PERSONALIZACIÓN DE RUP PARA PROYECTOS ACADÉMICOS DE DESARROLLO DE SOFTWARE

LUIS FELIPE TABARES BEDOYA
ltabares@eafit.edu.co

Proyecto de grado para obtener el título de Ingeniero de Sistemas

Asesor
Jorge Hernán Abad Londoño
Ingeniero Civil
Especialista en Desarrollo de Software

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
COORDINACIÓN DE PROYECTOS DE GRADO
MEDELLÍN
2011

A mis padres, Rodrigo y Gloria María, mi hermana, Catalina, y por supuesto, mi mejor amigo, José Escobar; por su apoyo y temple, durante esta larga etapa de mi formación profesional

Quiero dar gracias, mis gracias más sinceras, a todas las personas que contribuyeron sustancialmente a iniciar y concluir este proyecto de grado.

A Jorge Abad, mi asesor en el proyecto, por permitirme participar en su tesis de maestría, y darme la oportunidad de realizar mi proyecto de grado en tópicos que captan mi completo interés y que pretenden estrechar la brecha entre la academia y la industria; realmente es una suerte que muy pocos pueden tener en su proyecto de grado.

A la Universidad EAFIT, por darme una formación generosa como persona y profesional; por sus instalaciones, facilidades, y por supuesto, el plantel humano que la conforman.

CONTENIDO

pág.

LISTA DE FIGURAS

pág.

RESUMEN

Este proyecto de grado centra su objetivo en implementar la metodología EAFITUP para orientar la ejecución de proyectos académicos de desarrollo de software. Para el diseño de la metodología, se ha realizado una adaptación (personalización) de la metodología RUP (Proceso Unificado de Rational, por sus siglas en español), y se han incluido elementos de alternativas como OpenUP (Open Unified Process), UPEDU (Unified Process for EDUcation), Scrum, y XP (eXtreme Programming); así como también se han considerado buenas prácticas de CMMI (Capability Maturity Model Integration).

Esta metodología comprende adaptaciones para proyectos académicos de diferente magnitud, y establece una guía para cada una de las personas implicadas en el ciclo de vida del proceso de desarrollo de software. Está orientado principalmente a miembros del equipo de desarrollo que se dedican a las actividades del ciclo de vida: modelado del negocio, requisitos, análisis y diseño, implementación, pruebas, implantación, gestión de cambios, y gestión del proyecto. Es útil para analistas y usuarios finales (que especifican los requisitos de la estructura y comportamiento del sistema), para arquitectos (que diseñan los elementos que satisfacen esos requisitos), para desarrolladores (que convierten esos diseños en código ejecutable), para testers (que verifican y validan la estructura y comportamiento del sistema) y para los líderes del proyecto (directores de proyecto, docentes, líder de investigación, etc.).

Para la creación, configuración y publicación de la metodología EAFITUP, se ha utilizado el marco de trabajo EPF (Eclipse Process Framework), el cual utiliza el estándar para la representación de modelos de procesos de ingeniería de software e ingeniería de sistemas SPEM (Software and Systems Process Engineering Metamodel), versión 2.0, definido por el OMG (Object Management Group, Inc.).

Palabras clave: EAFITUP, Adaptación, Metodología, RUP, OpenUP, CMMI, Proceso, Desarrollo, Software, EPF, SPEM.

INTRODUCCIÓN

EAFITUP es un proyecto que propone una metodología para la ejecución de proyectos académicos de desarrollo de software; y está dirigido a los programas de pregrado y posgrado de la Universidad EAFIT. En esta metodología se establecen procesos que implementan adaptaciones de RUP (Proceso Unificado de Rational, por sus siglas en español), y utilizan algunos elementos de OpenUP (Open Unified Process), UPEDU (Unified Process for EDUcation), Scrum, y XP (eXtreme Programming); permitiendo hacer frente a proyectos académicos de diferente alcance (diferente magnitud).

En la actualidad, se presenta con normalidad que un proyecto académico de desarrollo de software que se ejecute en la Universidad EAFIT, no esté guiado por una metodología o proceso de desarrollo de software que incorpore buenas prácticas, y permita obtener un proyecto exitoso, es decir, un proyecto que se entrega bajo el presupuesto asignado, a tiempo y con la calidad especificada.

El ámbito académico, a diferencia del ámbito empresarial, ejerce menos control en el cumplimiento del cronograma y el presupuesto estimado, prestando solo interés en el entregable final (entregable software final); dejando a los proyectos sin exigencia en la adherencia de buenas prácticas, elaboración de documentación, utilización eficiente de recursos; y por consiguiente alta probabilidad de fracaso.

Un alto porcentaje de los proyectos de desarrollo de software fracasan por la carencia de un adecuado proceso de desarrollo, o por una utilización inapropiada. Muchos estudiantes se equivocan al pensar que un proyecto comienza con la programación, en un lenguaje determinado, de las necesidades que se expresan inicialmente; y ésta es una de las causantes más importantes de fracaso de un proyecto. Sin importar el tamaño del proyecto, es necesario que siempre exista una metodología o proceso de desarrollo de software.

Este proyecto pretende proporcionar a la Universidad EAFIT, la definición de un marco de trabajo (framework) que oriente a diferentes actores (estudiantes, docentes, grupos de investigación) en la ejecución de proyectos académicos de desarrollo de software, y permita la generación de aplicaciones de software que se ajusten al tiempo planeado, al presupuesto asignado y cumplan con estándares de calidad. La utilización de la metodología que se define en este marco de trabajo, facilitará a los estudiantes la adopción de procesos de desarrollo de software en la ejecución de sus proyectos, y de esta forma se crean oportunidades para estrechar la brecha existente entre las competencias de los profesionales que egresan y las competencias que demanda la industria.

El alcance de este proyecto hace parte del trabajo de tesis de Maestría en Ingeniería, del Ingeniero Jorge Abad Londoño; el cual tiene como objetivo la construcción de un marco de trabajo (framework) que guie a los estudiantes en la ejecución de sus proyectos académicos de desarrollo de software. Las metodologías propuestas en el marco de trabajo deben aplicarse en proyectos de desarrollo de software con fines académicos (prácticas, trabajos, proyectos y tesis), y se excluyen todos aquellos desarrollos de software de misión crítica.

La definición, configuración y publicación de los lineamientos metodológicos para la ejecución de proyectos de desarrollo de software, en el ámbito académico de la Universidad EAFIT, se ha realizado a través del marco de trabajo EPF (Eclipse Process Framework); que incluye el editor EPFC (EPF Composer), el cual se basa en el estándar para la representación de modelos de procesos de ingeniería de software e ingeniería de sistemas SPEM (Software and Systems Process Engineering Metamodel), versión 2.0, definido por el OMG (Object Management Group, Inc.). Con EPFC se pueden crear implementaciones en formato SPEM 2.0 de cualquier método, proceso o metodología de ingeniería de software.

El contexto del presente proyecto está enmarcado por metodologías tradicionales y metodologías ágiles para el desarrollo de software, ampliamente utilizadas y probadas por organizaciones que conforman la industria mundial. Por tanto, buena parte de este proyecto está sustentado en revisiones bibliográficas de diversas fuentes, y principalmente en la adaptación de la metodología RUP (Proceso Unificado de Rational, por sus siglas en español).

1. DEFINICIÓN DEL PROBLEMA

Actualmente la Universidad EAFIT, para los estudiantes de los programas de pregrado y posgrado, no ha definido un marco de referencia donde se establezca un proceso de desarrollo de software, que de uniformidad, formalidad, flexibilidad y orientación en la ejecución de los proyectos de desarrollo de software, en el ámbito académico.

Ante la ausencia de la definición de un proceso uniforme, formal y flexible, se presentan los siguientes hechos:

- La evaluación de los diferentes proyectos académicos se centra en el resultado final (el producto software), sin considerar el proceso elegido para el desarrollo y sus entregables, los cuales son de igual importancia al código ejecutable.

- Omisión en la adopción de buenas prácticas, que la industria ha identificado, para implementar y llevar a buen término proyectos de desarrollo de software.

- No existe un esquema de trabajo coherente, adoptado por los estudiantes, que defina el ciclo de vida y las mejores prácticas, para llevar un proyecto de desarrollo de software al éxito.

- Ausencia de uniformidad en los criterios para gestionar y ejecutar proyectos de desarrollo de software, en las diferentes asignaturas de la Universidad.

- Carencia de una metodología de desarrollo para los proyectos de software, que se ejecutan en los cursos relacionados con Ingeniería de Software.

- Falta de tipificación de los proyectos de desarrollo de software, que permita adaptar una metodología a los diferentes tipos de proyectos.

- Brecha entre las competencias del personal que demanda la industria de desarrollo de software, y los profesionales que egresan de la Universidad EAFIT; debido a la adopción discrecional (para no generalizar con el término ausencia) de una metodología de desarrollo (con sus buenas prácticas) para el desarrollo de software. Esta brecha, hace necesario que las compañías deban invertir en largos periodos de capacitación y entrenamiento, en conocimientos y habilidades que debieron haber sido adquiridas y puestas en práctica en el ciclo de formación profesional.

2. JUSTIFICACIÓN

De acuerdo al Dr. Frederick Brooks (ganador del ACM Turing Award, en 1999), el desarrollo de software es la tarea más compleja que puede iniciar un ser humano. Esta tarea es una actividad que frecuentemente se repite en la mayoría de las asignaturas que conforman el programa de Ingeniería de Sistemas, de la Universidad EAFIT; así como también es el campo de acción, donde la mayoría de los nuevos profesionales obtiene su primera experiencia laboral.

Una metodología formal, flexible y basada en buenas prácticas para el desarrollo de software, es una guía que define las actividades que se deben realizar para alcanzar los resultados esperados. Con la construcción de esta metodología se pretende implementar un marco de trabajo de desarrollo de software para proyectos académicos de diferente alcance, cuyo propósito es orientar a los estudiantes hacia el desarrollo de productos de software de alta calidad, que se ajusten a los requisitos iniciales, y estén dentro del costo y tiempo determinados.

En el contexto académico, al adoptar una metodología uniforme para el desarrollo de software, se identifican las siguientes ventajas:

Docente

- Uniformidad en el ciclo de vida de los proyectos, las responsabilidades, los entregables, y las buenas prácticas que se deben considerar.
- Recepción de informes de avance de los proyectos.
- Revisión y evaluación de los entregables al final de cada etapa que defina la metodología.
- Capacidad de detectar errores en etapas tempranas.

Estudiante

- Conocimiento y adopción de una metodología de desarrollo de software, apropiándose de buenas prácticas y mejorando los resultados en los siguientes proyectos.
- Uniformidad en el ciclo de vida de los proyectos, las responsabilidades de los participantes, los entregables que deben generarse, y las tareas que deben realizarse.

- Retroalimentación de la interpretación que se hace de la metodología.

- Mejora de los entregables, a medida que avancen las etapas de la metodología, y se ejecuten proyectos académicos de desarrollo de software.

- Corrección del rumbo, desde etapas tempranas, sin esperar hasta el final donde no hay margen de tiempo para cambios.

En el contexto empresarial, un número importante de las compañías que conforman la industria de desarrollo de software local, han establecido metodologías de desarrollo de software, la mayoría de ellas, adaptaciones de RUP (Proceso Unificado de Rational, por sus siglas en español), buscando ser más eficientes y rentables en la tarea de desarrollar software. Recordemos que RUP es una metodología estándar, ampliamente utilizada, que incorpora la mejores prácticas que se han probado en el campo, y que tiene como objetivo hacer alcanzables tanto pequeños como grandes proyectos de software.

Las compañías que definieron y apropiaron su proceso de desarrollo de software, han decidido conseguir la certificación o evaluación de sus procesos, de acuerdo a estándares internacionales; de tal forma que les permita llegar a nuevos clientes y mercados, los cuales son más exigentes en los requisitos que deben cumplir sus proveedores de servicios.

Por tanto, el estudiante de la Universidad EAFIT, al habituarse al seguimiento de una metodología de desarrollo de software, dentro de su formación profesional, le permitirá adquirir habilidades que facilitarán, agilizarán y aumentarán sus posibilidades de inmersión en el entorno laboral.

En el contexto laboral, las compañías que pertenecen a la industria del desarrollo de software, se verán favorecidas por los profesionales de la Universidad EAFIT, de la siguiente forma:

- Menor inversión en capacitación y entrenamiento de los profesionales que egresan de la Universidad EAFIT.

- Mayor rentabilidad económica, en los proyectos donde participen los profesionales que egresan de la Universidad EAFIT.

- Capacidad para asumir nuevos proyectos, sin afectar considerablemente los factores de calidad, presupuesto y tiempo.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Ayudar en la construcción de un marco de trabajo para la Universidad EAFIT, implementando una metodología que guiará la ejecución de los proyectos académicos de desarrollo de software, a través de las disciplinas de Gestión de proyectos, Modelado Empresarial, Requisitos, Análisis y Diseño, Implementación, Pruebas, Despliegue, y Gestión del cambio; teniendo como referente la metodología estándar RUP (Proceso Unificado de Rational, por sus siglas en español).

3.2 OBJETIVOS ESPECÍFICOS

- Conceptualizar y contextualizar las metodologías de desarrollo ampliamente utilizadas y probadas (metodologías referentes), en el ámbito académico de la Universidad EAFIT.

- Identificar los diferentes tipos de proyectos académicos de desarrollo de software, que se deben considerar en la metodología de desarrollo.

- Definir el ciclo de vida de desarrollo de software y las disciplinas correspondientes, para cada tipo de proyecto académico.

- Definir artefactos y entregables para cada disciplina objeto de este proyecto, de acuerdo a cada tipo de proyecto académico.

- Definir los roles y responsabilidades de los actores que participan en cada disciplina, de acuerdo a cada tipo de proyecto académico.

- Definir las actividades que se deben realizar en cada disciplina objeto de este proyecto, de acuerdo a cada tipo de proyecto académico.

- Definir los flujos de trabajo de cada disciplina; para la ejecución de cada tipo de proyecto académico.

- Conceptualizar, diseñar e implementar las metodologías necesarias para orientar los tipos de proyecto identificados.

- Crear, configurar y publicar la metodología de desarrollo de software, considerada para el ámbito académico de la Universidad EAFIT, utilizando el marco de trabajo EPF (Eclipse Process Framework).

4. MARCO TEÓRICO

El marco teórico hace referencia a tópicos fundamentales para el desarrollo de este proyecto de grado, es decir, modelos de calidad y metodologías de desarrollo de software. En este marco se describirán referentes importantes de estas dos temáticas, permitiendo establecer la base conceptual para implementar la metodología que busca orientar los proyectos académicos de desarrollo de software, en la Universidad EAFIT. Adicionalmente, se dará un concepto de la herramienta software que permitirá la creación, configuración y publicación de la metodología que se vaya a definir.

