Unidad de Compatibilización, Integración y Desarrollo de Productos Informáticos para la Defensa.

Tesis de maestría

Propuesta de metodología de desarrollo de software para su utilización en la unidad de compatibilización, integración y desarrollo de productos informáticos para la defensa (UCID).

Autor: Ing. Meylin Martinez Chong

Tutor: DrC. Javier Crespo Sánchez

Ciudad de la Habana
Julio 2010

"La vida cobra sentido cuando se hace de ella una aspiración a no renunciar a nada"

José Ortega y Gasset

Agradecimientos

A mi familia por apoyarme en este momento tan importante.
A mi novio Sergio por ayudarme a superar todos los obstáculos.
A mis compañeros de trabajo por su preocupación
A todos muchas gracias

Resumen

El presente trabajo tiene como título “Propuesta de metodología de desarrollo de software para su utilización en la unidad de compatibilización, integración y desarrollo de productos informáticos para la defensa (UCID)”, el centro de esta investigación la constituyen las metodologías de desarrollo de software.
En el ciclo de vida del software se deben completar una serie de **tareas** para obtener un producto de software. Los distintos componentes de software deben pasar por distintas **fases** o **etapas** durante el ciclo de vida, cada una de esas tareas puede ser abordada y resuelta de múltiples maneras: con distintas **herramientas** y utilizando diferentes **técnicas**. Es necesario saber además, cuándo podemos dar por concluida una tarea, **quién** debe realizarla, qué tareas **preceden** o anteceden a una determinada, qué **documentación** utilizaremos para llevarla a cabo. Estamos hablando de detalles organizativos, de un "**estilo**" de hacer las cosas. Pero yendo un poco más allá que un simple estilo, formalizando ese "estilo" añadiendo algo de rigurosidad y normas se obtiene una **metodología**. Todos los integrantes de un equipo de desarrollo deben seguir un criterio común a la hora de realizar las tareas del ciclo de vida. Ese criterio, esa manera común es una metodología de desarrollo.
Actualmente en la UCID se desarrolla software de diferentes tipos: desde aplicaciones web, desktop, frameworks, etc. Con diferentes características y tecnologías pero con un objetivo común: desarrollar software con altos niveles de calidad en función de la defensa de nuestro país.
Teniendo en cuenta el estudio realizado sobre las metodologías de desarrollo de software existentes en la actualidad; y las características y situación de los proyectos que se desarrollan en la UCID, se realiza una propuesta de metodología que nos permita además de concluir con éxito los productos, lograr niveles de calidad en los mismos.
Palabras claves:
Software, metodologías de desarrollo de software, fases, tareas, equipo de desarrollo.

Abstract

The present work has as title “Proposed of software development methodology for its usage in the coordinate, integration and development unit of computer products for the defense”; the center of this investigation constitutes it the software development methodologies.

In the cycle of life of the software they should be completed a series of tasks to obtain a software product. The different software components should go by different phases or stages during the cycle of life, each one of those tasks can be approached and resolved in multiple ways: with different tools and using different technical. It is necessary to also know, when we can give had concluded a task, who should carry out it, what tasks they precede or they precede to a certain one, what documentation we will use to carry out it. We are speaking of organizational details, of a "style" of making the things. But going a little further on that a simple style, formalizing that "style" being but rigorous and following norms a methodology is obtained. All the members of a development team should follow a common approach when carrying out the tasks of the cycle of life. That approach, that common way is a development methodology.

Currently in the UCID software of different types is developed: web applications, desktop, frameworks, etc. With different features and technologies but with a common objective: to develop software with high levels of quality in feature of the defense of our country. Keeping in mind the study carried out on software development methodologies; and the features and status of the projects that are developed in UCID, are carried out a methodology proposal that allow us besides concluding with success the products, to achieve levels of quality in the same ones.

Key word:

Software, software development methodologies, phases, tasks, development team.

Índice

Tabla de contenido

Índice de figuras

Índice de tablas

INTRODUCCIÓN

Según la definición del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE por sus siglas en inglés), "software es la suma total de los programas de ordenador, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo" y "un producto de software es un producto diseñado para un usuario". En este contexto, la Ingeniería de Software (SE del inglés "Software Engineering") es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software.

Durante los primeros años de la informática, el software se consideraba como un añadido. La programación era un "arte", para el que no existía ninguna metodología. Era un proceso que se realizaba sin ninguna planificación. En esta época toda la programación se desarrollaba a medida para cada aplicación, y en consecuencia tenía muy poca difusión, habitualmente quien lo escribía era porque lo necesitaba, y era quien lo mantenía.

En una segunda época (a partir de la mitad de la década de 1960) se estableció el software como producto y aparecieron las empresas dedicadas al desarrollo y distribución masiva del mismo, el programador solitario de antaño se reemplazó por un equipo de especialistas del software, cada uno centrado en una parte de la tecnología requerida para entregar una aplicaci6n concreta [Pressman, 2006].

La tercera era comenzó a mediados de la década de 1970, época en la que los sistemas informáticos aumentaron mucho en su complejidad, y nacieron las redes de ordenadores. Esto supuso mucha presión para los desarrolladores, aunque los ordenadores para uso personal, apenas estaban difundidos. Esta época acabó con la aparición de los microprocesadores.

La cuarta era de la evolución de los sistemas informáticos, comienza hacia 1990 y se dirige al impacto colectivo de los ordenadores y el software, en todos los entornos. La industria del software tiene un gran peso en la economía mundial. Aparecen las técnicas de redes neuronales, junto con la lógica difusa.

Hoy en día el software tiene un doble papel. Es un producto, pero simultáneamente es el vehículo para hacer entrega de un producto [Pressman, 2006]. Como producto permite el uso del hardware, ya sea, por ejemplo, un ordenador personal, un teléfono móvil. Como vehículo utilizado para hacer entrega del producto, actúa como base de control, por ejemplo un sistema operativo, o un sistema gestor de redes. El software hace entrega de lo que se considera como el producto más importante del siglo veintiuno, la información. El software transforma datos personales para que sean más útiles en un entorno local, gestiona información comercial para mejorar la competitividad, proporciona el acceso a redes a nivel mundial, y ofrece el medio de adquirir información en todas sus formas.

El término ingeniería del software comenzó a utilizarse para expresar el área de conocimiento que se estaba desarrollando en torno a las problemáticas que ofrecía el software. El IEEE en [IEEE93] desarrolló una definición completa sobre la Ingeniería del software: La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software: es decir, la aplicación de la ingeniería al software.

La ingeniería de software en la construcción y desarrollo de proyectos aplica métodos y técnicas para resolver los problemas que se presentan, la informática aporta herramientas y procedimientos sobre los que se apoya la ingeniería de software con el fin de mejorar la calidad de los productos de software, aumentar la productividad y trabajo de los ingenieros del software, facilitar el control del proceso de desarrollo del

mismo y suministrar a los desarrolladores las bases para construir software de alta calidad en una forma eficiente. [Álvarez, 2007].
La Ingeniería de Software es un proceso intensivo de conocimiento, que abarca la captura de requerimientos, diseño, desarrollo, prueba, implantación y mantenimiento. Generalmente a partir de un complejo esquema de comunicación en el que interactúan usuarios y desarrolladores, el usuario brinda una concepción de la funcionalidad esperada y el desarrollador especifica esta funcionalidad a partir de esta primera concepción mediante aproximaciones sucesivas. Este ambiente de interacción motiva la búsqueda de estrategias robustas para garantizar que los requisitos del usuario serán descubiertos con precisión y que además serán expresados en una forma correcta y sin ambigüedad, que sea verificable, trazable y modificable.
Para guiar este desarrollo se crearon metodologías, las cuales guían el proceso de desarrollo y definen con precisión cómo organizar las actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de software. Las metodologías y la experiencia han demostrado que la clave del éxito de un proyecto de software es la elección correcta de la metodología, que puede conducir al programador a desarrollar un buen sistema de software. La elección de la guía adecuada es más importante que utilizar las mejores y más potentes herramientas. El objetivo es convertir el desarrollo de software en un proceso formal, con resultados predecibles, que permita obtener un producto final de alta calidad y satisfaga las necesidades y expectativas del cliente.
Muchos ingenieros ante la tarea de investigar y estudiar las metodologías se encuentran en la disyuntiva de por cuál inclinarse debido a la diversidad de propuestas y diferencias en el grado de detalle de la información disponible y alcance de cada una de ellas. Lo cual trae como consecuencia que la comparación y/o clasificación de las mismas para su utilización no sea una tarea sencilla.
A lo largo de los años se han propuesto numerosas metodologías. Algunas han sido escritas por autores del ámbito académico, otras por autores del ámbito empresarial de desarrollo del software y otras por administraciones públicas. Algunas metodologías están escritas en infumables tochos de heroica lectura. Otras, se describen en apenas unas pocas páginas. Algunas metodologías intentan describirlo todo al detalle. Otras son más genéricas. Algunas hacen hincapié en los datos, otras en los usuarios, otras en las tareas, otras en la documentación o cualquier aspecto o combinación de aspectos que puedan darse en el desarrollo del software.
Existen una serie de metodologías llamadas tradicionales propuestas casi todas ellas con anterioridad a los años 90 que pretendían ayudar a los profesionales indicando pautas para realizar y documentar cada una de las tareas del desarrollo del software. Entre las principales metodologías tradicionales tenemos los ya tan conocidos Proceso Unificado Racional (RUP por sus siglas en inglés) y Framework de soluciones de Microsoft (MSF).
Luego de varias opiniones tanto a favor como en contra de las metodologías tradicionales se genera un nuevo enfoque denominado, métodos ágiles, las cuales ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.
Entre los principales métodos ágiles tenemos la Programación extrema (XP por sus siglas en inglés), Scrum, Cristal Methods, Proceso unificado ágil (AUP) entre otras. Como resultado de esta nueva teoría se crea un Manifiesto Ágil cuyas principales ideas son:

- Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados.

- Es más importante crear un producto software que funcione que escribir documentación exhaustiva.
- La colaboración con el cliente debe prevalecer sobre la negociación de contratos.
- La capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan.

Las metodologías ágiles son sin duda uno de los temas recientes en la ingeniería de software que están acaparando gran interés. Prueba de ello es que se está viviendo con intensidad un debate abierto entre los partidarios de las metodologías tradicionales (referidas peyorativamente como "metodologías pesadas") y aquellos que apoyan las ideas emanadas del "Manifiesto Ágil".
La Tabla 1, que se muestra a continuación, recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales ("no ágiles"). Estas diferencias que afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

Tabla 1. Diferencias entre las metodologías ágiles y las tradicionales [Canós, 2003].

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

Aunque ambos tipos de metodologías proponen recorrer las mismas etapas en el desarrollo del software se enfocan en aspectos diferentes durante las mismas, a continuación en la tabla 2 se muestra una comparación entre cada una de las etapas más comunes del desarrollo de software y los enfoques de metodologías.

Tabla 2. Diferencias por etapas y enfoque metodológico.

Modelos Tradicionales	ETAPA	Modelos Ágiles
Planificación predictiva y "aislada"	Análisis de requerimientos Planificación	Planificación adaptativa: Entregas frecuentes + colaboración del cliente
Diseño flexible y Extensible + modelos + Documentación exhaustiva	Diseño	Diseño Simple: Documentación Mínima + Focalizado en la comunicación
Desarrollo individual con Roles y responsabilidades estrictas	Codificación	Transferencia de conocimiento: Programación en pares + conocimiento colectivo
Actividades de control]: Orientado a los hitos + Gestión miniproyectos	Pruebas Puesta en Producción	Liderazgo-Colaboración: empoderamiento + auto-organización

Por otra parte uno de los padres de la ingeniería de software Ivar Jacobson planteó..."En el futuro no van a haber mas metodologías prescriptivas como lo fueron UP, procesos estilo CMMI o procesos agiles estilo XP, SCRUM y otros. En cambio, lo que va a haber es una paleta de prácticas. Las prácticas van a venir primero y las metodologías van a ser meras colecciones de prácticas que las organizaciones van a escoger de la paleta."
Dentro de la Industria Cubana del Software, la Universidad de las Ciencias Informáticas (UCI), juega un papel principal porque combina docencia y producción con el objetivo de convertirse en una universidad innovadora de excelencia científica, distinguida por el uso de las tecnologías. En la UCI se produce gran cantidad de software que contribuye a la informatización de los mismos procesos de la universidad, así como de muchas de las esferas del país y de otras naciones latinoamericanas, elevando así los ingresos por conceptos de producción de software.
En agosto del 2007 se crea la Unidad de Desarrollo y Compatibilización de Productos informáticos para la defensa (UCID) dentro de la UCI con el objetivo de desarrollar software para las Fuerzas Armadas Revolucionarias (FAR). Se comienzan a desarrollar

los primeros productos utilizando una configuración de la metodología RUP, que era la más conocida y utilizada en ese momento en la universidad. Un diagnóstico realizado por la dirección de Calidad de la Universidad en el año 2008 arrojó que la mayoría de los proyectos de desarrollo (más del 60 %) utilizaban la metodología RUP para desarrollar software [Diagnóstico, 2008]. Según sus principales autores esta es un marco de desarrollo de software personalizable, por lo que tiene que ser adaptada mediante una configuración previa a aplicarse, teniendo en cuenta: las características de la empresa, la manera en que trabaja el grupo de desarrollo y la composición del mismo [Jacobson, 2000].

Durante el año 2008 comienza a aumentar el número de proyectos y personas en la UCID dedicadas al desarrollo de software y los resultados obtenidos fueron discretos, no fue hasta el 2009 que se lograron obtener algunos resultados luego de muchas dificultades con el proceso de desarrollo. Los resultados de una encuesta realizada por la autora de esta tesis (ver anexo I) a los 5 líderes de los proyectos, desarrollados en la UCID, que entregaron su primera versión al cliente en el 2008; los cuales clasifican como: Sistemas de Procesamiento de Transacciones (SPT), o Transaction Processing Systems (TPS), que procesan los datos resultado de la ocurrencia de las transacciones (eventos que ocurren como parte del negocio: ventas, compras, depósitos, retiros, devoluciones, pagos, etc.) del negocio [Mendoza, 2005] [Laudon, 2006]; arrojaron que la configuración realizada a la metodología RUP era bien compleja de satisfacer, la realidad fue que algunos de las actividades y artefactos de los que se seleccionaron en la configuración no se utilizaron y otros que se realizaron no llegaron a satisfacer las necesidades de los diferentes roles, además de que las características de los proyectos, del personal y del propio centro cambiaron.

En la actualidad en la UCID se desarrollan más de 50 proyectos de diferentes tipos entre los que podemos encontrar, además de los SPT, a los Sistemas Soporte a la Decisión (SSD), o Decisión Support Systems (DSS) que utilizan normas y modelos de decisión, que junto con una base de datos soporta todas las fases del proceso de toma de decisiones. Son más analíticos que otros sistemas y son interactivos, también los Sistemas de Información para la Gestión (SIG), o Management Information Systems (MIS) que combinan las tecnologías de la información con procedimientos permitiendo suministrar información a los gestores de una organización para la toma de decisiones. [Muñoz, 2003] [Laudon, 2006], entre otros. La cantidad de solicitudes de desarrollo han aumentado considerablemente, llegando este año 2010 la cifra a 105. Además contamos con 130 profesionales dedicados solamente al desarrollo de software si a esto se le suma la cantidad de estudiantes vinculados a los proyectos obtenemos una cifra de alrededor 600 personas dedicadas solamente a esta actividad.

Por lo que se impone una mayor organización en el proceso de desarrollo en aras de lograr algunos objetivos como: que las aplicaciones desarrolladas en el centro hagan lo que se espera de ellas en todas las ocasiones y que lo hagan además con un consumo mínimo de recursos, en especial de tiempo, espacio y esfuerzo; que resistan a gran parte de los condicionantes externos que pudieran afectarle pero que sean lo suficientemente flexibles para ser adaptadas fácilmente a nuevas situaciones. Que los humanos u otros sistemas que interactúan con ellas lo hagan de forma sencilla y racional, entre otros.

Por todo esto se decidió enfocar nuestros esfuerzos a desarrollar una nueva propuesta de metodología para guiar el proceso de desarrollo de software en el centro UCID. Es por eso que el **problema** de esta investigación queda definido como: La metodología que se utiliza en la UCID para desarrollar software no se adecúa a las características del centro y de sus proyectos por lo que afecta el proceso de desarrollo.

El objeto de estudio de esta investigación lo constituye el Proceso de desarrollo de software en la Unidad de Compatibilización, Integración y Desarrollo de Productos Informáticos para la Defensa. Más específicamente **el campo de acción** está relacionado con la metodología de desarrollo que se utiliza en el mismo. **El objetivo general de la investigación** es proponer una metodología de desarrollo para su aplicación en la UCID que se adecúe, a las características del centro y de sus proyectos, permitiendo mejorar los resultados obtenidos en el proceso de desarrollo de software. El cual esta desglosado en los siguientes **objetivos específicos**:

1. Realizar la fundamentación teórica para justificar la investigación y dejar definida la posición del autor.
2. Elaborar el diseño teórico de la investigación que permitirá organizar y ejecutar la investigación.
3. Determinar las particularidades de la UCID y de sus equipos de desarrollo para producir software y exponer los problemas presentados durante el proceso de desarrollo.
4. Proponer una metodología para guiar el proceso de desarrollo de Software en la UCID.
5. Validar la propuesta utilizando el método Delphi y la matriz de perfil competitivo (MPC) utilizada en el área de planificación estratégica.
6. Aplicar en proyectos pilotos para validar su efectividad.

Hipótesis: Si se aplica una nueva metodología en el proceso de desarrollo de Software en la UCID, que se adecúe a las características del centro y de sus proyectos, se obtendrán mejores resultados en el mismo.

Marco conceptual

Proceso de desarrollo de software: Conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. El término usuario no solo se refiere a humanos sino también a otros tipos de sistemas. [Jacobson, 2000]

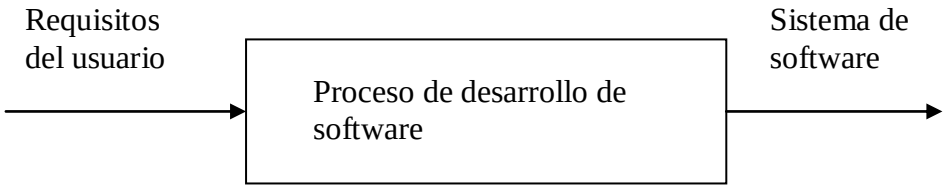


Figura 1. Proceso de desarrollo de software

Artefactos: Un producto o artefacto es un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final.
Un artefacto puede ser cualquiera de los siguientes:

- Un documento, como el documento de la arquitectura de software
- Un modelo, como el modelo de Casos de Uso o el de Diseño
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un caso de uso o un subsistema.

Roles: Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Las responsabilidades de un rol

son tanto el llevar a cabo un conjunto de actividades como el ser dueño de un conjunto de artefactos.
Actividades: Una actividad en concreto es una unidad de trabajo que una persona que desempeñe un rol puede ser solicitado a que realice. Las actividades tienen un objetivo en concreto, normalmente expresado en términos de crear o actualizar algún producto.
Flujos de trabajo: Un flujo de trabajo es una relación trabajadores, las actividades que realizan y los artefactos que producen.

Métodos de investigación.
Durante el desarrollo de la investigación se utilizaron métodos teóricos, empíricos y estadísticos.

Métodos teóricos
Lógicos
Hipotético – Deductivo
Para la elaboración de la hipótesis central de la investigación y para proponer nuevas líneas de trabajo a partir de los resultados parciales; este método desempeña un papel principal en la verificación de la hipótesis. Permite además llegar a conclusiones a partir de los conocimientos acumulados.
Sistémico
Para el estudio del objeto a través de sus componentes y las relaciones entre ellos.
Dialéctico
Para la determinación de las contradicciones existentes que caracterizan el comportamiento del objeto y así llegar a la solución del problema planteado. Este método tiene la ventaja de interrelacionar tanto el objeto como el sujeto.

Métodos investigativos particulares
La entrevista
Con el objetivo de obtener información sobre los grupos de desarrollo, específicamente relacionado con el proceso de producción en el que se encuentran vinculados.
La encuesta
Para obtener información sin la intervención del investigador, con un cuestionario más rígido dirigido a toda la muestra.

Tratamiento estadístico de los resultados
Para el tratamiento estadístico de los resultados se propone la utilización del SPSS (Statistical Package for social Sciences) es un software de análisis estadístico muy completo, con funciones de análisis descriptivo e inferencial, medidas de tendencia central y de dispersión, procesamiento de bases de datos con una cantidad de registros y campos virtualmente ilimitada, diseño automático de todo tipo de gráficos (de gráfico, barras, histogramas), funciones y distribuciones (recta, normal, etc), análisis de correlación y de regresión, así como de los más diversos pruebas o test paramétricas y no paramétricas (Chi cuadrado, t de student, de hipótesis, comparación de medidas, etc).
Pero también es necesario el análisis de los datos cualitativos que se proponen obtener en esta investigación. Los datos registrados, impresos, manuscritos o desgravados, en forma de notas tomadas durante la entrevista realizada, respuestas libres a preguntas abiertas, pueden ser procesados mediante el tratamiento cuantitativo de lo cualitativo. Este enfoque no es nuevo en la investigación, el procedimiento interpretativo estándar que se da, tanto de las preguntas abiertas como del análisis del contenido, comprende: reducción de los datos, selección de palabras claves, agrupamiento de frases en

dimensiones, edición de categorías, codificación de las mismas. El análisis se transforma en una cuantificación de códigos numéricos, el recuento de códigos y la obtención de frecuencias; independientemente de la estructura y significación de las categorías.
El procedimiento tradicional de la cuantificación de datos cualitativos es la categorización, la codificación y la tabulación. De este modo el dato textual se reduce a un tratamiento y análisis de datos numéricos. Interesa más la frecuencia de los códigos que el propio contenido de las categorías.
Además se propone utilizar Dimensions de SPSS: una plataforma completa de herramientas para diseño de encuestas, recogida y tratamiento de datos, análisis y publicación de resultados debido a que esta investigación utilizará métodos investigativos particulares como la entrevista y la encuesta. Dicha plataforma permite la integración de todas las etapas del proceso: recogida de datos multimodo que puede ser vía Web, en texto (escaneado), etc.
Tiene como ventajas:

1. El diseño vía Web de los cuestionarios.
2. La visión en tiempo real de los resultados de la recogida de los datos.
3. Para analizar los resultados utiliza análisis estadístico, minería de texto, minería de datos.

La metodología propuesta contribuirá a la mejora continua del proceso de desarrollo de software, permitiendo que el objetivo principal del desarrollo sea cumplido como es la exactitud del sistema, es decir la correspondencia entre el sistema y sus especificaciones, así como que el sistema cumpla con las necesidades del usuario y los productos se obtengan con la mayor calidad y en el menor tiempo posible. Facilitará además la comunicación entre usuario y técnico transformándola en amigable, sencilla y exenta de consideraciones técnicas, todas las partes implicadas podrán intercambiar información libremente. Será aplicable además a la construcción de cualquier tipo de software que se desarrolle en el centro, de cualquier tamaño y complejidad.
La tesis está estructurada en introducción, tres capítulos, conclusiones, recomendaciones bibliografía y anexos. En el capítulo 1 se realiza una descripción de las metodologías que existen para el desarrollo de software analizando las ventajas y desventajas de cada una de ellas. En el capítulo 2 se realiza la propuesta de la metodología a utilizar y en el capítulo 3 se valida la propuesta.

Capítulo 1. Metodologías de desarrollo de software.

Introducción

En este capítulo se lleva a cabo un estudio sobre algunas de las metodologías de desarrollo de software más populares y el papel de las mismas en la producción de software, haciendo énfasis en los elementos que las componen y puntualizando sus ventajas y desventajas. Además se realiza un análisis de las particularidades de la UCID para desarrollar software y de los problemas que presentaron los proyectos que se desarrollaron determinando sus causas.

1. Metodologías en el desarrollo de software

La formalización del proceso de desarrollo se define como un marco de referencia denominado ciclo de desarrollo del software o ciclo de vida del desarrollo de software; el cual se encuentra especificado en la ISO 12207 "Tecnologías de la información: procesos del ciclo de vida del software". Se puede describir como, "el período de tiempo que comienza con la decisión de desarrollar un producto software y finaliza cuando se ha entregado éste". Este ciclo, por lo general incluye, una fase de requisitos, fase de diseño, fase de implementación, fase de prueba, y a veces, fase de instalación y aceptación [ISO 12207, 2004].

Estas fases se descomponen en tareas que deben ser cumplimentadas para obtener un producto de software, cada una de las cuales puede ser abordada y resuelta de múltiples maneras, con distintas herramientas y utilizando distintas técnicas. Es necesario saber además cuándo se puede dar por concluida una tarea, quién debe realizarla, en que orden deben ejecutarse y que documentación se utilizará para llevarlas a cabo y cual se obtendrá como resultado final.

Estamos hablando de detalles organizativos, de un "estilo" de hacer las cosas, pero que yendo un poco más allá que un simple estilo, formalizando ese "estilo" añadiendo algo de rigurosidad y normas se obtiene una **metodología**. Todos los integrantes de un equipo de desarrollo deben seguir un criterio común a la hora de realizar las tareas del ciclo de vida. Ese criterio, esa manera común es una metodología de desarrollo.

Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el objetivo de hacerlo más predecible y eficiente, donde predecir no significa perder la capacidad adaptativa, no significa evitar la introducción de cambios en los requisitos, ni evitar que nuevos requisitos surjan sino definir un camino reproducible para obtener resultados confiables. Definen además una representación que permite facilitar la manipulación de modelos y la comunicación e intercambio de información entre todas las partes involucradas en la construcción de un sistema [Gacitúa, 2003].

Los proyectos exitosos son aquellos que son administrados siguiendo una serie de procesos que permiten organizar y luego controlar el proyecto, considerando válido destacar que aquellos procesos que no sigan estos lineamientos corren un alto riesgo de fracasar [Gabardini]. Los profesionales se ven envueltos en discusiones en la que muchos plantean sus opiniones como verdades indiscutibles, y en la que la decisión de qué metodología utilizar, parece ser un acto de fe antes que una evaluación de alternativas técnicas, con sus costos, beneficios y riesgos asociados. Es necesario destacar además que los métodos son importantes, pero no se debe perder de vista que el éxito del proyecto depende más de la comunicación efectiva con los interesados, el manejo de las expectativas, el valor generado para el negocio y las personas que participan en el proyecto.

Hoy en día existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Un ejemplo de ellas son las propuestas

tradicionales centradas específicamente en el control del proceso de ahí que hayan recibido la crítica de ser burocráticas, es decir, hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda, lo cual hace que se reduzca el tiempo efectivo para cumplir con el objetivo real: construir software [Díaz].
Por otra parte comienza a emplearse el término ágil para caracterizar metodologías luego de una reunión celebrada en Utah-EEUU en febrero de 2001, en esa reunión un grupo de 17 expertos esbozaron los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto, se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales. Tras esta reunión se creó La Alianza Ágil, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil" [Canós, 2003].
Las metodologías tradicionales están pensadas para el uso exhaustivo de la documentación durante todo el ciclo del proyecto mientras que las ágiles ponen vital importancia en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y al mantener una buena relación con el cliente, además se deberían aplicar en proyectos pequeños donde exista mucha incertidumbre, el entorno sea volátil y los requisitos no se conozcan con exactitud; a pesar de que aportan una elevada simplificación estas metodologías no renuncian a las prácticas esenciales para asegurar la calidad del producto [Palacio, 2006].

2. Descripción de las metodologías existentes para el desarrollo de software

Los Gurús de la ingeniería del software alrededor del mundo han aportado pensamientos tanto positivos y negativos sobre las metodologías y su aplicación; son muy utilizadas y pueden ser mezcladas con otras para lograr la complementación y están para quedarse, evolucionar y revolucionar las prácticas de ingeniería.
Entre las metodologías tradicionales, las cuales se enfocan en un proceso definido, estableciendo fases con documentación bien definida en cada una de ellas y se basan en la predicción podemos encontrar entre las más populares: El Proceso Unificado de desarrollo, MSF, EUP.

2.1 Tradicionales

2.1.1 El Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo fue creado por el mismo grupo de expertos que crearon *UML*, Ivar Jacobson, Grady Booch y James Rumbaugh en el año 1998. El objetivo que se perseguía con esta metodología era producir software de alta calidad, es decir, que cumpliera con los requerimientos de los usuarios dentro de una planificación y presupuesto establecidos, esta metodología concibió desde sus inicios el uso de *UML* como lenguaje de modelado.
Es un proceso dirigido por casos de uso que avanzan a través de una serie de flujos de trabajo (requisitos, análisis, diseño, implementación, prueba) que se repiten en las fases de inicio, elaboración, construcción y transición que define la misma. Está centrado en la arquitectura y es iterativo e incremental. Además de que cubre el ciclo de vida de desarrollo de un proyecto, toma en cuenta las mejores prácticas a utilizar en el modelo de desarrollo de software. A continuación se muestran estas prácticas:

• Desarrollo de software en forma iterativa.
• Manejo de requerimientos.
• Utiliza arquitectura basada en componentes.
• Modela el software visualmente.
• Verifica la calidad del software.
• Controla los cambios. [Jacobson, 2000]

El PU puede ser aplicado a diferentes tipos de software incluyendo proyectos de pequeña y gran escala con diferentes tipos de complejidades técnicas y administrativas. Cruzando diferentes dominios de aplicaciones y estructuras organizativas por lo tanto se considera un framework que provee una infraestructura para ejecutar proyectos pero no todos los detalles requeridos para ejecutar los mismos.

El Proceso Unificado es considerado un proceso porque "define quién está haciendo qué, cuándo lo hace y cómo alcanzar cierto objetivo, en este caso el desarrollo de software". Los conceptos clave del Proceso Unificado son:

Fases e iteraciones	¿Cuándo se hace?
Flujos de trabajo de procesos (actividades y pasos)	¿Qué se está haciendo?
Artefactos (modelos, reportes, documentos)	¿Qué se produjo?
Trabajador: un arquitecto	¿Quién lo hace?

Para apoyar el trabajo ha sido desarrollada, por la Compañía norteamericana Rational Corporation, la herramienta CASE (Computer Assisted Software Engineering) Rational Rose en el año 2000. Esta herramienta integra todos los elementos que propone la metodología para cubrir el ciclo de vida de un proyecto.

El proceso unificado describe cuatro fases:

Inicio: Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones. Se realiza el proceso de planificación general y un plan de trabajo detallado para la siguiente fase, así como el plan para la siguiente iteración.

Elaboración: Los casos de uso seleccionados para desarrollarse en esta fase permiten definir la arquitectura del sistema, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar del problema y comienza la ejecución del plan de manejo de riesgos, según las prioridades definidas en él. Al final de la fase se determina la viabilidad de continuar el proyecto y si se decide proseguir, dado que la mayor parte de los riesgos han sido mitigados, se escriben los planes de trabajo de las etapas de construcción y transición y se detalla el plan de trabajo de la primera iteración de la fase de construcción.

Construcción: En esta fase tiene lugar la implementación del sistema (codificación) y se finalizan aquellos aspectos del análisis y diseño que hubieran quedado por finalizar en la anterior fase. Como resultado de la fase obtenemos el producto y los manuales de usuario, así como la documentación del modelo de la aplicación. El criterio de evaluación de la fase es determinar si el producto es lo bastante maduro como para ser usado por los usuarios finales. El producto al finalizar esta fase se considera que se encuentra en una versión "beta" (ya que pueden surgir incidencias que se resolverán en la siguiente fase).

Transición: El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, comprobar la estabilidad del sistema y la satisfacción que ofrece. Capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto al inicio del mismo. Propone además 9 disciplinas totales que se dividen entre 2 grupos: 6 disciplinas básicas y 3 disciplinas de soporte:
Disciplinas básicas: Modelo del Negocio, Requerimientos, Análisis y Diseño, Implementación, Pruebas, Despliegue
Disciplinas de soporte: Configuración y Gestión de Cambios, Gestión de Proyectos y Entorno.
Negocio: El objetivo del modelo de negocio es entender la forma de funcionar de la organización del cliente, tanto en estructura como en dinámica de sus procesos. Su objetivo no es modelar por lo tanto el sistema software a implantar, sino las funciones y roles que realiza la organización para poder realizar más fácilmente la reingeniería de procesos o la implantación del nuevo sistema.
Requerimientos: El objetivo de esta disciplina es entender lo que el sistema tendría que hacer para automatizar el negocio y capturar este conocimiento en un modelo de casos de uso.
Análisis y Diseño: Se enfoca en analizar los requerimientos y diseñar el sistema capturando el conocimiento en modelos de análisis y diseño.
Implementación: En la implementación se implementa el sistema basado en el modelo de implementación.
Pruebas: Evalúa el sistema contra los requerimientos basado en el modelo de pruebas.
Despliegue: Se enfoca en desplegar el sistema basado en el modelo de despliegue.
Configuración: Disciplina que se centra en la administración de la configuración del sistema y las demandas de cambio.
Gestión de cambio: Su objetivo es la administración de los cambios en el sistema.
Entorno: Su objetivo es el ambiente del proyecto, incluyendo los procesos y herramientas.

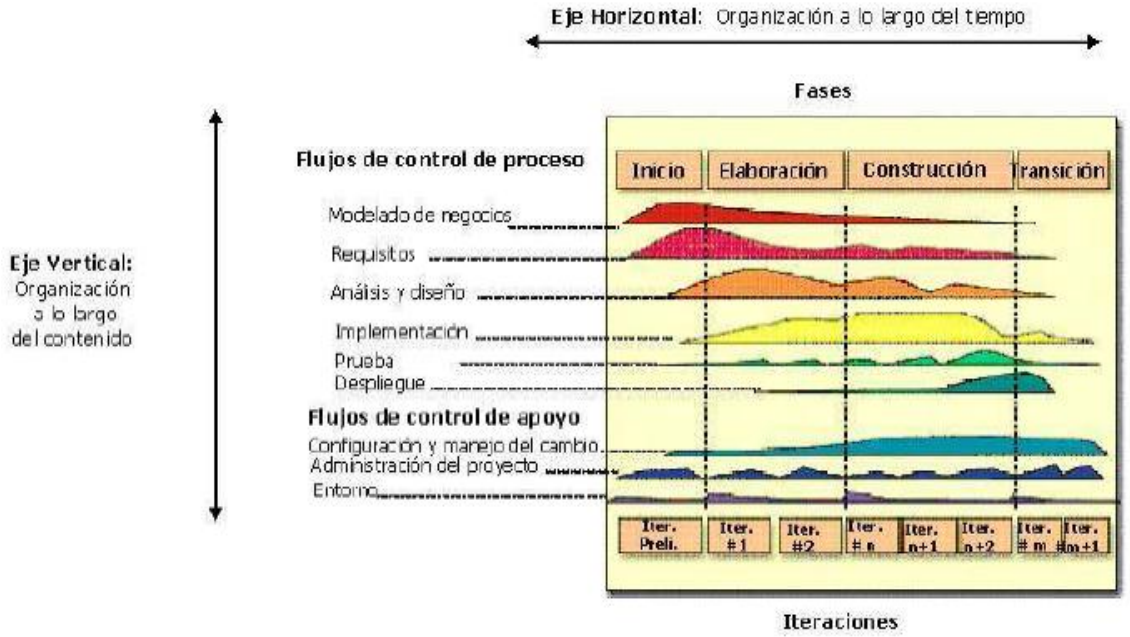


Figura 2. Estructura del proceso unificado de desarrollo de software [Letelier, 2002].

Entre los puntos clave que presenta el PU para lograr el éxito se encuentran:

- Tiene bien definido el proceso de desarrollo de software estructurado por flujos de trabajos.
- Brinda una guía para encontrar, organizar y documentar los requisitos funcionales.
- Propone el desarrollo del producto mediante iteraciones con hitos bien definidos.
- Plantea el desarrollo basado en componentes, que permite que el sistema se vaya creando a medida que se obtienen los componentes.
- Utiliza UML como lenguaje para visualizar, especificar, construir y documentar los artefactos. Lo que ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.
- Define disciplinas de gestión de cambios y de configuración para gestionar los cambios que aparecen durante el desarrollo y la participación de múltiples desarrolladores en un mismo incremento. [Rational, 2007]

Sin embargo en RUP se especifica qué se debe hacer pero no cómo hacerlo por lo que cada equipo de desarrollo que utiliza esta metodología la interpreta a su manera. Otro de los inconvenientes es que propone generar gran cantidad de artefactos en cada uno de los flujos por lo que se corre el riesgo de dedicar mucho tiempo a esta actividad y de que quede poco para la codificación, además del costo de mantener cada artefacto actualizado. Otra de las críticas que recibe el PU es que a pesar de tener una disciplina para la gestión de los cambios ve a los mismos como un factor de riesgo crítico en los proyectos de software.

2.1.2 Microsoft Solution Framework (MSF)

MSF es una metodología desarrollada por Microsoft Consulting Services en conjunto con varios grupos de negocios de Microsoft y otras fuentes de la industria. Más que una metodología es una serie de modelos flexibles interrelacionados que guían a una organización sobre como ensamblar los recursos, el personal y las técnicas necesaria para asegurar que su infraestructura tecnológica y sus soluciones cumplan los objetivos de negocio.MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Se puede utilizar junto con el Proceso Unificado

MSF tiene las siguientes características [Mendoza, 2004]:

• Adaptable: es utilizable para cualquier proyecto y no se limita a un equipo o proyecto específico.

• Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

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

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

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de componentes y finalmente el modelo de Aplicación.

Modelo de Arquitectura: Propone realizar un mapa de la arquitectura de la empresa para la implementación de la arquitectura de solución proveyendo detalles específicos de la solución para las vistas de la arquitectura de datos, tecnología y aplicación.

