UNIVERSIDAD NACIONAL DE LA AMAZONIA PERUANA

FACULTAD DE INGENIERIA DE SISTEMAS E INFORMÁTICA

"Metodología en el Desarrollo de Software"

INFORME DEL TRABAJO PRÁCTICO DE SUFICIENCIA

PARA OPTAR EL TITULO PROFESIONAL DE:

INGENIERO DE SISTEMAS E INFORMATICA

PRESENTADO POR EL BACHILLER:

WILFREDO MARCIAL GARCÍA PÉREZ

IQUITOS –PERÚ

2015

INFORMETE TÉCNICO DEL EXAMEN DE SUFICIENCIA PREVIA ACTUALIZACION ACADEMICA APROBADO EN SUSTENTACION PUBLICA POR EL JURADO EXAMINADOR, DESIGNADO POR EL DECANO DE LA FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA DE LA UNIVERSIDAD NACIONAL DE LA AMAZONIA PERUANA.

JURADOS:

.....

Dr. Luis Benjamín Irigoin Sánchez
Presidente

.....

Ing. Carlos Alberto García Cortegano
Primer Miembro

.....

Ing. Juan Manuel Verme Insua
Segundo Miembro

DEDICATORIA

A Dios, ya que gracias a él he logrado concluir mi carrera, a mis padres, porque ellos siempre estuvieron a mi lado brindándome sus apoyo y consejos para hacer de mí una mejor persona, a mis hermanos por sus palabras y su compañía, a mis amigos, compañeros, y todas aquellas personas que de una u otra manera han contribuido para el logro de mis objetivos .

Bach. Wilfredo Marcial García Pérez

RESUMEN

Mediante este proyecto se pretende determinar una metodología Ágil de desarrollo de software, para la aplicación en proyectos de pequeña escala, donde se intenta minimizar la burocracia que subyace en la utilización de complejos procesos determinísticos que son utilizados en proyectos con gran cantidad de recursos.

Para el desarrollo de esta metodología se comenzará por conocer las diferentes metodologías y notaciones actualmente empleadas, partiendo por definir que son metodologías tradicionales y metodologías Agiles. Se detallará las principales metodologías como son: Proceso Unificado Racional (RUP), Métrica V3, Programación Extrema (XP) y Scrum; conociendo los roles, fases, procesos, artefactos y herramientas utilizadas en cada una de ellas.

Como resultado del proyecto se define una Metodología Propuesta, definiendo los roles, las fases e hitos, disciplinas, artefactos y actividades, que se utilizan para el desarrollo y aplicación de la metodología propuesta. Además de brinda como anexo los formatos de los artefactos de la metodología.

ABSTRACT

This project aims to establish an agile methodology of software development, for use in small-scale projects, which attempts to minimize bureaucracy that underlies the use of complex deterministic processes that are used in projects with many resources.

For the development of this methodology will begin to know the different methodologies and notations currently employed, starting with defining methodologies that are traditional and agile methodologies. The principal methodologies such as will be detailed: Rational Unified Process (RUP), Metric V3, Extreme Programming (XP) and Scrum; knowing their roles, phases, processes, artifacts and tools used in each.

As a result of a methodology proposed project it is defined by defining the roles, phases and milestones, disciplines, artifacts and activities, which are used for the development and implementation of the proposed methodology. In addition to providing annexed the Templates of artifacts methodology.

INDICE

Índice de Contenidos

Dedicatoria ***3***
Resumen ***4***
Abstract ***5***
Índice de Contenidos ***6***
Índice de Figuras ***7***
Justificación ***8***
Objetivos ***9***
Desarrollo del tema ***10***
1. Capítulo I: Teoría de la Temática ***10***
1.1. Definición de metodología de desarrollo de software *10*
1.2. Ventaja del uso de una metodología *10*
1.3. Metodologías de desarrollo de software tradicionales y Agiles *11*
1.3.1. Metodologías tradicionales *11*
1.3.2. Metodologías Agiles *12*
1.3.3. ¿Metodologías Agiles o Metodologías Tradicionales? *12*
1.4. Principales metodologías tradicionales y agiles *13*
1.4.1. Metodologías tradicionales *13*
1.4.1.1. RUP (Proceso Unificado Racional *13*
1.4.1.2. Métrica V3 ..... *21*
1.4.2. Metodologías Ágiles *31*
1.4.2.1. Programación Extrema (XP) *31*
1.4.2.2. SCRUM *38*
2. Capitulo II: Desarrollo de la Metodología Propuesta ***48***
2.1. Roles *48*
2.2. Fases e Hitos *49*
2.3. Disciplinas dentro de las fases *51*
2.4. Artefactos *59*
Conclusiones **62**
Referencia Bibliográfica **63**
Anexos **64**
Anexo A: Templates (Formatos) de artefactos 64
A1: Documento de visión 65
A2: Lista de Riesgos 68
A3: Código y Nombre del Caso de Uso 70
A4: Documento de Especificación de Requerimiento de Software (SRS) 72
A5: Descripción de Arquitectura 74
A6: Caso de prueba 76
A7: Nota de entrega 78

Índice de Figuras

FIGURA 1 *: Fases del Proceso Unificado Racional - RUP …….……..…….. 16*
FIGURA 2 *: Fases que componen la metodología propuesta …….. 49*
FIGURA 3 *: Disciplinas que componen la metodología propuesta ……..... 51*
FIGURA 4 *: Disciplina Factibilidad en detalle …….. 52*
FIGURA 5 *: Disciplina Requerimientos – Análisis en detalle …… 53*
FIGURA 6 *: Disciplina Diseño en detalle ……..55*

JUSTIFICION

Con este trabajo de investigación se pretende desarrollar una nueva metodología ágil en el desarrollo de software, incorporando una nuevos templates y conceptos en el desarrollo de software que permita desarrollar aplicaciones con menos costos y en menor tiempo.

Lo anterior en cuanto se encontró que no se han desarrollado profundamente aspectos y conceptos básicos relacionados al desarrollo de metodologías que nos permitan desarrollar y aprender en menos tiempo y menos costo; y porque cuando se aplique esta metodología va a ser una necesidad social para el desarrollo de software. Este proyecto tiene el objetivo de determinar una metodología Ágil de desarrollo de software.

En la actualidad se cuenta con diversos tipos de metodologías para el desarrollo de software, lo cual genera una gran problemática sobre cual debemos utilizar a la hora de diseñar un software, para esto es necesario conocer a fondo las diversas metodologías existente, saber cómo funciona cada una de ellas, y así poder elegir correctamente la más adecuada según la necesidad que se tenga. Sin embargo existente entornos de desarrollo de software en el que no se cuenta con los recursos y personal suficiente para la aplicación de las metodologías existentes. Con la metodología Ágil propuesta se intenta minimizar la burocracia que subyace en la utilización de complejos procesos determinísticos que son utilizados en proyectos con gran cantidad de recurso.

OBJETIVOS

Objetivo General

Determinar una metodología Ágil de desarrollo de software, para la aplicación en proyectos de pequeña escala, para minimizar la burocracia que subyace en la utilización de complejos procesos determinísticos que son utilizados en proyectos con gran cantidad de recursos

Objetivos Específicos

- ✓ Estudiar las metodologías agiles actuales para la recopilación de información, que sirva como base para el desarrollo de la metodología propuesta a desarrollar.
- ✓ Determinar los roles, fases e hitos, artefactos y procesos de la metodología a proponer.
- ✓ Brindar los templetes (formatos) de los artefactos de la metodología propuesta.

DESARROLLO DEL TEMA

1. TEORÍA DE LA TEMÁTICA

1.1. Definición de metodología de desarrollo de software.

Es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo de software. Las metodologías definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas.

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado.

1.2. Ventaja del uso de una metodología.

Son muchas las ventajas que puede aportar el uso de una metodología:

Desde el punto de vista de gestión:

- ✓ Facilitar la tarea de planificación
- ✓ Facilitar la tarea del control y seguimiento de un proyecto.
- ✓ Mejorar la relación coste/beneficio
- ✓ Optimizar el uso de recursos disponibles
- ✓ Facilitar la evaluación de resultados y cumplimiento de los objetivos.

Desde el punto de vista de los ingenieros del software:

- ✓ Ayudar a la comprensión del problema
- ✓ Optimizar el conjunto y cada una de las fases del proceso de desarrollo.
- ✓ Facilitar el mantenimiento del producto final.
- ✓ Permitir la reutilización de partes del producto.

Desde el punto de vista del cliente o usuario:

- ✓ Garantía de un determinado nivel de calidad en el producto final
- ✓ Confianza en los plazos de tiempos fijados en la definición del proyecto.

1.3. Metodologías de desarrollo de software Tradicionales y Ágiles.

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán.

Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software.

Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. 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.

Según la filosofía de desarrollo se pueden clasificar las metodologías en dos grupos. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

La evolución de la disciplina de ingeniería del software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis en la planificación y las metodologías ágiles haciendo énfasis en la adaptabilidad del proceso, delinean las principales propuestas presentes.

1.3.1. Metodologías tradicionales

Las metodologías tradicionales son denominadas, a veces, de forma despectiva, como metodologías pesadas. Centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido

todo esto, en la fase inicial del desarrollo del proyecto. Otra de las características importantes dentro de este enfoque, son los altos costes al implementar un cambio y la falta de flexibilidad en proyectos donde el entorno es volátil. Las metodologías tradicionales (formales) se focalizan en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones, etc.)

1.3.2. Metodologías ágiles.

Este enfoque nace como respuesta a los problemas que puedan ocasionar las metodologías tradicionales y se basa en dos aspectos fundamentales, mejorar las decisiones y la planificación adaptativa. Basan su fundamento en la adaptabilidad de los procesos de desarrollo. Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.

1.3.3. ¿Metodologías ágiles o metodologías tradicionales?

En las metodologías tradicionales el principal problema es que nunca se logra planificar bien el esfuerzo requerido para seguir la metodología. Pero entonces, si logramos definir métricas que apoyen la estimación de las actividades de desarrollo, muchas prácticas de metodologías tradicionales podrían ser apropiadas. El no poder predecir siempre los resultados de cada proceso no significa que estemos frente a una disciplina de azar.

Lo que significa es que estamos frente a la necesidad de adaptación de los procesos de desarrollo que son llevados por parte de los equipos que desarrollan software.

Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. Estas metodologías pueden involucrar prácticas tanto de metodologías ágiles como de metodologías tradicionales. De esta manera podríamos tener una metodología por cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cuál usar.

Es importante tener en cuenta que el uso de un método ágil no vale para cualquier proyecto. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables.

En la tabla que se muestra a continuación aparece una comparativa entre estos dos grupos de metodologías.

Metodologías Agiles	Metodologías tradicionales
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
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

Cuadro 1. *Cuadro comparativo entre metodologías tradicionales y desarrollo ágil*

1.4. Principales metodologías tradicionales y ágiles.

1.4.1. Metodologías tradicionales

1.4.1.1. RUP (Proceso Unificado Racional).

RUP (Proceso Unificado Racional) es un proceso para el desarrollo de un proyecto de un software que define claramente quien, cómo, cuándo y qué debe hacerse en el proyecto. Como 3 características esenciales está dirigido por los Casos de Uso: que orientan el proyecto a la importancia para el usuario y lo que este quiere, está centrado en la arquitectura: que Relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y es iterativo e incremental: donde divide el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada.

Como filosofía RUP maneja 6 principios clave:

- **Adaptación del proceso**
  El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

- **Balancear prioridades**
  Los requerimientos de los diversos clientes a los cuales se les realizara el proyecto (en este caso solo un cliente), pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos (en el caso pertinente que satisfaga los deseos del cliente).

- **Colaboración entre equipos**
  El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc. Para nuestro caso y contradiciendo la colaboración entre equipos que es la que se aplica para un proyecto mayúsculo, este proyecto si se hace con una única persona, ya que el proyecto a realizar no es tan gigante como lo que podría representar un proyecto de una empresa grande.

- **Demostrar valor iterativamente**
  Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

- **Elevar el nivel de abstracción**
  Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o esquemas (frameworks) por nombrar algunos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

- **Enfocarse en la calidad**
  El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

1.4.1.1.1. Roles en RUP

Son las personas que participan en los procesos y tiene competencias específicas:

Analista de Sistema
Responsable del conjunto de requisitos, modelados en los casos de uso.