4.1 INGENIERÍA DE SOFTWARE

Hoy en día el software juega un papel central en casi todos los aspectos de la vida cotidiana, tales como, gobierno, finanzas, educación, transporte, medicina, etc. De hecho, actualmente la complejidad de los sistemas sigue creciendo, y anualmente se gastan miles de millones de pesos en actividades de desarrollo de software.

Según Roger Pressman[1], la Ingeniería de Software es una tecnología multicapa, ilustrada en la Figura 1.

Figura 1. Capas de la Ingeniería de Software

[1] PRESSMAN, Roger S. Software Engineering. A practitioner's approach. 5 ed. New York: McGraw-Hill Higher Education, 2000. p. 20

Estas capas se describen de la siguiente forma:

- Cualquier disciplina de ingeniería (incluida la Ingeniería de Software) debe basarse en un compromiso completo con la calidad. La gestión de calidad total y filosofías similares fomentan una cultura de mejoramiento continuo de los procesos, y esta cultura conduce al desarrollo de enfoques cada vez más maduros de Ingeniería de Software. El principio que soporta la Ingeniería de Software es un enfoque en la calidad.

- El fundamento de la Ingeniería de Software es la capa de proceso. El proceso define un marco de trabajo para un conjunto de áreas, las cuales forman la base para el control de gestión de proyectos de software, y forman el contexto donde se aplican métodos técnicos, se producen artefactos o productos de trabajo (especificaciones, modelos, documentos, reportes, etc.), se establecen hitos, se asegura la calidad, y los cambios se gestionan adecuadamente.

- Los métodos en Ingeniería de Software indican cómo construir técnicamente el software. Los métodos abarcan una amplia cantidad de tareas que incluyen análisis de requisitos, diseño, codificación, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

- Las herramientas en Ingeniería de Software proporcionan una ayuda automática o semiautomática para el proceso y sus métodos.

El objetivo de la Ingeniería de Software es obtener productos de software de calidad (durante su elaboración, desde el inicio hasta el final), mediante un proceso apoyado por métodos y herramientas.

Actualmente la Ingeniería de Software ha evolucionado hacia modelos que orientan a las organizaciones en la mejora de su capacidad para desarrollar software de calidad (capa de calidad), y también se han definido metodologías para la gestión y el desarrollo de software (capa de procesos y capa de métodos).

4.2 CMMI – MODELO DE CALIDAD

El Capability Maturity Model Integration (CMMI, de aquí en adelante) es un marco de referencia, que las organizaciones pueden emplear para evaluar y mejorar sus procesos de desarrollo, adquisición, y mantenimiento de productos y servicios. CMMI es la generación de una línea de modelos de madurez, la cual se inició a principios de los noventa con el CMM-SW (Capability Maturity Model for Software Engineering). Se les denomina modelos de madurez porque proponen adoptar dichas prácticas en forma gradual: primero deben ponerse en práctica áreas de proceso pertenecientes a un nivel determinado, para luego, sobre esta base, introducir las correspondientes al nivel siguiente.

4.2.1 Origen

CMMI, como se conoce ahora, surgió a partir de otro modelo llamado CMM (Capability Maturity Model), que fue desarrollado por el SEI (Software Engineering Institute), que pertenece a la Carnegie Mellon University, como fruto de muchas investigaciones. Dentro de estos trabajos se encontró que una organización posee tres elementos principales en los cuales pueden enfocarse para mejorar, son: personas, métodos y procedimientos y equipos y herramientas. Sin embargo, el componente más importante, encargado de mantener todos estos elementos de la estructura organizacional, unidos y en movimiento, son los procesos utilizados. Estos son los que realmente permiten que una empresa alcance la escalabilidad que se necesite, y lo más importante, incorpore conocimiento de cómo hacer las cosas mejor. Los procesos son los que permiten que dentro de una entidad se impulsen los recursos y se examinen las tendencias de los negocios, para poder seguirle el ritmo a los cambios continuos en la forma de hacer las cosas. De esta forma el SEI enunció la premisa, la cual sería su base de trabajo: “La calidad de un sistema o un producto está altamente influenciada por la calidad de los procesos utilizados para desarrollarlo y mantenerlo”

Se puede decir entonces que CMM permite determinar la capacidad, de las organizaciones de desarrollo de software, para producir de manera consistente y predecible productos de calidad superior. El modelo brinda guías para seleccionar estrategias de mejoramiento del proceso, mediante la determinación de las capacidades actuales del proceso y la identificación de los puntos críticos para mejorar el proceso y la calidad de software.

Desde 1991, CMM ha sido desarrollado dentro de muchas disciplinas, algunas de las más importantes son, modelos para Ingeniería de Sistemas, Ingeniería de Software y Adquisición de Software. Sin embargo, aunque los trabajos

desarrollados para estas áreas probaron ser exitosos y altamente útiles para organizaciones en diferentes industrias, el uso de múltiples modelos se convirtió en un problema debido a que las empresas querían que sus mejoras pudieran abarcar muchos grupos en diversas áreas. Por este motivo, las marcadas diferencias entre los modelos de disciplinas (Ingeniería de Sistemas, Ingeniería de Software y Adquisición de Software), utilizados por cada grupo, limitaban esta labor, con el agregado que al aplicar modelos, que no están integrados, dentro y a lo largo de toda una organización es costoso en términos de entrenamiento, evaluaciones y actividades de mejora. Con el objeto de solucionar este problema, el SEI se concentró en reunir en un solo marco de trabajo, los elementos más importantes de tres modelos igualmente sobresalientes, Capability Maturity Model for Software (CMM-SW), Systems Engineering Capability Maturity Model (SE-CMM) y Integrated Product Development Capability Maturity Model (IPD-CMM), y así fue como nació el proyecto CMM Integration (CMMI), un modelo que combina diversas disciplinas, que es suficientemente flexible como para soportar las numerosas aproximaciones de los modelos fuentes, y que además puede ser utilizado por aquellos que trabajan con los tres modelos originales o por aquellos que apenas están comenzando a manejar el concepto de CMM. Entonces CMMI es la evolución de CMM-SW, SE-CMM y el IPD-CMM.

Las mejores prácticas CMMI se publican en documentos llamados modelos. En la actualidad hay tres áreas de interés cubiertas por los modelos de CMMI: Desarrollo, Adquisición y Servicios.

En la versión actual de CMMI hay tres constelaciones:

- CMMI para el Desarrollo (CMMI-DEV o CMMI for Development). En él se tratan procesos de desarrollo de productos y servicios.

- CMMI para la Adquisición (CMMI-ACQ o CMMI for Acquisition). En él se tratan la gestión de la cadena de suministro, adquisición y contratación externa en los procesos del gobierno y la industria.

- CMMI para Servicios (CMMI-SVC o CMMI for Services), está diseñado para cubrir todas las actividades que necesitan gestionar, establecer y entregar servicios.

Dentro de la constelación CMMI-DEV, existen dos modelos:

- CMMI-DEV
- CMMI-DEV + IPPD (Integrated Product and Process Development)

Independientemente de la constelación/modelo que elige una organización, las prácticas CMMI deben adaptarse a cada organización en función de sus objetivos de negocio.

4.2.2 Capacidad y madurez

La capacidad del proceso es la habilidad inherente de un proceso para producir los resultados planeados. El principal objetivo de un proceso de software maduro, es el de producir productos de calidad que cumplan los requisitos del usuario. Cuando se habla de madurez del proceso se entiende como el crecimiento alcanzado en la capacidad del proceso de software, y se considera como una actividad de largo plazo.

En una compañía de software inmadura, el proceso de software es generalmente improvisado, no existen planes rigurosos, sus actividades se enfocan en resolver las crisis que se presentan, carecen de bases objetivas para evaluar la calidad de los productos o para resolver los problemas que se presentan. De manera contraria, cuando una organización alcanza cierto grado de madurez posee una gran habilidad para administrar el proceso de desarrollo y mantenimiento de software, se hacen pruebas y análisis costo-beneficio para mejorar el proceso, el director supervisa la calidad del producto y la satisfacción del cliente, se llevan registros, y todos las personas están involucrados en el proceso de desarrollo.

4.2.3 Proceso maduro

Para que un proceso pueda considerarse maduro debe cumplir las siguientes características:

- Está definido. El proceso es claro, sistemático, y suficientemente detallado. Además existe acuerdo entre el personal, la gerencia, y los proyectos respecto al proceso que se va a utilizar.

- Está documentado. El proceso está publicado, aprobado y fácilmente accesible. Una de las mejores maneras es a través de una Intranet, para apoyar los proyectos de desarrollo.

- El personal ha sido formado en el proceso. Los ingenieros de software y la gerencia han recibido cursos y entrenamiento en cada proceso que aplica a su trabajo.

- Es practicado. El proceso definido debe ser utilizado en las tareas habituales llevadas a cabo por los proyectos. El entrenamiento y la adaptación del proceso a la realidad de la compañía deben garantizar su aplicación en la vida real.

- Es apoyado. La gerencia no solo debe firmar y promover los procesos definidos, sino que además debe asignar responsabilidades al personal y a los directores de proyecto para su cumplimiento.

- Es mantenido. El proceso es revisado regularmente, para asegurarse que está adaptado para satisfacer las necesidades reales de los proyectos.

- Está controlado. Los cambios y actualizaciones del proceso son revisados, aprobados, y comunicados oportunamente a todos los usuarios.

- Se verifica. La gerencia mantiene mecanismos para asegurarse que todos los proyectos siguen el proceso actual.

- Se mide. La utilización, los beneficios y el rendimiento resultante del proceso se miden regularmente.

- Puede mejorarse. Existen mecanismos y apoyo a la gerencia para revisar e introducir cambios en el proceso, que permitan mejorar su eficacia e incorporar nuevas metodologías.

Por el contrario, un proceso inmaduro se caracteriza porque es fundamentalmente personal, no está documentado, es difícil compartirlo con otros miembros del equipo de trabajo, no es fácil reproducirlo en nuevos proyectos, no hay entrenamiento, no todo el mundo lo conoce, no se mide, no se aplica permanentemente (solo en algunas ocasiones), es percibido como poco eficiente, es interpretado de manera distinta, etc.

4.2.4 Estructura del modelo

El modelo CMMI consta de cinco niveles (representación escalonada), diseñados de manera que los niveles inferiores proveen las bases para que, de forma progresiva, se alcancen los superiores. Estas cinco etapas de desarrollo son conocidas como niveles de madurez (ver Figura 2), y en cada uno la organización alcanza una capacidad superior del proceso. CMMI plantea que una organización puede ubicarse en alguno de los cinco posibles niveles de madurez, dependiendo del grado de perfección de sus procesos.

Figura 2. CMMI - Niveles de madurez

- Nivel de madurez – Inicial. Las organizaciones en este nivel no disponen de un ambiente estable para el desarrollo y mantenimiento de software. Aunque se utilicen técnicas correctas de ingeniería, los esfuerzos se ven consumidos por falta de planificación. El éxito de los proyectos se basa, la mayoría de las veces, en el esfuerzo personal, aunque a menudo se producen fracasos y casi siempre retrasos y sobrecostos. El resultado de los proyectos es impredecible.

- Nivel de madurez – Administrado. En este nivel las organizaciones disponen de unas prácticas institucionalizadas de gestión de proyectos, existen unas métricas básicas y un razonable seguimiento de la calidad. La relación con subcontratistas y clientes está gestionada sistemáticamente.

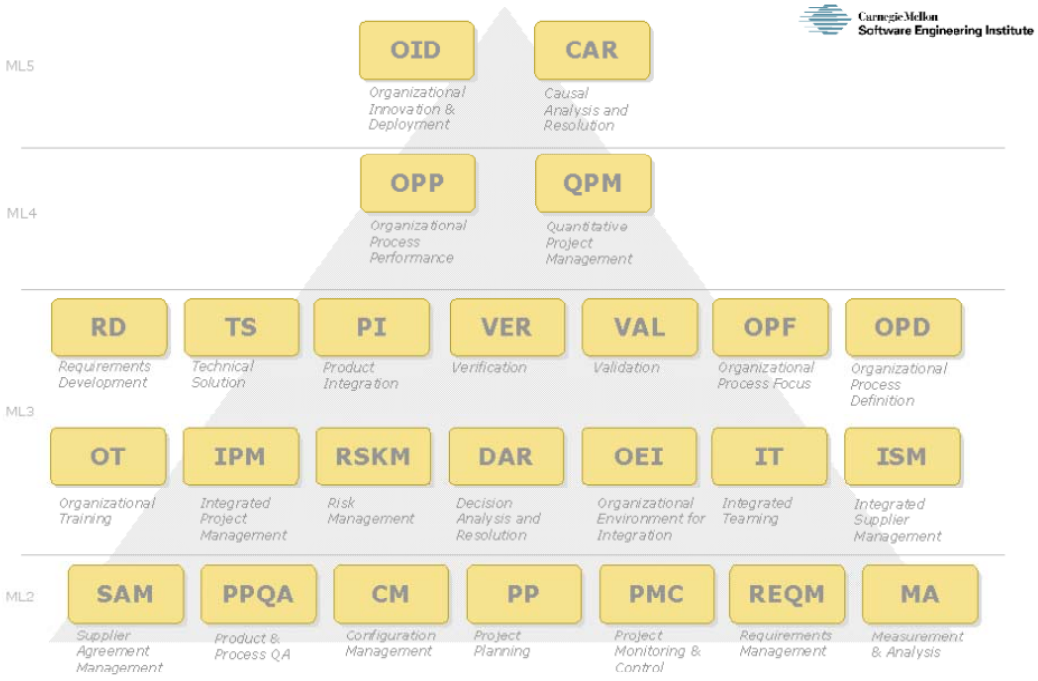
- Nivel de madurez – Definido. El proceso de software, para las actividades administrativas y técnicas, está documentado, estandarizado, e integrado en un proceso de software estándar dentro de la organización, que ayudará a obtener un rendimiento más efectivo. El grupo que trabaja en el proceso enfoca y guía sus esfuerzos al mejoramiento del proceso, facilita la introducción de técnicas y métodos, e informa a la gerencia el estado del proceso. La capacidad del proceso está basada en una amplia compresión, dentro de la organización, de las actividades, roles, y responsabilidades definidas en el proceso de software.

- Nivel de madurez – Administrado cuantitativamente. Se caracteriza porque las organizaciones disponen de un conjunto de métricas significativas de calidad y productividad, que se usan de modo sistemático para la toma de decisiones y la gestión de riesgos. El software resultante es de alta calidad.

- Nivel de madurez – Optimizado. La organización completa está volcada en la mejora continua de los procesos. Se hace uso intensivo de las métricas y se gestiona el proceso de innovación.

