Universidad de las Ciencias Informáticas

FACULTAD 7

Introducción de procedimientos ágiles en la producción de software en la Facultad 7 de la Universidad de las Ciencias Informáticas.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autora

Malay Rodríguez Villar

Tutora

Ing. Maidely Calderón Montero

Ciudad de La Habana

Junio de 2007

Ningún ejército puede detener la fuerza de una idea cuando llega a tiempo.

Víctor Hugo.

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los 27 días del mes de Junio del año 2007

Malay Rodríguez Villar

_____

Ing. Maidely Calderón Montero

_____

DATOS DE CONTACTO

Nombre y Apellidos del **Tutor**: Maidely Calderón Montero.

Email: mcalderon@uci.cu

Curriculum: Profesor graduado de Ing. Informático en el año 2004. Ha impartido asignaturas como Estructura de Datos, Sistemas de Bases de Datos e Ingeniería de Software. Posee categoría docente de Instructor y cursa la maestría de Gestión de Proyectos Informáticos. Ha presentado ponencias en eventos y forma parte del grupo de investigaciones de Bases de Datos y de Ingeniería de Software.

Nombre y Apellidos del **Co-Tutor**: Abel Meneses Abab

Email: abelma@uci.cu.

Profesor graduado de Ing. en Telecomunicaciones y Electrónica en el año 2004. Ha impartido asignaturas como Sistemas Operativos, Teleinformática II, Práctica Profesional; de las asignaturas del 2do perfil: GNU/Linux Básico, Propiedad Intelectual, Programación en Perl; e imparte postgrados de GNU/Linux nivel Básico, Programación Web. Posee categoría docente de Instructor recién graduado; ha cursado postgrados como: Ciencia, Tecnología y Sociedad, Gnux/Linux, Ideología y Política de la Revolución Cubana. Ha presentado ponencias en eventos y forma parte del grupo de investigaciones de Software Libre de la UCI y del Grupo Técnico Nacional. Es líder del Proyecto Unicornios (Servicios y Soporte para la Migración a SWL de la UCI).

Nombre y Apellidos del **Asesor**: Ailec Granda Dihigo.

Email: agranda@uci.cu.

Curriculum: Profesor graduado de Ing. en Ciencias Informáticas en el año 2006. Ha impartido asignaturas como Ingeniería de Software I, Ingeniería de Software I. Práctica Profesional, Sistema Operativo y Programación. Posee categoría docente de Instructor reciengraduado y cursa la maestría de Gestión de Proyectos Informáticos. Ha presentado ponencias en eventos y forma parte del grupo de investigaciones de Ingeniería de Software. Se desempeña como asesora de Ingeniería de Software en el dpto. Docente Central de Ingeniería y Gestión de Software.

Agradecimientos

Ante todo quiero agradecer el apoyo recibido de todas aquellas personas que de una forma u otra, directa o indirectamente, ayudaron en los momentos críticos y difíciles en la realización de esta investigación:

Particularmente, a mi tutora, la Ing. Maidely Calderón por su entrega y dedicación, sus atenciones y haber puesto en disposición todos sus conocimientos; por no abandonarme nunca y ser incondicional, más que mi tutora ha sido una gran compañera.

Muchos más podrían ser mencionados, compañeros y amigos, que estuvieron siempre en las buenas y en las malas a mi lado; a lo largo de estos 5 años de estudio que están diciendo adiós. Nunca los voy a olvidar.

Dedicatoria

A Rafael y Georgina, por ser los papas más lindos del mundo, por confiar ciegamente en mis pasos, por su amor incondicional, por haberme dado alas para volar.

A Joan David, por su paciencia y comprensión, por ser el sostén de mi vida; y amarme, haciendo de mí una mujer feliz. Una medalla por ser el mejor esposo.

A mi Hermano querido por cuidarme siempre. Rayner, eres el mejor de todos.

A familiares y amigos, por su cariño, confianza, atenciones y por todo lo bello que le han aportado a mi vida.

A nuestro Comandante Fidel, que es el guía de la grande y eterna Revolución Cubana por haberme permitido ser lo que soy hoy, una mujer de bien.

RESUMEN

Un nuevo grupo de metodologías ingenieriles ha surgido en los últimos años: las metodologías ágiles. Estos nuevos métodos han venido a ser una solución a la medida para un segmento importante de proyectos de desarrollo de software con características peculiares. Por lo que proponer una guía de procedimientos ágiles para el proceso de producción de software en la facultad 7 de la Universidad de las Ciencias Informáticas (UCI) es el objetivo de la investigación.

En la investigación se realiza una caracterización de las metodologías ágiles más conocidas y se determinan para cuales proyectos cada una es más factible. Por lo que surge la necesidad, de realizar un estudio de las principales características de los proyectos de la Facultad 7 y posteriormente clasificarlos según sus aspectos más significativos, permitiendo identificar las metodologías que sentaran las bases de vías de solución que contribuyan a mejorar el proceso productivo de la facultad.

Esta tesis ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva pues fomenta el desarrollo de la creatividad, aumenta el nivel de preocupación y responsabilidad de los miembros del equipo y ayuda al líder del proyecto a tener un mejor control del mismo.

Palabras claves: Metodologías ágiles, proceso de software, proyectos de desarrollo

TABLA DE CONTENIDOS

TABLA DE ANEXOS Y FIGURAS

INTRODUCCIÓN

Todo proceso de desarrollo de software es riesgoso y difícil de controlar, pero si no llevamos una metodología de por medio, resulta ser mucho más catastrófico y lo que obtenemos es clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos.

Una interrogante sale a la luz en el primer momento en que se va a emprender la tarea de realizar determinado proyecto: ¿Qué metodología de desarrollo de software se debe usar? Aunque resulta algo sencillo, la respuesta no lo es, pues, el proceso de desarrollo de software no es una tarea fácil, siendo centro de atención en los últimos años la solución más factible y rentable. 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 otros muchos. Por otra parte existen otras propuestas que se centran 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.

No cabe duda de que el tema es de imperante actualidad. La curiosidad que siente la mayor parte de los ingenieros de software, profesores, e incluso alumnos, sobre los métodos ágiles hace prever una fuerte proyección industrial de las metodologías ágiles. Por un lado, para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual considerando las dificultades de su introducción e inversión asociada en formación y herramientas.

Por otro, las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, nuevas tecnologías, etc. Esto último abriría interesantes canales en la producción de software en una

universidad como la nuestra, donde muchos de los proyectos que se desarrollan tienen características semejantes a esos para los cuales han sido especialmente creadas las metodologías ágiles.

En la Universidad de las Ciencias Informáticas (UCI) se desarrollan diversos proyectos productivos. Específicamente en la Facultad 7 se ejecutan sistemas de software relacionados con la salud. Para la modelación de dichos sistemas se emplea la metodología RUP (Proceso Unificado de Rational), basada en un enfoque de desarrollo dirigido a proyectos de gran envergadura.

No en todos los proyectos que se desarrollen es factible utilizar este tipo de metodología, por lo que puede traer aparejado el fracaso de los proyectos por diversos motivos como: exceso del tiempo de desarrollo de los proyectos, ocasionando los cambios constantes y no previstos en la modelación del sistema a implementar, entrega tardía de las versiones de prueba de los sistemas, afectaciones en los costos reales de los sistemas, así como la insatisfacción por parte del cliente con el producto, entre otras.

Para darle solución a la **situación problémica** descrita anteriormente, se plantea como **problema** para la presente investigación, la siguiente interrogante ¿Cómo emplear procedimientos ágiles en la producción de software en la facultad 7 de la Universidad de las Ciencias Informáticas (UCI) de acuerdo con las características de los proyectos que se desarrollan en la misma? Partiendo de la **idea** de que si se introducen procedimientos ágiles en el proceso de desarrollo de software se pueden lograr proyectos más eficientes. Siendo nuestro **objeto de estudio** el proceso de producción de software en la facultad 7 de la Universidad de las Ciencias Informáticas (UCI).

El **campo de acción** que abarca la investigación es utilización de procedimientos ágiles en el proceso de producción de software en la facultad 7 de la Universidad de las Ciencias Informáticas (UCI). Para la introducción y posterior utilización de dichos procedimientos en el proceso productivo se trazo el siguiente **objetivo general**: Proponer una guía de procedimientos ágiles para su uso en el proceso de producción de software en la facultad 7 de la Universidad de las Ciencias Informáticas (UCI).

Como **objetivos específicos** se definieron los siguientes:

- Caracterizar las metodologías ágiles para definir a que proyectos son aplicables.

- Caracterizar los proyectos productivos de la Facultad 7 para identificar los aspectos que sugieren cambios.
- Determinar cuáles procedimientos ágiles, son factibles utilizar en la producción de software en la Facultad 7.

Se pretende tener como **posibles resultados** la inserción de nuevos procedimientos en el proceso productivo de la Facultad 7. Evidentemente que para llegar a esto y cumplir los objetivos se hace necesario desarrollar **tareas**, siendo estas:

- Realizar un estudio de las metodologías ágiles de desarrollo de software.
- Realizar un estudio de las diferentes clasificaciones de proyectos de software.
- Seleccionar los proyectos productivos de la facultad como muestra de la investigación.
- Entrevistar a líderes de proyectos seleccionados.
- Clasificar los proyectos de acuerdo a sus características.
- Seleccionar los procedimientos adecuados según la clasificación de los proyectos.

La investigación está sustentada en 3 capítulos, estructura que se describe a continuación. En el capítulo 1 se realiza el estado del arte de la investigación, donde se esclarecieron los conceptos necesarios sobre las metodologías ágiles de desarrollo así como sus características. En el capítulo 2, se plantean las diferentes clasificaciones de sistemas de software y se realiza un estudio y posterior análisis de los proyectos de la facultad 7 de la UCI. Finalmente, en el capítulo 3, se presenta la propuesta de los procedimientos adecuados a utilizar en la modelación de los sistemas en correspondencia con sus características.

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

El software hoy en día es una parte crítica de todo negocio. Sin embargo aún existen problemas en el desarrollo de software como son la calidad insuficiente o muy variable y los excesos de tiempo en el desarrollo y los costos. Para resolver estos problemas una de las formas por la que se ha optado es mejorar el proceso de desarrollo, o sea la forma en que deben construirse el software, lo que equivale a mejorar la metodología que se sigue para construir los productos.

A lo largo de los años han surgido gran cantidad de enfoques metodológicos pero muchas veces el contexto del proyecto es menos predecible o estable que lo deseado.

Durante el transcurso de los años 90 varias personas se dieron cuenta de que las cosas estaban cambiando. Estas personas observaron la necesidad de desarrollar metodologías livianas y maniobrables que pudieran operar en un ambiente turbulento. Estas metodologías son conocidas hoy en día como “Metodologías ágiles”.

1.1 Definición de una metodología ágil.

1.1.1 Historia

En febrero del 2001, tras una reunión celebrada en Utah Estados Unidos, nace el término “ágil” aplicados al desarrollo de software. En esta reunión participan un grupo de expertos de la industria del software, incluyendo alguno de los creadores o impulsores de las metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Tras esta reunión se creó la Alianza Ágil [1], una organización dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil [2], un documento que resume la filosofía ágil. A continuación se expondrán las ideas más significativas del mismo.

1.1.2 Manifiesto Ágil

- **Propósito:** Descubrir mejores maneras de desarrollar software haciendo y ayudando a otros a hacerlo.

Valorando a través del mismo a:

- Los individuos e interacciones por sobre los procesos y herramientas.
- El software activo por sobre la documentación comprensiva, integral.
- La colaboración del cliente por sobre la negociación del contrato.
- La respuesta al cambio por sobre el seguimiento de un plan.

- **Principios**

1- Nuestra prioridad más alta es satisfacer al cliente a través de la entrega temprana y continua de software valioso. [3]

La implementación del principio de "valor del cliente" es una de esas actividades que son "más fácil de decir que de hacer". Las prácticas de dirección de proyecto tradicionales asumen que lograr un plan equivale al éxito del proyecto, equivale al valor de cliente demostrado. La volatilidad asociada con los proyectos de hoy en día, demanda que el valor del cliente frecuentemente se reevalúe, y que la satisfacción de los planes del proyecto originales puede no tener mucho que ver con el éxito final de un proyecto.

2- Acogemos los requerimientos cambiantes, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan los cambios en pro de la ventaja competitiva del cliente. [4]

La imprevisibilidad creciente del futuro es uno de los aspectos más desafiantes de la nueva economía. La turbulencia - en ambos, negocio y tecnología - provoca el cambio, puede verse como una amenaza de la cual cuidarse o como una oportunidad a ser abrazada.

En lugar de resistirse al cambio, el enfoque ágil se esfuerza por acomodarlo tan fácil y eficazmente como sea posible, mientras se mantiene conciencia de sus consecuencias. Aunque la mayoría de las

personas están de acuerdo que la retroalimentación es importante, ignoran a menudo el hecho de que el resultado de una retroalimentación aceptada, es el cambio. Las metodologías ágiles aprovechan este resultado, porque sus defensores entienden que facilitar el cambio es más eficaz que intentar prevenirlo.

3- Entregar software activo frecuentemente, desde un par de semanas a un par de meses, con preferencia a las escalas de tiempo más cortas. [5]

Durante muchos años, los gurús de procesos han estado diciéndonos a todos de usar un estilo incremental o reiterativo de desarrollo del software, con entregas múltiples de funcionalidad siempre creciente. Mientras esta práctica ha crecido, todavía no es predominante; sin embargo, es esencial para los proyectos ágiles. Además, que se presiona bastante para reducir el tiempo del ciclo de la entrega.

Sin embargo, recordar que la entrega no es lo mismo que dar una nueva edición (o "release") La gente de negocio puede tener razones válidas para no poner código en producción cada par de semanas.

4- La gente de negocio y los diseñadores trabajan juntos diariamente a lo largo del proyecto. [6]

Para comenzar, no se espera un conjunto detallado de requisitos que sea visado al principio del proyecto; más bien, se tiene una visión de alto nivel de los requisitos que está sujeta a cambios frecuentes. Claramente, esto no es suficiente para diseñar y codificar, de manera que el hueco se cierra con la interacción frecuente entre las personas de negocio y los desarrolladores. La frecuencia de este contacto sorprende a menudo a las personas. Se pone "diariamente" en el principio para dar énfasis al compromiso continuo del cliente de tomar parte activa, y de hecho, tomar responsabilidad en conjunto para el proyecto del software.

5- Construir proyectos alrededor de individuos motivados, deles el ambiente, satisfaga sus necesidades y confíe en ellos en conseguir el trabajo hecho. [7]

Desplegar todas las herramientas, tecnologías y procesos que guste, incluso los procesos ágiles, pero al final, son las personas las que representan la diferencia entre el éxito y fracaso. Se debe comprender que a pesar de lo duro que se trabaje proponiendo ideas del proceso, lo más que se puede

esperar es un efecto de segundo orden en un proyecto. Así que es importante maximizar ese factor de personas de primero orden.

Para muchas personas, la confianza es lo más difícil de entregar. Las decisiones deben ser tomadas por las personas que conocen más sobre la situación. Esto significa que los gerentes deben confiar en su personal para tomar las decisiones sobre las cosas que ellos pagan para conocer.

6- El método más eficaz y efectivo de llevar la información desde y hacia un equipo de desarrollo es la conversación cara a cara. [8]

Inevitablemente, al discutir las metodologías ágiles, surge el tema de la documentación. Los opositores parecen insultados a veces, mientras se burlan de la falta de documentación. Es suficiente para gritar, “¡el asunto no es la documentación - el asunto es la comprensión!” Sí, la documentación física tiene peso y sustancia, pero la medida real de éxito es abstracta: ¿las personas involucradas ganarán el entendimiento que ellos necesitan? Se usa porque tiene que hacerse, pero la mayoría de equipos de proyecto pueden y deberían usar técnicas de comunicación más directas que la escritura.

De manera que la distinción entre metodologías ágiles y documento céntrico no es la de documentación extensa contra ninguna documentación; sino más bien un concepto diferente de la mezcla de documentación y conversación requerida para facilitar el entendimiento.

