Universidad de las Ciencias Informáticas
Facultad 4

Título: Métrica para el estado de avance del flujo de implementación del SIGEP

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor(es): Dailys Díaz Fuentes

Rosario Rodríguez Torres

Tutor: Ing. Arturo César Arias Orizondo

Junio del 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 ____ días del mes de _____ del año _____.

Dailys Díaz Fuentes

_____
Firma del Autor

Rosario Rodríguez Torres

_____
Firma del Autor

Ing. Arturo César Arias Orizondo

_____
Firma del Tutor

DATOS DE CONTACTO

Ing. Arturo Cesar Arias Orizondo

Profesor Instructor

Graduado en el 2003 de Ingeniero Informático del Instituto Superior Politécnico José Antonio Echeverría (ISPJAE), Título de Oro, promedio 5 puntos.

Ha impartido las asignaturas de Sistemas de Bases de Datos, Ingeniería de Software I y II y Gestión de Software en la Universidad de Ciencias Informáticas, participando en la elaboración de los programas de estas asignaturas.

Miembro del tribunal de acreditación de competencias del Departamento de Ingeniería de Software durante los cursos 2004-2005, 2005-2006.

Vicedecano de producción e investigación de la UCI por dos cursos: 2004-2005, 2005-2006.

Miembro del consejo científico de la UCI.

Miembro de la comisión de carrera durante los cursos 2004-2005, 2005-2006.

Ha impartido cursos de postgrado sobre RUP y UML.

Participó en el Congreso y Feria TecnoInternet 2004 celebrado en Santiago de Chile, Chile.

Ha participado en evaluaciones de calidad de productos de software para Venezuela.

Diseñador de base de datos del proyecto SAFRE, primero de exportación de la UCI.

Líder del proyecto de desarrollo de software para la gestión del sistema penitenciario venezolano desde el 2005.

AGRADECIMIENTOS

A mis padres por darme la vida, por tanto amor y apoyo, por ser luz guía, por contagiarme de sus mayores fortalezas y ayudarme a construir mis sueños. A mami porque sé que solamente lo que viene de ti es verdadero y eterno, que además de ser preciosa, eres el ser más dulce que existe sobre la faz de la Tierra... (Ojalá, fuera como tú)... A pipo porque serás siempre mi inspiración para alcanzar mis metas, por enseñarme que todo se aprende y que todo esfuerzo es al final recompensa. Por ser los mejores del Mundo.

A mi hermano querido por estar siempre ahí, en las buenas y en las malas, por haber enriquecido mi vida con su alegría, por ser parte de mí.

A mi tía Rosario por creer siempre en mí y apoyarme incondicionalmente, por su increíble fortaleza y su gran corazón.

A mis abuelos por enseñarme los primeros pasos, por darme siempre su amor y cariño.

A toda mi familia por recordarme que existen personas valiosas en el mundo.

A Liem por entregarme tanto amor, comprensión y apoyo, por estar a mi lado compartiendo alegrías y tristezas, por ser parte esencial en mi vida.

A mi hermanita Dailys por saber cuidar esta amistad y hacerla demasiado especial.

A mis amigos y amigas por todo su apoyo, por darme días llenos de risas, por entregarme sus hombros para llorar y por dejarme compartir un poquito de cada uno de ustedes.

A mi tutor por su tiempo y dedicación.

Rosario

A mi mamá, sabiendo que jamás existirá una forma de agradecer una vida entera de amor y dedicación, deseo que sepas que este logro, es tuyo, que mi esfuerzo es inspirado en ti, y que mi único ideal eres tú.

A mi papá de la vida, Alfon, por todos estos años de amor, por ser el apoyo paternal de la niña de ayer y la muchacha de hoy, por soportarme… porque eres parte de este logro.

A tata, por su dulzura y su presencia en mi vida…el camino es largo y lleno de reveses, pero “NO TE RINDAS, LUCHA”.

A mis padres de cuna, mis abuelos, por sus noches de desvelos y su amor infinito… una de sus ilusiones realizadas.

A mi familia toda, por lo que ha sido y será, porque más linda es imposible.

A Josué, por su mitad, por su amor en todo momento, por entenderme y apoyarme, por una vida a mi lado.

A Magalys, por estar siempre al tanto y demostrarme su cariño, porque ocupa un lugar en mi familia y en mi corazón.

A mis amigos de ayer y de hoy, a todos los que han dado lo mejor de sí mismos sin esperar nada a cambio, por saber escuchar y brindar ayuda cuando es necesario, porque hace mucho ocupan un lugarcito en mi corazón.

A Shary, por regalarme su amistad y convertirse en una hermana más.

A mi tutor, por dedicarnos un espacio de su valioso tiempo y hacer posible este trabajo.

A todos mis maestros por su granito de arena.

A los que olvido.

Dailys

DEDICATORIA

A mis padres y mi hermano, con todo el amor de mi corazón.
A los seres más queridos entre mis seres queridos.

Rosario

A mi mamá, que es mi vida y mi gran amor.
A Alfon y a tata, sus “ladronzuelos”.
A todos los que han soñado con este momento.

Dailys

RESUMEN

Existe en la UCI[1] problemas con el control y seguimiento de los proyectos. Las causas fundamentales de estos problemas están dadas porque no existen tareas definidas, ni relaciones entre ellas; no se realizan seguimientos del progreso de las tareas y no se cuenta con datos históricos de proyectos ya finalizados que permitan comparar un proyecto nuevo con los anteriores, para acertar mejor en las estimaciones y lograr un buen control sobre estos. Como consecuencia, los problemas no se detectan a tiempo y los proyectos concluyen, casi siempre, fuera de fecha.

Uno de los flujos más importantes dentro del ciclo de desarrollo es el flujo de implementación, debido a esto existen muchas personas interesadas en conocer su avance. Muchas veces la información que se brinda no es objetiva, por esta razón el presente trabajo se centra en diseñar una métrica que permita al líder de proyecto cuantificar el estado de avance del software durante la implementación.

La novedad científica del trabajo está dada por el diseño de una métrica original para controlar y dar seguimiento al flujo de implementación de un proyecto de software y mantener informados a todos los interesados en el progreso del software.

