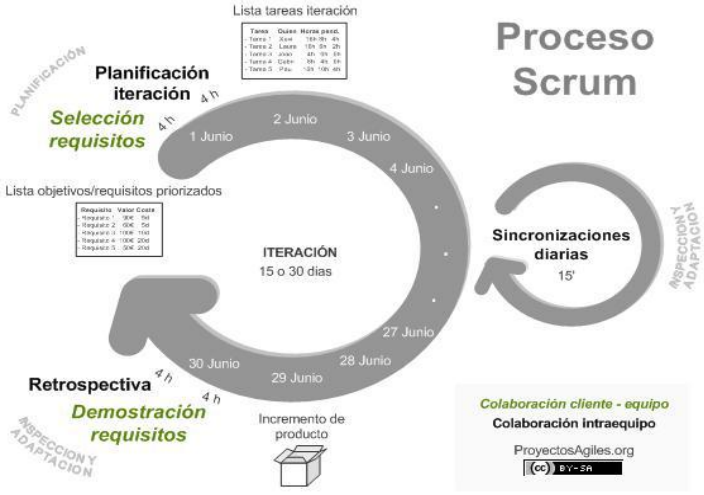


Figura 15. Ciclo de vida de Scrum.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas. De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla y el retorno de inversión mediante la re-planificación de objetivos del producto, que realiza durante la iteración con vista a las siguientes iteraciones.

Las actividades que se llevan a cabo en Scrum son las siguientes:

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Esta tiene dos partes (Proyectosagiles.org, 2013):

- **Selección de requisitos** (cuatro horas máximo). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
- **Planificación de la iteración** (cuatro horas máximo). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se auto asignan las tareas.

Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización(15 minutos máximos). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración y obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:

- ¿Qué he hecho desde la última reunión de sincronización?
- ¿Qué voy a hacer a partir de este momento?
- ¿Qué impedimentos tengo o voy a tener?

Durante la iteración, el Facilitador se encarga de que el equipo pueda cumplir con su compromiso, de que no se disminuya su productividad, elimina los obstáculos que el equipo no puede resolver por sí mismo y protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Esta tiene dos partes (Proyectosagiles.org, 2013):

- **Demostración** (cuatro horas máximo). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que hayan ocurrido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
- **Retrospectiva** (cuatro horas máximo). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

1.4.3.2.3 Documentos de Scrum

En la metodología Scrum se elaboran tres documentos:

- **Lista de objetivos / requisitos priorizada (product backlog):** esta representa la visión y expectativas del cliente respecto a los objetivos y entregas del producto o proyecto. El cliente es el responsable de crear y gestionar la lista (con la ayuda del Facilitador y del equipo, quien proporciona el coste estimado de completar cada requisito). Dado que reflejar las expectativas del cliente, esta lista permite involucrarle en la dirección de los resultados del producto o proyecto (proyectosagiles.org, 2013).
    - Contiene una breve descripción de todas las funciones deseadas en el producto (Mountain Goat Software, 2012).
    - En la lista se indican las posibles iteraciones y las entregas esperadas por el cliente (los puntos en los cuales desea que se le entreguen los objetivos/requisitos completados hasta ese momento), en función de la velocidad de desarrollo del (los) equipo(s) que trabajará(n) en el proyecto. Es conveniente que el contenido de cada iteración tenga una coherencia, de manera que se reduzca el esfuerzo de completar todos sus objetivos.
    - La lista también tiene que considerar los riesgos del proyecto e incluir los requisitos o tareas necesarios para mitigarlos.

- **Lista de tareas de la iteración (sprint backlog):** es elaborada por el equipo en la reunión de planificación de la iteración (Sprint Planning) como plan para completar los objetivos/requisitos seleccionados para la iteración y que se compromete a demostrar al cliente al finalizar la iteración, en forma de incremento de producto preparado para ser entregado. Esta lista permite ver las tareas donde el equipo está teniendo problemas y no avanza, con lo que le permite tomar decisiones al respecto. Para cada uno de los objetivos/requisitos se muestran sus tareas, el esfuerzo pendiente para finalizarlas y la auto-asignación que han hecho los miembros del equipo (proyectosagiles.org, 2013).
- **Gráficos de trabajo pendiente (burndown charts):** es un gráfico de trabajo pendiente que muestra la velocidad a la que se está completando los objetivos/requisitos. Permite extrapolar si el Equipo podrá completar el trabajo en el tiempo estimado (proyectosagiles.org, 2013).
- Se pueden utilizan los siguientes gráficos de esfuerzo pendiente:
    - Días pendientes para completar los requisitos del producto o proyecto (product burndown chart), realizado a partir de la lista de requisitos priorizada (product backlog).
    - Horas pendientes para completar las tareas de la iteración (sprint burndown chart), realizado a partir de la lista de tareas de la iteración (iteration backlog).

1.4.3.2.4 Ventajas de Scrum

A continuación se brindan algunas de las ventajas de Scrum (Proyectalis, 2014):

- Permite a las organizaciones eliminar los impedimentos clásicos en el desarrollo de los proyectos, aumentando la satisfacción de los clientes mediante la realización de entregas frecuentes de resultados tangibles e integrándolos activamente en el ciclo de desarrollo, lo cual proporciona entre otras una mayor adaptación y adecuación a sus necesidades.
- Potencia la formación de equipos de trabajo autosuficiente y multidisciplinario, reduciendo la carga de gestión y proporcionando a los miembros del equipo un entorno amigable y productivo para desarrollar sus habilidades al máximo. Este entorno proporciona, además, mayor calidad de vida a los trabajadores y mejora drásticamente la moral en las organizaciones.
- Se centra en el producto y las personas, y hace especial hincapié en la eliminación proactiva de todas las trabas e impedimentos que surjan durante el desarrollo. Así pues,

permite a muchas organizaciones alcanzar el llamado “Efecto Toyota”: cuatro veces la productividad media del sector, con doce veces la calidad.

- Es simple, aunque duro. Es sencillo combinar Scrum con otras metodologías y marcos de gestión de proyectos en las organizaciones.

1.4.3.3 Crystal Clear

Alistair Cockburn es el propulsor detrás de la serie de metodologías Crystal. Las mismas presentan un enfoque ágil, con gran énfasis en la comunicación, y con cierta tolerancia que las hace ideal en los casos en que sea inaplicable la disciplina requerida por XP. Crystal Clear es la encarnación más ágil de la serie, se define con mucho énfasis en la comunicación, y de forma muy liviana en relación a los entregables. Crystal maneja iteraciones cortas con retroalimentación frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios. Otra de las cuestiones planteadas es la necesidad de disponer de un usuario real aunque sea de forma parcial para realizar validaciones sobre la interface del usuario y para participar en la definición de los requerimientos funcionales y no funcionales del software (Cockburn, 2013).

Las personas involucradas escogen aquellos principios que les resultan efectivos, mediante la aplicación de la metodología en diversos proyectos, agregan o remueven principios en base al consenso grupal del equipo de desarrollo.

La familia Crystal dispone un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más pesado es el método. Cuanto más crítico es un sistema, más rigor se requiere. El código cromático se aplica a una forma tabular elaborada por Cockburn que se usa en muchas metodologías ágiles para situar el rango de complejidad al cual se aplica una metodología. En la Figura Nº16 se muestra una evaluación de las pérdidas que puede ocasionar la falla de un sistema y el método requerido según este criterio. Los parámetros son Comodidad (C), Dinero Discrecional (D), Dinero Esencial (E) y Vidas (L) (Amaro Calderón & Valverde Rebaza, 2009).

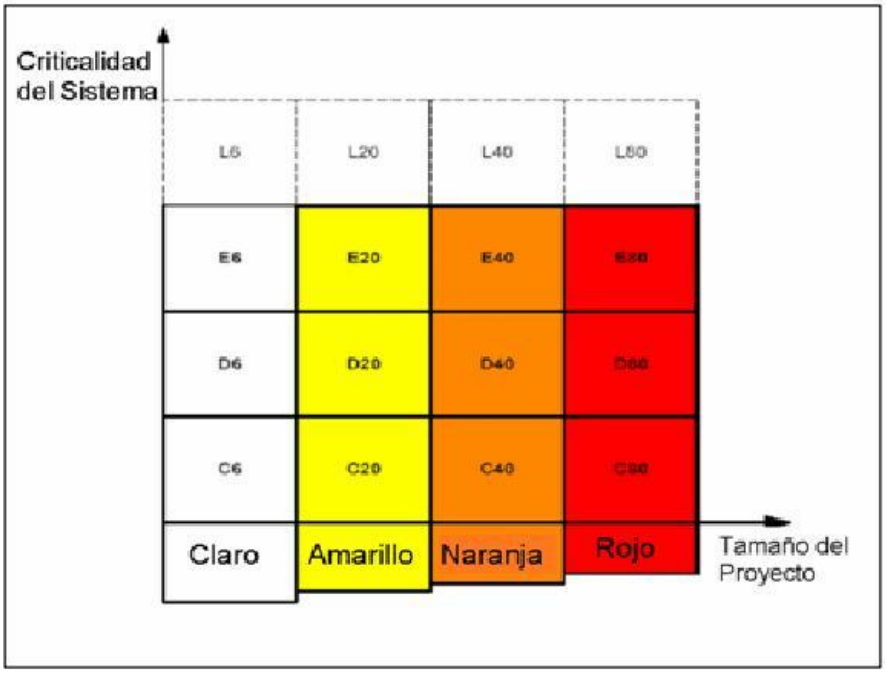


Figura 16. Familia del método Crystal.