Especificador de caso de uso

Responsable de la descripción detallada de un caso de uso, establece una comunicación estrecha y eficaz con el usuario.

Diseñador de interfaces

- Diseña las interfaces de usuario – Aspecto visual
- Desarrolla prototipos de interfaces de usuario para algunos casos de uso.

Arquitecto

- Describe la arquitectura y prioridades del modelo de casos de uso
- Garantiza la integridad del modelo de análisis
- Garantizarla integridad de los modelos de diseño y despliegue
- Garantiza la integridad del modelo de implantación.

Ingeniero de casos de uso

- Mantener la integridad de las realizaciones de casos de uso
- Detallar las realizaciones de casos de uso
- Verificar la correspondencia entre análisis y diseño

Ingeniero de Componentes

- Definir y mantener las responsabilidades, atributos, relaciones y requisitos especiales de una o varias clases del análisis
- Definir y mantener las operaciones, métodos, atributos, relaciones y requisitos de implantación de una o más clases del diseño Implantación
- Definir y mantener el código fuente de uno o varios componentes
- Desarrollar componentes de prueba que automatizan algunos de los procedimientos de prueba

Integrador de Sistemas

- Planificar la secuencia de construcciones necesarias en cada iteración
- Integrar cada construcción a partir de sus partes implementadas

Diseñador de prueba

- Garantizar la integridad del modelo de pruebas
- Planear las pruebas
- Seleccionar y describir casos de prueba y los procedimientos de prueba

Ingeniero de pruebas de integración

- Ejecutar las pruebas de integración Verificar el correcto funcionamiento de componentes
- Documentar los defectos

Ingeniero de pruebas de sistema

- Ejecutar las pruebas del sistema para cada iteración completa
- Verificar los resultados en conjunto con los usuarios finales
- Documentar los defectos
- Facilidad para tener familiaridad con el comportamiento observable del sistema

1.4.1.1.2. El ciclo de vida de RUP – FASES

RUP divide el proceso en 4 fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades

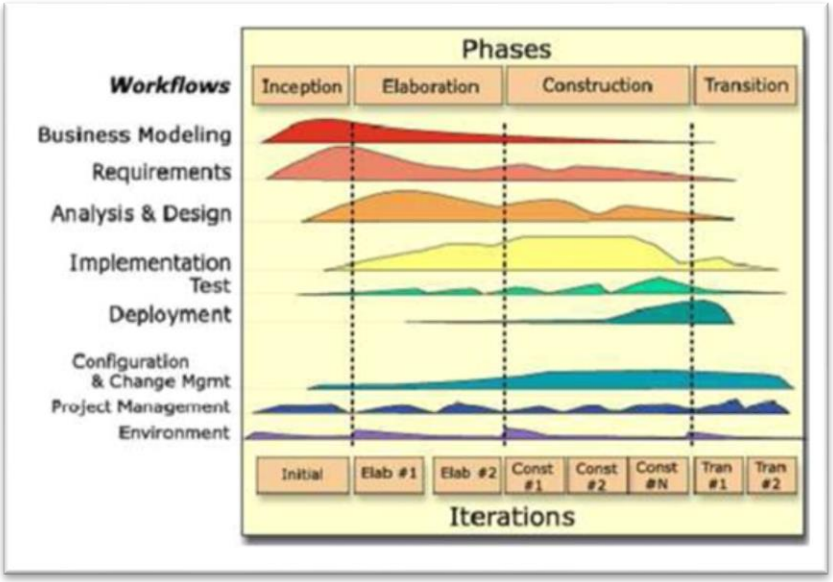

Figura 1. Fases del RUP

En las iteraciones de cada fase se hacen diferentes esfuerzos en diferentes actividades.

- **Inicio:**
  Se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos. Se define el alcance del proyecto.

- **Elaboración:**
  Se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos.

- **Construcción:**
  Se concentra en la elaboración de un producto totalmente operativo y eficiente junto con el manual de usuario.

- **Transición:**
  Se Instala el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados.

Descripción De Las Actividades

Dependiendo de las iteraciones del proceso el equipo de desarrollo puede realizar 7 tipos de actividades en este:

- **Fase De Inicio**

  Durante la fase de inicio las iteraciones hacen poner mayor énfasis en actividades modelado del negocio y de requisitos.

  Modelado del negocio

  - ✓ En esta fase el equipo se familiarizará más al funcionamiento de la empresa, sobre conocer sus procesos.
  - ✓ Entender la estructura y la dinámica de la organización para la cual el sistema va ser Desarrollado.
  - ✓ Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
  - ✓ Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.

  Requisitos

  - ✓ En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que especifiquemos.
  - ✓ Establecer y mantener un acuerdo entre clientes sobre lo que el sistema podría hacer.

- ✓ Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.
- ✓ Definir el ámbito del sistema.
- ✓ Proveer una base para estimar costos y tiempo de desarrollo del sistema.
- ✓ Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

❖ **Fase De Elaboración**

En la fase de elaboración, las iteraciones se orientan al desarrollo de la base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la base de la arquitectura.

Análisis y Diseño

- ✓ En esta actividad se especifican los requerimientos y se describen sobre cómo se van a implementar en los sistemas.
- ✓ Transformar los requisitos al diseño del sistema.
- ✓ Desarrollar una arquitectura para el sistema.
- ✓ Adaptar el diseño para que sea consistente con el entorno de implementación

❖ **Fase De Construcción**

Implementación

- ✓ Se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. El resultado final es un sistema ejecutable.
- ✓ Planificar qué subsistemas deben ser implementados y en qué orden deben ser integrados, formando el Plan de Integración.
- ✓ Cada implementador decide en qué orden implementa los elementos del subsistema.
- ✓ Si encuentra errores de diseño, los notifica.
- ✓ Se integra el sistema siguiendo el plan.

Pruebas

- ✓ Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida.
- ✓ Encontrar y documentar defectos en la calidad del software.
- ✓ Generalmente asesora sobre la calidad del software percibida.
- ✓ Provee la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- ✓ Verificar las funciones del producto de software según lo diseñado.
- ✓ Verificar que los requisitos tengan su apropiada implementación.

Despliegue

- ✓ Esta actividad tiene como objetivo producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:
- ✓ Probar el producto en su entorno de ejecución final.
- ✓ Empaquetar el software para su distribución.
- ✓ Distribuir el software.
- ✓ Instalar el software.
- ✓ Proveer asistencia y ayuda a los usuarios.
- ✓ Formar a los usuarios y al cuerpo de ventas.
- ✓ Migrar el software existente o convertir bases de datos.

❖ **Fase de Transición**

- ✓ Se libera el producto y se entrega al usuario para un uso real
- ✓ Se incluyen tareas de marketing, empaquetado atractivo, instalación, configuración, entrenamiento, soporte, mantenimiento, etc.
- ✓ Los manuales de usuario se completan y refinan con la información anterior
- ✓ Estas tareas se realizan también en iteraciones

1.4.1.1.3. Gestión Durante todo el proyecto

Gestión del proyecto

- Se vigila el cumplimiento de los objetivos, gestión de riesgos y restricciones para
- Desarrollar un producto que sea acorde a los requisitos de los clientes y los usuarios.
- Proveer un marco de trabajo para la gestión de proyectos de software intensivos.
- Proveer guías prácticas realizar planeación, contratar personal, ejecutar y monitorear el proyecto.
- Proveer un marco de trabajo para gestionar riesgos.

Configuración y control de cambios

El control de cambios permite mantener la integridad de todos los artefactos que se crean en el proceso, así como de mantener información del proceso evolutivo que han seguido.

Entorno

La finalidad de esta actividad es dar soporte al proyecto con las adecuadas herramientas, procesos y métodos. Brinda una especificación de las herramientas que se van a necesitar en cada momento, así como definir la instancia concreta del proceso que se va a seguir.

En concreto las responsabilidades de este flujo de trabajo incluyen:

- Selección y adquisición de herramientas
- Establecer y configurar las herramientas para que se ajusten a la organización.
- Configuración del proceso.
- Mejora del proceso.
- Servicios técnicos.

1.4.1.1.4. Artefactos de RUP

RUP en cada una de sus fases (pertenecientes a la estructura estática) realiza una serie de artefactos que sirven para comprender mejor tanto el análisis como el diseño del sistema (entre otros). Estos artefactos (entre otros) son los siguientes:

❖ **Inicio**:

- Documento Visión
- Especificación de Requerimientos

- **Elaboración**:

  - Diagramas de caso de uso

- **Construcción**:

  - Documento Arquitectura que trabaja con las siguientes vistas:

    Vista Lógica:

    - Diagrama de clases
    - Modelo E-R (Si el sistema así lo requiere)

    Vista De Implementación:

    - Diagrama de Secuencia
    - Diagrama de estados
    - Diagrama de Colaboración

    Vista Conceptual

    - Modelo de dominio

    Vista Física

    - Mapa de comportamiento a nivel de hardware

1.4.1.2. METRICA V3

Es una Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de información. Promovida por el Ministerio de Administraciones Públicas del gobierno español (recientemente suprimido, pasando sus competencias a la vicepresidencia primera del gobierno o Ministerio de la Presidencia) para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas.

Esta metodología propia está basada en el Modelo de Procesos del Ciclo de vida de desarrollo ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) así como en la norma ISO/IEC 15504 SPICE (Software Process Improvement And Assurance Standards Capability Determination).

En una única estructura la metodología MÉTRICA Versión 3 cubre distintos tipos de desarrollo: estructurado y orientado a objetos, facilitando a través de interfaces la realización de los procesos de apoyo u organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad y Seguridad.

1.4.1.2.1. Perfiles en Métrica V3

Para poder exponer una estructura de los participantes que se han identificado a lo largo de los procesos de la Metodología MÉTRICA Versión 3, se ha optado por establecer una serie de perfiles en los que se encuadran la totalidad de los participantes:

Perfil Directivo

El perfil requerido para este grupo de participantes incluye a personas con un nivel alto en la dirección de la organización, conocimiento de los objetivos estratégicos y de negocio que se persiguen y autoridad para validar y aprobar cada uno de los procesos realizados durante el desarrollo del Sistema de Información.

Se agrupan los siguientes participantes:

- ✓ Comité de Dirección
- ✓ Comité de Seguimiento
- ✓ Directores de usuarios
- ✓ Usuarios expertos

Perfil jefe de proyecto

Todos ellos ejercen labores de coordinación y dirección de equipos humanos especializados en la realización de actividades propias de un proceso o interfaz de MÉTRICA Versión 3. La figura principal es el Jefe de Proyecto, el cual recibe el apoyo de los distintos responsables durante la realización de procesos o determinadas actividades a lo largo del proyecto.

Se agrupan los siguientes participantes:

- ✓ Jefe de Proyecto
- ✓ Responsable de Implantación
- ✓ Responsable de Mantenimiento
- ✓ Responsable de Operación
- ✓ Responsable de Sistemas
- ✓ Responsable de Seguridad
- ✓ Responsable de Calidad

Perfil Consultor

La principal función de los Consultores es asesorar en las cuestiones sobre las que tienen un conocimiento especializado. Se diferencia así entre Consultor, que asesora en los aspectos relativos al negocio y Consultor Informático, con un nivel de especialización mayor en los aspectos relacionados con la informática, su aplicación e integración en la organización.

Se agrupan los siguientes participantes:

- ✓ Consultor
- ✓ Consultor Informático
- ✓ Consultor de las Tecnologías de la Información
- ✓ Consultor de Sistemas de Información
- ✓ Especialista en Comunicaciones
- ✓ Técnico de Sistemas
- ✓ Técnicos de Comunicaciones

Perfil analista

La responsabilidad de los Analistas es elaborar un catálogo detallado de requisitos que permita describir con precisión el sistema de información, para lo cual mantendrán entrevistas.

Se agrupan los siguientes participantes:

- ✓ Analista
- ✓ Administrador de Bases de Datos
- ✓ Equipo de Arquitectura
- ✓ Equipo de Formación
- ✓ Equipo de Implantación
- ✓ Equipo de Operación
- ✓ Equipo de Seguridad
- ✓ Equipo de Soporte Técnico
- ✓ Equipo de Proyecto
- ✓ Grupo de Aseguramiento de la Calidad

Perfil Programador

Dado que la participación y funciones de los programadores son concretas y limitadas a los procesos de Construcción y Mantenimiento de Sistemas de Información, el perfil de Programador hace referencia únicamente al participante Programador descrito en MÉTRICA Versión 3.