7- El software activo es la medida primaria de progreso. [9]

Muy a menudo, se ha visto equipos de proyectos que no comprenden que ellos están en problemas sino hasta un tiempo corto antes de la entrega. Ellos hicieron los requisitos a tiempo, el plan a tiempo, quizá incluso el código a tiempo, pero las pruebas y la integración tomó más tiempo de lo que ellos pensaron. Aquí se favorece el desarrollo reiterativo principalmente porque proporciona hitos que no pueden ser diluidos y que imparten una medida exacta del progreso y un entendimiento más profundo de los riesgos involucrados en cualquier proyecto dado.

“El software activo es la medida de progreso porque no hay ninguna otra manera de capturar las sutilezas de los requisitos: Los documentos y diagramas son demasiado abstractos para permitir al usuario darse cuenta del avance” [10]

8- Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un paso constante de manera indefinida. [11]

La industria se caracteriza por noches y fines de semana largos, durante los cuales las personas intentan deshacer los errores de una planificación inerte. Irónicamente, estas largas horas realmente no llevan a una productividad mayor.

La agilidad se basa en personas que están alertas y son creativas, y que puedan mantener esa vigilancia y creatividad a lo largo de todo el proyecto de desarrollo de software. El desarrollo sostenible significa hallar un ritmo de trabajo (40 o algo así, de horas por semana) que el equipo pueda sostener con el tiempo permaneciendo saludable.

9- La atención continua a la excelencia técnica y el buen diseño refuerza la agilidad. [12]

Cuando muchas personas miran al desarrollo ágil, ellos ven recordatorios de los esfuerzos del RAD "rápido y sucio" (DAR: Desarrollo de Aplicación Rápido- RAD por sus siglas en inglés.) de la última década. Pero, mientras el desarrollo ágil es similar al RAD por lo que se refiere a la velocidad y flexibilidad, hay una diferencia grande cuando se refiere a la limpieza técnica. Los enfoques ágiles dan énfasis a la calidad del diseño, porque la calidad del diseño es esencial para mantener la agilidad.

Uno de los aspectos de cuidado, sin embargo, es el hecho que los procesos ágiles asumen y animan la alteración de los requisitos mientras el código está siendo escrito. Como tal, el diseño no puede ser una actividad inicial completamente terminada antes de la construcción. En cambio, el diseño es una actividad continua que se realiza a lo largo del proyecto. Todas y cada una de las iteraciones tendrán un trabajo de diseño.

10- El arte de la simplicidad - el arte de maximizar la cantidad de trabajo no hecho - es esencial. [13]

Cualquier tarea de desarrollo de software puede ser abordada con una multitud de métodos. En un proyecto ágil, es particularmente importante usar enfoques simples, porque ellos son más fáciles de cambiar. Es más fácil agregar algo a un proceso que es demasiado simple que sacarle algo a un proceso que es demasiado complejo. Hay un sabor fuerte de minimalismo en todos los métodos ágiles. Incluyen solamente lo que todos necesitamos en lugar de lo que alguno necesite, para hacerle más fácil a los equipos agregar algo que satisfaga sus propias necesidades particulares.

11- Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto organizados. [14]

Contrariamente a lo que se ha oído, la forma no sigue a la función: La forma sigue al fracaso. "La forma de cosas hechas siempre está sujeta al cambio en respuesta a sus deficiencias o fracasos reales o percibidos, para poder funcionar apropiadamente" [15]

Las visiones de Petroski son similares a uno de los dos puntos claves de este principio - que los mejores diseños (arquitecturas, requerimientos) surgen del desarrollo y uso iterativo en lugar de los planes previos. El segundo punto del principio es que las propiedades emergentes (emergencia, una propiedad clave de sistemas complejos, dificultosamente se traduce a innovación y creatividad en las organizaciones humanas) se generan mejor en equipos auto organizados en que las interacciones son altas y las reglas del proceso son pocas.

12- A intervalos regulares, el equipo reflexiona en cómo volverse más eficaz, entonces afina y ajusta su conducta consecuentemente. [16]

Los métodos ágiles no son algo que se escoge y sigue servilmente. Puede empezar con uno de estos procesos, pero se reconoce que no se puede proponer el proceso que será correcto para toda situación. Así que cualquier equipo ágil debe refinar y reflexionar conforme avanza, mientras mejora constantemente sus prácticas en las circunstancias locales.

1.1.3 ¿Qué son las metodologías Ágiles?

Las Metodologías Ágiles o “ligeras” constituyen un nuevo enfoque en el desarrollo de software, mejor aceptado por los desarrolladores de e-projects (proyectos) que las metodologías convencionales (ISO-9000, CMM, etc.) debido a la simplicidad de sus reglas y prácticas, su orientación a equipos de desarrollo de pequeño tamaño, su flexibilidad ante los cambios y su ideología de colaboración [17]

Las metodologías ágiles se entienden como un conjunto de metodologías para desarrollo de software surgidas en la década de los 90.

Lo ágil se define (por los mismos agilistas) como la habilidad de responder de forma versátil al cambio para maximizar los beneficios.

Los valores y principios compartidos por toda metodología ágil fueron enunciados en el Manifiesto Ágil por la Alianza Ágil (antes mencionados).

1.2 Características de las metodologías ágiles.

De manera general, las metodologías ágiles pueden explicarse a través de los siguientes 4 principios fundamentales que son a su vez los propósitos del Manifiesto Ágil.

1. Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. Dado que el proceso de desarrollo es creativo, no es posible pensar que las personas funcionen respondiendo a órdenes o procesos rígidos.
2. Desarrollar software que funciona más que conseguir una buena documentación. Puesto que si el software no funciona la documentación no vale de nada. A nivel interno puede haber documentación, pero solo la necesaria y a nivel externo lo que el cliente requiera.
3. La colaboración con el cliente más que la negociación de un contrato. Supone que la satisfacción del cliente con el producto será mayor, mientras exista una conversación y realimentación continua entre este y la empresa.
4. Responder a los cambios más que seguir estrictamente un plan. Puesto que si un proyecto de software no es capaz de adaptarse a los cambios fracasara, especialmente en productos de gran

envergadura. La estrategia de planificación se basa en: planes detallados para las próximas semanas, planes aproximados para los próximos meses y muy generales para plazos mayores.

A nivel mundial existen varias metodologías para el desarrollo de software que encajan bajo el estandarte de ágil. Mientras todas ellas comparten muchas características, también hay algunas diferencias significativas. Dentro de las más conocidas podemos citar:

- XP (Extreme Programming – Kent Beck)
- SCRUM (Jeff Sutherland, Ken Schwaber),
- Crystal (Alistair Cockburn)
- DSDM (DSDM Consortium)
- Feature Driven Development (Jeff deLuca, Peter Coad, Together Soft),
- Adaptive Software Development (Jim Highsmith),
- Lean Development (Mary y TomPoppendieck, Robert Charette)

Estas metodologías serán caracterizadas a continuación teniendo en cuenta los siguientes parámetros:

1- Características Principales.
2- Roles.
3- Ciclo de desarrollo o proceso.
4- Prácticas.

1.2.1 XP (Extreme Programming – Kent Beck) (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. [18]

Roles

- *Programador*. El programador escribe las pruebas unitarias y produce el código del sistema.

- *Cliente*. Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

- *Encargado de pruebas (Tester).* 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*). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

- *Entrenador (Coach).* Es responsable del proceso global. Debe proveer guías al 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, en el que puedan surgir problemas.

- *Gestor (Big boss)*. Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje.

Proceso XP

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.

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

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. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

- El juego de la planificación.
- Entregas pequeñas.
- Metáfora.
- Diseño simple.
- Pruebas.
- Refactorización (Refactoring).
- Programación en parejas.
- Propiedad colectiva.
- Cliente in-situ.
- Estándares de programación.

Características de los proyectos:

- Requisitos imprecisos y muy cambiantes

- Existencia de un alto riesgo técnico

1.2.2 SCRUM (Jeff Sutherland, Ken Schwaber)

SCRUM define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Características

- Equipos auto dirigidos
- Utiliza reglas para crear un entorno ágil de administración de proyectos
- No prescribe prácticas específicas de ingeniería
- Los requerimientos se capturan como ítems de la lista reserva del producto
- El producto se construye en una serie de Sprints de un mes de duración.

Roles

Scrum Máster (Jefe del Equipo): Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que todo funciona según lo planeado. Su principal trabajo es remover impedimentos y reducir riesgos del producto. Este rol suele ser desempeñado por un Gerente de Proyecto o Líder de equipo.

Product Owner (Usuario interno o gerente): Es el responsable del proyecto, administra, controla y comunica la Backlog List. (Lista de reserva) Es el responsable de encontrar la visión del producto y reflejarla en la Backlog List.

Scrum Team (Equipo): Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de un Sprint. Sus tareas son: Effort Estimation (Estimar

Esfuerzo), crear el Sprint Backlog (reserva de la carrera), revisar la Product Backlog List (Lista de reserva del producto) y sugerir obstáculos que deban ser removidos para cumplir con los items que aparecen. Típicamente es un equipo de entre 5 y 10 personas cada una especializada en algún elemento que conforma los objetivos a cumplir, por ejemplo: Programadores, Diseñadores de Interfaz de usuario, etc. La dedicación de los miembros del equipo debería ser full-time con algunas excepciones. La membrecía solo puede cambiar entre sprints (no durante).

Customer (Cliente): El cliente participa en las tareas que involucran la lista Product Backlog.

Management (Gerente): Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el proyecto. Participa en la selección de objetivos y requerimientos y en la selección del Scrum Owner. Tiene la responsabilidad de controlar el progreso y trabaja junto con el Scrum Master en la reducción de la Product Backlog List.

Proceso

Scrum consta de tres fases:

1- Pregame (antes del juego)

- Planning (Planeamiento)

Consiste en la definición del sistema que será construido. Para esto se crea la lista Product Backlog a partir del conocimiento que actualmente se tiene del sistema. En ella se expresan los requerimientos priorizados y a partir de ella se estima el esfuerzo requerido. La Product Backlog List es actualizada constantemente con ítems nuevos y más detallados, con estimaciones más precisas y cambios en la prioridad de los ítems.

- Architecture / High level Design (Arquitectura /diseño de alto nivel)

El diseño de alto nivel del sistema se planifica a partir de los elementos existentes en la Product Backlog List. En caso de que el producto a construir sea una mejora a un sistema ya existente, se identifican los cambios necesarios para implementar los elementos que aparecen en la lista Product

Backlog y el impacto que pueden tener estos cambios. Se sostiene una Design Review Meeting para examinar los objetivos de la implementación y tomar decisiones a partir de la revisión. Se preparan planes preliminares sobre el contenido de cada release.

2- Development.(Desarrollo)

En esta fase se espera que ocurran cosas impredecibles. Para evitar el caos Scrum define prácticas para observar y controlar las variables técnicas y del entorno, así también como la metodología de desarrollo que hayan sido identificadas y puedan cambiar. Este control se realiza durante los Sprints. Dentro de variables de entorno encontramos: tiempo, calidad, requerimientos, recursos, tecnologías y herramientas de implementación. En lugar de tenerlas en consideración al comienzo del desarrollo, Scrum propone controlarlas constantemente para poder adaptarse a los cambios en forma flexible.

3- Postgame.(después del juego)

Contiene el cierre del release. Para ingresar a esta fase se debe llegar a un acuerdo respecto a las variables del entorno por ejemplo que los requerimientos fueron completados. El sistema está listo para ser liberado y es en esta etapa en la que se realiza integración, pruebas del sistema y documentación.

Prácticas

- Sprints (carrera)
- Sprint Planning Meeting (Reunión de planificación del Sprint)
- Daily Meetings (Reuniones diarias)
- Sprint Review Meeting (Reuniones de revision del Sprint)
- Design Review Meeting (Reuniones de revisión del diseño)
- Stabilization Sprints (Estabilización del Sprints)
- Meta Scrums.

Características de los proyectos

- Rápido cambio de requisitos.

- No mayor de 10 ingenieros (ideal)(con más se forman equipos)

1.2.3 Crystal (Alistair Cockburn)

Crystal Methodologies se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (Amarillo) (3 a 8 miembros) y Crystal Orange (Naranja) (25 a 50 miembros).

Características:

a) La cantidad de detalle necesario en la documentación de los requerimientos, diseño y planeamiento varía según las circunstancias del proyecto.

b) Quizás no sea posible eliminar todos los work products (productos del trabajo) y las notas promisorias inmediatas tales como los documentos de requerimientos, diseño y los planes para el proyecto, pero pueden reducirse a una comunicación corta, enriquecedora, e informal entre el equipo.

c) El equipo ajusta continuamente sus convenciones de trabajo para adaptarse a:

- Las personalidades particulares del equipo
- El ambiente local de trabajo actual
- Las particularidades de las asignaciones específicas

Roles

• Sponsor (patrocinador):

- ✓ Manifestación de los objetivos con prioridades de tradeoff (compensación)

• El Team (Equipo) como un grupo:

- ✓ Estructura del equipo y convenciones
- ✓ Resultados del Reflection Workshop (Taller de la reflexión)

• Coordinador:

- ✓ Mapa del Proyecto
- ✓ Release Plan
- ✓ Estado del Proyecto
- ✓ Lista de Riesgos
- ✓ Plan de Iteración & Status (estados)
- ✓ Viewing Schedule (horario de la visión)

• Experto en el Negocio y el Usuario Embajador

- ✓ Lista de Actores Objetivo
- ✓ Los Casos de Uso & el Archivo de Requerimientos

• Usuario Embajador

- ✓ El Perfil del Rol del Usuario

• Lead Designer

- ✓ La Descripción de la Arquitectura.

Proceso

La familia Crystal dispone un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más “pesado” es el método. Cuanto más crítico es un sistema, más rigor se requiere. El código cromático se aplica a una forma tabular elaborada por Cockburn que se usa en muchos métodos ágiles para situar el rango de complejidad al cual se aplica una metodología.

Prácticas

- Entregas Frecuentes
- Comunicación Osmótica
- Mejoras Reflexivas
- Seguridad Personal
- Enfoque
- Fácil acceso a los usuarios expertos
- Ambiente Técnico

Características de los proyectos:

- Amarillo, para 8 a 20 ingenieros; Naranja, para 20 a 50 ingenieros; Rojo, para 50 a 100 ingenieros

1.2.4 DSDM (DSDM Consortium) (Método dinámico del desarrollo de los sistemas)

· **Dynamic Systems Development Method (DSDM)** Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases.

Características:

- Must have (Debe tener): Son los requerimientos fundamentales del sistema. De éstos, el subconjunto mínimo ha de ser satisfecho por completo.
- Should have (Debería tener): Son requerimientos importantes para los que habrá una resolución en el corto plazo.
- Could have(Podría tener): Podrían quedar fuera del sistema si no hay más remedio.

- Want to have but won't have this time around (Se desea que tenga, pero no lo tendrá esta vuelta): Son requerimientos valorados, pero pueden esperar.

Roles

Define quince roles, algo más que el promedio de los métodos ágiles. Los más importantes son: Programadores y Programadores Senior. Son los únicos roles de desarrollo. El título de Senior indica también nivel de liderazgo dentro del equipo. Equivale a Nivel 3 de Cockburn. Ambos títulos cubren todos los roles de desarrollo, incluyendo analistas, diseñadores, programadores y verificadores.

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

Prácticas

1. Es imperativo el compromiso activo del usuario.
2. Los equipos de DSDM deben tener el poder de tomar decisiones.
3. El foco radica en la frecuente entrega de productos.

4. El criterio esencial para la aceptación de los entregables es la adecuación a los propósitos de negocios.
5. Se requiere desarrollo iterativo e incremental.
6. Todos los cambios durante el desarrollo son reversibles.
7. La línea de base de los requerimientos es de alto nivel. Esto permite que los requerimientos de detalle se cambien según se necesite y que los esenciales se capten tempranamente.
8. La prueba está integrada a través de todo el ciclo de vida. La prueba también es incremental. Se recomienda particularmente la prueba de regresión, de acuerdo con el estilo evolutivo de desarrollo.
9. Es esencial una estrategia colaborativa y cooperativa entre todos los participantes. Las responsabilidades son compartidas y la colaboración entre usuario y desarrolladores no debe tener fisuras.