Los métodos se llaman Crystal evocando las facetas de una gema: cada faceta es otra versión del proceso, y todas se sitúan en torno a un núcleo idéntico. Hay cuatro variantes de metodologías: Crystal Clear para equipos de ocho o menos integrantes; Amarillo, para ocho a 20; Naranja, para 20 a 50; Rojo, para 50 a 100. La más exhaustivamente documentada es Crystal Clear (CC), la misma que puede ser usada en proyectos pequeños de categoría D6. Como casi todos los otros métodos, CC consiste en valores, técnicas y procesos.

Los métodos Crystal no prescriben las prácticas de desarrollo, las herramientas o los productos que puede usarse, pudiendo combinarse con otros métodos como Scrum, XP y MSF. En un comentario, Cockburn, confiesa que cuando imaginó a Crystal Clear pensaba proporcionar un método ligero; comparado con XP, pero, Crystal Clear resultó muy pesado, si embargo, es más fácil de aprender e implementar (Amaro Calderón & Valverde Rebaza, 2009).

Crystal Clear está diseñado para proyectos pequeños (categoría D6), que comprenden hasta seis desarrolladores. Para evitar las limitaciones de comunicación de la estructura es conveniente que el equipo se ubique en una oficina común (Rivadeneira Molina, 2012).

Crystal Clear incluye descripciones de características del producto y casos de uso anotados, además de los artefactos: secuencias de entregas, modelos de objetos, manuales de usuario, casos de prueba y código.

Los roles de Crystal Clear son: patrocinador, diseñador de programas senior, diseñador de programas, usuario experto, experto en negocio, coordinador, verificador y escritor (Rivadeneira Molina, 2012).

Sobre las fases que abarca esta metodología, en la revisión bibliográfica realizada, no se encontró documentación donde fueran expuestas o explicadas las mismas. Solamente se pudo conocer que cubre pocas fases del ciclo de vida (Rivadeneira Molina, 2012).

Los artefactos que se utilizan para el modelado de requerimientos en esta metodología son:

- Declaración de la misión: documento de un párrafo o una página que describe el propósito.
- Lista de actores-objetivos: puede utilizarse alguna planilla de cálculo, una lista de dos columnas.
- Casos de uso anotados: para los requerimientos funcionales.
- Archivo de requerimientos: documentación que indica que debe construirse, quiénes los usarán, que valor tienen y qué restricciones afectan al diseño.

1.4.3.3.1 Valores o propiedades de Crystal Clear

Crystal Clear posee varios valores o propiedades, estas serán explicadas a continuación (Brito Acuña, 2009):

- **Entrega frecuente**: consiste en entregar software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del proyecto, pero puede ser diaria, semanal o mensual.
- **Comunicación osmótica:** todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un diseñador senior. Una reunión separada para que los concurrentes se concentren mejor es descrita como El Cono del Silencio.
- **Mejora reflexiva:** tomarse un pequeño tiempo (unas pocas horas por algunas semanas o una vez al mes) para pensar bien qué se está haciendo, cotejar notas, reflexionar y discutir.
- **Seguridad personal:** hablar cuando algo molesta: decirle amigablemente al administrador que la agenda no es realista, o a un colega que su código necesita mejorarse, o que sería conveniente que se bañase más seguido.

- **Foco**: saber lo que se está haciendo, tener la tranquilidad y el tiempo para hacerlo. Lo primero debe venir de la comunicación sobre dirección y prioridades, típicamente con el patrocinador ejecutivo. Lo segundo, de un ambiente en que las personas no se vea comprometidas a hacer otras cosas incompatibles.
- **Ambiente técnico con prueba automatizada**: administración de configuración e integración frecuente. Muchos equipos ágiles compilan e integran varias veces al día.

1.4.3.4 FDD

Fue desarrollada por Jeff De Luca y Peter Coad en 1997. Sucedió en 1997 que Jeff De Luca fue el gerente de proyecto, de un proyecto de desarrollo de software grande en Singapur. El problema dominio era tan compleja que Jeff se dio cuenta de que de que la tarea que no se pudo completar en tiempo, con los recursos disponibles, utilizando las estrategias tradicionales de desarrollo de software. Él por lo tanto, con la ayuda de Peter Coad y otros descubrió el modelado en la técnica de color y el concepto de función impulsado el desarrollo (Goyal, 2008). FDD es una metodología ágil diseñada para el desarrollo de software, basada en la calidad y el monitoreo constante del proyecto. Esta metodología se enfoca en iteraciones cortas, que permiten entregas tangibles del producto en un periodo corto de tiempo, como máximo dos semanas. A diferencia de otras metodologías ágiles FDD no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción (Universidad Unión Bolivariana, 2010).

1.4.3.4.1 Características de FDD

Algunas de las características de FDD son (Goyal, 2008):

- Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto.
- Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado.
- Propone tener etapas de cierre cada dos semanas: se obtienen resultados periódicos y tangibles.
- Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorear.
- Define claramente entregas tangibles y formas de evaluación del progreso del proyecto.
- No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción.

1.4.3.4.2 Actividades de FDD

Las actividades que se realizan en FDD son (Sehlhorst , 2006):

- **Desarrollar un modelo global:** al inicio del desarrollo se construye un modelo teniendo en cuenta la visión, el contexto y los requisitos que debe tener el sistema a construir. Este modelo se divide en áreas que se analizan detalladamente. Se construye un diagrama de clases por cada área.
- **Construir una lista:** se elabora una lista que resuma las funcionalidades que debe tener el sistema, cuya lista es evaluada por el cliente. Cada funcionalidad de la lista se divide en funcionalidades más pequeñas para un mejor entendimiento del sistema.
- **Planear:** se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asignan a los programadores jefes.
- **Diseñar:** se selecciona un conjunto de funcionalidades de la lista. Se procede a diseñar y construir las funcionalidades mediante un proceso iterativo, decidiendo qué funcionalidades se van a realizar en cada iteración. Este proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.
- **Construir:** se procede a la construcción total del proyecto.

1.4.3.4.3 Roles y responsabilidades de FDD

FDD define seis roles claves (Goyal, 2008):

- El gerente de proyecto: es la cabeza administrativa del proyecto responsable de chequear el progreso, la gestión de equipos, entre otros.
- El arquitecto jefe: es el responsable del diseño general del sistema. Él es el responsable del funcionamiento de las sesiones del taller de diseño, donde el equipo colabora en el diseño del sistema.
- El gerente de desarrollo: es responsable de liderar el desarrollo del día a día de las actividades. El gerente de desarrollo es responsable de resolver los conflictos cotidianos de recursos de los programadores.
- El jefe de los programadores: son experimentados desarrolladores que han pasado por todo el software un par de veces. Ellos participan en el análisis de requisitos de alto nivel y actividades de diseño del proyecto.

- Los propietarios de clase: son los desarrolladores que trabajan como miembros de los equipo de desarrollo pequeños bajo la dirección de un jefe programador para diseñar, codificar, probar y documentar las características requeridas por el nuevo sistema de software.
- Los expertos de dominio: son los usuarios, los patrocinadores, los analistas de negocio, o cualquier combinación de estos. Ellos son la base de conocimientos para que los desarrolladores entreguen un sistema correcto.
- Los expertos de dominio: necesitan buenas habilidades escritas y verbales de presentación. Su conocimiento y la participación son absolutamente crítico para el éxito del sistema que se está construyendo.

1.4.3.4.4 Ventajas de FDD

Algunas de las ventajas de FDD son (Steve R & Mac, 2001):

- Ayuda a resolverlos problemas que los procesos de desarrollo de software, tanto tradicionales como modernos no abordan.
- El equipo de desarrollo no malgasta el tiempo y dinero del cliente desarrollando soluciones innecesariamente generales y complejas, que en realidad no son un requisito del cliente.
- Cada componente del producto final es probado y satisface los requerimientos.
- Rápida respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos cortos de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Minimiza los costos frente a cambios.
- Simplicidad, al eliminar el trabajo innecesario.
- Atención a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo.
- Evita malentendidos de requerimientos entre el cliente y el equipo.

1.4.4 Metodologías web

Las metodologías tradicionales de ingeniería de software no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia. El tamaño, la complejidad y el número de aplicaciones web crecen en forma acelerada en la actualidad, por lo cual una

metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y reusabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios web, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr. En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada (Silva & Mercerat, 2010).

Construir la interfaz de una aplicación web es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación (Silva & Mercerat, 2010) .

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado. Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

1.4.4.1 OOHDM

OOHDM es una propuesta centrada en el diseño, que ofrece una serie de ideas adoptadas por muchas proposiciones y que ha dado muy buenos resultados (Stucky, 2009). Propone el desarrollo de aplicaciones hipermedia (Silva & Mercerat, 2010).

Hace una separación entre lo conceptual, la navegación y lo visual. Esta independencia hace que la gestión de aplicaciones sea mucho más fácil. Además, es la primera proposición que estudia a fondo los aspectos esenciales de la interfaz, no sólo en las aplicaciones multimedia, sino también en cualquiera de los sistemas desarrollados en la actualidad (Stucky, 2009).

OOHDM utiliza la meta-modelo de clases de UML, con pequeñas extensiones, para expresar el diseño conceptual.

1.4.4.1.1 Fases de OOHDM

- **Diseño Conceptual:** en esta fase se desarrolla el modelo conceptual de la aplicación, se representa con un diagrama de clases (Koch, 2009). También se construye un esquema conceptual representado por los objetos del dominio, las relaciones y

colaboraciones existentes establecidas entre ellos. En las aplicaciones hipermedia convencionales, cuyos componentes de hipermedia no son modificados durante la ejecución, se podría usar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De este modo, en los casos en que la información base pueda cambiar dinámicamente o se intenten ejecutar cálculos complejos, se necesitará enriquecer el comportamiento del modelo de objetos (Silva & Mercerat, 2010). Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real (Fraternali, 2009).