El valor práctico de la investigación consiste en disponer de una métrica para la gestión de proyectos adaptada a las características de la UCI, en la cual los resultados de su aplicación servirían de base para futuras estimaciones y planificaciones y los resultados del avance tendrían una base objetiva basada en la medición del software.

PALABRAS CLAVE

Estado de avance, flujo de implementación, métrica, estimación, planificación, medición.

[1] UCI: Universidad de las Ciencias Informáticas

ÍNDICE

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI) se adentra cada vez más en la producción de software empresarial con el reto de convertir a la industria de software cubana en un renglón fundamental de la economía del país. Una de sus misiones es la producción de software y servicios informáticos a partir de la vinculación estudio-trabajo como modelo de formación, logrando una fuerte relación Universidad-Empresa.

Como universidad desarrolladora de software que pretende insertarse en el mercado internacional, la UCI debe lograr que la planificación se convierta en un elemento esencial a tener en cuenta en el proceso productivo, para obtener productos de alta calidad.

La gestión de proyectos de software es una actividad protectora dentro de la ingeniería del software, empieza antes de iniciar cualquier actividad técnica y continúa a lo largo de la definición, del desarrollo y del mantenimiento del software. (PRESSMAN 1998)

La actividad de gestión del proyecto comprende medición y métricas, estimación, análisis de riesgos, planificación del programa, seguimiento y control. (PRESSMAN 1998)

En la UCI la planificación de proyectos de software es una tarea que se consolida con la práctica. Como universidad joven, el poco tiempo de experiencia no alcanza aún a obtener una base de registros históricos que permitan plantear una estrategia común para estimar y planificar con precisión todos los proyectos.

A pesar de existir varios sistemas de métricas, probadas y aceptadas en cientos de instituciones y empresas prestigiosas del mundo con resultados importantes, en la UCI no se cuenta con un sistema de métricas adaptado a sus características que permita evaluar el avance de los proyectos de software de gestión. Por tanto, la institución necesita utilizar un sistema de medida que le permita cuantificar los procesos asociados a la gestión de proyectos.

Ahora bien, para poder triunfar en este sector y lograr un servicio óptimo resulta imprescindible que estos softwares cuenten con la calidad requerida. Existen diversas formas de gestionar esta calidad, una de ellas, -quizás la más objetiva- es haciendo uso de las métricas del software.

Las métricas no son más que medidas o colecciones de datos de las actividades de los proyectos y recursos. Estas deben ser simples, objetivas, fáciles de coleccionar, fáciles de interpretar y difíciles de

malinterpretar. (RUP 2003) En la actualidad el uso de las métricas se extiende rápidamente entre la comunidad de desarrolladores de software debido a que las mismas producen indicadores a partir de los cuales se pueden tomar decisiones importantes.

Gran parte del software que se desarrolla en la UCI es del tipo de gestión, que se define como aplicaciones informáticas o programas informáticos diseñados para facilitar al usuario la realización de un determinado tipo de trabajo.

Uno de estos proyectos es el Sistema de Gestión Penitenciaria (SIGEP), el cual tiene como propósito dar respuesta a las necesidades de gestión, información y apoyo a la toma de decisiones de la Dirección General de Custodia y Rehabilitación del Recluso (DGCRR) de Venezuela, por lo que tiene como objetivo, desarrollar e implantar un sistema informático que soporte las decisiones estratégicas del Ministerio del Interior y Justicia y de la Dirección General de Custodia y Rehabilitación del Recluso.(ARIAS 2006)

En el proceso de desarrollo del SIGEP, existen muchas personas interesadas en conocer su avance (clientes, directivos). Por lo general la información que se brinda tiene carácter subjetivo y no cuenta con un alto grado de precisión. Por tanto, es necesario emplear un método que permita medir dicho avance a partir del cual se puedan tomar las decisiones gerenciales necesarias que se deriven de esa información.

Hasta el momento no existe una métrica que se ajuste al flujo de trabajo que rige la etapa de implementación del proyecto SIGEP, teniendo en cuenta que la medición de las actividades depende de la estructura arquitectónica de cada proyecto y de su equipo de desarrollo, y que la métrica si bien puede tener un principio general y aplicable a cualquier proyecto, sus parámetros deben reflejar las características del proyecto y del equipo de desarrollo.

Toda esta situación lleva a definir que el **problema científico** de la investigación corresponde a la necesidad de proponer y aplicar una métrica que permita cuantificar el estado de avance de la etapa de implementación del proyecto SIGEP.

Partiendo del problema planteado, el **objeto de estudio** se enfocará hacia la planificación de proyectos de software de gestión en la UCI.

El **campo de acción** lo constituyen las métricas utilizadas para medir el avance en la etapa de implementación.

Para resolver el problema planteado, se propone como **objetivo general**: Proveer a los interesados la información precisa sobre el estado de avance del SIGEP en su etapa de implementación, mediante la creación e implantación de una métrica que lo permita cuantificar.

Partiendo de lo antes expuesto esperamos como resultado: obtener una métrica que permita conocer el estado de avance de la implementación de un proyecto de software. La evaluación de los resultados de su aplicación serviría de base para futuras estimaciones y planificaciones. Los resultados del avance tendrían una base objetiva y no dependería de la subjetividad de la dirección del equipo de desarrollo. La métrica sería adaptable teniendo en cuenta que se basa en factores de ponderación que podrían ser calculados teniendo en cuenta las características de cada proyecto.

Para lograr el objetivo trazado se llevaron a cabo las siguientes **tareas de la investigación**:

- Identificar los métodos de estimación existentes relacionados con el tamaño, tiempo, esfuerzo y avance de proyectos.
- Identificar las tareas de implementación que se ejecutan en el proyecto SIGEP según su estructura arquitectónica.
- Definir los parámetros de los cuales depende la complejidad de las tareas de la implementación.
- Describir cómo se calcula el por ciento de avance en la etapa de implementación.
- Diseñar la métrica de avance de la implementación.
- Explicar la solución matemática de la métrica.
- Validar la métrica en la práctica.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Haciendo un estudio retrospectivo en la historia de la industria del software, se pudiera iniciar caracterizando los años 1960 como la era donde se aprendió a explotar la información tecnológica (KAN 2000), comenzando a vincular los software con las operaciones diarias de las instituciones que ya para los años 1970 registraban demoras masivas en los cumplimientos de las planificaciones y el exceso de los costos, enfocando todas las actividades a partir de entonces en torno a la planificación.