Modelo de equipo: El Modelo de Equipo de MSF muestra cómo estructurar equipos de alto desempeño para crear soluciones rápidamente y con buena calidad. Hace énfasis en las comunicaciones claras y en un equipo de iguales, con papeles y responsabilidades

claras en todo el proyecto. La calidad del producto se asegura por cada miembro del equipo. Existen 6 roles en el Modelo de Equipos. [Microsoft Corporation, 1999]

Figura 3. Roles del modelo de equipo

Rol	**Objetivo**
Gerente de Producto	Cliente satisfecho
Gerente de Programa	Entrega dentro de las restricciones del proyecto
Desarrollador	Entrega en función de especificaciones
Pruebas	Aseguramiento de funcionalidad
Educación a usuarios	Máximo aprovechamiento del producto por el usuario
Gerente de Logística	Asegurar el desplegado de la solución

Modelo de proceso: El Modelo de Procesos de MSF provee una estructura para el desarrollo de aplicaciones que consiste en 4 etapas distintas, cada una de las cuales culmina con una meta definida. Las 4 etapas son:
Fase 1: Visión. En esta fase el equipo y el cliente definen los requerimientos del negocio y los objetivos generales del proyecto. La fase culmina con el hito *Visión y Alcance aprobados*.
Fase 2: Planeación. Durante la fase de planeación el equipo crea un borrador del plan maestro del proyecto, además de un cronograma del proyecto y de la especificación funcional del proyecto. Esta fase culmina con el hito *Plan del proyecto aprobado*.
Fase 3: Desarrollo. Esta fase involucra una serie de liberaciones internas del producto, desarrollados por partes para medir su progreso y para asegurarse que todos sus módulos o partes están sincronizados y pueden integrarse. La fase culmina con el hito *Alcance completo*.
Fase 4: Estabilización. Esta fase se centra en probar el producto. El proceso de prueba hace énfasis en el uso y el funcionamiento del producto en las condiciones del ambiente real. La fase culmina con el hito *liberaciones aprobada*.
Fase 5: Implantación: En esta fase el equipo implanta la tecnología y los componentes utilizados por la solución, estabiliza la implantación, apoya el funcionamiento y la transición del proyecto, y obtiene la aprobación final del cliente. La fase termina con el hito *Implantación completa*. [Microsoft Corporation, 2002]
Modelo de gestión de riesgo: Se aplica continuamente a través del proceso de ciclo de vida. Es proactivo y no reactivo. Transita por la ejecución de las actividades siguientes: Identificación de riesgos, Análisis, Planeación, Seguimiento, Reporte, Control y Aprendizaje.
Modelo de diseño de componentes: La estructura de este modelo soporta las actividades del proyecto relacionadas con el diseño, a través del diseño conceptual, el

diseño lógico y el diseño físico, de la aplicación que se está construyendo. Las fases y los documentos del diseño conceptual, lógico y físico, proveen tres perspectivas diferentes para cada una de las 3 audiencias: los usuarios, el equipo y los desarrolladores. Además este modelo se relaciona con las fases 2 y 3 del modelo de proceso.

Modelo de aplicación: Enfocado a la funcionalidad del desarrollo. Este modelo contempla un diseño lógico en tres capas para el diseño de aplicaciones (soluciones) distribuidas multicapas. Define una aplicación como una red lógica de servicios distribuibles y reutilizables que cooperan en tareas comunes. Contempla tres categorías de servicios: Servicios de usuario, Reglas de negocio y Servicio de datos.

Gracias al seguimiento de este modelo, los desarrollos pueden ser reutilizables y diseñados de manera modular. Ello permite que una solución crezca y sea escalable al permitir fácilmente que algún módulo existente se modifique sin afectar los demás componentes, ó que se agregue un nuevo módulo encargado de nueva funcionalidad.

Entre las debilidades que presenta MSF es que por ser una metodología tan abierta, es necesaria la definición de artefactos específicos que permitan precisar las necesidades de los usuarios y las funcionalidades de los componentes por eso algunos autores proponen utilizarlas combinadas.

2.1.3 Enterprise Unified Process (EUP)

Existen determinados aspectos cruciales que no fueron considerados en el Proceso Unificado, principalmente la consideración de que existen numerosas empresas que no sólo se dedican al desarrollo de un simple proyecto de software sino que desarrollan numerosos proyectos y a su vez dan soporte. EUP extiende RUP para incluir una nueva fase de producción donde se opera y mantiene el sistema y una fase de jubilación en donde se remueve el sistema de producción en el final del ciclo de vida y 8 nuevas disciplinas:

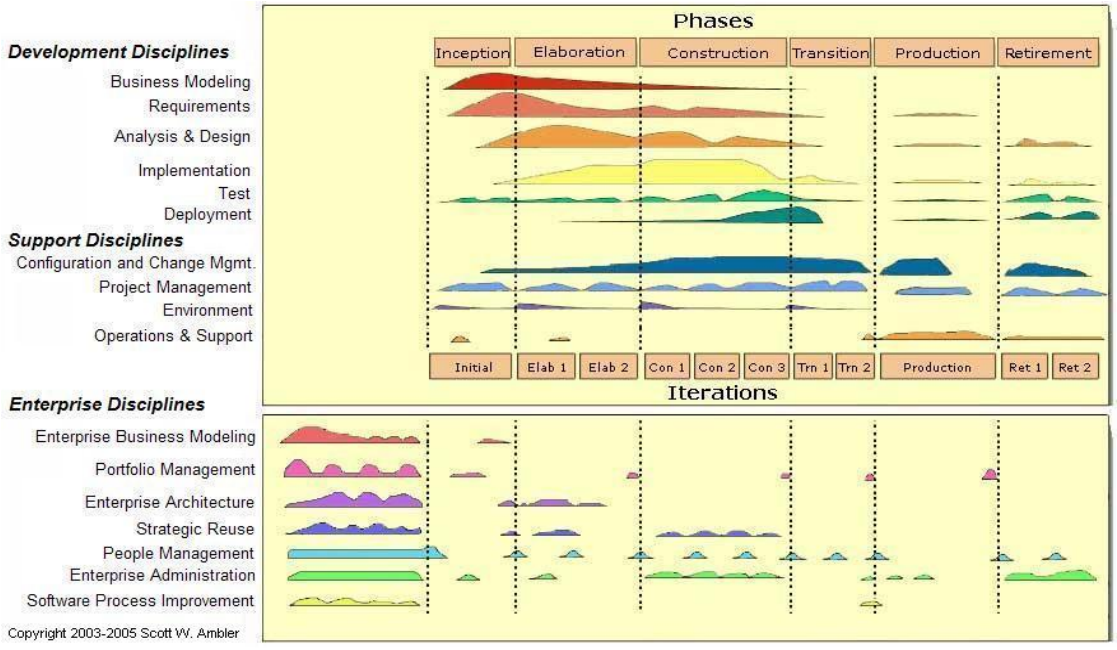


Figura 4. Estructura del Enterprise Unified Process (EUP)

Organizaciones que desarrollan y despliegan sistemas internos realizan más en términos de operaciones de soporte que compañías que producen software empaquetado. Se gasta más tiempo y esfuerzo en términos de soporte para mantener la diversificada base de clientes satisfechos y para esto no se tienen equipos operacionales, mientras que de la

manera formal se tiene un equipo entero dedicado sólo a estas operaciones [Ambler, 1999].

Disciplinas

A las disciplinas de soporte se le agrega una denominada **operación y soporte** la cual se encarga de operar y brindar soporte al sistema en un ambiente de producción, las metas de la misma son:

Operación

1. Enfocar las operaciones en asegurar que el software se desempeña correctamente.
2. Que la red está disponible y monitoreada
3. Que los datos apropiados se están salvando y pueden ser recuperados en cualquier momento.
4. Que los planes contra desastres han sido creados y que en caso de que ocurra alguno pueden ser ejecutados y restaurar el sistema a su funcionamiento normal.

Soporte

1. Asistir a los usuarios finales respondiendo sus preguntas.
2. Analizar los problemas que ellos encuentran en el entorno productivo.
3. Capturando nuevas funcionalidades
4. Aplicar cambios o realizar arreglos.

Un aspecto importante de esta disciplina para el éxito de la entidad es que debe realizarse de la manera más ágil posible.

EUP agrega además 7 disciplinas nuevas denominadas de empresa:

Modelando Negocio de empresa: Disciplina que se encarga de facilitar el entendimiento del negocio de la organización, ayuda a identificar las áreas del mismo que pueden ser mejoradas con la automatización o identificar nuevas áreas, así como las que necesitan ser reforzadas, también provee de información al equipo para delimitar el alcance del proyecto dentro de todo el negocio.

Gestión de cartera: Es una colección de proyectos de las tecnologías de la información propuestos y en progreso, así como los sistemas desplegados en la organización; todos encaminados a la creación de sistemas que soporten la estrategia de negocio. Esto se realiza para asegurar que todos los proyectos están visibles, planeados y alineados con los objetivos de la empresa. Es importante destacar que la administración de la cartera debe mantenerse a través de todo el ciclo de vida del proyecto hasta la fase de retiro.

Arquitectura de empresa: Esta arquitectura de empresa incluye frameworks, redes, configuraciones de despliegue e infraestructura de soporte que forman la arquitectura técnica de la empresa. Esto incluye el ambiente en que los productos serán desplegados en la empresa.

Reutilización estratégica: Esta disciplina representa un concepto muy importante en la industria de las tecnologías de la información, se puede rehusar código, componentes, patrones, artefactos y plantillas. Pero no es algo que pase simplemente sino que hay que trabajar en base a eso, por eso se propone realizar un plan o programa para planificar la reutilización.

Gestión de personas: Hoy día las organizaciones compiten por sobrevivir en dos aspectos fundamentales: los productos o servicios que ofertan y por las personas con talento para ejecutar tales actividades. Se necesita más que planes de ejecución de proyectos y tareas técnicas, por lo tanto esta disciplina propone administrar el equipo de desarrollo, ayudarlos a crecer, intermediar en las relaciones de unos con otros. Describe además el proceso de organización, monitoreo, educación y motivación del personal

para asegurarse que el equipo trabaja unido y contribuye exitosamente al proyecto con su organización.
Administración de empresas: Esta disciplina incluye la administración de recursos físicos, recursos de información y la seguridad de la empresa [Ambler, 2005].

Incluso los departamentos más pequeños de tecnologías de la información necesitan un proceso de software que los equipos puedan seguir y adaptar; los más grandes pueden necesitar diferentes procesos en dependencia de la categoría de los proyectos que realizan.
Beneficios de EUP

1. La utilización de una fase antes de inicio ayuda a las entidades que no se dedican al desarrollo de un solo producto, sino que tienen diferentes tipos de proyectos y dan soporte, a organizarse, a mantener una gestión de toda su actividad. La cartera de proyectos permite priorizar y ejecutar aquellos basados en las necesidades reales del negocio, permite además encontrar productos relacionados o posibles puntos de solapamiento.
2. Al comenzar a modelar el negocio antes de la fase de inicio se incrementa el esfuerzo en comprender el alcance aumentando las posibilidades de construir un sistema sobre las verdaderas necesidades de la empresa.
3. La utilización de la estrategia de rehúso puede reducir significativamente el costo de desarrollo de software.
4. La preparación del equipo de desarrollo y de soporte es esencial tanto dentro del ciclo de desarrollo del producto como para su desarrollo en el ambiente productivo después.

Este proceso es muy importante por agregar esa disciplina de soporte que es tan importante en el proyecto pero que casi nunca se ve su importancia hasta que se termina el proyecto y comienzan los problemas para el despliegue del mismo. Por otra parte esta fase de pre-inicio como algunos la llaman ayuda a aumentar la visión sobre lo que se quiere lograr en ese entorno de negocio y sobre todo lo que interactúa con el proyecto, antes de que comience la fase de inicio del proyecto.

2.2 Ágiles

2.2.1 Proceso Unificado Ágil

El AUP por sus siglas en inglés, es un acercamiento aerodinámico al desarrollo del software basado en el Proceso Unificado, que contiene disciplinas y entregables incremental en el tiempo.

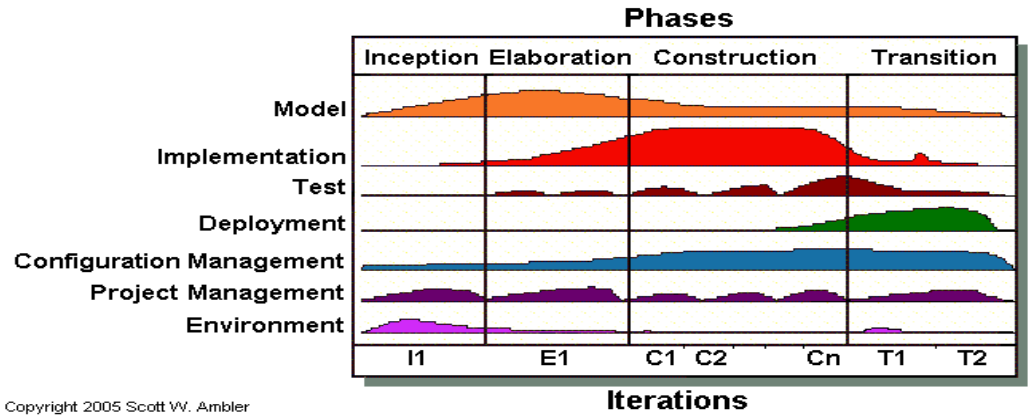


Figura 5. Estructura del Proceso Unificado Ágil

Se nota a primera vista que las disciplinas cambiaron, la disciplina modelo que antes no existía en el PU ahora contiene las disciplinas modelado de negocio, requisitos, análisis y diseño. De manera que modelo es una disciplina importante en AUP pero siempre manteniéndose de manera ágil, creando sólo los artefactos necesarios. Por otra parte la disciplina administración de la configuración y gestión del cambio es ahora la disciplina de gestión de configuración. En la versión ágil las actividades de gestión del cambio es típicamente parte de los esfuerzos de administración de requerimientos que son parte de la disciplina modelo. [Ambler, 2001].

Se divide como UP en cuatro fases:

1. **Inicio:** El objetivo es identificar el alcance del proyecto, la posible arquitectura de sistema y obtener una aceptación inicial de la ejecución del proyecto.
2. **Elaboración:** El objetivo es probar la arquitectura de sistema.
3. **Construcción:** El objetivo es construir un software que funcione de manera regular o incremental de acuerdo a las necesidades y prioridades de los clientes.
4. **Transición:** El objetivo es validar y desplegar el sistema en un ambiente de producción.

Lo componen las siguientes disciplinas:

1. **Modelo:** Su objetivo es entender el negocio de la organización y el dominio del problema para poder definir la dirección del proyecto e identificar una solución viable para el mismo.
2. **Implementación:** El objetivo de esta disciplina es transformar los modelos en código ejecutable y desarrollar un nivel básico de prueba, en particular las pruebas de unidad.
3. **Prueba:** El objetivo de esta disciplina es desarrollar una evaluación objetiva para asegurar la calidad. Esto incluye encontrar defectos, validar que el sistema funciona como se diseñó y verificar que los requerimientos se han cumplido.
4. **Despliegue:** El objetivo de esta disciplina es planificar el despliegue del sistema y ejecutar el plan para hacer el sistema disponible para los usuarios finales.
5. **Administración de la Configuración:** El objetivo de esta disciplina es administrar el acceso a los artefactos del proyecto.
6. **Administración del proyecto**. El objetivo de esta disciplina es dirigir las actividades que tienen lugar en el proyecto, esto incluye manejo de riesgos,

dirigir personas, asignar tareas, seguir el progreso, además de coordinar con personas y sistemas fuera del alcance del proyecto.

7. **Ambiente**: El objetivo de esta disciplina es darle soporte al resto del esfuerzo asegurando que los procesos, las guías y herramientas están disponibles para cuando el equipo las necesite. [Ambler, 2002].

Filosofías de AUP

1. *El equipo sabe lo que está haciendo*, no va a leer documentación detallada del proceso pero si necesitará alguna guía de alto nivel por momentos.
2. *Simplicidad,* todo esta descrito de manera precisa usando unas pocas páginas no miles de ellas.
3. *Agilidad,* AUP se rige por los principios de la Alianza Ágil
4. *Enfoque en actividades de alto valor,* el enfoque se centra en las actividades que realmente cuentan y no en cualquier cosa que pueda pasar en el proyecto.
5. *Herramientas,* se puede usar cualquier herramienta siempre y cuando sea sencilla de utilizar y preferiblemente que sea libre.

AUP es un acercamiento entre XP y el PU, un proceso que es ágil y todavía incluye artefactos a los que estamos acostumbrados. XP es un gran proceso pero dan mayor valor al individuo y a la colaboración con el cliente, además de que sus principios y prácticas son de sentido común, pero son llevadas al extremo como indica su nombre. El PU por otra parte tiene mucho que ofrecer en cuanto a artefactos y actividades pero puede reducirse a algo muy útil, como han planteado los expertos en el tema. AUP es algo entre estos dos, adoptando muchas de las técnicas ágiles de XP y de otros procesos ágiles como el denominado Modelado Ágil pero reteniendo algo de la formalidad del PU.

2.2.2 Proceso Unificado Básico (BUP)

Los proyectos pequeños tienen necesidades diferentes a los grandes proyectos. Por lo tanto se creó el Proceso Unificado Básico, un proceso basado en prácticas para proyectos y equipos pequeños. Es una versión aerodinámica de la versión de IBM de RUP optimizado para proyectos pequeños, constituidos por equipos de 3 a 6 personas y que involucran de 3 a 6 meses de esfuerzo de desarrollo. Presenta las principales características de RUP, desarrollo iterativo, dirigido por casos de uso, manejando los riesgos y centrado en la arquitectura. Las más opcionales partes de RUP han sido excluidas y muchos elementos se han unido, el resultado es un proceso mucho más simple.

Está organizado en dos dimensiones métodos y procesos. En la dimensión métodos es donde están definidos los elementos del método (roles, actividades, artefactos y guías) independiente de cómo ellos son aplicados en un ciclo de vida de un proyecto. La dimensión proceso es donde los elementos del método son aplicados en el sentido de su comportamiento, donde los mismos elementos de los métodos, muchos ciclos de vida para diferentes tipos de proyecto pueden ser creados.

Disciplinas

Pero el método BUP está enfocado en las siguientes disciplinas: requerimientos, arquitectura, desarrollo, pruebas, administración de proyecto y gestión del cambio.

Otras disciplinas y áreas fueron omitidas como modelación del negocio, ambiente, administración de requerimientos y administración de la configuración; estas son

consideradas muy avanzadas para proyectos pequeños o que van a ser manejadas por otras áreas de la organización y no por el equipo del proyecto.
Análisis y diseño no están en disciplinas separadas pero son absorbidas en otras disciplinas como arquitectura y desarrollo, porque el arquitecto tiene que efectuar algún nivel de análisis cuando se identifican soluciones, se utilizan patrones, se reutilizan componentes. El diseño de alto nivel es solo realizado por el arquitecto cuando se enfrenta a mayores componentes o interfaces.
Por otro lado los desarrolladores realizan algún nivel de análisis y diseño cuando identifican las clases y sus responsabilidades. El diseño en BUP puede aparecer antes o al mismo tiempo cuando los desarrolladores escriben el código y las pruebas de unidad.
Los roles que presenta son los esenciales, las tareas también fueron transformadas o unidas dentro de otras y la lista de artefactos es pequeña.

Ejemplo basado en BUP

Desarrollar una solución para un requerimiento **Actividad**

Diseñar la solución
Implementar pruebas de desarrollo
Implementar la solución
Correr las pruebas de desarrollo
Integrar la solución

Tareas

BUP es un proceso como se puede observar bastante sencillo y siempre recomendado para proyectos pequeños.

2.2.3 Proceso Unificado abierto Open/UP

OpenUP/Basic es un FrameWork de procesos de desarrollo de software de código abierto, permite abordar ágilmente el proceso de desarrollo de software, con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas. Es un proceso para pequeños equipos de desarrollo que valoran los beneficios de la colaboración y de los involucrados con el resultado del proyecto, por encima de formalidades innecesarias, está caracterizado por cuatro principios básicos interrelacionados:

- **Colaboración** para unificar intereses y compartir conocimientos.
- **Equilibrio** de prioridades competentes a maximizar el valor de los involucrados con el resultado del proyecto.
- Enfoque en la **articulación** de la arquitectura.
- **Desarrollo continuo** para obtener realimentación y realizar las mejoras respectivas.

Un proyecto en OpenUP se organiza en micro-incrementos y se aplica una colaboración intensiva en equipos autoorganizados. El proyecto se divide en iteraciones planificadas y las iteraciones se centran en el valor de las entregas incrementales hacia los clientes de manera predecible. Las iteraciones se organizan por fases y en cada una de ellas uno de los objetivos principales es la reducción del riesgo que se crea al final de la fase y aumentando el valor.
OpenUP aplica propuestas iterativas e incrementales dentro del ciclo de vida, tratando de ser manejable en relación con el PU. Es un proceso ágil, ligero, que promueve el desarrollo del software con buenas prácticas, haciéndolo un proceso pequeño extensible,

sin embargo asume que el equipo del proyecto no es con toda seguridad responsable de las actividades y decisiones que se asignan a otras áreas de la organización.

2.2.4 Metodologías Ágiles (MAs)

Estas metodologías se centran en otras dimensiones, como por ejemplo el factor humano o el producto de software; dan mayor valor al individuo, en un modelo de desarrollo incremental (pequeñas entregas con ciclos rápidos), cooperativo (desarrolladores y usuarios trabajan juntos en estrecha comunicación), directo (el método es simple y fácil de aprender) y adaptativo (capaz de incorporar los cambios). Sus claves son la velocidad y la simplicidad. De acuerdo con ellas, los equipos de trabajo se concentran en obtener lo antes posible una pieza útil que implemente sólo lo que sea más urgente; de inmediato requieren retroalimentación de lo que han hecho y lo tienen muy en cuenta. Luego prosiguen con ciclos igualmente breves, desarrollando de manera incremental. [Abrahamsson, 2002]. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

El objetivo de este análisis es conocer las características de las más reconocidas por los especialistas para poder combinar o adoptar algún principio ya que es muy recomendado por lo expertos que se utilice lo que mas efectivo sea para la organización y plantean que algunas de estas efectivamente deben ser utilizadas con un proceso más sólido. A continuación se describirán las principales características de las metodologías ágiles más populares: Extreme Programming, Scrum, Crystal Methods, Feature Driven Development, Dynamic Systems Development Method, Adaptive Software Development, Agile Modeling y Lean Development.

2.2.5 Extreme Programing (XP)

La Programación Extrema es sin duda alguna el método ágil más popular entre las MAs: 38% del mercado ágil contra 23% de su competidor más cercano, FDD. Luego vienen Adaptive Software Development con 22%, DSDM con 19%, Crystal con 8%, Lean Development con 7% y Scrum con 3%. Todos los demás juntos suman 9%. [Charette, 2004].

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en 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 coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. [Hunt, 2006]

El ciclo de vida ideal de XP consiste de seis fases [2]: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto y se divide (a grandes rasgos) en los siguientes pasos [12]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

XP se sostiene en cuatro valores: comunicación, simplicidad, retroalimentación y coraje. Pero tan conocidos como sus valores son sus prácticas. Kent Beck su promotor sostiene que se trata más de lineamientos que de reglas:

Juego de Planeamiento. Busca determinar rápidamente el alcance de la versión siguiente, combinando prioridades de negocio definidas por el cliente con las estimaciones técnicas de los programadores. Éstos estiman el esfuerzo necesario para implementar las historias del cliente y éste decide sobre el alcance y la agenda de las entregas. Las historias se escriben en pequeñas fichas, que algunas veces se tiran. Cuando esto sucede, lo único restante que se parece a un requerimiento es una multitud de pruebas automatizadas, las pruebas de aceptación.

Entregas pequeñas y frecuentes. Se produce un pequeño sistema rápidamente, al menos uno cada dos o tres meses. Se debe liberar una cada mes y se agregan pocos rasgos cada vez.

Metáforas del sistema. El sistema se define a través de una metáfora o un conjunto de metáforas, una "historia compartida" por clientes, managers y programadores que orienta todo el sistema describiendo como funciona.

Diseño simple. El énfasis se deposita en diseñar la solución más simple susceptible de implementarse en el momento. Se eliminan complejidades innecesarias y código extra, y se define la menor cantidad de clases posible. No debe duplicarse código. Nadie en XP llega a prescribir que no haya diseño concreto, pero el diseño se limita a algunas tarjetas elaboradas en sesiones de diseño de 10 a 30 minutos. Esta es la práctica donde se impone el minimalismo de YAGNI: no implementar nada que no se necesite ahora; o bien, nunca implementar algo que vaya a necesitarse más adelante; minimizar diagramas y documentos.

Prueba continua. El desarrollo está orientado por las pruebas. Los clientes ayudan a escribir las pruebas funcionales antes que se escriba el código. Esto es test-driven development. El propósito del código real no es cumplir un requerimiento, sino pasar las pruebas. Las pruebas y el código son escritas por el mismo programador, pero la prueba debería realizarse sin intervención humana, y es a todo o nada. Hay dos clases de prueba: la prueba unitaria, que verifica una sola clase, o un pequeño conjunto de clases; la prueba de aceptación verifica todo el sistema, o una gran parte.

Refactorización continúa. Se refactoriza el sistema eliminando duplicación, mejorando la comunicación y agregando flexibilidad sin cambiar la funcionalidad. El proceso consiste en una serie de pequeñas transformaciones que modifican la estructura interna preservando su conducta aparente. La práctica también se conoce como Mejora Continua de Código o Refactorización implacable. Se lo ha parafraseado diciendo: "Si funciona bien, arréglelo de todos modos".

Programación en pares. Todo el código está escrito por pares de programadores. Dos personas escriben código en una computadora, turnándose en el uso del ratón y el teclado.

El que no está escribiendo, piensa desde un punto de vista más estratégico y realiza lo que podría llamarse revisión de código en tiempo real. Los roles pueden cambiarse varias veces al día. Esta práctica no es en absoluto nueva. Hay antecedentes de programación en pares anteriores a XP [Hig00b].

Propiedad colectiva del código. Cualquiera puede cambiar cualquier parte del código en cualquier momento, siempre que escriba antes la prueba correspondiente.

Integración continúa. Cada pieza se integra a la base de código apenas está lista, varias veces al día. Debe correrse la prueba antes y después de la integración. Hay una máquina (solamente) dedicada a este propósito.

Ritmo sostenible, trabajando un máximo de 8 horas por día. Esta práctica se llama semana de 40 horas. En XP todo el mundo debe irse a casa a las cinco de la tarde. Dado que el desarrollo de software se considera un ejercicio creativo, se estima que hay que estar fresco y descansado para hacerlo eficientemente; con ello se motiva a los participantes, se evita la rotación del personal y se mejora la calidad del producto. Deben minimizarse los héroes y eliminar el "proceso neurótico". Aunque podrían admitirse excepciones, no se permiten dos semanas seguidas de tiempo adicional. Si esto sucede, se lo trata como problema a resolver.
Todo el equipo en el mismo lugar. El cliente debe estar presente y disponible a tiempo completo para el equipo. También se llama El Cliente en el Sitio. Como esto parecía no cumplirse (si el cliente era muy junior no servía para gran cosa, y si era muy senior no deseaba estar allí), se especificó que el representante del cliente debe ser preferentemente un analista. (Tampoco se aclara analista de qué; seguramente se definirá en una próxima versión).
Estándares de codificación. Se deben seguir reglas de codificación y comunicarse a través del código. Un valor apreciado en la práctica, el llamado "código revelador de intenciones". Como en XP rige un cierto purismo de codificación, los comentarios no son bien vistos. Si el código es tan oscuro que necesita comentario, se lo debe reescribir o refactorizar.
Espacio abierto. Es preferible una sala grande con pequeños cubículos o, mejor todavía, sin divisiones. Los pares de programadores deben estar en el centro. En la periferia se ubican las máquinas privadas. En un encuentro de espacio abierto la agenda no se establece verticalmente.
Reglas justas. El equipo tiene sus propias reglas a seguir, pero se pueden cambiar en cualquier momento. En XP se piensa que no existe un proceso que sirva para todos los proyectos; lo que se hace habitualmente es adaptar un conjunto de prácticas simples a las características de cada proyecto. [Beck, 1999].

Los roles de XP son pocos. Un cliente que escribe las historias y las pruebas de aceptación; programadores en pares; verificadores (que ayudan al cliente a desarrollar las pruebas); consultores técnicos; y, como parte del management, un coach o consejero que es la conciencia del grupo, interviene y enseña, y un seguidor de rastros (tracker) que colecta las métricas y avisa cuando hay una estimación alarmante, además de un Gran Jefe. El management es la parte menos satisfactoriamente caracterizada en la bibliografía, como si fuera un mal necesario. Lo importante es que el coach se vea como un facilitador, antes que como quien da las órdenes. Los equipos de XP son típicamente pequeños, de tres a veinte personas, y en general no se cree que su escala se avenga al desarrollo de grandes sistemas de misión crítica. [Highsmith, 2000].

XP posee elementos interesantes que se pudieran combinar con otra metodología perfectamente como lo son: Prueba continua, Propiedad colectiva del código, Integración continua, ritmo sostenible, trabajando un máximo de 8 horas por día, todo el equipo en el mismo lugar y estándares de codificación.
Por otra parte los obstáculos más comunes surgidos en proyectos XP son la "fantasía" [Larman 2004] de pretender que el cliente se quede en el sitio y la resistencia de muchos programadores a trabajar en pares. Craig Larman señala como factores negativos la ausencia de énfasis en la arquitectura durante las primeras iteraciones (no hay arquitectos en XP) y la consiguiente falta de métodos de diseño arquitectónico.

2.2.6 SCRUM

Esta es, después de XP, la metodología ágil mejor conocida y la que otros métodos ágiles recomiendan como complemento, aunque su porción del mercado 3% [Charette, 2004] es más bien modesta.

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle, Scrum no está concebido como método independiente, sino que se promueve como complemento de otras metodologías, incluyendo XP, MSF o RUP. Como método, Scrum enfatiza valores y prácticas de gestión, sin pronunciarse sobre requerimientos, implementación y demás cuestiones técnicas; de allí su deliberada insuficiencia y su complementariedad. Scrum se define como un proceso de gestión y control que implementa técnicas de control de procesos; se lo puede considerar un conjunto de patrones organizacionales. [Schwaber, 1995].

Los valores de Scrum son:

· Equipos auto-dirigidos y auto-organizados. No hay manager que decida, ni otros títulos; la excepción es el Scrum Master que debe ser 50% programador y que resuelve problemas, pero no manda. Los observadores externos pueden observar, pero no interferir ni opinar.

· Una vez elegida una tarea, no se agrega trabajo extra. En caso que se agregue algo, se recomienda quitar alguna otra cosa.

· Encuentros diarios con tres preguntas: ¿Qué has hecho desde el último encuentro?, ¿Qué obstáculos hay para cumplir la meta?, ¿Qué harás antes del próximo encuentro? Se realizan siempre en el mismo lugar, en círculo.

· Iteraciones de treinta días; se admite que sean más frecuentes.

· Demostración a participantes externos al fin de cada iteración.

· Al principio de cada iteración, planeamiento adaptativo guiado por el cliente. [Schwaber, 2002].

Scrum define seis roles:

1. **El Scrum Master.** Interactúa con el cliente y el equipo. Es responsable de asegurarse que el proyecto se lleve a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que progrese según lo previsto. Coordina los encuentros diarios, formula las tres preguntas canónicas y se encarga de eliminar eventuales obstáculos. Debe ser miembro del equipo y trabajar a la par.

Propietario del Proyecto. Es el responsable oficial del proyecto, gestión, control y visibilidad de la lista de acumulación o lista de retraso del producto (product backlog). Es elegido por el Scrum Máster, el cliente y los ejecutivos a cargo. Toma las decisiones finales de las tareas asignadas al registro y convierte sus elementos en rasgos a desarrollar.

3. **Equipo de Scrum.** Tiene autoridad para reorganizarse y definir las acciones necesarias o sugerir remoción de impedimentos.

4. **Cliente.** Participa en las tareas relacionadas con los ítems del registro.

5. **Management.** Está a cargo de las decisiones fundamentales y participa en la definición de los objetivos y requerimientos. Por ejemplo, selecciona al Dueño del Producto, evalúa el progreso y reduce el registro de acumulación junto con el Scrum Máster.

6. **Usuario.** La dimensión del equipo total de Scrum no debería ser superior a diez ingenieros. El número ideal es siete, más o menos dos. Si hay más, lo más recomendable es formar varios equipos. No hay una técnica oficial para coordinar equipos múltiples, pero se han documentado experiencias de hasta 800 miembros, divididos en Scrums de Scrum, definiendo un equipo central que se encarga de la

coordinación, las pruebas cruzadas y la rotación de los miembros. El texto que relata esa experiencia es Agile Software Development Ecosystems, de Jim Highsmith. [Highsmith, 2002b].

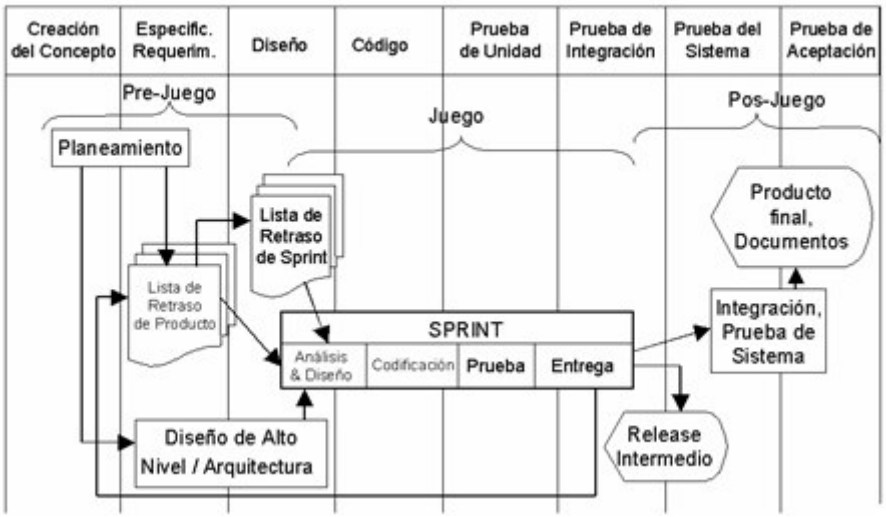


Figura 6. Estructura de Scrum. [Abrahamsson, 2002].

Pre-Juego: Planeamiento. El propósito es establecer la visión, definir expectativas y asegurarse la financiación. Las actividades son la escritura de la visión, el presupuesto, el registro de acumulación o retraso (backlog) del producto inicial y los ítems estimados, así como la arquitectura de alto nivel, el diseño exploratorio y los prototipos. El registro de acumulación es de alto nivel de abstracción.
Pre-Juego: Montaje (Staging). El propósito es identificar más requerimientos y priorizar las tareas para la primera iteración. Las actividades son planificación, diseño exploratorio y prototipos.
Juego o Desarrollo. El propósito es implementar un sistema listo para entrega en una serie de iteraciones de treinta días llamadas "corridas" (sprints). Las actividades son un encuentro de planeamiento de corridas en cada iteración, la definición del registro de acumulación de corridas y los estimados, y encuentros diarios de Scrum.
Pos-Juego: Liberación. El propósito es el despliegue operacional. Las actividades, documentación, entrenamiento, mercadeo y venta

Scrum presenta elementos interesantes que se podrían combinar como: Encuentros diarios, Iteraciones cortas treinta días o más frecuentes, Demostración a participantes externos al fin de cada iteración.
Entre los puntos débiles podemos encontrar que los equipos sean auto-dirigidos y auto-organizados esto podría no resultar en equipos jóvenes y con poca experiencia.

2.2.8 Crystal Methodologies

Los métodos se llaman Crystal evocando las facetas de una gema: cada faceta es otra versión del proceso, y todas se sitúan en torno a un núcleo idéntico. Hay cuatro variantes de metodologías: Crystal Clear ("Claro como el cristal") para equipos de 8 o menos integrantes; Amarillo, para 8 a 20; Naranja, para 20 a 50; Rojo, para 50 a 100. Se promete seguir con Marrón, Azul y Violeta. La más exhaustivamente documentada es Crystal Clear (CC), y es la que se ha de describir a continuación. Como casi todos los otros métodos, CC consiste en valores, técnicas y procesos.

Los siete valores o propiedades de CC [Cockburn, 2002] son:
Entrega frecuente. Consiste en entregar software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del proyecto, pero puede ser diaria, semanal, mensual o lo que fuere. La entrega puede hacerse sin despliegue, si es que se consigue algún usuario que suministre retroalimentación.
Comunicación osmótica. Todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un diseñador senior; eso se llama Experto al Alcance de la Oreja. Una reunión separada para que los concurrentes se concentren mejor es descripta como El Cono del Silencio.
Mejora reflexiva. Tomarse un pequeño tiempo (unas pocas horas cada alguna semana o una vez al mes) para pensar bien qué se está haciendo, cotejar notas, reflexionar, discutir.
Seguridad personal. Hablar cuando algo molesta: decirle amigablemente al manager que la agenda no es realista, o a un colega que su código necesita mejorarse, o que sería conveniente que se bañase más seguido. Esto es importante porque el equipo puede descubrir y reparar sus debilidades. No es provechoso encubrir los desacuerdos con gentileza y conciliación. Técnicamente, estas cuestiones se han caracterizado como una importante variable de confianza y han sido estudiadas con seriedad en la literatura.
Foco. Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo. Lo primero debe venir de la comunicación sobre dirección y prioridades, típicamente con el Patrocinador Ejecutivo. Lo segundo, de un ambiente en que la gente no se vea compelida a hacer otras cosas incompatibles.
Fácil acceso a usuarios expertos. No hay un dogma de vida o muerte sobre esto, como sí lo hay en XP. Un encuentro semanal o semi-semanal con llamados telefónicos adicionales parece ser una buena pauta. Otra variante es que los programadores se entrenen para ser usuarios durante un tiempo. El equipo de desarrollo, de todas maneras, incluye un Experto en Negocios.
7. **Ambiente técnico con prueba automatizada, management de configuración e integración frecuente.** Compilar e integrar varias veces al día.

CC enfatiza el proceso como un conjunto de ciclos anidados. En la mayoría de los proyectos se perciben siete ciclos: (1) el proyecto, (2) el ciclo de entrega de una unidad, (3) la iteración (nótese que CC requiere múltiples entregas por proyecto pero no muchas iteraciones por entrega), (4) la semana laboral, (5) el período de integración, de 30 minutos a tres días, (6) el día de trabajo, (7) el episodio de desarrollo de una sección de código, de pocos minutos a pocas horas.