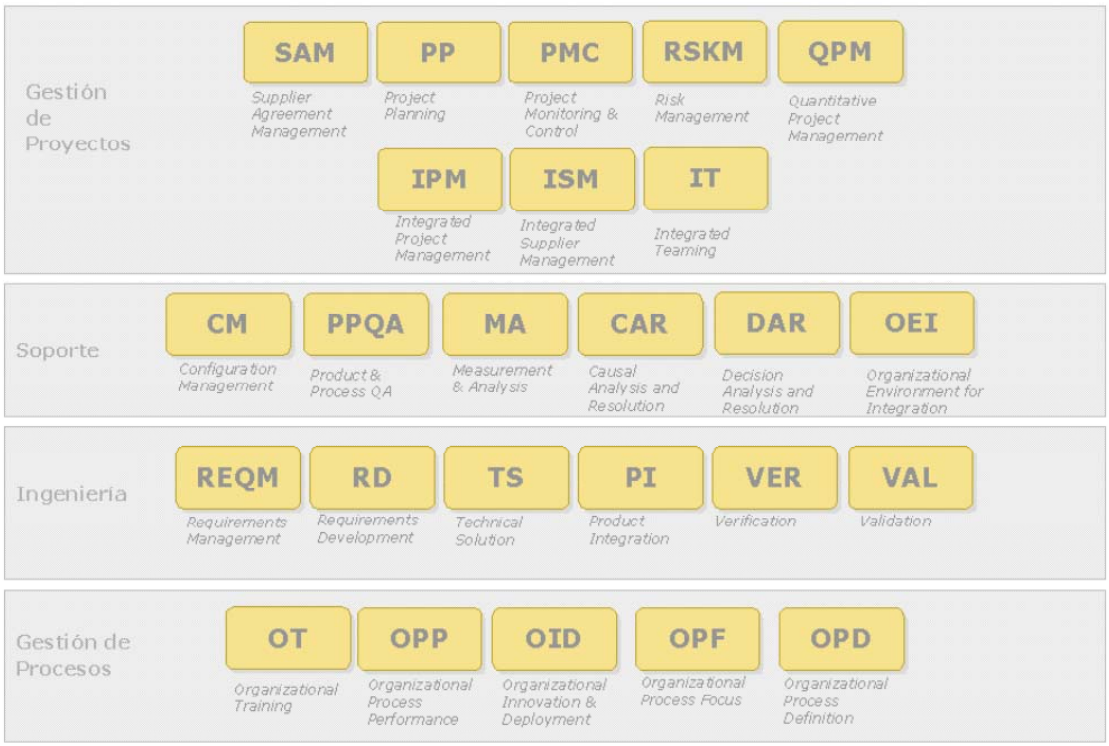
Cada nivel de madurez, con excepción del inicial, se caracteriza por un conjunto de áreas de proceso que agrupan buenas prácticas que, al ser ejecutadas colectivamente, permiten cumplir con algún objetivo que es considerado importante para el modelo. En la Figura 3, se ilustra las áreas de proceso en cada nivel de madurez, y se puede apreciar que antes de introducir prácticas de un nivel determinado deben estabilizarse las prácticas del nivel anterior.

Figura 3. Áreas de proceso por nivel de madurez



Además de presentar las áreas de proceso por nivel de madurez, el modelo propone una vista alternativa (representación continua), en donde las áreas están agrupadas por categoría, estas categorías reciben el nombre de niveles de capacidad (ver Figura 4). Esta es una diferencia sustancial respecto a los modelos anteriores, que sólo permitían una representación u otra (por ejemplo, CMM-SW proponía una representación por niveles de madurez, mientras que el SE-CMM lo hacía por niveles de capacidad).

Figura 4. Áreas de proceso por nivel de capacidad



4.2.5 Áreas de proceso

Un área de proceso se define como un conjunto de prácticas al interior de un área, que cuando se implementan colectivamente, satisfacen un conjunto de objetivos que son considerados esenciales para lograr una mejora en esa área.

En CMMI, las áreas de proceso son el sustento del modelo y constituyen los elementos claves para definir la madurez de una organización de software. Estas constituyen un grupo de prácticas relacionadas con un área de trabajo del modelo, en donde se definen específicamente los objetivos de cada una de esas áreas.

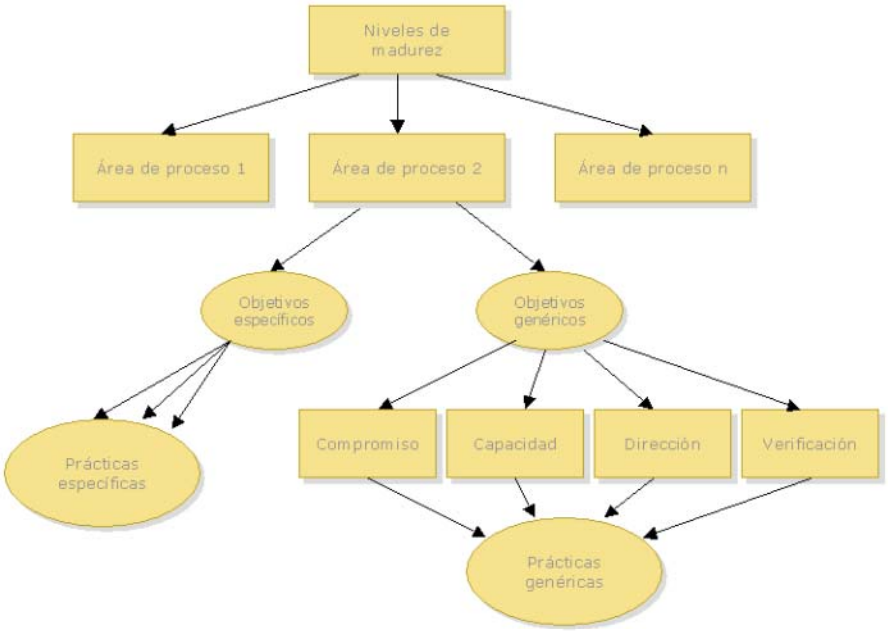
El modelo CMMI define las siguientes áreas de proceso:

- Análisis de Causas y Resolución (CAR)
- Gestión de la configuración (CM)
- Análisis de Decisiones y Resolución (DAR)
- Gestión Integrada de Proyectos (IPM)
- Medición y Análisis (MA)
- Innovación y Despliegue Organizacionales(OID)
- Definición de Procesos Organizacionales (OPD)
- Enfoque Organizacional del Proceso (OPF)
- Rendimiento del Proceso Organizacional (OPP)
- Entrenamiento Organizacional (OT)
- Monitorización y Control de Proyectos (PMC)
- Planificación de Proyectos (PP)
- Aseguramiento de Calidad de Procesos y Productos (PPQA)
- Integración de Producto (PI)
- Gestión Cuantitativa de Proyectos (QPM)
- Gestión de Requisitos (REQM)
- Desarrollo de Requisitos (RD)
- Gestión de Riesgos (RSKM)
- Gestión de Acuerdos con Proveedores (SAM)
- Solución Técnica (TS)
- Validación (VAL)
- Verificación (VER)

4.2.6 Arquitectura del modelo

En la representación escalonada, cada nivel de madurez contiene varias áreas de proceso, las que a su vez quedan definidas por un objetivo genérico, y uno o varios objetivos específicos. Cada uno de ellos tiene vinculado un conjunto de prácticas, llamadas genéricas y específicas respectivamente (ver Figura 5).

Figura 5. Estructura de la representación por niveles



- Los objetivos y prácticas genéricas tienen que ver con el grado de institucionalización de los procesos (compromiso con la ejecución, capacidad para ejecutar, dirección de la ejecución, verificación de la ejecución). Son llamados así porque son los mismos para múltiples áreas de proceso (aunque hay aspectos específicos para cada una de ellas). Cumplir con un objetivo genérico de un área de proceso determinada, implica tener un mayor control de la planificación e implementación de los procesos vinculados a esa área de proceso.

- Los objetivos y prácticas específicas están vinculados a un área de proceso determinada. Son considerados elementos que deben ser satisfechos para implementar exitosamente los procesos relacionados con un área de proceso en particular.

Componentes obligatorios

Los componentes obligatorios describen lo que una organización debe alcanzar para satisfacer un área. Este alcance debe ser visiblemente implementado en los procesos de una organización. Los componentes obligatorios en CMMI son los objetivos genéricos y los específicos. Ahora bien, la satisfacción de tales objetivos

es utilizada en las evaluaciones, como base para decidir si hay cumplimento de un área de proceso.

- Objetivos genéricos. Son llamados genéricos porque el mismo objetivo aplica para muchas áreas de proceso. Así mismo, describen una característica que debe estar presente para institucionalizar los procesos que se ubican dentro de un área.

- Objetivos específicos. Describen las características que deben estar presentes para satisfacer un área de proceso. Este tipo de componentes al igual que los objetivos generales, son elementos usados en evaluaciones de madurez para determinar si hay cumplimiento de un área de proceso.

Componentes esperados

Los componentes esperados describen lo que una organización puede implementar para alcanzar un componente obligatorio, y guían a aquellos que realizan mejoramiento de procesos o evaluaciones de rendimiento. Estos componentes incluyen las prácticas específicas y genéricas.

- Prácticas genéricas. Son llamadas genéricas porque la misma práctica aplica para múltiples áreas de proceso y pueden ser definidas como descripciones de actividades que son importantes en la consecución de los objetivos genéricos.

- Prácticas específicas. Es una descripción de una actividad que es considerada importante para alcanzar un objetivo específico.

CMMI, como su nombre lo indica, es un modelo que propone un conjunto de mejores prácticas, que pueden emplearse para evaluar y mejorar procesos; de ninguna manera debe suponerse que estamos ante la descripción de un proceso. Definir el proceso de desarrollo de software, es un trabajo que le compete a la organización de desarrollo, de tal forma que cumpla con los atributos y mejores prácticas propuestos por el modelo.

4.3 METODOLOGÍAS DE DESARROLLO

Un proceso de software detallado y completo suele denominarse metodología. Una metodología debe definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología a proyectos, guías para uso de herramientas de apoyo, etc.

El objetivo de una metodología de software es incrementar la calidad del software (en todas las fases por las que pasa), a través de una mayor transparencia y control del proceso. Da igual si es un ámbito académico o empresarial, hay que producir lo esperado, en el tiempo esperado, y dentro de los costos esperados. El proceso de desarrollo de software debe hacer que las decisiones para aumentar la calidad sean reproducibles en cada desarrollo.

Metodologías formales

Las metodologías formales son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema. Puede decirse, que RUP (Proceso Unificado de Rational, por sus siglas en español) es un caso particular, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto, realizando una configuración adecuada, podría considerarse una metodología ágil.

Metodologías ágiles

Un proceso es ágil cuando el desarrollo de software es incremental (entregas pequeñas de software, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es fácil de aprender y modificar, bien documentado), y adaptable (permite realizar cambios de último momento).

4.3.1 RUP (Rational Unified Process)

El Proceso Unificado de Rational (Rational Unified Process en inglés, y habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

En definitiva, el RUP es una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software, con el objetivo de hacer realizables tanto pequeños como grandes proyectos software.

RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

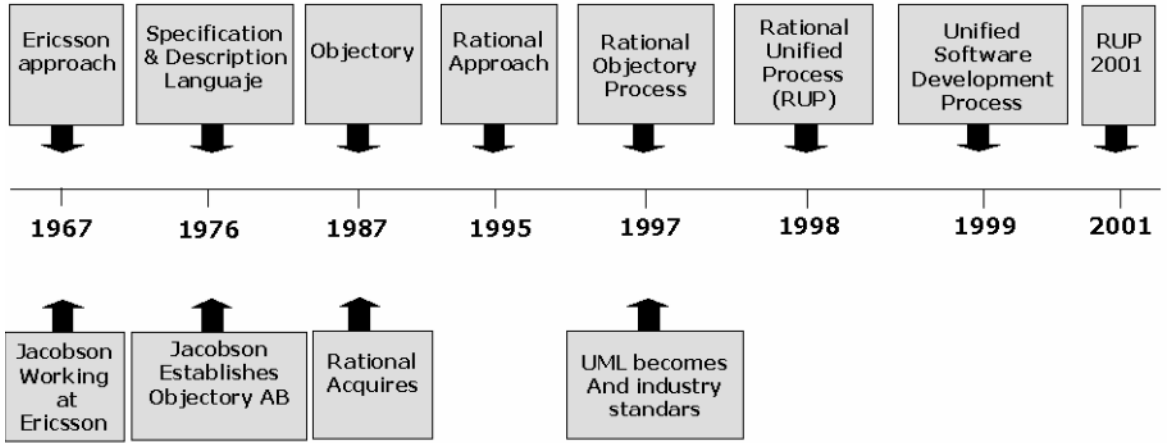
4.3.1.1 Origen

El antecedente más importante se ubica en 1967 con la Metodología Ericsson (Ericsson Approach) elaborada por Ivar Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de Caso de Uso. Entre los años de 1987 a 1995, Ivar Jacobson fundó la compañía Objectory AB y lanza el proceso de desarrollo Objectory (abreviación de Object Factory).

Posteriormente en 1995, Rational Software Corporation adquiere Objectory AB, y entre 1995 y 1997 se desarrolla ROP (Rational Objectory Process) a partir de Objectory 3.8 y del Enfoque Rational (Rational Approach), adoptando UML como lenguaje de modelado.

Desde ese entonces y a la cabeza de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software desarrolló e incorporó diversos elementos para expandir ROP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza RUP (Rational Unified Process). En la Figura 6, se puede apreciar la cronología de la historia de RUP.

Figura 6. Cronología de RUP



4.3.1.2 Características

Las características principales de RUP son:

- Guiado/Manejado por casos de uso. La razón de ser de un sistema software es servir a los usuarios, ya sean humanos u otros sistemas; un caso de uso es una facilidad que el software debe proveer a sus usuarios. Los casos de uso constituyen la guía fundamental, establecida para las actividades a realizar durante todo el proceso de desarrollo, incluyendo el diseño, la implementación y las pruebas del sistema.

- Centrado en la arquitectura. La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requisitos no funcionales. Es como una radiografía del sistema que estamos desarrollando, lo suficientemente completa como para que todos los implicados en el desarrollo tengan una idea clara de qué es lo que están construyendo, pero lo suficientemente simple. Se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de lo demás. Todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, recibe este nombre porque lo forman las vistas lógica, de implementación, proceso y despliegue, más la de casos de uso que es la que da cohesión a todas.

- Iterativo e incremental. Para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, cada una de las cuales debe ser considerada como un mini-proyecto, cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo.

- Desarrollo basado en componentes. La creación de sistemas de software necesita dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen, o que se desarrollan, y maduran sus componentes.

- Utilización de un único lenguaje de modelado. UML es adoptado como único lenguaje de modelado para el desarrollo de todos los modelos.