La función del programador, miembro del equipo de proyecto, es construir el código que dará lugar al producto resultante en base al diseño técnico realizado por el analista o analista programador, generando también el código asociado a los procedimientos de migración y carga inicial de datos. Igualmente se encarga de la realización de las pruebas unitarias y participa en las pruebas de conjunto de la aplicación.

1.4.1.2.2. Procesos Principales de métrica V3

La metodología descompone cada uno de los procesos en actividades, y éstas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes.

Los procesos de la estructura principal de MÉTRICA Versión 3 son los siguientes:

- Planificación de sistemas de información.
- Desarrollo de sistemas de información.
- Mantenimiento de sistemas de información.

1.4.1.2.3. Planificación de sistemas de información:

El objetivo de un Plan de Sistemas de Información es proporcionar un marco estratégico de referencia para los Sistemas de Información de un determinado ámbito de la Organización.

El resultado del Plan de Sistemas debe, por tanto, orientar las actuaciones en materia de desarrollo de Sistemas de Información con el objetivo básico de apoyar la estrategia corporativa, elaborando una arquitectura de información y un plan de proyectos informáticos para dar apoyo a los objetivos estratégicos.

Como productos finales de este proceso se obtienen los siguientes, que podrán constituir la entrada para el siguiente proceso de Estudio de Viabilidad del Sistema:

- Catálogo de requisitos de PSI que surge del estudio de la situación actual en el caso de que sea significativo dicho estudio, del diagnóstico que se haya llevado a cabo y de las necesidades de información de los procesos de la organización afectados por el plan de sistemas.

- Arquitectura de información que se compone a su vez de los siguientes productos:
  - Modelo de información.
  - Modelo de sistemas de información.
  - Arquitectura tecnológica.
  - Plan de proyectos.
  - Plan de mantenimiento del PSI.

1.4.1.2.4. Desarrollo de sistemas de información.

El proceso de Desarrollo de MÉTRICA Versión 3 contiene todas las actividades y tareas que se deben llevar a cabo para desarrollar un sistema, cubriendo desde el análisis de requisitos hasta la instalación del software. Además de las tareas relativas al análisis, incluye dos partes en el diseño de sistemas: arquitectónico y detallado. También cubre las pruebas unitarias y de integración del sistema, aunque siguiendo la norma ISO 12.207 no propone ninguna técnica específica y destaca la importancia de la evolución de los requisitos.

El desarrollo en MÉTRICA Versión 3 lo constituyen los procesos:

- Estudio de viabilidad del sistema (EVS).
- Análisis del sistema de información (ASI).
- Diseño del sistema de información (DSI).
- Construcción del sistema de información (CSI).
- Implantación y aceptación del sistema (IAS).

Estudio de viabilidad del sistema (EVS).

El propósito de este proceso es analizar un conjunto concreto de necesidades, con la idea de proponer una solución a corto plazo. Los criterios con los que se hace esta propuesta no serán estratégicos sino tácticos y relacionados con aspectos económicos, técnicos, legales y operativos.

El resultado final de este proceso son los productos relacionados con la solución que se propone para cubrir la necesidad concreta que se planteó en el proceso, y que depende de si la solución conlleva desarrollo a medida o no:

Contexto del sistema (con la definición de las interfaces en función de la solución).

- ✓ Impacto en la organización.
- ✓ Coste/beneficio de la solución.
- ✓ Valoración de riesgos de la solución.
- ✓ Enfoque del plan de trabajo de la solución.

Planificación de la solución.
Solución propuesta:

- Descripción de la solución.
- Modelo de descomposición en subsistemas.
- Matriz de procesos/localización geográfica.
- Matriz datos/localización geográfica. Entorno tecnológico y comunicaciones.
- Estrategia de implantación global del sistema.
- Descripción de los procesos manuales.
- Si la alternativa incluye desarrollo:
- Modelo abstracto de datos/Modelo de procesos.
- Modelo de negocio/Modelo de dominio.
- Si la alternativa incluye un producto software estándar de mercado:
- Descripción del producto.
- Evolución del producto.
- Costes ocasionados por el producto.
- Estándares del producto.
- Descripción de adaptación si es necesaria.

Análisis del sistema de información (ASI).

El propósito de este proceso es conseguir la especificación detallada del sistema de información, a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

Los productos resultantes del Análisis del Sistema de Información, dependen del tipo de desarrollo de que se trate y se detallan a continuación especificando los que son distintos, según los dos tipos de desarrollo a los que da respuesta MÉTRICA Versión 3:

- Descripción general del entorno tecnológico.
- Glosario de términos.
- Catálogo de normas.
- Catálogo de requisitos.
- Especificación de interfaz de usuario.

En este proceso es muy importante la participación de los usuarios, a través de técnicas interactivas, como diseño de diálogos y prototipos, que permiten al usuario familiarizarse con el nuevo sistema y colaborar en la construcción y perfeccionamiento del mismo.

Diseño del sistema de información (DSI).

El propósito del Diseño del Sistema de Información (DSI) es obtener la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información. A partir de dicha información, se generan todas las especificaciones de construcción relativas al propio sistema, así como la especificación técnica del plan de pruebas, la definición de los requisitos de implantación y el diseño de los procedimientos de migración y carga inicial, éstos últimos cuando proceda.
Este proceso consta de un primer bloque de actividades, que se realizan en paralelo, y cuyo objetivo es obtener el diseño de detalle del sistema de información que comprende a partición física del sistema de información, independiente de un entorno tecnológico concreto, la organización en subsistemas de diseño, la especificación del entorno tecnológico sobre el que se despliegan dichos subsistemas y la definición de los requisitos de operación, administración del sistema, seguridad y control de acceso. En el caso de diseño orientado a objetos, conviene señalar que se ha contemplado que el diseño de la persistencia se lleva a cabo sobre bases de datos relacionales.

De este primer bloque de actividades se obtienen los siguientes productos:

- Catálogo de requisitos (se completa).
- Catálogo de excepciones.
- Catálogo de normas para el diseño y construcción.
- Diseño de la arquitectura del sistema.
- Entorno tecnológico del sistema.
- Procedimientos de operación y administración del sistema.
- Procedimientos de seguridad y control de acceso.
- Diseño detallado de los subsistemas de soporte.
- Modelo físico de datos optimizado.
- Asignación de esquemas físicos de datos a nodos.

Un segundo bloque de actividades complementa el diseño del sistema de información, en el que se generan todas las especificaciones necesarias para la construcción del sistema de información:

- Las especificaciones de construcción de los componentes del sistema (módulos o clases, según el caso) y de las estructuras de datos.
- Los procedimientos de migración y sus componentes asociados.
- La definición y revisión del plan de pruebas, y el diseño de las verificaciones de los niveles de prueba establecidos.
- El catálogo de excepciones que permite establecer un conjunto de verificaciones relacionadas con el propio diseño o con la arquitectura del sistema.
- La especificación de los requisitos de implantación.

Construcción del sistema de información (CSI).

La construcción del Sistema de Información (CSI) tiene como objetivo final la construcción y prueba de los distintos componentes del sistema de información, a partir del conjunto de especificaciones lógicas y físicas del mismo, obtenido en el Proceso de Diseño del sistema de Información (DSI). Se desarrollan los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación, estos últimos cuando proceda.

Para conseguir dicho objetivo, se recoge la información relativa al producto del diseño, especificaciones de construcción del sistema de información, se prepara el entorno de construcción, se genera el código de cada uno de los componentes del sistema de información y se van realizando, a medida que se vaya finalizando la construcción, las pruebas unitarias de cada uno de ellos y las de integración entre subsistemas.

Como resultado de dicho proceso se obtiene:

- Resultado de las pruebas unitarias.
- Evaluación del resultado de las pruebas de integración.
- Evaluación del resultado de las pruebas del sistema.
- Producto software:
  - Código fuente de los componentes.
  - Procedimientos de operación y administración del sistema.
  - Procedimientos de seguridad y control de acceso.
  - Manuales de usuario.
  - Especificación de la formación a usuarios finales.
  - Código fuente de los componentes de migración y carga inicial de datos.
  - Procedimientos de migración y carga inicial de datos.
  - Evaluación del resultado de las pruebas de migración y carga inicial de datos.

Implantación y aceptación del sistema (IAS).

Este proceso tiene como objetivo principal, la entrega y aceptación del sistema en su totalidad, que puede comprender varios sistemas de información desarrollados de manera independiente, según se haya establecido en el proceso de Estudio de Viabilidad del Sistema (EVS), y un segundo objetivo que es llevar a cabo las actividades oportunas para el paso a producción del sistema.

Para el inicio de este proceso se toman como punto de partida los componentes del sistema probados de forma unitaria e integrada en el proceso Construcción del Sistema de Información (CSI), así como la documentación asociada. El Sistema se someterá a las Pruebas de Implantación con la participación del usuario de operación cuya responsabilidad, entre otros aspectos, es comprobar el comportamiento del sistema bajo las condiciones más extremas. También se someterá a las Pruebas de Aceptación cuya ejecución es responsabilidad del usuario final.

En este proceso se elabora el plan de mantenimiento del sistema de forma que el responsable del mantenimiento conozca el sistema antes de que éste pase a producción. También se establece el acuerdo de nivel de servicio requerido una vez que se inicie la producción. El acuerdo de nivel de servicio hace referencia a servicios de gestión de operaciones, de soporte a usuarios y al nivel con el que se prestarán dichos servicios.

Como resultado de este proceso se obtienen los siguientes productos:

- Plan de implantación del sistema en su totalidad.
- Equipo de implantación que realizará la implantación.
- Plan de formación del equipo de implantación (esquema, materiales, recursos necesarios, planificación y especificación de la formación de usuarios finales).
- Evaluación de las pruebas de implantación del sistema por parte del usuario de operación.
- Evaluación de las pruebas de aceptación del sistema por parte del usuario final.
- Plan de mantenimiento previo al paso a producción.
- Acuerdo de nivel de servicio del sistema.
- Sistema en producción.

1.4.1.2.5. Mantenimiento de sistemas de información

El objetivo de este proceso es la obtención de una nueva versión de un sistema de información desarrollado con MÉTRICA, a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema o por la necesidad de una mejora del mismo.

Como consecuencia de esto, sólo se considerarán en MÉTRICA Versión 3 los tipos de Mantenimiento Correctivo y Evolutivo. Se excluyen los tipos de Mantenimiento Adaptativo y perfectivo, que abarcan actividades tales como la migración y la retirada de software que precisarían el desarrollo de un tipo de metodología específica para resolver su cometido. Ante una petición de cambio de un sistema de información ya en producción, se realiza un registro de las peticiones, se diagnostica el tipo de mantenimiento y se decide si se le da respuesta o no, en función del plan de mantenimiento asociado al sistema afectado por la petición, y se establece con qué prioridad

La definición de la solución al problema o necesidad planteada por el usuario que realiza el responsable de mantenimiento, incluye un estudio del impacto, la valoración del esfuerzo y coste, las actividades y tareas del proceso de desarrollo a realizar y el plan de pruebas de regresión.

Los productos que se obtienen en este proceso son los siguientes:

- Catálogo de peticiones de cambio.
- Resultado del estudio de la petición.
- Propuesta de solución.
- Análisis de impacto de los cambios.
- Plan de acción para la modificación.
- Plan de pruebas de regresión.
- Evaluación del cambio.
- Evaluación del resultado de las pruebas de regresión.

1.4.2. Metodologías ágiles

1.4.2.1. PROGRAMACIÓN EXTREMA (XP)

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. Kent Beck, el padre de XP, describe la filosofía de XP en sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

1.4.2.1.1. Roles XP

Programador:

El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

Cliente:

El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

Encargado de pruebas (Tester):

El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Encargado de seguimiento (Tracker)

El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.

Entrenador (Coach):

Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el **proceso correctamente.**

Consultor:

Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.

Gestor (Big boss):

Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

1.4.2.1.2. PROCESOS DE XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

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.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

1.4.2.1.3. FASES

Planificación
En esta etapa se plantea los alcances del sistema a desarrollar.

Se presentará el siguiente documento:

- ✓ **Historias de usuario**: en este documento irán los requerimientos del sistema, según esta metodología los clientes son los que se encargan de llenar dicho documento, sin embargo en este proyecto el cliente va ser reemplazado por el encargado del proyecto y por el asesor.
- ✓ **Plan de entregas**
- ✓ **Plan de iteraciones**

Diseño
En esta etapa se plantea el diseño que tendrá el sistema a desarrollar.