Para los 1980, la productividad incrementaba significativamente, disminuía el costo y la competencia en la industria crecía considerablemente. Venía madurándose la idea de predecir, analizar y evaluar los distintos atributos y características de los productos y procesos, que participan en el desarrollo y mantenimiento del software, mediante métricas e indicadores que expresaran la realidad lo más cuantificada posible y con mejoras continuas e incrementales que se refinarían a lo largo de los 1990 definida como la era de la calidad. (KAN 2000) A partir de entonces la competencia por obtener un producto óptimo y refinado comienza a exigir más precisión a la hora de estimar para planificar y medir cuánto esfuerzo, recursos, y tiempo supondrá construir un sistema o producto específico de software, durante la gestión de proyectos.

1.2. Planificación en la gestión de proyectos

La planificación, en el mundo de la Industria del Software, se ha convertido en uno de los principales retos para la gestión de proyectos y una actividad fundamental para desarrollar software de alta calidad.

Comprender el ámbito del trabajo a realizar, los riegos en los que se puede incurrir, las tareas que han de llevarse a cabo, las etapas a recorrer, los recursos y el coste del proyecto, así como el plan a seguir, previéndose las posibles situaciones de encontrar escenarios con el mejor y peor de los casos, representan acciones esenciales para conseguir un alcance exitoso en un proyecto de software.

Precisamente, la gestión de proyectos de software, es la encargada de proporcionar este conocimiento, comenzando antes de iniciar el proceso de desarrollo, evolucionando a medida que se va de un nivel conceptual a la realidad y finaliza en el momento que se abandona el software.

La planificación va sucediendo como un proceso de descubrimiento de la información que de lugar a realizar estimaciones más razonables a medida que se avanza.

Los proyectos bien ejecutados pasan por tres etapas básicas para crear la planificación del software. Primero, se estima el tamaño del producto, luego el esfuerzo necesario para construir un producto con este tamaño y por último la duración cronológica del proyecto. (BRITO y DIAZ 2007)

Una planificación adecuada requiere:

- Todas las actividades bien definidas en el método.
- El esfuerzo y el tiempo se asigne inteligentemente a cada actividad.
- Las relaciones entre las actividades indicadas correctamente.
- Los hitos situados rigurosamente espaciados para que se pueda seguir el progreso.

El objetivo de la planificación se logra mediante un procesamiento de la información que lleve a estimaciones razonables. Es por esto que se requiere de:

- Experiencia, lógica, sentido común: casos anteriores, proyectos similares, etc.
- Técnicas de estimación: a partir de determinados parámetros se predeterminan variables a estimar como el costo, el esfuerzo y el tiempo necesarios para obtener el software. Donde el esfuerzo se traduce al total de tiempo que gasta una persona trabajando en el desarrollo del proyecto de software (horas persona / mes persona).

Para llevar a cabo la planificación de un proyecto es necesario seguir una serie de pasos sustentados por un conjunto de actividades organizativas y de estimación, donde se generan artefactos fundamentales para llevar un seguimiento continuo y tener un control pleno de las tareas a realizar en un proyecto de producción de software. A continuación se describen las actividades asociadas a la planificación de proyectos software:

1.2.1. Ámbito del software que se va a desarrollar

El ámbito del software es la primera actividad llevada a cabo durante la planificación del proyecto de software. (PRESSMAN 1998)

En esta actividad el desarrollador del software y el cliente se encargan de definir el ámbito y los objetivos del proyecto. El ámbito se define como: un pre-requisito para la estimación y la obtención de la información necesaria, identifica las funciones primordiales que debe llevar a cabo el software y, lo que es más importante, intenta limitar esas funciones de manera cuantitativa. Los objetivos identifican los fines globales del proyecto sin considerar cómo se llegará a esos fines.

Una vez entendidos los objetivos y el ámbito del proyecto, se han de considerar soluciones alternativas. Aunque se estudien con muy poco detalle, las alternativas han de permitir seleccionar el mejor enfoque, dadas las restricciones impuestas por las fechas tope de entrega, los límites presupuestarios, la disponibilidad de personal, las interfaces técnicas y multitud de otros factores.

1.2.2. Estimación de los recursos requeridos

La segunda actividad de la planificación de proyectos de software es la estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software.

La estimación de recursos, costos y planificación temporal de un proyecto requiere experiencia, una buena información histórica y confiar en medidas cuantitativas cuando todo lo que se conoce son datos cualitativos. La estimación conlleva un riesgo que lleva a la incertidumbre.

- La complejidad del proyecto tiene un gran efecto en la incertidumbre, presente en toda planificación y, es una medida relativa que se ve afectada por la familiaridad con esfuerzos anteriores (experiencia adquirida en proyectos anteriores).

- El tamaño del proyecto es otro factor importante que puede afectar a la precisión de las estimaciones. Conforme aumenta el tamaño, crece rápidamente la interdependencia entre varios elementos del software.

El grado de incertidumbre en la especificación de requisitos y la disponibilidad de información histórica tienen efecto en el riesgo de la estimación.

Recursos de desarrollo de un proyecto:

- Recursos humanos: Es el conjunto de capital humano que está bajo el control de la empresa en una relación directa de empleo, en este caso personas, para resolver una necesidad o llevar a cabo cualquier actividad en dicha empresa.

- Componentes de software reutilizables: Bloques de software que pueden reducir drásticamente los costos de desarrollo y acelerar la entrega.

- Herramientas Hardware/Software: Proporcionan la infraestructura de soporte al esfuerzo de desarrollo.

Características específicas de los recursos:

- Descripción del recurso: Se detallan las características específicas del recurso.

- Informes de disponibilidad: Se especifica el tiempo disponible en el que puede ser usado el recurso.

- Fecha cronológica en la que se requiere el recurso: Se especifica la fecha en orden cronológico, en la cual se necesita hacer uso del recurso.

- Tiempo durante el que será utilizado el recurso: Se especifica el tiempo de uso del recurso.

Recursos humanos:

Los recursos humanos representan el conjunto del capital humano que está bajo el control en un proyecto determinado en una relación directa de empleo, en este caso personas, para resolver una necesidad o llevar a cabo cualquier actividad en un proyecto.