- Proceso integrado. Se establece una estructura que abarca los ciclos, fases, flujos de trabajo, mitigación de riesgos, control de calidad, gestión del proyecto y control de configuración. Además esta estructura cubre a las herramientas para soportar la automatización del proceso, soportar flujos individuales de

trabajo, para construir los diferentes modelos, e integrar el trabajo a través del ciclo de vida y a través de todos los modelos.

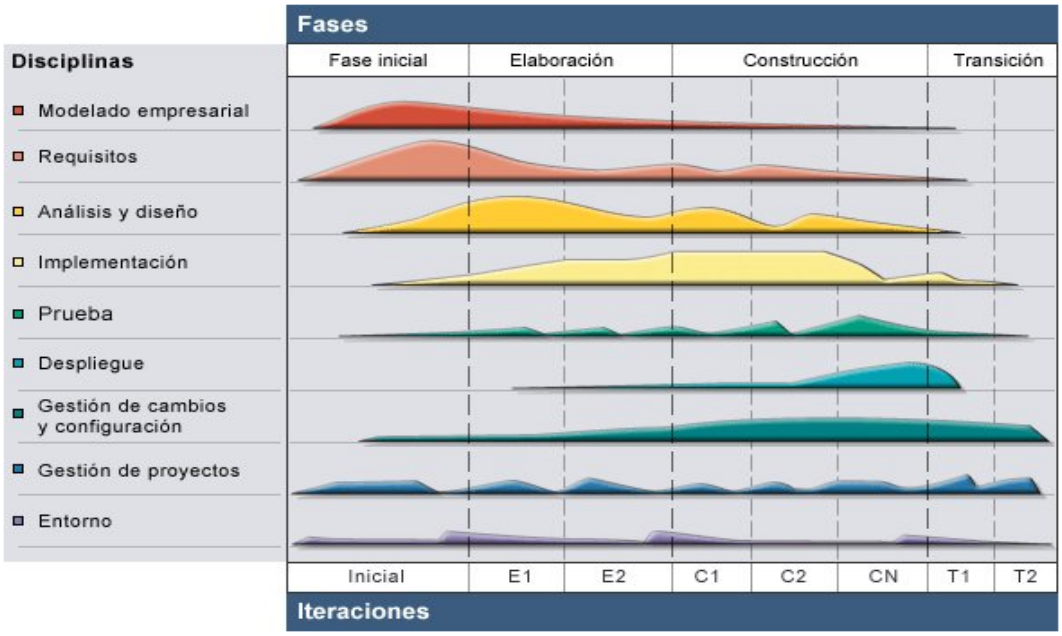
4.3.1.3 Ciclo de vida

La estructura de RUP consta de dos dimensiones (ver Figura 7)

- Fases: donde se encuentran las cuatro principales etapas del proyecto, representa el tiempo y contiene aspectos dinámicos del proceso. Cada una de estas etapas se encuentra dividida por iteraciones.

- Disciplinas: representan las actividades lógicas y describen el proceso en términos de componentes de proceso, flujos de trabajo, actividades y roles.

Figura 7. Ciclo de vida RUP

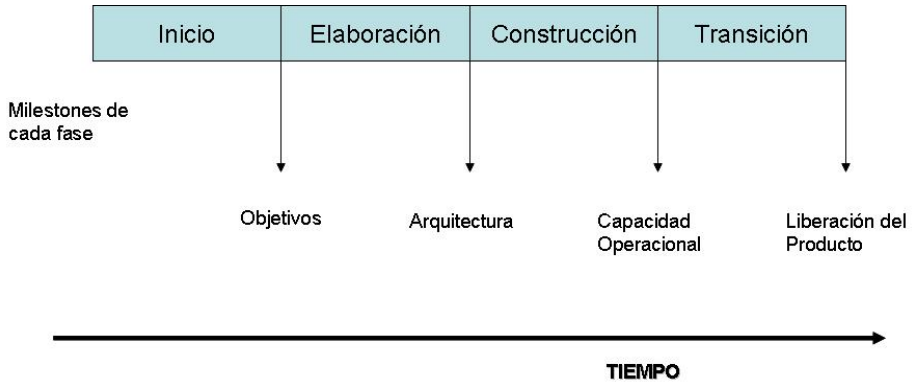


Cada fase termina con unos criterios de evaluación previamente definidos, que permiten determinar si es apropiado o no continuar a la siguiente fase o si alguno de los componentes de la fase actual debe verificarse y evaluarse nuevamente para proseguir. En cada uno de estos puntos, los patrocinadores del proyecto y aquellos que lo están ejecutando realizan estimaciones sobre el proyecto para

posteriores acciones a tomar, teniendo en cuenta lo que se ha hecho y la planeación de lo que se debe hacer.

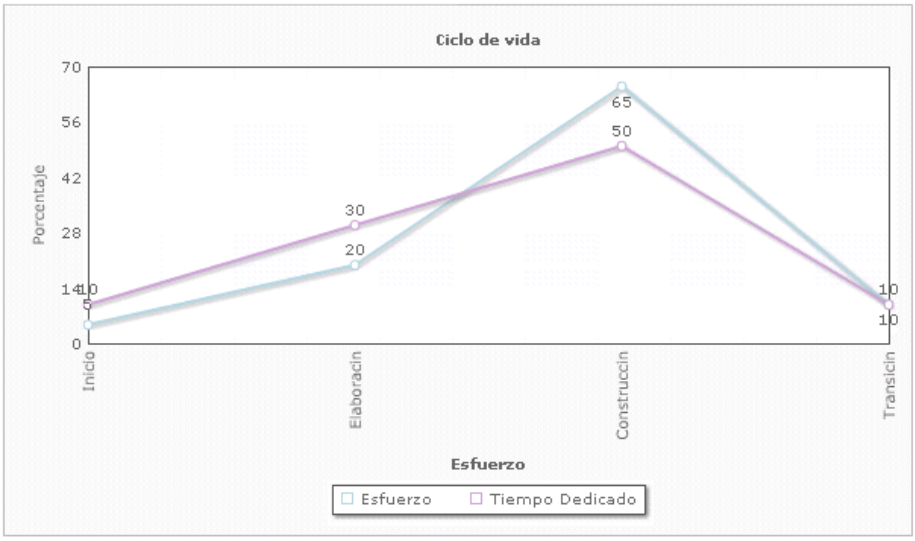
Desde la perspectiva de gestión del proyecto, cada fase del ciclo de vida concluye cuando se alcanza un hito (ver Figura 8). La finalización de cada una de las fases permite valorar y aprobar el cumplimiento de los objetivos de dicha fase para continuar con la siguiente.

Figura 8. Hitos en RUP



La fase más larga y en la que se necesita mayor esfuerzo es la fase de construcción, como podemos ver en la Figura 9.

Figura 9. Distribución de esfuerzo y tiempo en RUP



Fases

El proceso de software diseñado por RUP se divide principalmente en 4 fases, al completar las 4 fases se cumple un ciclo, logrando un producto para el cliente.

- Fase de Inicio

  Durante la fase de inicio se desarrolla una descripción del producto final, y se presenta el análisis del negocio.

  Objetivos

  - Especificar la visión final del producto y sus casos de negocios.
  - Afinar el alcance del proyecto
  - Establecer los servicios de negocio a desarrollar

  Hito: Objetivos del sistema

- Fase de Elaboración

  Durante la fase de elaboración se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura.

  Objetivos

  - Definir la arquitectura del sistema
  - Definir y especificar los servicios
  - Definir la interfaz gráfica del usuario

  Hito: Línea base de la arquitectura

- Fase de Construcción

  Durante la fase de construcción se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo.

  Objetivos

  - Construir el producto, evolucionando la visión del negocio, la arquitectura, y los planes; hasta que el producto esté listo para transferencia a la comunidad de usuarios

- Desarrollar los servicios, casos de uso (funcionalidades), módulos, subsistemas del proyecto.
- Realizar desarrollo por iteraciones, priorizando los servicios, casos de usos, módulos y/o subsistemas más críticos para el negocio.
- Realizar pruebas funcionales de cada uno de los servicios, casos de usos, módulos y/o subsistemas a medida que se vayan desarrollando.

Hito: Capacidad operativa inicial

- Fase de Transición

Durante la fase de transición se busca finalizar el producto para la entrega al usuario. Finalizada esta fase se obtiene una versión del producto.

Objetivo

- Lograr la transición desde el producto a la comunidad de usuarios, la cual incluye: manufactura, entrega, entrenamiento, soporte, y mantenimiento del producto.

Hito: Lanzamiento del producto

Disciplinas

- Modelado de procesos

El propósito de la disciplina de modelado de procesos consiste en entender el negocio de la organización, y entender los procesos que se involucran para el desarrollo del producto.

Su alcance es:

- Evaluar el estado actual de la organización y su capacidad de adaptar nuevos sistemas, para apoyar procesos específicos de los cuales ha surgido una necesidad.
- Explorar los procesos actuales del negocio, los roles y las responsabilidades.
- Identificar y evaluar estrategias potenciales para modificar los procesos, si es necesario.

- Desarrollar modelos de dominio que representen la unidad de negocio relacionada con el desarrollo de un nuevo proyecto.

- Requisitos

  El propósito de la disciplina de requisitos es establecer entre todos los interesados, un acuerdo frente a lo que el sistema pueda hacer y a su vez define el alcance total del sistema.

  Su alcance es:

  - Establecer y mantener un acuerdo sobre cuál es la funcionalidad esperada.
  - Entregar a los desarrolladores una descripción clara de los requisitos del sistema.
  - Definir los límites del sistema.
  - Definir el “Look and Feel” de la aplicación.
  - Proveer la base para la creación del plan de iteraciones, la estimación de costos y esfuerzos.

- Análisis y diseño.

  El propósito de la disciplina de análisis y diseño es transformar los requisitos de negocio en especificaciones de software, detallar y probar la arquitectura de la solución.

  Su alcance es:

  - Transformar los requisitos en el diseño del sistema a implementar.
  - Definir una arquitectura robusta para el sistema.
  - Entregar a los desarrolladores una especificación clara de los requisitos del sistema.
  - Adaptar el diseño al ambiente de implementación.

- Implementación

  El propósito de la disciplina de Implementación es codificar o configurar los componentes de software que conforman la solución para ensamblarlos y desplegarlos en unidades totalmente funcionales.

  Su alcance es:

- Implementar la solución en términos de creación, codificación y personalización de componentes.
- Definir la organización de componentes como subsistemas o módulos.
- Probar los componentes como unidades aisladas.
- Liberar módulos totalmente funcionales.

- Pruebas

  El propósito de la disciplina de pruebas es evaluar y determinar la calidad del producto.

  Su alcance es:

  - Buscar y documentar defectos encontrados en el software.
  - Determinar la calidad del software.
  - Validar la especificación de los requisitos frente al sistema liberado.
  - Validar el diseño y arquitectura del producto de software.
  - Validar la implementación de los requisitos.
  - Asegurar que todos los defectos fueron corregidos antes de desplegar el producto.
  - Asegurar la correcta ejecución del proceso de desarrollo.

- Despliegue

  El propósito de la disciplina de despliegue es asegurar que el producto quede disponible para su utilización, por parte de los usuarios finales del proyecto.

  Su alcance es:

  - Especificar la instalación del producto.
  - Probar la instalación del producto.
  - Crear el material para consulta y capacitación del usuario.
  - Validar el diseño del producto de software.
  - Asegurar la implementación de los requisitos suplementarios (requisitos no funcionales).
  - Entregar el producto funcionando al usuario.

- Gestión de la configuración y el cambio

  El propósito de la disciplina de gestión de la configuración y el cambio es la definición y conducción de los artefactos que deben ser controlados, dado su alto nivel de interacción dentro del proyecto.

  Su alcance es:

  - Controlar los cambios a los requisitos.
  - Controlar los cambios a los artefactos.
  - Mantener la integridad de los artefactos.
  - Versionar los artefactos.
  - Permitir conocer el estado de un artefacto.

- Gestión del Proyecto

  El propósito de la disciplina de gestión del proyecto es proveer un marco de planeación, ejecución, coordinación y control de todas las actividades que hacen parte del proyecto, además de unificar los canales de comunicación dentro del proyecto.

  Su alcance es:

  - Proveer un marco de trabajo para proyectos de implementación y desarrollo de software.
  - Proveer prácticas para la planeación, ejecución y monitoreo de proyectos.
  - Proveer un marco de trabajo para el manejo de riesgos.

- Gestión del Ambiente

  El propósito de la disciplina de gestión del ambiente es proveer el soporte necesario para la implementación, describiendo los ambientes, las herramientas y la logística para facilitar la ejecución del proyecto.

  Su alcance es:

  - Proveer la infraestructura (hardware, software y comunicaciones) para la implementación.
  - Proveer las herramientas para la implementación.
  - Proveer las actividades, los artefactos y los lineamientos necesarios para el soporte del proceso.

4.3.1.4 Buenas prácticas

RUP identifica seis buenas prácticas, con las que define una forma efectiva de trabajar para los equipos de desarrollo de software.

- Gestión de requisitos
  RUP brinda una guía para encontrar, organizar, documentar, y seguir los cambios de los requisitos funcionales y restricciones. Utiliza una notación de Caso de Uso y escenarios para representar los requisitos.

- Desarrollo de software iterativo
  Desarrollo del producto mediante iteraciones con hitos bien definidos, en las cuales se repiten las actividades pero con distinto énfasis, según la fase del proyecto.

- Desarrollo basado en componentes
  La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes.

- Modelado visual (utilizando UML)
  UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software. Utilizar herramientas de modelado visual facilita la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. El modelado visual también ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. En resumen, el modelado visual ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.

- Verificación continua de la calidad
  Es importante que la calidad de todos los artefactos se evalúe en varios puntos durante el proceso de desarrollo, especialmente al final de cada iteración. En esta verificación las pruebas juegan un papel fundamental y se integran a lo largo de todo el proceso. Para todos los artefactos no ejecutables, las revisiones e inspecciones también deben ser continuas.

- Gestión de los cambios
  El cambio es un factor de riesgo crítico en los proyectos de software. Los artefactos software cambian no sólo debido a acciones de mantenimiento

posteriores a la entrega del producto, sino que durante el proceso de desarrollo, especialmente importantes por su posible impacto son los cambios en los requisitos. Por otra parte, otro gran desafío que debe abordarse es la construcción de software con la participación de múltiples desarrolladores, posiblemente distribuidos geográficamente, y quizás en distintas plataformas. La ausencia de disciplina rápidamente conduciría al caos. La Gestión de Cambios y de Configuración es la disciplina de RUP encargada de este aspecto.

4.3.1.5 Adaptaciones de RUP

OpenUP (Open Unified Process)

En el 2006, IBM lanza al mundo del software una metodología ágil llamada OpenUP, diseñada para el desarrollo de proyectos pequeños de software y que toma las mejores prácticas del RUP.