Se presentarán los siguientes documentos:

- ✓ **Glosario de términos**: en este documento se especifican los métodos y clases que tendrá el sistema.
- ✓ **Prototipo de interfaz de usuario**: en este documento se presenta el diseño del sistema.
- ✓ **Prototipo de base de datos:** en este documento se detalla la base de datos que tendrá el sistema.
- ✓ **Tarjetas CRC**: ésta son una técnica para la representación de sistemas orientados a objetos. Representan objetos, la clase a la que pertenecen, sus objetivos o responsabilidades y las clases colaboradoras.

Desarrollo
Se presentará el siguiente documento:

- ✓ **Estándar de programación:** en este documento se detalla los estándares de programación que se utilizarán en el sistema.

Pruebas

En esta etapa se desarrollan las pruebas al sistema, al ser un sistema inteligente las pruebas se van a enfocar prácticamente en la eficiencia del algoritmo y la calibración de las constantes de relajación.

Se presentará el siguiente documento:

- ✓ **Pruebas de optimización:** en este documento se detallará las pruebas que se realizaron para calibrar la constante de relajación y las pruebas de rendimiento y tiempo de ejecución del algoritmo.

1.4.2.1.4. PRÁCTICAS XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. XP apuesta por un crecimiento lento del costo del cambio y con un comportamiento asintótico. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las prácticas que describiremos a continuación.

El juego de la planificación

Es un espacio frecuente de comunicación entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. Esta práctica se puede ilustrar como un juego, donde existen dos tipos de jugadores: Cliente y Programador. El cliente establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes. Este juego se realiza durante la planificación de la entrega, en la planificación de cada iteración y cuando sea necesario reconducir el proyecto.

Entregas pequeñas

La idea es producir rápidamente versiones del sistema que sean operativas, aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema pero sí que constituyan un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

Metáfora

En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema. Martin Fowler en explica que la práctica de la metáfora consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Este conjunto de nombres ayuda a la nomenclatura de clases y métodos del sistema.

Diseño simple

Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente. Kent Beck dice que en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos.

Pruebas

La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial.

Refactorización (Refactoring)

La refactorización es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. La refactorización mejora la estructura interna del código sin alterar su comportamiento externo. Robert Martin señala que el diseño del sistema de software es una cosa viviente. No se puede imponer todo en un inicio, pero en el transcurso del tiempo este diseño evoluciona conforme cambia la funcionalidad del sistema. Para mantener un diseño apropiado, es necesario realizar actividades de cuidado continuo durante el ciclo de

vida del proyecto. De hecho, este cuidado continuo sobre el diseño es incluso más importante que el diseño inicial. Un concepto pobre al inicio puede ser corregido con esta actividad continua, pero sin ella, un buen diseño inicial se degradará.

Programación en parejas

Toda la producción de código debe realizarse con trabajo en parejas de programadores. Según Cockburn y Williams en un estudio realizado para identificar los costos y beneficios de la programación en parejas, las principales ventajas de introducir este estilo de programación son: muchos errores son detectados conforme son introducidos en el código (inspecciones de código continuas), por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor (continua discusión de ideas de los programadores), los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y finalmente, los programadores disfrutan más su trabajo. Dichos beneficios se consiguen después de varios meses de practicar la programación en parejas. En los estudios realizados por Cockburn y Williams este lapso de tiempo varía de 3 a 4 meses.

Propiedad colectiva del código

Cualquier programador puede cambiar cualquier parte del código en cualquier momento. Esta práctica motiva a todos a contribuir con nuevas ideas en todos los segmentos del sistema, evitando a la vez que algún programador sea imprescindible para realizar cambios en alguna porción de código.

Integración continúa

Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Todas las pruebas son ejecutadas y tienen que ser aprobadas para que el nuevo código sea incorporado definitivamente. La integración continua a menudo reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado o compartido. Martin Fowler en afirma que el desarrollo de un proceso disciplinado y automatizado es esencial para un proyecto controlado, el equipo de desarrollo está más preparado para modificar el

código cuando sea necesario, debido a la confianza en la identificación y corrección de los errores de integración.

40 horas por semana

Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen al final ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.

Cliente in-situ

El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita, ya que esta última toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada. En Jeffries indica que se debe pagar un precio por perder la oportunidad de un cliente con alta disponibilidad. Algunas recomendaciones propuestas para dicha situación son las siguientes: intentar conseguir un representante que pueda estar siempre disponible y que actúe como interlocutor del cliente, contar con el cliente al menos en las reuniones de planificación, establecer visitas frecuentes de los programadores al cliente para validar el sistema, anticiparse a los problemas asociados estableciendo llamadas telefónicas frecuentes y conferencias, reforzando el compromiso de trabajo en equipo.

Estándares de programación

XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios.

Comentarios respecto de las prácticas

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. Esto se ilustra en la Figura

1, donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí.

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

1.4.2.2. SCRUM

Scrum es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

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

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos

balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas. De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla y el retorno de inversión mediante la replanificación de objetivos que realiza al inicio de cada iteración.

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

1. **Selección de requisitos (4 horas máximo).** El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
2. **Planificación de la iteración (4 horas máximo**). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.

Ejecución de la iteración

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

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

Durante la iteración el Facilitador se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

Elimina los obstáculos que el equipo no puede resolver por sí mismo.

Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

1. **Demostración (4 horas máximo).** El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.

2. **Retrospectiva (4 horas máximo)**. El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

1.4.2.2.1. Roles

Product Owner

Las responsabilidades del Product Owner (que puede ser interno o externo a la organización) son:

- ✓ Ser el representante de todas las personas interesadas en los resultados del proyecto (internas o externas a la organización, promotores del proyecto y usuarios finales [idealmente también debería ser un usuario clave] o consumidores finales del producto) y actuar como interlocutor único ante el equipo, con autoridad para tomar decisiones.
- ✓ Definir los objetivos del producto o proyecto.
- ✓ Dirigir los resultados del proyecto y maximizar su ROI (Return Of Investment).
  - Es el propietario de la planificación del proyecto: crea y mantiene la lista priorizada con los requisitos necesarios para cubrir los objetivos del producto o proyecto, conoce el valor que aportará cada requisito y calcula el ROI a partir del coste de cada requisito que le proporciona el equipo.
  - Reparte los objetivos/requisitos en iteraciones y establece un calendario de entregas.
  - Antes de iniciar cada iteración replanifica el proyecto en función de los requisitos que aportan más valor en ese momento, de los requisitos completados en la iteración anterior y del contexto del proyecto en ese momento (demandas del mercado, movimientos de la competencia, etc.).
- ✓ Colaborar con el equipo para planificar, revisar y dar detalle a los objetivos de cada iteración:
  - Participar en la reunión de planificación de iteración, proponiendo los requisitos más prioritarios a desarrollar, respondiendo a las dudas del

equipo y detallando los requisitos que el equipo se comprometer a hacer.

- Estar disponible durante el curso de la iteración para responder a las preguntas que puedan aparecer.
- No cambiar los requisitos que se están desarrollando en una iteración, una vez está iniciada.
- Participar en la reunión de demostración de la iteración, revisando los requisitos completados.

Scrum Master (facilitador)

Lidera al equipo llevando a cabo las siguientes responsabilidades:

- ✓ Velar por que todos los participantes del proyecto sigan las reglas y proceso de Scrum, encajándolas en la cultura de la organización, y guiar la colaboración intraequipo y con el cliente de manera que las sinergias sean máximas. Esto implica:
  - Asegurar que la lista de requisitos priorizada esté preparada antes de la siguiente iteración.
  - Facilitar las reuniones de Scrum (planificación de la iteración, reuniones diarias de sincronización del equipo, demostración, retrospectiva), de manera que sean productivas y consigan sus objetivos.
  - Enseñar al equipo a autogestionarse. No da respuestas, si no que guía al equipo con preguntas para que descubra por sí mismo una solución.
- ✓ Quitar los impedimentos que el equipo tiene en su camino para conseguir el objetivo de cada iteración (proporcionar un resultado útil al cliente de la manera más efectiva) y poder finalizar el proyecto con éxito. Estos obstáculos se identifican de manera sistemática en las reuniones diarias de sincronización del equipo y en las reuniones de retrospectiva.
- ✓ Proteger y aislar al equipo de interrupciones externas durante la ejecución de la iteración(introducción de nuevos requisitos, "secuestro" no previsto de un miembro del equipo, etc.). De esta manera, el equipo puede mantener su productividad y el compromiso que adquirió sobre los requisitos que completaría en la iteración [notar, sin embargo, que el equipo debe reservar tiempo para colaborar con al cliente en la preparación de la lista de requisitos para la próxima iteración].

Team (equipo)

Grupo de personas que de manera conjunta desarrollan el producto del proyecto. Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan (así como de su calidad) en cada iteración y en el proyecto.

El tamaño del equipo está entre 5 y 9 personas. Por debajo de 5 personas cualquier imprevisto o interrupción sobre un miembro del equipo compromete seriamente el compromiso que han adquirido y, por tanto, el resultado que se va a entregar al cliente al finalizar la iteración. Por encima de 9 personas, la comunicación y colaboración entre todos los miembros se hace más difícil y se forma subgrupos.

De cualquier manera, se puede hacer Scrum con 3 personas y se ha utilizado en proyectos con 250 personas en varios equipos. Cuando es necesario que más de un equipo trabaje de manera ágil en un mismo proyecto, existen diferentes técnicas que permiten esta colaboración, desde el Scrum de Scrums hasta equipos de integración que dedican parte de su tiempo a trabajar con los equipos de desarrollo, siempre completando incrementos de producto de manera regular.

Es un equipo autoorganizado, que comparte información y cuyos miembros confían entre ellos. Realiza de manera conjunta las siguientes actividades:

- ✓ Seleccionar los requisitos que se compromete a completar en una iteración, de forma que estén preparados para ser entregados al cliente.
- ✓ Estimar la complejidad de cada requisito en la lista de requisitos priorizada del producto o proyecto.
- ✓ En la reunión de planificación de la iteración decide cómo va a realizar su trabajo:
    - Seleccionar los requisitos que pueden completar en cada iteración, realizando al cliente las preguntas necesarias.
    - Identificar todas las tareas necesarias para completar cada requisito.
    - Estimar el esfuerzo necesario para realizar cada tarea.
    - Cada miembro del equipo se autoasigna a las tareas.
- ✓ Durante la iteración, trabajar de manera conjunta para conseguir los objetivos de la iteración. Cada especialista lidera el trabajo en su área y el resto colaboran si es necesario para poder completar un requisito.
- ✓ Al finalizar la iteración:

- Demostrar al cliente los requisitos completados en cada iteración.
- Hacer una retrospectiva la final de cada iteración para mejorar de forma continua su manera de trabajar.

Todos los miembros del equipo trabajan en la misma localización física, para poder maximizar la comunicación entre ellos mediante conversaciones cara a cara, diagramas en pizarras blancas, etc. De esta manera se minimizan otros canales de comunicación menos eficientes, que hacen que las tareas se transformen en un "pasa pelota" o que hacen perder el tiempo en el establecimiento de la comunicación (como cuando se llama repetidas veces por teléfono cuando la persona no está en su puesto).

El equipo debe ser estable durante el proyecto, sus miembros deben cambiar lo mínimo posible, para poder aprovechar el esfuerzo que les ha costado construir sus relaciones interpersonales, engranarse y establecer su organización del trabajo.

1.4.2.2.2. Actividades

Planificación de la iteración (Sprint Planning)

La planificación de las tareas a realizar en la iteración se divide en dos partes:

- ❖ Primera parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas:
  - ✓ El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto, pone nombre a la meta de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella.
  - ✓ El equipo examina la lista, pregunta al cliente las dudas que le surgen, añade más condiciones de satisfacción y selecciona los objetivos/requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
- ❖ Segunda parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas.

  El equipo planifica la iteración, elabora la táctica que le permitirá conseguir el mejor resultado posible con el mínimo esfuerzo. Esta actividad la realiza el equipo dado que ha adquirido un compromiso, es el responsable de organizar su trabajo y es quien mejor conoce cómo realizarlo.