Características de los proyectos:

- Desarrollo de soluciones de negocios sujetas a márgenes de tiempo estrechos.

1.2.5 Adaptive Software Development (Jim Highsmith) (Desarrollo adaptante Del software)

· **Adaptive Software Development (ASD)**. Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

ASD presupone que las necesidades del cliente son siempre cambiantes. La iniciación de un proyecto involucra definir una misión para él, determinar las características y las fechas y descomponer el proyecto en una serie de pasos individuales, cada uno de los cuales puede abarcar entre cuatro y ocho semanas. Los pasos iniciales deben verificar el alcance del proyecto; los tardíos tienen que ver con el diseño de una arquitectura, la construcción del código, la ejecución de las pruebas finales y el despliegue.

Aspectos claves de ASD son:

1. Un conjunto no estándar de “artefactos de misión” (documentos para ti y para mí), incluyendo una visión del proyecto, una hoja de datos, un perfil de misión del producto y un esquema de su especificación.
2. Un ciclo de vida, inherentemente iterativo.
3. Cajas de tiempo, con ciclos cortos de entrega orientados por riesgo.

Un ciclo de vida es una iteración; este ciclo se basa en componentes y no en tareas, es limitado en el tiempo, orientado por riesgos y tolerante al cambio. Que se base en componentes implica concentrarse en el desarrollo de software que trabaje, construyendo el sistema pieza por pieza. En este paradigma, el cambio es bienvenido y necesario, pues se concibe como la oportunidad de aprender y ganar así una ventaja competitiva; de ningún modo es algo que pueda ir en detrimento del proceso y sus resultados.

ASD se concentra más en los componentes que en las tareas; en la práctica, esto se traduce en ocuparse más de la calidad que en los procesos usados para producir un resultado. En los ciclos adaptativos de la fase de Colaboración, el planeamiento es parte del proceso iterativo, y las definiciones de los componentes se refinan continuamente. La base para los ciclos posteriores (el bucle de Aprendizaje) se obtiene a través de repetidas revisiones de calidad con presencia del cliente como experto, constituyendo un grupo de foco de cliente. Esto ocurre solamente al final de las fases, por lo que la presencia del cliente se suplementa con sesiones de desarrollo conjunto de aplicaciones (JAD).

Características de los proyectos:

- ✓ Proyectos de complejidad creciente.
- ✓ Proyectos apenas un poco fuera del caos.

1.2.6 Feature Driven Development (Jeff de Luca, Peter Coad, Together Soft) (Característica - desarrollo conducido)

· **Feature Driven Development (FDD)**. Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software.

Los principios de FDD son pocos y simples:

- Se requiere un sistema para construir sistemas si se pretende escalar a proyectos grandes.
- Un proceso simple y bien definido trabaja mejor.
- Los pasos de un proceso deben ser lógicos y su mérito inmediatamente obvio para cada miembro del equipo.
- Vanagloriarse del proceso puede impedir el trabajo real.
- Los buenos procesos van hasta el fondo del asunto, de modo que los miembros del equipo se puedan concentrar en los resultados.
- Los ciclos cortos, iterativos, orientados por rasgos (features) son mejores.

Hay tres categorías de rol en FDD: roles claves, roles de soporte y roles adicionales. Los seis roles claves de un proyecto son: (1) administrador del proyecto, quien tiene la última palabra en materia de visión, cronograma y asignación del personal; (2) arquitecto jefe (puede dividirse en arquitecto de dominio y arquitecto técnico); (3) manager de desarrollo, que puede combinarse con arquitecto jefe o manager de proyecto; (4) programador jefe, que participa en el análisis del requerimiento y selecciona rasgos del conjunto a desarrollar en la siguiente iteración; (5) propietarios de clases, que trabajan bajo la guía del programador jefe en diseño, codificación, prueba y documentación, repartidos por rasgos y (6) experto de dominio, que puede ser un cliente, patrocinador, analista de negocios o una mezcla de todo eso.

Los cinco roles de soporte comprenden (1) administrador de entrega, que controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él; reporta al manager del proyecto; (2) abogado/gurú de lenguaje, que conoce a la perfección el lenguaje y la tecnología; (3) ingeniero de construcción, que se encarga del control de versiones de los builds y publica la documentación; (4) herramientista (toolsmith), que construye herramientas ad hoc o mantiene bases de datos y sitios Web y (5) administrador del sistema, que controla el ambiente de trabajo o productiza el sistema cuando se lo entrega.

Los tres roles adicionales son los de verificadores, encargados del despliegue y escritores técnicos. Un miembro de un equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas.

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema. La parte iterativa soporta desarrollo ágil con rápidas adaptaciones a cambios en requerimientos y necesidades del negocio. Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida. Típicamente, la iteración de un rasgo insume de una a tres semanas. Las fases son:

1) **Desarrollo de un modelo general.** Cuando comienza este desarrollo, los expertos de dominio ya están al tanto de la visión, el contexto y los requerimientos del sistema a construir. A esta altura se espera que existan requerimientos tales como casos de uso o especificaciones funcionales. FDD, sin embargo, no cubre este aspecto. Los expertos de dominio presentan un ensayo en el que los miembros del equipo y el arquitecto principal se informan de la descripción de alto nivel del sistema. El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego de cada ensayo, un equipo de desarrollo trabaja en pequeños grupos para producir modelos de objeto de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema.
2) **Construcción de la lista de rasgos.** Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos. Los rasgos son pequeños ítems útiles a los ojos del cliente. Son similares a las tarjetas de historias de XP y se escriben en un lenguaje que todas las partes puedan entender. Las funciones se agrupan conforme a diversas actividades en áreas de dominio específicas. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad. Los rasgos que requieran más de diez días se descomponen en otros más pequeños.
3) **Planeamiento por rasgo.** Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia, y se asigna a los programadores jefes. Las listas se priorizan en secciones que se llaman paquetes de diseño. Luego se asignan las clases definidas en la selección del modelo general a programadores individuales, o sea propietarios de clases. Se pone fecha para los conjuntos de rasgos.

4) **Diseño por rasgo y Construcción por rasgo.** Se selecciona un pequeño conjunto de rasgos del conjunto y los propietarios de clases seleccionan los correspondientes equipos dispuestos por rasgos. Se procede luego iterativamente hasta que se producen los rasgos seleccionados. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. Puede haber varios grupos trabajando en paralelo. El proceso iterativo incluye inspección de diseño, codificación, prueba de unidad, integración e inspección de código. Luego de una iteración exitosa, los rasgos completos se promueven al build principal. Este proceso puede demorar una o dos semanas en implementarse.

FDD consiste en un conjunto de "mejores prácticas" que distan de ser nuevas pero se combinan de manera original. Las prácticas canónicas son:

- Modelado de objetos del dominio, resultante en un framework (marco) cuando se agregan los rasgos. Esta forma de modelado descompone un problema mayor en otros menores; el diseño y la implementación de cada clase u objeto es un problema pequeño a resolver. Cuando se combinan las clases completas, constituyen la solución al problema mayor. Una forma particular de la técnica es el modelado en colores [CLD00], que agrega una dimensión adicional de visualización. Si bien se puede modelar en blanco y negro, en FDD el modelado basado en objetos es imperativo.
- Desarrollo por rasgo. Hacer simplemente que las clases y objetos funcionen no refleja lo que el cliente pide. El seguimiento del progreso se realiza mediante examen de pequeñas funcionalidades descompuestas y funciones valoradas por el cliente. Un rasgo en FDD es una función pequeña expresada en la forma <acción> <resultado> <por | para | de | a> <objeto> con los operadores adecuados entre los términos. Por ejemplo, calcular el importe total de una venta; determinar la última operación de un cajero; validar la contraseña de un usuario.
- Propiedad individual de clases (código). Cada clase tiene una sola persona nominada como responsable por su consistencia, performance e integridad conceptual.
- Equipos de Rasgos, pequeños y dinámicamente formados. La existencia de un equipo garantiza que un conjunto de mentes se apliquen a cada decisión y se tomen en cuenta múltiples alternativas.
- Inspección. Se refiere al uso de los mejores mecanismos de detección conocidos. FDD es tan escrupuloso en materia de inspección como lo es Evo.

- Builds regulares. Siempre se tiene un sistema disponible. Los builds forman la base a partir de la cual se van agregando nuevos rasgos.
- Administración de configuración. Permite realizar seguimiento histórico de las últimas versiones completas de código fuente.
- Reporte de progreso. Se comunica a todos los niveles organizacionales necesarios.

Características de los proyectos:

- ✓ Proyectos mayores.
- ✓ Proyectos de misión crítica.

1.2.7 Lean Development (Mary y Tom Poppendieck, Robert Charette) (Desarrollo magro)

· **Lean Development**. Los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

1. Satisfacer al cliente es la máxima prioridad.
2. Proporcionar siempre el mejor valor por la inversión.
3. El éxito depende de la activa participación del cliente.
4. Cada proyecto LD es un esfuerzo de equipo.
5. Todo se puede cambiar.
6. Soluciones de dominio, no puntos.
7. Completar, no construir.
8. Una solución al 80% hoy, en vez de una al 100% mañana.
9. El minimalismo es esencial.
10. La necesidad determina la tecnología.
11. El crecimiento del producto es el incremento de sus prestaciones, no de su tamaño.
12. Nunca empujes LD más allá de sus límites.

Dado que LD es más una filosofía de management (gerencia) que un proceso de desarrollo no hay mucho que decir del tamaño del equipo, la duración de las iteraciones, los roles o la naturaleza de sus etapas.

1.3 Panorámica nacional y marco de desarrollo en la Universidad de las Ciencias Informáticas (UCI).

En Cuba no existió cultura de industria en las empresas de software hasta hace no más de un década, lo que trae consigo que no se fabricara software comercial sino artesanal por completo. Antes del año 2000 quienes utilizaban metodologías en Cuba eran los centros adscriptos a universidades o las propias universidades como la CUJAE que utilizaban sus metodologías: ADESA, ADOOSI para sistemas de gestión estructurados u Orientado a Objeto, y MultiMet para sistemas hipermedia en general, no otras.

Fue solo a partir del surgimiento de la metodología RUP (que impactó en el mundo) que se comenzó con la diversificación en ese sentido. Contribuyó a ello el interés político del país de pasar de artesanos a industriales en la producción de software lo que hizo que nacionalmente se comenzaran a interesar por este tema todos aquellos que producían software.

Por tales causas se puede asegurar que las empresas adscriptas al MIC (Ministerio de la Informática y las Comunicaciones) lo que más utilizan es RUP, aunque con ciertas modificaciones para contextualizarlo a sus intereses. Además, en las universidades, especialmente en la Universidad de las Ciencias Informáticas (UCI), la formación docente en la materia que incluye a las metodologías de desarrollo, están sustentadas sobre la base de la metodología RUP (Rational Unified Process, en español Proceso Unificado de Rational) por ser la más completa, factible, robusta y la mas conocida de todas.

En nuestro país de las metodologías analizadas anteriormente la más difundida es XP por ser precisamente la más reconocida, lo que no significa que sea la mejor. Esto nos da la medida de la poca usabilidad y conocimientos de los procesos ágiles, aunque otras empresas de desarrollo utilizan metodologías ligeras o ágiles, sin determinarse un por ciento de aceptación de las mismas.

Después de haberse caracterizado cada una de las metodologías, así como los proyectos para los cuales cada una es más factible, se ve la necesidad de la utilización de procedimientos ágiles en

proyectos que se adecuen a las características imperantes. El uso de procedimientos ágiles no es para todos pues depende da varios factores y hay que tener en cuenta varias cosas que las caracterizan; pero una cosa es cierta, son extensamente aplicables y deben ser usadas por mas personas de las que actualmente la tienen en cuenta.

En este capítulo se realizo un estudio de las metodologías ágiles más conocidas en el mundo, abordando las características principales de las mismas y describiendo brevemente sus procesos, además de que se abordó la situación nacional con relaciona este tema que es sin dudas reciente.

CAPÍTULO 2 ESTUDIO DEL PROCESO PRODUCTIVO EN LA FACULTAD 7

Hacer un estudio de los proyectos de desarrollo es un eslabón esencia a la hora de revolucionar los mismos; hay que identificar problemas y posibles causas y determinar aspectos del proceso de desarrollo pueden ser cambiado para viabilizar y rentabilizar dicho proceso.

2.1 Clasificación de los Proyectos.

En el libro Ingeniería de Software un enfoque práctico de Roger Presuman se hace una caracterización genenérica de los proyectos teniendo en cuenta el tipo de software que desarrollan, siendo estas clasificaciones las siguientes:

Software de Sistema: Es un conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistema (por ejemplo: compiladores, editores y utilidades de gestión de archivos.) procesan estructuras de información complejas pero determinadas. Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados. En cualquier caso, el área del software de sistema se caracteriza por una fuerte interacción con el hardware de la computadora; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; una estructura de datos complejas y múltiples interfaces externas.

Software de tiempo real: El software que coordina/analiza/controla sucesos del mundo real conforme ocurre, se denomina de tiempo real. Entre los elementos del software de tiempo real se incluye: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo, y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse la respuesta en tiempo real (típicamente en el rango de un milisegundo a un segundo)

Software de gestión: El proceso de la información comercial constituye la mayor de las áreas de desarrollo de aplicación del software. Los sistemas discretos (por ejemplo: nominas cuentas de haberes-débitos, inventarios, etc.) han evolucionado hacia el software de sistema de información de gestión (SIG)

que accede a una o más bases de datos que contienen información comercial. Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, las aplicaciones da software de gestión también realizan cálculo iterativo (por ejemplo: el procesamiento de transacciones en puntos de ventas.)

Software de ingeniería y científico: Esta caracterizado por los algoritmos de manejo de números: las aplicaciones van desde la astronomía hasta la vulcanología, desde el análisis de la presión de los automotores y dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación de automática: Sin embargo, las nuevas aplicaciones del área de ingeniería/ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadoras, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a coger características del software de tiempo real e incluso de software de sistema.

Software empotrado: reside en memoria de solo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. Puede ejecutar funciones muy limitadas y curiosas (el control de las teclas de un horno de microondas) o suministrar una función significativa y con capacidad de control (por ejemplo funciones digitales en un automóvil, tales como de la gasolina, indicadores en el salpicadero, sistemas de frenado. etc.).

Software de computadoras personales: El procesamiento de textos, las hojas de cálculo, los gráficos por computadoras, multimedia, entretenimientos, gestión de base de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas son algunas de los cientos de aplicaciones.

Software basado en Web: las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, CGI, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales). En la esencia, la red viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquiera con un modem.

Software de inteligencia artificial (IA): Hace uso de algoritmos no numéricos para resolver problemas complejos para os que no son adecuados el calculo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en conocimientos, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas, y los juegos son representativos de las aplicaciones de esta categoría.

2.2 Características de los proyectos de la Facultad 7

Los proyecto de desarrollo de software puede ser caracterizado teniendo en cuenta disimiles puntos que sirven de guía para clasificarlos.

En la Facultad 7 de la Universidad de las Ciencias Informáticas la organización de los proyectos está dada por la agrupación de los mismos en áreas temáticas, las mismas serán expuestas a continuación.

Áreas temáticas de la facultad 7	**Proyectos que se desarrollan**
Planificación y control	Estadística
	Base material
	Docencia
	Colaboración médica
	ARS
Sistema especializado	Fisioterapia
	SIUM
	Nefrología
Sistema de gestión hospitalaria	Hospitales
	LIS
	FMS
Sistema de atención primaria	APS
Imágenes y señales	PACS
	DIANA
Calidad	Calidad de software

La recogida de los datos (teniendo en cuenta importantes parámetros que son la base de una posterior clasificación y agrupación de los proyectos, posibilitando un mejor estudio y análisis de los mismos) que caracterizan cada uno de los proyectos fue posible a través de entrevistas realizadas a los líderes de proyectos.