- **Análisis de requerimientos:** el paso inicial es obtener los requerimientos de las partes interesadas. Para esto, primero es necesario identificar los casos de uso, a través de los actores y las tareas que ellos deben realizar. Luego, los casos de uso son acopiados (o bosquejados) para cada tarea y tipo de actor, utilizando Diagramas de Interacción de Usuario (UID, User Interaction Diagram). Estos diagramas proveen una representación concisa utilizando una metáfora gráfica del flujo de información entre el usuario y la aplicación durante la ejecución de una tarea. Los UID son validados con el actor del caso de uso y rediseñado, si fuese necesario, a partir del retorno que haya brindado este último.
- **Diseño navegacional:** la primera generación de aplicaciones web fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia. En OOHDM, la navegación es considerada un paso crítico en el diseño de aplicaciones. En esta fase se desarrolla un modelo navegacional, el cual es construido como una vista sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual (Rossi, Pastor, Schwabe, & Olsina, 2012).

El diseño de navegacional es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son las tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La principal estructura primitiva del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.

Los contextos navegacionales juegan un rol similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada (Silva & Mercerat, 2010) .

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos, así como métodos cuando se navega en un particular contexto.

- **Diseño de interfaz abstracta:** una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas (Casteleyn, Florian, Dolog, & Matera, 2009).

Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario (Silva & Mercerat, 2010).

El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones web) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

El modelo de interfaz ADV (del inglés, Abstract Data View) especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos y la disposición de las propiedades de las ADV en la pantalla real son hechas en la fase de implementación.

- **Implementación:** en esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación (Silva & Mercerat, 2010). Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería

comportarse. A fin de implementar todo en un entorno web, el diseñador debe decidir, además, qué información debe ser almacenada.

1.4.4.1.2 Ventajas de OOHDM

Algunas de las ventajas que ofrece OOHDM son las siguientes (Martinez Mena, 2012):

- Permite desarrollar aplicaciones en la web.
- Propone un proceso predeterminado indicando las actividades a realizar y los productos que se deben obtener en cada fase del desarrollo.
- OOHDM hace uso de la orientación a objetos y de diagramas estandarizados a la hora de desarrollar interfaces.
- OOHDM asienta una notación de diagramas bastante completa, que permite incorporar en forma expresa los mecanismos oportunos de las aplicaciones hipermedias tales como nodos, vínculos, imágenes y estructuras de acceso.

1.4.4.1.3 Desventajas de OOHDM

OOHDM presenta algunas deficiencias (Stucky, 2009).

- No se puede lidiar con lo que se puede hacer dentro del sistema y en qué momento de la navegación o la interfaz se puede hacer, dejando estos aspectos como una tarea de implementación.
- No ofrecen ningún mecanismo para trabajar con múltiples actores.
- Solo puede ser utilizada para aplicaciones web o multimedia que sean simples, con una complejidad funcional mínima.

1.4.4.2 UWE

UWE nació a finales de los años 90 con la idea de encontrar una forma estándar para analizar y diseñar modelos de sistemas web. El objetivo por el cual nació esta metodología fue utilizar un lenguaje común, o por lo menos, definir un metamodelo basado en el mapeo a lo largo de las diferentes etapas. En esta época UML proponía convertirse en un estándar para el modelamiento de sistemas. Por este motivo, UWE se adhirió a UML y no a otra técnica de modelado. UWE se ha adaptado a las nuevas características de los sistemas web como: transacciones y personalizaciones, por otro lado ha evolucionado para incorporar técnicas de

ingeniería de software como el modelamiento orientado a aspectos y nuevos lenguajes de transformación para mejorar la calidad del diseño(Knapp, Koch, Moser, & Zhang, 2012).

La propuesta de Ingeniería Web basada en UML es una metodología detallada para el proceso de autoría de aplicaciones, con una definición exhaustiva del proceso de diseño que debe ser utilizado. Este proceso, iterativo e incremental, incluye flujos de trabajo y puntos de control (Brito Acuña, 2009).

La metodología UWE cubre todo el ciclo de vida de desarrollo de las aplicaciones web, propone un enfoque iterativo y orientado a objetos basado en el proceso de desarrollo de software unificado. El objetivo principal del enfoque UWE es el diseño sistemático seguido de una generación semi- automática de las aplicaciones web(Kraus & Koch, 2009).

1.4.4.2.1 Características de UWE

Los diagramas que se desarrollan en esta metodología se pueden adaptar como mecanismos de extensión basados en estereotipos que proporciona UML. Estos mecanismos de extensión son los que UWE utiliza para definir estereotipos que son los que finalmente se utilizarán en las vistas especiales para el modelado de aplicaciones web. De esta manera se obtiene una notación UML adecuada para un dominio específico, a lo que se conoce como perfil UML (Narváez, Baldeón, Hinojosa, & Martínez, 2010).

Algunos de los diagramas usados por UWE: diagramas de estado, de secuencia, de colaboración y diagramas de actividad.

Otras características relevantes del proceso y método de autoría de UWE son el uso del paradigma orientado a objetos, su orientación al usuario, la definición de un meta-modelo (modelo de referencia) que da soporte al método, y el grado de formalismo que alcanza debido al soporte que proporciona para la definición de restricciones sobre los modelos (Brito Acuña, 2009).

1.4.4.2.2 Actividades de modelado de UWE

Las actividades base de modelado de UWE son el análisis de requerimientos, la realización del modelo lógico-conceptual, el modelo de navegación y el modelo de presentación. A la realización de estos modelos se pueden sumar otros como: el modelo de interacción temporal y el modelo de visualización de escenarios web.

A continuación se explicarán estos modelos:

- **Modelo lógico-conceptual:** UWE apunta a construir un modelo conceptual de una aplicación web, procura no hacer caso, en la medida de lo posible, de cuestiones relacionadas con la navegación y los aspectos de interacción de la aplicación web. La construcción de este modelo lógico-conceptual se debe llevar a cabo de acuerdo con los casos de uso que se definen en el análisis de requerimientos.

  El modelo lógico-conceptual incluye los objetos implicados en las actividades típicas que los usuarios realizarán en la aplicación web.
- **Modelo de navegación:** consta de la construcción de dos modelos de navegación: el modelo del espacio de navegación y el modelo de la estructura de navegación. El primero especifica qué objetos serán visitados por el navegador a través de la aplicación. El segundo define cómo se relacionarán.
- **Modelo de presentación**: describe dónde y cómo los objetos de navegación y accesos primitivos serán presentados al usuario, es decir, una representación esquemática de los objetos visibles al usuario.
- **Modelo de interacción temporal:** presenta los objetos que participan en la interacción y la secuencia de los mensajes enviados entre ellos.
- **Modelo de visualización de escenarios web:** permiten detallar la parte dinámica del modelo de navegación, especificando los eventos que disparan las situaciones, definen condiciones y explícitamente incluyen las acciones que son realizadas. Junto con el modelo de interacción temporal, los escenarios web proveen la representación funcional dinámica del modelo de navegación.

1.4.4.2.3 Fases de UWE

UWE cubre todo el ciclo de vida de este tipo de aplicaciones centrando además su atención en aplicaciones personalizadas o adaptativas.

Las fases o etapas a utilizar son (Roque Espinoza, 2010):

- **Fase de requisitos:** en esta fase se trata de diferentes formas: las necesidades de información, las necesidades de navegación, las necesidades de adaptación y las de interfaz de usuario, así como algunos requisitos adicionales. Centra el trabajo en: el estudio de los casos de uso, la generación de los glosarios y el prototipado de la interfaz de usuario.

- **Fase de análisis y diseño:** UWE distingue entre: el diseño conceptual de modelo de usuario, de navegación, de presentación, de adaptación, de la arquitectura, en el diseño detallado de las clases y en la definición de los subsistemas e interfaces.
- **Fase de implementación:** UWE incluye implementación de la arquitectura, de la estructura del hiperespacio, del modelo de usuario, de la interfaz de usuario, de los mecanismos adaptativos y las tareas referentes a la integración de todas estas implementaciones.

1.4.4.2.4 Estructura del paquete

Todos los elementos de modelado UWE están contenidos dentro de un paquete de nivel superior que se añade a los tres paquetes de nivel superior UML. La estructura de los paquetes dentro del paquete UWE representada en la figura 17, es análoga a la estructura del paquete de nivel superior UML (en gris). El paquete fundación contiene todos los elementos básicos de modelado estático, el paquete elementos de comportamiento depende de este y contiene todos los elementos para el modelado del comportamiento y, finalmente, el paquete de modelos de gestión, que también depende del paquete de fundación, contiene todos los elementos para describir los modelos propios específicos de la metodología UWE. Estos paquetes UWE dependen de los paquetes de nivel superior UML correspondientes (Kraus & Koch, 2009).

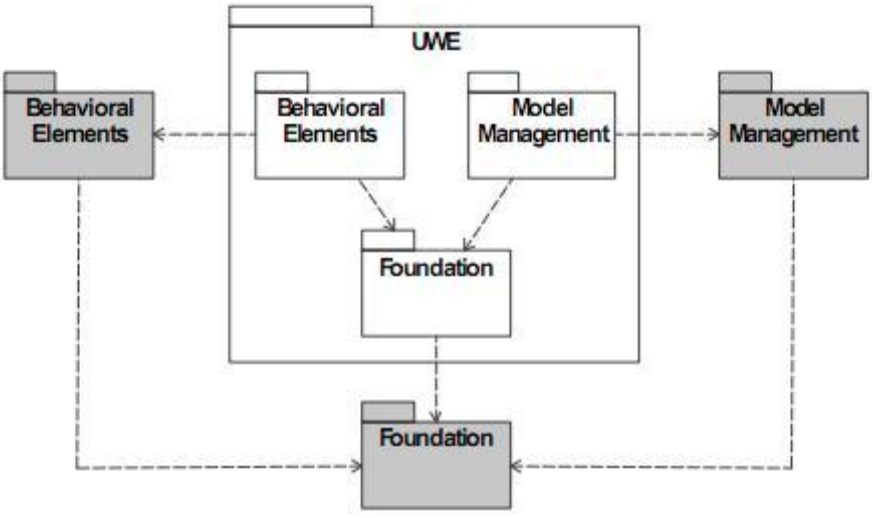


Figura 17. Paquetes de nivel superior UWE.