- Define las tareas necesarias para poder completar cada objetivo/requisito, creando la lista de tareas de la iteración (Sprint Backlog) basándose en la definición de completado.
- Realiza una estimación conjunta del esfuerzo necesario para realizar cada tarea.
- Cada miembro del equipo se autoasigna a las tareas que puede realizar.

Estos son tiempos máximos en el caso de iteraciones mensuales. En iteraciones de tamaño menor el tiempo es proporcionalmente inferior, y se puede ir reduciendo conforme el equipo va ganando experiencia en este tipo de reuniones, aunque también dependerá de la complejidad a desarrollar en la iteración.

Ejecución de la iteración (Sprint)

En Scrum un proyecto se ejecuta en bloques temporales cortas y fijas (iteraciones de un mes natural y hasta de dos semanas). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea susceptible de ser entregado con el mínimo esfuerzo cuando el cliente (Product Owner) lo solicite.

Cada día el equipo realiza una reunión de sincronización, donde cada miembro inspecciona el trabajo de los otros para poder hacer las adaptaciones necesarias, comunica cuales son los impedimentos con que se encuentra, actualiza el estado de la lista de tareas de la iteración (Sprint Backlog) y los gráficos de trabajo pendiente (Burndown charts).

El Facilitador (Scrum Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

- ✓ Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- ✓ Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Reunión diaria de sincronización del equipo (Scrum Daily Meeting)

El objetivo de esta reunión es facilitar la transferencia de información y la colaboración entre los miembros del equipo para aumentar su productividad, al poner de manifiesto puntos en que se pueden ayudar unos a otros.

Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la

iteración, obstáculos que pueden impedir este objetivo) para al finalizar la reunión poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso conjunto que el equipo adquirió para la iteración (en la reunión de planificación de la iteración).

Cada miembro del equipo debe responder las siguientes preguntas en un timebox de cómo máximo 15 minutos:

- ✓ ¿Qué he hecho desde la última reunión de sincronización? ¿Pude hacer todo lo que tenía planeado? ¿Cuál fue el problema?
- ✓ ¿Qué voy a hacer a partir de este momento?
- ✓ ¿Qué impedimentos tengo o voy a tener para cumplir mis compromisos en esta iteración y en el proyecto?

Como apoyo a la reunión, el equipo cuenta con la lista de tareas de la iteración, donde se actualiza el estado y el esfuerzo pendiente para cada tarea, así como con el gráfico de horas pendientes en la iteración.

Demostración de los requisitos completados (Sprint Review)

Reunión informal donde el equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo, haciendo un recorrido por ellos lo más real y cercano posible al objetivo que se pretende cubrir.

En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.

Se realiza en un timebox de cómo máximo 4 horas.

Retrospectiva (Sprint Retrospective)

Con el objetivo de mejorar de manera continua su productividad y la calidad del producto que está desarrollando, el equipo analiza cómo ha sido su manera de trabajar durante la iteración, por qué está consiguiendo o no los objetivos a que se comprometió al inicio de la iteración y por qué el incremento de producto que acaba de demostrar al cliente era lo que él esperaba o no:

- ✓ Qué cosas han funcionado bien.
- ✓ Cuales hay que mejorar.
- ✓ Qué cosas quiere probar hacer en la siguiente iteración.
- ✓ Qué ha aprendido.

Cuáles son los problemas que podrían impedirle progresar adecuadamente. El Facilitador se encargará de ir eliminando los obstáculos identificados que el propio equipo no pueda resolver por sí mismo.

Notar que esta reunión se realiza después de la reunión de demostración al cliente de los objetivos conseguidos en la iteración, para poder incorporar su feedback y cumplimiento de expectativas como parte de los temas a tratar en la reunión de retrospectiva

Se realiza en un timebox de cómo máximo 3 horas (si la iteración es mensual).

Replanificación del proyecto

En las reuniones de planificación de entregas y/o durante el transcurso de una iteración, el cliente va trabajando en la lista de objetivos/requisitos priorizada del producto o proyecto, añadiendo requisitos, modificándolos, eliminándolos, repriorizándolos, cambiando el contenido de iteraciones y definiendo un calendario de entregas que se ajuste mejor a sus nuevas necesidades.

Los cambios en la lista de requisitos pueden ser debidos a:

- ✓ Modificaciones que el cliente solicita tras la demostración que el equipo realiza al final de cada iteración sobre los resultados obtenidos, ahora que el cliente entiende mejor el producto o proyecto.
- ✓ Cambios en el contexto del proyecto (sacar al mercado un producto antes que su competidor, hacer frente a urgencias o nuevas peticiones de clientes, etc).
- ✓ Nuevos requisitos o tareas como resultado de nuevos riesgos en el proyecto.
- ✓ Etc.

Para realizar esta tarea, el cliente colabora con el equipo:

- ✓ Para cada nuevo objetivo/requisito conjuntamente se hace una identificación inicial de sus condiciones de satisfacción (que se detallarán en la reunión de planificación de la iteración).
- ✓ El cliente obtiene del equipo la re-estimación de costes de desarrollo para completar los objetivos/requisitos siguientes, según la definición de completado. El equipo ajusta el factor de complejidad, el coste para completar los requisitos y su velocidad de desarrollo en

función de la experiencia adquirida hasta ese momento en el proyecto.

- ✓ El cliente re-prioriza la lista de objetivos/requisitos en función de estas nuevas estimaciones.

Hay que notar que el equipo sigue trabajando con los objetivos/requisitos de la iteración en curso, (que de hecho eran los más prioritarios al iniciar la iteración). No es posible cambiar los requisitos que se desarrollan durante la iteración. En la reunión de planificación de la iteración el cliente presentará la nueva lista de requisitos para que sea desarrollada

2. DESARROLLO DE LA METODOLOGIA PROPUESTA

En este capítulo se verá en detalle cómo está estructurada la metodología ágil propuesta. En particular la misma fue concebida con la idea de ser aplicada en pequeños grupos de desarrollo, sin necesidad de invertir en costosas herramientas o de capacitar en forma extensiva a las personas involucradas.

Por tratarse de una metodología ágil se intenta minimizar la burocracia que subyace en la utilización de complejos procesos determinísticos que son utilizados en proyectos con gran cantidad de recursos. Es por esto que se han anexado en este capítulo aquellos templates (formatos) que pueden ser utilizados para que el equipo de desarrollo estandarice los entregables que generará. Esto permitirá que la transición hacia la metodología sea lo menos dolorosa posible, logrando disminuir la curva de aprendizaje inherente a un cambio de proceso.

2.1. Roles:

Una de las razones principales de la adopción de una metodología para desarrollo consiste en la definición de las tareas que serán llevadas a cabo por los individuos que participan del proyecto. Los roles definen ese conjunto cohesivo de actividades realizadas y artefactos mantenidos que son llevados a cabo por personas o por grupos de personas. No sólo se refieren a personas internas al desarrollo del software, sino que también involucran a los usuarios u otras personas que se vean afectadas por el proyecto, cualquiera de los denominados stakeholders.

2.1.1. Líder de Proyecto (Project Manager):

Tiene a su cargo la planificación del proyecto, a lo largo de todo el ciclo de vida, incluida la planificación en detalle de cada iteración; asigna recursos y delega responsabilidades en los mismos; fomenta la cohesión del grupo y lleva a cabo actividades destinadas a eliminar fricciones; organiza las reuniones a ser realizadas; monitorea el progreso del proyecto y establece estrategias para mitigar los riesgos que se puedan presentar.

2.1.2. Analista (Functional Analyst):

Tiene a su cargo el relevamiento, mediante el cual se obtienen los requerimientos de la aplicación a ser construidos en cada iteración; realiza la especificación de los requerimientos; prepara el documento de Visión.

2.1.3. Programador o Desarrollador (Designer-Programmer)

Tiene a su cargo la codificación de los componentes a desarrollar en la iteración; debe crear y ejecutar los tests unitarios realizados sobre el código desarrollado; es responsable de las clases que ha desarrollado debiendo documentarlas, actualizarlas ante cambios y mantenerlas bajo el control de configuración de las mismas.

2.1.4. Tester

Tiene a su cargo la generación de pruebas funcionales a partir de los requerimientos extraídos por los Analistas.

2.2. Fases e Hitos

Como cualquier metodología, ésta se compone de un conjunto de fases que son realizadas a lo largo del tiempo, las mismas conforman un período de tiempo encuadrado entre hitos significativos para el proyecto.

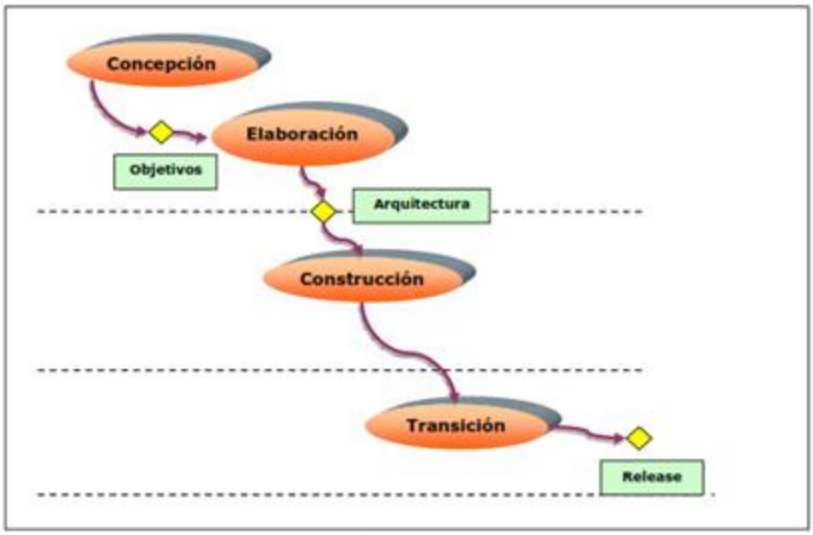

Figura 2: Fases que componen la Metodología Propuesta

- De acuerdo a la Figura, se observa la primera fase denominada **Concepción** que consiste en la definición de las características que tendrá la aplicación, el alcance de la misma, la identificación de los stakeholders, la customización del proceso a ser usado y llevar a cabo la planificación general del proyecto. Esta fase provee la fundación en que todo el posterior trabajo es basado. Es crítico en esta fase llevar a cabo los primeros relevamientos que se realizan con el Cliente de forma de adquirir

conocimiento del dominio y analizar si los costos del proyecto estarán justificados o bien conviene comprar algún software empaquetado o cancelar la ejecución. La fase de Concepción finaliza con un hito mayor denominado Objetivos del Ciclo de Vida en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

- La siguiente fase denominada Elaboración, se refiere a la exploración de los requerimientos más críticos (funcionales y no funcionales) que involucra el proyecto, así como las decisiones técnicas más importantes que quedarán plasmadas en el documento de Arquitectura. El objetivo principal consiste en asegurar la factibilidad técnica respecto a la realización del proyecto. Es a partir de la próxima fase cuando se comienza con la construcción del software, en donde se comprometen la totalidad de los recursos necesarios para que el desarrollo se complete en las iteraciones planificadas. Asimismo, se podrán incorporar recursos en forma limitada tratando de no obstaculizar el normal transcurso del proyecto debido a la capacitación que estos últimos requerirán. La fase de Elaboración finaliza con un hito mayor denominado Arquitectura del Ciclo de Vida en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

- Una vez que pasamos a las etapas de producción tenemos dos fases concatenadas que se realizan en forma repetitiva por cada presentación de la aplicación. Estas fases son Construcción y Transición. Durante la Construcción se terminan de especificar los casos de uso correspondientes a la iteración, se diseñan los mismos bajo la arquitectura candidata presentada, y se codifican todos los componentes definidos por los casos de uso, ejecutándose el testing correspondiente y la integración. Cuando se quiera pasar un cierto conjunto de componentes al entorno productivo se tendrá una fase de Transición en la cual se llevarán a cabo las actividades de despliegue necesarias. La fase de Transición finaliza con un hito mayor denominado presentación del Producto en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

- Cabe mencionar que dentro de las iteraciones de construcción se llevan a cabo actividades relacionadas con Requerimientos, Diseño, Testing, etc., ya que se trata de un proceso iterativo e incremental es que se va llevando a cabo tareas en paralelo y la aplicación va evolucionando hasta cumplir los casos de uso definidos.