Los tipos de entrevistas son tres: la estructurada que sigue un orden en las preguntas que ya han sido predefinidas y sigue este guión, ejemplo una entrevista a un político o a un científico. La no estructurada en las que se cambia en parte el orden, incluso hasta las preguntas según como se va desarrollando el evento, ejemplo una entrevista a un artista, a un deportista. Por último está el tipo mixto donde no hay ningún guión ni en el orden ni en el contenido de las preguntas y se va por donde el diálogo lo lleve y es informal, ejemplo un presentador de televisión a otro presentador de televisión. Claro está, en todos los casos se conserva el sentido u objeto de la entrevista.

Para llevar a cabo dicha entrevista (definida estructurada) que tenía como objetivo principal la facilitación del estudio y análisis de los proyectos de la facultad mediante la recopilación de los elementos necesarios, fue preciso la elaboración de un guión de la entrevista (ver anexo 1) donde se tuvieran en cuenta todos los aspectos importantes que podían ser olvidados, así como para tener y definir un orden en el proceso de recogida de datos.

De ahí que se realizara la caracterización de los mismos por cada uno de los parámetros, lo cual queda expuesto a continuación:

Estadística

- Tipo de proyecto: Gestión.
- Cantidad de integrantes: 20 personas.
- Roles: analista, jefe de proyecto, diseñador de interfaz, diseñador de base de Datos, desarrollador documentador técnico.
- Entorno de desarrollo: LAMP (Linux en el servidor, PHP, Apache, MySQL).
- Envergadura: Medio.
- Tiempo de duración estimado: Período corto de desarrollo.
- Complejidad: media.

- Organización: Por modulo.
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

  - Poca preparación de los estudiantes, poco conocimiento de las herramientas y estándares.
  - Falta de organización en la planificación de tareas y definición de responsabilidades.

Base material

- Tipo de proyecto: Gestión.
- Cantidad de integrantes: 15 personas.
- Roles: Jefe de proyecto, documentador técnico, diseñador de interfaz, diseñador de Base de Datos, desarrollador.
- Entorno de desarrollo: plataforma libre (PHP, MySQL, Apache, Linux, Smarty, Ajax).
- Envergadura: medio.
- Tiempo de duración estimado: 6 meses.
- Complejidad: Media.
- Organización: Por módulos pero el desarrollo uno primero y otro después, no al mismo tiempo.
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

  - Poco tiempo de máquina para el desarrollo de las tareas.
  - Poca preparación de los estudiantes, poco conocimiento de las herramientas y estándares.
  - Falta de organización en la planificación de tareas y definición de responsabilidades.

Docencia

- Tipo de proyecto: Gestión.
- Cantidad de integrantes: 30 personas.
- Roles: analista, desarrollador, diseñador de base de datos, documentador.
- Entorno de desarrollo: plataforma libre (PHP, MySQL, Apache, Linux, Smarty, Ajax).
- Envergadura: medio.
- Tiempo de duración estimado: 5 a 6 meses.
- Complejidad: alta.
- Organización: modulo (menos el analista, tienen todos los roles).
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

    - Usuario no define bien los requisitos.

Colaboración médica

Es un proyecto que esta en fase de definición actualmente pero su tendencia es ser o caracterizarse semejante a los proyectos del área temática al cual pertenece, siendo este Planificación y control.

- Tipo de proyecto: Gestión.
- Cantidad de integrantes: Por definir.
- Roles: Analista, Desarrollador, Diseñador de Base de Datos, Documentador.
- Entorno de desarrollo: LAMP (Linux en el servidor, PHP, Apache, MySQL).
- Complejidad: media.
- Metodología utilizada: RUP.

Fisioterapia

- Tipo de proyecto: De Gestión.
- Cantidad de integrantes: 15 personas.

- Roles: programador, diseñador de Base de Datos, Analista, Prueba.
- Entorno de desarrollo: Lenguaje PHP, Servidor Apache, Gestor de Base de Datos MySQL.
- Envergadura: Mediana.
- Tiempo de duración estimado: un año.
- Complejidad: Alta.
- Organización: Equipos por roles.
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

  - Demora en la entrega de las tareas.
  - Deficiencia en la programación debido a los cambios.

SIUM

- Tipo de proyecto: Gestión.
- Cantidad de integrantes: 15 a 20 personas (proceso de crecimiento).
- Roles: arquitecto, desarrollador, diseñador de Base de Datos, Diseñador de Interfaz, analista.
- Entorno de desarrollo: plataforma libre (PHP, MySQL, Apache).
- Envergadura: medio.
- Tiempo de duración estimado: 6 meses.
- Complejidad: Media.
- Organización: por modulo (todos los roles).
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

  - Comunicación con el cliente.

Nefrología

- Tipo de proyecto: gestión.

- Cantidad de integrantes: 25 personas.
- Roles: arquitecto, desarrollador, diseñador de Base de Datos, Diseñador de Interfaz, analista.
- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: no está definido.
- Complejidad: Alta.
- Organización: por módulos (todos los roles).
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

    - Capacitación de los estudiantes.

Hospitales

- Tipo de proyecto: de Gestión.
- Cantidad de integrantes: 40 personas (aéreas temáticas).
- Roles: Analista, Diseñador de Base de Datos, Arquitecto, Desarrollador, Planificador, Calidad, Diseñador de interfaz.
- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: un año.
- Complejidad: media.
- Organización: equipos de desarrollo (2 analistas, 1 diseñador de Base de Datos, 2 implementadores, 1 diseñador de interfaz).
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

    - Comunicación escasa con el cliente.
    - Tecnología.

LIS

- Tipo de proyecto: de Gestión.
- Cantidad de integrantes: 25 personas.
- Roles: Analista, Diseñador de Base de Datos, Arquitecto, Desarrollador, Planificador, Calidad, Diseñador de interfaz.
- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: un año.
- Complejidad: media.
- Organización: equipos de desarrollo (2 analistas, 1 diseñador de Base de Datos, 2 implementadores, 1 diseñador de interfaz).
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:
    - Comunicación escasa con el cliente.
    - Tecnología.

FMS

- Tipo de proyecto: de Gestión.
- Cantidad de integrantes: 25 personas.
- Roles: Analista, Diseñador de Base de Datos, Arquitecto, Desarrollador, Planificador, Calidad, Diseñador de interfaz.
- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: un año.
- Complejidad: Media.
- Organización: Equipos de desarrollo (2 analistas, 1 diseñador de Base de Datos, 2 implementadores, 1 diseñador de interfaz).
- Metodología utilizada: RUP.

- Problemas en el proceso de desarrollo:

  - Comunicación escasa con el cliente.
  - Tecnología.

APS

- Tipo de proyecto: de Gestión.
- Cantidad de integrantes: 71 estudiantes, 6 profesores, 3 médicos de softel.
- Roles: Analista, Programador, Planificador, Diseñador de Base de Datos, Prueba, Documentación.
- Entorno de desarrollo: Plataforma libre, PHP, MySQL, Apache.
- Envergadura: Grande.
- Tiempo de duración estimado: Largo.
- Complejidad: Alta.
- Organización: por modulo en cada uno todos los roles.
- Metodología utilizada: RUP.
- Problemas en el proceso de desarrollo:

  - La planificación por los riesgos que se corren en la UCI.
  - Preparación de los estudiantes.
  - Desarrollo sobre una plataforma existente sin mucha documentación y ayuda.
  - Los estudiantes no asumen las tareas con responsabilidad y compromiso.
  - Retraso en la entrega.

PACS

- Tipo de proyecto: Sistema.
- Cantidad de integrantes: 30 miembros (en diferentes momentos).
- Roles: Jefe de Proyecto, arquitecto, desarrolladores, integrador, documentador, especificadores del diseño, control de calidad, análisis, desarrollador científico.

- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: 18 meses (2 versiones).
- Complejidad: Alta.
- Organización: Equipos de trabajo.
- Metodología utilizada: No exactamente usan una metodología definida, utilizan las plantillas de RUP e implementan algunas de las prácticas de DSDM.
- Problemas en el proceso de desarrollo:

  - Falta de horario priorizado en la producción.
  - Falta de recursos específicos para la gestión de proyectos.

DIANA

- Tipo de proyecto: de PC.
- Cantidad de integrantes: 8 miembros.
- Roles: Jefe de Proyecto, arquitecto, desarrolladores, integrador, documentador, especificadores del diseño, control de calidad, análisis, desarrollador científico.
- Entorno de desarrollo: .NET.
- Envergadura: Grande.
- Tiempo de duración estimado: 12 meses (dos versiones).
- Complejidad: Media.
- Organización: Equipos de desarrollo.
- Metodología utilizada: No exactamente usan una metodología definida, utilizan las plantillas de RUP e implementan algunas de las prácticas de DSDM.
- Problemas en el proceso de desarrollo:

  - Falta de horario priorizado en la producción.
  - Falta de recursos específicos para la gestión de proyectos.

Haciendo un análisis de la entrevista realizada a la totalidad de los líderes de proyectos de la facultad se puede decir que: se identificaron tres tipos de software, de gestión, de sistema y de PC, teniendo los de gestión mayor porciento de desarrollo; la cantidad de personas que involucran los mismos se incluyen en cuatro grupos, menos de 10, de 10 a 20, de 20 a 30 y más de 30 miembros, con mayor incidencia los de 10 a 20 y de 20 a 30 miembros; las tecnologías utilizadas son LAMP y .NET, teniendo LAMP mayor porciento de usabilidad, aunque no muy significativo.

Casi la totalidad de los proyectos usan RUP (Proceso Unificado de Rational) como metodología de desarrollo, solo dos no hacen uso de la misma (aclarar que según la opinión de los líderes de los proyectos esto se debe a que es la única que el estudiante conoce y puede aplicar); todos los proyectos involucran los roles principales (analista, diseñador de BD, diseñador de interfaz, programador, líder); generalmente son proyectos de complejidad media, en solo tres es alta y los problemas son común en casi la totalidad de los proyectos, solo en dos de ellos no se centraron en las mismas ideas (ver anexo 2).

De lo antes expuesto se puede decir además que los proyectos de la facultad 7 de la Universidad de las Ciencias Informáticas en su gran mayoría son proyectos de gestión con complejidad media caracterizados por ciclos relativamente cortos de desarrollo y un número reducido de integrantes y en los casos de ser amplio dicho número la organización es por equipos pequeños.

En la facultad casi la totalidad de los proyectos usan RUP (Proceso Unificado de Rational) como metodología para guiar el proceso de desarrollo de los mismos, no precisamente por ser la más factible sino por ser la única que conocen los miembros del equipo de desarrollo, de ahí que esto sea un problema seriamente influyente en el proyecto y su desarrollo.

Por otra parte, se cuenta con el resultado de 2 proyectos pertenecientes al área temática de GPI (Grupo de Procesamiento de Imágenes) que han tenido un buen desempeño en sus desarrollos, obteniendo resultados satisfactorios. Dichos proyectos no usan RUP como metodología a seguir, mas bien tienen enfocado su proceso de desarrollo con métodos ágiles como son el uso de algunas prácticas de XP que le han dado resultado y el uso de DSDM, aunque sus plantillas siguen siendo las que propone RUP. No han definido un proceso en especifico, pero sin duda alguna esta tendencia le ha dado buenos resultados.

No se trata de hacer a un lado el RUP porque evidentemente es una metodología muy completa y adaptable (puede verse en un estilo muy tradicional como en el ambiente ágil, todo depende del enfoque que se le de.) ya que provee a cada miembro del equipo de las guías de proceso, plantillas y mentores de herramientas necesarios para que el equipo completo tome ventaja de, entre otras, las siguientes mejores prácticas:

- Desarrollar software iterativamente: Permite una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad. Ayuda a atacar los riesgos mediante la producción de releases ejecutables progresivos y frecuentes que permiten la opinión e involucramiento del usuario. A través de las iteraciones que generan releases ejecutables, se logra detectar en forma temprana los desajustes e inconsistencias entre los requerimientos, el diseño, el desarrollo y la implementación del sistema, manteniendo al equipo de desarrollo focalizado en producir resultados.
- Administrar los requerimientos: Es un enfoque sistemático para hallar, documentar, organizar y monitorear los requerimientos cambiantes de un sistema. Permite:

    a) Que las comunicaciones estén basadas en requerimientos claramente definidos.
    b) Que los requerimientos puedan ser priorizados, filtrados y monitoreados.
    c) Que sea posible realizar evaluaciones objetivas de funcionalidad y performance.
    d) Que las inconsistencias se detecten más fácilmente.
- Utilizar arquitecturas basadas en componentes: El proceso de software debe focalizarse en el desarrollo temprano de una arquitectura robusta ejecutable, antes de comprometer recursos para el desarrollo en gran escala.
- Modelizar software visualmente: Las abstracciones visuales ayudan a comunicar diferentes aspectos del software; comprender los requerimientos, ver como los elementos del sistema se relacionan entre sí, mantener la consistencia entre diseño e implementación y promover una comunicación precisa.
- Verificar la calidad de software: Es necesario evaluar la calidad de un sistema respecto de sus requerimientos de funcionalidad, confiabilidad y performance. La actividad fundamental es el testing, que permite encontrar las fallas antes de la puesta en producción. El aseguramiento de la

calidad se construye dentro del proceso, en todas las actividades, involucrando a todos los participantes, utilizando medidas y criterios objetivos, permitiendo así detectar e identificar los defectos en forma temprana.

- Controlar los cambios al software: La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. Es también una guía para establecer espacios de trabajo seguros para cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando los cambios de todos los elementos de software (modelos, código, documentos, etc.).

Aunque es precisamente eso lo peor que tiene, el abarcar tantas cosas, en proponer muchas vías y nada en especifico, ser un armazón de procesos. Por lo que la realidad demuestra que no está siendo eficiente en muchos de los proyectos, pues su enfoque es totalmente pesado e incomodo de acuerdo con las características de los mismos.

Es por ello la necesidad de puntualizar los principales problemas que están afectando la productividad y avance en este sentido en la facultad.

2.3 Clasificación de los proyectos de la Facultad 7

Independientemente de los datos obtenidos a través de las entrevistas realizadas a los jefes de proyectos de la facultad hubo un intercambio interesante con un tesista de la facultad donde se recopilaron algunos datos mas detallados de los principales problemas que acarrean los proyectos en su proceso de desarrollo que afectan el desarrollo y la eficiencia de los mismos, siendo estos los expuestos a continuación:

- Problemas asociados a la experiencia técnica de los miembros del proyecto

  ✓ Desconocimiento en el manejo de las herramientas tecnológicas por falta de preparación y superación de los miembros del equipo ya sea por la falta de recursos tecnológicos como por la actitud y desinterés de los mismos miembros.

- ✓ Demoras en la planificación de otras tareas del proyecto producto a la sobrecarga de actividad de uno o varios miembros del proyecto debido a la causa del problema anteriormente planteado.
- ✓ La poca capacidad de seguimiento y cumplimiento de la tarea provocando que muchas se atrasen o no se cumplan debido a la relación profesor – estudiante que esta desproporcionada.
- ✓ Poca motivación de los miembros del proyecto provocando incumplimientos e irresponsabilidades debido a la poca información del área en que trabajan o simplemente la no preferencia del miembro por lo que tiene la responsabilidad de hacer.

- Problemas asociados al cliente.

- ✓ Bajo conocimiento del cliente hace que los requisitos sean cambiantes pues lo que se explica muchas veces resulta siendo lo que no es y provoca inconformidades.
- ✓ Poca interacción de los miembros del proyecto con el cliente que son realmente los que saben lo que quieren y como lo quieren producto a la poca planificación de visitas por parte de los mismos.
- ✓ Desconocimiento en la aplicación de técnicas para el levantamiento de requisitos, afectando la comunicación con el cliente.

- Problemas asociados a la definición y seguimiento del proceso de desarrollo del software.

- ✓ Baja calidad del plan de revisiones técnicas formales, lo que trae como consecuencia no tomar las medidas inmediatas que ayudan a detener las dificultades, atrasando el desarrollo del producto debido a la juventud de la universidad en estos campos.