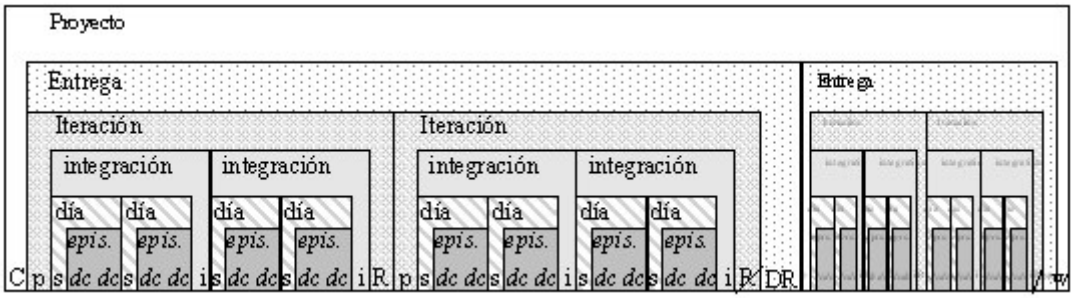


Figura 7. Estructura de CC. [Cockburn, 2002].

Hay ocho roles nominados en CC: Patrocinador, Usuario Experto, Diseñador Principal, Diseñador-Programador, Experto en Negocios, Coordinador, Verificador, Escritor.

Patrocinador. Produce la Declaración de Misión con Prioridades de Compromiso. Consigue los recursos y define la totalidad del proyecto.
Usuario Experto. Junto con el Experto en Negocios produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación, etcétera.
Diseñador Principal. Produce la Descripción Arquitectónica. El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.
Diseñador-Programador. Produce, junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de Migración, las Pruebas y el Sistema Empaquetado. Cockburn no distingue entre diseñadores y programadores. Un programa en CC es "diseño y programa"; sus programadores son diseñadores-programadores. En CC un diseñador que no programe no tiene cabida.
Experto en Negocios. Junto con el Usuario Experto produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.
Coordinador. Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, la Lista de Riesgos, el Plan y Estado de Iteración y la Agenda de Visualización.
Verificador. Produce el Reporte de Bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
Escritor. Produce el Manual de Usuario. [Cockburn, 2002]

Los métodos Crystal no prescriben las prácticas de desarrollo, las herramientas o los productos que pueden usarse, pudiendo combinarse con otros métodos como Scrum, XP y Microsoft Solutions Framework. En su comentario a [Highsmith, 2000] Cockburn confiesa que cuando imaginó CC pensaba proporcionar un método ligero; comparado con XP, sin embargo, CC resulta muy pesado.
También propone una serie de principios que pondrían combinarse: Mejora reflexiva, Seguridad personal, Foco, Programación lado a lado.

2.2.9 Feature-Driven Development (FDD)

FDD, en cambio, es un método ágil, iterativo y adaptativo. A diferencia de otras MAs, no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica. FDD no requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso. Sus impulsores son Jeff De Luca y Peter Coad.
Hay tres categorías de rol en FDD: roles claves, roles de soporte y roles adicionales. Los seis roles claves de un proyecto son: (1) administrador del proyecto, quien tiene la última palabra en materia de visión, cronograma y asignación del personal; (2) arquitecto jefe (puede dividirse en arquitecto de dominio y arquitecto técnico); (3) manager de desarrollo, que puede combinarse con arquitecto jefe o manager de proyecto; (4) programador jefe, que participa en el análisis del requerimiento y selecciona rasgos del conjunto a desarrollar en la siguiente iteración; (5) propietarios de clases, que trabajan bajo la guía del programador jefe en diseño, codificación, prueba y documentación, repartidos por rasgos y (6) experto de dominio, que puede ser un cliente, patrocinador, analista de negocios o una mezcla de todo eso.

Los cinco roles de soporte comprenden (1) administrador de entrega, que controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él; reporta al manager del proyecto; (2) abogado/gurú de lenguaje, que conoce a la perfección el lenguaje y la tecnología; (3) ingeniero de construcción, que se encarga del control de versiones y publica la documentación; (4) herramientista, que construye herramientas o mantiene bases de datos y sitios Web y (5) administrador del sistema, que controla el ambiente de trabajo.
Los tres roles adicionales son los de verificadores, encargados del despliegue y escritores técnicos. Un miembro de un equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas. [Coad, 2000].

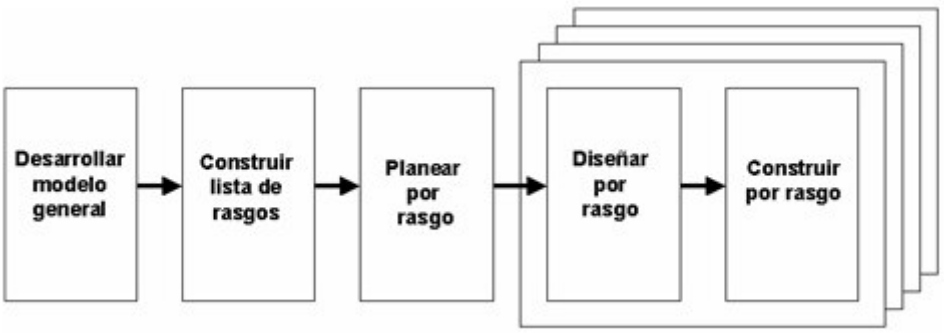


Figura 8. Ciclo de FDD. [Abrahamsson, 2002].

Desarrollo de un modelo general. Cuando comienza este desarrollo, los expertos de dominio ya están al tanto de la visión, el contexto y los requerimientos del sistema a construir. Los expertos de dominio presentan un ensayo en el que los miembros del equipo y el arquitecto principal se informan de la descripción de alto nivel del sistema. El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego de cada ensayo, un equipo de desarrollo trabaja en pequeños grupos para producir modelos de objeto de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema
Construcción de la lista de rasgos. Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos. Los rasgos son pequeños ítems útiles a los ojos del cliente. Se escriben en un lenguaje que todas las partes puedan entender y las funciones se agrupan conforme a diversas actividades en áreas de dominio específicas. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad. Los rasgos que requieran más de diez días se descomponen en otros más pequeños.
Planeamiento por rasgo. Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia, y se asigna a los programadores jefes. Las listas se priorizan en secciones que se llaman paquetes de diseño. Luego se asignan las clases definidas en la selección del modelo general a programadores individuales, o sea propietarios de clases. Se pone fecha para los conjuntos de rasgos.
Diseño por rasgo y Construcción por rasgo. Se selecciona un pequeño conjunto de rasgo del conjunto y los propietarios de clases seleccionan los correspondientes equipos dispuestos por rasgos. Se procede luego iterativamente hasta que se producen los rasgos seleccionados. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. Puede haber varios grupos trabajando en paralelo. El proceso iterativo incluye inspección de diseño, codificación, prueba de unidad, integración e inspección de código. Luego de una iteración exitosa, los rasgos completos se promueven a la

integración principal. Este proceso puede demorar una o dos semanas en implementarse. [Abrahamsson, 2002].
Entre las críticas principales que se le realizan a FDD es que es demasiado jerárquico para ser un método ágil, porque demanda un programador jefe, quien dirige a los propietarios de clases, quienes dirigen equipos. Otro rasgo llamativo de FDD es que no exige la presencia del cliente.

2.2.10 Dynamic Systems Development Method (DSDM)

Originado en los trabajos de Jennifer Stapleton, directora del DSDM Consortium, DSDM se ha convertido en el framework de desarrollo rápido de aplicaciones (RAD) más popular de Gran Bretaña. [Stapleton, 1997]. DSDM puede complementar metodologías de XP, RUP o Microsoft Solutions Framework, o combinaciones de todas ellas. Se dice que ahora las iniciales DSDM significan Dynamic Solutions Delivery Method. Ya no se habla de sistemas sino de soluciones, y en lugar de priorizar el desarrollo se prefiere enfatizar la entrega. El libro más reciente que sintetiza las prácticas se llama DSDM: Business Focused Development. [DSDM, 2003].
DSDM consiste en cinco fases:
1. Estudio de viabilidad.
2. Estudio del negocio.
3. Iteración del modelo funcional.
4. Iteración de diseño y versión.
5. Implementación.

Las últimas tres fases son iterativas e incrementales. De acuerdo con la iniciativa de mantener el tiempo constante, las iteraciones de DSDM son cajas de tiempo. La iteración acaba cuando el tiempo se consume. Se supone que al cabo de la iteración los resultados están garantizados. Una caja de tiempo puede durar de unos pocos días a unas pocas semanas.

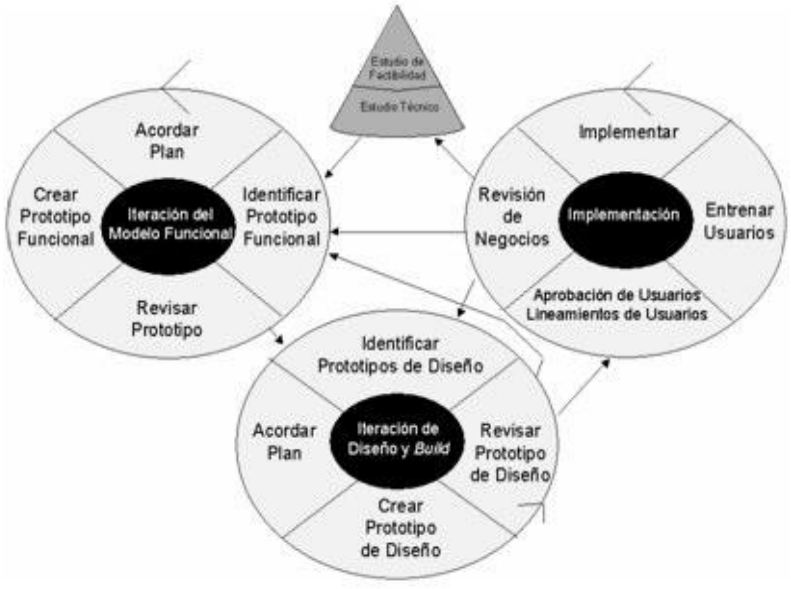


Figura 9. Proceso de desarrollo de DSDM.

Estudio de factibilidad. Se evalúa el uso de DSDM o de otra metodología conforme al tipo de proyecto, variables organizacionales y de personal.
Estudio del negocio. Se analizan las características del negocio y la tecnología. Se describen los procesos de negocio y las clases de usuario en una Definición del Área de

Negocios. La Definición utiliza descripciones de alto nivel, como diagramas de entidad-relación o modelos de objetos de negocios. Otros productos son la Definición de Arquitectura del Sistema y el Plan de Bosquejo de Prototipado.
Iteración del modelo funcional. En cada iteración se planea el contenido y la estrategia, se realiza la iteración y se analizan los resultados pensando en las siguientes.
Iteración de diseño y construcción. Aquí es donde se construye la mayor parte del sistema. El producto es un Sistema Probado que cumplimenta por lo menos el conjunto mínimo de requerimientos acordados. El diseño y la construcción son iterativos y el diseño y los prototipos funcionales son revisados por usuarios.
Despliegue. El sistema se transfiere del ambiente de desarrollo al de producción. Se entrena a los usuarios, que ponen las manos en el sistema [DSDM, 2003].

DSDM define quince roles, los más importantes son:
Coordinador técnico. Define la arquitectura del sistema y es responsable por la calidad técnica del proyecto, el control técnico y la configuración del sistema.
Usuario embajador. Proporciona al proyecto conocimiento de la comunidad de usuarios y disemina información sobre el progreso del sistema hacia otros usuarios. Se define adicionalmente un rol de Usuario Asesor (Advisor) que representa otros puntos de vista importantes; puede ser alguien del personal de IT o un auditor funcional.
Visionario. Es un usuario participante que tiene la percepción más exacta de los objetivos del sistema y el proyecto. Asegura que los requerimientos esenciales se cumplan y que el proyecto vaya en la dirección adecuada desde el punto de vista de aquéllos.
Patrocinador Ejecutivo. Es la persona de la organización que detenta autoridad y responsabilidad financiera, y es quien tiene la última palabra en las decisiones importantes.
Facilitador. Es responsable de administrar el progreso del taller y el motor de la preparación y la comunicación.
Escriba. Registra los requerimientos, acuerdos y decisiones alcanzadas en las reuniones, talleres y sesiones de prototipado. [Stapleton, 1997].

2.2.11 Adaptive Software Development (ASD)

James Highsmith, consultor de Cutter Consortium, desarrolló ASD hacia el año 2000 la cual presupone que las necesidades del cliente son siempre cambiantes. La iniciación de un proyecto involucra definir una misión para él, determinar las características y las fechas y descomponer el proyecto en una serie de pasos individuales, cada uno de los cuales puede abarcar entre cuatro y ocho semanas. Los pasos iniciales deben verificar el alcance del proyecto; los tardíos tienen que ver con el diseño de una arquitectura, la construcción del código, la ejecución de las pruebas finales y el despliegue. [Highsmith, 2000].
Aspectos claves de ASD son:
1. Un conjunto no estándar de "artefactos de misión" incluyendo una visión del proyecto, una hoja de datos, un perfil de misión del producto y un esquema de su especificación
2. Un ciclo de vida, inherentemente iterativo.
3. Cajas de tiempo, con ciclos cortos de entrega orientados por riesgo.

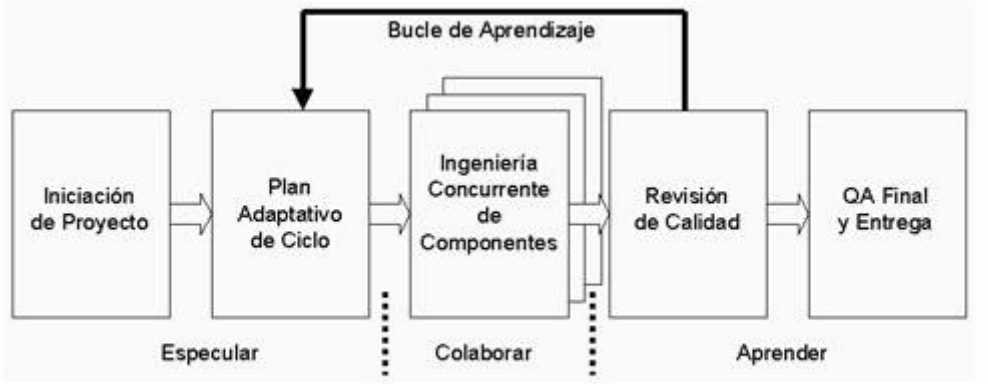


Figura 10. Fases del ciclo de vida de ASD. [Highsmith, 2000].

Un ciclo de vida es una iteración; este ciclo se basa en componentes y no en tareas, es limitado en el tiempo, orientado por riesgos y tolerante al cambio. Que se base en componentes implica concentrarse en el desarrollo de software que trabaje, construyendo el sistema pieza por pieza.
En este paradigma, el cambio es bienvenido y necesario, pues se concibe como la oportunidad de aprender y ganar así una ventaja competitiva; de ningún modo es algo que pueda ir en detrimento del proceso y sus resultados. [Riehle, 2000].
La idea subyacente a ASD (y de ahí su particularidad) radica en que no proporciona un método para el desarrollo de software sino que más bien suministra la forma de implementar una cultura adaptativa en la empresa, con capacidad para reconocer que la incertidumbre y el cambio son el estado natural. El problema inicial es que la empresa no sabe que no sabe, y por tal razón debe aprender.

2.2.12 Lean Development (LD)

Definida por Bob Charette's a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. LD se inspira en doce valores centrados en estrategias de gestión. [Highsmith, 2002b]:
1. Satisfacer al cliente es la máxima prioridad.
2. Proporcionar siempre el mejor valor por la inversión.
3. El éxito depende de la activa participación del cliente.
4. Cada proyecto LD es un esfuerzo de equipo.
5. Todo se puede cambiar.
6. Soluciones de dominio, no puntos.
7. Completar, no construir.
8. Una solución al 80% hoy, en vez de una al 100% mañana.
9. El minimalismo es esencial.
10. La necesidad determina la tecnología.
11. El crecimiento del producto es el incremento de sus prestaciones, no de su tamaño.
12. Nunca empujes LD más allá de sus límites.

Dado que LD es más una filosofía de management que un proceso de desarrollo no hay mucho que decir del tamaño del equipo, la duración de las iteraciones, los roles o la naturaleza de sus etapas. Últimamente LD ha evolucionado como Lean Software Development (LSD); su figura de referencia es Mary Poppendieck. [Poppendieck, 2001].

Los valores de LD se han reformulado a LSD manteniendo los siguientes puntos de coincidencia:
1. Eliminar basura – Entre la basura se cuentan diagramas y modelos que no agregan valor al producto.
2. Minimizar inventario – Igualmente, suprimir artefactos tales como documentos de requerimiento y diseño.
3. Maximizar el flujo – Utilizar desarrollo iterativo.
4. Solicitar demanda – Soportar requerimientos flexibles.
5. Otorgar poder a los trabajadores.
6. Satisfacer los requerimientos del cliente – Trabajar junto a él, permitiéndole cambiar de ideas.
7. Hacerlo bien la primera vez – Verificar temprano y refactorizar cuando sea preciso.
8. Abolir la optimización local – Alcance de gestión flexible.
9. Asociarse con quienes suministran – Evitar relaciones de adversidad.
10. Crear una cultura de mejora continua. [Poppendieck, 2001]

LD se encuentra hoy en América del Norte en una situación similar a la de DSDM en Gran Bretaña, llegando al 7% entre los MAs a nivel mundial (algo menos que Crystal pero el doble que Scrum). Existen abundantes casos de éxito documentados empleando LD y LSD, la mayoría en Canadá.

2.2.13 Agile Modeling (AM)

Agile Modeling (AM) fue propuesto por Scott Ambler no tanto como un método ágil cerrado en sí mismo, sino como complemento de otras metodologías, sean éstas ágiles o convencionales. Ambler recomienda su uso con XP, Microsoft Solutions Framework, RUP o EUP.
AM es una estrategia de modelado (de clases, de datos, de procesos) pensada para contrarrestar la sospecha de que los métodos ágiles no modelan y no documentan. Se lo podría definir como un proceso de software basado en prácticas cuyo objetivo es orientar el modelado de una manera efectiva y ágil.
Los principios de AM especificados por Ambler incluyen:
1. **Presuponer simplicidad.** La solución más simple es la mejor.
2. **El contenido es más importante que la representación**. Pueden ser notas, pizarras o documentos formales. Lo que importa no es el soporte físico o la técnica de representación, sino el contenido.
3. **Abrazar el cambio.** Aceptar que los requerimientos cambian.
4. **Habilitar el esfuerzo siguiente.** Garantizar que el sistema es suficientemente robusto para admitir mejoras ulteriores; debe ser un objetivo, pero no el primordial.
5. **Todo el mundo puede aprender de algún otro**. Reconocer que nunca se domina realmente algo.
6. **Cambio incremental**. No esperar hacerlo bien la primera vez.
7. **Conocer tus modelos.** Saber cuáles son sus fuerzas y sus debilidades.
8. **Adaptación local**. Producir sólo el modelo que resulte suficiente para el propósito.
9. **Maximizar la inversión del cliente.**
10. **Modelar con un propósito.** Si no se puede identificar para qué se está haciendo algo ¿para qué molestarse?
11. **Modelos múltiples.** Múltiples paradigmas en convivencia, según se requiera.
12. **Comunicación abierta y honesta.**
13. **Trabajo de calidad.**
14. **Realimentación rápida.** No esperar que sea demasiado tarde.

15. **El software es el objetivo primario.** Debe ser de alta calidad y coincidir con lo que el usuario espera.
16. **Viajar ligero de equipaje.** No crear más modelos de los necesarios.
17. **Trabajar con los instintos de la gente**. [Ambler, 2002]

Como AM se debe usar como complemento de otras metodologías, nada se especifica sobre métodos de desarrollo, tamaño del equipo, roles, duración de iteraciones, trabajo distribuido y criticalidad, todo lo cual dependerá del método que se utilice.
Procesos como EUP son simplemente UP+AM. Algunos de estos principios y prácticas aunque no se piense utilizar una MAs, pueden ser utilizados para agilizar el desarrollo.

3. Metodologías de desarrollo en la Universidad de las Ciencias Informáticas (UCI)

3.1 Metodología ágil Malay Rodríguez Villar revisión 1 (MA-MRV-R1)

En el año 2007 se hizo en la Facultad 7, de la Universidad de las Ciencias Informáticas, un estudio basado en las principales características de los proyectos productivos, pues estaban presentando problemas durante el desarrollo, razón por la que se le otorgó como tesis de grado a la estudiante Malay Rodríguez Villar la "Introducción de procedimientos ágiles en la producción de software en la Facultad 7 de la Universidad de las Ciencias Informáticas". El resultado de dicha tesis fue una propuesta de la utilización de la metodología ágil SCRUM, para la planificación de los proyectos que usaran métodos ágiles como proceso de desarrollo y para el desarrollo las mejores prácticas de XP, según la autora, procurando que el proceso se realizara de manera efectiva y eficiente. [Rodríguez, 2007]
SCRUM puede ser visto como una interfaz entre el equipo y los clientes. SCRUM deja un agujero en el medio, ejemplo: como el equipo debe hacer su trabajo diario, en el cual XP encaja muy bien. Una vez que SCRUM es puesto en práctica el equipo es auto administrado y puede escoger como hacer el trabajo. El desarrollo de software ágil es todo sobre círculos de retroalimentación. SCRUM y XP se complementan el uno al otro. Como se muestra en la figura anterior, (Amarillo –XP, Verde –SCRUM) [Kniberg, 2007].

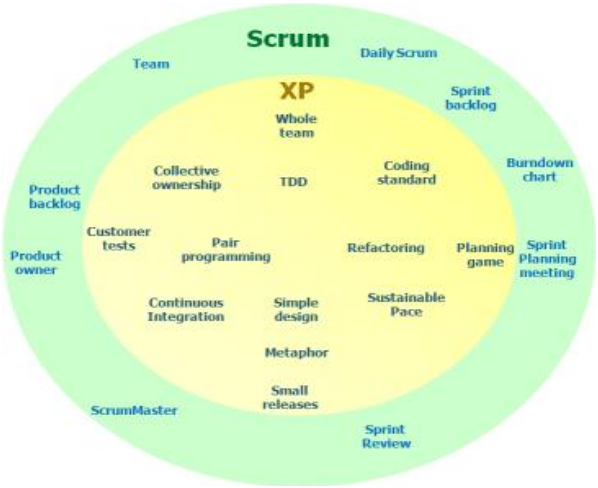


Figura 11. Unión de SCRUM + XP.

A continuación la propuesta [Rodríguez, 2007]:

Planificación ↔ Definición

- Entrevista con el cliente (concepción inicial).

- Juego de la planificación.
- Captura de requisitos:
    - Creación de la Lista de Reserva del Producto (LRP).
    - Priorización de la LRP.
    - Definir las historias de usuario.
    - Asignar las tareas de las historias de usuario.
- Valoración del esfuerzo.
- Diseño con las metáforas.
- Creación de las tarjetas CRC.
- Refactorización (eliminar complejidad, diseñar lo más simple que se pueda).
- Reunión de la revisión del diseño.

Desarrollo

- Junta de planificación.
    - Definir las historias de usuarios a implementar.
    - Tareas para lograr dicha implementación.
- Fase de programación:
    - Programación en pareja.
    - Propiedad colectiva.
    - Integración continúa.
    - Estándares de programación.
    - Cliente en sitio.
- Junta de seguimiento.
- Junta de revisión.
- Pruebas:
    - Unitarias.
    - Autorizadas.

Entrega

- Entrega de la documentación.
- Entrenamiento.
- Marketing.

Mantenimiento

- Soporte. [Rodríguez, 2007]

3.2 Metodología ágil Gladys Marsi Peñalver Romero Unicornios revisión 2 (MA-GMPR-UR2)

En el año 2008 en la Facultad 10, de la Universidad de las Ciencias Informáticas, se tomo la decisión de aplicar la propuesta MA-MRV-R1 en proyectos de software libre, en particular en el proyecto UNICORNIOS, razón por la que se le otorgó como tesis de grado a la estudiante Gladys Marsi Peñalver Romero el tema "Metodología ágil para proyectos de software libre." Como resultado de esta tesis se aplicó la propuesta MA-MRV-R1 en tres iteraciones de dos meses cada una en seis proyectos productivos y se obtuvieron con esta prueba las deficiencias y dificultades en el desarrollo de los mismos así como la comprobación de la necesidad de incorporar nuevos artefactos a la propuesta. La propuesta MA-GMPR-UR2 mantiene la utilización de la metodología ágil SCRUM, para la planificación de los proyectos y para el desarrollo las mejores prácticas de XP [Peñalver, 2008]. A continuación la propuesta [Peñalver, 2008]:

Planificación ↔ Definición

- Entrevista con el cliente (concepción inicial).

- Plantilla Concepción del sistema

- Juego de la planificación.
  - Plantilla de Plan de Releases
- Captura de requisitos:
  - Creación de la Lista de Reserva del Producto (LRP).
  - Priorización de la LRP.
  - Definir las historias de usuario.
    - Plantilla Modelo de Historia de Usuario del negocio
  - Asignar las tareas de las historias de usuario.
- Valoración del esfuerzo.
- Valoración de riesgos
  - Plantilla Lista de riesgos
- Diseño simple
  - Plantilla Modelo de diseño
- Diseño con las metáforas.
- Refactorización
- Refactorización (eliminar complejidad, diseñar lo más simple que se pueda).
- Reunión de la revisión del diseño.

Desarrollo

- Junta de planificación.
  - Definir las historias de usuarios a implementar.
  - Tareas para lograr dicha implementación.
    - Plantilla de Tareas de Ingeniería
    - Plantilla Cronograma de producción
- Fase de programación:
  - Programación en pareja.
  - Propiedad colectiva.
  - Integración continúa.
  - Estándares de programación.
    - Plantilla de Estándares de programación
  - Cliente en sitio.
- Junta de seguimiento.
- Taller técnico
- Junta de revisión.
- Pruebas:
  - Unitarias.
    - Plantilla Caso de Prueba de aceptación
  - Autorizadas.

Entrega

- Entrega de la documentación.
- Entrenamiento.
  - Plantilla Manual de usuario.
  - Plantilla Manual de Identidad.
  - Plantilla Manual de desarrollo.
- Instalación
- Marketing.

Mantenimiento

- Soporte.
  - Plantilla de Gestión de Cambios

3.3 Procedimiento para la Producción de Software de Gestión

En el año 2009 en la Universidad de las Ciencias Informáticas, se desarrolló una investigación como tesis de pregrado por las estudiantes Geidys Álvarez Llanes y Lilybeth Rodríguez García cuyo título fue "Procedimiento para la Producción de Software de Gestión". El resultado de dicha tesis fue una propuesta de la utilización de las metodologías más usadas en la UCI XP y El Proceso Unificado del Desarrollo del Software y de ellas lo mejor que propone cada una.
Del PU se tomaron sus cuatro fases de desarrollo: inicio, elaboración, construcción y transición con sus respectivos objetivos, así como también los nueve flujos de trabajo, dentro de los mismos seis de desarrollo y tres de soporte, además los artefactos, trabajadores y actividades. De la metodología Programación Extrema se tomaron las características principales en la que se basa: pruebas unitarias, re-fabricación y programación en pares, así como lo que tiene establecido para los derechos del cliente y del desarrollador. A continuación la propuesta, las etapas y los artefactos:

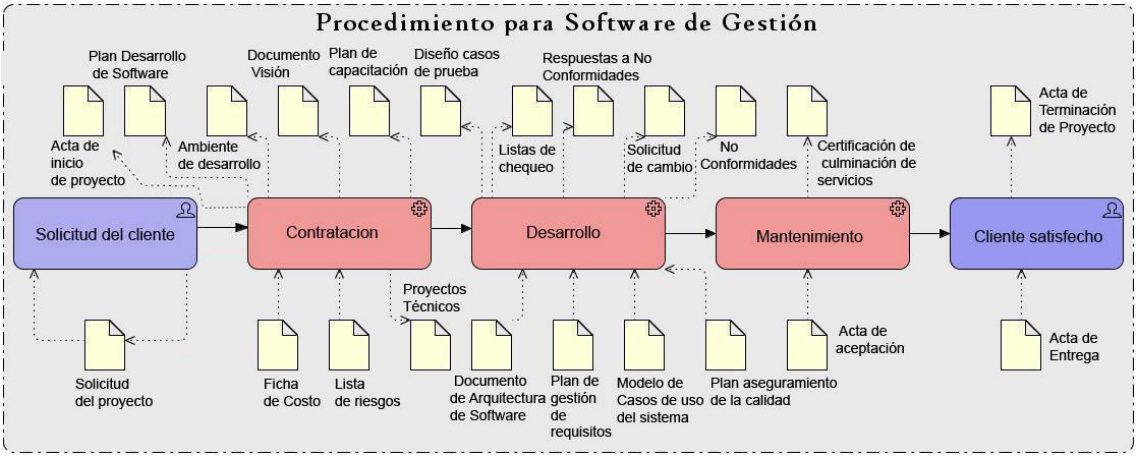


Figura 12. Procedimiento para software de gestión

Solicitud del Cliente

- Solicitud del proyecto.
- Cronograma general.

Etapa de Contratación

- Contrato o acuerdo de colaboración.
- Acta de inicio de proyecto.
- Indefiniciones.
- Proyectos Técnicos.
- Plan Desarrollo de Software.
- Presupuesto.
- Ficha de Costo.
- Lista de riesgos.
- Plan Mitigación de Riesgos.
- Ambiente de desarrollo.
- Plan de capacitación.
- Documento Visión.
- Glosario de términos.
- Informe del Levantamiento de Información para la Arquitectura de Información.

Etapa de Desarrollo

- Arquitectura de Información.
- Documento de Arquitectura de Software.
- Especificación de requisitos.
- Modelo de Casos de uso del sistema.
- Modelo del Negocio.
- Plan de gestión de requisitos.
- Diseño casos de prueba.
- Modelo de Despliegue.
- Glosario de términos
- Listas de chequeo.
- Plan aseguramiento de la calidad.
- No Conformidades.
- Respuestas a No Conformidades.
- Solicitud de cambio.

Etapa de Mantenimiento

- Acta de aceptación.
- Certificación de culminación de servicios.

Gestión de la Configuración

- Pedido de cambio.
- Plan de gestión de la configuración.
- Establecimiento de las políticas de gestión de configuración y salvas.

Etapa de Culminación de Servicios. Cliente Satisfecho

- Acta de Aceptación.
- Acta de Entrega.
- Acta de Terminación de Proyecto.

Las metodologías de desarrollo utilizadas en los proyectos que se desarrollan en la UCI son: RUP en el 63% de los casos, XP en un 2%, Scrum en el 1%, OpenUp en un 7% y el restante 27% utiliza otras metodologías. La figura 3 muestra este comportamiento. [Diagnóstico, 2008].

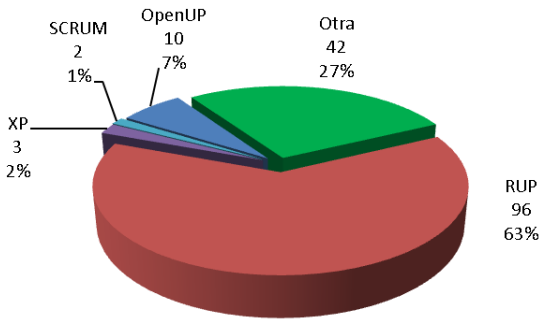


Figura 13. Metodologías de desarrollo utilizadas por los proyectos. [Diagnóstico, 2008].

4. *Características de la UCID*

4.1 Estructurales

Gestión y Soporte: encierra la gestión centralizada de la producción a partir de una Oficina de Gestión de Proyectos (OGP), el área de soporte a las aplicaciones en despliegue y el aseguramiento y gestión de la calidad.
Producción: en las áreas productivas se encuentran los centros de desarrollo, los cuales tienen la misión del desarrollo y compatibilización de productos informáticos.
Aseguramiento a la Producción: incluye el desarrollo y asimilación de tecnologías para ser empleadas en la producción (Marcos de Trabajo, Arquitectura y Patrones, Estándares) y la gestión de la infraestructura de producción (Centro de Datos, Gestión de Configuración).

Figura 14. Estructuras centrales de dirección, gestión y producción.

A su vez los centros temáticos dividen el trabajo en **Líneas de producción de software**, las cuales surgen por la necesidad de concentrar los recursos y esfuerzos en el desarrollo de un conjunto de proyectos que comparten características y satisfacen las necesidades específicas de un propósito. En ella se materializan los principales aspectos prácticos orientados a la construcción del producto: los principales elementos de gestión, el aseguramiento tecnológico, el diseño y desarrollo de los componentes horizontales del dominio de solución.

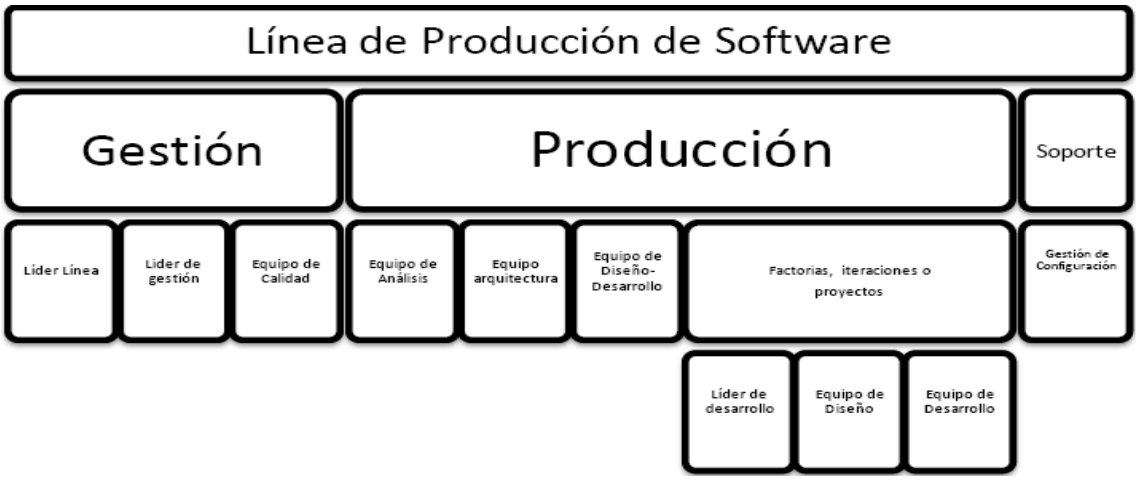


Figura 15. Estructura de la línea de producción de software.

El proyecto es desarrollado como resultado del trabajo conjunto de los equipos centrales de la línea y las factorías. Las factorías están compuestas por dos equipos: un equipo de diseño y un equipo de desarrollo. Está dirigida por un Líder de factoría y se ocupa de la implementación de los componentes que se definan a nivel de línea según la especialización de las mismas. La cantidad de factorías se define en dependencia del contenido de trabajo de la línea a la cual pertenecen.

Oficina de Gestión de Proyectos

La Oficina de Gestión de Proyectos (OGP) centraliza y coordina la dirección de todos los proyectos que se ejecutan en la UCID, hace énfasis en la planificación, la priorización y controla la ejecución de los mismos. La OGP se crea por la demanda de una gerencia eficaz, el aumento del número de proyectos, así como la creciente complejidad de los mismos; a esto se suma la necesidad de centralizar la información y gestión de todos los proyectos que se realizan paralelamente, además de crear una atmósfera positiva y respaldar a la jefatura del centro. Las responsabilidades de la OGP se resumen de la siguiente manera:

1. Orientación metodológica de los procesos de gestión y desarrollo de software.
2. Gestión centralizada de los recursos de los proyectos.
3. Inicio y cierre del proyecto.
4. Gestión centralizada de cronogramas.
5. Elaboración de la información estadística de los proyectos.
6. Gestión centralizada de riesgos.
7. Gestión del Sistema de Ciencia e Innovación Tecnológica

Figura 16. Estructura de la oficina de gestión de proyectos.

1. Área de Gestión de Equipo de desarrollo.
    - Gestión centralizada de los recursos de los proyectos.
2. Área de Seguimiento y control de los proyectos.
    - Inicio y cierre del proyecto
    - Gestión de centralizada de cronogramas.
    - Gestión centralizada de riesgos.
3. Área de Atención metodológica.
    - Orientación metodológica de los procesos de gestión y desarrollo de software.
    - Elaboración de la información estadística de los proyectos.

- Gestión del Sistema de Ciencia e Innovación Tecnológica.

4.2 Características de los equipos de desarrollo

Debilidades

- Los equipos de desarrollo están compuestos y son dirigidos en su mayoría por personal con pocos o ningún año de experiencia, las edades oscilan entre los 23 y los 28 años, exceptuando a la jefatura de la UCID.
- Están complementados además con estudiantes desde 2do hasta 5to año.
- La categoría científica es muy baja, no existen másteres ni doctores.
- La permanencia de los civiles que representan el % de la UCID es muy baja solo de 1 año para los hombres y tres las mujeres.

Fortalezas

- Todos los profesionales son miembros de las FAR comprometidos con la revolución, la sociedad y con la entidad, la mayoría militantes de la UJC o del PCC con una alta preparación política e ideológica.
- La ubicación de la UCID dentro de la UCI nos da la posibilidad de la integración con los distintos centros de la misma.

4.3 Características de los proyectos

- Son proyectos complejos que combinan dentro de si distintos tipos como: TPS, SSD, SIG, etc.
- El tiempo de respuesta debe ser corto entre 2 y 3 meses.
- Deben tener un elevado nivel de calidad porque se ejecutan principalmente para los jefes de las FAR.
- Los usuarios no se encuentran a tiempo completo con el equipo de desarrollo.
- Tiene que ser desarrollados para que estén en explotación el mayor tiempo posible.
- Tienen que permitir que les sean realizadas adaptaciones o modificaciones de una manera fácil.
- Tienen que estar documentados y ejecutarse en el tiempo previsto o adelantarse.
- Tienen que permitir la integración con otros proyectos.