2.3. Disciplinas dentro de las Fases

Para planificar un proyecto y permitir tener visibilidad sobre el progreso y el grado de avance del mismo se propone dividir el proceso en disciplinas que conforman agrupaciones de actividades en las cuales se produce un conjunto particular de artefactos. Las disciplinas cubiertas son las que se muestran en la siguiente figura y se irán realizando dentro de cada fase con diversos grados de paralelismo entre ellas.

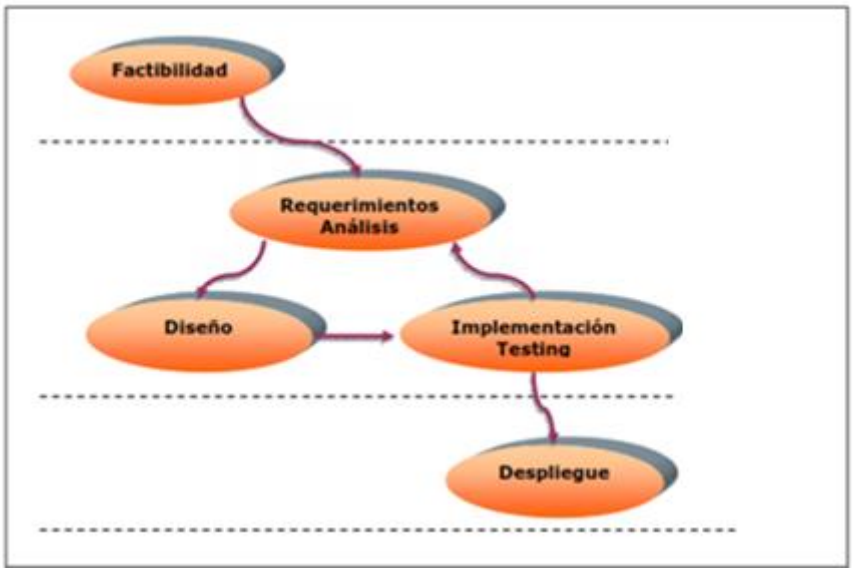

Figura 3. Disciplinas que componen la metodología Propuesta

2.3.1. Factibilidad

Durante la disciplina de Factibilidad se produce el primer contacto entre el equipo de desarrollo y el cliente. Esta disciplina permite al equipo de desarrollo adquirir todo el conocimiento posible acerca del dominio, las necesidades de los usuarios, los objetivos y expectativas que debe cumplir el software que está siendo construido.

Mediante las actividades incluidas en esta disciplina, seremos capaces de responder las siguientes preguntas: ¿Es conveniente construir la aplicación? ¿Es conveniente para el cliente el desarrollar un sistema con todas las complejidades inherentes para satisfacer las necesidades del usuario? ¿Cuáles son las posibles soluciones que se pueden plantear?
A medida que se ejecuta esta disciplina, se va realizando el Modelado del Dominio tendiente a entender las operaciones que

forman parte del dominio del problema, el dominio manejado por el cliente. En forma conjunta, los analistas comenzarán a entender el problema planteado comenzando a bosquejar cuáles son los features que tendrá el sistema y comenzará a realizar modelos y diagramas planteando una arquitectura preliminar que pueda servir de solución al problema en cuestión.

Durante esta disciplina el énfasis está puesto en la exploración tanto del problema como de la solución. Es esencial el aprendizaje del dominio del cliente, de cómo el sistema puede solucionar los problemas que se presentan, de cómo esta solución puede ser realizada con las herramientas disponibles, de las restricciones impuestas sobre el sistema.

Como se observa en la Figura 4, el entendimiento del negocio permite al equipo de desarrollo conocer explorar soluciones adecuadas a las necesidades de los usuarios; este conocimiento puede ser plasmado en un documento de Modelo del Dominio que debe ser construido conjuntamente con los usuarios y validados por estos.

Mientras se va desarrollando el Modelo del Dominio, se van explorando las distintas posibilidades en relación al dominio de la solución. La solución técnica propuesta deberá ajustarse a los estándares que posea el cliente, y el equipo de desarrollo deberá asegurarse de la factibilidad de su implementación. Para este punto se puede tener un documento de Descripción de Arquitectura que detalle las decisiones técnicas tomadas a lo largo de la Factibilidad.

Figura 4. Disciplina Factibilidad en detalle

Como se observa en el detalle, una vez que se finaliza la exploración y se han creado los documentos que comuniquen el resultado de la

misma al cliente, se deberá tener concurrencia de este y de los demás stakeholders en relación al compromiso para que el proyecto sea continuado o cancelado.

La Factibilidad finaliza con la planificación del contenido funcional que será desarrollado en cada una de las iteraciones en que se construirá la aplicación. En dicho plan, se determinarán cuáles serán las iteraciones con presentaciones o entregas, las cuales servirán como hitos del progreso del proyecto tanto para el cliente como para el equipo de desarrollo.

2.3.2. Requerimientos-Análisis

Una vez completadas las actividades dentro de la disciplina de Factibilidad, comienza el desarrollo netamente iterativo. Las iteraciones se suceden mediante la realización de actividades relacionadas con las subsiguientes disciplinas. A continuación, explicaremos en detalle la primera disciplina con que comienzan las iteraciones.
En la disciplina de Requerimientos-Análisis se realizan actividades tendientes a capturar las necesidades de los stakeholders, transformar las mismas en características de alto nivel y especificarlas posteriormente en casos de uso. Como se observa en la Figura 5 se deberá especificar el espectro funcional el cual servirá para que los desarrolladores diseñen e implementen los casos de uso y también el espectro no funcional que posteriormente utilizará el arquitecto para construir la arquitectura de la aplicación y el prototipo.

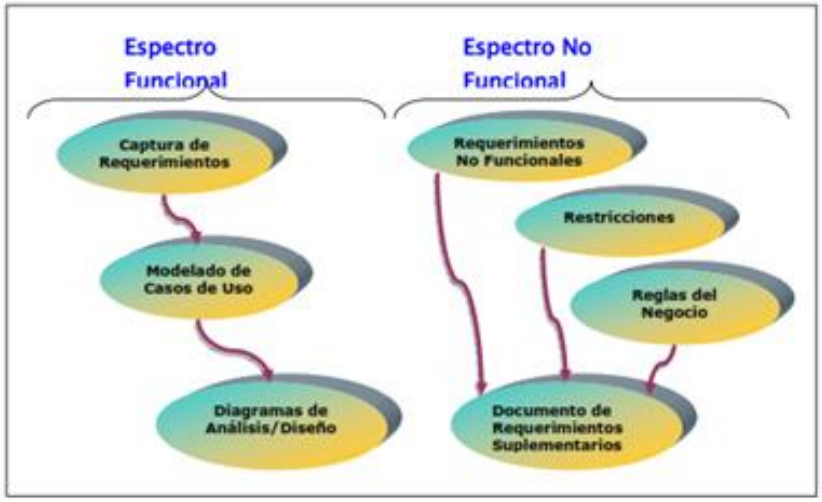

Figura 5. Disciplina Requerimientos – Análisis en detalle

Dadas las características de la metodología tendientes a seguir un proceso ágil que puede administrar requerimientos altamente volátiles se sugiere utilizar el patrón denominado Orientación al Cliente, el cual consiste en maximizar la comunicación directa con las personas que terminarán utilizando el sistema para asegurarnos de estar construyendo lo que estas necesitan.

Una de las técnicas más utilizadas en los procesos ágiles son las entrevistas, las cuales tendrán un carácter más informal ya que se podrán dar en cualquier momento que el analista deba aclarar cuestiones respecto a los requerimientos, detallar aspectos de un caso de uso, etc. En caso que se necesite hacer participar a un número importante de stakeholders en el relevamiento, se recomiendan sesiones de brainstorming (reunión creativa) dirigidas por el Líder de Proyecto a las cuales asisten los analistas documentando todo las decisiones tomadas.

Posteriormente a la Captura de Requerimientos se realiza la primera aproximación de los mismos en una forma estructurada que consiste en los casos de uso, propuestos por Ivar Jacobson a principios de los ’90. Los mismos se han transformado en un estándar en la industria de la informática y han trascendido más allá de la comunidad de objetos de donde surgieron para ser utilizados ampliamente en proyectos de todo tipo en el marco de IS/IT. La principal ventaja de los mismos es que son escritos en el lenguaje del cliente y no demandan tener conocimientos técnicos para ser entendidos. Son esenciales en los procesos ágiles en que se intenta que exista comunicación constante y fluida entre los miembros del equipo y el Cliente.

2.3.3. Diseño

La disciplina de Diseño consiste en plasmar el problema planteado (que se halla en forma de casos de uso o requerimientos contenidos implícitamente por Analistas o Clientes) en el dominio de la solución. Asimismo, también se realiza la definición de la arquitectura siendo validada mediante algún prototipo (conocidos como Proof Of Concept).

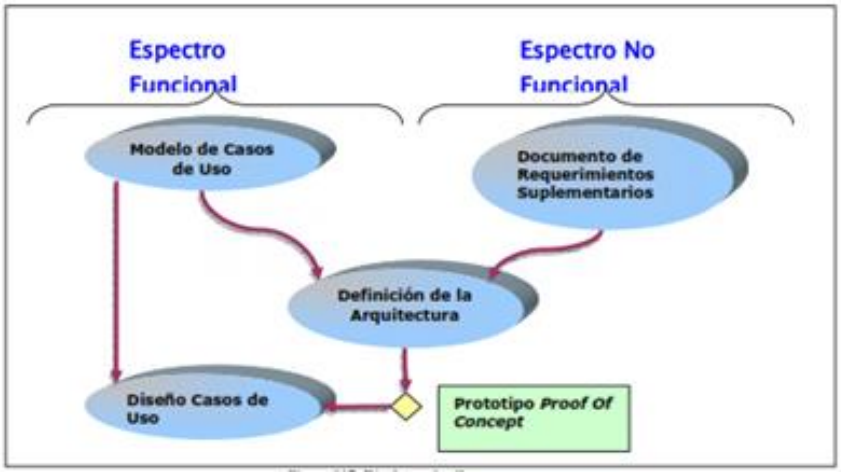

Figura 6.Disciplina Diseño en detalle

Durante las primeras iteraciones cabe destacar la necesidad de diseñar la infraestructura sobre la cual será construido el software. Para esto el arquitecto tomará del conjunto de casos de uso relevados hasta el momento aquellos que considere más importantes desde un punto de vista técnico y también analizará los requerimientos suplementarios que deberán ser satisfechos por la arquitectura propuesta. Con esto construirá un modelo de arquitectura documentado que especificará los elementos a partir del cual será construido el sistema, las interacciones entre dichos elementos, los patrones que se usaron para su composición, y las restricciones sobre dichos patrones.

Finalmente, para probar la factibilidad de la arquitectura el Arquitecto construirá un prototipo ejecutable utilizado para verificar la conformación a los requerimientos de la aplicación.

El Diseño es realizado tomando como input los casos de uso escritos en la disciplina anterior y detallando los mismos en un esquema de clases, atributos y métodos del lenguaje elegido. Los miembros del equipo adaptan su modelo mental para cerrar la brecha que existe del dominio del problema, el qué, al dominio de la solución, el cómo. Este proceso es plasmado mediante la generación de Diagramas de Secuencia para aquellas interacciones significativas desde un punto de vista técnico las cuales podrán ser anexadas al Documento de Arquitectura. El nivel de ceremonia estará acorde a la criticidad del sistema que se desarrolla. En casos minimalistas y donde se desee tener un mínimo realce metodológico el diseño podrá ser realizado en papel o en un pizarrón, en casos de mayor ceremonia se

utilizarán herramientas de modelado visual que ayuden en este proceso.

Una función esencial es la de asignación de funcionalidad en que el Líder de Proyecto se encargará de nombrar a un responsable por cada unidad de funcionalidad especificada. Puede tratarse de un caso de uso, de un grupo de clases, de un paquete. Lo importante es que exista un responsable para cada artefacto que mantenga actualizada la trazabilidad requerida por la metodología pudiendo realizar cualquier tipo de modificación sobre el mismo y manteniendo un adecuado control de cambios.

Una vez finalizado el proceso de diseño detallado de las clases a ser construidas durante la iteración, se procede a la siguiente disciplina que es la de ImplementaciónTesting.