- ✓ No se cuenta con un buen plan de desarrollo que permita la planificación adecuada de los proyectos productivos de la facultad o esta planificación no es seguida y llevada adecuadamente.
- ✓ Cambios constantes que afectan la planificación del producto debido a la propia dinámica de la universidad y la poca interacción con el área productiva.

- Problemas asociados a las herramientas de desarrollo.

- ✓ Poca o ninguna preparación en una determinada herramienta por lo que en la primera etapa de desarrollo es preciso dedicar tiempo a la capacitación y superación de los miembros del proyecto.

- Problemas asociados al tamaño del producto.

- ✓ La división de proyectos en grupos de trabajo para un mejor desarrollo del mismo a veces nos es tan mejor pues si no se logra que el trabajo se haga a la par lo que trae consigo es atrasos.

- Problemas asociados a la complejidad del sistema a construir y la tecnología punta que requiere el sistema.

- ✓ Incorrecta selección de la tecnología a utilizar a la hora de definir el proyecto.
- ✓ No siempre tiene que ser RUP (Proceso Unificado de Rational) pues la metodología a seguir en el proceso de desarrollo del software es definido en la reunión de concepción del proyecto, donde en dependencia de las características que tiene el mismo es que se decide que proceso metodológico será el mejor y el más adecuado utilizar.

La mayoría de todos estos problemas generan lo que se puede decir que es el mal del proceso productivo de la facultad, su talón de Aquiles que no es más que los atrasos en las fechas de entrega que tienen asignado los proyectos.

Ahora bien, si se tiene en cuenta todos estos factores a la hora de clasificar los proyectos productivos de la facultad y teniendo en cuenta caracterización realizada a cada uno de ellos se puede decir que:

1. Proyecto con requisitos imprecisos y muy cambiantes

   - ✓ Docencia
   - ✓ SIUM

2. Proyectos en los cuales existe un alto riesgo técnico.

   - ✓ BM (base material)
   - ✓ Estadística
   - ✓ Nefrología
   - ✓ APS
   - ✓ Hospitales
   - ✓ DIANA
   - ✓ PACS

3. Proyectos con rápido cambio de requisitos

   - ✓ Hospitales
   - ✓ Fisioterapia

4. Proyecto con número de miembros no mayor de 10 (con mas formar equipos)

   - ✓ Docencia
   - ✓ BM
   - ✓ Estadística
   - ✓ SIUM
   - ✓ Nefrología
   - ✓ Hospitales

- ✓ Fisioterapia
- ✓ DIANA
- ✓ PACS

5. Proyectos con desarrollos de soluciones de negocios sujetas a márgenes de tiempo estrecho.

- ✓ Docencia
- ✓ DIANA
- ✓ PACS

6. Proyectos de complejidad cruente

- ✓ Ninguno

7. Proyectos poco fuera del caos

- ✓ Ninguno

8. Proyectos mayores

- ✓ APS

9. Proyectos de misión critica

- ✓ Ninguno

Las clasificaciones utilizadas van en correspondencia con las características de las diferentes metodologías ágiles que fueron analizadas en el capítulo anterior, con el fin de identificar las mayores incidencias y poder definir cuáles son las que más se adecuan acorde con las características de los proyectos, por lo que se llego a la conclusión que SCRUM, XP y DSDM son las que más incidencias tienen y evidentemente esto será un punto de partida en la búsqueda de la posible solución.

Como fue visto, en este capítulo se realizó un estudio de los aspectos más significativos de los proyectos de la Facultad 7, que permitió la caracterización y clasificación de los mismos, así como la identificación de los principales problemas que tienen que enfrentar dichos proyectos en todo su proceso de desarrollo. Siendo el punto de partida de la propuesta solución, que contribuirá a la mejora en el proceso de desarrollo de los proyectos y por ende del proceso productivo de la facultad.

CAPÍTULO 3 DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA

Durante el transcurso de esta investigación se ha abordado temas relacionados con las metodologías ágiles de desarrollo como han sido sus características fundamentales y las diferentes metodologías que están implicadas en este contexto, además se ha hecho un estudio del proceso productivo de la facultad basado en las principales características de los proyectos productivos.

La producción de software incluye aspectos importantes y determinante, pero sin dudas y es a lo que menos se le presta atención es a una selección adecuada de la metodología a seguir para llevar adelante dicho proceso. No a la esquematización, hay que documentarse y buscar vías alternativas que también resultan muy satisfactorias cuando las condiciones de los proyectos van muy acorde a los que ellas exigen. La universidad y principalmente la facultad 7 no está ajena a esta situación, proyectos que fracasan, otros que ven su desarrollo seriamente retardo y afectado por el tiempo, donde se buscan las causas en la preparación y motivación de los estudiantes sin darnos cuenta que este puede ser nuestro gran y único problema

Precisamente es el objetivo de dicho capítulo darle solución a dicha problemática haciendo uso adecuado de las diferentes metodologías ágiles de desarrollo que existen.

Ahora bien, para darle solución al problema planteado se propone explicar, guiar y orientar como seria introducir procedimientos ágiles en el proceso productivo de la facultad.

Antes de comenzar con la propuesta se deben esclarecer algunos términos que han sido empleados a lo largo de toda la investigación y que son un eslabón importante cuando se habla de hacer software. Uno de estos elementos es el proceso de software ¿Qué es? ¿Por qué es importante? ¿Cuáles son los pasos? ¿Cuál es el producto obtenido? Estas y otras preguntas surgen y es necesario darle respuesta.

Proceso de software

El proceso de software no es más que una serie de pasos predecibles a seguir – un mapa de carreteras que te ayude a obtener el resultado oportuno de calidad. Pero ¿Qué es exactamente desde un

punto de vista técnico? Pues bien, es un marco de trabajo de las tareas que se requieren para construir software de alta calidad; define el enfoque que se toma cuando el software es tratado por la ingeniería. Su importancia estada dada por el hecho de que proporciona estabilidad, control y a una actividad que puede, si no se controla, volverse caótica. Realmente a un nivel detallado el proceso que adoptemos depende del software que se esté construyendo. Ahora, desde el punto de vista de un ingeniero de software los productos obtenidos son programas, documentos y datos que se producen como consecuencia de las actividades de ingeniería de software definidas por el proceso. [19]

Gestión de proyectos

Por otra parte, la Gestión de Proyectos implica la planificación, supervisión y control no solo del proceso de software, sino también del personal y los eventos que ocurren mientras evoluciona el software desde la fase preliminar a la implementación operacional. Es importante debido a que la construcción del software es una empresa compleja, particularmente si participa mucha gente, trabajando durante un periodo de tiempo relativamente largo, lo que hace que necesiten ser gestionados. Los pasos principales se basan en la comprensión de las cuatro P (persona, producto, proceso y proyecto). El personal debe estar organizado para desarrollar el trabajo del software con efectividad, la comunicación con el cliente debe ocurrir para que se comprendan el alcance del producto y los requisitos, debe seleccionarse el proceso adecuado para el personal y el producto. El proyecto debe planificarse estimando el esfuerzo y el tiempo para cumplir las tareas, definiendo los productos del trabajo, estableciendo puntos de control de calidad y estableciendo mecanismos para controlar y supervisar el trabajo definido en la planificación. Lo que se obtiene del mismo es un plan de proyecto que se realiza al comienzo de la actividad de gestión, el plan define el proceso y las tareas a realizar el personal que realizará el trabajo y los mecanismos para evaluar los riesgos, controlar el cambio y evaluar la calidad. [20]

Ingeniería de software

La ingeniería de software es la aplicación de un enfoque sistémico, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software. 2. El estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software [21]

El trabajo que se asocia en la ingeniería de software se puede dividir en tres fases genéricas, con independencia del área de aplicación, tamaño o complejidad del proyecto las mismas son:

1. Fase de Definición: se centra sobre el qué, aquí se intenta que información ha de ser procesada, que función y rendimiento se desea, que comportamiento del sistema, que interfaces va a ser establecidas, que duración se necesita para definir un sistema correcto, en fin, han de identificarse los requisitos claves del sistemas y del software. Cuenta con tres tareas principales: ingeniería de sistemas o de información, planificación del proyecto y análisis de los requisitos.
2. Fase de Desarrollo: se centra en el cómo, se intenta definir cómo han de diseñarse las estructuras de datos, como ha de implementarse la función dentro de una arquitectura de software, como han de implementarse los detalles de procedimentales, como han de caracterizarse interfaces, cómo ha de traducirse el diseño en un lenguaje de programación y cómo ha de realizarse la prueba. Cuenta con tres tareas: diseño del software, generación de código y prueba del software.
3. Fase de Mantenimiento: Se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que va evolucionando el entorno del software y a cambios debido a las mejoras producidas por los requisitos cambiantes del cliente. Tipos de cambios: Corrección, Adaptación, Mejora, Prevención.

3.1 PROCEDIMIENTOS

Vasados en los conceptos anteriores se procederá a plantear y explicar la propuesta de los procedimientos ágiles que sustentaran la gestión de los proyectos y la ingeniería de software a seguir por los mismos.

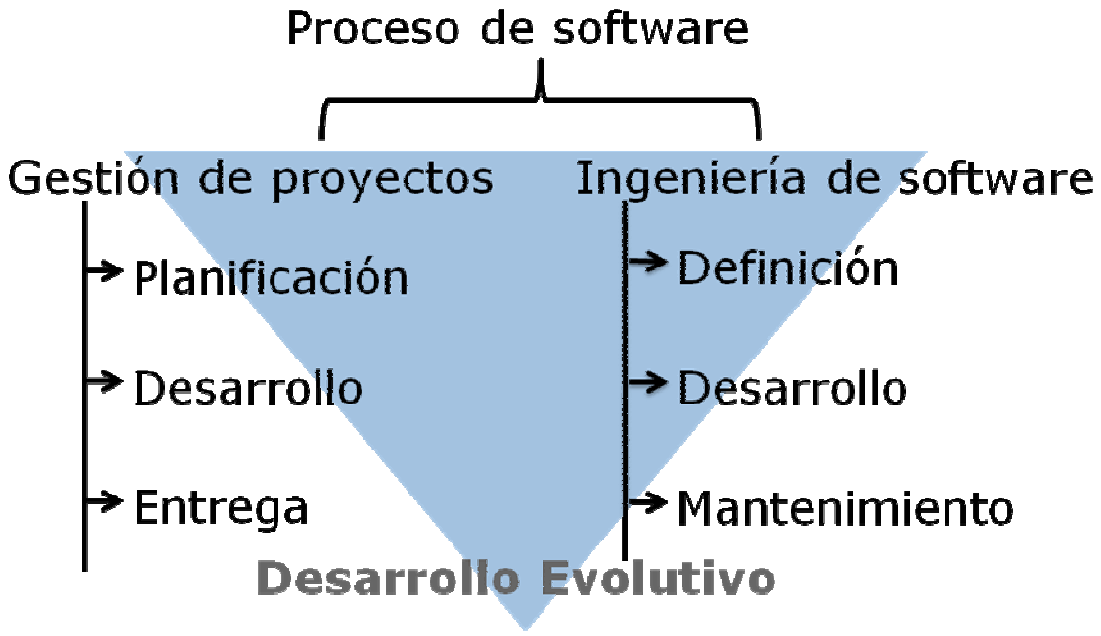


Figura 1. Esquema de la propuesta.

3.1.1 Gestión del proyecto

Antes de hacer la presentación Inicial del proyecto es necesario conversar con el usuario/cliente de que se trata el proyecto (semejante a lo que se hace hoy en día en la facultad) y se sugiere que se tenga en cuenta los siguientes aspectos:

- Usar la planificación Inicial de Proyecto como elemento mínimo de la planificación del proyecto completo.
- Explicar al cliente/usuario y dejar constancia en la definición técnica que parte del enfoque de solución del proyecto incluye usar métodos ágiles, mencionando de cuales se tratan y que el modelo de cicle de vida será desarrollo evolutivo.
- El detalle de la planificación se realiza en cada iteración

Como propuesta, se usará SCRUM para la planificación de los proyectos que usarán métodos ágiles como metodología para su proceso de desarrollo pues SCRUM es una forma de gestionar

proyectos de software, no es una metodología de análisis, ni de diseño, es una metodología de gestión del trabajo. Aquí no se pretende implantar dicha metodología en su totalidad, de la misma serán tomadas algunas prácticas para su implantación, describiéndolo a continuación.

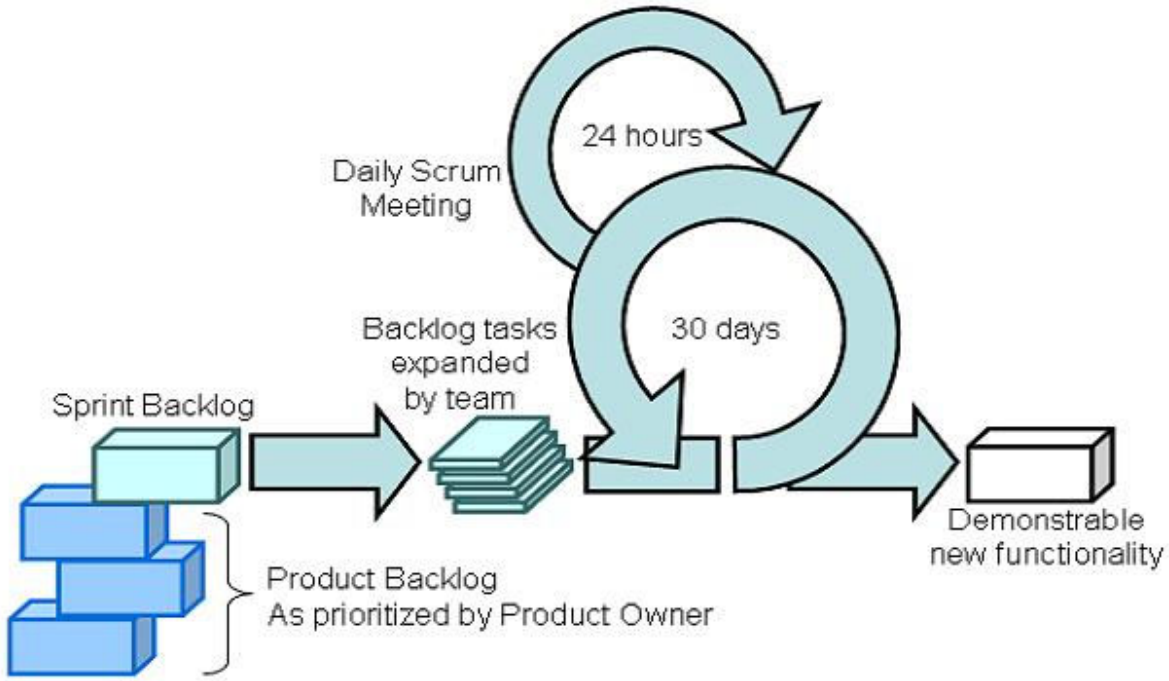


Figura 2. Proceso de desarrollo de SCRUM.

3.1.1.1Planificación

Propósito**:** Establecer la visión, fijar expectativas, y asegurar financiamiento.

Entrada: la concepción inicial del producto

Actividades**:** Escribir la visión, presupuesto, Reserva del producto con estimaciones iniciales

- Crear la Lista de reserva del producto (Product Backlog List) y controlar su consistencia: Posibles elementos de esta lista son requerimientos técnicos y del negocio, funciones, errores a reparar, defectos,

mejoras y actualizaciones tecnológicas requeridas.

Es importante controlar la consistencia de la lista. Para esto se agregan, modifican, eliminan, especifican y priorizan sus elementos.

Roles: Usuario Interno, cliente.

Lista de Reserva del Producto (Product Backlog List) (Ver anexo 8)

Es una lista priorizada que define el trabajo que se va a realizar en el proyecto. Cuando un proyecto comienza es muy difícil tener claro todos los requerimientos sobre el producto. Sin embargo, suelen surgir los más importantes que casi siempre son más que suficientes para un Sprint.

Esta lista puede crecer y modificarse a medida que se obtiene más conocimiento acerca del producto y del cliente. Con la restricción de que solo puede cambiarse entre Sprint. El objetivo es asegurar que el producto definido al terminar la lista es el más correcto, útil y competitivo posible y para esto la lista debe acompañar los cambios en el entorno y el producto.