1.4.4.3 *W2000*

W2000 es una metodología que está organizada en paquetes que se van desarrollando en un proceso de 4 fases, que componen los modelos necesarios para el diseño de una aplicación web. W2000 toma como notación los diagramas propuestos por UML (Vilariño de Almeida, 2010). Utiliza UML tanto para la notación del dominio, para la notación de navegación como para la notación de representación (Villarroel Acevedo & Rioseco Reinoso, 2011).

1.4.4.3.1 *Fases de W2000*

Las fases de W2000 son las siguientes (Vilariño de Almeida, 2010):

- **Análisis de requisitos:** en esta fase se identifican los límites, objetivos y requerimientos funcionales de los procesos del negocio para el cual va dirigida la aplicación web. Los requerimientos funcionales se elaboran según la percepción de las diferentes clases de usuario.
- **Diseño de hipermedia:** en esta fase se realiza el diseño de la información, la navegación y la presentación. Comienza con una redacción de cada uno de estos aspectos y posteriormente se generan modelos con la estructura de cada uno de ellos. Estos modelos se van refinando para introducir todos los detalles que se deben establecer antes de implementar la aplicación.
- **Diseño de los servicios:** en esta fase se especifican las principales transacciones del negocio que va a soportar la aplicación a desarrollar. Dicha especificación está orientada a los usuarios.
- **Actividades de personalización:** esta fase se realiza en caso de ser necesaria. Se definen las características que se van a personalizar y posteriormente se introducen en los modelos generados en las fases anteriores.

1.4.4.4 *SOHDM*

SOHDM tiene similitudes con OOHDM entre otras metodologías, pero se propone una especificación de requisitos basada en escenarios (R, 2009).

SOHDM es una metodología para el desarrollo de aplicaciones multimedia que se divide en seis fases que hay que realizar de forma secuencial. Sin embargo, el proceso de desarrollo es un proceso cíclico, en el sentido de que al realizar una fase se puede regresar a alguna de las anteriores para refinarla y adaptarla mejor (Escalona Cuaresma, 2009).

Como su propio nombre indica, SOHDM está basado en los escenarios para elaborar las aplicaciones multimedia. Los escenarios se definen en la fase de análisis y se utilizan para el modelado de objetos (Koch, 2009).

A pesar de que SOHDM se asemeja bastante a OOHDM y EORM (del inglés, Enhanced Object Relationship Methodology), difiere de ellas en el sentido de que mientras que estas dos metodologías sólo trabajan en la fase de diseño, SOHDM engloba también la fase de análisis (Koch, 2009) .

1.4.4.4.1 Fases de SOHDM

A continuación se explican las fases de SOHDM (Escalona Cuaresma, 2009):

- **Análisis:** en esta fase se debe realizar un estudio de las necesidades de la aplicación, del entorno de trabajo y de los actores. La finalidad principal de esta fase es conseguir los escenarios que representen las actividades que se pueden llevar a cabo en el sistema.

Para ello, lo primero que se debe realizar es un diagrama de contexto, tal y como se propone en los diagramas de flujos de datos. En este diagrama de contexto es necesario detectar a las entidades externas que se comunican con el sistema, así como los eventos que provocan esa comunicación.

Una vez detectados estos eventos, se debe elaborar lo que se denomina lista de eventos. La lista de eventos que será una tabla con una estructura similar a la mostrada en la figura 18 que se muestra a continuación

Nombre de la entidad externa	Nombre del evento
Entidad externa 1	Evento 1 en el que participa la entidad
	...
	Evento n en el que participa la entidad
...	...
Entidad externa m	Evento 1 en el que participa la entidad
	...
	Evento p en el que participa la entidad

Figura 18. Estructura de la lista de eventos.

En esta figura se detallan todas las entidades en la columna de la izquierda y en la columna de la derecha todos los eventos en los que cada una participa. A modo de ejemplo, en la figura 19 se muestra una tabla donde la única entidad externa es el usuario, que se ve afectado por el evento de conectarse al sistema y el de petición de datos.

Nombre de la entidad externa	Nombre del evento
Usuario	Conexión al sistema
	Petición de datos

Figura 19. Ejemplo de la lista de eventos.

Partiendo de este ejemplo, por cada evento diferente se debe elaborar un escenario. Estos se van a representar mediante los denominados SAC (del inglés, Scenario Activity Chart). En estos escenarios se va a describir el proceso de trabajo que se va a seguir en el sistema cuando se produzca la situación que el escenario representa. En la figura 20 se muestra un ejemplo de un escenario, en el que se describe cómo se produce la conexión al sistema por parte del usuario.

Escenario para la conexión al sistema

1. El usuario se conecta al sistema
2. El sistema solicita la clave
3. El usuario introduce la clave
4. El sistema comprueba que la clave es correcta
5. El sistema muestra el menú principal

Figura 20. Ejemplo de escenario.

- **Modelado de objetos:** en esta fase los escenarios representados mediante SAC son usados para conseguir modelar los objetos del sistema. En la fase de modelado de objetos, los escenarios van a ser transformados en objetos según la propuesta de las tarjetas CRC (Clases, Responsabilidades y Colaboración). Esta propuesta tiene como objetivo presentar un formato sencillo e informal para conseguir un diccionario de datos para las clases del sistema. En ellas cada, clase tiene asociado una ficha (tarjetas CRC) en la que se almacena: su nombre, sus atributos, su superclase, sus subclases, sus componentes, las asociaciones en las que participa, las otras clases con las que colabora y los eventos detallados en los SAC de los que es responsable.

Las tarjetas CRC irán acompañadas de un diagrama de clases, CSD (del inglés, Class Structure Diagram), que representará gráficamente lo recogido en las tarjetas CRC. En principio, la nomenclatura seguida fue la de OMT pero ya ha asumido la de UML. Nuevamente, para nuestro ejemplo, este modelo coincidiría con el de la figura 21 (Escalona Cuaresma, 2009).

- **Diseño de vistas:** en la fase de diseño de vistas, los objetos serán reorganizados en unidades navegacionales. Una unidad navegacional representa una vista de los objetos del sistema. Las vistas van a estar muy relacionadas con el concepto de nodos de OOHDM. La vista es una agrupación de información que se presenta agrupada al usuario bajo un determinado criterio. En SOHDM, las vistas pueden ser:

    - Vistas base: son aquellas que toman todos los datos de una única clase.
    - Vistas de asociación: toma los datos de dos clases que se encuentran relacionadas mediante una asociación en el modelo de clases.
    - Vistas de colaboración: toma los datos de clases que se encuentran relacionadas mediante una relación de colaboración.
- **Diseño navegacional:** en esta fase se van a definir los enlaces o hiperenlaces que existen entre las diferentes vistas. Aquí las vistas definidas en la fase anterior se relacionan a través de estructuras de acceso.

El diseño de navegación utiliza escenarios para determinar la estructura de nodos de acceso. Se definen como mecanismo de tipo menú que permite a los usuarios acceder a otras partes de documentos hipermedia.

El primer paso a realizar es definir lo que se conoce como nodos de estructuras de acceso ANS (del inglés, Access Nodes Structures). Estos son nodos especiales como: diccionarios, menús, entre otros, que sirven como punto de partida para la navegación. Una vez definidos los ASN, estos y las vistas definidas en la fase anterior, se conectan mediante flechas que indican el sentido de la navegación. Se obtiene así un grafo en el que se representa cómo se navega desde un punto de información (vista o ASN) a otro. A este modelo se le denomina enlaces navegacionales.

A pesar de que el grafo que surge es bastante intuitivo, SOHDM propone una alternativa para representar los enlaces navegacionales. Por cada componente conexa de este grafo, se elabora una matriz, denominada matriz de enlaces navegacionales. En esta matriz se indica cómo los nodos de esa componente se relacionan entre sí.

- **Diseño de la implementación:** en esta fase se van a generar esquemas de páginas que van a representar los puntos de información definidos en la fase anterior dentro de un entorno determinado. Para cada esquema se debe indicar: su nombre, su título, las vistas que engloba, una breve descripción de su significado y una lista con los enlaces que tiene.

Tras definir estos puntos de información se hace un diseño de la interfaz de usuario.

SOHDM tiene prevista una nomenclatura normalizada para representar los posibles elementos que se pueden encontrar en una pantalla: botones, imágenes, listas, entre otras.

Una vez definida la interfaz de usuario y los esquemas, es necesario definir la base de datos.