Para llevar a cabo la organización del personal de un proyecto es necesario tener controlada toda la gestión de recursos humanos, la cual se proyecta desde perspectivas más amplias de incorporar ideas relacionadas con el desarrollo de la organización y la calidad de vida en el trabajo.

Recursos o componentes de software reutilizables:

Cualquier estudio sobre recursos de software estaría incompleto sin estudiar la reutilización, que se basa en la creación y la reutilización de bloques de construcción de Software.

Existen cuatro categorías de recursos de software que se deben tener en cuenta a medida que avanza la planificación:

- Componentes ya desarrollados: El software existente se puede adquirir de una tercera parte o provenir de uno desarrollado internamente para un proyecto anterior. Llamados componentes

CCYD (componentes comercialmente ya desarrollados), estos componentes están listos para utilizarse en el proyecto actual y se han validado totalmente.

- Componentes ya experimentados: Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual. Los miembros del equipo de software actual ya han tenido la experiencia completa en el área de la aplicación representada para estos componentes. Las modificaciones, por tanto, requeridas para componentes de total experiencia, tendrán un riesgo relativamente bajo.

- Componentes con experiencia parcial: Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial. Los miembros del equipo de software actual han limitado su experiencia sólo al área de aplicación representada por estos componentes. Las modificaciones, por tanto, requeridas para componentes de experiencia parcial tendrán bastante grado de riesgo.

- Componentes nuevos: Los componentes de software que el equipo de software debe construir específicamente para las necesidades del proyecto actual.

1.3. Herramientas de Planificación

Existen mundialmente un sin número de herramientas para la planificación, desde herramientas específicas desarrolladas particularmente para una empresa determinada, hasta herramientas generales que incluyen grandes módulos y aspectos específicos.

Entre estas últimas se destacan:

- Microsoft Project
- Gantt Project
- Gantt PV
- Dot Project
- Agile Track

1.3.1. Características del Microsoft Project

Es una herramienta flexible, eficaz y fácil de utilizar para la planificación y seguimiento de los proyectos. Permite controlar los proyectos, ayuda a mantener informados a quienes participan en ellos y es compatible con la forma en que se trabaja actualmente. Controla y supervisa el progreso de proyectos simples y complejos mediante la programación y el seguimiento de las actividades. Permite administrar más de un proyecto. Con la información que proporciona Microsoft Project (MS-PROJECT) se puede optimizar la programación de tareas y se establecen plazos realistas para completar estas. Permite además realizar un seguimiento del presupuesto del proyecto y de todo lo que esto conlleva.

Microsoft Project permite introducir y modificar la información de la línea base de la empresa. Para aprovechar el máximo de su experiencia, crea un plan que represente lo que se espera del proyecto. Después si se sigue el progreso real en el Microsoft Project, se puede utilizar la información de la línea base para comparar el plan original con el curso real del proyecto.

Microsoft Project es un sistema de control de progreso de proyectos generales, no sólo de proyectos de software, más bien adaptado a proyectos muy grandes. Su uso en proyectos de software pequeños tiene varios inconvenientes porque supone la desconexión entre el proceso de control del progreso del proyecto y el control de la calidad debido a que con el Project:

- Es imposible determinar medidas de calidad durante la ejecución de las tareas a no ser que estas sean introducidas en el campo de notas de las tareas manualmente.

- Existe dificultad para actualizar el progreso estimado de una tarea (en el tiempo), debiéndose introducir manualmente un valor porcentual de completamiento subjetivo en la tarea.

- Es muy difícil obtener información que esté esparcida en varios proyectos, por ejemplo: el tiempo total trabajado por una persona en un rango de tiempo específico; debido a que en Microsoft Project cada proyecto tiene su propio fichero independiente de los demás, lo cual trae como consecuencia la posibilidad de duplicidad de información.

Lo más importante del MS-PROJECT para la ayuda a la planificación de proyectos es que ofrece la posibilidad de mostrar el método CPM (Método del Camino Crítico) y el diagrama PERT (o red de actividades). Ambas técnicas desarrollan una descripción de la red de tareas del proyecto, es decir, una representación gráfica o tabular de las tareas que deben realizarse desde el principio hasta el final del proyecto. También ofrecen una gestión de recursos completa y un conjunto de diagramas y

mecanismos de control que permiten realizar una presentación brillante y muy profesional de la planificación de un proyecto.

Además del diagrama PERT, se suele utilizar una especie de diagrama temporal llamado diagrama de Gantt, el cual es un diagrama sencillo pero muy eficiente. Por tal razón, este diagrama es utilizado por la mayoría de las herramientas informatizadas de ayuda a la planificación, para introducir los datos de las diferentes tareas o actividades que forman la WBS (Work Breakdown Structure), que no es más que la descomposición estructural de los trabajos que se deben realizar, es decir, la lista estructurada de todas las actividades y tareas de un proyecto.

1.3.2. Características del Gantt Project

Gantt Project es una herramienta de uso libre con interfaz similar al MS-PROJECT pero mucho más ligera. Basa todo su contenido en una representación gráfica (Representación Gantt) para programar, organizar las tareas y asignar personas y recursos. Esta herramienta es tan completa que permite definir los días libres que tiene asignados cada trabajador, así como los generales.

Gantt Project es una utilidad que permite diseñar diagramas de Gantt al igual que MS-PROJECT para planificar todas las tareas y las actividades de un proyecto en el tiempo establecido, facilitando una visualización amena del estado de progreso de cada actividad.

Gantt Project está programado en Java y corre en entornos Windows y Linux, genera archivos XML pero permite generar otro tipo formatos como: jpg, png, html, pdf y csv.

1.3.3. Características del Gantt PV

Gantt PV es un programa gratuito, de apariencia sencilla y sin grandes complicaciones, para planificación de proyectos, descomposición, representación y seguimiento de tareas sobre diagrama de Gantt. Corre en entornos Windows, MacOS y Linux.

Gantt PV se utiliza principalmente para la gerencia de proyectos a través de las llamadas cartas Gantt, que luego de ser creadas son exportadas como Páginas Web.

1.3.4. Características del Dot Project

Dot Project fue creado por Dotmarketing.org en el año 2000, con el objetivo de construir una herramienta para la planificación, gestión y seguimiento de múltiples proyectos para clientes diferentes,

quienes pueden disponer también de acceso para monitorizar la evolución del desarrollo de dicho proyecto. Es una herramienta gratuita, construida por aplicaciones de código abierto.