El OpenUp es un proceso modelo y extensible, dirigido a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Como es apropiado para proyectos pequeños y de bajos recursos

- Permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios que se precisan en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

UPEDU (Unified Process for EDUcation)

UPEDU o Proceso unificado para la educación, es un proceso de desarrollo de software especializado para educación, adaptado por Pierre-N. Robillard (École Polytechnique de Montréal), Philippe Kruchten (IBM Rational Software) y Patrick d'Astous (École Polytechnique de Montréal).

UPEDU es una adaptación de RUP para propósitos particulares, y por lo tanto hereda las características más importantes de RUP, por ejemplo:

- Es iterativo e incremental
- El proceso se divide en cuatro fases de tiempo; las fases de inicio, de elaboración, de construcción y de transición. Para cada fase, los artefactos del sistema (productos del trabajo) y los flujos del trabajo son definidos por el proceso.
- Las actividades se categorizan en disciplinas

4.3.2 Metodologías ágiles

El concepto de metodologías ágiles es relativamente nuevo pero cada vez más conocido. Éstas se centran en el uso de mínima documentación en el momento de desarrollar los proyectos y buscar ser ágiles en cuanto a dar una respuesta adecuada a los cambios de los requisitos.

Las metodologías ágiles forman parte del movimiento de desarrollo ágil de software, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto. De forma que una metodología ágil es la que tiene como principios que:

- Los individuos y sus interacciones son más importantes que los procesos y las herramientas.
- El software que funciona es más importante que la documentación exhaustiva.
- La colaboración con el cliente en lugar de la negociación de contratos.
- La respuesta delante del cambio en lugar de seguir un plan cerrado.

Las metodologías ágiles son sin duda uno de los temas recientes en ingeniería de software que acapara gran interés. Prueba de ello es que han tenido un espacio destacado en la mayoría de conferencias y workshops celebrados en los últimos años. Además ya es un área con cabida en prestigiosas revistas internacionales. En la comunidad de la ingeniería del software, se está viviendo con intensidad un debate abierto entre los partidarios de las metodologías tradicionales y aquellos que apoyan las ideas que surgieron del Manifiesto Ágil. La curiosidad que siente la mayor parte de ingenieros de software, profesores, e incluso alumnos, sobre las metodologías ágiles ha permitido un camino hacia una proyección industrial. Por un lado, para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual, considerando las dificultades de su introducción e inversión asociada en formación y herramientas. Por otro, las características de los proyectos para los cuales las metodologías ágiles han sido

especialmente pensadas, se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

4.3.2.1 El Manifiesto Ágil

El Manifiesto Ágil comienza enumerando los principales valores del desarrollo ágil. Según el manifiesto se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Si se sigue un buen proceso de desarrollo, pero el equipo falla, el éxito no está asegurado; sin embargo, si el equipo funciona, es más fácil conseguir el objetivo final, aunque no se tenga un proceso bien definido. No se necesitan desarrolladores brillantes, sino desarrolladores que se adapten bien al trabajo en equipo. Así mismo, las herramientas (compiladores, depuradores, control de versiones, etc.) son importantes para mejorar el rendimiento del equipo, pero el disponer más recursos que los estrictamente necesarios también pueden afectar negativamente. En resumen, es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

- Desarrollar software que funciona más que conseguir una buena documentación. Aunque se parte de la base de que el software sin documentación es un desastre, la regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental. Si una vez iniciado el proyecto, un nuevo miembro se incorpora al equipo de desarrollo, se considera que los dos elementos que más le van a servir para ponerse al día son: el propio código y la interacción con el equipo.

- La colaboración con el cliente más que la negociación de un contrato. Las características particulares del desarrollo de software hacen que muchos proyectos hayan fracasado por intentar cumplir unos plazos y unos costos preestablecidos al inicio del mismo, según los requisitos que el cliente manifestaba en ese momento. Por ello, se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta puesto que hay muchas variables en juego, debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para unos pocos meses.

Los valores anteriores inspiran los doce principios del manifiesto. Estos principios son las características que diferencian un proceso ágil de uno tradicional. Los dos primeros son generales y resumen gran parte del espíritu ágil. Son:

- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor. Un proceso es ágil si a las pocas semanas de empezar ya entrega software que funcione aunque sea rudimentario. El cliente decide si pone en marcha dicho software con la funcionalidad que ahora le proporciona o simplemente lo revisa e informa de posibles cambios a realizar.

- Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva. Este principio es una actitud que deben adoptar los miembros del equipo de desarrollo. Los cambios en los requisitos deben verse como algo positivo. Les va a permitir aprender más, a la vez que logran una mayor satisfacción del cliente. Este principio implica además que la estructura del software debe ser flexible para poder incorporar los cambios sin demasiados costos adicionales. El paradigma orientado a objetos puede ayudar a conseguir esta flexibilidad.

Luego existen una serie de principios que tienen que ver directamente con el proceso de desarrollo de software a seguir.

- Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas. Las entregas al cliente se insiste en que sean software, no planificaciones, ni documentación de análisis o de diseño.

- La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto. El proceso de desarrollo necesita ser guiado por el cliente, por lo que la interacción con el equipo es muy frecuente.

- Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo. La gente es el principal factor de éxito, todo lo demás (proceso, entorno, gestión, etc.) queda en segundo plano. Si cualquiera de ellos tiene un efecto negativo sobre los individuos debe ser cambiado.

- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo. Los miembros de equipo deben hablar entre ellos, éste es el principal modo de comunicación. Se pueden crear documentos pero no todo estará en ellos, no es lo que el equipo espera.

- El software que funciona es la medida principal de progreso. El estado de un proyecto no viene dado por la documentación generada o la fase en la que se encuentre, sino por el código generado y en funcionamiento. Por ejemplo, un proyecto se encuentra al 50% si el 50% de los requisitos ya están en funcionamiento.

- Los procesos ágiles promueven un desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante. No se trata de desarrollar lo más rápido posible, sino de mantener el ritmo de desarrollo durante toda la duración del proyecto, asegurando en todo momento que la calidad de lo producido es máxima.

Finalmente los últimos principios están más directamente relacionados con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

- La atención continua a la calidad técnica y al buen diseño mejora la agilidad. Producir código claro y robusto, es la clave para avanzar más rápidamente en el proyecto.

- La simplicidad es esencial. Tomar los caminos más simples que sean consistentes con los objetivos establecidos. Si el código producido es simple y de alta calidad será más sencillo adaptarlo a los cambios que puedan surgir.

- Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos. Todo el equipo es informado de las responsabilidades y éstas recaen sobre todos sus miembros. Es el propio equipo el que decide la mejor forma de organizarse, de acuerdo a los objetivos que se persigan.

- En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. Puesto que el entorno está cambiando continuamente, el equipo también debe ajustarse al nuevo escenario de forma continua. Puede cambiar su organización, sus reglas, sus convenciones, sus relaciones, etc., para seguir siendo ágil.

4.3.2.2 eXtreme Programming (XP)

XP (Programación Extrema, por sus siglas en español) 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 realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y valentía 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.

XP se basa en la simplicidad, la comunicación y el reciclado continuo de código, para algunos no es más que aplicar una pura lógica. Los principios y prácticas son de sentidos comunes pero llevados al extremo, de ahí proviene su nombre.

Origen

XP surgió gracias a Kent Beck. Kent, anteriormente a la creación de la XP, había trabajado, principalmente, con Smalltalk, el primer lenguaje orientado a objetos que realmente se hizo popular. Siendo un amante de la orientación a objetos, Kent trabajó junto con Ward Cunnigham intentando encontrar un acercamiento al desarrollo de software que permitiese hacer las cosas más simples y de forma más eficiente.

A raíz de sus investigaciones, Kent entró a formar parte en un proyecto de DaimlerChrysler, que se denominó C3 (Chrysler Comprehensive Compensation), dónde, debido a las necesidades del propio proyecto, se vió obligado a aplicar sus investigaciones y a ponerlas en práctica, surgiendo lo que denominó, Extreme Programming.

El denominar a esta metodología Extreme Programming surge por el énfasis de tener unas buenas prácticas y seguirlas constantemente, pero seguirlas de una forma extrema, es decir, los principios de la XP deben ser seguidos sin alejarse de ellos ni un ápice, y cualquier otra práctica será ignorada.

Ciclo de vida

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requisitos al comienzo de un proyecto.

Por esto, se trata de realizar ciclos de desarrollo cortos, llamados iteraciones, con entregables funcionales al finalizar cada ciclo. Típicamente un proyecto con XP lleva de 10 a 15 ciclos o iteraciones.

Aunque el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases.

- Fase de exploración
  Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de simples historias de usuarios. Los programadores estiman los tiempos de desarrollo con base en esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias, ya que estarán basadas en datos de muy alto nivel, y podrían variar cuando se analicen más en detalle en cada iteración. Esta fase dura típicamente un par de semanas, y el resultado es una visión general del sistema, y un plazo total estimado.

- Fase de planificación
  La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas, o Release Plan.

- Fase de iteraciones
  Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una, un entregable funcional que implementa las historias de usuario asignadas a la iteración. Como las historias de usuario no tienen suficiente detalle como para permitir su análisis y desarrollo, al principio de cada iteración se realizan las tareas necesarias de análisis, recabando con el cliente todos los datos que sean necesarios. El cliente, por lo tanto, también debe participar activamente durante esta fase del ciclo. Las iteraciones son

también utilizadas para medir el progreso del proyecto. Una iteración terminada sin errores es una medida clara de avance.

- Fase de puesta en producción
  Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa. En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste.

Buenas prácticas

La principal suposición que se realiza en XP es la posibilidad de disminuir el costo del cambio, a lo largo del proyecto, de tal forma que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles que facilitan el desarrollo de software, y a la aplicación disciplinada de las siguientes prácticas:

- El juego de la planificación
  Hay una comunicación frecuente entre el cliente y los desarrolladores. El equipo técnico realiza la estimación del esfuerzo para la implementación de las historias de usuario, y los clientes deciden sobre el alcance y tiempo de las entregas y de cada iteración.

- Entregas pequeñas
  Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 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. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

- Refactorización (Refactoring)
  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 posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- Programación en parejas
  Toda la producción de código debe realizarse en parejas de desarrolladores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, …).

- Propiedad colectiva del código
  Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

- Integración continua
  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.

- Cliente in-situ
  El cliente tiene que estar presente y disponible, todo el tiempo, para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente 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.

- Estándares de programación
  XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

4.3.2.3 Scrum

Scrum es una metodología de desarrollo muy simple, que necesita de trabajo duro, porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.

Scrum es una metodología ágil, y como tal:

- Es un modo de desarrollo adaptable, antes que predictivo
- Orientado a las personas, más que a los procesos
- Emplea el modelo de construcción incremental basado en iteraciones y revisiones

Comparte los principios estructurales del desarrollo ágil, es decir, a partir del concepto o visión de la necesidad del cliente, construye el producto de forma incremental a través de iteraciones breves que comprenden fases de especulación, exploración y revisión. Estas iteraciones, que en Scrum se llaman Sprints, se repiten de forma continua hasta que el cliente acepta el producto.

Las iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución en cortas reuniones diarias, donde todo el equipo revisa el trabajo realizado el día anterior y el previsto para el siguiente.

Origen

Scrum es un marco para la ejecución de prácticas ágiles en el desarrollo de proyectos, que toma su nombre y principios de las observaciones sobre nuevas prácticas de producción, realizadas por Hirotaka Takeuchi e Ikujijo Nonaka a mediados de los 80.

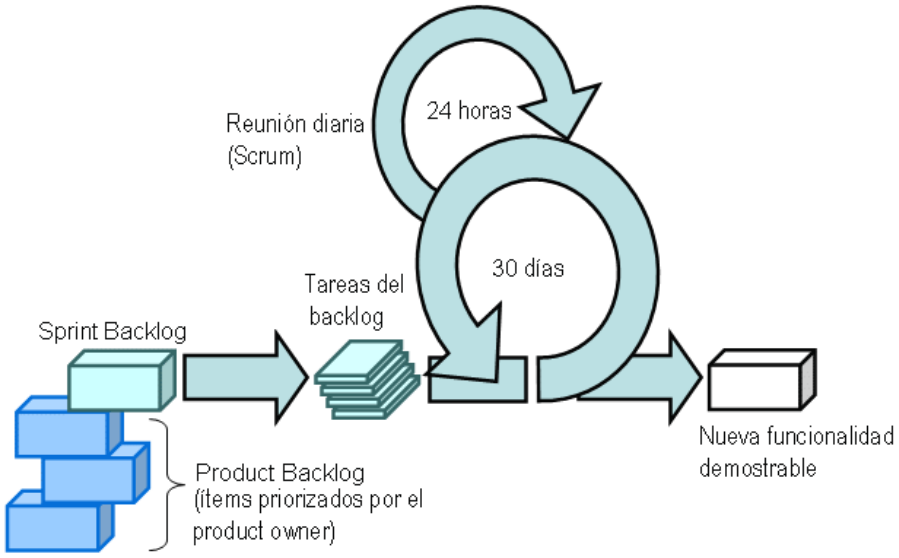
Aunque las prácticas observadas por estos autores surgieron en empresas de productos tecnológicos, también se emplean en entornos que trabajan con requisitos inestables y que necesitan rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados proyectos de software.

Jeff Sutherland aplicó el modelo Scrum al desarrollo de software en 1993, en Easel Corporation. En 1995 lo presentó junto con Ken Schwaber como proceso formal, también para gestión del desarrollo de software en OOPSLA (conferencia Object-Oriented Programming Systems & Applications). Más tarde, en 2001 ellos serían dos de los promulgadores del Manifiesto Ágil. En el desarrollo de software Scrum está considerado como modelo ágil por la Agile Alliance.

Ciclo de vida

Scrum define como Sprint a cada iteración de desarrollo, y según las características del proyecto y las circunstancias del Sprint puede determinarse una duración desde una hasta dos meses, aunque no suele ser recomendable hacerlos de más de un mes. El Sprint es el núcleo central que proporciona la base de desarrollo iterativo e incremental.

Figura 10. Proceso Scrum



Los elementos que conforman la metodología Scrum son:

- Las reuniones
    - Planificación del Sprint: Jornada de trabajo previa al inicio de cada Sprint, en la que se determina cuál va a ser el trabajo y los objetivos que se deben cumplir en esa iteración.
    - Seguimiento del Sprint: Breve revisión del equipo del trabajo realizado hasta el día anterior, lo previsto para realizar, y los problemas identificados.
    - Revisión del Sprint: Análisis y revisión del incremento generado.

- Los artefactos

    - Pila del producto (Producto backlog): Lista de requisitos de usuario que se origina con la visión inicial del producto, y va creciendo y evolucionando durante el desarrollo.