2.3.4. Implementación - Testing

En la disciplina de Implementación-Testing se lleva a cabo la producción del software. El equipo de desarrollo se encarga de implementar la funcionalidad especificada en los casos de uso mediante el lenguaje de programación elegido.

Dado el énfasis que se propone en reducir la brecha comunicacional entre las personas que forman el equipo, esto mismo repercute en la preferencia de elegir para proyectos de esta envergadura aquellos lenguajes de programación que minimicen la brecha entre la realidad y la realización de la misma en un lenguaje. Idealmente se fomenta el uso de frameworks o tecnologías bajo el paradigma de orientación a objetos, como Java, MS. NET o Smalltalk, los cuales permiten el máximo de eficiencia por línea de código para aplicaciones de IS/IT.

En paralelo a la construcción de los componentes se realiza la revisión de los mismos mediante el testing, el cual tiene una gran relevancia.

La metodología define diversas pruebas que podrán ser realizadas en las iteraciones. Las mismas incluyen:

Pruebas Unitarias:

Definición: las Pruebas Unitarias son pruebas realizadas a nivel de las clases y métodos construidos para la aplicación. Estas pruebas

son implementadas por los mismos programadores en el lenguaje de programación utilizando clases de prueba encargadas de verificar el comportamiento correcto de los métodos en una clase. Para ello es conveniente contar con algún framework de testing unitario que le facilite al equipo de desarrollo la creación, mantenimiento y ejecución de una suite de pruebas automatizadas.

Alcance: las mismas son utilizadas para controlar la calidad del código construido y son ideales para testing de integración y de regresión.

Pruebas Funcionales:

Definición: las Pruebas Funcionales verifican el flujo de interacción de la funcionalidad definida en los casos de uso. Esta funcionalidad – que está escrita en forma narrativa en los casos de uso y es plasmada mediante la interacción de objetos enviándose mensajes entre sí en los diagramas de secuencia – debe ser especificada en pruebas automatizadas que serán creadas por el Usuario en conjunción con la ayuda técnica del Tester y serán ejecutadas por el Tester durante cada iteración. Cabe remarcar que debe ser el Usuario el que define el contenido de las mismas ya que de esta forma proveen el feedback que permite al equipo de desarrollo continuar las iteraciones asegurándose de la calidad del producto construido.

Alcance: las mismas son utilizadas durante todo el desarrollo y su alcance está definido por el conjunto completo de requerimientos funcionales definidos en los casos de uso.

Pruebas de Aceptación:

Definición: el objetivo de las Pruebas de Aceptación es la verificación que el software construido en las iteraciones cumple con las funcionalidades solicitadas por el usuario y está listo para ser utilizado en el ambiente del Cliente. Estas pruebas revisten gran relevancia porque implican que el Cliente da conformidad respecto al sistema desarrollado y lo acepta

Alcance: las mismas son realizadas durante las Iteraciones con Release en las que se lleva a cabo la transición de la aplicación al ambiente de producción en el Cliente.

Despliegue

La disciplina de Despliegue permite que el sistema construido sea transferido al ambiente de producción para ser utilizado por el usuario. Esto no significa que el software debe estar implementado en su totalidad, cubriendo el 100% de los aspectos funcionales y no funcionales sino que en una iteración determinada se desee liberar una versión con un porcentaje de los casos de uso implementados. Dado el esquema de desarrollo iterativo e incremental, durante las iteraciones ya mencionadas se va agregando funcionalidad y se hace "crecer" al sistema hasta que cumpla con los requerimientos que se especificaron al principio, sumados a los cambios surgidos a lo largo del desarrollo.

Durante esta disciplina, se deben realizar actividades tendientes a hacer una entrega completa con una funcionalidad previamente determinada la cual será desplegada en producción. Las actividades incluidas dentro de esta disciplina son:

- ✓ Generar Manuales del Usuario, de Operación, etc.
- ✓ Realizar un empaquetamiento de componentes junto con scripts de instalación que el Cliente utilice para instalar la aplicación en su entorno.
- ✓ Ejecutar el conjunto completo de pruebas funcionales creadas durante el desarrollo, hasta obtener la aceptación del Cliente.
- ✓ En caso de ser necesario, definir las actividades de migración al nuevo sistema.
- ✓ Capacitar a los Usuarios que utilizarán el nuevo sistema.
- ✓ Generar la Nota de Entrega con la totalidad de los artefactos que se le entregan al Cliente (con su correspondiente versionado) al final de la disciplina.

2.4. Artefactos

Las metodologías ágiles en su mayoría tienden a minimizar la cantidad de artefactos generados en un proyecto. Sin embargo, existen muchos artefactos necesarios para propósitos de comunicación, diseño, gestión sin los cuales un proyecto no llegaría al éxito. La metodología propuesta enumera algunos artefactos que considera requisitos mínimos para tener madurez en el proceso de desarrollo. En la sección de Anexos de esta tesis se han colocado templates para la mayor parte de los artefactos aquí descritos.

2.4.1. Documento de Visión

Definición: el Documento de Visión es realizado durante la fase de Concepción y sirve como contrato entre el Cliente y el Equipo de Desarrollo respecto a lo que se va a construir. En la Visión deberán identificarse los stakeholders del proyecto, la totalidad de las características del sistema a ser construido y los requerimientos suplementarios que se detectaron en forma temprana. En forma optativa podrá incluirse un resumen de los casos de uso críticos del aplicativo.

Responsable: la Visión es construida por un Analista Funcional en conjunto con el Líder de Proyecto. La misma debe ser mantenida para reflejar los cambios al alcance durante el proyecto.

2.4.2. Plan de Proyecto:

Definición: el Plan de Proyecto es un documento mantenido por el Líder de Proyecto con toda la información de gestión. El mismo suele incluir un Gantt con el cronograma y las tareas del proyecto. Dependiendo del nivel de formalidad se generará un WBS y/o algún otro tipo de plan a incluirse dentro de este (plan de testing, de comunicaciones, de administración de requerimientos, de administración de cambios, etc.). El Plan de Proyecto es creado en forma temprana durante la Concepción y actualizado durante las fases posteriores.

2.4.3. Lista de Riesgos:

Definición: la Lista de Riesgos se utiliza para capturar los riesgos más relevantes que se presentan en el proyecto, y para poder monitorearlos, asignarles prioridad, impacto y probabilidad de ocurrencia. Sirve para planificar las iteraciones y para tener en vista aquellos riesgos que, por su criticidad, deberán ser mitigados lo más tempranamente posible.

Responsable: el Líder de Proyecto es el responsable y el principal consumidor de este documento y deberá mantenerlo actualizado hasta la terminación del proyecto.

2.4.4. Modelo de Casos de Uso y Casos de Uso:

Definición: los Casos de Uso se utilizarán para especificar los requerimientos funcionales de la aplicación. Los mismos servirán para guiar los demás artefactos y para la construcción de los componentes de la aplicación. Asimismo, se recomienda la generación del Modelo de Casos de uso para tener una visualización gráfica del universo funcional de la aplicación y poder comunicarla tanto internamente como al Cliente para su validación. Los Casos de Uso y el Modelo serán creados en las primeras fases del proyecto, aunque podrán también ser especificados en las iteraciones de construcción.

Responsable: los Analistas son responsables de la creación y el mantenimiento de estos artefactos, los cuales serán consumidos por el resto del equipo de desarrollo.

2.4.5. Documento de Especificación de Requerimientos de Software (SRS):

Definición: el documento de SRS se utilizara para especificar los requerimientos funcionales de carácter más técnico de la aplicación. También en este se incluirían los requerimientos no funcionales, las reglas de negocio, y las restricciones que se aplicarán a la solución. En este documento se pondrán todos los requerimientos que queden fuera de los casos de uso. La idea es tener algún tipo de constancia documental de todos aquellos requerimientos de carácter técnico que debe contemplar la solución.

Responsable: los Analistas son responsables de la creación y el mantenimiento de este artefacto, el cual será consumido por el resto del equipo de desarrollo.

2.4.6. Descripción de la Arquitectura:

Definición: el documento de Descripción de la Arquitectura también conocido como SAD especifica los aspectos técnicos de la solución propuesta por el equipo de desarrollo. Sirve como medio de comunicación entre el Arquitecto y el equipo en relación a cómo deberán ser implementados los casos de uso de la aplicación.

2.4.7. Casos de Prueba:

Definición: los Casos de Prueba contienen la especificación de cómo será validado el sistema. Estos son realizados basándose en los casos de uso y planteando todos los escenarios posibles que éstos pueden contener. Dado que puede ser bastante burocrático realizar la totalidad de Casos de Prueba existentes en un sistema, se recomienda generar los más importantes y después anotar las variantes en un Caso de Prueba Estándar.

2.4.8. Planilla de Incidentes:

Definición: la Planilla de Incidentes será utilizada para registrar y tener seguimiento de aquellos bugs, mejoras, tareas que el Tester o alguna persona de aseguramiento de la calidad necesite reportar. Son los denominados sistemas de Issue Tracking que pueden ser implementados manualmente o mediante alguna herramienta.

Responsable: la Planilla de Incidentes será utilizada por todos los miembros del equipo. En particular será muy utilizada por los Testers quienes cargarán todos los bugs encontrados durante la ejecución de los Casos de Prueba los cuales serán leídos y corregidos por los Programadores.

2.4.9. Nota de Entrega:

Definición: la Nota de Entrega sirve para especificar aquello que se le entrega al Cliente en una entrega determinada de la aplicación. En la misma se constatará el número de versión, los cambios de la versión anterior, los bugs conocidos y las mejoras efectuadas.

Responsable: cualquier miembro del equipo de desarrollo que vaya a entregar algún artefacto al Cliente deberá crear la correspondiente Nota de Entrega.

CONCLUSIONES

- ✓ Se logró desarrollar una metodología ágil de desarrollo de software para la aplicación en proyectos de pequeña escala, mediante las especificaciones de los roles, fases e hitos, disciplinas y artefactos, proporcionando templetes de los artefactos de la metodología propuesta.

- ✓ Se logró estudiar las diferentes metodologías agiles existentes en la actualidad para la recopilación de información precisa, que sirva como base para el desarrollo de una nueva metodología ágil propuesta.

- ✓ Se determinó, se logró estudiar y conocer los diferentes roles, faces e hitos, artefactos y procesos existentes en la metodología que se está proponiendo.

- ✓ Se brindó los Templetes(Formatos) de los artefactos de la metodología propuesta para un mayor entendimiento, desarrollo y aplicación de la metodología propuesta.

REFERENCIA BIBLIOGRAFÍA

- Metodologías para el desarrollo de software.
  "INGENIERIA DEL SOFTWARE". SOMMERVILLE, I.
  Ed. Addison-Wesley.

- Metodología RUP
  G. Booch, J. Rumbaugh, I. Jacobson: "El lenguaje unificado de modelado. Guía de usuario". 2ª Edición. Addison Wesley, 2006.

- Métrica V.3
  "Metodología de planificación y desarrollo de sistemas de información. Métrica Versión 3". m.a.p.
  Editado por el Ministerio para las Administraciones Públicas.
  URL disponible:
  http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

- Metodologías agiles para el desarrollo de software: Extreme Programming XP
  Cáceres, Paloma y Marcos, Esperanza, "Procesos Ágiles para el Desarrollo de Aplicaciones Web"; http://www.dlsi.ua.es/webe01/articulos/s112.pdf

- SCRUM la metodología de desarrollo ágil por excelencia
  URL disponible:
  http://vassdigital.com/blog/scrum-la-metodologia-de-desarrollo-agil-por-excelencia/

- SCRUM como metodología
  "Proyectos Ágiles.org como gestionar proyectos con Scrum";
  http://www.proyectosagiles.org/beneficios-de-scrum
  Meza Martínez, Jorge Iván, "Introducción a la implementación de Scrum";
  http://www.jorgeivanmeza.com/

ANEXOS

Anexo A - Templates de Artefactos

A continuación se proponen distintos templates para los artefactos recomendados. Los mismos podrán ser tomados por el Coordinador como punto de partida al momento de implementar la metodología en la organización.