Es una aplicación basada en Web, multiusuario, soporta varios lenguajes y es software libre. Está programada en PHP y utiliza MySQL como base de datos, aunque puede ser utilizado también el Postgres. La plataforma recomendada para utilizar Dot Project se denomina LAMP (Linux + Apache + MySQL + PHP). Esta herramienta utiliza el diagrama de Gantt para representar el seguimiento y control de sus actividades en un tiempo determinado.

1.3.5. Características de Agile Track

Es una herramienta de interfaz sencillo para planificación y seguimiento de proyectos. Se utiliza específicamente para el desarrollo de software en equipos reducidos con metodologías ágiles, especialmente “Extreme Programming”. Gestiona ciclos de desarrollo basados en iteraciones, con seguimiento de historias de usuario, tareas y bugs.

Es multiplataforma, puede ser utilizado en Windows y Linux sin ningún problema. Consta de dos módulos: el servidor que trabaja con MySQL, y el cliente para el seguimiento de los proyectos. Es un desarrollo Open Source, de uso gratuito con licencia AFL (Academic Free License), el cual usa las gráficas de Gantt para tener una visión del seguimiento del proyecto. Además, permite ejecutar una demo del cliente completamente operativa desde la Web del proyecto, sobre un navegador (Máquina Virtual Java).

Todas las herramientas antes mencionadas permiten llevar a cabo una planificación detallada de las actividades que se realizan en un proyecto de software, además posibilita visualizar su seguimiento, para así poder llevar un control pleno de estas tareas, para que el software se obtenga con la calidad requerida.

1.4. Estimación

La estimación de la duración de las actividades que conforman el desarrollo de software es un tema que concierne a la gestión y control de proyectos para asegurar el éxito.

La estimación, como actividad antecesora y constante de lo que luego será la planificación, proporciona valores aproximados de costos, tiempos y esfuerzos que se necesitarán para el desarrollo del producto a construir. Esa aproximación, requiere experiencia, acceder a una buena información histórica y el coraje de confiar en predicciones (medidas) cuantitativas cuando todo lo que existe son

datos cualitativos (PRESSMAN 1998); sin embargo, contar con dicha información en etapas tempranas de desarrollo, permite tomar decisiones importantes antes de llevarlo a cabo, es decir, que de cierta forma nos estamos atreviendo a predecir el futuro y así lo expresa también Ana Sánchez Capuchino definiendo la estimación como: el proceso que proporciona un valor, a un conjunto de variables para la realización de un trabajo, dentro de un "rango aceptable de tolerancia". (CAPUCHINO 1996)

En efecto, Ana María Moreno Sánchez Capuchino dice: La primera tarea en la gestión de proyectos es la estimación. (CAPUCHINO 1996)

Se han desarrollado varias técnicas de estimación para el desarrollo de software, que tienen en común los siguientes aspectos:

- Se ha de establecer de antemano el ámbito del proyecto.
- Como base para la realización de estimaciones, se usan las métricas del software.
- El proyecto se desglosa en partes más pequeñas que se estiman individualmente.

La estimación es siempre difícil de realizar por diversas razones, algunas de ellas son:

- No existe un modelo de estimación universal.
- Son muchas las personas implicadas en el proyecto, desde la alta dirección de la empresa a los ejecutivos del proyecto, que precisan de las estimaciones.
- La utilidad de una estimación varía con la etapa de desarrollo en que se encuentra el proyecto.
- Las estimaciones precisas son difíciles de formular, sobre todo al inicio del proyecto.
- La estimación suele hacerse superficialmente, sin tener en cuenta el esfuerzo necesario para hacer el trabajo.
- La rapidez del cambio de las metodologías y las tecnologías no permiten la estabilización del proceso de estimación.
- Los estimadores pueden no tener experiencias sobre aquello que pretenden estimar.

- El estimador suele hacer la estimación en función del tiempo que a él le llevaría en realizar el trabajo, sin tener en cuenta la experiencia y formación de la persona que realmente lo realiza.

- El estimador tiende a reducir en alguna medida sus estimaciones para hacer más aceptable su oferta.

1.5. Métodos de Estimación

En el marco de trabajo durante el proceso de desarrollo de software se clasifican los siguientes métodos de estimación:

-Empíricos.

-Analógicos.

-Teóricos.

-Heurísticos.

-Las estimaciones global y detallada.

-Juicio del experto.

-Método Puntos de Casos de Uso.

-Modelo COCOMO.

-Puntos de Función.

-COCOMO II y los Puntos de Función.

1.5.1. Empíricos

Cualquier estimación se debe basar en un modelo empírico, o sea, algo subjetivo producto de la experiencia, que relacione un atributo de interés con otros atributos mensurables. Este modelo empírico es el punto de partida para cada método de estimación.

1.5.2. Analógicos

Este método hace la estimación de un proyecto nuevo por analogía con las estimaciones de proyectos anteriores comparables y que estén terminados. En la analogía pueden variar factores como el tamaño, complejidad, usuarios. Utiliza medidas de los atributos del modelo empírico a fin de caracterizar el caso actual, para el que se realiza la estimación. Las medidas conocidas para el caso actual son usadas para buscar un conjunto de datos que identifiquen casos análogos. La predicción se hace intercalando desde uno o varios casos análogos al caso actual.

Las ventajas que proporciona este método es un menor costo en tiempo y recursos que el método del juicio del experto. Como desventajas cabría destacar que las estimaciones de proyectos anteriores no siempre se ajustan a nuevos proyectos, ya que muchos de los factores de estas estimaciones no siempre se mantienen. (ESCORIAL 2006)

1.5.3. Teóricos

Los métodos de estimación teóricos proponen un modelo numérico basado en el modelo empírico. Los modelos teóricos deben ser validados empíricamente, por comparación con los datos actuales de las medidas.

1.5.4. Heurísticas

Los métodos heurísticos suelen usarse como extensiones de otros métodos. Las heurísticas son reglas empíricas, desarrolladas mediante experiencia, que obtienen conocimiento acerca de relaciones entre atributos del modelo empírico. Las heurísticas se pueden utilizar para ajustar estimaciones realizadas con otros métodos.