4.4 Problemas detectados en el desarrollo de software

En la encuesta realizada a los líderes de los proyectos de la UCID (ver anexo 1), que desarrollaron los primeros productos con una adaptación de la metodología RUP, estos se refieren a que los principales problemas detectados fueron:

- El proceso de desarrollo no estaba bien definido ya que RUP te dice que hacer pero no como hacerlo.
- Todos los proyectos realizaban interpretaciones diferentes, no existían estándares ni quien rectorara esa actividad.
- No podían satisfacer la mayoría de las actividades y los artefactos que propone la metodología, además de que era prácticamente imposible mantener los modelos y diagramas actualizados, y que algunos de los artefactos realizados no

eran realmente útiles dedicando mucho tiempo a desarrollarlos y la utilidad luego era muy poca.

- El tiempo de desarrollo siempre demoraba más de lo planificado trayendo como consecuencias perdida de la confianza de los clientes.

Finalmente cuando se obtuvieron los primeros resultados aparecieron otros problemas:

- Numerosas solicitudes de cambio por diferentes motivos:
    a. Insuficiente entendimiento del negocio
    b. Errores introducidos por malas interpretaciones de diagramas.
- Poca calidad del producto
- Desorganización en la entrega de los productos por no estar bien preparada y coordinada la fase de transición.

En la mayoría de los proyectos hubo que implementar nuevas funcionalidades o reprogramar otras que ya existían lo cual trajo como consecuencia un aumento en el tiempo de desarrollo y por lo tanto atraso de la fase de pruebas trayendo insatisfacciones a el cliente y perdida de la confianza en el equipo de desarrollo.

Conclusiones

- Las metodologías juegan un papel importante en la organización y control del desarrollo de software por lo tanto seleccionar una que se adecúe a las características de la entidad y los proyectos que desarrolla, analizarla y adaptarla de ser necesario, contribuye inequívocamente, a mejorar la producción de software en dicha entidad.
- Existen una gran cantidad de metodologías en el contexto internacional divididas en dos grandes grupos: tradicionales y ágiles. Las primeras centradas en el proceso, su control y planificación. La segunda apostando por las personas e intentado cambiar los paradigmas de la ingeniería del software. Decidirse por alguna de ellas es una tarea difícil.
- En la Universidad la mayoría de los proyectos de desarrollo de software utilizan todavía las metodologías tradicionales aunque ya existen otros que están apostando por las ágiles. Algunos especialistas plantean que la combinación de tradicionales con ágiles es una buena alternativa mientras que otros han ido mas allá planteando que en el futuro no existirán ni unas ni otras y solo existirán paletas de prácticas y principios de las cuales los desarrolladores podrán tomar y utilizar las que quieran.

Capítulo 2. Solución propuesta

Introducción

El siguiente capítulo describe la solución propuesta para resolver el problema planteado. La misma se basa en combinar algunas de las metodologías de desarrollo, utilizando de cada una lo que más se adecúe a las necesidades de la UCID, teniendo en cuenta también las prácticas y los principios que contribuyan a agilizar el proceso sin poner en riesgo la calidad del mismo.

1. *Objetivo*

Identificar las fases en las que se dividirá el ciclo de vida del proyecto en las que se define qué trabajo técnico se hará en cada una, cuándo se generarán los entregables, cómo se revisarán, verificarán y validarán, quién está involucrado y cómo controlar y aprobar cada una. Definir los flujos de trabajo y dentro de cada uno las actividades, los roles con sus responsabilidades y los artefactos (Ver anexo II).

2. *Fases*

Las fases permiten mejorar el planeamiento, ejecución y control del proyecto, se caracterizan por la conclusión y la aprobación de uno o más artefactos entregables que servirán de entrada a la próxima fase o incluso para decidir la continuidad del proyecto. La consecución exitosa de cada fase es indispensable para poder continuar adelante.

- Se propone utilizar una pre-fase como la que presenta EUP donde se puedan desarrollar actividades que permitan aprovechar mejor los recursos y maximizar los resultados, haciendo más efectiva la gestión de los proyectos y el uso de los recursos que se utilizan para el desarrollo. Permitiendo que la UCID realice una buena ejecución de su estrategia tecnológica [Medellín, 2006].
- Se propone utilizar una fase de inicio con objetivos similares a los que presenta la propuesta en el Proceso Unificado de Desarrollo pero se modifican las actividades y los artefactos que se generan en ella y se le incorpora un hito de validación de los mismos por un consejo técnico.
- Se propone utilizar una fase de modelación que aunque presenta objetivos similares a la de Elaboración, propuesta en el Proceso Unificado de Desarrollo, como son la especificación del sistema, su diseño y la construcción de la arquitectura se proponen otras actividades y los artefactos generados son diferentes. Entre los principales cambios se encuentran la utilización de la Business Process Modeling Notation (BPMN) para modelar los procesos de negocio [Weske, 2007], lo cual mejora la comunicación con usuarios que no son especialistas y que no están con el equipo de desarrollo; las especificaciones de requisitos de software (ERS)(ver anexo XXII) para describir el funcionamiento del sistema, las cuáles son ágiles para describir; y la utilización de artefactos que se corresponden con una arquitectura basada en componentes, que permite una mayor reutilización e integración con otros sistemas. Se incorpora también un hito de validación de los artefactos por el consejo técnico.
- Se propone utilizar una fase de construcción que presenta objetivos similares a la propuesta en el Proceso Unificado de Desarrollo: codificación y documentación del sistema, pero los artefactos que recibe como entrada son diferentes y se introdujo una actividad de liberación ejecutada por el centro de calidad de la UCID para obtener una versión beta del sistema, lista para ser entregada a los usuarios. Se incorpora además un hito de validación de los artefactos por el consejo técnico.

- Se propone utilizar una fase de explotación experimental con objetivos similares a la fase de estabilización propuesta en MSF, los de estabilizar la versión beta entregada, pero se modifican las actividades y los artefactos de entrada y salida. Esta fase se centra en probar el producto en una entidad piloto, el proceso de prueba hace énfasis en el uso y el funcionamiento del producto en las condiciones del ambiente real. La fase culmina con el hito de aceptación del producto, listo para ser desplegado, lo cual se revisa también en un consejo técnico.
- Se propone utilizar una fase de despliegue como la de implantación propuesta en MSF que persigue el objetivo de implantar la tecnología y los componentes utilizados por la solución, se estabiliza la implantación, se apoya el funcionamiento y la transición del proyecto, y se obtiene la aceptación final del cliente. Se modifican también en esta fase las actividades y los artefactos de entrada y salida, terminando con la aceptación final de los resultados del despliegue.

A continuación se muestran las fases y se describe cada una de ellas:

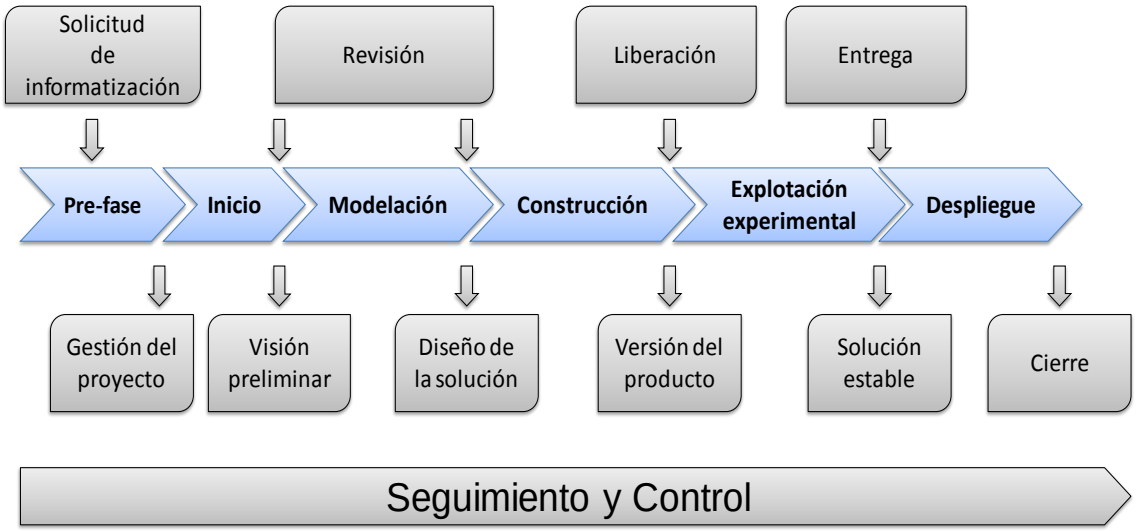


Figura 17. Fases del proceso de desarrollo.

Pre-fase

El objetivo de esta fase es organizar la gestión del proyecto a nivel de entidad, realizar un análisis macro de las necesidades del mismo contra la cartera de proyectos de la UCID fomentando la reutilización. Permitirá además organizar la gestión del equipo de desarrollo en los momentos preliminares y otorgarle una prioridad a la solicitud dentro del plan de desarrollo de la línea productiva. En esta fase se presenta a un consejo técnico la propuesta del proyecto. El consejo realiza el análisis de la solicitud y la aprobación quedando evidenciado en el acta de constitución del proyecto.
Hitos principales:

- Principales necesidades del usuario (Solicitud inicial)
- Cronograma general
- Macroprocesos de la empresa, mapas de procesos
- Propuesta de solución
- Propuesta de tecnologías y arquitectura
- Propuesta de equipo de desarrollo y elementos de capacitación

- Acta de constitución del proyecto.

Inicio

Esta fase tiene como propósito definir y acordar el alcance del proyecto con los usuarios, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software o evaluar la que ya existe y producir el plan de las fases y el de iteraciones. Se realiza el proceso de planificación general y un cronograma detallado para la siguiente fase, así como el plan para la siguiente iteración.
Hitos principales:

- Cronograma de ejecución
- Lista de riesgos
- Propuesta de nuevos elementos arquitectónicos
- Cronograma de ejecución para la próxima fase

Modelación

En esta fase se definen los procesos del dominio del problema, se identifican los principales riesgos que presenta el proyecto y se realiza un plan de mitigación, se identifican las necesidades del usuario de las que se derivan los requerimientos del producto a desarrollar. Se define la arquitectura del software, se diseña la solución preliminar del problema determinando los subsistemas y módulos en los que se dividirá el producto y la forma de construcción, y comienza la ejecución del plan de manejo de riesgos, según las prioridades definidas en él. Se realiza la planificación detallada de la siguiente fase.
Hitos principales:

- Necesidades del usuario representadas por procesos de negocio y requisitos del sistema.
- Especificación de la arquitectura de sistema
- Cronograma de ejecución para la próxima fase

Construcción

En esta fase tiene lugar la implementación del sistema (codificación) y se finalizan aquellos aspectos del diseño que hubieran quedado por finalizar en la fase anterior. Como resultado, obtenemos el producto y los manuales de usuario, así como la documentación del modelo de la aplicación. El criterio de evaluación de la fase es determinar si el producto es lo bastante maduro como para ser usado por los usuarios finales. El producto al finalizar esta fase se considera que se encuentra en una versión "beta" (ya que pueden surgir incidencias que se resolverán en la siguiente fase).
Hitos principales:

- Versión beta del producto
- Manuales de usuario y documentación de la aplicación
- Normativas para la fase de explotación experimental.
- Cronograma de ejecución para la próxima fase

Explotación experimental

Tiene como propósito asegurar que el software está listo para realizar las pruebas de aceptación por los usuarios. Incluye las pruebas del producto como parte de su preparación para ser entregado, y la realización de ajustes en respuesta a la retroalimentación recibida de los usuarios. En este punto del ciclo de vida la

retroalimentación de los usuarios debe enfocarse fundamentalmente en ajustes específicos y de corto alcance al producto junto a otros temas como configuración, instalación, y usabilidad.
Hitos principales:

- Versión beta del producto implantada
- Producto estable
- Acta de aceptación del producto
- Cronograma de ejecución para la próxima fase

Despliegue

Se realiza la generalización del producto en las entidades según lo aprobado en el Cronograma de implantación. Se estabiliza la implantación, apoyando el funcionamiento y la transición del proyecto, y se obtiene la aprobación final del cliente.
Hitos principales:

- Versión veta del producto desplegada
- Resultados del despliegue (Ver anexo II)

Al final de cada fase se reúne el grupo de gestión de la línea y los líderes de los diferentes equipos que la componen para determinar si los objetivos de la fase se cumplieron y así presentar al Consejo Técnico Formal para su evaluación y dar paso o no a la fase siguiente.

3. *Definición de roles*

Los roles son una definición abstracta que especifican el comportamiento y las responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas. Sus responsabilidades abarcan tanto el llevar a cabo un conjunto de actividades como responder por la elaboración de un conjunto de artefactos. Además describen cómo los individuos deberán comportarse en el contexto de un proyecto. A continuación se muestra la propuesta de roles a utilizar.

❖ **Dirección**

Agrupa los roles que están involucrados fundamentalmente en la gestión y control del desarrollo del producto.

- **Líder de la línea de desarrollo:** Es la conciencia del grupo, interviene y enseña. Debe proporcionar un proceso de desarrollo limpio y eficiente, entrenando a los miembros de la línea en las cuestiones relacionadas a él, y asegurar que no se obstaculice el trabajo de sus miembros. Interactúa con el cliente y el equipo. Es responsable de asegurarse que el proyecto se lleve a cabo de acuerdo al proceso de desarrollo y que progrese según lo previsto, tiene la última palabra en materia de visión, cronograma y asignación de personal. Coordina los encuentros diarios y se encarga de eliminar eventuales obstáculos. Sigue los rastros del desarrollo, colectando métricas y analizándolas para avisar cuando hay una estimación alarmante. Es importante que se vea como un facilitador, antes que como quien da las órdenes. También debe ajustar el proceso a cada uno de los proyectos que tiene la línea, así como asegurar la retroalimentación del proceso con la experiencia obtenida en los proyectos y asistir al Líder de gestión en la planificación. Debe ser miembro del equipo y trabajar a la par.

- **Líder de gestión:** Gestiona los recursos, humanos y de todo tipo, coordina las prioridades de desarrollo con el jefe de línea y planifica las iteraciones, asigna y controla el trabajo, define de conjunto la organización de la línea. Garantiza los procesos de gestión del equipo, la planificación y la gestión de riesgos a toda la línea. Es atendido metodológicamente por la Oficina de Gestión de Proyectos y proporciona a la misma toda la información de los proyectos bajo su responsabilidad.

❖ **Analistas**

Agrupa los roles que están involucrados fundamentalmente en la identificación y descripción de los procesos de negocio y extracción y especificación de los requisitos del software.

- **Funcionales:** Responsables de detallar la organización o parte de ella. Identificar los procesos de negocio, describirlos y proponerles mejoras. Además de revisar y aprobar los requisitos cuando se determinan.
- **Analistas:** Responsables de representar lo que definan los *funcionales*. Además de dirigir y coordinar el proceso de extracción de requisitos, especificando los detalles de cada una de las funcionalidades del sistema. Asume las responsabilidades del funcional cuando este no esta presente.

❖ **Desarrolladores**

Agrupa los roles involucrados fundamentalmente en el diseño de la arquitectura, el diseño detallado y la implementación del software.

- **Arquitecto de sistema:** Responsable de la descripción arquitectónica del software, la identificación de los subsistemas, módulos y componentes y la integración y comunicación entre los elementos de la arquitectura. Realiza funciones de coordinador entre las factorías, mentor y programador más experto. Máximo responsable de la reutilización dentro del desarrollo.
- **Arquitecto de datos:** Responsable del diseño del almacén de datos persistentes utilizado por el sistema. Realiza las pruebas de concepto de la base de datos, que garanticen la integridad de los datos, el rendimiento y otros principios.
- **Líder de factoría:** Es un programador jefe, que participa en los análisis de los requerimientos y diseño de subsistemas y componentes principales junto con los arquitectos. Organiza y dirige el trabajo de diseño e implementación e integración de las clases, componentes, etc. que sean asignados a su factoría de acuerdo a las orientaciones de la dirección de la Línea de Producción.
- **Diseñador:** Responsable del diseño del sistema al interior de las clases y componentes, dentro de los límites de los requisitos y la arquitectura.
- **Desarrollador de interfaz de usuario:** Coordina el diseño de la interfaz de usuario, utiliza los requisitos de usabilidad y crea prototipos candidatos de interfaz de usuario de acuerdo a los requisitos. Encargado de realizar el trabajo artístico que requiera el proyecto (iconos, pantalla de splash, gráficos, CSS, etc.).
- **Desarrollador de lógica de negocio:** Responsable de la codificación, prueba y documentación de las clases y componentes, trabajan bajo la guía de los diseñadores y jefes de factoría.

❖ **Pruebas**

Agrupa todos los roles que tratan habilidades específicas para el aseguramiento de la calidad en el proceso de desarrollo del software.

- **Especialista de calidad:** Responsable de identificar y definir las pruebas y revisiones necesarias, monitorizar los resultados de las mismas y el progreso del aseguramiento de la calidad en la línea productiva y evaluar el desarrollo en general percibido como resultado de las actividades de prueba.
- **Probador:** Encargado de las actividades de pruebas fundamentales, dirige las pruebas y registra los resultados de esas pruebas.

❖ **Soporte**

Rol encargado de producir los materiales de soporte a los usuarios como por ejemplo: manuales de usuarios, textos de la ayuda, etc. Desarrollar el material de entrenamiento que permita enseñar a los usuarios a utilizar el sistema y capacitarlos con los mismos. También tiene dentro de sus responsabilidades instalar el sistema y mantenerlo, principalmente en las fases de explotación experimental y despliegue. Asistiendo a los usuarios finales, respondiendo sus preguntas; analizando los problemas que ellos encuentran en el entorno; capturando nuevas funcionalidades y aplicando cambios o realizando arreglos.

❖ **Gestor de configuración**

Rol encargado de administrar la configuración del proceso de desarrollo garantizando el acceso de todos los miembros a los artefactos del proyecto.

Comité de roles

Esta estructura se crea como sugerencia de la dirección del centro con el objetivo de reforzar el conocimiento práctico de todas las personas que desarrollan su actividad profesional en el mismo o están interesados en la investigación en esta área especifica. Por cada rol definido existirá un comité que básicamente es dirigido por un coordinador general y un secretario. La OGP coordinará el trabajo de los comité de roles y se apoyará en estos para gestionar la capacitación y certificación de los profesionales y estudiantes.

4. ***Disciplinas***

Se proponen en total 15 disciplinas, las que se pueden agrupar fundamentalmente en tres grupos: de empresa, de desarrollo y de soporte. En el primer grupo se encuentran 5: modelando negocio de empresa, gestión de cartera de proyectos, arquitectura de empresa, reutilización estratégica y gestión de persona las 4 últimas se proponen en la metodología EUP pero se le modificaron las actividades a realizar y los artefactos que reciben como entrada y que generan como salida.

En el grupo de desarrollo se encuentran otras 5: modelación del negocio, requerimientos, diseño, implementación y pruebas. Estas se proponen en prácticamente todas las metodologías que existen diferenciándose en las actividades y artefactos que proponen y en la profundidad con que se realizan las mismas. Para estas disciplinas se proponen: los artefactos de la técnica BPM, las ER para describir el sistema, diseño de alto nivel a través de diagramas de secuencia que propone el PU, artefactos orientados a describir arquitecturas basadas en componentes y casos de pruebas como propone el PU pero con diferentes plantillas.

En el grupo de soporte se encuentran las últimas 5: despliegue, configuración, gestión de cambio, ambiente y soporte, estas también se proponen en el PU y en EUP, pero se le modifican las actividades y artefactos en correspondencia con las particularidades del centro. A continuación se describen las disciplinas.

4.1 Modelando Negocio de empresa

Descripción

Facilita el entendimiento del negocio de la entidad que solicita la informatización, ayuda a identificar las áreas que puedan ser mejoradas con el proyecto o identificar nuevas áreas, también provee de información al equipo para delimitar el alcance del proyecto (Ver anexo III). Lográndose a través de las siguientes actividades:

1. Recibir solicitud de informatización
2. Determinar organigrama de la entidad
3. Identificar los macroprocesos de la entidad
4. Identificar los procesos de la entidad
5. Describir los procesos de la entidad

Actividades

1. Recibir solicitud de informatización

La entidad recibe la solicitud de informatización y realiza un cronograma general que permitirá realizar las primeras actividades de identificación de conjunto con el usuario.

2. Determinar el organigrama de la entidad

La organización se divide en áreas físicas y lógicas y cada una de estas tiene responsabilidades y objetivos diferentes. La estructura de la organización puede ser representada a través de los organigramas. Estos son una guía o plano de la organización en el que se representan las unidades que la componen y las relaciones de autoridad y comunicación entre las mismas, esto le permitirá al equipo de desarrollo ubicarse dentro del alcance de la organización.

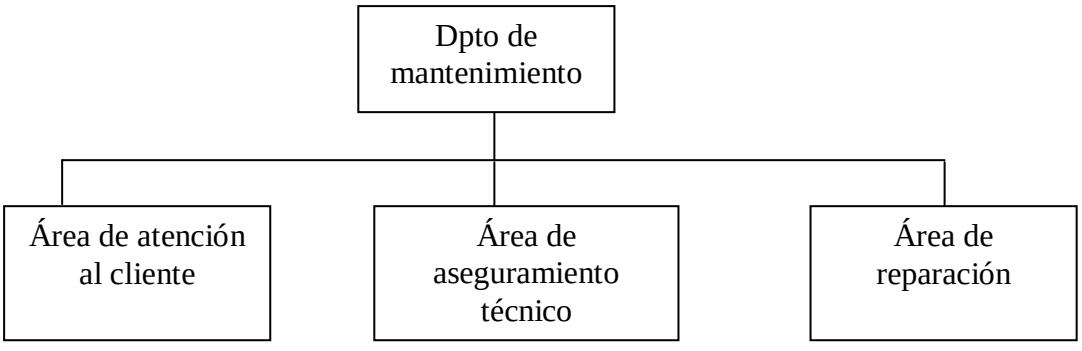


Figura 18. Organigrama general de empresa.

Además señalar en cada área los diferentes puestos de trabajo lo que permitirá identificar los roles dentro de cada departamento de la organización y la cantidad de trabajadores por rol para de esta forma tener identificados los posibles involucrados en el negocio.

3. Identificar los macroprocesos

La descripción gráfica de la organización es el macroproceso o red de procesos. Un macroproceso es: el propósito, función o servicio de una entidad o dependencia, generalmente establecido por la norma de creación de la misma [Henares, 2006]. En general los macroprocesos recogen un conjunto de procesos que permiten alcanzar el resultado propuesto por la institución.

Figura 19. Diagrama de interrelación de macroprocesos.

Tabla 3. Ficha de macroproceso.

Ficha de macroproceso			
Código	Macroproceso	Objetivo	Procesos
[Número que lo identifica]	[Nombre del macroproceso]	[Objetivo general donde se explique el resultado esperado]	[Nombre de los procesos en los que se divide]
[Breve descripción del macroproceso, áreas que involucra]			

4. Identificar los procesos

La ISO 9001:2008 define al proceso como: "Conjunto de actividades mutuamente relacionadas o que interactúan, las cuales transforman elementos de entrada en resultados" [ISO 9001, 2000], pueden ser identificados a partir de encuestas y entrevistas a los usuarios.

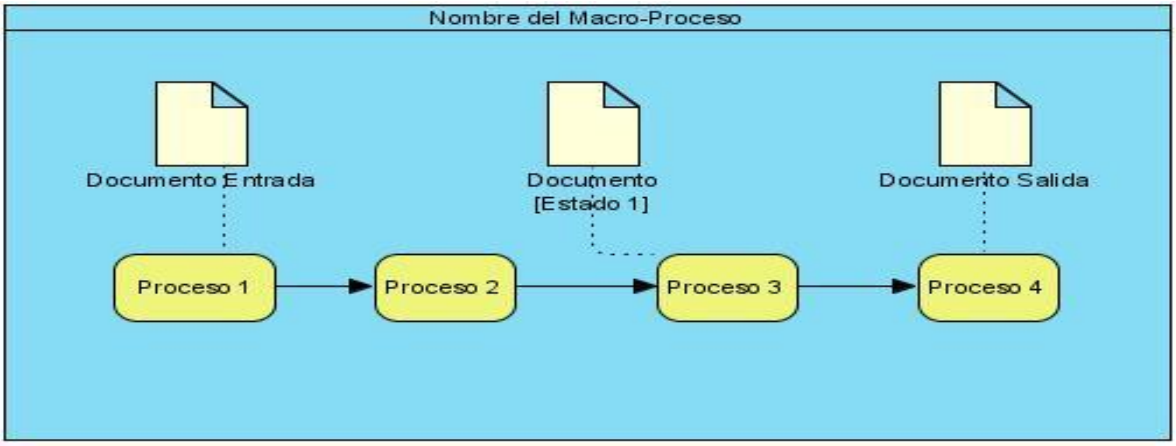


Figura 20. Mapa de proceso.

5. Describir los procesos de la entidad

Esta actividad tiene como objetivo realizar una descripción de alto nivel de los procesos para poder conocerlos y ubicar de manera fácil dentro de la cartera de proyectos la posible solución.

Tabla 4. Ficha de proceso.

Ficha de proceso					
Nombre del proceso: [Nombre del proceso]			**Responsable:** [Nombre del responsable del proceso]		
Misión: [Razón de ser del proceso]					
Precondición: [requisito indispensable para realizar e l proceso, estado en que debe estar el negocio para dar inicio a este proceso]					
Síntesis del proceso					
Flujo normal del proceso					
Disparador/ Proveedor/ Cliente	**Entrada**	**Actividad**			**Salida**
		No	**Nombre**	**Nivel de informatización**	
[Nombre del rol que lleva a cabo la actividad y descripción]	[Nombre de cada información de entrada (documento)]			[Automática, del usuario o manual]	[Nombre de cada información de salida (documento)]
Flujos alternos del proceso					
Actividad a partir de la cual se genera el flujo y la alternativa a seguir:					
Disparador/ Proveedor/ Cliente	**Entrada**	**Actividad**			**Salida**
		No	**Nombre**	**Nivel de informatización**	
Flujos paralelos del proceso					
Disparador/ Proveedor/ Cliente	**Entrada**	**Actividad**			**Salida**
		No	**Nombre**	**Nivel de informatización**	
Descripción de las entradas/Salidas:					
Nombre	**Tipo**	**Estado**		**Tipo**	**Responsable**
[Nombre de la entrada o salida]	[Entrada/ Salida]	[Estado por el que puede transitar una entrada. Ej.: creado, enviado]		[Digital, documento físico]	[Rol que la recibe (entrada), Rol que la produce (salida)]
Post-condición: [estado en que debe quedar el negocio terminado este proceso]					
Reglas el proceso: [restricciones que afectan el proceso de negocio]					

Reglas asociadas al proceso	Clasificación			
	Información	**Flujo del proceso**	**Clientes o proveedores**	**Relación**
[Nombre de todas las restricciones asociadas al negocio]	[marcar con X si está vinculada a los documentos de entrada/Salida]	[marcar con X si está encausada dentro del flujo del proceso]	[marcar con X si está vinculada al cliente]	[marcar con X si la regla pertenece a alguna relación]

Para cada proceso que contiene el macroproceso se identifican cuales se pueden informatizar completamente y cuales solo algunas actividades. Que una actividad se pueda informatizar o no depende del nivel de informatización de la mismas. Se han identificado tres tipos de actividades en dependencia de su nivel informatización las cuales aparecen explicadas en la tabla 5.

Tabla 5. Clasificación de actividades según su nivel de informatización.

Clasificación		
Automáticas	**Del usuario**	**Manuales**
Actividades que puedan ser gestionadas por un sistema informático sin la ayuda humana. Ejemplo: Verificar existencia de producto	Actividades que para su funcionamiento óptimo las debería realizar el usuario con la ayuda de hardware especializado. Ejemplo: Para la actividad: captar huellas digitales, sería necesario un lector de huellas, otro ejemplo es el cajero automático.	Actividades que sólo pueden ser llevadas a cabo por el hombre. Ejemplo: entregar mercancía al cliente

En caso que las actividades sean **del usuario** se acuerda con el cliente del sistema cómo se van a llevar a cabo, si se mantienen como se realizan actualmente o se va a utilizar hardware especializado (en caso de que exista ese hardware). Si alguno de los procesos que van a ser informatizados tiene alguna entrada o salida proveniente de un proceso perteneciente a un macroproceso que no ha sido contemplado dentro del sistema a construir se verifica con el cliente si esa funcionalidad estará dentro de los límites del sistema.

Resumen

Tabla 6. Entradas, actividades y salidas de la disciplina modelando negocio de empresa.

Modelando negocio de empresa		
Entradas	**Actividades**	**Salidas**
▪ Solicitud inicial	▪ Determinar el organigrama de la entidad ▪ Identificar los macroprocesos ▪ Identificar los procesos ▪ Describir los procesos	▪ Cronograma general ▪ Organigrama de la entidad ▪ Ficha de macroproceso ▪ Mapa de procesos

		▪ Ficha de proceso
Roles que intervienen ▪ OGP (área de seguimiento y control de los proyectos)		

4.2 Gestión de cartera de proyectos

Descripción

"Una cartera de proyectos es el conjunto de proyectos que una organización genera, ejecuta y administra simultáneamente en un momento dado" [Premio nacional de tecnología, 2006]. La gestión de la cartera de proyectos "es un proceso dinámico de decisiones, por el cual una lista de proyectos activos y nuevos se revisa constantemente. Los nuevos proyectos son evaluados, seleccionados y priorizados a partir de técnicas, métodos y criterios establecidos por la organización. Los proyectos existentes se pueden acelerar, eliminar o despriorizar. Los recursos y las personas pueden ser asignados y reasignados por los proyectos. El proceso se caracteriza por información incierta y cambiante, oportunidades dinámicas, objetivos y consideraciones estratégicas múltiples, interdependencia entre proyectos, y la participación de múltiples tomadores de decisiones y locaciones" [Cooper, 2002].
Esto se realiza para asegurar que todos los proyectos están visibles, planeados y alineados con los objetivos de la empresa. Es importante destacar que la administración de la cartera debe mantenerse a través de todo el ciclo de vida del proyecto hasta la fase de retiro. La principal función de esta disciplina es la de obtener las tecnologías y los recursos necesarios para la ejecución de los proyectos, pero se complementa con otras como: alinear la tecnología en todas las áreas de la organización, proteger el patrimonio tecnológico de la organización (Ver anexo IV). Tiene las siguientes actividades principales:

1. Alinear la cartera de proyectos
2. Integración de los proyectos
3. Selección de proyectos
4. Balancear la cartera de proyectos
5. Asignación de recursos
6. Monitoreo

Actividades

1. Alinear la cartera de proyectos

La cartera de proyectos es la expresión concreta de la estrategia de la entidad, por lo tanto esta actividad demuestra cómo se utilizan los recursos y cuáles son las prioridades de la misma, aunque no se encuentre de manera explícita. El objetivo principal de esta actividad es que los proyectos que desarrolle se correspondan con sus estrategias de negocio. Para esto es necesario definir:

- ✓ Estrategia: Se describe cómo actuará la entidad a mediano plazo, que tipo de actividades realizará (mejora o adaptación de los productos existentes, desarrollo totalmente nuevo, infraestructura, preparación, etc.)
- ✓ Los escenarios estratégicos: Negocios en los que se planea atacar, productos que se piensan modificar o desarrollar, tecnologías que se requieren adquirir o desarrollar.

2. Integración de los proyectos

Se reúnen todas las solicitudes de proyectos y toda la información que se tenga de los mismos se presentan en el consejo técnico que se encargará de evaluar su integración a la cartera.

3. Selección de proyectos

Se seleccionan los proyectos que más beneficios a la entidad y a los clientes produzcan, basándose en decisiones estratégicas de negocio y en prioridades definidas por la dirección de la entidad. Se pueden responder las siguientes preguntas para facilitar la selección:

- ✓ ¿Cuáles son los factores que hacen que los proyectos de la entidad tengan éxito?
- ✓ ¿Cuáles son las características de los proyectos que han tenido éxito en la entidad?
- ✓ ¿Cuáles son las características de un buen proyecto?
- ✓ ¿Qué se debe evitar en la realización de un proyecto?
- ✓ ¿En qué tipo de proyectos la entidad tiende a tener éxito?

Además en la selección influye la identificación de la posible solución en la cartera de proyectos de la entidad, proyectos finalizados o en desarrollo, que se podrían utilizar para dar solución a la nueva solicitud.

4. Balancear la cartera de proyectos

La composición de la cartera se ve determinada por la naturaleza del negocio, la orientación estratégica de la empresa y el tipo de proyectos a ejecutar. El objetivo de esta actividad es que los proyectos que se ejecuten estén debidamente balanceados, de acuerdo a los parámetros definidos en la entidad. Se pueden utilizar los siguientes parámetros para balancear:

- ✓ Proyectos de largo, mediano y corto plazo.
- ✓ Proyectos de alto, medio y bajo riesgo.
- ✓ Proyectos de mejora de productos actuales y de desarrollo de nuevos productos.
- ✓ Proyectos de investigación básica, mejora de procesos o tecnología.
- ✓ Proyectos ejecutados interna y externamente.

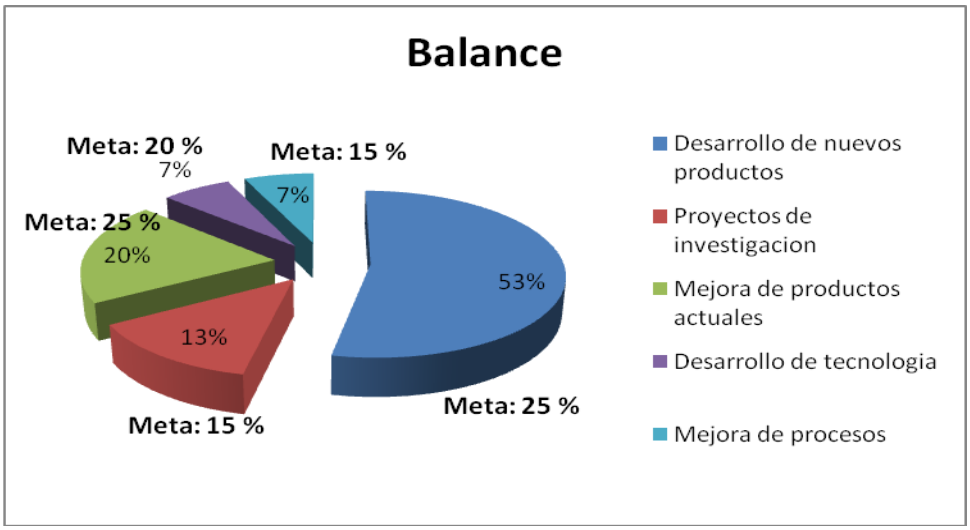


Figura 19. Gráfico de balance de los proyectos.

5. Monitoreo

Se realiza un seguimiento de cada uno de los proyectos de la cartera, midiendo y comparando los avances y resultados obtenidos contra los objetivos y metas planteadas. Se revisa la situación de los proyectos los avances logrados, recursos utilizados, perspectivas, dificultades y acciones necesarias. Se realizan procesos de auditorías a la cartera para realizar ajustes de tiempo y recursos o decidir si se mantiene la composición de la cartera o se adapta a las nuevas condiciones de la entidad. Se informa a todos los involucrados la situación de la cartera.

Resumen

Tabla 7. Entradas, actividades y salidas de la disciplina gestión de cartera de proyectos.

Gestión de cartera de proyectos		
Entradas	**Actividades**	**Salidas**
▪ Objetivos estratégicos de la entidad	▪ Alinear cartera de proyectos ▪ Integración de proyectos ▪ Selección de proyectos ▪ Balancear cartera ▪ Monitoreo	▪ Estrategia entidad ▪ Escenarios estratégicos ▪ Conjunto de proyectos ▪ Mapa de procesos ▪ Ficha de procesos ▪ Cartera de proyectos ▪ Cartera de proyectos balanceada ▪ Resultados monitoreo
Roles que intervienen ▪ OGP (área de seguimiento y control de los proyectos)		

4.3 Arquitectura de empresa

Descripción

Esta disciplina propone la creación de una vista de la arquitectura de la entidad, incluye frameworks, configuraciones de despliegue e infraestructura de soporte que forman la arquitectura técnica de la entidad. Es necesaria esta vista para la identificación de solicitudes que incluyen nuevos desarrollos en esta área (Ver anexo V). Realiza la siguiente actividad:

1. Evaluar la arquitectura

Actividades

1. Evaluar arquitectura

El propósito de esta actividad es realizar una evaluación de la arquitectura de la entidad para determinar su utilización en la solución y detectar algún requisito en la solicitud que implique nuevos desarrollos tecnológicos.

Resumen

Tabla 8. Entradas, actividades y salidas de la disciplina arquitectura de empresa.

Arquitectura de empresa		
Entradas	**Actividades**	**Salidas**
▪ Solicitud inicial ▪ Ficha de macroproceso ▪ Mapa de procesos ▪ Ficha de procesos ▪ Propuesta de solución ▪ Vista arquitectónica y tecnológica.	▪ Evaluar arquitectura	▪ Propuesta de nuevos elementos arquitectónicos.
Roles que intervienen ▪ OGP (área de seguimiento y control de los proyectos)		

4.4 Reutilización estratégica

Descripción

Esta disciplina permite realizar un estudio en el repositorio de la entidad en busca de rehusar código, componentes, patrones, artefactos y plantillas (Ver anexo VI). Realiza la siguiente actividad:

1. Identificar componentes reutilizables

Actividades

1. Identificar componentes reutilizables

El propósito de esta actividad es realizar una identificación de los componentes que puedan ser reutilizados dentro del repositorio de componentes.

Resumen

Tabla 9. Entradas, actividades y salidas de la disciplina reutilización estratégica.

Reutilización estratégica		
Entradas	**Actividades**	**Salidas**
▪ Solicitud inicial ▪ Mapa de procesos ▪ Ficha de	▪ Identificar componentes reutilizables	▪ Propuesta de solución

proceso ▪ Propuesta de solución ▪ Repositorio de componentes		
Roles que intervienen ▪ OGP (área de seguimiento y control de los proyectos)		

4.5 Gestión de personas

Descripción

Esta disciplina propone administrar los equipos de desarrollo a nivel macro, ayudarlos a crecer, intermediar en las relaciones de unos con otros si ejecutan proyectos de conjunto. Describe además el proceso de organización, monitoreo, educación y motivación del personal para asegurarse que el equipo trabaja unido y contribuye exitosamente al proyecto con su organización (Ver anexo VII). Tiene las siguientes actividades:

1. Asignar equipo
2. Asignar prioridad
3. Gestionar capacitación

Actividades

1. Asignar equipo

El propósito de esta actividad es asignar el equipo de desarrollo que llevará a cabo la ejecución del proyecto. Aquí tendrá influencia la selección que se haya hecho de la cartera de proyectos, sino se encontró nada que pudiera ser utilizado se le asignará al equipo que mejores condiciones tenga para desarrollar la nueva solicitud.