    - Pila del Sprint (Sprint backlog): Lista de los trabajos que debe realizar el equipo durante el Sprint, para generar el incremento previsto.

    - Incremento: Resultados de cada Sprint.

Buenas prácticas

- Revisión de las iteraciones
  Al finalizar cada Sprint se lleva a cabo una revisión con todas las personas implicadas en el proyecto.

- Desarrollo incremental
  Las personas implicadas no trabajan con diseños o abstracciones. El desarrollo incremental implica que al final de cada iteración se dispone de una parte de producto operativa, que se puede inspeccionar y evaluar.

- Desarrollo evolutivo
  Scrum considera a la inestabilidad como una premisa, y se adoptan técnicas de trabajo para permitir la evolución, sin degradar la calidad de la arquitectura que también evoluciona durante el desarrollo. Durante el desarrollo se genera el diseño y la arquitectura final de forma evolutiva. Scrum no los considera como productos que deban realizarse en la primera "fase" del proyecto. El desarrollo ágil no es un desarrollo en fases.

- Auto-organización
  En la ejecución de un proyecto son muchos los factores impredecibles en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al director de proyecto. En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

- Colaboración
  Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. La auto-organización funciona como un control eficaz, si cada miembro del equipo colabora de forma abierta con los demás, según sus capacidades, y no según su rol o su puesto.

4.4 EPF COMPOSER

El editor Eclipse Process Framework Composer, en adelante EPF Composer, es una herramienta gratuita, desarrollada por la comunidad Eclipse, que sirve para crear y modificar contenido de métodos, procesos o metodologías, y generar automáticamente documentación adecuada en formato para la web.

Por lo general, existen dos problemas clave que deben abordarse para implementar un proceso

- En primer lugar, los desarrolladores necesitan entender los métodos y las prácticas más importantes de desarrollo de software.

- En segundo lugar, los equipos de desarrollo necesitan definir cómo aplicar sus métodos de desarrollo y sus mejores prácticas, a lo largo del ciclo de vida de desarrollo

EPF Composer tiene dos propósitos principales:

- Primero, proporcionar una base de conocimiento que se pueda consultar, gestionar y desplegar.

  EPF Composer está diseñado para ser un sistema de gestión y publicación de contenidos, que proporciona una estructura de administración y presentación común para todo el contenido, es decir, es mucho más que un sistema de gestión documental en el que se almacenan y acceden los documentos con sus diferentes formas y formatos.

  Todo el contenido que se maneja en EPF Composer puede ser publicado como un sitio web con métodos, guías, y procesos; y a su vez ser desplegado en servidores Web, permitiendo ser utilizado por diferentes equipos.

  El contenido puede incluir elementos desarrollados como documentos técnicos, guías, plantillas, principios, mejores prácticas, procedimientos y reglamentos internas, material de capacitación, y cualquier otra descripción general de los métodos.

- Segundo, proporcionar capacidades de ingeniería de procesos, que permita a los arquitectos, ingenieros de proceso, y directores de proyectos, seleccionar, ajustar, y rápidamente ensamblar procesos para sus proyectos.

  EPF Composer ofrece catálogos de procesos predefinidos de proyectos típicos, que pueden ser adaptados a las necesidades particulares. También proporciona la construcción de bloques de proceso, llamados patrones de capacidad, que representan las mejores prácticas de desarrollo para disciplinas, tecnologías, o estilos de gestión. Estos bloques forman un conjunto de herramientas que permiten el ensamble rápido de procesos basados en necesidades específicas de los proyectos. EPF Composer también permite la creación de librerías de patrones de capacidad. Adicionalmente, los procesos que se crean a través de EPF Composer, se pueden publicar y desplegar como sitios Web.

El proyecto Eclipse Process Framework tiene como objetivo proporcionar soluciones a problemas comunes que los equipos de desarrollo enfrentan en la creación y gestión de sus métodos y procesos. Por ejemplo:

- Los equipos de desarrollo necesitan un acceso fácil y centralizado a la información: varias organizaciones de desarrollo no mantienen bases de datos centralizadas para sus prácticas y procesos. Por lo general, los procesos no está documentados en absoluto, o están físicamente distribuidos en diferentes formatos de presentación. Los procesos necesitan desplegarse y accederse en las instalaciones donde se ejecutan los proyectos, para proporcionar la documentación del proceso, mientras que el trabajo se está realizando.

- Es difícil integrar procesos de desarrollo que se implementan en un formato propietario: cada libro y publicación presenta el contenido de método y el proceso utilizando un formato diferente. La falta de estándares ampliamente adoptados, y conceptos claramente definidos, para representar el contenido y los procesos, hacen que sea difícil integrar los procesos de diferentes proveedores.

- Los equipos carecen de una base de conocimientos actualizada para capacitarse en métodos y mejores prácticas: antes de utilizar los procesos de desarrollo, los equipos deben ser entrenados.

- Los equipos necesita dimensionar correctamente sus procesos: los procesos deben adaptarse no sólo para un proyecto en particular, sino también continuamente durante todo el ciclo de vida del proyecto. Por lo tanto, los procesos se deben ampliar o reducir de acuerdo a las necesidades de la organización, proyecto, o una fase específica de un proyecto; con la propiedad de ensamblar fácilmente procesos nuevos o existentes.

EPF Composer ofrece los siguientes puntos clave:

- Creación de procesos con editores de estructura de desglose y diagramas de flujo de trabajo, utilizando editores de proceso de múltiple presentación, diferentes vistas del proceso, y sincronización de las vistas con los cambios que se realicen en los procesos.

- Reutilización de patrones de mejores prácticas, para un ágil ensamble de procesos con la facilidad de arrastrar y soltar.

- Seleccionar, combinar, adaptar y rápidamente ensamblar configuraciones de proceso a partir de contenidos de método, para los proyectos de desarrollo de una organización

- Una estructura de gestión y apariencia común, para el contenido de una organización

- Creación de métodos y procesos con contenidos enriquecidos, como texto, imágenes y multimedia, manteniendo la independencia de la arquitectura del proceso

- Administración del contenido utilizando interfaces de usuario simples y editores de texto enriquecido para la creación de descripciones del contenido

- Múltiples vistas del contenido por medio de representaciones de texto y gráficas, y de editores que permiten el uso de estilos, imágenes, tablas, hipervínculos, y edición de HTML.

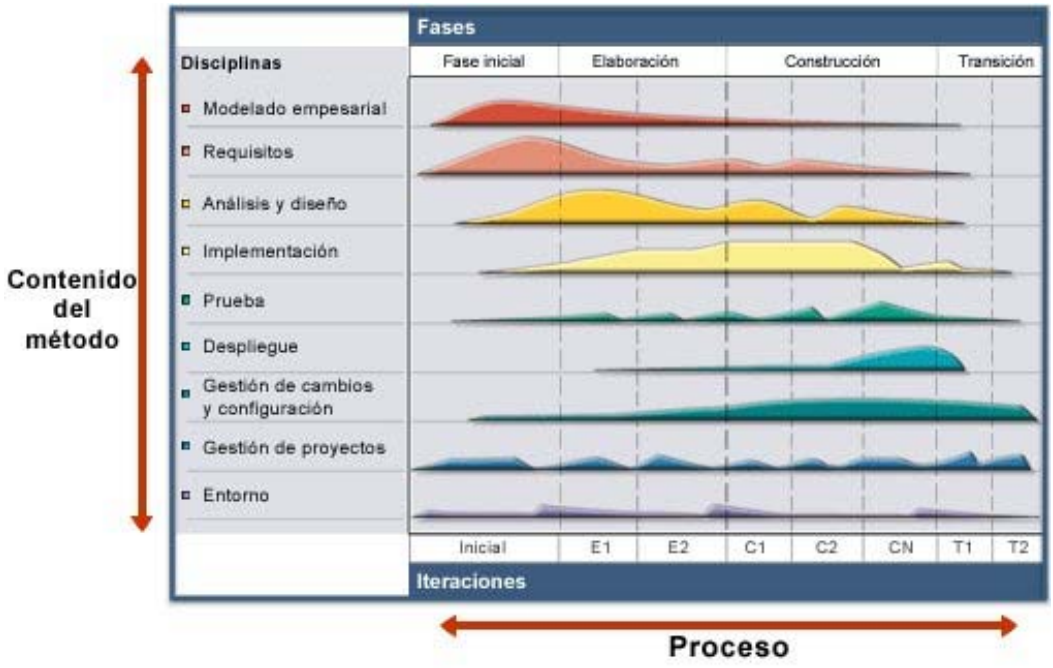
- Publicación de configuraciones de proceso utilizando diferentes formatos, como HTML y PDF

Terminología y conceptos

Para un trabajo eficiente con el EPF Composer, es necesario entender algunos conceptos que se utilizan para organizar el contenido.

El principio más importante en EPF Composer es la separación, en dos dimensiones diferentes, del contenido del método y su aplicación en procesos. El contenido (Method Content) define lo que se va a producir, las habilidades que se necesitan, y las explicaciones paso a paso que describen como alcanzar los objetivos específicos de desarrollo. Las descripciones del contenido son independientes del ciclo de vida desarrollo. Los procesos (Processes) describen el ciclo de vida de desarrollo. Los procesos toman los elementos de contenido y los relacionan en secuencias ordenadas que se adaptan a diferentes tipos de proyecto.

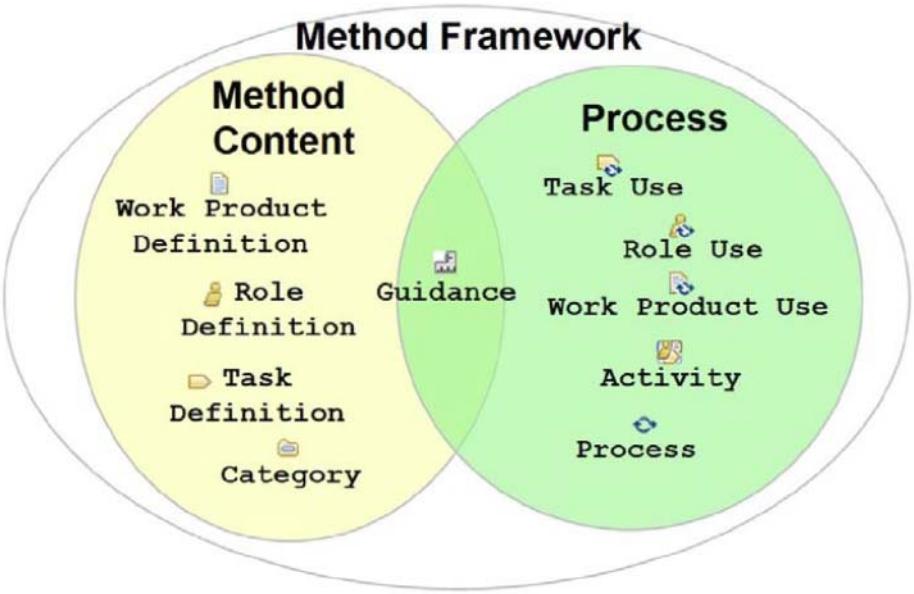
Figura 11. Contenido del método vs Proceso (RUP)



En la Figura 11 se muestra un ejemplo de cómo esta separación se representa en la metodología RUP (Proceso Unificado de Rational, por sus siglas en español). El contenido del método (Method Content) se clasifica en disciplinas, a lo largo del eje Y. El proceso (Process) representa el tiempo, a lo largo del eje X. Este es el ciclo de vida de un proyecto de desarrollo. Aquí se ilustra cuando el trabajo será realizado.

EPF Composer permite que el contenido sea estructurarlo de una forma específica, por medio de un esquema predefinido. Este esquema es una evolución de la especificación SPEM (Software and Systems Process Engineering Metamodel) versión 2.0, anteriormente UMA (Unified Method Architecture). Los contenidos de método y los procesos no se limitan a la Ingeniería de Software, también pueden abarcar otras disciplinas de diseño e ingeniería, tales como la ingeniería mecánica, transformación del negocio, ciclos de ventas, etc.

Figura 12. EPF Method Framework



La Figura 12 ofrece un resumen de los elementos clave que se utilizan en el EPF Composer, y sus relaciones con los procesos o el contenido de métodos. Como se puede apreciar, el contenido (Method Content) es enunciado principalmente a través de productos de trabajo, roles, tareas, y guías. Las categorías son necesarias para la publicación de las metodologías de proceso en su sitio web. Las guías, como listas de verificación, ejemplos, o planes de trabajo, también se pueden definir para proporcionar tutoriales de un proceso.

En el lado derecho de la Figura 12, se muestran los elementos utilizados para representar los procesos (Process) en EPF Composer. El principal elemento de proceso es la actividad, la cual puede ser anidada para definir estructuras de desglose de trabajo, así relacionarse entre sí para definir un flujo de trabajo. Las actividades también contienen descriptores que hacen referencia al contenido. En

EPF Composer se utilizan las actividades para definir procesos, y éstos se clasifican en procesos de entrega y patrones de capacidad.

Los procesos de entrega (Delivery processes) representa una plantilla de proceso completa e integrada, para la realización de un tipo específico de proyecto. Un proceso de entrega describe un ciclo de vida de proyecto completo, de principio a fin, y puede ser utilizado como referencia para la ejecución de proyectos con características similares.

Los patrones de capacidad (Capability patterns) son componentes que expresan y comunican el proceso de un área clave, como una disciplina o una práctica. Se utilizan como bloques de construcción para formar procesos de entrega o patrones de capacidad más grandes. Esto permite asegurar una óptima reutilización y aplicación de las principales mejores prácticas en la creación de procesos, en EPF Composer.

5. RESULTADOS

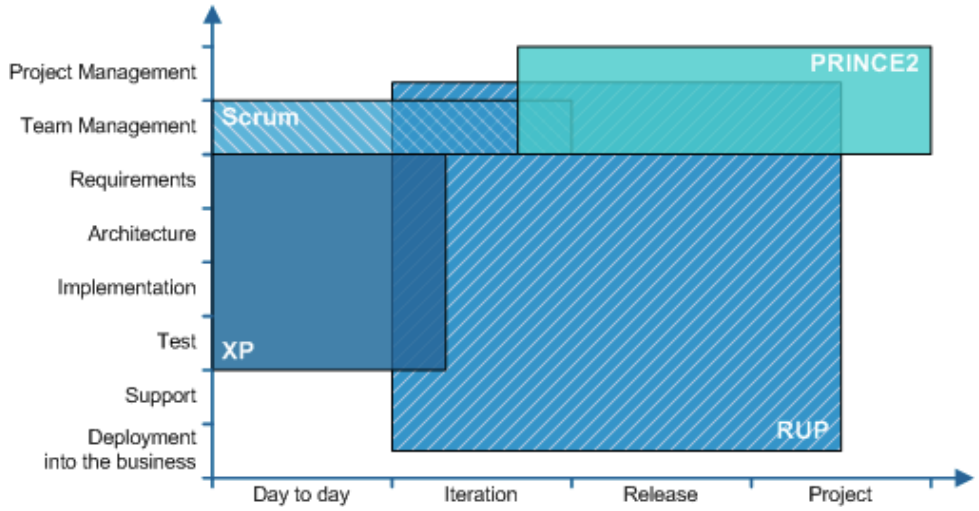
Para la Universidad EAFIT, una de sus principales responsabilidades en los programas, donde la ingeniería de software es un tópico de gran relevancia, debe ser preparar a los estudiantes para trabajar en la industria del software. Esta responsabilidad debe verse reflejada, asegurando que los objetivos y resultados de los programas, se correspondan con la estructura y contenido de los mismos. Los proyectos académicos de desarrollo de software son el punto más crítico en el cumplimiento de esta responsabilidad, al tratar de hacer una comparación, o al menos una analogía, entre la realidad del desarrollo de software en la industria con la realidad actual en la universidad.