1.5.5. Las estimaciones global y detallada

La estimación global, conocida también como descendente, se hace teniendo en cuenta las funcionalidades del producto, pasándose posteriormente al detalle. La estimación detallada o ascendente empieza por la estimación de los esfuerzos individuales, los cuales se suman para obtener el esfuerzo del proyecto. (ESCORIAL 2006)

1.5.6. Juicio del experto

Las opiniones de los expertos se basa principalmente en juicios emitidos por uno o varios expertos avalados por su experiencia en entornos similares y apoyados, en algunos casos, en datos objetivos

obtenidos de proyectos anteriores y almacenados (ESCORIAL 2006), aunque no se incluyen dentro del marco de trabajo para seleccionar métodos de estimación, ya que estos métodos no se pueden caracterizar fácilmente. Existen algunos métodos más específicos como el Experto Puro y el Wideband Delphies.

La desventaja de este método es el alto costo en tiempo y recursos humanos necesarios para su implantación, así como la subordinación al nivel de experiencia y conocimientos en el entorno que puedan aportar los técnicos. Las ventajas que posee indican que las estimaciones parciales son neutralizadas y se presenta una estimación global. Por otro lado las estimaciones suministradas por este grupo de expertos difícilmente pueden ser obviadas gracias a la trascendencia que la organización otorga a este proceso, al proporcionar costosos recursos a esta tarea. (ESCORIAL 2006)

Durante la práctica de los métodos mencionados, se han originado otros más formales y eficientes, reconocidos y utilizados por su nivel de precisión:

-Basados en el tamaño: Método de Putnam, Método COCOMO (Modelo de Construcción de Costo).

-Basados en las funciones: Puntos de Función, Puntos de casos de uso y Puntos de Objeto.

-Combinados: Puntos de Función y de Objetos con COCOMO.

1.5.7. Puntos de Casos de Uso

Este método estima el esfuerzo de desarrollo de un producto de software a partir de los casos de uso y algunos factores de complejidad técnica y ambiente que influyen en el desarrollo. Fue propuesto originalmente por Gustav Karner y posteriormente refinado por muchos otros autores.

Este método exige la existencia de un modelo de casos de uso, por lo que se deberá comenzar a aplicar, una vez que se tenga algún entendimiento del dominio del problema o cuando se estén realizando las labores de arquitectura y dimensionamiento del tamaño del sistema. (HERNÁNDEZ 2002)

El método utiliza los actores y casos de uso identificados para calcular el esfuerzo que costará desarrollarlos. A los casos de uso se les asigna una complejidad basada en transacciones, que son pares de pasos acción-usuario->respuesta-sistema de los escenarios de los casos de uso. A los actores se les asigna una complejidad basada en el tipo de actor, es decir, si son interfaces con

usuarios o si son interfaces con otros sistemas (API o Protocolo). También se utilizan factores de entorno y de complejidad técnica para afinar el resultado.

Una vez asignada la complejidad, se calculan los puntos de caso de uso no ajustados, el TCF (factor de complejidad técnica) y el EF (factor del entorno). Con ellos, se calculan los puntos de caso de uso o UCP, que finalmente se traducen a esfuerzo en horas-hombre con un sencillo cálculo.

Una de las principales desventajas de este método es que no existe una teoría de cómo escribir o estructurar correctamente los casos de uso, por lo que todas las medidas de tamaño y estimación serán afectadas por la rigurosidad de los analistas. (HERNÁNDEZ 2002)

1.5.8. Modelo COCOMO

COCOMO fue propuesto y desarrollado por Barry Boehm en 1981, es uno de los modelos de estimación de costo mejor documentado, estudiado y utilizado en la industria de software. El modelo permite, basándose en un grupo de ecuaciones no lineales obtenidas mediante técnicas de regresión a través de un histórico de proyectos ya realizados (MARCELO 2006); estimar el esfuerzo, costo y tiempo que se requiere en un proyecto de software a partir de una medida del tamaño del mismo, expresada en el número de líneas de código que se estimen generar para la creación del producto software. El modelo original ha evolucionado a un modelo más completo llamado COCOMO II.

1.5.9. Puntos de Función

El método de puntos de función fue creado por Allan Albretch y se basa principalmente en la identificación de los componentes del sistema informático en términos de transacciones y grupos de datos lógicos que son relevantes para el usuario en su negocio. A cada uno de estos componentes les asigna un número de puntos por función basándose en el tipo de componente y su complejidad; y la sumatoria de esto, da los puntos de función sin ajustar. El ajuste es un paso final basándose en las características generales de todo el sistema informático que se está contando.

Los objetivos de calcular Puntos de Función son:

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir atributos independientemente de la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica de tamaño que de soporte al análisis de la calidad y la productividad.

- Proporcionar un medio para la estimación del software.
- Proporcionar un factor de normalización para la comparación de distintos software.

El análisis de los Puntos de Función se desarrolla considerando cinco parámetros básicos externos del Sistema (PRESSMAN 1998):

- Entrada (EI, External Input).
- Salida (EO, External Output).
- Consultas (EQ, External Query).
- Ficheros Lógicos Internos (ILF, Internal Logic File).
- Ficheros Lógicos Externos (EIF, External Interface File).

Con estos parámetros, se determinan los puntos de función sin ajustar (PFsA), luego se utiliza COCOMO para calcular el esfuerzo de desarrollo y otros indicadores a partir de la conversión de los puntos de función sin ajustar en líneas de código fuente. A este valor, se le aplica un Factor de Ajuste obtenido en base a unas valoraciones subjetivas sobre la aplicación y su entorno; es decir, las características generales del sistema. Se debe aplicar el coeficiente de conversión de acuerdo a la experiencia para obtener el esfuerzo de desarrollo.