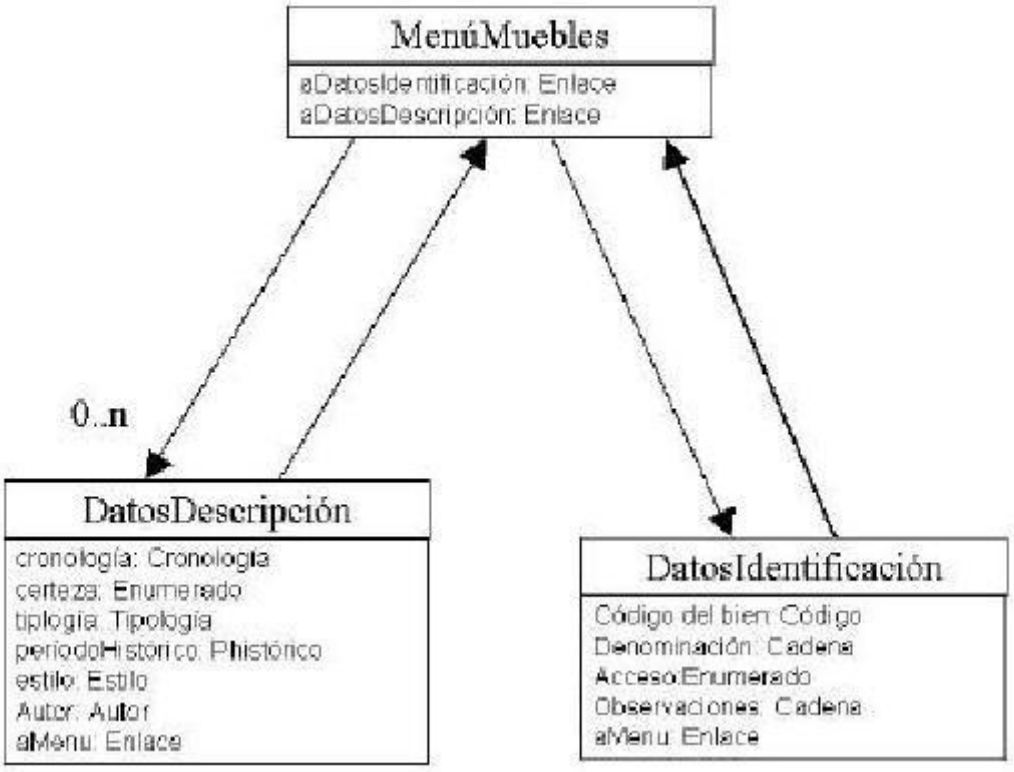


Figura 21. Ejemplo del modelo de diseño navegacional de SOHDM.

En principio, las aplicaciones hipermedia deben definirse en sistemas gestores orientados a objeto, pero, acercándose más a la realidad, permiten el uso de sistemas gestores de bases de datos relacionales. Para poder llevar una representación orientada a objetos a un modelo relacional, es necesario aplicar técnicas de conversión.

- **Construcción:** en la fase de construcción se debe implementar una aplicación hipermedia ejecutable en función de las pantallas y las páginas definidas en la fase anterior. Igualmente debe desarrollarse la base de datos física para soportar la aplicación.

1.5 Características y pasos de un estudio comparativo

Características de un estudio comparativo (Demeyer, 2011)**:**

A pesar de lo que comúnmente se cree por investigadores de la ciencia de la computación, en los estudios comparativos no se establece un ranking absoluto de los objetos de comparación. No obstante, ello no significa que el estudio no tenga un propósito específico, ulterior al objetivo planteado por el investigador.

En general, los estudios comparativos persiguen obtener las diferencias y similitudes encontradas entre los objetos de comparación. Pero debe tenerse cuidado en no favorecer a ningunos de ellos con una selección parcializada de los criterios cualitativos. Tales criterios, incluyendo aquellos que son cuantitativos, deben ser analizados con el objetivo de pesarlos con base en la aplicación de los objetos comparativos en un caso de estudio. En muchos casos, tales casos de estudio son tomados de artículos encontrados durante la revisión de la literatura.

La lista de criterios escogidos debe ser completa y reusable, es decir, que contiene todos los criterios que debiera, y que los tales pueden ser utilizados en otros estudios. La principal contribución en tal caso es la replicación, lo cual implica que un lector, si la selección fue correcta, podrá realizar por sí mismo ese estudio y llegar a los mismos resultados.

La revisión de la literatura en la que se apoya un estudio comparativo, comprende tres pasos fundamentales. Primero debe definirse la estrategia de la búsqueda bibliográfica: términos claves, herramientas, bases de datos especializadas, índices bibliográficos. Luego debe definirse el ámbito o alcance de la búsqueda, por lo que se debe contar con criterios de inclusión o exclusión de los resultados obtenidos. Por último, se debe especificar qué información debe ser obtenida. En este momento, la salida suele ser clasificaciones, taxonomías, mapas conceptuales, tablas y gráficos de frecuencia.

Al final, el estudio resulta en una comparación, la cual se realiza típicamente en forma de tablas.

Pasos a seguir para la realización de un estudio comparativo (Demeyer, 2011)**:**

Cuando se va a realizar un estudio comparativo debe seguirse una serie de pasos, siendo el primero de estos hacer una revisión de la literatura. Posteriormente se establecen los objetos de comparación. Después se establecen los criterios de comparación y, finalmente, se realiza el análisis comparativo.

1.6 Criterios para la comparación de metodologías de desarrollo de software

Después de una revisión bibliográfica sobre metodologías de desarrollo de software, de varios autores relacionados con el tema, se decidió proponer un conjunto de criterios para la comparación de estas metodologías de desarrollo de software.

Estos criterios fueron agrupados en tres grupos: criterios comunes, criterios para comparar las metodologías tradicionales y las ágiles y criterios para comparar las metodologías web. Para ello se tuvo en cuenta que algunos criterios están claramente especificados en todas las metodologías, no siendo así para el resto.

A continuación se presentan los criterios propuestos y se plantean los aspectos que serán analizados en cada caso. Una explicación más detallada de estos criterios se realizará en el capítulo 2.

Criterios comunes:

- **Ciclo de vida**: se requiere conocer qué fases del ciclo de vida de la aplicación web son consideradas por la metodología de desarrollo de software, para de esta forma conocer su alcance en el desarrollo del proyecto (Villarroel Acevedo & Rioseco Reinoso, 2011).
- **Énfasis en el uso de UML:** se requiere conocer qué diagramas de UML realizan las metodologías, teniendo en cuenta los nueve diagramas que se realizan en las versiones de UML anteriores a la 2.0, los cuales son:
    - Diagrama de casos de uso.
    - Diagrama de actividades.
    - Diagrama de objetos.
    - Diagrama de clases.
    - Diagrama de secuencia.
    - Diagrama de despliegue.
    - Diagrama de estados.
    - Diagrama de colaboración.
    - Diagrama de componentes.
- **Características del equipo de desarrollo:** se requiere conocer qué características debe poseer el equipo de desarrollo para poder utilizar una metodología determinada. En este criterio se tomarán como características a analizar: el tamaño del equipo de desarrollo y las relaciones interpersonales entre los miembros del equipo, donde se analizará si el equipo de desarrollo trabaja unido o disperso.
- **Características del proyecto:** se requiere conocer qué características debe poseer el proyecto para poder utilizar una metodología determinada. En el presente criterio se tomará como característica a analizar el tamaño del proyecto.

Criterios para comparar las metodologías tradicionales y las ágiles:

- **Facilidad de cambio:** se requiere conocer si estas metodologías permiten o no realizar cambios en los requerimientos una vez que hayan sido especificados (Bozo Parraguez & Crawford Labrin, 2009).

- **Interacción del cliente con el equipo de desarrollo:** se requiere conocer si el cliente tiene o no interacción con el equipo de desarrollo a medida que avanza el proyecto.

Criterios para comparar las metodologías web:

- **Tipo de modelado:** se requiere conocer qué tipo de método de modelado web utiliza cada metodología web (Vilariño de Almeida, 2010). Los tipos de métodos de modelado son:
    - Modelado de requerimientos.
    - Modelado de comportamiento contenido.
    - Modelado de comportamiento hipertexto.
    - Modelado de comportamiento presentación.
    - Modelado de estructura contenido.
    - Modelado de estructura hipertexto.
    - Modelado de estructura presentación.
    - Modelado de personalización.
- **Grado de detalle en las descripciones:** se requiere conocer la forma en que se define el tratamiento de requisitos (Escalona & KOCH, 2009). Este criterio será analizado teniendo en cuenta:
    - **Orientación al proceso:** se analizará si la metodología describe o no el proceso para realizar la definición, captura y validación de requisitos, y en caso de hacerlo, de qué forma lo hace.
    - **Orientación a la técnica:** se analizará si la metodología describe o no técnicas a aplicar durante el proceso.
    - **Orientación al resultado:** se analizará si la metodología describe o no el producto a obtener, y en caso de hacerlo, de qué forma lo hace.

1.7 Conclusiones

En este capítulo se expusieron las diferentes metodologías para el desarrollo de software, se realizó un estudio de las metodologías utilizadas para el desarrollo de aplicaciones web, enfocándose en aquellas que utilizan UML, y se establecieron los criterios para comparar las mismas.

Capítulo 2 Comparación de las metodologías para el desarrollo de aplicaciones web con UML

2.1 Introducción

En este capítulo se realiza la comparación de las diferentes metodologías para el desarrollo de aplicaciones web con UML, descritas en el capítulo anterior, atendiendo a los criterios de comparación propuestos.

2.2 Criterios comunes

En este epígrafe se abarcarán criterios que permitan la comparación de las metodologías tradicionales, ágiles y web, descritas anteriormente.

2.2.1 Ciclo de vida

El ciclo de vida del software describe el desarrollo de software desde la fase inicial hasta la fase final. Se refiere al conjunto de fases por las que pasa el sistema que se está desarrollando.
En la tabla 2 se exponen las diferentes metodologías y se realiza la comparación atendiendo al ciclo de vida que estas poseen, para así saber su alcance a lo largo del desarrollo del proyecto.
En esta tabla se puede apreciar que no todas las metodologías analizadas realizan completamente el ciclo de vida y que en casos particulares como en la metodología web: W2000 solo se llega a la fase de diseño.