Los proyectos académicos de desarrollo de software deben proporcionar a los estudiantes la oportunidad de poner en práctica el conocimiento y las habilidades adquiridas en cursos anteriores, en contextos de problemas de diferente alcance, desde la concepción hasta la implementación de la solución. Los escenarios para estas experiencias deben emular a la realidad de la industria tan cerca como sea posible, utilizando clientes reales (docentes, líderes de investigación, directores de proyecto), procesos, herramientas y criterios de tiempo y calidad. Los escenarios también deben reflejar la realidad de las responsabilidades que se asignan tanto a una persona como a un equipo de trabajo, el compromiso entre el tiempo y la calidad, y la necesidad de adaptarse y hacer frente a los imprevistos. El aprendizaje al practicar y la toma de decisiones críticas, en entornos con algo de incertidumbre, también deben formar parte de esa realidad.

El marco del trabajo EAFITUP propone una metodología para orientar el desarrollo de software en proyectos académicos de diferente alcance. La metodología propuesta es una adaptación de metodologías ampliamente utilizadas y probadas en las organizaciones que conforman la industria nacional de software, con una inclinación pronunciada hacia la metodología RUP (Proceso Unificado de Rational, por sus siglas en español).

Esta inclinación hacia la metodología RUP, fue propuesta por el Ingeniero Jorge Abad Londoño, en su trabajo de tesis para la Maestría en Ingeniería, y aquí se presentan razones suficientes para esta decisión. De acuerdo a la figura 13 (eje X, rangos de tiempo, y eje Y, disciplinas de trabajo), se puede apreciar una comparación entre las metodologías descritas en el marco teórico, y se evidencia que la metodología RUP cubre todas las disciplinas que se pretenden poner en práctica, en la ejecución de proyectos académicos de la Universidad EAFIT.

Figura 13. Enfoque de RUP, XP y Scrum



Otras características valoradas para la selección de RUP como la metodología referente, son:

- RUP es una metodología simple, bien definida, y muy bien documentada. Estas características son fundamentales para EAFITUP, donde los estudiantes con una orientación mínima deben adquirir un conocimiento práctico de los elementos clave del proceso de software y ser productivos en los roles asignados para la ejecución de los proyectos académicos de desarrollo de software.

- RUP promueve la utilización de casos de uso para la descripción de los requisitos de software. Los estudiantes de la Universidad EAFIT están familiarizados con el Lenguaje de Modelado Unificado (UML, por sus siglas en inglés), y los casos de uso, por las asignaturas cursadas en sus programas.

- RUP se enfoca en el desarrollo basado en componentes, como uno de sus mejores prácticas, concepto que se debe enseñar a los estudiantes.

- RUP es un estándar en la industria, que han adaptado diferentes compañías en el medio local y nacional, permitiéndoles ser más eficientes y rentables en la labor de desarrollar software.

- RUP estructura el desarrollo en cuatro fases, cada una ejecutada en una o más iteraciones permitiendo una temprana mitigación del riesgo y durante todo el proceso. El énfasis en la mitigación del riesgo en proyectos

académicos, con equipos de estudiantes desarrollando un producto de software, dentro de un plazo de tres o cuatro meses, es una necesidad.

- RUP es configurable. Esta flexibilidad es fundamental para adaptarse a proyectos de diferente magnitud en el ámbito académico.

De acuerdo a lo anterior, la metodología EAFITUP implementa una personalización de la metodología RUP, e incorpora algunas prácticas de las metodologías OpenUP, XP y Scrum, permitiendo de esta forma establecer una guía para el desarrollo de software en proyectos académicos de diferente magnitud, en la Universidad EAFIT.

5.1 METODOLOGÍA PROPUESTA

En este capítulo se procede a detallar la metodología propuesta, empezando por las mejores prácticas de desarrollo de software que serán implementadas y que formarán la línea base para determinar cómo deben ser abordados los proyectos académicos de desarrollo de software al emplear la presente propuesta metodológica.

5.1.1 Mejores prácticas

La metodología EAFITUP adopta las siguientes mejores prácticas, dirigidas a facilitar el desarrollo de software de proyectos académicos de diversa magnitud, con el fin de que se desarrolle productos de software con alta calidad, y aprovechando al máximo los recursos disponibles de una forma eficaz y eficiente.

- Adaptar la metodología
  EAFITUP es una metodología de desarrollo adaptable, que tiene como objetivo mantener la agilidad durante el proceso de desarrollo, establecer planes realistas, y estimaciones conforme a las condiciones del proyecto y durante todo el ciclo de vida del proyecto. EAFITUP es un marco de trabajo que se presenta como adaptable, y no se descarta que se incluyan componentes externos (contenidos) a los que se presentan, y a su vez tampoco descarta que sus componentes sirvan para otros marcos de trabajo.

- Elevar el nivel de abstracción
  EAFITUP favorece a que se tenga un alto nivel de abstracción para reducir la complejidad y mejorar la comunicación entre los stakeholders de los proyectos, a través de la recomendación de emplear herramientas de modelado de alto nivel como UML, el empleo de estándares abiertos, la definición temprana de la arquitectura, y el desarrollo de componentes para permitir la reutilización.

- Centrarse en la arquitectura
  EAFITUP además de emplear los casos de uso para guiar el proceso de desarrollo, presta especial atención a la definición temprana de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción. La arquitectura de los proyectos es representada a través del modelo de las 4+1 vistas, con el fin de proveer una representación arquitectónica estándar para que todos los stakeholders en el desarrollo la puedan comprender, discutir y analizar.

- Código estándar
  EAFITUP favorece el empleo de código estándar dentro de las tareas de implementación, a fin de favorecer la reducción de la complejidad, la reutilización de componentes y que cualquier stakeholder con los conocimientos pertinentes pueda entender, revisar y opinar sobre cualquier componente implementado.

- Demostrar valor iterativamente
  En EAFITUP las fases se dividen en iteraciones, cuyo resultado es una versión ejecutable. El objetivo de la metodología con cada iteración será mitigar los riesgos de mayor a menor, donde el concepto de riesgo se refiere a ciertos requisitos que son más críticos a la hora de ejecutar el proyecto. Cada iteración debe ser contralada y se debe abordar una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Cada iteración se analiza cuando termina. Se puede determinar si han aparecido nuevos requisitos o han cambiado los existentes, afectando a las iteraciones siguientes.

  La retroalimentación de una iteración finalizada, permite reajustar los objetivos para las siguientes iteraciones. Esta dinámica continua hasta que se haya finalizado por completo el ciclo de vida.

  EAFITUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones, en número variable según el proyecto, y en las que se hace un mayor o menor énfasis de las distintas actividades. Las iteraciones deben estar dirigidas por el riesgo. En cada fase participan todas las

disciplinas, pero dependiendo de la fase, el esfuerzo que se dedica a una disciplina varía.

Si se presentan inconvenientes para terminar una iteración planificada, en lugar de retrasar el final de ésta, se recomienda eliminar algunos de los requisitos, y se dejan para la siguiente iteración.

- Dirigido por casos de uso
  Para EAFITUP los casos de uso son la herramienta estándar empleada para especificar los requisitos funcionales del sistema, son los que guían el análisis y diseño, implementación y pruebas de todo el sistema, y adicionalmente son los elementos que permiten la trazabilidad.

- Enfocarse en la calidad
  EAFITUP contiene mecanismos para que la calidad de todos los artefactos se evalúe en varios puntos durante todo el ciclo de vida, especialmente al final de cada iteración. En la metodología EAFITUP se pueden encontrar tareas enfocadas a revisar y evaluar, las cuales juegan un papel fundamental para asegurar calidad no solo al final de los proyectos, sino que por el contrario durante todo su ciclo de vida.

- Enfocarse en los riesgos
  La gestión de los riesgos es contemplada en EAFITUP desde el inicio del proyecto hasta el final del mismo. El enfoque en los riesgos es el punto de partida para programar las actividades que deben ejecutarse durante las iteraciones.

- Fomentar del aprendizaje
  EAFITUP favorece el aprendizaje de las experiencias obtenidas en cada uno de los proyectos realizados, para obtener más eficacia y eficiencia en futuros proyectos. Esta práctica se fortalece a través de las continuas retroalimentaciones que se dan en diversas actividades entre los stakeholders (estudiantes, docentes, etc.)

- Gestionar los cambios
  El cambio es un factor de riesgo crítico en los proyectos de software, ante los cuales EAFITUP crea las condiciones necesarias, a través de sus actividades, para gestionarlos con un enfoque ágil lo más tempranamente posible con su proceso iterativo e incremental, con la participación continua de los stakeholders y con las actividades de retroalimentación. Los artefactos de software cambian debido a acciones de refinación o maduración durante el proceso de desarrollo.

EAFITUP asume que las cosas pueden cambiar y que ningún proyecto está aislado del impacto de estos cambios. Para encarar de manera eficientemente cualquier cambio que se presente, el equipo de proyecto debe mantenerse ágil para reportar y gestionar los cambios, y todos los stakeholders deben participar de manera activa para obtener diferentes perspectivas de una posible solución.

5.1.2 Estructura del proceso

La metodología EAFITUP propone una estructura como la de RUP, la cual tiene dos dimensiones:

- Eje horizontal

  Representa el tiempo y es considerado el eje de los aspectos dinámicos del proceso. Indica las características del ciclo de vida del proceso, en términos de fases, iteraciones e hitos.

- Eje vertical

  Representa los aspectos estáticos del proceso. Describe el proceso en términos de componentes de proceso, disciplinas, actividades, tareas, artefactos y roles.

5.1.3 Fundamentos de la metodología

EAFITUP establece una estructura que cubre todo el ciclo de vida de desarrollo de software, incluyendo fases, roles, actividades, artefactos, disciplinas, flujos de trabajo, gestión de riesgos, control de calidad, gestión del proyecto y control de configuración. En general, esta metodología tiene su base en los contenidos de la metodología RUP, e incorpora algunos elementos de OpenUP, XP, y Scrum.

Cabe destacar que los elementos en EAFITUP fueron considerados mediante un análisis de varias metodologías, en la que se compararon las mismas respecto a sus elementos, lo cual permitió la selección de elementos que han tenido éxito en la industria de desarrollo de software, así como también de elementos que se adaptan a las necesidades de los proyectos académicos que se ejecutan en la Universidad EAFIT.

Especificando los elementos que fueron identificados, analizados y comparados de cada metodología, se puede decir que las mejores prácticas para el desarrollo de software que se recopilan en EAFITUP, hacen parte de las metodologías RUP, OpenUP y Scrum.

EAFITUP propone una estructura que es una adaptación de la propuesta por metodología RUP, es decir, la conforman aspectos dinámicos y estáticos. Las fases e hitos corresponden los aspectos dinámicos y las disciplinas que corresponde a los aspectos estáticos.

Las metodologías en las que se basa EAFITUP son algunas de las más utilizadas, además de que permiten la adaptación, es decir, son marcos de trabajo que permiten seleccionar elementos según las características de cada proyecto. La selección de esos elementos permitió definir la metodología EAFITUP para el desarrollo de software, con un enfoque de calidad y cumpliendo con las necesidades de los proyectos académicos de la Universidad EAFIT.

5.2 ESTRUCTURA DINÁMICA DE LA METODOLOGÍA

EAFITUP se repite a lo largo de una serie de ciclos que constituyen la vida de un proyecto. Cada ciclo concluye con una generación del producto y consta de cuatro fases. Cada fase se subdivide a la vez en iteraciones, el número de iteraciones en cada fase es variable.

Cada fase concluye con un hito bien definido, un punto en el tiempo en el cual se deben tomar ciertas decisiones críticas y alcanzar la meta clave antes de pasar a la siguiente fase, ese hito principal de cada fase se compone de hitos menores que podrían ser los criterios aplicables a cada iteración. Los hitos son puntos de control en los cuales los involucrados en el proyecto revisan el progreso del proyecto.

El ciclo de vida propuesto por la metodología EAFITUP para un proyecto de software se compone en el tiempo de cuatro fases secuenciales que son: Inicio, Elaboración, Construcción y Transición. Al final de cada fase el director de proyecto realiza una evaluación para determinar si los objetivos se cumplieron y así pasar a la fase siguiente.

5.2.1 Inicio

El propósito general es establecer los objetivos para el ciclo de vida del producto.

Los objetivos específicos de esta fase son:

- Establecer el ámbito del proyecto y sus límites.
- Encontrar los casos de uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
- Identificar al menos una arquitectura candidata para los escenarios principales.
- Estimar el costo en recursos y tiempo de todo el proyecto.
- Identificar riesgos, y las fuentes de incertidumbre.

El hito en esta fase finaliza con el establecimiento de la viabilidad del proyecto, el acuerdo del alcance del producto, e identificación de los principales riesgos.

5.2.2 Elaboración

El propósito general es plantear la arquitectura para el ciclo de vida del producto. Se construye un modelo de la arquitectura, que se desarrolla en iteraciones sucesivas hasta obtener el producto final, este prototipo debe contener los casos de uso críticos que fueron identificados en la fase de inicio. En esta fase se realiza la recopilación de la mayor parte de los requisitos funcionales, y se gestionan los riesgos que interfieran con los objetivos del sistema

Los objetivos específicos de esta fase son:

- Madurar, validar y establecer la arquitectura.
- Demostrar que la arquitectura propuesta soportara la visión, con un costo y tiempo razonables.
- Refinar la estimación del costo de recursos y tiempo del proyecto
- Mitigar los riesgos identificados.

El hito en esta fase finaliza con la obtención de una línea base de la arquitectura del sistema, la recopilación de la mayoría de los requisitos y la mitigación de los riesgos importantes.