Podemos decir entonces que los puntos de función aparecen con ventajas substanciales por sobre las líneas de código, para fines de estimación temprana del tamaño del software, y por ende, del esfuerzo de desarrollo. Además es una medida ampliamente utilizada, y con éxito, en muchas organizaciones que desarrollan software en forma masiva. (HURTADO 2007)

1.5.10. COCOMO II y los Puntos de Función

Debido a la complejidad de los proyectos de software, el modelo original COCOMO, fue modificado, denominándose al modelo actual COCOMO II. El nuevo modelo permite determinar el esfuerzo y tiempo de un proyecto de software a partir de los puntos de función sin ajustar, lo cual supone una gran ventaja, dado que en la mayoría de los casos es difícil determinar el número de líneas de código de que constará un nuevo desarrollo, en especial cuando se tiene poca o ninguna experiencia previa en proyectos de software. Esto hace que ambos modelos, Puntos de Función y COCOMO sean

perfectamente compatibles y complementarios. Su triunfo depende ampliamente de la adaptación del modelo a las necesidades de la organización, usando datos históricos; los cuales no siempre están disponibles.

1.6. Mediciones

Cuando se planifica un proyecto de software es esencial hacer estimaciones: del esfuerzo humano requerido, de la duración cronológica del proyecto y del costo, a través de las mediciones de software.

Lord Kelvin en una ocasión dijo: *"Cuando pueda medir lo que está diciendo y expresarlo con números, ya conoce algo sobre ello; cuando no pueda medir, cuando no pueda expresar lo que dice con números, su conocimiento es precario y deficiente: puede ser el comienzo del conocimiento, pero en tus pensamientos apenas estás avanzando hacia el escenario de la ciencia"*

Es una afirmación indiscutible que las mediciones son cruciales en el progreso de todas las ciencias. El progreso científico se logra a través de observaciones y generalizaciones basadas en datos y mediciones y la derivación de teorías como resultado de estas. (KAN 2000)

Para lograr una buena calidad del producto es necesario identificar las mediciones y los criterios que serán utilizados para identificar el nivel deseado de la calidad y determinar si se está alcanzando, siendo los métodos de estimación y las métricas de software una de las vías más utilizadas para cuantificar y valorar resultados.

1.7. Métricas de software

El área de las mediciones de software, es una de las áreas en la ingeniería de software donde se ha investigado desde hace más de 30 años, esta área se conoce también como métricas de software. Existe una confusión en la utilización de los términos métricas de software y mediciones de software, sin embargo, en la literatura los términos métrica, medida o medición son usados como sinónimos. (ZUSE 1995)

Las métricas de software se pueden definir como: "La continua aplicación de técnicas basadas en la medición al proceso de desarrollo de software y a sus productos para proveer información administrativa significativa y oportuna, junto con el uso de esas técnicas para mejorar el proceso y sus productos". (WESTFALL 1995) La figura 1 ilustra una expansión de esta definición para enfatizar que las métricas de software proveen la información necesitada por los ingenieros para decisiones técnicas.

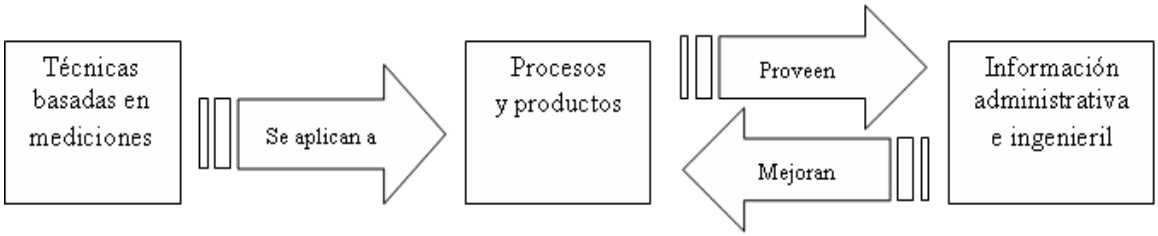

Figura 1 Métricas del software

Las mediciones de software son usadas para medir atributos específicos de un producto de software o del proceso de desarrollo de software. (NUSENOFF and BUNDE 1993) Se usan fundamentalmente para: (ZUSE 1995)

- Obtener las bases para la estimación.
- Seguir el progreso de los proyectos.
- Determinar la complejidad (relativa).
- Ayudar a comprender cuando se ha alcanzado un estado deseado de calidad.
- Analizar los defectos.
- Validar experimentalmente las mejores prácticas.

En resumen, ayudan a tomar mejores decisiones.

La funcionalidad principal de las métricas, es medir aspectos o particularidades específicas de los productos obtenidos. Las métricas del software responden a dos objetivos fundamentales, la valoración y la estimación. Las principales magnitudes dentro de la valoración son la calidad, fiabilidad y la productividad; mientras que las magnitudes de la estimación corresponden al esfuerzo y al tiempo.

Las métricas de software deben medir el proceso, el proyecto, el producto y los recursos (HUMPHREY 1995) (RUP 2003), partiendo del hecho de que:

- proceso: es la secuencia o las actividades invocadas para producir el producto de software (y otros artefactos).
- producto: son los artefactos del proceso, incluyendo el software, los documentos y modelos.

- proyecto: son todos los recursos del proyecto, actividades y artefactos.

- recursos: son las personas, los métodos y herramientas, el tiempo, esfuerzo y presupuesto, disponibles para el proyecto.

1.7.1. Métricas del producto

Los productos están compuestos por artefactos, los cuales pueden ser documentos, modelos, módulos, o componentes, por tanto, las métricas del producto deben hacerse sobre la base de medir cada uno de los artefactos. Las métricas del producto describen características como el tamaño, complejidad, rasgos del diseño, rendimiento y nivel de calidad. (KAN 2000)

En general las características a medir de los artefactos del producto son (RUP 2003):

- Tamaño: Las métricas del tamaño del producto se refieren generalmente al volumen del producto desarrollado. Incluyen líneas de código (LOC), número de ficheros, páginas de la documentación, etc.

- Calidad:

    - Defectos: indicadores de que un artefacto no funciona como ha sido especificado, o cualquier otra característica indeseable.

    - Complejidad: medición de la complejidad de una estructura o un algoritmo: mientras mayor sea la complejidad y más difícil sea de comprender y modificar la estructura del sistema, mayor probabilidad habrá de que falle.

    - Acoplamiento: mediciones de cuantos elementos del sistema están interconectados y cuan extensivamente.

    - Cohesión: mediciones de cuán bien un elemento o un componente cumple con los requerimientos de tener un sólo y bien definido propósito.

    - Primitividad: el grado en el cual las operaciones o métodos de una clase pueden estar compuestos por otros de la misma clase.