2. Asignar prioridad

El propósito de esta actividad es asignar una prioridad a la nueva solicitud, revisar los proyectos que tiene asignado el equipo de desarrollo elegido y los cronogramas de los mismos e incluir la ejecución de la nueva solicitud dentro de ellos.

3. Gestionar capacitación

El propósito de esta actividad es gestionar la capacitación necesaria para que el equipo de desarrollo pueda ser capaz de ejecutar el nuevo proyecto.

Resumen

Tabla 10. Entradas, actividades y salidas de la disciplina gestión de personas.

Gestión de personas		
Entradas	**Actividades**	**Salidas**
▪ Propuesta de solución ▪ Capital humano ▪ Propuesta de	▪ Asignar equipo ▪ Asignar prioridad ▪ Gestionar capacitación	▪ Propuesta de equipo de desarrollo ▪ Prioridad de solicitud

equipo de desarrollo ▪ Proyectos asignados al equipo de desarrollo		▪ Plan de capacitación
Roles que intervienen ▪ OGP (área de seguimiento y control de los proyectos)		

Estas disciplinas de gestión de entidad aunque tienen su mayor desarrollo en la Profase deben mantenerse durante todo el ciclo del producto hasta ir desapareciendo en dependencia de la necesidad en el resto de las fases.

4.6 Modelación del negocio

Descripción

El objetivo del modelo de negocio es entender el funcionamiento de la organización del cliente, tanto en estructura como en la dinámica de sus procesos, por parte del equipo de desarrollo. De esta forma se van determinando necesidades operacionales, así como restricciones que presenta la entidad, obteniéndose finalmente un entendimiento del negocio (Ver anexo VIII). Tiene las siguientes actividades:

1. Descripción de los procesos de negocio
2. Estudio de proceso de negocio existente
3. Mejora de procesos
4. Validación de los procesos de negocio

Actividades

1. Descripción de los procesos de negocio

A través de entrevistas a los usuarios se realizará una descripción detallada de los procesos del negocio, que incluye una representación gráfica de las actividades que ocurren en cada uno de ellos. Estos diagramas facilitan la interpretación y están compuestos por:

- Involucrados en la realización del mismo.
- Beneficiarios del proceso (cliente)
- Conjunto de actividades que se llevan a cabo, permitiendo visualizar el flujo y la secuencia de las mismas. Las actividades internas de cualquier proceso las realizan personas, grupos o departamentos de la organización (consultar organigrama).
- Información de entrada y salida de cada actividad y su estado.
- Restricciones a tener en cuenta.

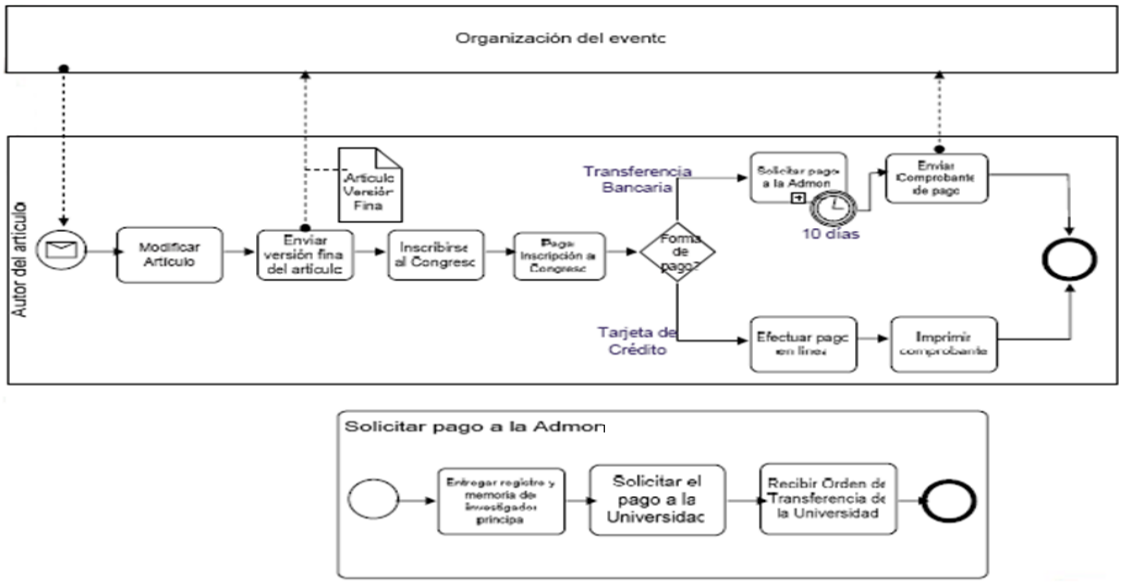


Figura 21. Diagrama de proceso.

En la representación gráfica del proceso se representa cada tipo de actividad con un estereotipo diferente, como se muestra a continuación en la figura 22:

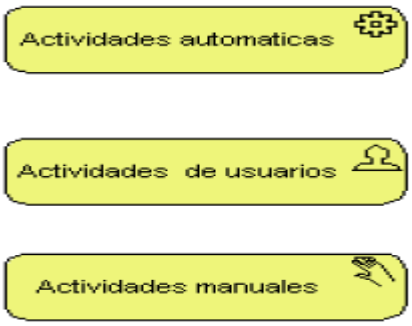


Figura 22. Estereotipo de las actividades en dependencia de su nivel de informatización.

2. Estudio de proceso de negocio existente

El propósito de esta actividad es realizar un estudio detallado de procesos de negocios existentes con el objetivo de lograr la reutilización de aquellos que tengan semejanza en el dominio del problema, así como identificar mejoras que se puedan realizar en el proceso y lograr efectividad en el mismo.

3. Mejora de procesos

Es necesario determinar si los procesos pueden ser optimizados desde el punto de vista estructural, que es el tipo de mejora que puede ayudar a la construcción del sistema informático. Dicha mejora se basa en aportaciones creativas, imaginación y sentido crítico. Dentro de esta categoría entran:

1. La redefinición de destinatarios.
2. La redefinición de expectativas.
3. La redefinición de los resultados generados por el proceso.
4. La redefinición de los involucrados.
5. La redefinición de la secuencia de actividades.

Las categorías de la 1 a la 4 deben realizarse de conjunto con la dirección de la entidad y los responsables de los procesos. Pero la que si tiene gran importancia para la obtención de requisitos del sistema a construir es la 5 con el objetivo de:

- Eliminar trabajo que no sea estrictamente necesario para las necesidades del negocio.
- Reducir etapas intermedias y transiciones de fase.
- Eliminar duplicidades.

En este caso se propone usar la misma ficha y diagrama de procesos pero esta tendrá una sección adicional donde se explique en qué consiste la mejora y en el diagrama del proceso se representará el proceso con las mejoras realizadas.

6. Validación de los procesos de negocio

El propósito de esta actividad es validar que los procesos del negocio han sido descritos sin inconsistencias y que los errores encontrados durante la definición de los mismos hayan sido corregidos, además de comprobar que la documentación que se genera durante esta actividad proporcione un alto grado de aseguramiento de los procesos.

Resumen

Tabla 11. Entradas, actividades y salidas de la disciplina modelación de negocio.

Modelación del negocio		
Entradas	**Actividades**	**Salidas**
▪ Ficha de macroproceso ▪ Mapa de procesos ▪ Ficha de proceso	▪ Descripción de procesos de negocio ▪ Estudio de procesos de negocio existentes ▪ Mejora de procesos ▪ Validación de los procesos de negocio	▪ Diagrama de proceso ▪ Glosario de términos ▪ Modelo conceptual
Roles que intervienen ▪ Analistas (línea de desarrollo y funcionales)		

4.7 Requerimientos

Descripción

El objetivo de esta disciplina es especificar las condiciones o capacidades que el sistema debe cumplir y las restricciones bajo las cuales debe operar para automatizar el negocio, logrando un entendimiento entre el equipo de desarrollo y el cliente (Ver anexo IX). Tiene las siguientes actividades:

1. Identificación y clasificación de requisitos
2. Especificación de requisitos
3. Validación de requisitos
4. Gestión de requisitos

Actividades

✓ Identificación y clasificación de requisitos

En esta actividad se derivan los requerimientos del sistema a través de la observación de sistemas existentes y entrevistas con usuarios, de forma tal que se defina cómo el

sistema o producto se ajusta mejor a las necesidades del negocio y cómo va a ser utilizado este por los usuarios. Se deben definir además los requerimientos no funcionales, determinando cómo se va a comportar el sistema y qué cualidades debe tener. Esta actividad debe ser cooperativa e iterativa, logrando que exista de ambas partes comprensión y formalidad.

✓ Especificación de requisitos

Una vez definidos claramente los requisitos funcionales y no funcionales que va a tener el sistema, se debe realizar una descripción detallada de cada uno de ellos, de forma que sea entendible por los clientes y usuarios. La descripción de los requerimientos no funcionales identificados se detalla en el documento Especificaciones Complementarias. Estos pueden ser clasificados en: Requisitos de Usabilidad, Confiabilidad, Rendimiento, Soporte, Interfaz, Portabilidad, Seguridad, así como en requerimientos mínimos de Software y Hardware bajo los cuales el sistema debe operar.
El documento Especificación de Requisitos del Software debe contener los conceptos tratados, precondiciones y post-condiciones, una descripción detallada del flujo del requisito, las validaciones, y una propuesta de los prototipos de interfaz de usuario que debe realizar el equipo de desarrollo.

✓ Validación de requisitos

Para asegurar que los requisitos han sido establecidos sin ambigüedades o inconsistencias y que los errores encontrados durante la definición de los mismos hayan sido corregidos, se realizará una revisión a las especificaciones realizadas para ser aprobadas. Las que terminarán siendo firmadas por los usuarios.

✓ Gestión de requisitos

Esta actividad tiene como objetivo mantener los requisitos bien identificados, y poder seguir su desarrollo durante todo el ciclo de vida del producto. La gestión de requisitos incluye además, actividades de documentación, revisión, y control de cambios. El uso de este artefacto posibilitará tener en cuenta para cada requisito atributos como:

- Estado (Posibles estados del requisito: Identificados, Descritos, en Validación, Aprobados).
- Esfuerzo (Tiempo de desarrollo).
- Beneficio (Beneficio que aporta al negocio).
- Estabilidad (Identificar cuáles requisitos han sido incorporados, eliminados, modificados, o no han sufrido cambios en el periodo establecido).

Resumen

Tabla 12. Entradas, actividades y salidas de la disciplina requerimientos.

Requerimientos		
Entradas	**Actividades**	**Salidas**
▪ Modelo conceptual ▪ Diagrama de proceso ▪ Ficha de	▪ Identificación y clasificación de requisitos ▪ Especificación de requisitos ▪ Validación de requisitos	▪ Lista de requisitos identificados ▪ Especificaciones de requisitos ▪ Especificaciones

proceso ▪ Mapa de proceso	▪ Gestión de requisitos	complementarias ▪ Especificaciones de requisitos[firmadas] ▪ Requisitos priorizados ▪ Matriz de seguimiento
Roles que intervienen ▪ Analistas (línea de desarrollo y funcionales)		

4.8 Diseño

Descripción

Se enfoca en analizar los requerimientos y diseñar el sistema transformando el conocimiento en modelos, inicialmente da una visión del software y tras posteriores refinamientos sirve como esquema para la implementación del mismo (Ver anexo X). Tiene las siguientes actividades:

1. Identificar componentes
2. Especificar componentes
3. Realizar diagrama de componentes
4. Describir la integración de componentes
5. Integrar los elementos de la arquitectura
6. Diseñar el modelo de despliegue
7. Diseñar clases
8. Realizar diagrama de interacción
9. Integrar los elementos del diseño
10. Diseñar interfaz de usuario
11. Diseñar la base de datos
12. Especificar los datos
13. Describir trazabilidad
14. Especificar réplica
15. Evaluar el rendimiento

Actividades

1. Identificar componentes

Esta actividad tiene el propósito de identificar componentes que es probablemente la actividad más compleja que realiza el arquitecto de sistema. Esta se obtiene del análisis de: el listado de requisitos del sistema, la solución propuesta en la pre-fase y los artefactos obtenidos en la disciplina de negocio (mapa de procesos y mapa conceptual) de ser necesario. Es compleja además porque no necesariamente existirá una relación uno a uno con los conceptos definidos y que deberán relacionarse con dichos componentes, sino que existirán componentes que generen las clases persistentes (que son las que aparecerán entonces en la base de datos), y otros componentes que lo que harán será generar lógica de negocio. Se agrupan los requisitos por funcionalidades comunes y se les asignan componentes. A partir de la dependencia entre componentes y la prioridad de los requisitos se determina la complejidad de los componentes y la criticidad de los mismos. Evaluar por cada componente su complejidad, así como qué tan importante es, desde el punto de vista de la integración, ambos criterios son

variables de peso a la hora de definir una prioridad de implementación, así como a la hora de la aceptación o no de un cambio determinado.
Tomando el valor de la criticidad del componente se establecen valores de prioridad que equivalen al orden de desarrollo que deben tener los componentes (a menor valor mayor prioridad).

Tabla 13. Análisis de complejidad y criticidad de los componentes.

<table>
<tr><th colspan="5">Componentes</th></tr>
<tr><td>Componente</td><td colspan="4"><<Nombre del componente>></td></tr>
<tr><td>Descripción</td><td colspan="4"><<Descripción del componente>></td></tr>
<tr><td colspan="5">Requisitos <<cantidad numérica>></td></tr>
<tr><td>No</td><td>Requisito</td><td colspan="2">Código</td><td>Complejidad</td></tr>
<tr><td><<número consecutivo>></td><td><<Nombre del requisito>></td><td colspan="2"><<Subsistema_>>
<<Componente_>>
<<Nombre del requisito>></td><td><<(Valor 1 - 10)>></td></tr>
<tr><td>Servicios que brinda</td><td colspan="4"><<Cantidad de servicios>></td></tr>
<tr><td>Componentes dependientes</td><td colspan="4"><<Cantidad de Componentes que dependen>></td></tr>
<tr><td>Tipo de Componente</td><td colspan="4"><<Núcleo, Configuración o Tecnológico>></td></tr>
<tr><td>Complejidad</td><td colspan="4">= cantidad de requisitos + (sumatoria de la complejidad de los requisitos del componente * 4) + (cantidad de servicios del componente * 2)</td></tr>
<tr><td>Criticidad de Integración</td><td colspan="4">Cantidad de los componentes que dependen, más la sumatoria de la criticidad de los componentes dependientes</td></tr>
<tr><td>Prioridad</td><td colspan="4">El valor de priorización equivale al orden de desarrollo que deben tener los componentes (a menor valor mayor prioridad). Para establecer esta prioridad se toma como partida la criticidad calculada para el componente.</td></tr>
</table>

2. Realizar diagrama de componentes

Esta actividad tiene como propósito obtener una vista general de los componentes, sus dependencias e integraciones, así como su importancia desde el punto de vista de la integración.

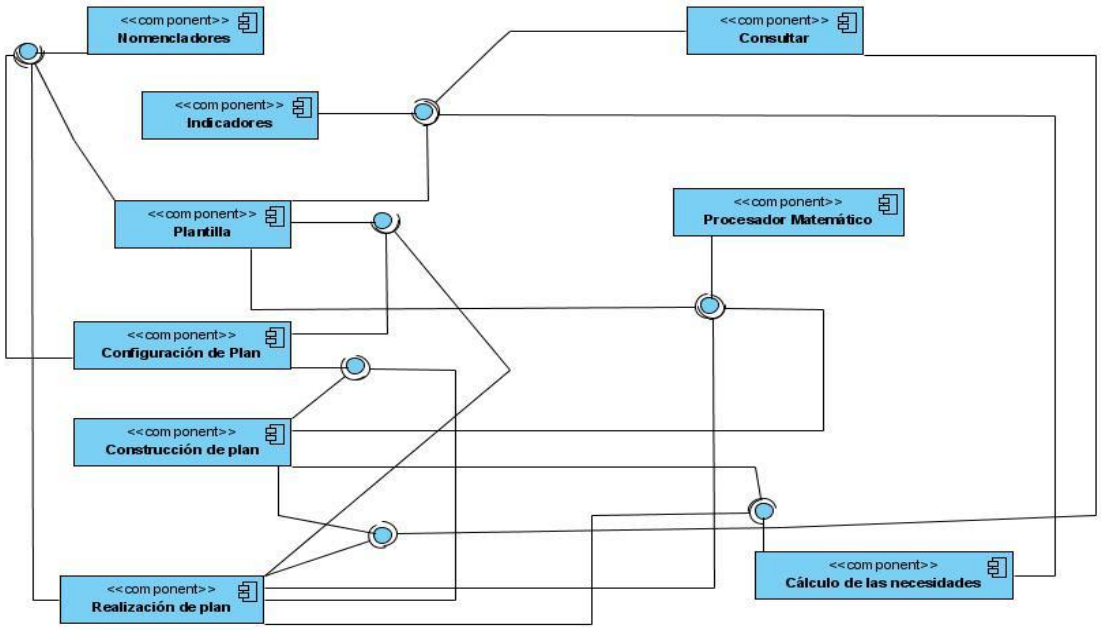


Figura 23. Mapa de componentes.

3. Especificar componentes

Esta actividad tiene como propósito especificar cada componente definido:

3.1 Servicios que provee

Tabla 14. Servicios del componente.

Componente				
Servicio	**Código**	**Descripción**	**Entrada**	**Salida**
<<Nombre del servicio>>	<<Subsistema>> <<Componente>> <<No. consecutivo>>	<<Descripción del servicio>>	<<Atributos de entrada>>	<<Atributos de salida>>

3.2 Reutilización
<<Aquí se especifican las reutilizaciones de otros componentes o elementos externos en la solución del subsistema. >>

3.3 Patrones y estilos arquitectónicos
<<Aquí se describen los patrones y estilos a utilizar en el componente. Incluir descripción de forma de empleo en la solución propuesta. >>

3.4 Clases
<< Clases del diseño>>

4. Describir la integración de componentes

Esta actividad tiene como propósito representar todos los componentes definidos, de forma matricial, y en las intercepciones se especificarán los servicios que consume el componente en la vertical del horizontal.

Tabla 15. Matriz de integración.

Componentes	**<<Nombre del componente 1>>**
<<Nombre del componente 1>>	<<Códigos de los servicios que brinda Componente 1 y consume Componente 2>>

5. Integrar los elementos de la arquitectura

Esta actividad tiene como propósito agrupar en un solo artefacto todas las responsabilidades arquitectónicas de un subsistema, la comunicación y dependencias entre los componentes y módulos que lo integran, así como plasmar la criticidad y complejidad de los componentes con vistas a su reutilización y prioridad en el desarrollo. Se utiliza para conformar la línea base de la arquitectura del sistema.

6. Diseñar el modelo de despliegue

Esta actividad tiene como propósito representar la configuración del hardware en la cual se desplegará el sistema identificando nodos procesadores, dispositivos y protocolos además de las conexiones entre estos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos.

- **Procesadores:** Nodos que tienen capacidad de procesamiento, computadoras por lo general. Ej. Maquina Cliente, Servidor de Datos, Servidor Web, Servidor de Aplicaciones, Servidor de Correo.
- **Dispositivos:** Nodos que no tienen capacidad de procesamiento. Ej. Impresora, Scanner, WebCam, Lector de Tarjeta.
- **Protocolos:** Estándares que deben existir implementados en la red entre máquinas, para efectuar cierta comunicación. EJ. SMTP, POP3 para correo, RMI para componentes distribuidos sobre Java, DCOM para componentes distribuidos de Microsoft, ADO, JDBC, OLE-DB para encuestar bases de datos.

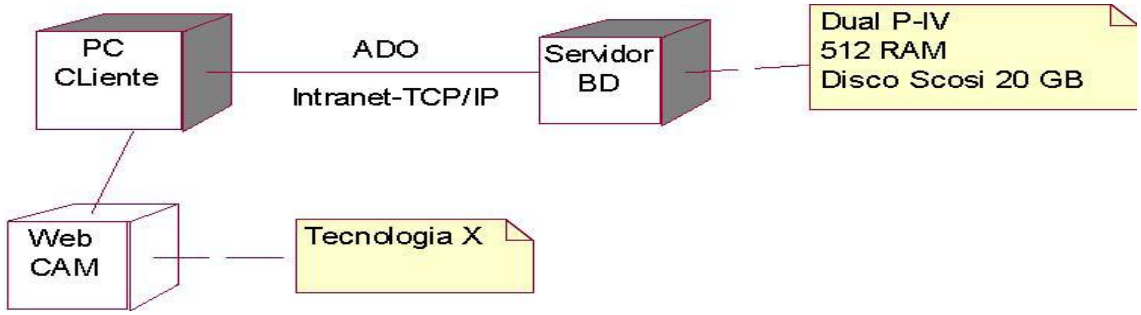


Figura 24. Diagrama de despliegue.

7. Diseñar clases

Esta actividad tiene cómo propósito definir diagramas de clases, los cuales son de estructura estática. Se representan además sus atributos, responsabilidades y cómo se relacionan entre ellas. Con esta actividad se pueden representar tanto las clases no persistentes como las persistentes.

8. Realizar diagrama de interacción

Esta actividad tiene como propósito representar cómo interactúan estas clases no persistentes en los diferentes escenarios del requisito. La mayoría de estas clases existe mientras dure la interacción, lo cual se representa con sus líneas de vida dibujadas desde arriba hasta abajo y presentan un foco de control que es un rectángulo delgado y estrecho que representa el período de tiempo durante el cual un objeto ejecuta una acción.

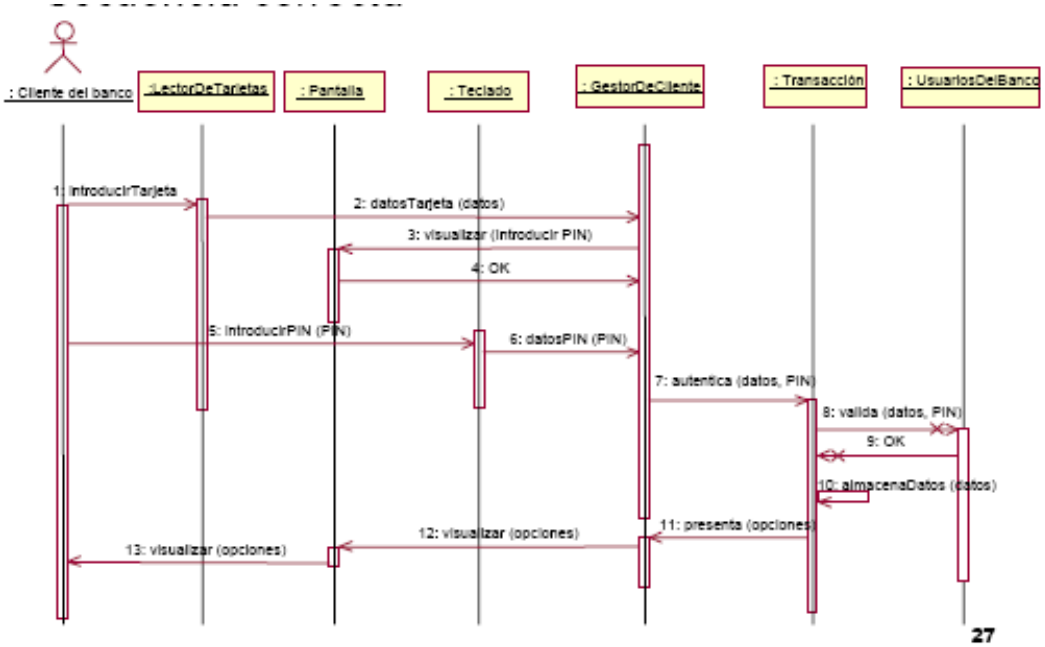


Figura 25. Diagrama de interacción.

9. Integrar los elementos del diseño

Esta actividad tiene como propósito agrupar en un sólo artefacto todos los artefactos del diseño el cual se utiliza como una abstracción para la implementación.

10. Diseñar interfaz de usuario

Esta actividad tiene como propósito describir cómo se comunica el software consigo mismo, con otros sistemas que operen junto a él o con los usuarios. Posibilita que los usuarios tengan una idea en fases tempranas de lo que será el software. Para ello se tiene en cuenta estándares de usabilidad, de apariencia de la interfaz, iconos, vistas y representaciones visuales de los objetos, así como los menús de los objetos y ventanas.

11. Diseñar la base de datos

Esta actividad tiene como propósito representar lógica y físicamente la información persistente manejada por el sistema. Se crea a partir de un conjunto de clases del diseño persistentes en el modelo de diseño aunque pudiera ser creado también a partir de una ingeniería inversa de un almacenamiento de datos existente (base de datos). Incluye tablas, índices, vistas, restricciones, triggers, procedimientos almacenados y cualquier otro elemento que se necesite para almacenar, recuperar y eliminar objetos persistentes.

12. Especificar los datos

Esta actividad tiene como propósito describir en un artefacto todas las tablas del modelo de datos especificando de cada una de ellas nombre, esquema en que se encuentra, atributos (Nombre, tipo de dato, pk/fk, null, descripción) y relaciones (de, de multiplicidad, desde multiplicidad).

13. Describir trazabilidad

Esta actividad tiene como propósito representar la trazabilidad de los datos.

Tabla 16. Matriz de trazabilidad.

Matriz de trazabilidad					
Líneas entrada →	**Subsistema**				
Líneas salida ↓	**Tabla origen**	**Variable**	**Relación**	**Tabla destino**	**Variable**
Subsistema					

14. Especificar réplica

Esta actividad tiene como propósito especificar cómo se realizará la réplica de los datos.

Tabla 17. Réplica de la información.

Réplica de la información						
Tabla	**Origen de creación**	**Entidad lo puede cambiar**	**Replica hacia abajo**	**Replica hacia arriba**	**Estático**	**Dinámico**

15. Evaluar el rendimiento

Esta actividad tiene como propósito asegurar que la base de datos realiza de manera rápida y correcta todas las operaciones. Se obtiene como resultado un documento donde se especifican las pruebas realizadas.

Resumen

Tabla 18. Entradas, actividades y salidas de la disciplina requerimientos.

Diseño		
Entradas	**Actividades**	**Salidas**
▪ Listado de requisitos identificados ▪ Especificación de requisitos ▪ Matriz de seguimiento ▪ Modelo conceptual	▪ Identificar componentes ▪ Especificar componentes ▪ Realizar diagrama de componentes ▪ Describir la integración de componentes ▪ Integrar los elementos de la arquitectura	▪ Análisis de complejidad y criticidad. ▪ Mapa de componentes ▪ Especificación de componentes ▪ Matriz de integración

▪ Propuesta de solución ▪ Repositorio de componentes ▪ Documento visión ▪ Estándar de diseño de interfaz ▪ Estándar de diseño de base de datos	▪ Diseñar el modelo de despliegue ▪ Diseñar clases ▪ Realizar diagrama de interacción ▪ Integrar los elementos del diseño ▪ Diseñar interfaz de usuario ▪ Diseñar la base de datos ▪ Especificar los datos ▪ Describir trazabilidad ▪ Especificar réplica ▪ Evaluar el rendimiento	▪ Especificación de la arquitectura ▪ Modelo de despliegue ▪ Diagrama de clases ▪ Diagrama entidad-relación ▪ Diagrama de secuencia ▪ Modelo de diseño ▪ Prototipo de interfaz de usuario ▪ Base de datos ▪ Diccionario de datos ▪ Matriz de traza ▪ Réplica ▪ Resultados de las pruebas.
Roles que intervienen ▪ Analistas (línea de desarrollo y funcionales) ▪ Arquitecto de sistema ▪ Líder de factoría ▪ Diseñador ▪ Desarrollador de interfaz de usuario ▪ Arquitecto de datos		

4.9 Implementación

Descripción

La disciplina de implementación comienza con el resultado del diseño y se implementa el sistema en términos de componentes que se convierten en código fuente (Ver anexo XI). Tiene las siguientes actividades:

1. Crear estructura del sistema
2. Implementar interfaz de usuario
3. Implementar clases
4. Integrar clases
5. Realizar pruebas unitarias
6. Validar implementación

Actividades

- Crear estructura del sistema

Esta actividad tiene el propósito de crear la estructura de empaquetamiento del sistema (subsistemas, módulos y componentes) que se especificó en el artefacto especificación de la arquitectura.

- Implementar interfaz de usuario

Esta actividad tiene el propósito implementar las interfaces de usuario utilizando formularios, controles, etc., que permitan procesar e intercambiar la información con los usuarios. Se implementan los prototipos obtenidos en el diseño según los estándares definidos.

- Implementar clases

Esta actividad tiene el propósito de implementar los procesos incluyendo cálculos, restricciones y reglas, todo relacionado al negocio. Se crean los componentes de la aplicación dentro del estándar de estructura física de los componentes, se implementa la lógica de negocio de los componentes según los estándares de codificación y la tecnología seleccionada.

- Integrar clases

Esta actividad tiene el propósito de integrar todas las clases obtenidas: de interfaz de usuario, de negocio, de acceso a datos para darle funcionalidad al componente como un todo.

- Realizar pruebas unitarias

Esta actividad tiene el propósito de realizarle a los componentes pruebas unitarias para verificar su correcto funcionamiento.

- Validar implementación

Esta actividad tiene el propósito de verificar que los componentes se hayan implementado según la especificación de los requisitos, la especificación de la arquitectura y el modelo de diseño. También es necesario realizar pruebas de caja blanca para verificar que el código cumple con los estándares de codificación.

Resumen

Tabla 19. Entradas, actividades y salidas de la disciplina implementación.

Implementación		
Entradas	**Actividades**	**Salidas**
▪ Especificación de la arquitectura ▪ Especificación de requisito ▪ Prototipo de interfaz de usuario ▪ Diagrama entidad-relación ▪ Modelo de diseño	▪ Crear estructura del sistema ▪ Implementar interfaz de usuario ▪ Implementar clases ▪ Integrar clases ▪ Realizar pruebas unitarias ▪ Validar implementación	▪ Estructura de sistema ▪ Interfaz de usuario funcional ▪ Clases de negocio ▪ Clases de acceso a datos ▪ Servicios ▪ Componente ▪ Componente [probado]

		▪ Componente [validado]
Roles que intervienen ▪ Arquitecto de sistema ▪ Desarrollador de interfaz de usuario ▪ Desarrollador de lógica de negocio ▪ Líder de factoría		

4.10 Pruebas

Descripción

Evalúa el sistema contra los requerimientos basado en el modelo de pruebas, el objetivo principal es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que este pudiera tener. Es importante considerar que las pruebas de software no garantizan que un sistema esté libre de errores, sino que se detecten la mayor cantidad de defectos posibles para su debida corrección (Ver anexo XII). Tiene las siguientes actividades:

1. Planificar el aseguramiento de la calidad
2. Revisar artefactos
3. Diseñar casos de prueba
4. Realizar pruebas unitarias
5. Realizar pruebas de integración

Actividades

1. Planificar el aseguramiento de la calidad

Esta actividad permite definir la visión global del proceso de pruebas y revisiones, describiendo los objetivos de ambos procesos, identificando los elementos que serán probados o revisados. Se revisan todos los artefactos obtenidos en el expediente de proyecto con las listas de chequeo que son elaboradas en los comités de roles y los componentes son probados con los diseños de casos de pruebas. La planificación de los procesos de prueba y revisiones, brindan las bases para realizar un control y seguimiento al proyecto a lo largo de todas las fases del proceso de desarrollo, a fin de garantizar además del cumplimiento de los compromisos, la calidad en el proceso. Todo esto sirve como referencia para futuros proyectos y poder realizar análisis e identificar aspectos de mejora.

2. Revisar artefactos

Esta actividad tiene el propósito de evaluar la calidad de los artefactos desarrollados durante el proceso, en cuanto al cumplimiento de las normas, metodologías y estándares aplicables en cada caso. Permite además verificar el cumplimiento del proceso de desarrollo del software establecido en el centro, mantener un control sobre la documentación y eliminar a tiempo los defectos detectados.

3. Diseñar casos de prueba

Esta actividad tiene el propósito de diseñar los casos de prueba, que no son más que un conjunto de condiciones o variables bajo las cuáles se determina si un requisito es parcial o completamente satisfactorio. Su propósito es especificar una forma de probar

el sistema que incluya las entradas, los resultados esperados y las condiciones bajo las que ha de probarse. Los casos de prueba ayudan a validar que el sistema desarrollado realice las funciones para las que ha sido creado en base a los requerimientos del usuario, por lo que resulta importante diseñar al menos un caso de prueba para cada requisito del sistema con el objetivo de asegurar que todas las funcionalidades sean comprobadas.

4. Realizar pruebas unitarias

Esta actividad tiene el propósito de realizar pruebas unitarias al sistema, las cuales constituyen la escala más pequeña de las pruebas, están basadas en la funcionalidad de los módulos o componentes y permiten asegurar que cada uno de estos funcione correctamente por separado. En las pruebas unitarias los errores están más acotados y son más fáciles de localizar, lo que facilita a los implementadores la solución de los mismos permitiendo llegar a la integración con un mayor grado de seguridad del correcto funcionamiento de cada parte del sistema por separado. Para ejecutar las mismas se seleccionan los casos de pruebas que fueron diseñados para cada funcionalidad de forma independiente.

5. Realizar pruebas de integración

Esta actividad tiene el propósito de realizar pruebas de integración, las cuales constituyen el mecanismo para comprobar el correcto ensamblaje del sistema completo. Estas pruebas se realizan al terminar las pruebas unitarias con el objetivo de verificar que los módulos o componentes que conforman un sistema funcionan correctamente una vez que han sido integrados. Para la realización de las mismas debe tenerse en consideración un conjunto de aspectos tales como, el grado de complejidad de los módulos o componentes, su importancia funcional con respecto a las especificaciones del sistema, la interrelación entre los mismos y las estadísticas de error en las pruebas unitarias de forma tal que se prueben primeramente los más críticos. Para ejecutar las mismas se seleccionan los casos de pruebas que fueron diseñados teniendo en cuenta el grado de dependencia entre los componentes para realizar una funcionalidad específica.

Resumen

Tabla 20. Entradas, actividades y salidas de la disciplina de pruebas.

Pruebas		
Entradas	**Actividades**	**Salidas**
▪ Cronograma de la fase de construcción ▪ Expediente de proyecto ▪ Listas de chequeo ▪ Especificación de requisitos ▪ Componentes	▪ Planificar el aseguramiento de la calidad ▪ Revisar artefactos ▪ Diseñar casos de prueba ▪ Realizar pruebas unitarias ▪ Realizar pruebas de integración	▪ Plan de revisiones ▪ Plan de pruebas ▪ No conformidades ▪ Casos de prueba
Roles que intervienen ▪ Especialista de calidad ▪ Analista ▪ Probador		

4.11 Despliegue

Descripción

El objetivo de esta disciplina es transferir el sistema del ambiente de desarrollo al de producción y realizarlo basado en el modelo de despliegue. Esta actividad se planifica con detalle y al final de la ejecución de dicho plan el sistema se encuentra disponible para los usuarios finales (Ver anexo XIII). Tiene las siguientes actividades:

1. Realizar material de soporte
2. Realizar material de entrenamiento
3. Redactar normativas
4. Revisar documentación
5. Realizar capacitación
6. Implantar solución

Actividades

1. Realizar material de soporte

Esta actividad tiene como propósito desarrollar los materiales de soporte como manual de usuario y ayuda del sistema.

2. Realizar material de entrenamiento

Esta actividad tiene como propósito desarrollar los materiales de entrenamiento como clases, cursos, tutoriales, multimedia, etc.

3. Redactar normativas

Esta actividad tiene como propósito redactar las normativas de despliegue, documento donde se describe cómo se va a llevar a cabo la prueba experimental o despliegue, las entidades que estarán involucradas elementos organizativos de la actividad, etc.

4. Revisar documentación

Esta actividad tiene como propósito revisar las normativas para asegurar que todo ha quedado especificado en el documento.

5. Realizar capacitación

Esta actividad tiene como propósito capacitar al equipo de implantación con los materiales que se prepararon para ello.

6. Implantar solución

Esta actividad tiene como propósito implantar el software en las entidades especificadas en las normativas.

Resumen

Tabla 21. Entradas, actividades y salidas de la disciplina de despliegue.

Despliegue		
Entradas	**Actividades**	**Salidas**
▪ Especificación de requisitos ▪ Versión beta del producto	▪ Realizar material de soporte ▪ Realizar material de entrenamiento ▪ Redactar normativas ▪ Revisar documentación ▪ Implantar solución	▪ Manual de usuario ▪ Ayuda de la aplicación ▪ Clases ▪ Tutoriales, multimedia ▪ Normativas para el despliegue ▪ Equipo de implantación [capacitado] ▪ Versión beta del producto [implantada]
Roles que intervienen ▪ Soporte ▪ Usuario		

4.12 Configuración

Descripción

Disciplina que se centra en la administración de la configuración del sistema administrando el acceso a los artefactos del proyecto (Ver anexo XIV). Tiene las siguientes actividades:

1. Asignar permisos repositorio
2. Asignar permisos herramienta gestión de proyecto
3. Configurar usuarios en el gestor documental

Actividades

1. Asignar permisos repositorio

Esta actividad tiene como propósito configurar según el rol que tenga cada usuario el permiso que tiene en el expediente de proyecto: lectura, escritura o ambos. Los roles están definidos en el artefacto proceso de desarrollo y el directorio de equipo asocia cada miembro del proyecto con un rol. De esta forma se garantiza que todos los miembros del equipo tengan el acceso que les corresponde a los artefactos generados durante el desarrollo.

2. Asignar permisos herramienta gestión de proyecto

Esta actividad tiene como propósito configurar los usuarios que tendrán permiso para actualizar la herramienta en la que se lleva la gestión del proyecto, tomando como entrada el directorio del equipo.

3. Configurar usuarios en el gestor documental

Esta actividad tiene como propósito configurar según el directorio del equipo los usuarios que podrán acceder al gestor documental.

Resumen

Tabla 22. Entradas, actividades y salidas de la disciplina de configuración.