Para gestionarla se puede usar Excel.

Sprint

Un Sprint es el procedimiento de adaptación de las cambiantes variables del entorno (requerimientos, tiempo, recursos, conocimiento, tecnología). Son ciclos iterativos en los cuales se desarrolla o mejora una funcionalidad para producir nuevos incrementos. Durante un Sprint el producto es diseñado, codificado y probado. Y su arquitectura y diseño evolucionan durante el desarrollo.

El objetivo de un Sprint debe ser expresado en pocas palabras para que sea fácil de recordar y esté siempre presente en el equipo. Es posible definir una serie de restricciones que el equipo deba aplicar durante un Sprint, por ejemplo: un Sprint tiene una duración planificada de entre una semana y un mes. No es posible introducir cambios durante el Sprint, por lo tanto para planificar su duración hay que pensar en cuanto tiempo se puede comprometer a mantener los cambios fuera del Sprint. Dependiendo del tamaño del sistema, la construcción de un release puede llevar entre 3 y 8 Sprint. Por otra parte

podrían formarse equipos para desarrollar en forma paralela distintos grupos de funcionalidad.

- Priorizar la Lista de reserva del producto: Esta actividad se basa en considerar que elementos tienen más o menos influencia en el éxito del proyecto en un momento dado; considerando que los elementos con mayor prioridad se realizan primero.

  Roles: Usuario Interno

- Valoración del esfuerzo: Es un proceso iterativo que reúne toda la información que haya acerca de un elemento para tener un mayor nivel de precisión en la estimación. Siempre se mide el esfuerzo que falta para cumplir con el o los objetivos tanto a nivel de la lista de reserva del producto como para la Reserva del Sprint.

  Roles: Usuario Interno, Miembros del Proyecto

- Reunión de Revisión del diseño: En esta instancia se comunica el diseño a los interesados para revisar el cumplimiento de los ítems especificados en la Reserva del producto

  Salida: Lista de reserva del producto, Arquitectura

3.1.1.2 Desarrollo

Propósito**:** Implementar un sistema listo para entrega en una serie de iteraciones de 30 días. (El tiempo puede decrementarse a la medida que se está refinando el producto)

Entrada**:** Lista de reserva del producto.

Actividades**:** Reunión de Planificación de la Iteración. Definir la Reserva de la Iteración. Reuniones de Coordinación y Revisión de la Iteración.

- Junta de planificación del Sprint: Es una reunión organizada por el Líder del Proyecto, que se realiza en dos fases.

- ✓ La primera fase tiene como objetivo establecer que ítems de la Lista de Reserva del Producto van a ser realizados durante el Sprint. Esto se realiza a partir de lo que el Equipo considera que puede construir durante el Sprint.
- ✓ En la segunda fase se decide como se van a alcanzar los objetivos del Sprint. En esta fase se crea la Reserva del Sprint, indicando qué tareas debe desempeñar el equipo para cumplir con dichos objetivos.

Roles primera fase: Lieder del Proyecto, Customer, User, Management. Usuario Interno, Miembros del Proyecto

Roles segunda fase: Miembros del Proyecto, Líder del Proyecto, Usuario Interno

- Junta de seguimiento: Las reuniones se realizan en el mismo lugar y a la misma hora una vez por semana, idealmente a principios de semana para definir el trabajo para el día. Tienen una duración de 30 minutos, no son para resolver problemas, en ellas se realizan tres preguntas:

  - ¿Qué hiciste? (Destacar que el trabajo que se hizo para la consecución del objetivo es valido).
  - ¿Qué harás? (Al salir de la reunión todo el mundo sabe lo que tiene que hacer y todos están alineados. Mucha gente no sabe organizarse y pierde el tiempo en cosas menos importantes por lo que esto le sirve para organizar su trabajo diario).
  - ¿Qué obstáculos ves en tu camino? (Es el momento para que salga a la luz todos los problemas que tienen las personas para la realización de su trabajo).

Sugerencia: Estas reuniones no pueden ser sustituidas por reportes vía e-mail por dos motivos:

- El equipo entero ve todo el paisaje cada día
- Es un elemento de presión para que el individuo haga lo que dijo que va a hacer

Roles: Miembros del Proyecto

- Junta de revisión: Es una reunión informal que tiene como regla que su preparación no

puede tomar más de 2 horas. En ella el equipo presenta lo que ha logrado durante el Sprint. Generalmente toma la forma de una demo de las nuevas características o la arquitectura.

Roles: Customers, Management, Usuario Interno, otros interesados

Salida: Incremento del producto

3.1.1.3 Entrega

Propósito: Puesta en Operación

Actividades**:** Documentación, Entrenamiento, Marketing, Ventas

Contiene el cierre del release. Para ingresar a esta fase se debe llegar a un acuerdo respecto a las variables del entorno por ejemplo que los requerimientos fueron completados. El sistema está listo para ser liberado y es en esta etapa en la que se realiza integración, pruebas del sistema y documentación.

Salida: Producto

3.1.1.4 Definición de Roles

• Líder del Proyecto (Scrum Master)

Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con las prácticas y que todo funciona según lo planeado. Su principal trabajo es remover impedimentos y reducir riesgos del producto.

- Coordinar y facilitar las reuniones
- Asegurar que se consigue los objetivos de la reunión de planificación de la iteración

• Cliente (Customer)

El cliente participa en las tareas que involucran la lista de reserva del producto

– Presentar la Reserva del producto al equipo, enfatizando el valor y prioridades del mismo

– Definir la meta de la iteración

– Aprobar las modificaciones en la Reserva del producto y en el alcance de la iteración

• Miembros del Proyecto (Scrum Team)

Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de un Sprint. Sus tareas son: Estimar esfuerzo, crear la reserva del Sprint, revisar la Lista de Reserva del Producto y sugerir obstáculos que deban ser removidos para cumplir con los items que aparecen.

Típicamente es un equipo de entre 5 y 10 personas cada una especializada en algún elemento que conforma los objetivos a cumplir, por ejemplo: Programadores, Diseñadores de Interfaz de usuario, etc. La dedicación de los miembros del equipo debería ser full-time con algunas excepciones. La membrecía solo puede cambiar entre sprints (no durante).

• Gerente (Management)

Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el proyecto. Participa en la selección de objetivos y requerimientos y en la selección del Usuario Interno. Tiene la responsabilidad de controlar el progreso y trabaja junto con el Jefe de Proyecto en la reducción de la Lista de Reserva del Producto.

• Usuario Interno (Product Owner)

Es el responsable del proyecto, administra, controla y comunica la Lista de Reserva. Es el responsable de encontrar la visión del producto y reflejarla en la lista de Reserva.

3.1.1.5 Planificación de una Iteración

Propósito de la Iteración

• Transformar un subconjunto de la Reserva del producto en un incremento en la funcionalidad del producto que sea potencialmente entregable a los usuarios. Para ello, Usuario Interno, el Líder del Proyecto y los Miembros del Proyecto se juntan, previamente a cada iteración, para determinar en qué funcionalidad del producto trabajará el equipo durante la próxima iteración; esta reunión se denomina "Reunión de Planificación de la Iteración" y cuenta con dos partes:

Parte 1 de Planificación de la Iteración

• Objetivo: El equipo define una meta para la iteración, así como también selecciona las características de la Reserva del producto que van a ser planificadas para conseguir la meta propuesta

• Duración: 1 a 4 horas dependiendo de qué tan obvio es el trabajo que hay que desarrollar durante la próxima iteración

• Entradas:

- El producto resultante de la iteración previa
- La reserva del producto priorizado
- El estado actual de la tecnología y las condiciones del negocio

Parte 2 de Planificación de la Iteración

• Objetivo: Definir el trabajo que es necesario desarrollar durante la iteración. Esto da origen la Reserva de la Iteración.

• Duración: 4 horas

• Salidas:

- Reserva de la Iteración: Lista de tareas de no más de 16 horas cada una
- Arquitecturas y Diseños

– Asignación de recursos a cada tarea
– Fechas para la reunión de revisión de los resultados de la iteración, y para la próxima reunión de planificación

Reserva de la Iteración

Descripción de la tarea	Responsable	Estado	Horas remanentes		
			1	2	3
		(Completada En progreso No comenzada)			

Tabla 1. Reserva de la Iteración

Recomendación general: Para gestionar el proyecto sólo necesitamos dos listas y se puede usar Excel para manejarlas.

3.1.2 Ingeniería de software

Como propuesta para llevar a cabo el proceso de desarrollo del proyecto se tomará en cuenta las mejores prácticas de la metodología XP, procurando que el proceso sea efectivo y eficiente.

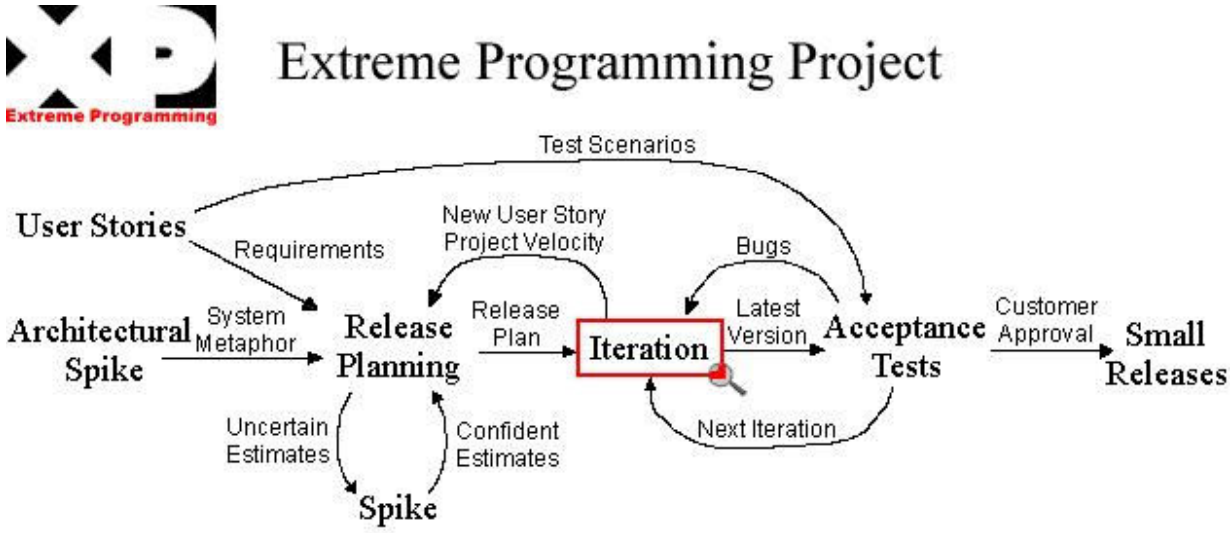


Figura 3. Modelo XP (Programación Extrema)

3.1.2.1 Definición

La planificación

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.

1 Cada entrega es lo más corta posible:

- Contenga requisitos más valiosos del sistema (básicos)
- Reducen el riesgo: mayor retroalimentación desde el cliente, y más frecuente

2 Minimizar el nº de historia de usuarios que componen una entrega: No realizar historias de usuarios a medias.

3 A cada programador se le asigna una tarea de la historia de usuario

Historias de usuario (user stories) (Ver anexo 4)

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software, lo que equivaldría a los casos de uso en el proceso unificado. Las mismas son escritas por los clientes como las tareas que el sistema debe hacer y su construcción depende principalmente de la habilidad que tenga el cliente para definirlas. Son utilizadas como el único documento de requisitos que se genera en *XP*. Son escritas en lenguaje natural, sin un formato predeterminado, no excediendo su tamaño de unas pocas líneas de texto.

Las historias de usuario guían la construcción de los tests de aceptación, elemento clave en *XP* (deben generarse uno o más tests para verificar que la *story* ha sido correctamente implementada) y son utilizadas para estimar tiempos de desarrollo. En este sentido, sólo proveen detalle suficiente para hacer una estimación razonable del tiempo que llevara implementarla. En el momento de implementar se deben detallar a través de la comunicación con el cliente. Son la base para las pruebas funcionales

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.

1 Decisiones de negocio (cliente):

- **Alcance:** ¿Cuándo debe estar listo el producto para que sea valioso

en producción?

- **Prioridad:** Prioriza la incorporación de las historias de usuario.
- **Composición de entregas:** ¿Qué se necesita para que el negocio sea mejor antes de tener el software?
- **Fechas de entrega:** Fechas cuando el software funcionando causaría una gran diferencia.

2 Decisiones técnicas (programadores y otros):

- **Estimaciones**: ¿Cuánto tiempo tardará en implementarse una historia de usuario?
- **Consecuencias:** Tener en cuenta las consecuencias técnicas de determinadas decisiones de negocio
- **Proceso:** Organización del proceso y el equipo
- **Planificación detallada:** Dentro de una entrega, qué historias de usuario se realizan primero. Intentar trasladar los segmentos de desarrollo más arriesgados al principio, intentando respetar las prioridades del negocio

3 Reunión diaria “Stand-up Meeting”

- Todo el equipo
  - ✓ Problemas
  - ✓ Soluciones
- De pie en un círculo
  - ✓ Evitar discusiones largas
  - ✓ Sin conversaciones separadas

Captura de requisitos:

1 Historias del Usuario (*User-Stories*)

- Establecen los requisitos del cliente

- Trozos de funcionalidad que aportan valor
- Se les asignan tareas de programación con un nº de horas de desarrollo (ver anexo 7)
- Las establece el cliente

Roles: Cliente, Entrenador (jefe de proyecto), Encargado de seguimiento.

Diseño

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.

1 Se diseña “la cosa más simple que pueda funcionar”
2 Uso de tarjetas CRC
3 Diseño de software correcto, es aquel que:

- Supera todas las pruebas
- No tiene lógica duplicada
- Pone de manifiesto las intenciones importantes de los programadores
- Tiene el mínimo número de clases y métodos

Tarjetas CRC (Ver anexo 3)

Las clases pueden también ser descubiertas usando tarjetas Colaboración-Responsabilidad-Clase (CRC por sus siglas en inglés). Fueron introducidas por r Ward Cunningham y Kent Beck en OOPSLA en 1989. Una tarjeta CRC es una tarjeta indexada de 3 x 5 que muestra: El nombre de la clase y su descripción, La responsabilidad de la clase, Conocimiento interno de la clase, Servicios brindados por la

clase, Los colaboradores para las responsabilidades (Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad)

- Una sección de tarjeta CRC
  - Un grupo de personas es escogida para personificar un escenario.
  - Una tarjeta es creada para cada objeto en el escenario.
  - A cada participante se le asigna un grupo de tarjetas.
  - La persona se transforma en la "clase".
  - El escenario definido es actuado por los participantes.
  - En las tarjetas se anotan las responsabilidades y colaboraciones.
  - Nuevas tarjetas son creadas para cada nuevo objeto descubierto.

- Beneficios de las Tarjetas CRC.
  - A medida que más y más escenarios con completados, emergen rastros de colaboración.
  - Las tarjetas pueden ser físicamente arregladas para representar esas colaboraciones.
  - Esto puede ayudar a identificar jerarquías de generalización/especificación, o jerarquías de agregación entre las clases.
  - Las tarjetas CRC son más efectivas para grupos novatos en técnicas OO porque ellas:
    - Previenen la focalización en los detalles OOP.
    - Previenen la generalización prematura.
    - Remarcan el "pensamiento objeto".

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 [22] 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.

1 Da un contexto al equipo para entender los elementos básicos y sus relaciones
2 Proporciona integridad conceptual

Refactorización

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 [23]. Robert Martin [24] 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.

1 Refactorización = Mejora del código.
2 Intentar eliminar complejidad.
3 Código duplicado: Refactorización.
4 Se plantea su aplicación después de implementar cada historia de usuario.

Roles: Entrenador, programador, consultor, cliente.

3.1.2.2 Desarrollo

Codificación

Programación en pareja

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 [25], 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.

1 Toda el código se escribe en parejas

- Se produce código de mayor calidad
- Extiende el conocimiento

2 Se realiza el trabajo de 1 persona en casi la mitad del tiempo y mejor (cuestionable)

Propiedad colectiva

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.