5.2.3 Construcción

El propósito general es alcanzar la capacidad operacional del producto de forma incremental a través de iteraciones sucesivas. En esta fase todas las características, componentes, y requisitos deben ser integrados, implementados, y probados en su totalidad, obteniendo una versión aceptable del producto; comúnmente llamada versión beta.

Los objetivos específicos de esta fase son:

- Desarrollar servicios, casos de uso (funcionalidades), módulos, y/o subsistemas del proyecto.
- Realizar desarrollo por iteraciones, priorizando los servicios, casos de usos, módulos y/o subsistemas más críticos para el negocio.
- Realizar pruebas funcionales de cada uno de los servicios, casos de usos, módulos y/o subsistemas a medida que se vayan desarrollando.

El hito en esta fase finaliza con el desarrollo del sistema con calidad de producción y la preparación para la entrega al equipo de transición. Toda la funcionalidad debe haber sido implementada y las pruebas para el estado beta de la aplicación completadas. Si el proyecto no cumple con estos criterios de cierre, entonces la transición deberá posponerse una iteración.

5.2.4 Transición

El propósito general es entregar el producto funcional en manos de los usuarios finales (docente, comunidad, grupo de investigación), con toda la documentación donde se especifique la instalación, configuracion, y usabilidad del producto.

Los objetivos específicos de esta fase son:

- Conseguir un producto final que cumpla los requisitos definidos.
- Garantizar que el usuario está en capacidad de instalar y operar el sistema.

El hito en esta fase se alcanza cuando el cliente revisa y acepta los artefactos que le han sido entregado.

5.3 ESTRUCTURA ESTÁTICA DE LA METODOLOGÍA

Una metodología de desarrollo de software define quién hace qué, cómo y cuándo.

La metodología EAFITUP define cuatro elementos: los roles, que responden a la pregunta ¿Quién?; las actividades, que responden a la pregunta ¿Cómo?; los artefactos, que responden a la pregunta ¿Qué?; y los flujos de trabajo de las disciplinas que responden a la pregunta ¿Cuándo?

5.3.1 Disciplinas

La metodología EAFITUP se organiza en disciplinas. Las disciplinas están compuestas por un conjunto de tareas, las cuales tienen como objetivo producir artefactos.

Las disciplinas que conforman esta metodología se dividirán en dos grupos. El primero comprende las disciplinas fundamentales asociadas con las áreas de ingeniería:

- Modelado empresarial
- Requisitos
- Análisis y Diseño
- Implementación
- Pruebas
- Despliegue

El segundo grupo lo integran las disciplinas llamadas de soporte o gestión:

- Gestión del proyecto
- Gestión de cambios

La descripción de cada una de las disciplinas puede consultarse en la publicación de la metodología, la cual es un sitio Web estático y es parte integral de este proyecto de grado.

5.3.2 Artefactos

La metodología EAFITUP propone treinta y dos (32) artefactos que pueden ser generados durante el proceso de desarrollo de software.

Es fundamental que antes del comienzo del proceso de desarrollo se decida cuales son los artefactos que serán empleados a lo largo del ciclo de vida del desarrollo del proyecto, así como es recomendable que se defina entre las partes los artefactos que deben ser entregados.

Es recomendable que se utilicen la mayoría de los artefactos que se definen en la metodología, principalmente si la magnitud del proyecto es grande. Mientras más artefactos detallen en profundidad los elementos de un sistema, mejor será la compresión del proyecto, por parte del equipo de trabajo.

La lista de los artefactos que se definen en EAFITUP, se presentan agrupados por la disciplina a la que pertenecen:

- Disciplina – Modelado empresarial
    - Modelo de Procesos de Negocio

- Disciplina – Requisitos
    - Solicitudes de los stakeholders
    - Historias de usuario
    - Visión del sistema
    - Glosario
    - Documento de Requisitos del Sistema (DRS)
    - Diagrama de contexto
    - Modelo de Casos de Uso
    - Caso de Uso

- Disciplina – Análisis y Diseño
    - Prototipo de interfaz de usuario
    - Documento de Arquitectura de Software (DAS)

- Disciplina – Implementación
    - Prueba Unitaria
    - Componente
    - Registro de Pruebas Unitarias
    - Componente Operacional (Build)

- Disciplina – Pruebas
    - Lista de ideas de prueba
    - Caso de prueba
    - Script de pruebas
    - Registro de pruebas

- Disciplina – Despliegue
    - Manual de instalación
    - Manual de usuario

- Disciplina – Gestión del proyecto
    - Plan del proyecto
    - Plan de iteración
    - Lista de riesgos
    - Lista de elementos de trabajo
    - Gráfica del trabajo pendiente (Burndown chart)
    - Informe de avance
    - Registro de revisión
    - Sistema
    - Acta de entrega del proyecto

- Disciplina – Gestión de cambios
    - Solicitud de cambio (CR)
    - Repositorio de versiones

La descripción de cada uno de los artefactos puede consultarse en la publicación de la metodología, la cual es un sitio Web estático y es parte integral de este proyecto de grado.

5.3.3 Roles

Una de las razones principales de la adopción de esta metodología para el desarrollo de software consiste en la definición de las tareas que serán llevadas a cabo por los estudiantes que participan en un proyecto. En EAFITUP un rol define las responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Los roles se encargan de la realización de tareas, las cuales generan artefactos.

La metodología EAFITUP propone ocho (8) roles básicos:

- Director de proyecto
- Analista
- Arquitecto
- Desarrollador
- Tester
- Stakeholder
- Cualquier rol
- Docente

La descripción cada uno de los roles puede consultarse en la publicación de la metodología, la cual es un sitio Web estático y es parte integral de este proyecto de grado.

5.4 NOTAS DE LA METODOLOGÍA

Actualmente la metodología EAFITUP implementa tres ciclos de vida de desarrollo, asociados a los tipos de proyectos académicos de desarrollo de software, que se han identificado en la Universidad EAFIT.

- Proceso Tipo 1 (Básico)
- Proceso Tipo 2 (Medio
- Proceso Tipo 3 (Completo)

La descripción del ciclo de vida para cada tipo de proyecto puede consultarse en la publicación de la metodología, la cual es un sitio Web estático y es parte integral de este proyecto de grado.

La herramienta EPF Composer permitió la creación, configuración y publicación de la metodología EAFITUP. Esta publicación es un sitio Web estático, que incluye todo el contenido y procesos de la metodología, y es parte integral de este proyecto de grado. El editor EPF Composer facilita la edición del contenido y los procesos definidos, permitiendo realizar actualizaciones de cualquier tipo en la metodología, por ejemplo, crear un nuevo artefacto, un nuevo rol, o una nueva tarea; modificar un contenido o todo un proceso, etc.

Una consideración importante, EAFITUP no enseñará a los estudiantes cómo convertirse en buenos desarrolladores. EAFITUP guiará a los estudiantes en qué hacer, y es probable que en futuras versiones, hasta cómo hacer muchas cosas, pero es requisito previo tener conocimientos básicos en desarrollo de software.

6. CONCLUSIONES

- EAFITUP es un marco de trabajo que establece una metodología unificada para ejecutar los proyectos académicos de desarrollo de software, que los estudiantes afrontan en las asignaturas que pertenecen al área de ingeniería de software, en la Universidad EAFIT. Estas asignaturas corresponden tanto a programas de pregrado, como de posgrado.

- EAFITUP implementa una metodología de desarrollo de software haciendo una adaptación de la metodología de desarrollo RUP.

- EAFITUP es una metodología adaptable a las necesidades del proyecto, prueba de esto, es que actualmente dentro del marco de trabajo se han definido tres procesos que permiten guiar las actividades de desarrollo de software en proyectos académicos de diferente alcance.

- EAFITUP es una metodología que genera valor iterativamente, está dirigida por casos de uso, se centra en la definición de una arquitectura, y el desarrollo se basa en componentes.

- El ciclo de vida de un proceso de desarrollo en EAFITUP consiste de cuatro fases (inicio, elaboración, construcción y transición), donde cada fase puede incorporar una o varias iteraciones, donde se realizan actividades de aseguramiento de calidad y mitigación de riesgos, de manera continua.

- Existe una marcada diferencia en el conocimiento de la capacidad de adaptación de la metodología de desarrollo RUP para la ejecución de proyectos de software de cualquier tamaño; mientras en el ámbito académico se desconoce o se cataloga a RUP como una metodología pesada, en el ámbito de la industria local y nacional se conoce y valora esta propiedad, lo que ha permitido a muchas compañías (por no decir la mayoría) a definir sus procesos de desarrollo de software como una adaptación de la metodología de desarrollo RUP.

- Al utilizar la especificación SPEM versión 2.0 (estándar para la definición de procesos de desarrollo de sistemas y de software), a través de la herramienta EPC Composer, para la definición de la metodología propuesta; se dan las

- EPF Composer permite generar un sitio Web de la metodología de desarrollo EAFITUP, con todas las descripciones de los elementos que la componen; permitiendo que la metodología sea material de consulta permanente por los equipos de desarrollo que ejecuten los proyectos académicos de software.

- Es necesario establecer programas de capacitación y entrenamiento para los usuarios (estudiantes, docentes, grupos de investigación, etc.) de la nueva propuesta metodológica, con el propósito de facilitar la adopción de la metodología de desarrollo. Adicionalmente, se deben considerar la creación o modificación de asignaturas que capaciten a los estudiantes en actividades de gestión de proyectos, gestión de riesgos, y gestión de la configuración y el cambio.

- En el momento que EAFITUP se establezca como la metodología unificada y de fácil consulta, para orientar los proyectos académicos de desarrollo de software en la Universidad EAFIT, se deben definir mecanismos para recibir, revisar y valorar la retroalimentación de los usuarios, con el propósito de ajustar y mejorar la metodología. La metodología de desarrollo debe ser un activo que mejore continuamente.

- La utilización de la metodología EAFITUP por parte de los estudiantes, permitirá mejorar la capacidad de adopción y adaptación de un proceso de desarrollo para la ejecución de sus proyectos de software; con el principal propósito de estrechar la brecha existente entre las competencias de los profesionales que egresan y las competencias que demanda la industria.

BIBLIOGRAFÍA

AMBYSOFT. Best practices for software development [en línea]. <http://www.ambysoft.com/>

BERGANDY, Jan. Teaching Software Engineering with Rational Unified Process (RUP). ASEE New England Section 2006 Annual Conference. 2006

BERGSTRÖM, Stefan y RABERG, Lotta. How to adopt RUP in your Project. En: Adopting the rational unified process: Success with the RUP. Addison Wesley, 2003. p. 127-140.

BOOCH, Grady; RUMBAUGH, James y JACOBSON, Ivar. The Unified Modeling Language: User guide. 2 ed. Westford, MA, Addison Wesley, 2005.

CENTRO NACIONAL DE TECNOLOGÍAS DE INFORMACIÓN. Metodología de la red nacional de integración y desarrollo de software libre (MeRinde): Guía detallada.

COCKBURN, Alistair. Writing effective use cases. Addison Wesley, 2001.

ECLIPSE. Introduction to OpenUP (Open Unified Process) [en línea]. <http://www.eclipse.org/epf/general/OpenUP.pdf>

ÉCOLE POLYTECHNIQUE DE MONTRÉAL. Unified Process for Education (UPEDU) [en línea]. <http://www.upedu.org/>

IBM. IBM Rational Method Composer [en línea]. <http://www-01.ibm.com/software/awdtools/rmc/>

IBM. IBM Rational Unified Process (RUP) [en línea]. <http://www-01.ibm.com/software/awdtools/rup/>

IBM. IBM Rational Unified Process (RUP) para proyectos pequeños [en línea]. <http://cgrw01.cgr.go.cr/rup/RUP.es/LargeProjects/index.htm#core.base_rup/guidances/supportingmaterials/welcome_2BC5187F.html>

IBM. Rational Process Library: The industry's most robust collection of best practices guidance [en línea]. <http://www-01.ibm.com/software/awdtools/rmc/library/>

IBM. Using RUP to manage small projects and teams [en línea]. <http://www.ibm.com/developerworks/rational/library/jul05/kohrell/>

INSTITUTO COLOMBIANO DE NORMALIZACIÓN Y CERTIFICACIÓN. Documentación, presentación de tesis, trabajos de grado y otros trabajos de investigación. NTC 1486. 6 ed. Bogotá D.C., ICONTEC, 2008.

INSTITUTO COLOMBIANO DE NORMALIZACIÓN Y CERTIFICACIÓN. Referencias bibliográficas: Contenido, forma, y estructura. NTC 5613. Bogotá D.C., ICONTEC, 2008.

JACOBSON, Ivar; BOOCH, Grady y RUMBAUGH, James. El Proceso Unificado de desarrollo de software. Madrid, Addison Wesley, 2000.

JOSKOWICZ, José. Reglas y prácticas en eXtreme Programming. Universidad de Vigo. 2008.

KROLL, Per; KRUCHTEN, Philippe y BOOCH, Grady. The rational unified process made easy: A practitioner's guide to the RUP. Addison Wesley, 2003.

KRUCHTEN, Philippe. Architectural Blueprints: The 4+1 view model of software architecture. IEEE Software, 1995.

KRUCHTEN, Philippe. The Rational Unified Process: An Introduction. 3 ed. Addison Wesley, 2003.

LARMAN, Craig. Agile and Iterative development: A manager's guide. Addison Wesley, 2004.

LETELIER, Patricio. Proceso de desarrollo de software. Universidad Politécnica de Valencia. 2003.

LETELIER, Patricio y PENADÉS, María Carmen. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). Universidad Politécnica de Valencia. 2003.

MARTÍNEZ, Alejandro y MARTÍNEZ, Raúl. Guía a Rational Unified Process. Albacete, Escuela Politécnica Superior de Albacete, 2000.

PALACIO, Juan y RUATA, Claudia. Scrum Manager: Gestión de proyectos. 2011.

PRESSMAN, Roger. Software Engineering: A Practitioner´s Approach. 5 ed. New York, McGraw-Hill Higher Education, 2000.

PROJECT MANAGEMENT INSTITUTE. PMBoK (Project Management Body of Knowledge), version 4 [en línea]. <http://www.pmi.org/PMBOK-Guide-and-Standards.aspx>

RUMBAUGH, James; JACOBSON, Ivar y BOOCH, Grady. The Unified Modeling Language: Reference Manual. 2 ed. Boston, MA, Addison Wesley, 2004.

SOFTWARE ENGINEERING INSTITUTE. CMMI for development, version 1.3 [en línea]. <http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm>

SOLUCIONES INFORMÁTICAS GLOBALES. CMMI – Capability Maturity Model Integration [en línea]. <http://www.globales.es/imagen/internet/Informaci%C3%B3n%20General%20CMMI.pdf>

TOBARRA, Manuel. CMM y RUP: Una perspectiva común. Universidad de Castilla – La Mancha. 2003

VILLAGRA, Sergio. Una introducción a CMMI. Axentia. 2006