Configuración		
Entradas	**Actividades**	**Salidas**
▪ Proceso de desarrollo de software ▪ Directorio de equipo	▪ Asignar permisos repositorio ▪ Asignar permisos herramienta gestión de proyecto ▪ Configurar usuarios en el gestor documental	▪ Permisos repositorio ▪ Herramienta de gestión de proyecto [configurada] ▪ Gestor documental [configurado]
Roles que intervienen ▪ Gestor de configuración		

4.13 Gestión de cambio

Descripción

Su objetivo es la administración de los cambios en el sistema (Ver anexo XV). Tiene las siguientes actividades:

1. Clasificar no conformidad
2. Asignar no conformidad
3. Convocar al comité de gestión de cambio de la línea
4. Archivar no conformidad
5. Analizar no conformidad
6. Convocar al comité de gestión de cambio de la UCID
7. Analizar cambios
8. Asignar cambios
9. Archivar solicitud
10. Resolver no conformidad
11. Revisar no conformidad

Actividades

1. Clasificar no conformidad

Esta actividad tiene como objetivo clasificar la no conformidad en error o mejora.

2. Asignar no conformidad

Esta actividad tiene como objetivo en caso de que la no conformidad sea clasificada como correctiva asignar la no conformidad para que sea arreglada.

3. Convocar al comité de gestión de cambio de la línea

Esta actividad tiene como objetivo en caso de que la no conformidad sea clasificada como mejora convocar al comité de gestión de cambio de la línea productiva para que sea analizada.

4. Archivar no conformidad

Esta actividad tiene como objetivo en caso de que la no conformidad sea clasificada como perfectiva y no sea imprescindible su realización para el buen funcionamiento del sistema, archivar la misma para que pueda ser tomada en cuenta en próximas versiones.

5. Analizar no conformidad

Esta actividad tiene como objetivo en caso de que sea convocado el comité de gestión de cambio de la línea analizar la no conformidad, con parámetros como: complejidad de la misma, cantidad de componentes que afecta y si lleva cambios en la estructura de datos.

6. Convocar al comité de gestión de cambio de la UCID

Esta actividad tiene como objetivo convocar al comité de gestión de cambio de la UCID en caso de que el cambio involucre componentes de otra línea productiva.

7. Analizar cambios

Esta actividad tiene como objetivo analizar la no conformidad teniendo en cuenta elementos como correspondencia del cambio solicitado con las reglas de negocio establecidas en las líneas involucradas, esfuerzo necesario para realizar el mismo, beneficios obtenidos con el cambio, disponibilidad de personal para realizarlo.

8. Asignar cambios

Esta actividad tiene como objetivo asignar los cambios a las líneas necesarias en caso de que se aprueben los mismos.

9. Archivar solicitud

Esta actividad tiene como objetivo archivar los cambios en caso de que no se aprueben realizar los cambios.

10. Resolver no conformidad

Esta actividad tiene como objetivo resolver las no conformidades en caso de que el comité determine que deben realizarse.

11. Revisar no conformidad

Esta actividad tiene como objetivo revisar las no conformidades resueltas para asegurar que se haya realizado lo que el usuario solicitó y que no se hayan introducido errores en el sistema.

Resumen

Tabla 23. Entradas, actividades y salidas de la disciplina de gestión de cambio.

Gestión de cambio		
Entradas	**Actividades**	**Salidas**
▪ No conformidad	▪ Clasificar no conformidad ▪ Asignar no conformidad ▪ Convocar al comité de gestión de cambio de la línea ▪ Archivar no conformidad ▪ Analizar no conformidad ▪ Convocar al comité de gestión de cambio de la UCID ▪ Analizar cambios ▪ Asignar cambios ▪ Archivar solicitud ▪ Resolver no conformidad ▪ Revisar no conformidad	▪ Comité de gestión de cambio de la línea [convocado] ▪ No conformidad [archivada] ▪ Respuesta de usuario ▪ Solicitud ▪ Acta de aceptación de cambios ▪ Acta de no aceptación de cambios ▪ Actualización ▪ Acta de liberación
Roles que intervienen ▪ Líder de gestión ▪ Comité de gestión de cambio de la línea ▪ Comité de gestión de cambio del centro ▪ Líder de factoría		

4.14 Ambiente

Descripción

Disciplina que se considera de apoyo y se centra en la configuración de las actividades necesarias para configurar el proceso para un proyecto. Describe las actividades que se requieren para desarrollar las pautas que soportan el proyecto, proporcionando a la organización el ambiente de desarrollo de software, definiendo los procesos y las herramienta que soportarán el resto de las disciplina para la ejecución satisfactoria de todas las actividades (Ver anexo XVI). Tiene las siguientes actividades:

1. Adaptar el proceso
2. Obtener las herramientas para el proceso
3. Capacitar el capital humano

Actividades

1. Adaptar el proceso

Esta actividad tiene como objetivo definir cómo el proyecto va a utilizar el proceso de desarrollo de software, adaptándolo para que coincida con las necesidades específicas del proyecto. Proporciona a los miembros del equipo la orientación para ejecutar todos los procesos configurados para el proyecto, los educa y asesora sobre su utilización.

2. Obtener las herramientas para el proceso

Esta actividad tiene como objetivo describir el hardware y las herramientas de software a usar como parte del desarrollo.

3. Capacitar el capital humano

Esta actividad tiene como objetivo entrenar al personal sobre el proceso de desarrollo configurado o las herramientas a través de actividades de entrenamiento que deben ser planificadas, cursos, concentraciones prácticas, asesoramientos, talleres, etc. Debe trasferir el conocimiento y las responsabilidades a los miembros del proyecto.

Resumen

Tabla 24. Entradas, actividades y salidas de la disciplina de ambiente.

Ambiente		
Entradas	**Actividades**	**Salidas**
▪ Proceso de desarrollo de software ▪ Expediente de proyecto	▪ Adaptar el proceso ▪ Obtener las herramientas para el proceso ▪ Capacitar el capital humano	▪ Proceso de desarrollo de software [configurado] ▪ Expediente de proyecto [configurado] ▪ Herramientas ▪ Capital humano [capacitado]
Roles que intervienen ▪ Ambiente		

4.15 Soporte

Descripción

Disciplina que se encarga de brindar soporte al sistema, una vez que el mismo se encuentra en un ambiente fuera de la producción el cual puede ser en explotación experimental o despliegue, con el objetivo de mantener la diversificada base de clientes satisfechos (Ver anexo XVII). Tiene las siguientes actividades:

1. Registrar no conformidad
2. Recibir no conformidad
3. Enviar respuesta
4. Conciliar

5. Gestionar no conformidad
6. Entregar actualización
7. Actualizar
8. Validar solución
9. Cerrar no conformidad

Actividades

1. Registrar no conformidad

Esta actividad tiene como objetivo registrar las no conformidades que los usuarios finales encuentran durante las fases de explotación experimental o despliegue.

2. Recibir no conformidad

Esta actividad tiene como objetivo recibir las no conformidades que envían los usuarios y aceptarlas o rechazarlas. Son rechazadas aquellas solicitudes que no procedan por desconocimiento del usuario. Se aceptan todas aquellas que sean errores de funcionamiento del sistema o propuestas de mejoras del mismo.

3. Enviar respuesta

Esta actividad tiene como objetivo informar a los usuarios de las no conformidades que son rechazadas.

4. Conciliar

Esta actividad tiene como objetivo conciliar con los usuarios las no conformidades rechazadas y si están de acuerdo proceder al cierre de las mismas.

5. Gestionar no conformidad

Esta actividad tiene como propósito gestionar la no conformidad que viene de la prueba experimental o del despliegue, esta gestión se especifica en la disciplina gestión de cambio. La misma devuelve un resultado que puede o no, conllevar a una actualización del software, el cual es informado a la persona que desempeña el rol de soporte.

6. Entregar actualización

Esta actividad tiene como propósito entregar el instalador que resuelve las no conformidades para actualizar la versión beta del producto.

7. Actualizar

Esta actividad tiene como propósito actualizar el software si la gestión de la no conformidad conllevó a un arreglo en el mismo.

8. Validar solución

Esta actividad tiene como propósito que el usuario valide si con la actualización realizada se resolvió la no conformidad.

9. Cerrar no conformidad

Esta actividad tiene como propósito cerrar las no conformidades que sean validadas por los usuarios finales.

Resumen

Tabla 25. Entradas, actividades y salidas de la disciplina de soporte.

Ambiente		
Entradas	**Actividades**	**Salidas**
▪ Versión beta del producto [implantada]	▪ Registrar no conformidad ▪ Recibir no conformidad ▪ Enviar respuesta ▪ Conciliar ▪ Gestionar no conformidad ▪ Entregar actualización ▪ Actualizar ▪ Validar solución ▪ Cerrar no conformidad	▪ No conformidad [registrada] ▪ Actualización ▪ No conformidad [cerrada] ▪ Solución aceptada
Roles que intervienen ▪ Usuario ▪ Soporte ▪ Líder de gestión		

5. *Prácticas*

Las metodologías ágiles proponen algunas prácticas que combinadas con las actividades que los diferentes roles realizan pueden ayudar a obtener un mejor resultado de la misma. A continuación de propone un conjunto de estas, para ser utilizadas.

5.1 El juego de la planeación

✓ Combinar prioridades de negocio definidas por el cliente con las estimaciones técnicas de los arquitectos basados en la propuesta inicial de solución.

En todas las fases propuestas existe la actividad planificación que tiene como artefacto de salida el cronograma de ejecución para la próxima fase. La práctica juego de planeación combina prioridad de cliente con estimaciones de tiempo lo cual ayuda a corregir que se planifique para las primeras entregas lo que menos valor de negocio tiene para el cliente o lo que más esfuerzo pudiera llevar en el desarrollo.
En esta planificación se definen iteraciones con una duración entre 30 y 45 días. El resultado es un incremento ejecutable que se muestra al cliente.

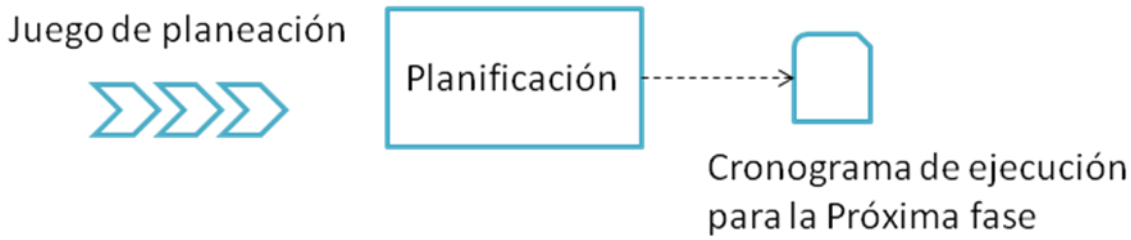


Figura 26. Juego de planeación.

5.2 Diseño simple

- ✓ Diseñar la solución más simple, eliminando complejidades innecesarias y código extra, definiendo la menor cantidad de clases posible, realizando sesiones de diseño de 10 a 30 minutos y minimizando diagramas.

Combinar con las actividades diseñar clases y realizar diagrama de interacción que se encuentran en la disciplina diseño, para evitar que: se extienda en el tiempo la misma generando exceso de código.

5.3 Refactorización continúa

- ✓ Se reestructura el código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. El proceso consiste en una serie de pequeñas transformaciones que modifican la estructura interna preservando su conducta aparente.

Combinar con la actividad integrar clases que se realiza después de haber codificado y se pueden detectar estos elementos que se pueden mejorar en el código.

5.4 Propiedad colectiva del código

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

Combinar con las actividades de codificación, en la disciplina de implementación y de resolver no conformidad, en la de gestión de cambio de manera que todos los programadores tengan conocimiento del código.

5.5 Integración continua

- ✓ Cada componente es integrado al sistema a medida que se construye. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

Combinar con la actividad implementar clase de la disciplina implementación, de esta manera no se espera al final de la construcción para integrar los componentes disminuyendo los riesgos de integración.

5.6 Estándares de programación

- ✓ La comunicación de los programadores es a través del código, con lo cual es indispensable seguir algunos estándares de programación para mantener el código legible.

Combinar con la actividad implementar clase de la disciplina implementación, seguir los estándares de codificación definidos.

Conclusiones

- Para desarrollar software se necesita una guía que indique qué hacer, quién debe hacer, cómo hacerlo y en qué momento.
- Se propusieron seis fases: Pre-fase, Inicio, Modelación, Construcción, Explotación experimental y Despliegue para dividir el proceso de desarrollo y se definieron los hitos de cada una de ellas.
- Se propusieron disciplinas agrupadas en tres grupos fundamentales: empresa, desarrollo y soporte. Se describieron las actividades a desarrollar en cada una de ellas y los artefactos de entrada y de salida.

Capítulo 3. Validación de la solución

Introducción

En este capítulo se lleva a cabo la validación de la solución propuesta. Se comienza realizando un análisis de los resultados de la encuesta, realizada a los líderes de los proyectos que utilizaron la configuración de RUP para desarrollar la primera versión, la cual muestra los principales problemas a los que se enfrentaron. Luego se valida la propuesta utilizando: el método de expertos Delphi y la matriz de perfil competitivo (MPC) propuesta en [Méndez, 2006]. Por último se muestran los resultados de la aplicación de la misma en cuatro proyectos que se desarrollan en la UCID que se consideran segundas versiones de los referenciados en la encuesta.

1. *Situación de los proyectos de desarrollo de software de la UCID que aplicaron RUP y sus consecuencias*

Haciendo un análisis de los resultados de la encuesta realizada a los líderes de los 5 proyectos de la UCID que entregaron su primera versión al cliente en el 2008 se identificaron las fases de RUP en que se presentaron los mayores problemas. En la tabla 26 se han especificado las fases ejecutadas y se representa con una x aquellas que presentaron problemas durante su ejecución.

Tabla 26. Fases con problemas.

Fases	Proyectos					Total
	A	B	C	D	E	
Inicio	x		x	x	x	4
Elaboración		x	x	x		3
Construcción					x	1
Transición	x	x	x	x	x	5

Revisando esta información se puede concluir que la fase con mayores problemas para ejecutarse fue la de transición y que detrás de esta se ubicaron inicio y elaboración respectivamente, mientras que la que menos problemas presentó fue la de construcción. En la figura 25 se muestran las causas que más contribuyeron a los contratiempos.

Figura 27. Total de causas que identificaron los proyectos.

Como se pudo apreciar en la gráfica las causas que más seleccionaron los proyectos fueron la realización de actividades y artefactos inadecuados. En el anexo XVIII se muestran una serie de gráficas que representan la cantidad de artefactos desarrollados

por disciplinas. Como se puede observar los artefactos mas realizados en la disciplina de negocio fueron los diagramas de casos de uso del negocio y los diagramas de actividad seguidos por las descripciones textuales de los mismos. En la disciplina de requerimientos los requerimientos funcionales y el modelo de casos de uso. En la disciplina de análisis los paquetes del análisis y en diseño los modelos de despliegue y de datos seguidos del documento de descripción de la arquitectura. En la disciplina de implementación los componentes y los paquetes o subsistemas de implementación, en pruebas las no conformidades y los diseños de casos de prueba. En despliegue los manuales de usuario y en gestión de la configuración y el cambio las solicitudes de cambio.

En la gráfica 26 que se muestra a continuación se puede observar como la mayor parte de los proyectos considera que la calidad de los artefactos es regular y en la 27 se muestran las causas.

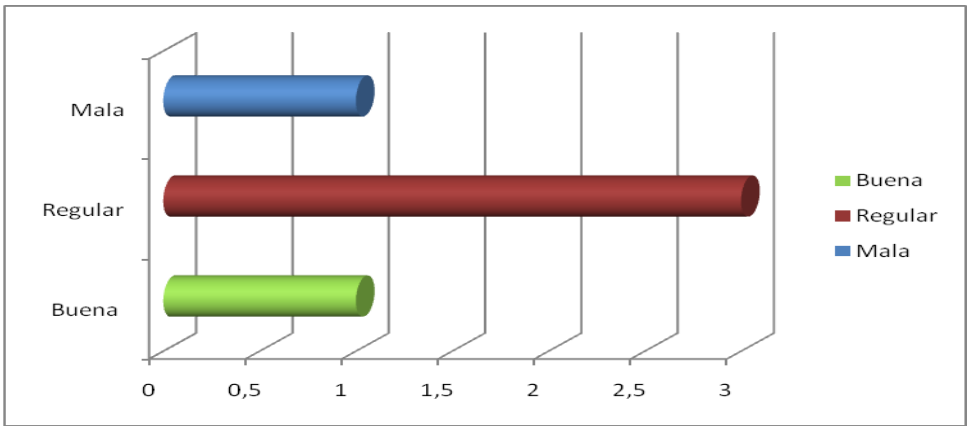


Figura 28. Calidad de los artefactos.

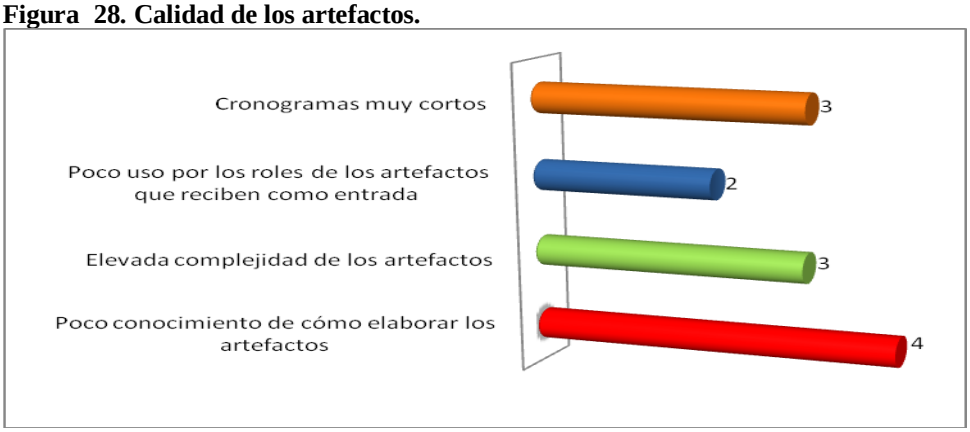


Figura 29. Causas de la poca calidad.

En la tabla 27 se muestra el tiempo de desarrollo de los proyectos, donde se puede observar que solo uno se desarrolló en menos de un año.

Tabla 27. Tiempo de desarrollo de los proyectos.

Tiempo	Proyectos					Total
	A	B	C	D	E	
Menos de 1 año		x				1
Menos de 2 años			x	x		2
Más de 2 años	x				x	2

Como se puede observar en la tabla la mayor parte de los proyectos se demoraron más de un año en entregar la primera versión.

En la tabla 28 se muestra la cantidad de no conformidades recibidas de los proyectos al realizar la primera entrega.

Tabla 28. Cantidad de no conformidades reportadas.

Tiempo	Proyectos					Total
	A	B	C	D	E	
Ninguna						
Muy pocas						
Pocas						
Muchas		x		x	x	3
Muchísimas	x		x			2

Los proyectos que seleccionaron la opción Muchas presentaron valores entre 50 y 80 no conformidades y los que seleccionaron Muchísimas valores entre 100 y 150. A continuación en la figura 30 se muestran las opciones seleccionadas como posibles causas de la aparición de las mismas.

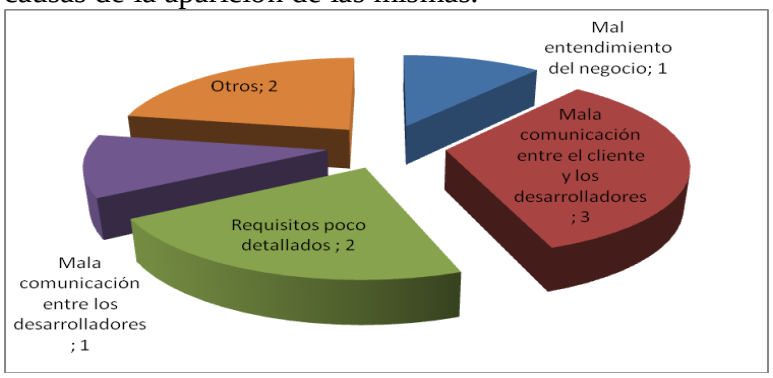


Figura 30. Causas de las no conformidades.

Las causas que más influyeron en la aparición de no conformidades fueron la mala comunicación entre el cliente y el equipo de desarrollo y los requerimientos poco detallados.

2. *Implementación del método Delphi*

Para llevar a cabo la validación de la solución propuesta se utilizó el método de expertos Delphi, aplicándose la encuesta que aparece en el anexo II. Se seleccionaron 9 posibles expertos y se evaluaron sus conocimientos y habilidades relacionadas con:

12. Proceso de desarrollo de software
13. Metodologías de desarrollo de software
    a. Tradiciones
    b. Ágiles

Se estableció el coeficiente de competencia para cada posible experto a partir de la fórmula:

$$K = \frac{1}{2}(kc + ka)$$

Donde: **K:** Coeficiente de competencia de cada experto.
kc: Coeficiente de conocimiento o información que tiene el experto acerca del problema, calculado sobre la valoración del propio experto en una escala del 0 al 10 y multiplicado por 0,1. La evaluación "0" indica que éste no tiene ningún conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa que el experto tiene pleno

conocimiento de la problemática tratada. El experto deberá marcar con una cruz en la casilla que estime pertinente. (Ver anexo XIX)
ka: Coeficiente de argumentación o fundamentación de los criterios del experto, obtenido como resultado de la suma de los puntos alcanzados a partir de una tabla patrón como se muestra en la tabla 29. (Ver anexo XX)

Tabla 29. Elementos para seleccionar el coeficiente de competencia de los expertos.

Fuentes de argumentación	**Grado de influencia de cada una de las fuentes en sus criterios**		
	A (Alto)	**M(Medio)**	**B(Bajo)**
Análisis teóricos realizados por usted	0.3	0.2	0.1
Su experiencia adquirida	0.5	0.4	0.2
Conocimiento de trabajos de autores nacionales	0.05	0.05	0.05
Conocimiento de trabajos de autores extranjeros	0.05	0.05	0.05
Su conocimiento del estado del problema en el extranjero	0.05	0.05	0.05
Su intuición	0.05	0.05	0.05

El coeficiente de competencia calculado para cada experto se muestra en la tabla 30.

Tabla 30. Coeficiente de competencia calculado para cada experto.

Experto	Coeficiente de competencia	Coeficiente alto Si 0,8 < K < 1,0	Coeficiente medio Si 0,5 < K < 0.8	Coeficiente medio Si K < 0,5
1	0,85	x		
2	0,85	x		
3	0,85	x		
4	0,75		x	
5	0,85	x		
6	0,85	x		
7	0,75		x	
8	0,7		x	
9	0,9	x		

Todos los coeficientes de los posibles expertos se encuentran por encima de 0,25 que es el rango mínimo necesario. De los 9 expertos posibles se decidió escoger aquellos cuyo coeficiente se ubicó en el rango establecido para alto y uno con medio, además de tener en cuenta, la disposición para participar con sus opiniones en el desarrollo de la investigación, la efectividad de su actividad profesional y su experiencia profesional. Finalmente se seleccionaron 7 expertos.
Se envió una síntesis de la propuesta de metodología de desarrollo de software para su utilización en la Unidad de compatibilización y desarrollo de productos informáticos para la defensa (UCID) a cada experto seleccionado y el cuestionario que aparece en el anexo II para que evaluaran la misma. El cuestionario elaborado consta de nueve preguntas de las cuales las dos primeras están encaminadas a evaluar la importancia de la metodología en el proceso de desarrollo de software que sirva de guía para lograr la obtención de un producto funcional con calidad en el menor tiempo posible que satisfaga las expectativas del usuario.

Las seis preguntas restantes están enfocadas en evaluar la adecuación de la propuesta en cuanto a:

- Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.
- Adecuación de las fases propuestas
- Adecuación de las disciplinas propuestas
- Adecuación de las actividades propuestas
- Adecuación de los artefactos propuestos
- Adecuación de las técnicas propuestas

Finalmente la última pregunta pretende recoger cualquier elemento ya sea señalamiento o recomendación que pudiera contribuir con el desarrollo de la investigación.
Las respuestas a estas preguntas han sido categorizadas en: muy adecuado (C1), bastante adecuado (C2), adecuado (C3), poco adecuado (C4), no adecuado (C5).

2.1 Análisis de los resultados

Al evaluar la importancia de la metodología en el proceso de desarrollo de software para la obtención de un producto funcional con calidad que satisfaga las expectativas del cliente los expertos coincidieron en la necesidad de que la misma sea una guía para el desarrollo y no un impedimento y que por lo tanto cada miembro del equipo debe conocerla en detalle y saber qué papel juega dentro de la misma. En cuanto a la validación de la propuesta, se utilizaron las siguientes herramientas:

- Microsoft Excel.
- SPSS 13.0 para Windows.

Para llevar a cabo el análisis de los resultados se utilizaron métodos estadísticos tal y como indica el método Delphi. La tabla 31 muestra la frecuencia absoluta.

Tabla 31. Frecuencias absolutas.

No	Aspectos	C1	C2	C3	C4	C5	total
1	Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.	5	2	0	0	0	7
2	Adecuación de las fases propuestas	6	1	0	0	0	7
3	Adecuación de las disciplinas propuestas	7	0	0	0	0	7
4	Adecuación de las actividades propuestas	4	3	0	0	0	7
5	Adecuación de los artefactos propuestos	5	2	0	0	0	7
6	Adecuación de las técnicas propuestas	4	1	1	0	0	7

A partir de la frecuencia absoluta se obtienen las frecuencias acumuladas donde los datos de cada fila (menos la primera) se obtienen a partir de la suma de la anterior tal y como lo muestra la tabla 32.

Tabla 32. Frecuencias absolutas acumuladas.

No	Aspectos	C1	C2	C3	C4	C5
1	Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.	5	7	7	7	7
2	Adecuación de las fases propuestas	6	7	7	7	7
3	Adecuación de las disciplinas propuestas	7	7	7	7	7
4	Adecuación de las actividades propuestas	4	7	7	7	7
5	Adecuación de los artefactos propuestos	5	7	7	7	7
6	Adecuación de las técnicas propuestas	4	5	6	6	6

La frecuencia absoluta acumulada se divide entre la cantidad de expertos y se obtiene la frecuencia relativa acumulada tal y como aparece en la tabla 33. En la frecuencia relativa acumulada desaparece la columna C5.

Tabla 33. Frecuencias relativas acumuladas.

No	Aspectos	C1	C2	C3	C4
1	Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.	0,5554	0,7776	0,7777	0,7777
2	Adecuación de las fases propuestas	0,6665	0,7776	0,7777	0,7777
3	Adecuación de las disciplinas propuestas	0,7776	0,7776	0,7777	0,7777
4	Adecuación de las actividades propuestas	0,4443	0,7776	0,7777	0,7777
5	Adecuación de los artefactos propuestos	0,5554	0,7776	0,7777	0,7777
6	Adecuación de las técnicas propuestas	0,4443	0,5554	0,6666	0,6666

Por último se buscan las imágenes de las frecuencias relativas acumuladas por medio de la función (Distribución Normal. Standard Invertida) y se adicionan las siguientes salidas:
Suma: Sumatoria de cada fila y de cada columna según sea el caso.
P: Promedio de la suma de cada fila.
N: División de la sumatoria de las sumas de las filas entre el resultado de multiplicar el número de categorías por el número de pasos.
N-P: Es entonces el valor promedio que le otorgan los expertos consultados a cada paso de la metodología.
Punto de corte: Promedio de la suma de cada columna.

La tabla 34 resume estos resultados:

Tabla 34. Puntos de corte.

No	Aspectos	C1	C2	C3	C4	Suma	P	N-P	
1	Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.	0,13	0,76	0,76	0,76	2,43	0,60	-0,03	Muy Adecuado
2	Adecuación de las fases propuestas	0,43	0,76	0,76	0,76	2,72	0,68	-0,11	Muy Adecuado
3	Adecuación de las disciplinas propuestas	0,76	0,76	0,76	0,76	3,05	0,76	-0,19	Muy Adecuado
4	Adecuación de las actividades propuestas	-0,13	0,76	0,76	0,76	2,15	0,53	0,03	Muy Adecuado
5	Adecuación de los artefactos propuestos	0,13	0,76	0,76	0,76	2,43	0,60	-0,03	Muy Adecuado
6	Adecuación	-0,13	0,13	0,43	0,43	0,86	0,21	0,35	Bastante

	de las técnicas propuestas								adecuado
	Suma	1,19	3,96	4,25	4,25	13,66			
	Ptos de corte	0,19	0,66	0,70	0,70				

Los puntos de corte sirven para determinar la categoría o grado de adecuación de cada paso de la propuesta según la opinión de los expertos consultados. Para determinar cual es el grado de adecuación de cada aspecto a validar se realiza como muestra la tabla 35.

Tabla 35. Grado de adecuación de los aspectos a validar.

Muy adecuado	**Bastante adecuado**	**Adecuado**	**Poco adecuado**	**No adecuado**
N-P=<0,19	N-P=<0,66	N-P=<0,70	N-P=<0,70	

A partir de este análisis cinco de los seis aspectos a validar de la propuesta fueron considerados por los expertos muy adecuados demostrando su utilidad y la adecuación de sus componentes y el sexto fue considerado como bastante adecuado, además se evaluó la concordancia entre criterios de los expertos calculando **W=** 0,01263362 y luego se calculó el **Chi-Square =** 0,44217687, se obtuvo el de las tablas estadísticas (Siegel, 1974) **Chi-Square tabla =** 1,635 y se compararon donde: **Chi-Square calculado < Chi-Square tabla** lo que demuestra que existe concordancia en el criterio de los expertos.

3. *Evaluación de la metodología propuesta según la matriz de perfil competitivo (MPC).*

En el trabajo especial de grado titulado "Modelo de evaluación de metodologías para el desarrollo de software" presentado por Elvia Margarita Méndez en julio del 2006 en Caracas, Venezuela, para optar por el título de especialista en gerencia de proyectos se propone la matriz de perfil competitivo utilizada en el área de planificación estratégica para evaluar las metodologías de desarrollo de software. Para construir la matriz se realizan los siguientes pasos:

1. Identificar las variables claves en el sector, con un mínimo de cinco.
2. Ponderar las variables con valores entre 0,0 y 1,0 para cada una. Tomando en consideración que la sumatoria de las ponderaciones para todos los valores no podrá exceder de 1,0.
3. Clasificar las variables asignando un valor de clasificación para cada una, el cual vendrá especificado de acuerdo a los siguientes criterios:

    - Se asigna un valor de 1 si se considera a la variable como una debilidad mayor.
    - Se asigna un valor de 2 si se considera a la variable como una debilidad menor.
    - Se asigna un valor de 3 si se considera a la variable como una fortaleza menor.
    - Se asigna un valor de 4 si se considera a la variable como una fortaleza mayor.

4. Obtener un resultado ponderado para esto se multiplica el peso de cada variable por el valor asignado para obtener una calificación ponderada.
5. Obtener el total ponderado para cada metodología para lo cual se suman los resultados ponderados de cada una de las variables a fin de determinar el total ponderado para cada una.
6. Comparar las metodologías, haciendo la salvedad que las cifras sólo revelan la fuerza relativa de cada metodología, de manera que sea una ayuda para la toma de decisiones, pero no son una medida exacta [Méndez, 2006].

En el anexo XXI se propone la matriz de perfil competitivo, utilizando las variables definidas en [Méndez, 2006], para evaluar la propuesta y se compara con otras de las metodologías analizadas. Como resultado se obtuvo que las mayores ponderaciones se presentaran en las metodologías EUP y en la propuesta realizada en esta investigación. En este resultado influyeron los altos valores asignados en variables como: el ajuste a los objetivos de la empresa, el cubrir completamente el ciclo de desarrollo, el especificar con exactitud quién obtiene cada resultado, que se debe poder enseñar y que debe soportar la evolución del sistema, entre otros.

4. Aplicación de la propuesta en proyecto piloto

Para validar la propuesta realizada se seleccionaron 4 proyectos que se desarrollan en la UCID y que son nuevas versiones de los desarrollados con la configuración de RUP. Se analizaron los resultados obtenidos en varios momentos diferentes.
En la pre-fase durante la ejecución de la disciplina reutilización estratégica donde se comprobó que para el desarrollo de las cuatro solicitudes analizadas se podían reutilizar componentes que ya se encontraban desarrollados en otros proyectos lo cual contribuye a disminuir el tiempo de desarrollo de la solución.

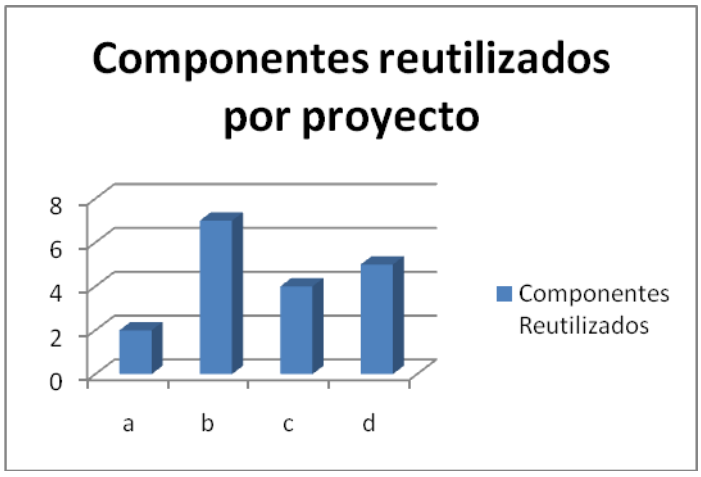


Figura 31. Disciplina reutilización estratégica

También durante la Pre-fase en la ejecución de la disciplina Arquitectura de empresa se obtuvieron resultados satisfactorios pues se detectaron los proyectos que podían reutilizar tecnología y los que necesitaban nuevos desarrollos tecnológicos, antes de ser entregados al equipo de desarrollo permitiendo reorientar los primeros esfuerzos al equipo tecnológico, logrando de esta manera una coordinación de las actividades de los distintos equipos del centro.

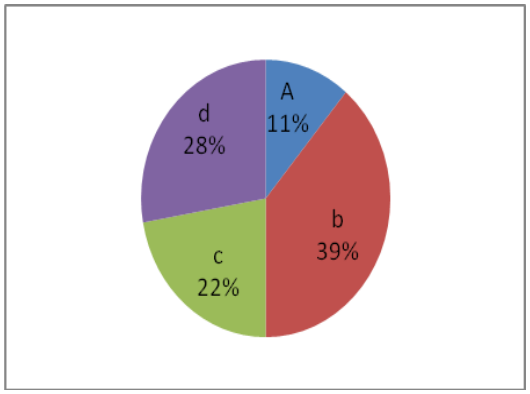


Figura 32. Tecnología reutilizada

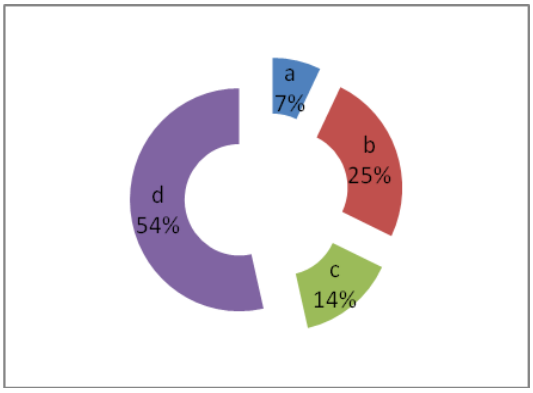


Figura 33. Nuevas tecnologías

En la fase de Modelación se recogieron la cantidad de actividades y artefactos propuestos y los reales desarrollados durante las disciplinas de desarrollo: Modelación del negocio, Requerimientos, Diseño e Implementación, en todos los casos se pudo comprobar que se realizaron la mayoría de las propuestas, como se puede observar en las siguientes figuras.

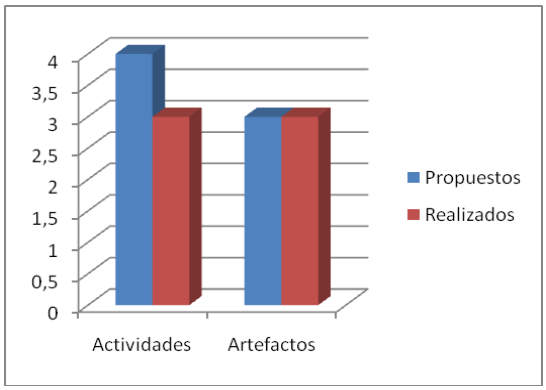


Figura 34. Modelación del negocio

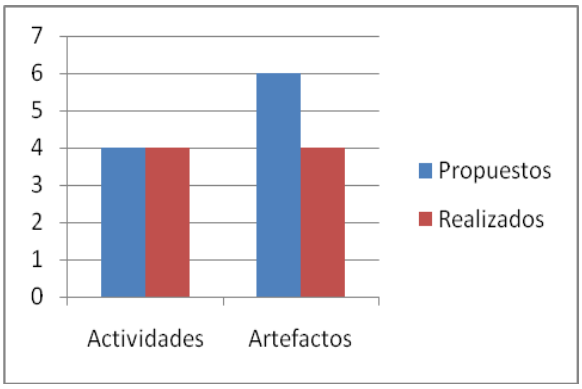


Figura 35. Requerimientos

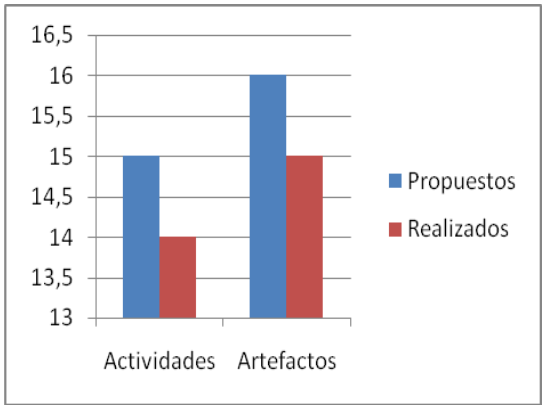


Figura 36. Diseño

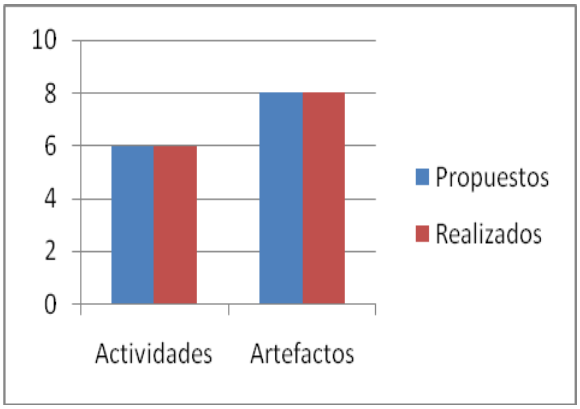


Figura 37. Implementación

Además se recogieron los resultados del tiempo de desarrollo de los proyectos, a continuación se muestran dos gráficas, en la figura 38 se puede observar el tiempo de las primeras versiones y en la 39 el de las segundas, aunque algunos de ellos se encuentran todavía en construcción se percibe por el por ciento en que está dicha fase que el tiempo será inferior.