Metodología		Ciclo de vida					
		Especificación	Análisis	Diseño	Codificación	Prueba	Mantenimiento
Tradicionales	RUP	x	x	x	x	x	x
	MSF	x	x	x	x	x	
	ICONIX		x	x	x		
Ágiles	XP		x	x	x	x	x
	SCRUM		x		x		
	FDD			x	x		
Web	OOHDM	x		x	x		
	UWE	x	x	x	x	x	x
	W2000	x	x	x			
	SOHDM	x	x	x	x		

Tabla 2. Comparación atendiendo al ciclo de vida.

2.2.2 Énfasis en el uso de UML

En relación al modelado del software, los seguidores del movimiento ágil y sus críticos focalizan la discusión entorno a la codificación. Por un lado se sostiene que el código es el único entregable que realmente importa, desplazando el rol del análisis y diseño en la creación de software. Los críticos del proceso precisan que el énfasis en el código puede conducir a la pérdida de la memoria corporativa o conocimiento organizacional, porque hay poca documentación y modelos para apoyar la creación y evolución de sistemas complejos (Bozo Parraguez & Crawford Labrin, 2009).

Esta es una de las razones que refleja la importancia que tiene que la metodologías utilizadas para el desarrollo del sistemas posean una amplia documentación. De esta forma, en situaciones futuras cuando se sufran cambios, existirán modelos que contengan diagramas, por los cuales los desarrolladores puedan giarse.

En la tabla que se muestra a continuación se exponen las diferentes metodologías tradicionales, ágiles y web tratadas anteriormente, y se hace la comparación atendiendo al énfasis en el uso de UML que estas poseen.

Metodología		Diagramas de UML								
		Caso de Uso	Actividad	Objetos	Secuencia	Colaboración	Clases	Despliegue	Componentes	Estado
Tradicionales	RUP	x	x	x	x	x	x	x	x	x
	MSF	x	x	x	x	x	x	x	x	x
	ICONIX	x	x	x	x	x	x	x	x	x
Ágiles	XP						x			
	SCRUM	x					x			x
	CRYSTAL CLEAR	x		x						
	FDD				x		x			
Web	OOHDM	x				x	x			
	UWE	x	x	x	x	x	x	x	x	x
	W2000	x	x	x	x	x	x	x	x	x
	SOHDM			x						

Tabla 3. Comparación atendiendo al uso de UML.

En la comparación mostrada en la tabla anterior se puede apreciar que las metodologías tradicionales y las metodologías web: UWE y W2000 son las que realizan todos los diagramas de UML. En el caso de las metodologías ágiles se puede ver que mayormente no realizan casi ningún diagrama de UML.

2.2.3 Características del equipo de desarrollo

Para la realización de un proyecto se crean equipos de desarrollo, los cuáles pueden ser grandes, medianos o pequeños, atendiendo a las características del proyecto o al personal destinado para el mismo. Estos equipos pueden trabajar en un solo lugar o de manera separada. Por tal motivo, es importante conocer qué características del equipo de desarrollo permite cada metodología para decidir cuál utilizar en un proyecto determinado.

Se consideran como equipos de tamaño pequeño a los que están conformados por diez o menos integrantes y de tamaño grande a los que poseen más de diez (Amaro Calderón & Valverde Rebaza, 2009).

En la tabla que se muestra a continuación se exponen las diferentes metodologías y se hace la comparación atendiendo a las características del equipo.

Metodología		Características del equipo	
		Tamaño	Relaciones interpersonales
Tradicionales	RUP	Grupos grandes	Trabajan separados
	MSF		
	ICONIX		
Ágiles	XP	Grupos pequeños	Trabajan unidos
	SCRUM		
	CRYSTAL CLEAR		
	FDD		
Web	OOHDM	Grupos pequeños	Trabajan unidos
	UWE		
	W2000		
	SOHDM		

Tabla 4. Comparación atendiendo a las características del equipo de desarrollo.

A partir de la comparación anterior se puede determinar que las metodologías ágiles y las web no son adecuadas cuando el equipo de desarrollo se encuentra físicamente disperso o cuando el equipo de desarrollo es grande.

Cuando los desarrolladores y clientes no pueden realizar la comunicación cara a cara planteada, aunque se podría establecer una comunicación cara a cara usando tecnologías tales como videoconferencia u otras, las alternativas en general no son tan eficaces como se esperaría (Bozo Parraguez & Crawford Labrin, 2009).

2.2.4 *Características del proyecto*

Los proyectos a desarrollar por los miembros del equipo de desarrollo pueden ser grandes, medianos o pequeños, atendiendo a la cantidad de requerimientos funcionales a implementar.
En el momento de seleccionar una metodología es importante conocer a qué tipo de proyecto es posible aplicarla, atendiendo a su tamaño.
En la tabla que se muestra a continuación se exponen las diferentes metodologías y se hace la comparación atendiendo a las características del proyecto.

Metodología		Tamaño del proyecto
Tradicionales	**RUP**	Grandes, medianos y pequeños.
	MSF	
	ICONIX	
Ágiles	**XP**	Pequeños y medianos.
	SCRUM	
	CRYSTAL CLEAR	
	FDD	
Web	**OOHDM**	Pequeños y medianos.
	UWE	
	W2000	
	SOHDM	

Tabla 5. Comparación atendiendo a las características del proyecto.

En la comparación mostrada en la tabla anterior se puede apreciar que las metodologías tradicionales se pueden utilizar en proyectos de cualquier tamaño, no siendo así para el resto de las metodologías.

2.3 *Criterios para comparar las metodologías tradicionales y las ágiles*

En este epígrafe se abarcan criterios que permiten la comparación de las metodologías tradicionales y ágiles, descritas en el capítulo anterior.

2.3.1 *Facilidad de cambio*

Hace algunos años se viene hablando sobre la crisis del software: proyectos que no cumplen sus presupuestos y exceden sus plazos de entrega. En muchos casos la causa es la falta de dinámica en determinar los requerimientos y controlar sus cambios a lo largo del proceso de desarrollo (Bozo Parraguez & Crawford Labrin, 2009).

Cualquier tipo de software debe cumplir con sus requerimientos para satisfacer a sus usuarios, pero debería poder sufrir cambios durante el desarrollo del mismo en caso que sea necesario.

Actualmente se puede observar un amplio espectro de diferentes formas de abordar un proceso de desarrollo de software, en cuyos extremos destacan, por un lado, los conservadores y formalistas, quienes se apegan a un proceso de desarrollo tradicional bien planificado, haciendo énfasis en los detalles para asegurar la calidad de los productos de software, aspirando a niveles de control y repetición. Y en el otro extremo el manifiesto ágil, que valora más las respuestas a los cambios, a través del uso de fuertes interacciones con usuarios, que el seguimiento estricto a una planificación. En estos casos, se argumenta que la rigidez de los planes o contratos muchas veces deriva en que cuando el sistema es entregado el problema que se pretendía resolver cambió o ya no existe (Bozo Parraguez & Crawford Labrin, 2009).

En la tabla que se muestra a continuación se exponen las diferentes metodologías y se hace la comparación atendiendo a la facilidad de cambio que estas poseen a la hora de modificar los requerimientos funcionales del sistema.

	Metodología						
	Tradicionales			**Ágiles**			
	RUP	***MSF***	***Iconix***	***XP***	***SCRUM***	***Crystal Clear***	***FDD***
Facilidad de Cambio	Resistencia ante los cambios.			Se adapta a los cambios.			

Tabla 6. Comparación atendiendo a la facilidad de cambio.

En la comparación mostrada en la tabla anterior se puede apreciar que las metodologías tradicionales poseen cierta resistencia a los cambios, mientras que las metodologías ágiles se adaptan a los mismos.

2.3.2 Interacción del cliente con el equipo de desarrollo

El equipo de desarrollo es el encargado de satisfacer las demandas del cliente a la hora de implementar el software. El cliente interactúa con el equipo en muchas ocasiones sólo mediante reuniones fijadas o cuando este tiene alguna duda de lo que quiere el mismo, pero en otros casos el cliente es parte del equipo de desarrollo.

En la tabla que se muestra a continuación se exponen las diferentes metodologías y se hace la comparación atendiendo a la interacción del cliente con el equipo de desarrollo, para saber cuál de ellas utilizar en un momento determinado.

Metodología		**Interacción del cliente con el equipo de desarrollo**
Tradicionales	**RUP**	El cliente interactúa con el equipo de desarrollo mediante reuniones.
	MSF	
	ICONIX	
Ágiles	**XP**	El cliente es parte del equipo de desarrollo.
	SCRUM	
	CRYSTAL CLEAR	
	FDD	

Tabla 7. Comparación atendiendo a la interacción del cliente con el equipo de desarrollo.

En la comparación mostrada en la tabla anterior se puede apreciar que en las metodologías tradicionales el cliente solo interactúa con el equipo de desarrollo mediante reuniones prefijadas con antelación, en cambio, en las metodologías ágiles el cliente es parte del equipo de desarrollo.

2.4 Criterios para comparar las metodologías web

En este epígrafe se abarcan criterios que permiten la comparación de las diferentes metodologías web explicadas en el capítulo anterior.

2.4.1 Tipo de modelado

Según la revisión de las diferentes propuestas metodológicas se observó que no todas proponen el modelado de los distintos elementos que caracterizan las aplicaciones web. En todo desarrollo web es fundamental realizar el modelado de la estructura del contenido y del hipertexto, este tipo de modelado se considera básico en todo diseño web. En la tabla siguiente se muestran los tipos de modelado que llevan a cabo las propuestas metodológicas web estudiadas (Vilariño de Almeida, 2010).