Los templates que se ofrecen son para los siguientes artefactos:

- **Visión**
- **Lista de Riesgos**
- **Especificación de Casos de Uso**
- **Documento de Especificación de Requerimientos de Software (SRS)**
- **Descripción de la Arquitectura**
- **Casos de Prueba**
- **Nota de Entrega**

Descripción del Proyecto
Documento de Visión

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción..... 2
Referencias 2
2. Posicionamiento.....2
Situación Actual 2
3. Stakeholders y Usuarios.....2
Stakeholders identificados..... 2
Usuarios identificados..... 2
4. Características del Producto 2
Features de Alto Nivel.....2
5. Otros Requerimientos.....2
Requerimientos No Funcionales.....3
Restricciones.....3
Reglas del Negocio Clave.....3

<<Datos de la empresa>>

1. Introducción

[Descripción del documento y del contexto del proyecto de desarrollo. Se identifica al Cliente y el Propósito detrás de la construcción del sistema.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Posicionamiento

Situación Actual

[Descripción del problema actual que se presenta y la forma en que el sistema beneficiará a los usuarios mediante una solución dada.]

3. Stakeholders y Usuarios

Stakeholders identificados

[Tabla que muestra a los stakeholders involucrados en el proyecto. Se da una breve descripción de cada uno y se analiza el impacto de este en la solución.]

Usuarios identificados

[Tabla que muestra a los usuarios que tendrá el sistema. Se da una breve descripción de cada uno y se analiza las responsabilidades en relación al sistema en construcción.]

4. Características del Producto

[A continuación se detallan las características del producto en forma de features de alto nivel. Estos identifican las capacidades que tendrá el sistema para cubrir las necesidades de los usuarios.]

Features de Alto Nivel

[Feature 1.]
[Feature 2.]
...
[Feature X.]

5. Otros Requerimientos

[A continuación se detallan las características del producto en forma de requerimientos no funcionales de alto nivel, restricciones a ser impuestas a la solución y reglas del negocio que determinarán la descripción de la arquitectura candidata resultante.]

Requerimientos No Funcionales

[Requerimiento No Funcional 1.]
[Requerimiento No Funcional 2.]
...
[Requerimiento No Funcional X.]

Restricciones

[Restricción 1.]
[Restricción 2.]
...
[Restricción X.]

Reglas del Negocio Clave

[Regla del Negocio 1.]

[Regla del Negocio 2.]
...
[Regla del Negocio X.]

Descripción del Proyecto
Lista de Riesgos

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción 2
Propósito 2
2. Riesgos 2
<<Identificación del Primer Riesgo>> 2
 Descripción 2
 Criticidad 2
 Probabilidad de Ocurrencia 2
 Impacto 2
 Estrategia de Mitigación 2
 Plan de Contingencia 2
<<Identificación del Segundo Riesgo>> 2
..... 2

<<Datos de la empresa>>

1. Introducción

Propósito

[Breve comentario del propósito de este documento..]

2. Riesgos

[A continuación se enumerarán todos los riesgos que presenta el proyecto. En dicha lista deben figurar Tanto los riesgos de management identificados por el Líder de Proyecto, como los riesgos más técnicos identificados por el Arquitecto.]

<<Identificación del Primer Riesgo>>

- **Descripción**

  [Breve descripción del riesgo.]

- **Criticidad**

  [Indicar el nivel de criticidad en caso de que el riesgo se materialice. El rango puede estar en una escala de alta, media o baja.]

- **Probabilidad de Ocurrencia**

  [Indicar la probabilidad de que ocurra el riesgo en el proyecto. El rango puede estar en una escala de alta, media o baja.]

- **Impacto**

  [El impacto está dado por el producto de la criticidad y la probabilidad de ocurrencia En base a esto los riesgos serán priorizados y monitoreados por el Líder de Proyecto y el Arquitecto. Por ejemplo, un riesgo con criticidad alta y probabilidad de ocurrencia alta deberá ser monitoreado frecuentemente.]

- **Estrategia de Mitigación**

  [Indicar la estrategia a ser utilizada para minimizar el impacto del riesgo en caso que este se materialice.]

- **Plan de Contingencia**

  [Indicar el plan de contingencia que será tenido en cuenta en caso que la estrategia de mitigación no tenga éxito.]

<<Identificación del Segundo Riesgo>>

...

Descripción del Proyecto
Código y Nombre del Caso de Uso

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Breve Descripción 2
2. Actores 2
3. Flujo de Eventos 2
3.1. Flujo Básico 2
3.2. Flujos Alternativos 2
3.3. Flujos de Excepción 2
4. Requerimientos Suplementarios 2
5. Precondiciones 2
6. Poscondiciones 2
7. Puntos de Extensión 2
8. Diagramas Asociados 2
9. Supuestos y Dudas 2

<<Datos de la empresa>>

1. Breve Descripción

[Breve descripción del propósito del caso de uso. Recordar que el mismo debe dar algún valor a algún actor del sistema.]

2. Actores

[Enumerar los actores que participan en este caso de uso.]

3. Flujo de Eventos

[A continuación se describe la secuencia de pasos desde que el actor dispara el caso de uso hasta que la interacción llega a su fin.]

3.1. Flujo Básico

[El flujo básico representa la interacción normal del caso de uso. Es la situación en que todos los pasos se ejecutan normalmente, sin existir excepciones ni errores.]

3.2. Flujos Alternativos

[Los flujos alternativos se dan ante alternativas complejas dentro de la ejecución del flujo las cuales ocurren ante ciertos eventos alternos de la interacción.]

3.3. Flujos de Excepción

[Los flujos de excepción ocurren cuando existen excepciones en la ejecución del flujo básico debida a errores internos de la aplicación o de la interacción del usuario con esta.]

4. Requerimientos Suplementarios

[Indicar aquellos requerimientos suplementarios relacionados únicamente con este caso de uso.
Recordar que los requerimientos suplementarios globales van en el SRS]

5. Precondiciones

[Indicar las condiciones que deben darse para que se pueda disparar la ejecución de este caso de uso.]

6. Poscondiciones

[Indicar como quedará el sistema después que se termine la ejecución de este caso de uso.]

7. Puntos de Extensión

[Puntos a partir de los cuales se extiende el flujo básico de este caso de uso. Relacionado con la relación <<extends>> de UML.]

8. Diagramas Asociados

[Colocar aquellos diagramas asociados con este caso de uso. En este punto se puede colocar el modelo de casos de uso afectado a este caso de uso, diagrama de estado, diagrama de actividad, prototipo de
GUI, etc.]

9. Supuestos y Dudas

[Dejar constancia de los supuestos que se han tomado en la especificación del casos de uso así como las dudas que el Analista tiene pendiente y deberán ser resueltas con el Cliente.]

Descripción del Proyecto
Documento de Especificación de Requerimientos de Software (SRS)

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción 2
Propósito 2
Referencias 2
2. Requerimientos Funcionales 2
Detalle de Requerimientos Funcionales 2
3. Requerimientos Suplementarios 2
Requerimientos No Funcionales 2
Restricciones 2
Reglas del Negocio 2

<<Datos de la empresa>

1. Introducción

Propósito

[Breve comentario del propósito de este documento..]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Requerimientos Funcionales

Detalle de Requerimientos Funcionales

[A continuación se enumerarán todos aquellos requerimientos de carácter funcional que tengan que ver con aspectos más técnicos relacionados con la solución. Los mismos quedan fuera de las especificaciones de los casos de uso.]

3. Requerimientos Suplementarios

Requerimientos No Funcionales

[A continuación se enumerarán todos aquellos requerimientos no funcionales que son globales a la aplicación. Los mismos refieren a las cualidades sistémicas asociadas con: usabilidad, performance, mantenibilidad, seguridad, tolerancia, etc.]

Restricciones

[A continuación se enumerarán todas las restricciones que se aplican a la solución planteada. Las mismas restringen los grados de libertad al momento de diseñar una solución determinada a un problema.]

Reglas del Negocio

[A continuación se enumerarán todas aquellas reglas del negocio globales a la aplicación. Las mismas tienen que ver con las políticas y/o condiciones que se dan en los procesos del negocio y que deben ser implementadas en la solución.]

Descripción del Proyecto
Descripción de Arquitectura

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción 2
Propósito 2
Referencias 2
2. Objetivos Arquitectónicos 2
3. Vista de Casos de Uso 2
4. Vista Lógica 2
5. Vista de Despliegue 2
6. Bibliografía 2

1. Introducción

Propósito

[Breve comentario del propósito de este documento..]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Objetivos Arquitectónicos

[Se mencionarán los requerimientos no funcionales y restricciones que guiaron el trabajo del arquitecto.
Los mismos estarán basados en lo que fue documentado en el SRS, pero haciendo énfasis sobre como la arquitectura candidata resuelve cada uno de estos.]

3. Vista de Casos de Uso

[Es una descripción de una vista de la arquitectura del software a través de los casos de usos.
Representa el conjunto de escenarios de la iteración que describen o representan la funcionalidad requerida para el proyecto.]

4. Vista Lógica

[Es una descripción de una vista de la arquitectura del software a través de la evaluación realizada de los requerimientos funcionales y no funcionales. Representa la organización en paquetes de servicios y subsistemas y la organización de estos subsistemas en layers. También se describen aquí las realizaciones de caso de uso más importantes.]

5. Vista de Despliegue

[La vista de despliegue permitirá describir la distribución física posible de los diferentes nodos de procesamiento describiendo además el modelo de crecimiento de la solución.]

6. Bibliografía

[En este apartado se podrá mencionar la bibliografía de donde se puede obtener más información de los elementos mencionados en este documento.]

Descripción del Proyecto
Caso de Prueba

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción 2
Propósito 2
Referencias 2
2. Definición de Casos de Prueba 2
Número Caso de Prueba 2
Título Caso de Prueba 2
ID de Caso de Uso 2
Módulo/Componente 2
Funcionalidad 2
Modo de Prueba 2
Preparación de Configuración 2
Procedimiento de la Prueba 2
Resultados Esperados 2

1. Introducción

Propósito
[Breve comentario del propósito de este documento..]

Referencias
[Indicaremos todos los artefactos relacionados con este documento.]

2. Definición de Casos de Prueba

Número Caso de Prueba
[Número del caso de prueba usado para identificación y posteriormente para tener estadísticas.]

Título Caso de Prueba
[Título descriptivo del caso de prueba.]

ID de Caso de Uso
[Identificación del caso de uso. Por ejemplo, podría ser algo como CUXXX Alta de Usuarios.]

Módulo/Componente
[Módulo de la aplicación que esta siendo probado.]

Funcionalidad
[Funcionalidad de la aplicación que se prueba.]

Modo de Prueba
[Manual o automático (mediante algún script o robot).]

Preparación de Configuración
[Preparación de configuración para llevar a cabo la prueba. En este punto colocaremos cualquier infraestructura o configuración o carga de datos que requiera la prueba.]

Procedimiento de la Prueba
[Secuencia de pasos para llevar a cabo la prueba.]

Resultados Esperados
[Resultados esperados por la prueba.]

Descripción del Proyecto
Nota de Entrega

<Versión>
<Autor>

Historia de Revisión

FECHA	Versión	Descripción	Autor

Tabla de Contenidos

Tabla de Contenidos 1

1. Introducción 2
Propósito 2
Referencias 2
2. Información de la Entrega 2
Overview 2
3. Nuevos Features 2
Lista de Nuevos Features 2
Features 2
4. Bugs Conocidos o Limitaciones 2
Lista de Bugs Conocidos 2
Bugs 2
Lista de Limitaciones 2
Limitaciones 3

<<Datos de la empresa>

1. Introducción

Propósito

[Breve comentario del propósito de este documento.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Información de la Entrega

Overview

[Breve descripción del contenido de la entrega que se lleva a cabo. Se enumeran todos los artefactos que la misma incluye, identificando el número de versión donde corresponda.]

3. Nuevos Features

Lista de Nuevos Features

[Listado de los features que están contenidos en la versión que se entrega al Cliente. Orientado a entregables de implementación.]

Features

[Feature 1.]
[Feature 2.]
...
[Feature X.]

4. Bugs Conocidos o Limitaciones

Lista de Bugs Conocidos

[Listado de los bugs conocidos que no han podido ser corregidos para la corriente entrega. Los mismos serán identificados de forma que el Cliente los conozca y no necesite reportarlos nuevamente al equipo.]

Bugs

[Bug 1.]
[Bug 2.]
...
[Bug X.]

Lista de Limitaciones

[Listado de las limitaciones de la versión entregada. Se detallarán aquellas cuestiones que quedaron fuera de la implementación corriente.]

Limitaciones

[Limitación 1.]
[Limitación 2.]
...
[Limitación X.]