1 Cualquiera puede modificar el código en cualquier momento Se evitan cuellos de botella en la codificación

2 Todos asume las responsabilidades sobre el conjunto del sistema

3 Todos conocen algo sobre todas las partes y conocen muy bien aquéllas en las que trabajan.

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 [26] 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.

1 El código se integra y se prueba después de pocas horas
2 Existe una ordenador dedicado para la integración
3 Cada pareja integra su código en dicho ordenador

Cliente en sitio

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 [27] 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.

1 Cliente real = Aquel que usará el sistema cuando esté en producción
2 El cliente real debe estar con el equipo de trabajo:

- Responder preguntas
- Resolver disputas
- Establecer prioridades
- Discutir mejoras

Estándares de programación.

Enfatizar 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.

1 Son fundamentales cuando los programadores cambian de pareja o hacen *refactoring* del código de otros

2 Se consigue un código con el mismo estilo, homogéneo, legible.

Roles: programador, cliente, gestor

Pruebas

Pruebas (ver anexo 3)

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.

1 Las pruebas unitarias se escriben ANTES que el código

2 Pruebas automatizadas

3 Permiten el desarrollo de proyectos de forma rápida y segura
4 Pruebas unitarias: programadores
5 Pruebas funcionales: cliente
6 Resultado: Un programa cada vez más seguro

Roles: Encargado de pruebas, programador.

Recomendación General: *Espacio de trabajo:* Espacio abierto

Ahora bien, 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 (obtenida de [28]), donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí.

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

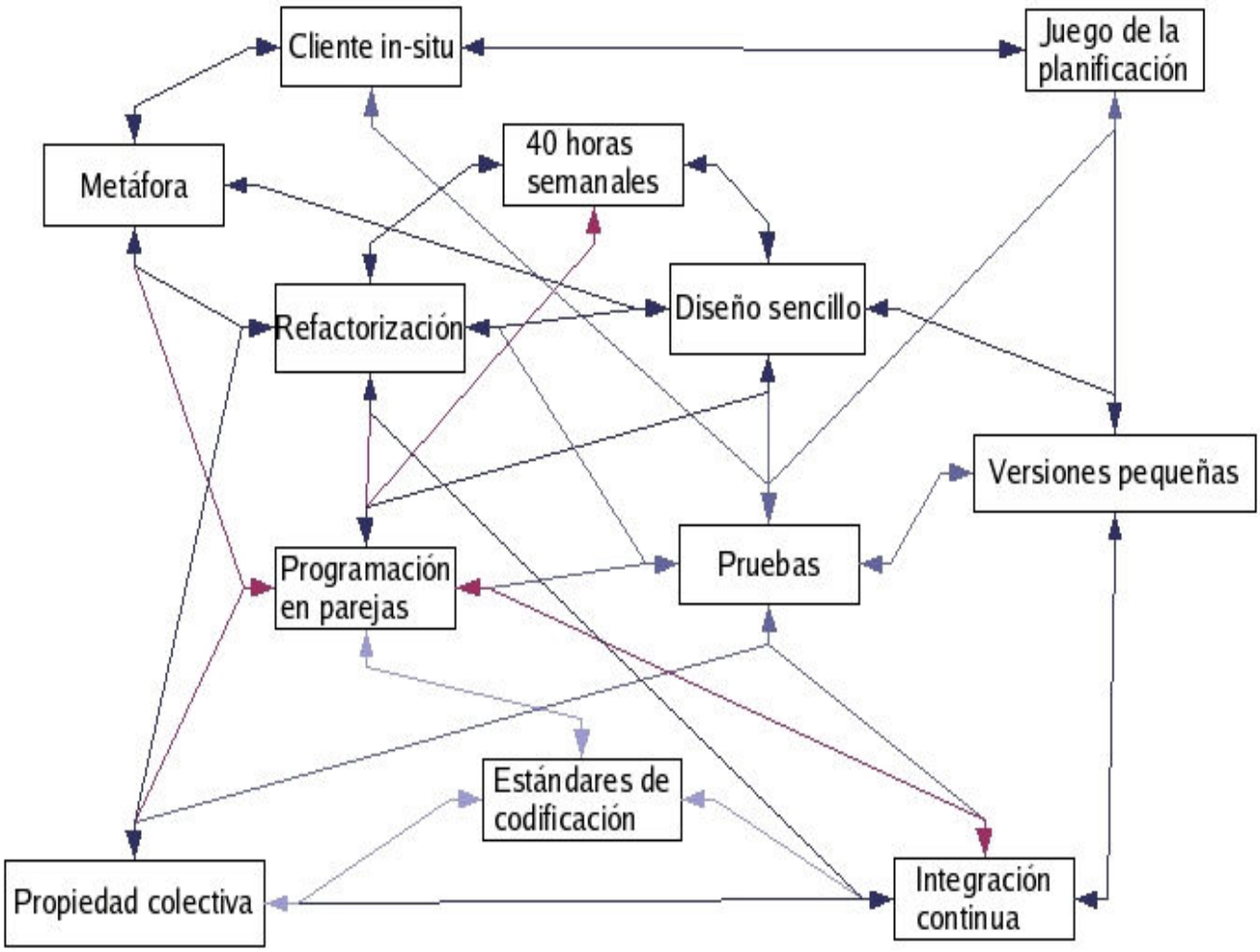


Figura 3. Las prácticas se refuerzan entre sí

3.1.2.3 Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

3.1.2.4 Definición de Roles

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

3.2 Resultados Prácticos

Esto es solo los primeros pasos de esta gran tarea de revolucionar el proceso productivo y llevarlos al enfoque ágil, por lo que no se cuenta con datos concretos del índice de éxito de proyectos, pero se asegura que estos procedimientos tendrán mucho auge porque están sustentados en metodologías que tienen el apoyo de muchos gurús en la ingeniería de software, porque se basa en procesos sencillos y porque tiene sentido.

3.3 Valoración

Con esta propuesta se pretende dar solución a muchos de los problemas que afectan el desarrollo y buena ejecución de proyecto como son el comportamiento de los estudiantes y la motivación y entrega de los mismos pues al llevarse un control de que se debe hacer y cuando debe entregarse una determinada tarea, el nivel de preocupación y responsabilidad de los miembros del equipo aumenta. Además, cuando un estudiante tiene que prepararse para desarrollar una tarea de gran envergadura consume más tiempo en su preparación que si dicha tarea es dividida en sub tareas que viabilicen con mayor facilidad su aprendizaje y le posibiliten avanzar.

Además, esta ayudará al líder de proyecto a llevar un mejor control de las tareas así como la

planificación de las mismas. También reconocer la tendencia al compañerismo y solidaridad, no dejando margen al egoísmo e individualidad.

Por otra parte involucrar a los miembros del equipo de desarrollo en las decisiones sobre el proyecto y sus vías de desarrollo, lo puede ser de gran ayuda a la hora de aumentar la motivación, si dejar fuera que se logra una mayor interacción con el cliente al lograr que este sea parte del equipo.

Sin duda alguna en un ambiente tan dinámico como el de la universidad intentar predecir cosas complejas implica intentar predecir un futuro complejo e incierto y concentrarse en las tareas y tiempos, puede provocar que se ignoren riesgos, acuerdos y aspectos técnicos.

Siempre tener en cuenta que una metodología ágil puede ser una buena forma de empezar cuando no existe un proceso o existe pero no reacciona bien a los cambios o existe pero el equipo no está contento con él. Además, es fácil de financiar, a los programadores les gusta y al cliente le agrada el valor añadido.

En este capítulo se confeccionó una guía de procedimientos ágiles teniendo en cuenta las características de los proyectos que se desarrollan en la Facultad 7, partiendo de las metodologías SCRUM, XP y DSDM, todo sobre la base del proceso de software y sus dos vertientes: La gestión de proyectos y la ingeniería de software.

CONCLUSIONES

En esta investigación se realizó una guía de procedimientos ágiles la cual se propone ser introducida en el proceso productivo de la Facultad 7, lográndose además:

- Caracterizar las metodologías ágiles más conocidas a nivel mundial.
- Caracterizar los proyectos productivos de la Facultad 7 teniendo en cuenta una serie de parámetros establecidos.
- Determinar cuáles procedimientos ágil son factibles utilizar en la producción de software en la Facultad 7, de acuerdo con las características de los proyectos que allí se desarrollan, quedando como resultado una propuesta metodológica de los procedimientos ágiles a seguir.

RECOMENDACIONES

Como resultado de la investigación y elementos a tener en cuenta, se hacen las siguientes recomendaciones a la dirección de la facultad, estas pueden ser desarrolladas en trabajos de diplomas en el curso 2007-2008:

- Dar continuidad a esta investigación para cumplir con la implantación de los procedimientos ágiles propuestos.
- Diseñar un curso optativo mediante el cual sean preparado los estudiantes para la utilización de estos procedimientos.

REFERENCIAS BIBLIOGRÁFICAS

[1] ÁGIL, A., 2006 Disponible en: http://www.agilealliance.org

[2] Ídem a [1]

[3] TRADUCIDO, G. C. T. D. I. Y. *El Manifiesto ágil*. Disponible en: http://www.sdmagazine.com/

[4] Idem [3]

[5] Idem [3]

[6] Idem [3]

[7] Idem [3]

[8] Idem [3]

[9] Idem [3]

[10] THOMAS, D. *The Pragmatic Programmer*. Addison Wesley, 1999. p. coautor

[11] Idem [3]

[12] Ídem [3]

[13] Ídem [3]

[14] Ídem [3]

[15] PETROSKI, H. *La Evolución de las Cosas Útiles*. Vintage Books, 1994. p.

[16] Idem [3]

[17] ELISA GALLO, M. V., 2006. Disponible en: http://www.esi.es/Berrikuntza

[18] BECK, K. *Extreme Programming Explained. Embrace Change*. Pearson Education, 1999. p. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

[19] PRESSMAN, R. *Ingeniería de Software un enfoque practico*. 5ta p.

[20] Idem [19]

[21] Idem [19]

[22] FOWLER, M. *Is Design Dead?*, 2001.Disponible en: www.martinfowler.com/articles/designDead.html

[23] FOWLER, M., BECK, K., BRANT, J. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. p.

[24] MARTIN, R. *Continuos Care vs. Initial Design*, 2002. [Disponible en: www.objectmentor.com/resources/articles/Continuous_Care.pdf.

[25] COCKBUN, A., WILLIAMS, L. *The Costs and Benefits of Pair Programming*. Humans and Technology Technical Report, 2000. p.

[26] Idem [22]

[27] JEFFRIES, R., ANDERSON, A., HENDRICKSON, C. *Extreme Programming Installed*. Addison-Wesley. , 2001. p.

[28] Idem [18]

BIBLIOGRAFÍA

BALAKISNAN, S. K., G. . *Resolving Gathering of User Requirements – A Lightweight Methodology Approach in the Development of Medium Maturity Level Web Sites*, 2006. [2006]. Disponible en: http://www.ucti.edu.my/wps/issue1/wp-06-04-paper.pdf

De los Métodos Heterodoxos en la construcción de Software. Disponible en: http://www.willydev.net/descargas/Heterodoxia.pdf

Desarrollo de Software Orientado a los Negocios con Métodos Ágiles. 2006]. Disponible en: http://www.agileshift.cl/Tutorial/DesarrolloAgilParte2.pdf#search=%22%20%22feature%20driven%20development%22%22

HARRISON, N. B. *A Study of Extreme Programming in a Large Company*, 2006]. Disponible en: http://www.agilealliance.org/system/article/file/1292/file.pdf

HERNÁN., S. M. *Diseño de una Metodología Ágil de Desarrollo de Software.* Buenos Aires, Universidad de Buenos Aires., 2004. p.

Herramienta que implementa eXtreme Programming para la gestión de requisitos. Disponible en: http://www.sqs.es/documentos/eXtreme.pdf#search=%22definici%C3%B3n%20%22metodolog%C3%ADa%20XP%22%22

JOSÉ H. CANÓS, P. L. Y. C. P. Metodologías Ágiles en el Desarrollo de Software.

PENADÉS, P. L. Y. M. C. *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*, 2006]. Disponible en: http://www.willydev.net/descargas/masyxp.pdf

RIEHLE, D. *A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn from Each Other*, 2006]. Disponible en: http://www.riehle.org/computer-science/research/2000/xp-2000.pdf

RUÍZ, J. A. B. *Feature Driven Development (Desarrollo Guiado por Funcionalidad*)*, 2006]. Disponible en: http://www.di.uniovi.es/~claudio/isoft/recursos/ProgExtrema6.pdf#search=%22%20%22feature%20driven%20development%22%22

SOFTHOUSE. *Scrum in five minutes*. Disponible en: http://www.softhouse.se/Uploades/Scrum_eng_webb.pdf

ANEXOS

Anexo 1 Guión de la entrevista.

1- Dentro de los tipos de proyectos existentes, ¿Dónde ubicaría usted al proyecto?

2- ¿Cuántas personas conforman el equipo de desarrollo?

3- ¿Cuáles son los roles existentes en el proyecto?

4- ¿Qué entorno de desarrollo tiene el proyecto?

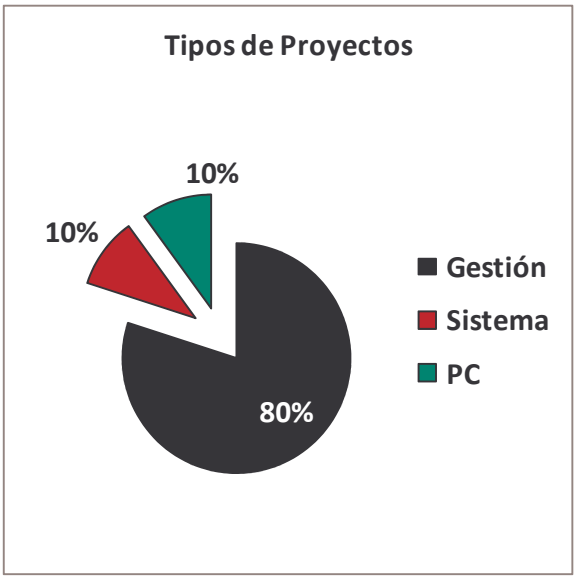
5- ¿Cómo caracterizaría usted al proyecto en cuanto a envergadura, tiempo de desarrollo y complejidad?

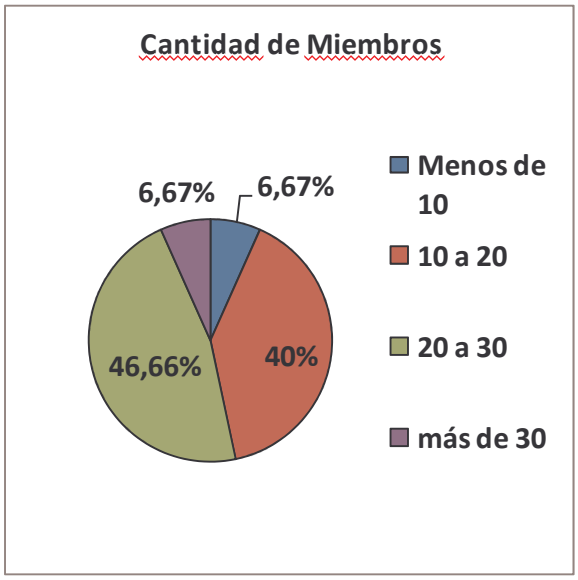
6- ¿Cómo se organiza el proyecto en cuanto a la planificación (equipo de desarrollo)?