MÉTODOS DE MODELADO WEB	Modelado							
	Requerimientos	Comportamiento			Estructura			Personalización
		Contenido	Hipertexto	Presentación	Contenido	Hipertexto	Presentación	
OOHDM	X	X	X	X	X	X	X	X
SOHDM	X	X	X	X	X	X	X	X
UWE	X	X	X	X	X	X	X	X
W2000	X				X	X	X	X

Tabla 8. Comparación atendiendo al tipo de modelado.

En la comparación mostrada en la tabla anterior se puede apreciar que casi todas las metodologías web analizadas realizan el modelado de los distintos elementos que caracterizan las aplicaciones web.

2.4.2 Grado de detalle en las descripciones

Cuando se realiza el tratamiento de requisitos, las descripciones pueden realizarse de tres formas diferentes: con orientación al proceso, con orientación a la técnica y con orientación al producto (Escalona & KOCH, 2009).

En la tabla que se muestra a continuación se realiza la comparación de las diferentes metodologías web analizadas anteriormente, atendiendo al grado de detalle que las mismas poseen a la hora de realizar el tratamiento de requisitos.

	Orientación al proceso	Orientación a la técnica	Orientación al resultado
OOHDM	Describe el proceso sin detallarlo.	Describe claramente las técnicas y la forma de aplicarlas.	No comenta nada sobre el producto resultante.
SOHDM	No describe ningún proceso.		
UWE	Describe claramente los pasos a seguir.	Enumera las técnicas a aplicar.	Describe el contenido del producto sin entrar en detalle de su estructura.
W2000	Describe el proceso sin detallarlo.	Enumera las técnicas a aplicar.	No comenta nada sobre el producto resultante.

Tabla 9. Comparación atendiendo al grado de detalle en las descripciones.

En la comparación mostrada en la tabla anterior se puede apreciar que de las metodologías web analizadas de antemano, solo la metodología UWE describe claramente los pasos a seguir a la hora de realizar la orientación al proceso. En el caso de la orientación a la técnica a utilizar solo las metodologías OOHDM y SOHDM son las que describen claramente las técnicas y la forma en que las aplican, y en el caso de la orientación al resultado solo la metodología UWE describe el contenido del producto sin entrar en detalle de su estructura, y que las demás metodologías no comentan nada sobre el producto resultante.

2.5 Conclusiones

Se realizó la comparación de las diferentes metodologías descritas en el capítulo anterior para el desarrollo de las aplicaciones web con UML, atendiendo a los criterios comunes, criterios para comparar las metodologías tradicionales y las ágiles, y criterios para comparar las metodologías web. A modo de conclusión se puede decir que las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

CONCLUSIONES GENERALES

Con la realización del presente trabajo investigativo se arribó a las siguientes conclusiones:

- Toda metodología debe ser adaptada al contexto del proyecto (recursos humanos, tiempo de desarrollo, entre otros).
- Se siguieron los pasos para la realización de un estudio comparativo.
- Se logró realizar un estudio de los fundamentos teóricos y metodológicos que sustentan la comparación de metodologías que utilizan UML para el desarrollo de aplicaciones web. En el cual se establecieron criterios adecuados para la comparación: criterios comunes, criterios para comparar las metodologías tradicionales y las ágiles, y criterios para comparar las metodologías web.
- Se realizó una comparación de las metodologías que utilizan UML para el desarrollo de aplicaciones web. En esta comparación se determinó que las metodologías tradicionales analizadas y las metodologías web: UWE y W2000 realizan todos diagramas de UML propuestos en las versiones anteriores a la 2.0, y que las metodologías ágiles realizan muy pocos de estos diagramas.
- En esta comparación también se determinó que las metodologías ágiles se adaptan a los cambios a lo largo del proyecto y que el cliente es parte del equipo de desarrollo, en contradicción con las metodologías tradicionales que poseen cierta resistencia a los cambios y que el cliente sólo interactúa con el equipo mediante reuniones prefijadas.

.

RECOMENDACIONES

- Hacer un estudio comparativo de las metodologías de desarrollo para aplicaciones web con UML alrededor del tema de la calidad, en el cual se tengan en cuenta los siguientes aspectos: usabilidad, portabilidad, simplicidad, eficiencia, entre otros.
- Basado en el estudio comparativo, realizar la selección de la metodología más adecuada para el desarrollo de aplicaciones web con UML, con vistas a su potencial utilización en una herramienta CASE.

REFERENCIAS BIBLIOGRÁFICAS

Amaro Calderón, S. D., & Valverde Rebaza, J. C. (2009). *Metodologías Ágiles.* Recuperado el 6 de marzo de 2014, de Universidad Nacional de Trujillo: http://www.sisman.utm.edu.ec/libros/FACULTAD%20DE%20CIENCIAS%20ZOOT%C3%89CNICAS/CARRERA%20DE%20INGENIER%C3%8DA%20EN%20INFORMATICA%20AGROPECUARIA/07/INGENIERIA%20DEL%20SOFTWARE%20I/METODOLOGIAS%20AGILES.pdf

Anízio Maia, J. (2013). *JUGManaus.* Recuperado el 20 de mayo de 2014, de Construindo Softwares com Qualidade e Rapidez Usando ICONIX: https://cde4b5d3-a-62cb3a1a-s-sites.googlegroups.com/site/fredslz/artigo_ConstruindoSoftwarescomQualid.pdf?attachauth=ANoY7coGWIrYA8RufDJ15IjuVkPnzMqNT4oCBPth_N2OzpN2_D9JjSD3c3KqX9jnMTYi7s-nMNsAavjVWwPzNni3weu7Dmn_Qg_J7ZPvq066xTs47_RNvxrJ3v1fFMJZo_ycNEGbq

Bautista, J. (2010). *UNIVERSIDAD UNION BOLIVARIANA.* Recuperado el 6 de marzo de 2014, de PROGRAMACIÓN EXTREMA (XP): http://ingenieriadesoftware.mex.tl/images/18149/PROGRAMACI%C3%93N%20EXTREMA.pdf

Body of Knowledge. (27 de abril de 2014). *Scrum Manager*. Recuperado el 10 de marzo de 2014, de http://www.scrummanager.net/bok/index.php?title=El_manifiesto_%C3%A1gil

Bona, C. (2010). *AVALIAÇÃO DE PROCESSOS DE SOFTWARE: UM ESTUDO DE CASO EM XP E ICONIX.* Recuperado el 27 de mayo de 2014, de Universidade Federal de Santa Catarina: ftp://atenas.cpd.ufv.br/dpi/mestrado/Gerais/TeseIconix.pdf

Bozo Parraguez, J., & Crawford Labrin, B. (2009). *Métodos Ágiles como Alternativa al Proceso de Desarrollo Web.* Recuperado el 6 de marzo de 2014, de Congreso de Ciencias de la Computación: http://sedici.unlp.edu.ar/bitstream/handle/10915/22775/Documento_completo.pdf?sequence=1

Brito Acuña, K. (2009). *BIBLIOTECA VIRTUAL de Derecho, Economía y Ciencias Sociales*. Recuperado el 6 de marzo de 2014, de SELECCIÓN DE METODOLOGÍAS DE DESARROLLO PARA APLICACIONES WEB EN LA FACULTAD DE INFORMÁTICA DE LA UNIVERSIDAD DE CIENFUEGOS,Grupo de investigación eumednet (SEJ-309) de la Universidad de Málaga: http://www.eumed.net/libros-

gratis/2009c/584/Descripcion%20de%20las%20metodologias%20existentes%20para%20el%20desarrollo%20de%20software.htm

Casteleyn, S., Florian, D., Dolog, P., & Matera, M. (2009). *Engineering Web Applications.* New York: Springer-Verlag Berlin Heidelberg .

Cockburn, A. (2013). *Crystal Clear. A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects.* Recuperado el 29 de abril de 2014, de http://users.dcc.uchile.cl/~nbaloian/cc1001-03/ejercicios/crystalclearV5d.pdf

Demeyer, S. (2011). *Antwerp Systems and software Modelling.* Obtenido de University of Antwerp: http://www.win.ua.ac.be

Escalona Cuaresma, M. J. (octubre de 2009). *Metodologías para el desarrollo de sistemas de información global: análisis comparativo y propuesta.* Recuperado el 11 de abril de 2014, de http://www.lsi.us.es/docs/informes/EstadoActual.pdf

Escalona, M. J., & KOCH, N. (2009). *Journal of Web Engineering.* Munich, Alemania: Rinton Press.

Fraternali, P. (2009). Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *revista acm computing surveys*.

Gomaa, H. (2011). *SOFTWARE MODELING AND DESIGN. UML, Use Cases, Patterns, and Software Architectures.* New York: Cambridge University Press.

Goyal, S. (2008). Agile Techniques for Project Management and Software Engineering. *Major Seminar On Feature Driven Development.* Pittsburgh (Pensilvania): Pearson Education.

Gruzado Nuño, I., García de Egido, J., & Portugal Alonso, J. (noviembre de 2012). *slideshare.* Recuperado el 20 de marzo de 2014, de Metodologías Orientadas a Objetos, Trabajo Teórico de Programación Avanzada: httpwww.slideshare.netmartin8730metodologias1.html

Hans, E. E., & Magnus, P. (2000). *Business Modeling with UML: Business Patterns at Work .* Canadá: Library of Congress cataloging-in-Publication Data.

Instituto Tecnológico de Veracruz. (2010). *Metodologías para el desarrollo de aplicaciones Web.* Recuperado el 26 de febrero de 2014, de http://www.prograweb.com.mx/pweb/0205metodologiasWeb.html

Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *El Proceso Unificado de Desarrollo de Software.* Madrid: PEARSON EDUCACIÓN.