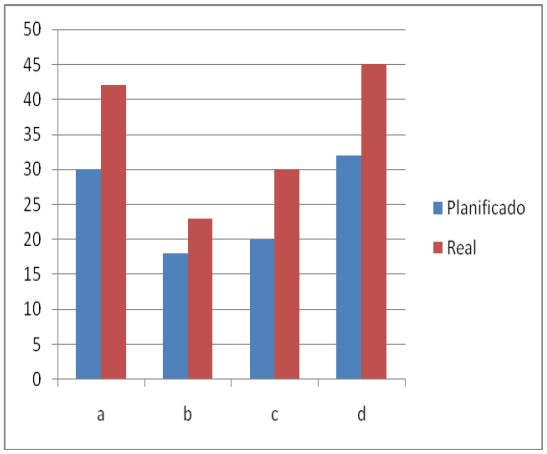


Figura 38. Tiempo de desarrollo 1era versión

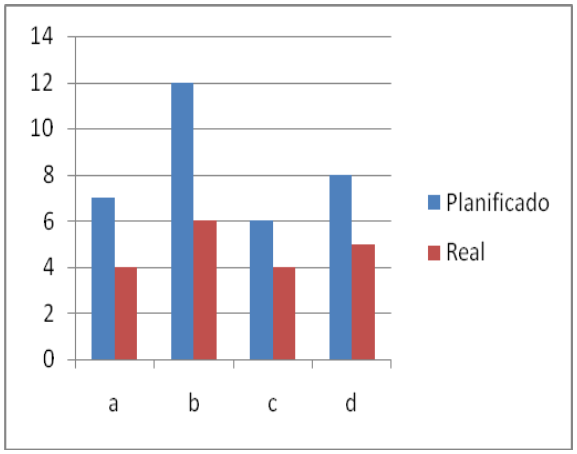


Figura 39. Tiempo de desarrollo 2da versión

Fases de construcción: A = 90%, B= 80 %, C=95%, D= 89%. Por lo tanto se puede observar que el tiempo se ha reducido considerablemente.

Conclusiones

1. Utilizando el método Delphi se evaluó la utilidad de la propuesta de metodología de desarrollo de software para su utilización en la Unidad de compatibilización y desarrollo de productos informáticos para la defensa (UCID), teniendo en cuenta los siguientes aspectos:
   - ✓ Contribución de la propuesta a la obtención de productos con calidad que cumplan las expectativas de los clientes en el menor tiempo posible.
   - ✓ Adecuación de las fases propuestas
   - ✓ Adecuación de las disciplinas propuestas
   - ✓ Adecuación de las actividades propuestas
   - ✓ Adecuación de los artefactos propuestos
   - ✓ Adecuación de las técnicas propuestas

Y los primeros aspectos resultaron evaluados de muy adecuados mientras el último fue bastante adecuado.

2. Utilizando la matriz de perfil competitivo (MPC) que se aplica en las áreas de planificación estratégicas se evaluó la metodología propuesta, utilizando las variables definidas en [Méndez, 2006]. Se obtuvieron los mayores valores en la EUP y la propuesta de esta investigación, este método no es definitivo pero si ayuda a la toma de decisión de la dirección de la entidad.
3. Se aplicó la propuesta en 4 proyectos que se desarrollan en la UCID obteniendo resultados positivos como: mejora en la gestión de los proyectos a nivel de empresa lo que permite optimizar el desarrollo, mayor utilización de las actividades y los artefactos propuestos y disminución del tiempo de desarrollo de los proyectos.

Conclusiones

Una vez realizada esta investigación se concluye que:

1. Las metodologías de desarrollo de software guían el proceso definiendo con precisión quién hace que, cuándo y cómo lo hace, la elección correcta de la misma es una de las claves para lograr el éxito.
2. Los proyectos se encuentran en una disyuntiva al momento de seleccionar la metodología por cual inclinarse debido a la diversidad de propuestas y diferencias en el grado de detalle de la información disponible y alcance de cada una de ellas, para lograr el éxito en esta actividad es necesario escoger la que más se adapta a las características de la entidad y de los proyectos que esta ejecuta.
3. Se propuso una metodología que se adecúa a las características de la UCID y a los proyectos que en ella se desarrollan.
4. La metodología propuesta se divide en fases y especifica los hitos a lograr en cada una de ellas para su completo entendimiento. Las fases se dividen en disciplinas que deben ser recorridas durante el proceso y a su vez estas se descomponen en actividades, que son realizadas por un rol generando artefactos que se convierten en entradas y salidas de las mismas y se propone un número de prácticas ágiles que complementan las actividades.
5. La metodología propuesta fue validada por un grupo de expertos en proceso de desarrollo y metodologías de desarrollo evaluando de muy adecuados y bastante adecuados los elementos de la misma y se le aplicó la matriz de perfil competitivo propuesta en [Méndez, 2006] para validar su adecuación a la entidad obteniéndose resultados positivos.
6. Se aplicó en 4 proyectos pilotos obteniendo resultados positivos como: mejora en la gestión de los proyectos a nivel de empresa lo que permite optimizar el desarrollo, mayor utilización de las actividades y los artefactos propuestos y disminución del tiempo de desarrollo de los proyectos.

Recomendaciones

Con el objetivo de mejorar la solución propuesta se recomienda:

1. Aplicar la metodología propuesta en los diferentes tipos de proyectos que se desarrollan en la UCID.
2. Seleccionar otro conjunto de prácticas ágiles que puedan contribuir a una retroalimentación más efectiva entre clientes y desarrolladores.

Bibliografía

[Abrahamsson, 2002] Abrahamsson, Pekka. *Agile Software development methods: A minitutorial. VTTTechnical Research Centre of Finland,* http://www.vtt.fi/virtual/agile/seminar2002/Abrahamsson_agile_methods_minitutorial.pdf. 2002

[Álvarez, 2007] Álvarez, Juana. *Ingeniería de Software. http://www.educando.edu.do/educanblog/index.php?blogId=435*.2007

[Ambler, 1999] Ambler, Scott W. *Enhancing the Unified Process*. Software Development Magazine, October. www.sdmagazine.com/documents/s=753/sdm9910b/9910b.htm. 1999

[Ambler, 2001] Ambler, S.W. *Agile Modeling and the Unified Process.* www.agilemodeling.com/essays/agileModelingRUP.htm. 2001

[Ambler, 2002] Ambler, S.W. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: John Wiley & Sons. www.ambysoft.com/agileModeling.html. 2002

[Ambler, 2005] Ambler, Scott W. *Introduction to the Enterprise Unified Process (EUP),* October 2005.

[Beck, 1999] Beck, Kent. *Extreme Programming Explained: Embrace Change*. Reading, Addison Wesley, 1999.

[Canós, 2003] Canós, José H. L.P. *Metodologías Ágiles en el Desarrollo de Software*.2003

[Charette, 2004] Charette, Robert. *The decision is in: Agile versus Heavy Methodologies.* Cutter Consortium, Executive Update, 2(19), http://www.cutter.com/freestuff/apmupdate.html, 2004.

[Coad, 2000] Coad, Peter L.D. *Java modeling in color with UML: Enterprise components and process*. Prentice Hall. 2000

[Cockburn, 2002] Cockburn, Alistair. *Crystal Clear. A human-powered methodology for small teams, including The Seven Properties of Effective Software Projects*. Humans and Technology. 2002

[Cooper, 2002] Cooper, R.G. Edgett; S.J, Kleinschmidt, E.J. *Portfolio management – fundamental to new product success" in PDMA toolbook for new product development.* Wiley-Sons. USA. 2002.

[Díaz] Díaz, María I, *La incertidumbre y la ingeniería de software*; http://www.acis.org.co/fileadmin/Revista_102/dos.pdf.

[DSDM, 2003] DSDM Consortium, S. *DSDM: Business Focused Development*. 2ª edición, Addison-Wesley, 2003.

[Figueroa] Figueroa, Roberth G. S.C. *Metodologías tradicionales vs Metodologías agiles.*

[Gabardini] Gabardini, J.L. *Balanceo de Metodologías Orientadas al Plan y Ágiles. Herramientas para la Selección y Adaptación;* http://www.rmya.com.ar/Download/Paper_BMOPA.pdf.

[Gacitúa, 2003] Gacitúa, Ricardo A, *Métodos de desarrollo de software: El desafío pendiente de la estandarización*, 2003.

[Highsmith, 2000] Highsmith, Jim. *Extreme Programming*. EBusiness Application Development, Cutter Consortium, Febrero de 2000.

[Highsmith, 2002a] Highsmith, Jim. *What is Agile Software Development*. Crosstalk, http://www.stsc.hill.af.mil/crosstalk/2002/10/highsmith.html, Octubre de 2002.

[Highsmith, 2002b] Highsmith, Jim. *Agile Software Development Ecosystems*. Boston, Addison Wesley, 2002.

[Hunt, 2006] Hunt, J. (2006), *Agile Software Construction*, Springer-Verlag London 2006, ISBN-13: 978-1-85233-944-9. 2006.

[ISO 12207, 2004] ISO/IEC 12207, *Tecnologías de la información: procesos del ciclo de vida del software*. 2004. www.iso.org/iso/home.htm. 2004.

[Jacobson, 2000] Jacobson I, B. E. R. J. *El proceso unificado de desarrollo de software*, Pearson Education S.A., Madrid. 2000.

[Kniberg, 2007] Kniberg, Henrik. *SCRUM y XP desde las trincheras.Cómo hacemos SCRUM.* InfoQ, 2007.

[Larman 2004] Larman, Craig. *Agile & Iterative Development. A Manager's Guide*. Reading. Addison-Wesley. 2004.

[Medellín, 2006] Medellín, Enrique. *Gestión de cartera de proyectos tecnológicos.* México D.F 2006.

[Méndez, 2006] Méndez, Elvia M. *Modelo de evaluación de metodologías para el desarrollo de software*.2006.

[Mendoza, 2004] Mendoza, María A, *Metodologías de Desarrollo de Software*. http://www.willidev.net/InsiteCreation/v1.0/descargas/cualmetodología.pdf. Jun 2004.

[Microsoft Corporation, 1999] Microsoft Corporation *Microsoft Solutions: The Collaboration in Corporative World*. Microsoft Corporation, Marzo 1999.

[Microsoft Corporation, 2002] Microsoft Corporation. *MSF Process Model v. 3.1*. Microsoft Corporation. http://www.microsoft.com/msf/. June 2002

[Muñoz, 2003] Muñoz, A. *Sistemas de información en las empresas on line*. Hipertext.net, núm. 1, 2003. ISSN 1695-5498.www.hipertext.net. 2003

[Palacio, 2006] Palacio, Juan. *Gestión de proyectos ágil: conceptos básicos*. www.navegapolis.net/files/s/NST-003_01.pdf. 2006

[Peñalver, 2008] Peñalver, Gladys. *MA-GMPR-UR2 Metodología ágil para proyectos de software libre*. http://bibliodoc.uci.cu/TD/TD_1309_07.pdf. 2008.

[Poppendieck, 2001] Poppendieck, Mary. "Lean Programming". http://www.agilealliance.org/articles/articles/LeanProgramming.htm. 2001.

[Premio nacional de tecnología, 2006] Premio nacional de tecnología. *Guía de participación 2005* .México 2005.

[Pressman, 2006] Pressman, R. *Ingeniería del Software. Un enfoque práctico*, McGraw-Hill/Interamericana de España. 2006.

[Rational, 2007] Racional. *Ayuda e línea de Rational Suite 2007®*, www-01.ibm.com/software/awdtools/rup. 2007.

[Riehle, 2000] Riehle, Dirk. *A comparison of the value systems of Adaptive Software Development and Extreme Programming: How methodologies may learn from each other.*http://www.riehle.org/computer-science/research/2000/xp-2000.pdf, 2000.

[Rodríguez, 2007] Rodríguez M. *Introducción de procedimientos ágiles en la producción de software en la Facultad 7 de la Universidad de las Ciencias Informáticas.* http://bibliodoc.uci.cu/TD/TD_0693_07.pdf. 2007.

[Schwaber, 1995] Schwaber, Ken. *The Scrum development process*. OOPSLA '95 Workshop on Business Object Design and Implementation, Austin, 1995.

[Schwaber, 2002] Schwaber Ken, Beedle, Mike. *Agile software development with Scrum*. Prentice-Hall, 2002.

[Siegel, 1974] Siegel S. *Estadística no paramétrica aplicada a las ciencias de la conducta*. p (262-273). México D. F. Ed. Trillas. 1974.

[Stapleton, 1997] Stapleton, Jennifer. *Dynamic Systems Development Method – The method in practice.* Addison Wesley, 1997.

[Weske, 2007] Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures*, cap XIV, 368 p. 265 illus., Hardcover; ISBN: 978-3-540-73521-2 © Springer-Verlag Berlin Heidelberg, www.bpm-book.com. 2007

Anexos

Anexo I: Encuesta a líderes de proyectos productivos de la UCID

Proyecto: _____

Sobre metodología de desarrollo de software:

1. ¿Utiliza la metodología definida en el centro para desarrollar el proyecto?

___ Si __No

2. ¿Considera que la metodología está bien especificada?

___ Si __No

2. ¿En cuál de las fases se presentaron más problemas durante el desarrollo?

__ Inicio
__ Elaboración
__Construcción
__Transición

Seleccione las que considere que fueron las principales causas

___ Mala planificación de la fase
___ Actividades inadecuadas
___ Artefactos inadecuados
___ Problemas de los usuarios
___ Otras

3. Marque con una x los artefactos realizados durante el desarrollo

--- Modelo de dominio
--- Modelo de casos de uso del negocio
--- Descripción de los casos de uso del negocio
--- Diagramas de actividad
--- Descripción de actores, trabajadores y entidades del negocio.
--- Modelo de objetos.
--- Requerimientos funcionales
--- Requerimientos complementarios
--- Modelo de casos de uso.
---Descripción detallada.
--- Modelo de análisis
--- Realización de caso de uso
--- Paquetes del análisis
---Modelo de diseño
--- Documento de arquitectura.
--- Realización de caso de uso de diseño
---Paquetes y subsistemas de diseño
--- Mecanismos de diseño
--- Modelo de despliegue
--- Modelo de datos
--- Modelo de implementación
--- Plan de integración de construcciones

--- Componentes
--- Paquetes o subsistemas de implementación
--- Modelo de pruebas
--- Plan de pruebas
--- Casos de prueba
--- Procedimientos de prueba
--- No conformidades
--- Planes de implantación
--- Clases y tutoriales
--- Manuales de usuario
--- Plan de gestión de la Configuración
--- Plan de gestión del cambio
--- Solicitud de cambio
--- Ambiente de desarrollo
--- Lista de riesgos
--- Plan de mitigación de riesgos

3. Del total de artefactos realizados cuántos considera que contribuyeron en el proceso de desarrollo a la comunicación entre los roles:

Menos de la mitad __
La mitad __
Más de la mitad __

4. ¿Como califica la calidad de los artefactos desarrollados?

Buena (B) __
Regular (R) __
Mala (M) __

En caso de seleccionar R o M cuáles cree que fueron las causas:
__ Poco conocimiento de cómo elaborar los artefactos
__ Elevada complejidad de los artefactos
__ Poco uso, por los diferentes roles, de los artefactos que reciben como entrada
__ Cronogramas muy cortos

5. Tamaño del equipo de desarrollo
Menos de diez __
Entre diez y veinte __
Más de treinta __
6. Experiencia de la dirección del equipo en otros proyectos
Ninguna __
De 1- 3 __
Más de 3 __
7. Presencia de los especialistas funcionales
Permanente __
Eventual __
Ninguna __

8. Tiempo de permanencia de los miembros del equipo en el proyecto

Entre 1 y 2 años __
Hasta 3 años __
Más de 3 años __

9. ¿Que tiempo se demoró el desarrollo de la primera versión del proyecto?
   Menos de 1 año __
   Menos de 2 años __
   Más de 2 años __

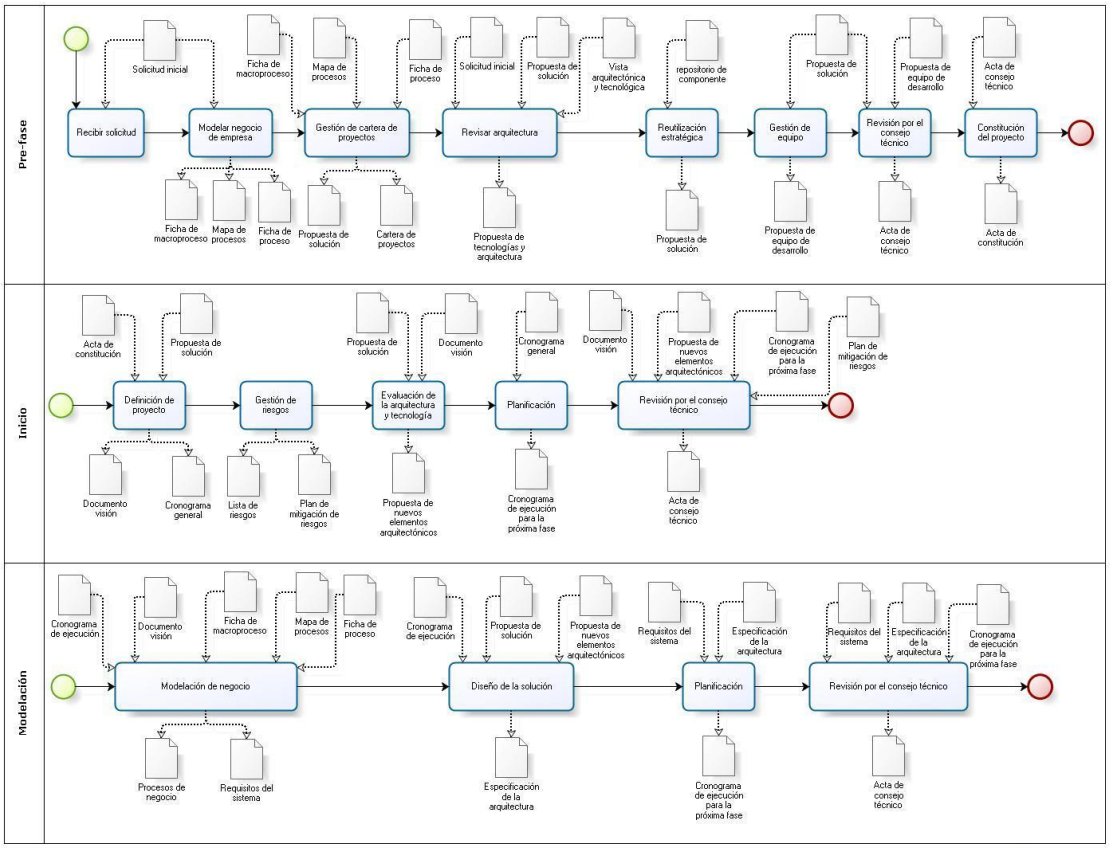
10. ¿Cuántas solicitudes de cambio se obtuvieron al entregar la primera versión?

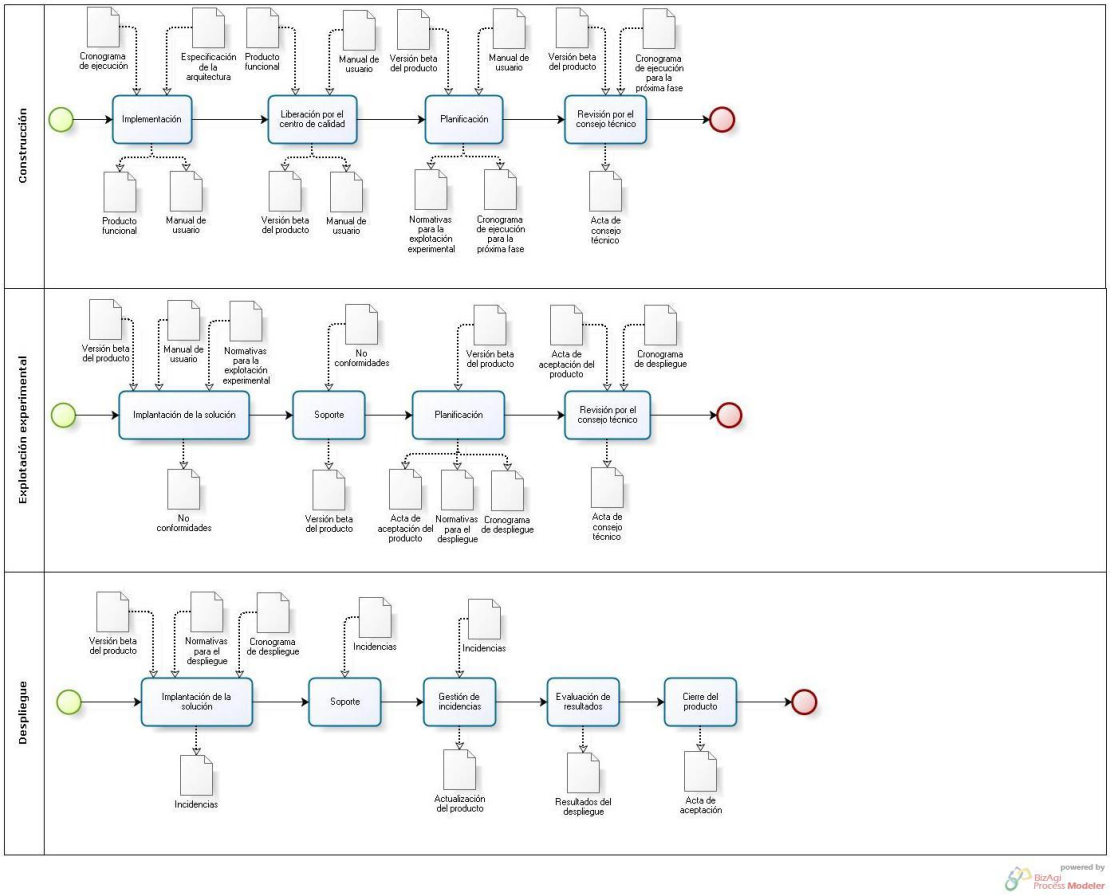
___ Ninguna ___ Muy pocas ___ Pocas ___ Muchas ___ Muchísimas

11. ¿A qué piensa usted se deben las solicitudes de cambio?

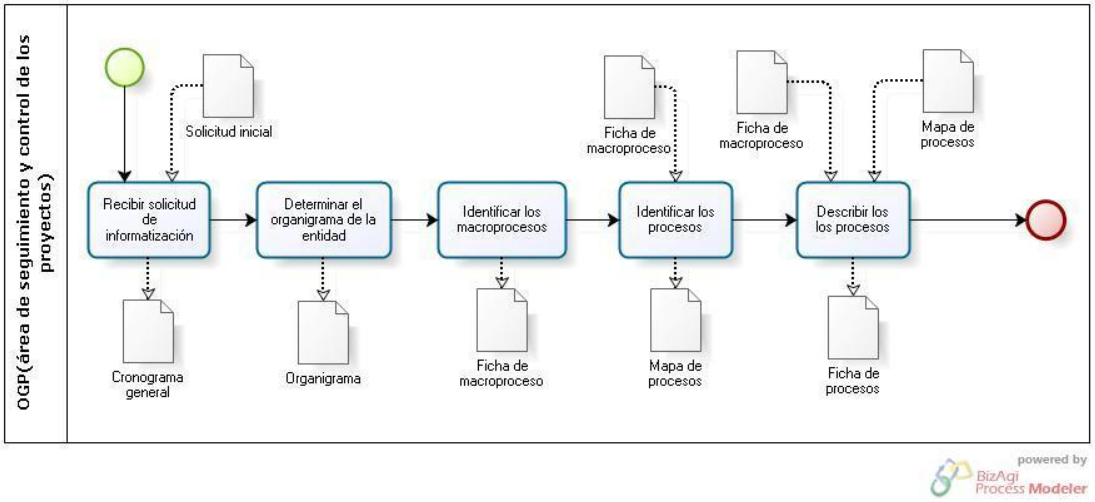
___ Mal entendimiento del negocio
___ Mala comunicación entre el cliente y los desarrolladores
___ Requisitos poco detallados
___ Mala comunicación entre los desarrolladores
___ Problemas del lenguaje de programación o de BD.
___ Otros ¿Cuáles?

Anexo II: Fases y sus hitos principales

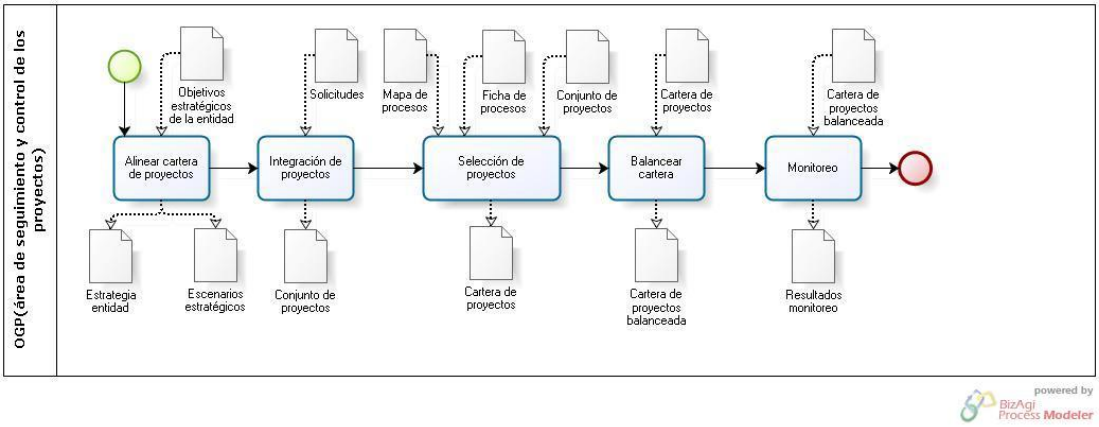




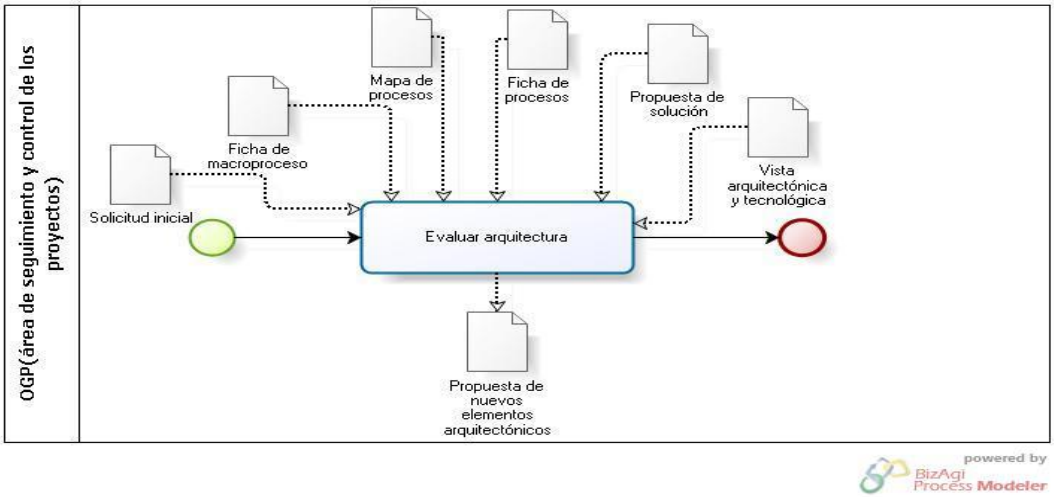
Anexo III: Actividades de la disciplina modelando negocio de empresa.



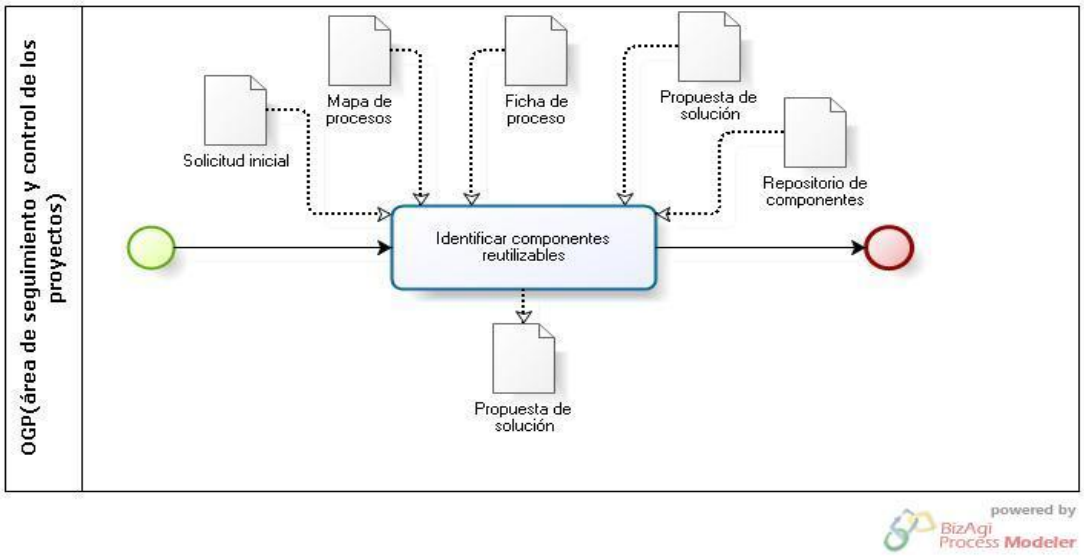
Anexo IV: Actividades de la disciplina gestión de cartera de proyectos.



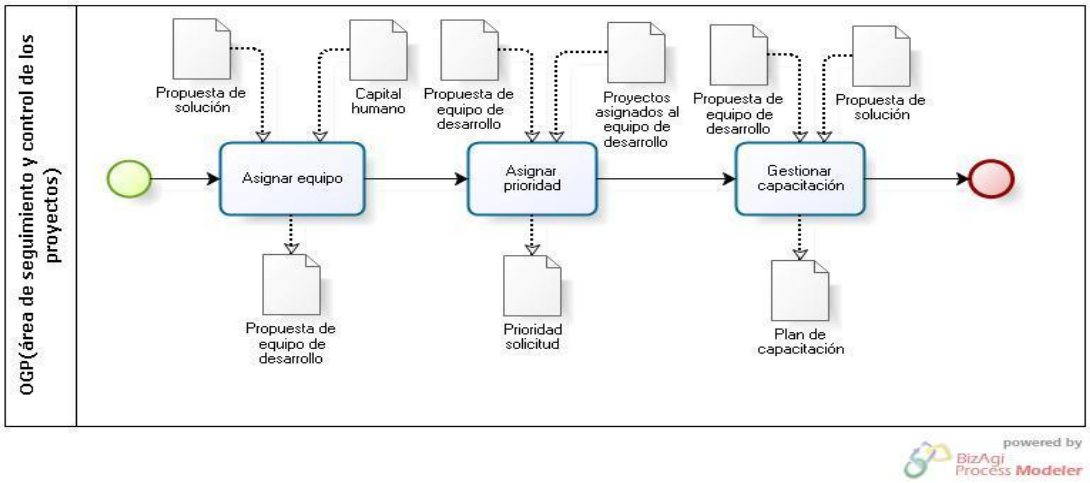
Anexo V: Actividades de la disciplina arquitectura de empresa



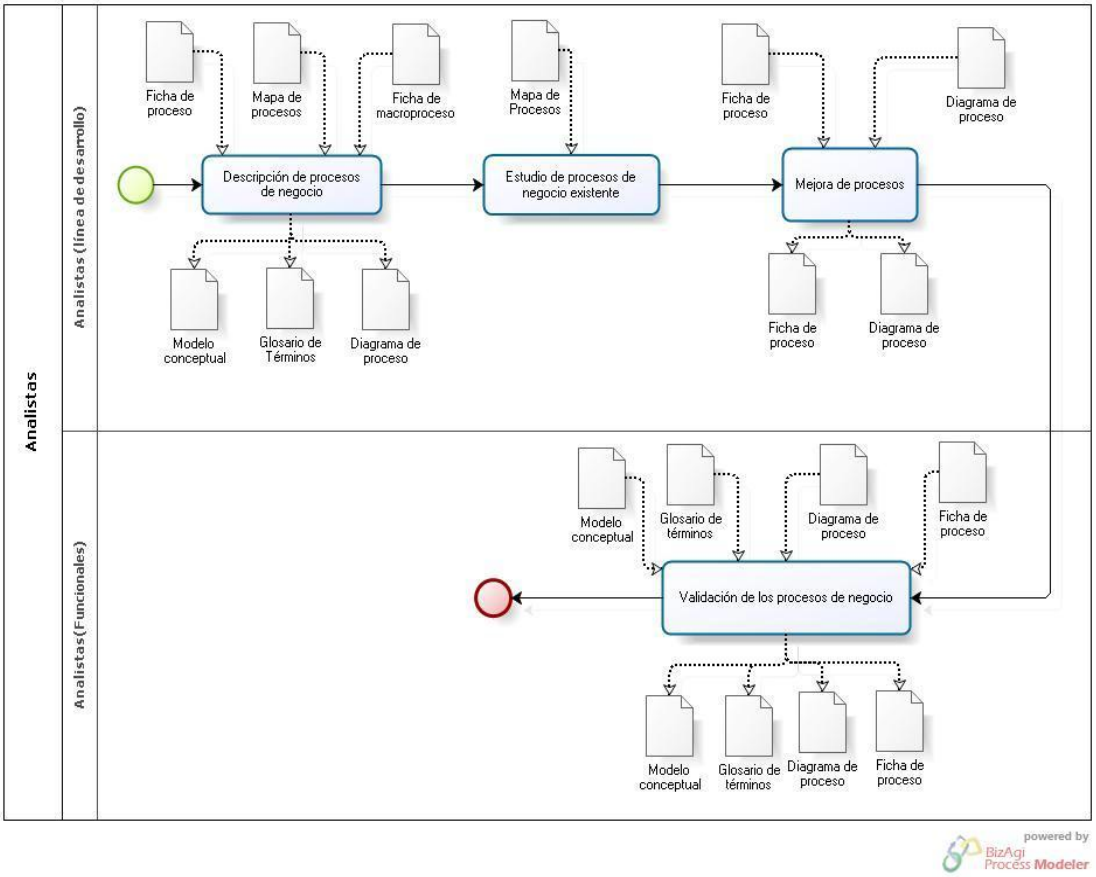
Anexo VI: Actividades de la disciplina reutilización estratégica.