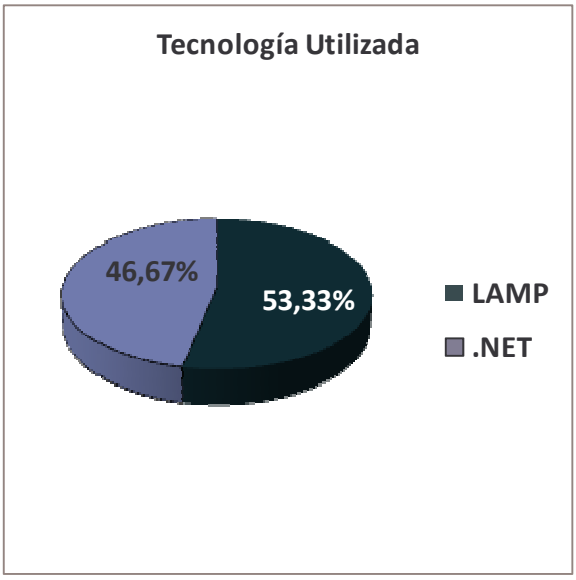
7- ¿Qué metodología de desarrollo de software utilizan en el proyecto para guiar el proceso?

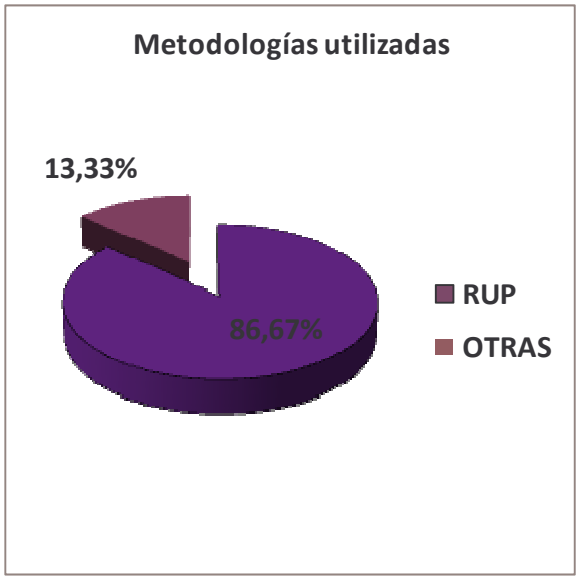
8- ¿Podría decirme los principales problemas que se han presentado a lo largo del desarrollo del proyecto?

Anexo 2 Gráficos de análisis de los proyectos.

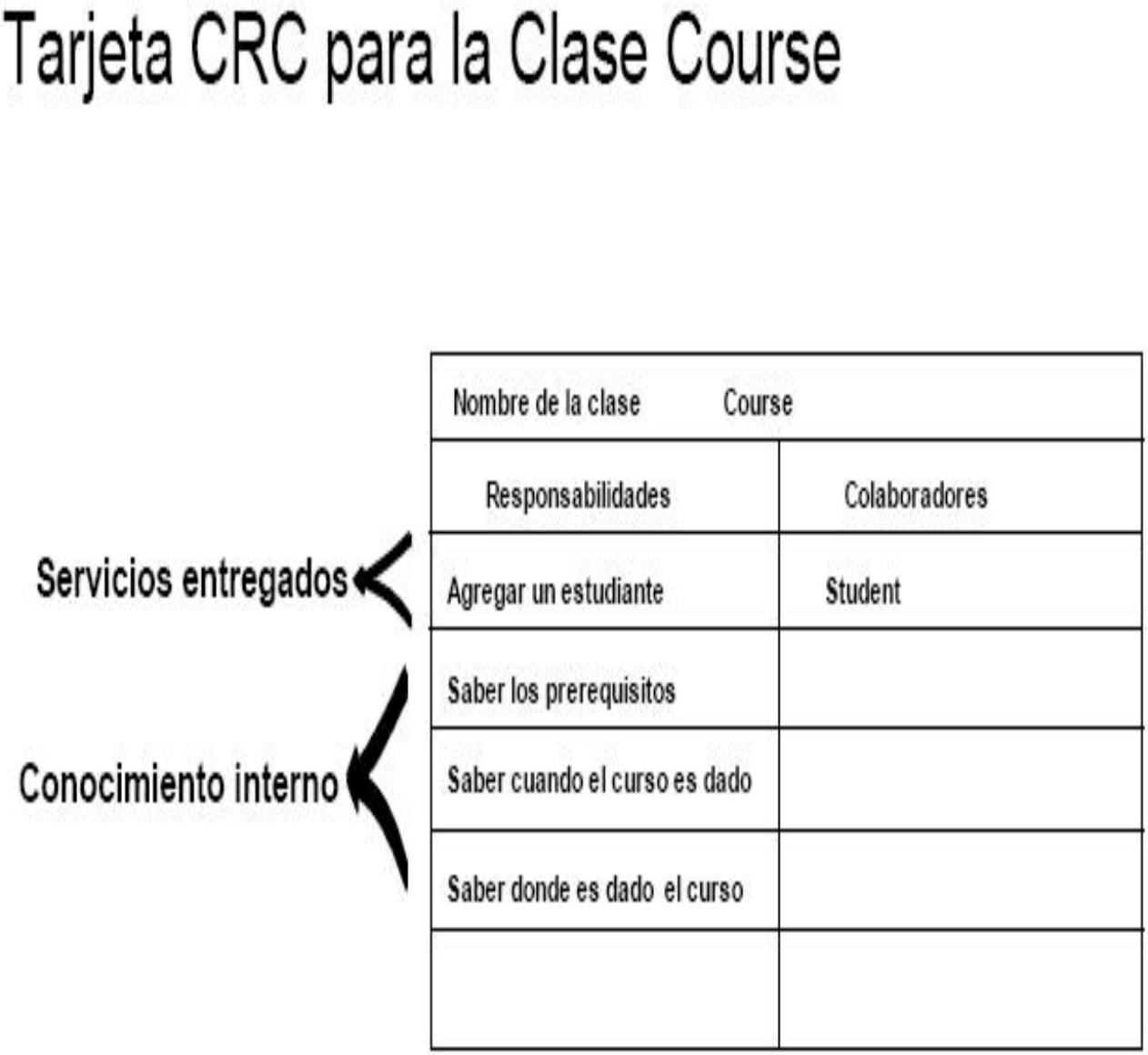








Anexo 3 Tarjeta CRC (Ejemplo)



Anexo 4: Plantilla caso de prueba XP

Caso de Prueba de Aceptación	
Código Caso de Prueba:	**Número Historia de Usuario:**
Descripción de la Prueba:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

Anexo 5: Plantilla Historia de usuario

Historia de Usuario	
Número:	**Nombre Historia de Usuario:**
Modificación de Historia de Usuario Número:	
Usuario:	**Iteración Asignada:**
Prioridad en Negocio: (Alta / Media / Baja)	**Puntos Estimados:**
Riesgo en Desarrollo: (Alto / Medio / Bajo)	**Puntos Reales:**
Descripción:	
Observaciones:	

Anexo 6: Plantilla de Registro de Tiempos

Registro de Tiempos

Proyecto:

Equipo :

Integrante:

Fecha (dd/mm)	Rol	Hora Inicio	Hora Fin	Tiempo Interrupciones (en horas)	Actividad Realizada*

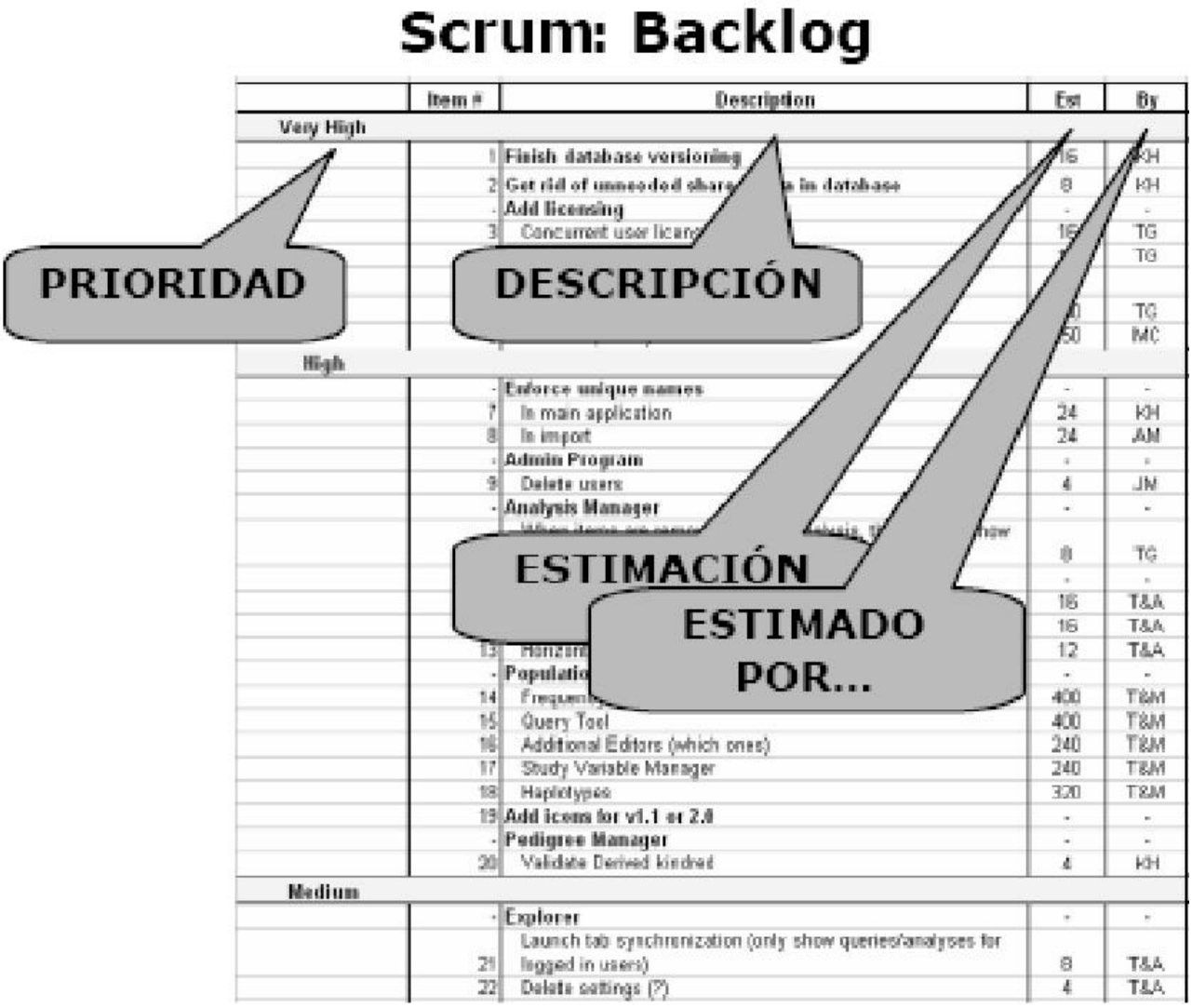
*Actividades realizadas:

Planificación, Requisitos (captura, especificación, validación), Diseño, *Refactoring*, Programación (sin incluir refactoring ni corrección de defectos), Integración, Pruebas (especificación y ejecución), Corrección de defectos, Seguimiento del proyecto, Aprendizaje (de herramientas o lenguajes), Otras (indicar cuál).

Anexo 7: Plantilla Tarea de ingeniería

	Tarea de Ingeniería
Número Tarea:	**Número Historia de Usuario:**
Nombre Tarea:	
Tipo de Tarea : Desarrollo / Corrección / Mejora / Otra (especificar)	**Puntos Estimados:**
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción:	

Anexo 8: Lista de Reserva del Producto (LRP)



	Item #	Description	Est	By
Very High				
	1	Finish database versioning	16	KH
	2	Get rid of unneeded share[illegible] in database	8	KH
	-	Add licensing	-	-
	3	Concurrent user licens[illegible]	16	TG
				TG
			[illegible]0	TG
			[illegible]50	MC
High				
	-	Enforce unique names	-	-
	7	In main application	24	KH
	8	In import	24	JM
	-	Admin Program	-	-
	9	Delete users	4	JM
	-	Analysis Manager	-	-
		[illegible]	8	TG
			-	-
			16	T&A
			16	T&A
	13	Horizon[illegible]	12	T&A
	-	Populatio[illegible]	-	-
	14	Frequen[illegible]	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	Add icons for v1.1 or 2.0	-	-
	-	Pedigree Manager	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	Explorer	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

GLOSARIO DE TÉRMINOS

Agilidad: Nos dice el diccionario que es una calidad de un ágil o persona ágil. Definición de ágil: Ligero, pronto, expedito, dícese de la persona que mueve o utiliza sus miembros con soltura.
También podemos describir a la agilidad como una combinación de flexibilidad, velocidad y elasticidad. En el contexto de la investigación es la habilidad de responder de forma versátil al cambio para maximizar los beneficios.

Artefactos: En tecnología, es un dispositivo concebido y fabricado, sea de modo artesanal o industrial, por una o más personas.

Builds: Estructuras.

Calidad: La palabra calidad tiene múltiples significados. La calidad de un producto o servicio es la percepción que el cliente tiene del mismo. Es una fijación mental del consumidor que asume conformidad con un producto o servicio determinado, que solo permanece hasta el punto de necesitar nuevas especificaciones. La calidad es un conjunto de propiedades inherentes a un objeto que le confieren capacidad para satisfacer necesidades implícitas o explícitas.

Ciclo de vida: Es un proceso por el cual los analistas de sistemas, los ingenieros de software, los programadores y los usuarios finales elaboran sistemas de información y aplicaciones informáticas.

CMM: Por sus siglas en Ingles, en español, Modelo de Capacidad y madurez. Es un modelo de evaluación de los procesos de una organización.

Desarrollo evolutivo: Forma en la que la especificación y el desarrollo están intercalados. Cuenta con tres actividades concurrentes: Especificación, Desarrollo y Validación.

Desarrollo incremental: Forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Es una combinación del Modelo de Cascada y Modelo Evolutivo.

Glosario de términos

Estándares: Es una especificación que regula la realización de ciertos procesos o la fabricación de componentes para garantizar la interoperabilidad.

Gurús: Une la experiencia de una veintena de profesionales de la comunicación, tan conocedores como apasionados del mundo de las nuevas tecnologías de la información e Internet. Son un equipo especializado en generar contenidos multiplataforma que lleva desarrollando proyectos de este tipo desde hace más de siete años.

Gurú [Indio]

1. Maestro con capacidad para orientar en el camino de la liberación, que vive tan solo para transmitir sus conocimientos.
2. En Occidente, persona respetada por dominar una materia.

Herramientas: Son los ambientes de apoyo necesario para automatizar las prácticas de Ingeniería de Software.

Hitos: Aptitudes que pueden ser identificadas y que sirven de guía para el desarrollo normal.

Iteraciones: En el contexto de un proyecto se refieren a la técnica de desarrollar y entregar componentes incrementales de funcionalidades de un negocio. Una iteración resulta en uno o más paquetes atómicos y completos del trabajo del proyecto que pueda realizar alguna función tangible del negocio. Múltiples iteraciones contribuyen a crear un producto completamente integrado.

Métodos: Son las maneras que se efectúan las tareas de Ingeniería de Software o las actividades del ciclo de vida.

Metodología ágil: Nuevo enfoque metodológico orientado a la gente y los resultados

Metodología de desarrollo: Es una versión amplia y detallada de un ciclo de vida COMPLETO de desarrollo de sistemas que incluye: Reglas, procedimientos, métodos, herramientas, funciones individuales y en grupo por cada tarea, productos resultantes, normas de Calidad

Glosario de términos

Metodologías tradicionales: Metodologías basadas en procesos.

Procedimiento: Son los mecanismos de gestión que soportan a los métodos: El control de los proyectos, el control de la calidad.

Proceso de software: Es un proceso complejo que involucra diversas tareas de gestión y desarrollo. Como resumen de las etapas para la creación de un software, se pueden mencionar: Análisis, Desarrollo, Construcción, Pruebas (unitarias e integradas), Paso a Producción.

Prototipo: Un prototipo es una representación limitada del diseño de un producto que permite a las partes responsables de su creación experimentar su uso, probarlo en situaciones reales y explorar su uso. Un prototipo puede ser cualquier cosa, desde un trozo de papel con sencillos dibujos a un complejo software.

Proyecto de desarrollo: Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto.

Requisitos: Capacidades, condiciones o cualidades que el sistema debe cumplir y tener.

Sprint: Carrera breve a todo correr. Correr velozmente.