Knapp, A., Koch, N., Moser, F., & Zhang, G. (2012). *ArgoUWE: A CASE Tool for Web Applications.* Germany: EMSISE'03.

Koch, N. (2009). *A Comparative Study of Methods for Hypermedia Development.* Recuperado el 30 de abril de 2014, de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.9988&rep=rep1&type=pdf

KOCH, N., & KRAUS, A. (2002). The Expressive Power of UML-based Web Engineering1. *Ludwig-Maximilians-Universität München*, 4-15.

Kraus, A., & Koch, N. (2009). *A Metamodel for UWE.* Recuperado el 29 de abril de 2014, de Ludwig-Maximilians-Universität München: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.504&rep=rep1&type=pdf

Laboratorio Nacional de Calidad del Software de INTECO. (marzo de 2009). *Ingeniería del software: metodologías y ciclos de vida.* España: Delta Publicaciones. Recuperado el 27 de enero de 2014, de http://wiki.monagas.udo.edu.ve/index.php/Metodolog%C3%ADas_para_el_desarrollo_de_software

Letelier, P., & Penadés, C. (2010). Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Técnica Administrativa, Buenos Aires*. Obtenido de Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).

Martinez Mena, M. G. (julio de 2012). *Dirección y construcción de un sistema de geoposición en visual.net ambientada como Aplicación Web para el manejo de información general de pólizas de seguros que integre el manejo de una Base de Datos textual en MySQL y una Base de Datos Gráfica(AutoCA .* Recuperado el 8 de abril de 2014, de Universidad Politécnica Salesiada Sede Quito - Campus sur: http://dspace.ups.edu.ec/bitstream/123456789/4752/1/UPS-ST000904.pdf

Medín, C. F. (2009). *Investigación exploratoria de metodologías de desarrollo ágiles.* Recuperado el 2014 de marzo de 20, de Universidad Abierta Interamericana, Rosario, Santa Fe, Argentina.: http://www.cedis-it.com.ar/images/Documentos_base/est2008_metodologiasagiles.pdf

Microsoft Developer Network. (2013). *Microsoft Solutions Framework (MSF) Overview*. Recuperado el 28 de mayo de 2014, de http://msdn.microsoft.com/es-es/library/jj161047.aspx

Mountain Goat Software. (2012). *mountaingoatsoftware*. Recuperado el 29 de mayo de 2014, de Scrum Product Backlog: http://www.mountaingoatsoftware.com/agile/scrum/product-backlog

Narváez, A., Baldeón, P., Hinojosa, C., & Martínez, D. (2010). *Experiencia de desarrollo de una aplicación web utilizando la metodología UWE y el lenguaje QVT en la transformación de modelos.* Recuperado el 11 de abril de 2014, de Escuela Politécnica del Ejército, Ecuador: http://repositorio.espe.edu.ec/bitstream/21000/4596/2/T-ESPE-032708-A.pdf

OMG.org. (2012). *Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure.* Published by ISO.

Peña Pérez, S., Jiménez Ruiz, B., Valverde Pérez, Y., Aramayo Cuellar, A., & Salazar Zorrilla, I. (12 de octubre de 2012). *METODOLOGÍA DE DESARROLLO DE SOFTWARE -MSF-.* Recuperado el 6 de marzo de 2014, de UNIVERSIDAD AUTÓNOMA GABRIEL RENE MORENO, Santa Cruz – Bolivia: http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&cad=rja&ved=0CE4QFjAI&url=http%3A%2F%2Fbeymarjimenez.files.wordpress.com%2F2012%2F10%2Fmsf-documento.docx&ei=Py8WU5mpAcKgkAfbzIG4Ag&usg=AFQjCNHUg3fjljID_9EAdbfgEt-j1vlx8g

Pollice, G., Augustine, L., Lowe, C., & Madh, J. (2004). *Software Development for Small Teams: A RUP-Centric Approach.* Wesley: Pearson Education, Inc.

Proyectalis. (2014). *Proyectalis*. Recuperado el 6 de marzo de 2014, de Curso “Scrum: metodología ágil de gestión de proyectos”: http://www.proyectalis.com/servicios/formacion/scrum/

proyectosagiles.org. (2013). *proyectosagiles.org*. Recuperado el 29 de mayo de 2014, de Lista de objetivos / requisitos priorizada (Product Backlog): http://www.proyectosagiles.org/lista-requisitos-priorizada-product-backlog

proyectosagiles.org. (2013). *proyectosagiles.org*. Recuperado el 29 de mayo de 2014, de Lista de tareas de la iteración (Sprint Backlog): http://www.proyectosagiles.org/lista-tareas-iteracion-sprint-backlog

proyectosagiles.org. (2013). *proyectosagiles.org*. Recuperado el 29 de mayo de 2014, de Gráficos de trabajo pendiente (Burndown charts): http://www.proyectosagiles.org/graficos-trabajo-pendiente-burndown-charts

Proyectosagiles.org. (2013). *Proyectosagiles.org*. Recuperado el 6 de marzo de 2014, de Qué es SCRUM: http://www.proyectosagiles.org/que-es-scrum

R, J. (2009). *Requirements Engineering In Current Web Engineering Methodologies.* Recuperado el 29 de abril de 2014, de Department of Computer Science and Software: http://ijcta.com/documents/volumes/vol2issue3/ijcta2011020318.pdf

Rivadeneira Molina, S. G. (mayo de 2012). *METODOLOGÍAS ÁGILES ENFOCADAS AL MODELADO DE REQUERIMIENTOS.* Recuperado el 19 de marzo de 2014, de Universidad Nacional de la Patagonia Austral, Río Turbio, Provincia de Santa Cruz, República Argentina: http://proyuacouart.wikispaces.com/file/view/ICT+Requerimientos+%C3%A1giles+vcf.pdf

Roque Espinoza, J. T. (2010). *Desarrollo de un sistema de información web para mejorar el proceso de evaluación y presentación de perfiles de proyectos de investigación científica y tecnológica a nivel nacional en FINCyT-PCM.* Recuperado el 8 de abril de 2014, de UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS: http://www.concytec.gob.pe/portalsinacyt/images/stories/corcytecs/lima/roque_espinoza_javier_teodocio_2010.pdf

Rossi, G., Pastor, O., Schwabe, D., & Olsina, L. (2012). *Web Engineering: Modelling and Implementing Web Applications.* Valencia: Springer-Verlag London Limited .

Rumbaugh, J., Jacobson, I., & Booch, G. (1998). *El Lenguaje Unificado de Modelado. Manual de Referencia.* Cupertino, California: Addison Wesley.

Schmuller, J. (2011). *Sams Teach Yourself UML in 24 Hours.* Distrito Federal: Sams Publishing.

Sehlhorst , S. (27 de marzo de 2006). *TynerBlain*. Recuperado el 28 de mayo de 2014, de Foundation Series: Feature Driven Development (FDD) Explained: http://tynerblain.com/blog/2006/03/27/foundation-series-feature-driven-development-fdd-explained/

Silva, D. A., & Mercerat, B. (2010). Construyendo aplicaciones Web con una metodología orientada a objetos. *REVISTA COLOMBIANA DE COMPUTACION*, 79 - 95. Recuperado el 26 de febrero de 2014, de Construyendo aplicaciones Web con una metodología orientada a objetos.

Standing, C. (2010). *Methodologies for developing web applications.* Joondalup, Australia: ELSEVIER.

Steve R, P., & Mac, F. (2001). *A Practical Guide to Feature-Driven Development.* Pearson Education.

Stucky, W. (2009). *The European Online Magazine for the IT Professiona.* UPGRADE grows and matures.

Universidad Unión Bolivariana. (2010). *Ingeniería de Software*. Recuperado el 10 de marzo de 2014, de Metodología Crystal en métodos ágiles: http://ingenieriadesoftware.mex.tl/59189_Metodologia-Crystal.html

Uñoja, R. H. (23 de abril de 2012). *Ingenería de Software.* Recuperado el 6 de marzo de 2014, de METODOLOGIAS DE DESARROLLO DE SOFTWARE TRADICIONALES VS AGILES: http://tallerinf281.wikispaces.com/file/view/Comparacion%25Metodologias%25RaquelCP2.pdf

Vilariño de Almeida, J. C. (19 de mayo de 2010). *Modelo para la selección de la metodología de desarrollo web de una aplicación según sus características funcionales.* Recuperado el 22 de abril de 2014, de Universidad Católica Andrés Bello- Caracas: http://biblioteca2.ucab.edu.ve/anexos/biblioteca/marc/texto/AAS2255.pdf

Villarroel Acevedo, R., & Rioseco Reinoso, C. (2011). Una comparación de metodologías para el modelado de. *Revista Cubana de Ciencias Informáticas*.

WikiUDO. (11 de julio de 2012). *Metodologías para el desarrollo de software*. Recuperado el 27 de enero de 2014, de Universidad de Oriente (Monagas -Venezuela): http://wiki.monagas.udo.edu.ve/index.php/Metodolog%C3%ADas_para_el_desarrollo_de_software

Zamudio, L., Ramírez, L., Alvarez, M., & Rodríguez, A. (octubre de 2009). *UML (Unified Modeling Language), Universidad Yacambú*. Recuperado el 26 de febrero de 2014, de http://www.oocities.org/es/avrrinf/tabd/Foro/Foro_UML.htmhttp://www.oocities.org/es/avrrinf/tabd/Foro/Foro_UML.htm

Zamuriano, R. (9 de febrero de 2010). *Desarrollo de aplicaciones Web en MicroSoft C# modeladas en UML*. Recuperado el 20 de marzo de 2014, de http://es.scribd.com/doc/26613313/14/MODELO-DE-OBJETO-DEL-NEGOCIO