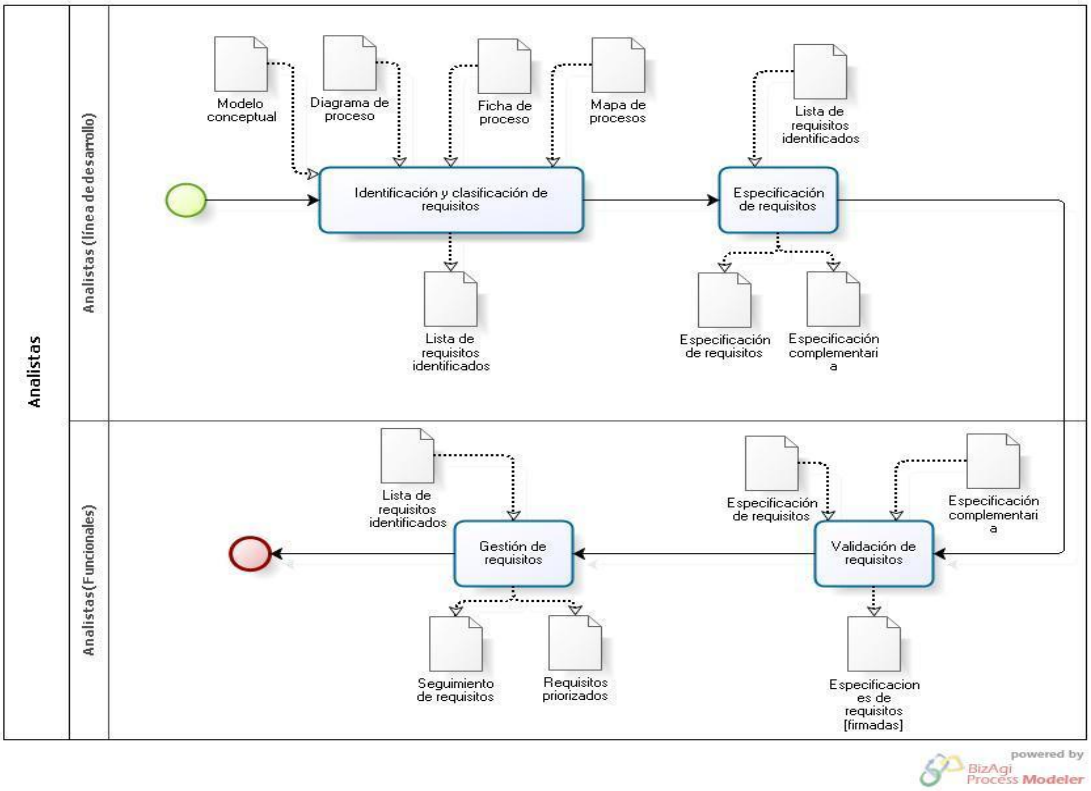
Anexo VII: Actividades de la disciplina gestión de personas



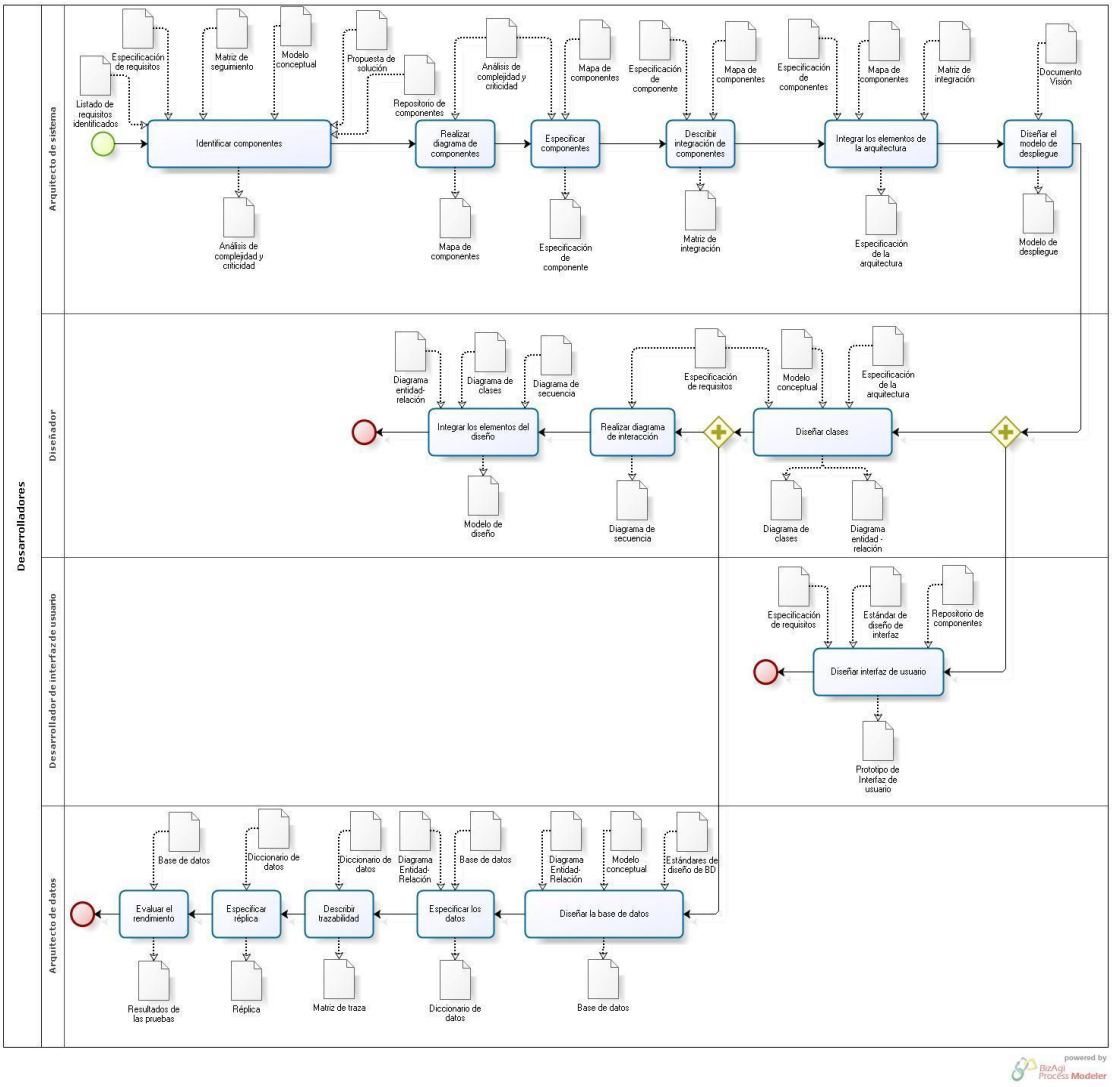
Anexo VIII: Actividades de la disciplina Modelación del negocio



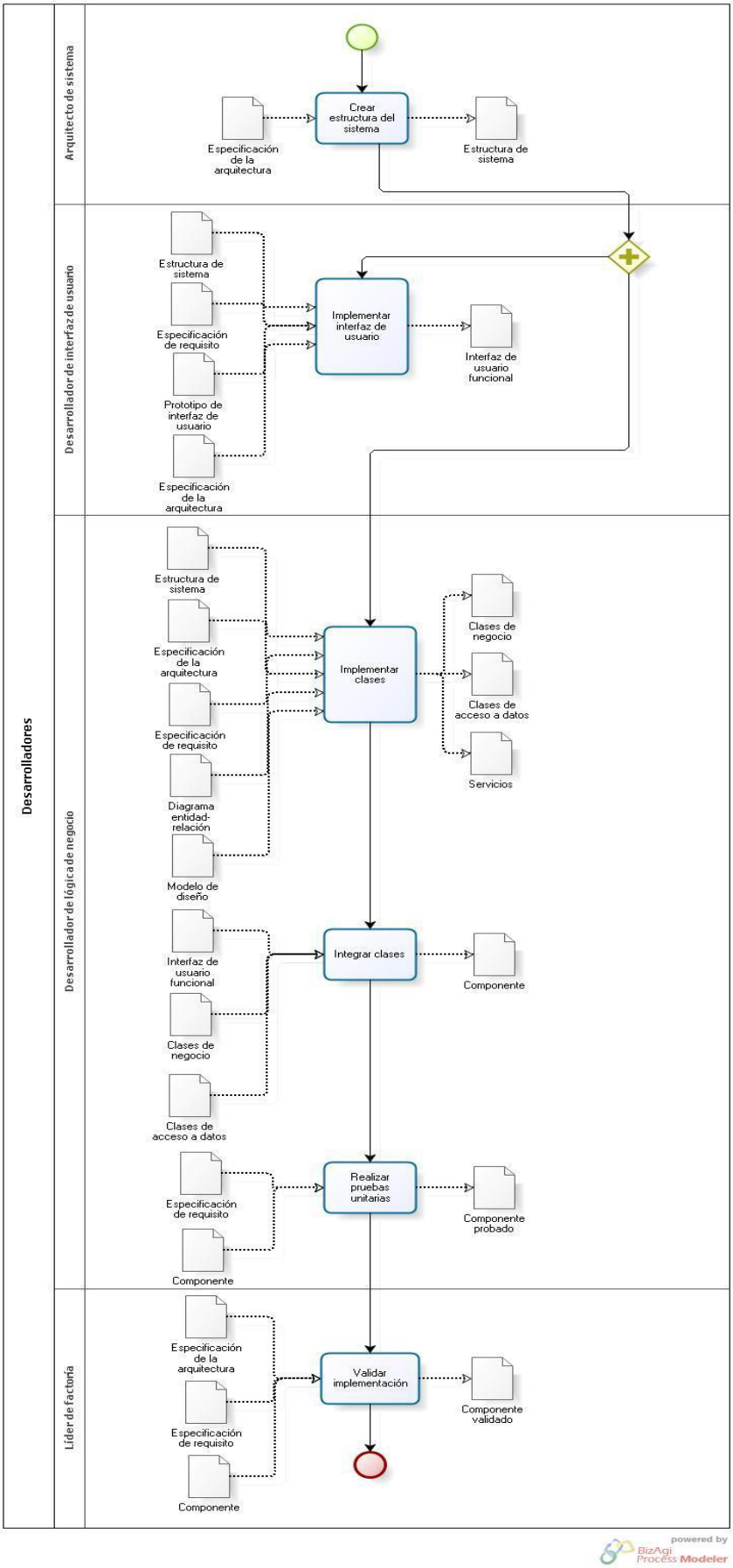
Anexo IX: Actividades de la disciplina Requerimientos



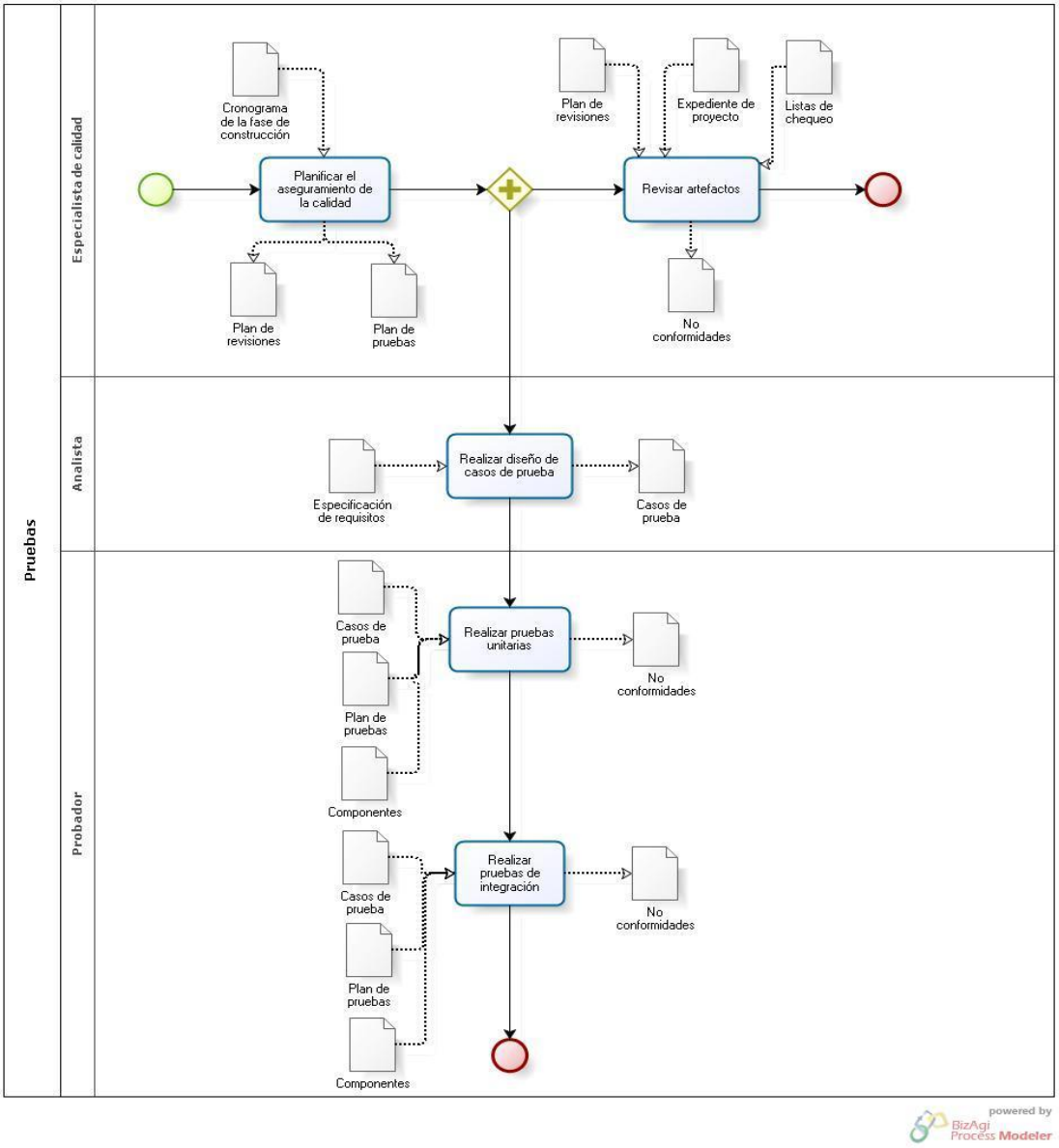
Anexo X: Actividades de la disciplina diseño



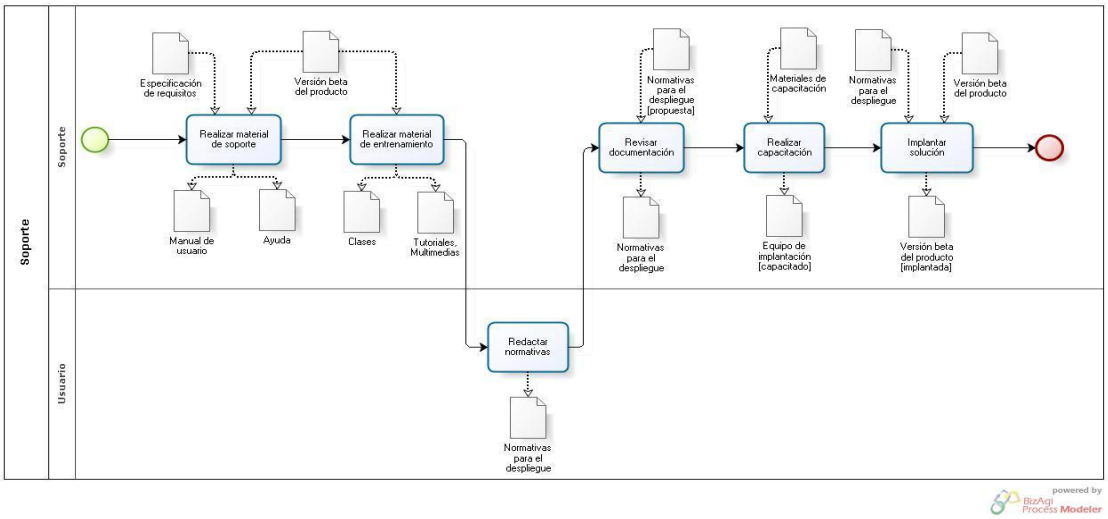
Anexo XI: Actividades de la disciplina implementación



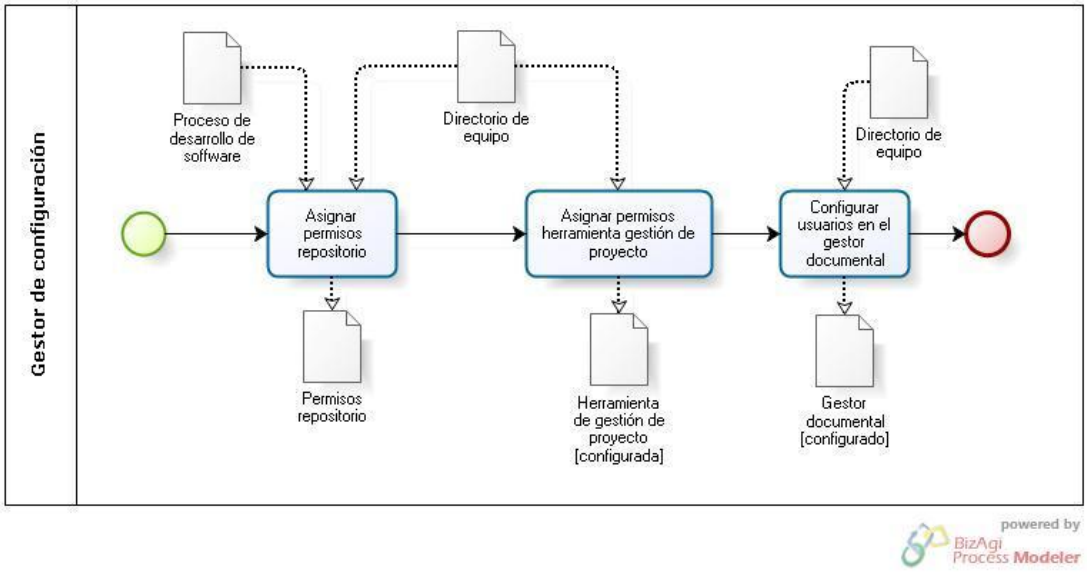
Anexo XII: Actividades de la disciplina pruebas



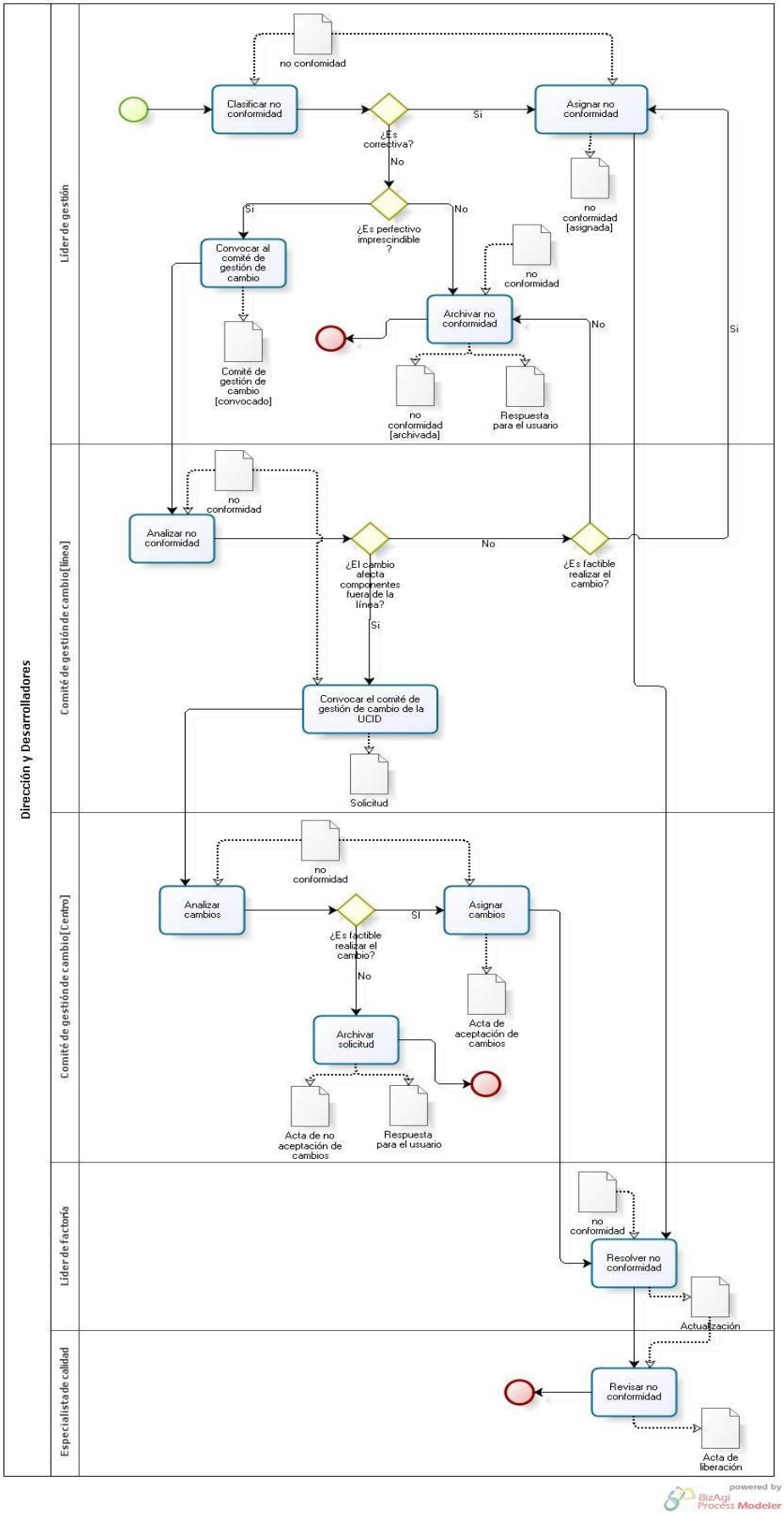
Anexo XIII: Actividades de la disciplina despliegue



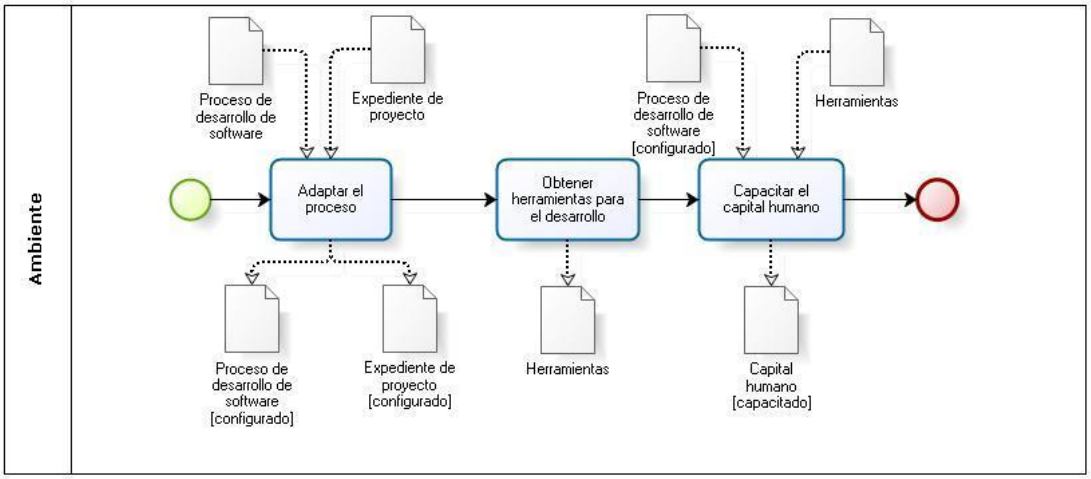
Anexo XIV: Actividades de la disciplina configuración



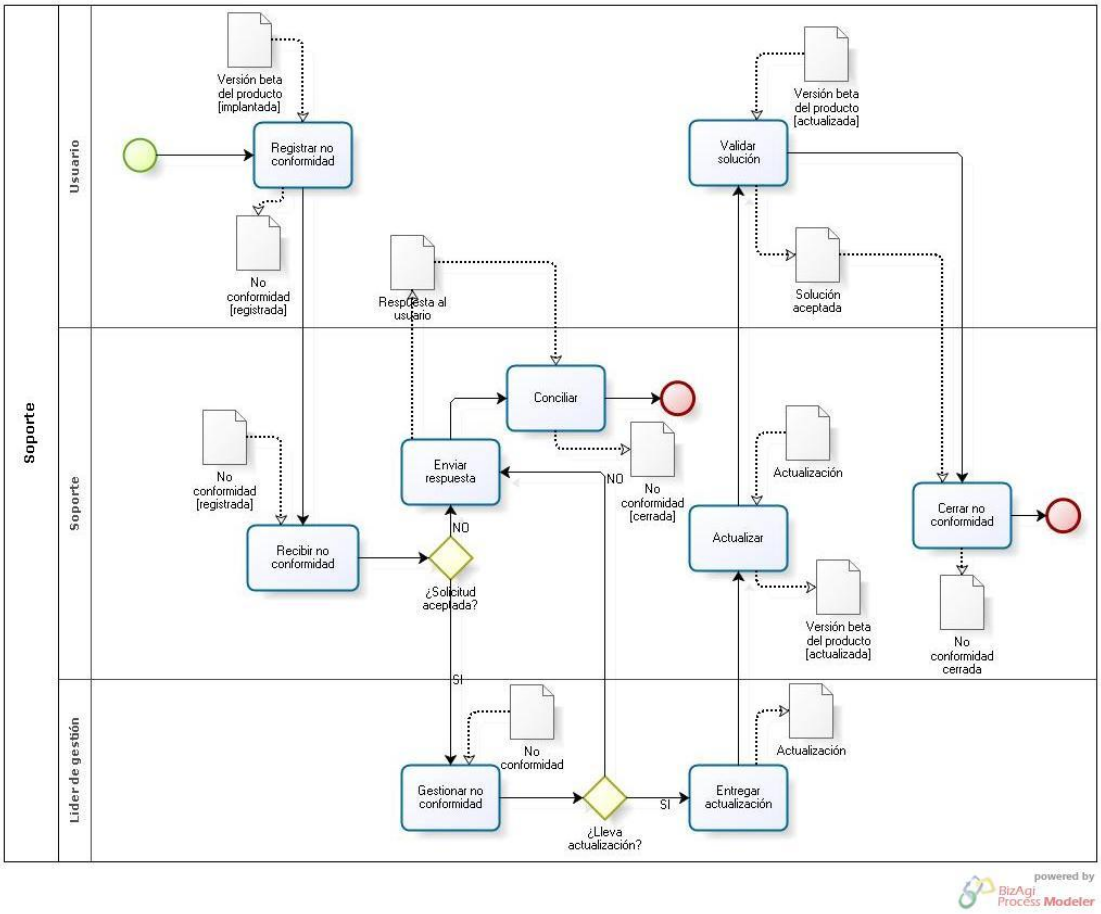
Anexo XV: Actividades de la disciplina gestión de cambio



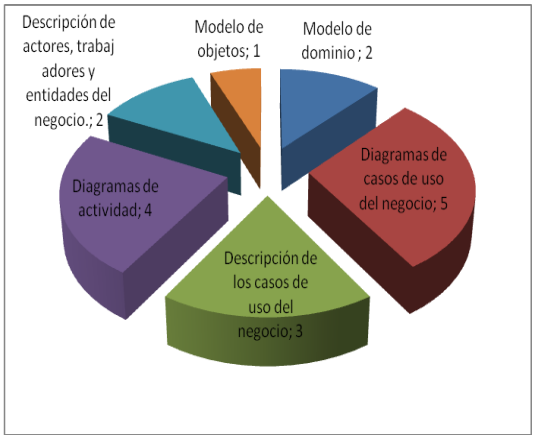
Anexo XVI: Actividades de la disciplina ambiente



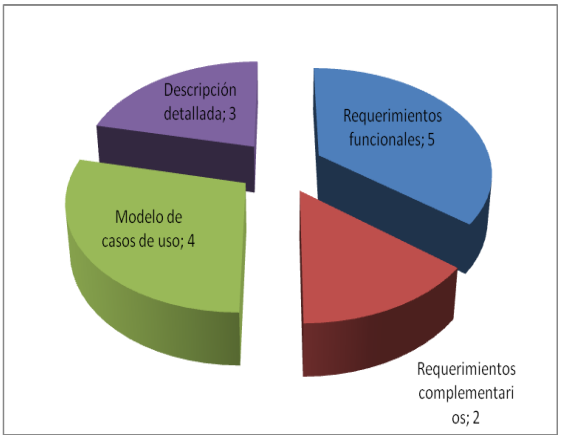
Anexo XVII: Actividades de la disciplina soporte



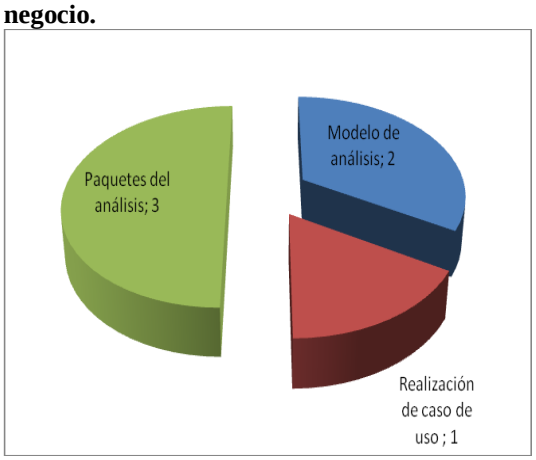
Anexo XVIII: Gráficos que muestran los artefactos desarrollados por disciplina.



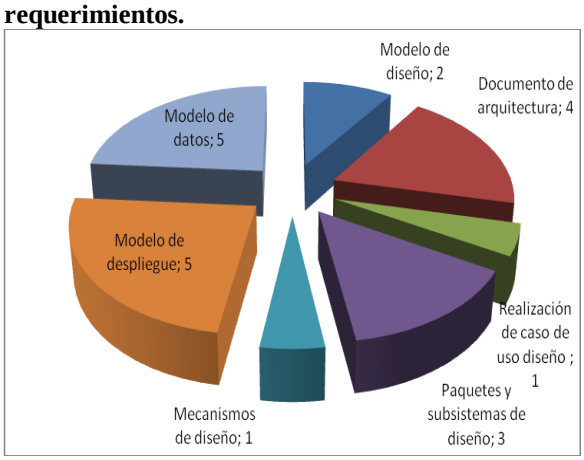
Cantidad de artefactos de la disciplina negocio.



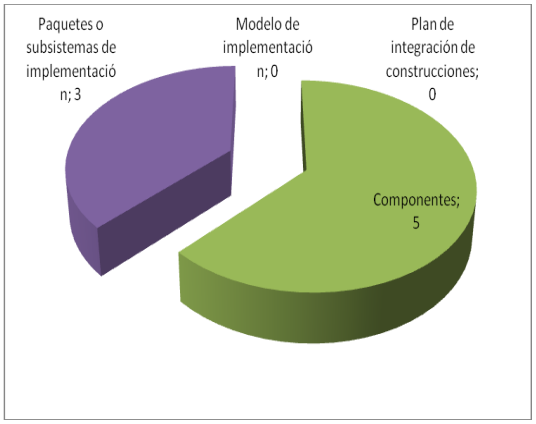
Cantidad de artefactos de la disciplina requerimientos.



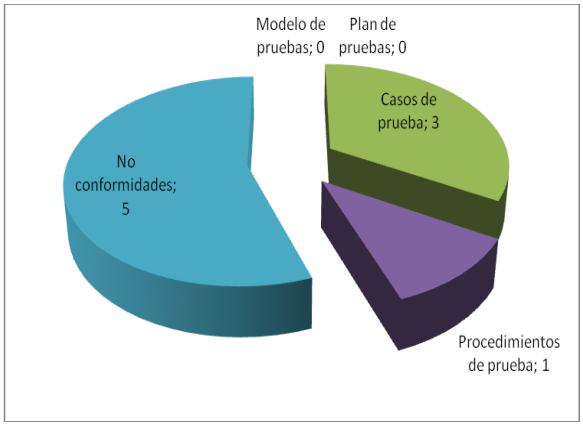
Cantidad de artefactos de la disciplina análisis.



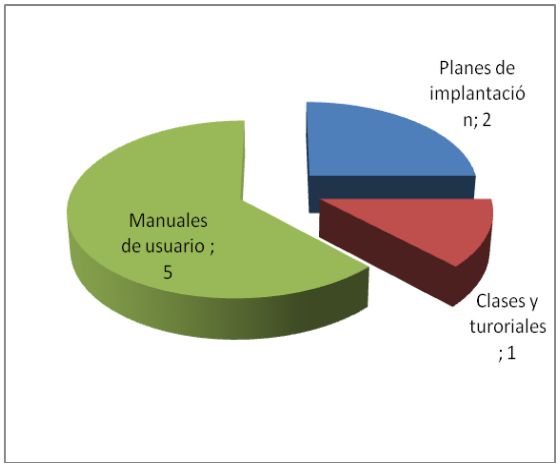
Cantidad de artefactos de la disciplina diseño.



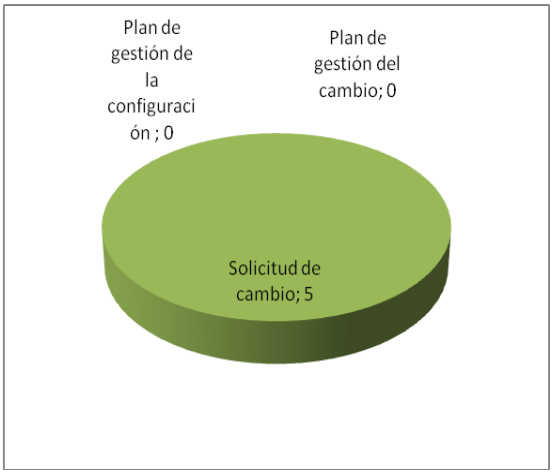
Cantidad de artefactos de la disciplina implementación.



Cantidad de artefactos de la disciplina pruebas.



Cantidad de artefactos de la disciplina despliegue.



Cantidad de artefactos de la disciplina gestión de la configuración y el cambio.

Anexo XIX: Encuesta de validación de la propuesta.

Compañero
Teniendo en cuenta su experiencia profesional y sus características personales usted ha sido seleccionado para colaborar con una investigación encaminada a proponer una metodología de desarrollo de software para su utilización en la UCID.
Nombre y Apellidos:
Cargo que ocupa:
Categoría científica:
Número de veces que Ud ha sido seleccionado como experto:

Años de experiencia en la actividad objeto de estudio:

Años de experiencia profesional u ocupacional:

Años de trabajo en la organización:
1- Marque con una cruz (X), en una escala creciente de 1 a 10, el valor que se corresponde con el grado de conocimiento e información que usted tiene sobre el tema objeto de investigación:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

2- Realice una autovaloración, según la tabla que a continuación se le ofrece, de sus niveles de argumentación o fundamentación sobre el tema que se investiga:

Fuentes de argumentación	**Grado de influencia de cada una de las fuentes en sus criterios**		
	A (Alto)	**M(Medio)**	**B(Bajo)**
Análisis teóricos realizados por usted (AT)			
Su experiencia adquirida (EA)			
Conocimiento de trabajos de autores nacionales (AN)			
Conocimiento de trabajos de autores extranjeros			
Su conocimiento del estado del problema en el extranjero			
Su intuición			

GRACIAS POR SU AYUDA.

Cuestionario

1. ¿Considera necesario que el equipo de desarrollo entienda completamente la metodología utilizada para desarrollar el software?
Si_____ No___ ¿Por qué?
_____
_____
_____
_____

2. ¿Considera usted que la metodología de desarrollo tal y como se lleva a cabo en la actualidad contribuye a obtener productos que cumplan con las expectativas de los clientes en el menor tiempo posible?

Completamente____ En gran medida____ Poco____ Muy poco____ Para nada____
¿Por qué?
_____
_____
_____
_____

3. Evalúe la propuesta en cuanto a su contribución a la obtención de productos que cumplan con las expectativas de los clientes en el menor tiempo posible

Muy adecuada ____ Bastante adecuada ____ Adecuada ____ Poco adecuada ____ No adecuada ____ ¿Por qué?
_____
_____
_____
_____

4. Las fases en que se divide la propuesta son:

Muy adecuadas ____ Bastante adecuadas ____ Adecuadas ____ Poco adecuadas ____
No adecuadas ____ Otras consideraciones al respecto:
_____
_____
_____
_____

5. Las disciplinas propuestas son:
Muy adecuadas ____ Bastante adecuadas ____ Adecuadas ____ Poco adecuadas ____
No adecuadas ____ Otras consideraciones al respecto:
_____
_____
_____
_____

6. Las actividades propuestas son:
Muy adecuadas ____ Bastante adecuadas ____ Adecuadas ____ Poco adecuadas ____
No adecuadas ____ Otras consideraciones al respecto:

_____
_____
_____
_____

7. Los artefactos propuestos son:
Muy adecuadas ____ Bastante adecuadas ____ Adecuadas ____ Poco adecuadas ____
No adecuadas ____ Otras consideraciones al respecto:
_____
_____
_____
_____

8. Las prácticas propuestas son:
Muy adecuadas ____ Bastante adecuadas ____ Adecuadas ____ Poco adecuadas ____
No adecuadas ____ Otras consideraciones al respecto:
_____
_____
_____
_____

9. Elabore un comentario general sobre la propuesta que está siendo evaluada que aporte elementos a la mejora de la misma.
_____
_____
____

Anexo XIX: Tabla con los resultados de la autovaloración de los expertos.

Especialistas	1	2	3	4	5	6	7	8	9	10	Kc
1							**x**				0,8
2								**x**			0,9
3									**x**		0,8
4								**x**			0,9
5								**x**			0,8
6									**x**		0,8
7								**x**			0,7
8							**x**				0,7
9								**x**			0,9

Anexo XX: Tabla con el grado de influencia de cada una de las fuentes del conocimiento. Frecuencia por expertos.

Especialistas	AT	EA	AN	AE	PC	I	Ka
1	0.2	0.5	0.05	0.05	0.05	0.05	0,9
2	0.3	0.2	0.05	0.05	0.05	0.05	0,8
3	0.1	0.4	0.05	0.05	0.05	0.05	0,9
4	0.2	0.5	0.05	0.05	0.05	0.05	0,6
5	0.3	0.4	0.05	0.05	0.05	0.05	0,9
6	0.2	0.4	0.05	0.05	0.05	0.05	0,9
7	0.3	0.2	0.05	0.05	0.05	0.05	0,8
8	0.2	0.4	0.05	0.05	0.05	0.05	0,7
9	0.2	0.4	0.05	0.05	0.05	0.05	0,9

Anexo XXI: Matriz de perfil competitivo (MPC)

Variables	Ponderación	Proceso Unificado		EUP		XP		SCRUM		Propuesta UCID	
		Puntos	Resultado Ponderado	Puntos	Resultado Ponderado	Puntos	Resultado Ponderado	Puntos	Resultado Ponderado	Puntos	Resultado Ponderado
La metodología debe ajustarse a los objetivos.	0,080	3	0,24	4	0,32	1	0,080	1	0,080	4	0,32
La metodología debe cubrir el ciclo entero del desarrollo de software.	0,086	4	0,344	4	0,344	3	0,258	2	0,172	4	0,344
La metodología debe integrar las distintas fases del ciclo de desarrollo.	0,086	4	0,344	4	0,344	2	0,172	2	0,172	4	0,344
La metodología debe incluir la realización de validaciones	0,080	1	0,08	1	0,08	4	0,32	3	0,24	3	0,24
La metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo.	0,075	4	0,3	4	0,3	3	0,225	3	0,225	4	0,3
La metodología debe ser la base de una comunicación efectiva.	0,078	3	0,234	3	0,234	3	0,234	3	0, 234	3	0,234
La metodología debe funcionar en un entorno dinámico orientado al usuario.	0,064	1	0,064	1	0,064	4	0,256	4	0,256	2	0,128
La metodología debe especificar claramente los responsables de los resultados.	0,078	3	0,234	3	0,234	1	0,078	3	0,234	4	0,312
La metodología debe poder emplearse en un entorno amplio de proyectos de software	0,083	1	0,083	2	0,166	1	0,083	1	0,083	3	0,249

La metodología se debe poder enseñar.	0,075	3	0,225	3	0,225	3	0,225	2	0,15	4	0,3
La metodología debe estar soportada por herramientas CASE.	0,070	3	0,21	3	0,21	2	0,14	2	0,14	3	0,21
La metodología debe soportar la eventual evolución del sistema.	0,075	3	0,225	3	0,225	3	0,225	3	0,225	4	0,3
La metodología debe contener actividades conducentes a mejorar el proceso de desarrollo de software.	0,072	2	0,144	3	0,216	2	0,144	2	0,144	3	0,216
			2,727		2,962		2,44		2,355		3,497

Anexo XXII: Artefacto especificación de requisito

Precondiciones	
Flujo de eventos	
Flujo básico	
Pos-condiciones	
Flujos alternativos	
Pos-condiciones	
Validaciones	
Relaciones	**Requisitos Incluidos**
	Extensiones
Conceptos	
Requisitos especiales	Son los requisitos no funcionales específicos para el requisito.
Asuntos pendientes	Posibles mejoras al requisito.