- Totalidad: medición de la magnitud en la cual un artefacto cumple con todos los requerimientos (plan / real)

- Rastreabilidad: Indicadores de que los requerimientos de determinado nivel se están satisfaciendo por determinados artefactos, o que todos los artefactos tengan razón de existir.

- Volatilidad: el grado de cambio de un artefacto debido a defectos o a cambios en los requerimientos.

- Esfuerzo: medición del trabajo (Unidad de tiempo del personal) que se necesita para producir un artefacto.

1.7.2. Métricas del proceso

Los procesos de software se pueden definir como una secuencia de pasos requeridos para desarrollar o dar mantenimiento a los softwares. La definición de un proceso de software es la descripción del proceso como tal. Un equipo de trabajo que no usa ningún proceso para organizar su forma de trabajo es como un equipo de béisbol, en el que algunos miembros del equipo juegan balompié, algunos béisbol y otros fútbol. Bajo estas condiciones incluso los mejores jugadores individuales forman un equipo pobre. Sin embargo, un equipo que sigue una definición de procesos consistentes puede coordinar mejor el trabajo individual de sus miembros y dar continuidad a su proceso.

Un proceso definido identifica y simplifica tareas de rutina y ayuda a pensar con mayor precisión en el trabajo a realizar. Una vez que los procesos están definidos y medidos se pueden cambiar y mejorar.

Las métricas del proceso son entonces las que cuantifican el comportamiento de los procesos, los cuales son generalmente objetivos, absolutos, explícitos y dinámicos.

Para caracterizar completamente un proceso, las métricas deben ser definidas con el nivel de formalidad más bajo en las actividades planificadas. Las actividades deben ser planificadas por el administrador del proyecto, usando un conjunto inicial de estimados. Por lo tanto se debe mantener un registro de los valores reales durante el transcurso del tiempo y cualquier modificación de lo estimado que se haga.

1.7.3. Métricas de recursos

Los elementos a ser medidos en las métricas de los recursos son las personas (su experiencia, habilidades, costos y desempeño), los métodos y las herramientas (en términos de efectos en la productividad y la calidad), el costo, tiempo, esfuerzo, presupuesto (recursos consumibles y recursos remanentes). (RUP 2003)

Las métricas de recursos se aplican, fundamentalmente, a las horas de labor, el principal recurso del desarrollo de software. Aquí lo que concierne son las horas trabajadas, categorías de trabajo y realización de tareas.

Obteniendo datos detallados sobre el tiempo, se pueden identificar muchas actividades ajenas, reuniones administrativas, tareas burocráticas, uso inadecuado de las facilidades de la computación todas las cuales gastan tiempo y reducen la productividad. (HUMPHREY 1995) Cualquier interrupción tiene 3 costos potenciales: la pérdida de tiempo, el tiempo adicional que toma reconstruir el momento en que fue interrumpido y la probabilidad incremental de cometer errores. Un foco principal de cualquier esfuerzo para mejorar la productividad o el ciclo de tiempo debe identificar y reducir estas distracciones.

1.7.4. Métricas del proyecto

Las métricas del proyecto son aquellas que describen las características del proyecto y la ejecución de este. Algunos ejemplos pudieran ser: el número de programadores de un software, el costo, planificación y productividad del equipo. (HUMPHREY 1995) El proyecto necesita ser caracterizado en términos de tipo, tamaño, complejidad y formalidad, debido a que estos aspectos condicionan expectativas sobre las distintas tendencias a seguir.

Las métricas generales a almacenar sobre un producto son (RUP 2003):

- Modularidad: Promedio de daños debido a cambios perfectivos o correctivos en la implementación.
- Adaptabilidad: promedio de esfuerzo debido a cambios perfectivos o correctivos en la implementación.
- Madurez: tiempo de prueba activo / número de cambios correctivos.
- Mantenimiento: mantenimiento productivo / desarrollo productivo
- Progreso del proyecto: debe reportarse basándose en el plan del proyecto desde la perspectiva del valor devengado.

1.7.5. Otras métricas

Además, existen otro gran número de métricas entre las más importantes están la métrica de los problemas de los clientes, la métrica de la satisfacción del cliente y las métricas de programación y esfuerzo.

Una métrica que es usada ampliamente por los desarrolladores en la industria de software, mide los problemas que los clientes encuentran cuando usan un producto. Desde el punto de vista del cliente, todos los problemas que ellos encuentran cuando usan un producto de software, no sólo los defectos válidos, son problemas del software. Por ejemplo, problemas de usabilidad, documentación e información no claras, o incluso errores de los usuarios. A estos se les llaman “problemas no orientados a los defectos”, los cuales junto a los defectos constituyen el espacio total de los problemas de un software desde el punto de vista de los clientes. (KAN 2000)

Esta métrica es una medición intermedia entre la medición de los defectos y la satisfacción del cliente. Para reducir los problemas de los clientes, hay que reducir los problemas funcionales del producto y además hay que mejorar otros factores como la usabilidad, documentación, problemas redescubiertos, etc.

1.8. Características de una buena métrica

Los principios fundamentales que deben seguir las métricas son (RUP 2003):

- las métricas deben ser simples, objetivas, fáciles de coleccionar, fáciles de interpretar y difíciles de malinterpretar.

- la colección de las métricas debe ser automática y no intrusiva, o sea, no interferir en las actividades de los desarrolladores.

- las métricas deben contribuir a la evaluación de la calidad temprana en el ciclo de vida, cuando los esfuerzos por mejorar la calidad del software son efectivos.

- los valores absolutos y las tendencias de las métricas, deben ser usados activamente por el personal administrativo y el personal ingenieril, para comunicar progreso y calidad en un formato coherente.

- la selección de un mínimo o más extensivo conjunto de métricas, dependerá de las características y contexto del proyecto: si es muy grande o si tiene restricciones de seguridad o de confiabilidad de los requerimientos; y si el equipo de desarrollo y de valoración (evaluación) es conocedor de las métricas, lo cual hará muy útil coleccionar y analizar las métricas técnicas.

Una taxonomía (clasificación) de métricas debe incluir (RUP 2003):

- progreso en términos de tamaño, tiempo y complejidad.
- estabilidad en términos de índice de cambios en requerimientos, implementación, tamaño, tiempo, o complejidad.
- modularidad en términos del ámbito de cambios.
- calidad en términos del número y tipo de errores.
- madurez en términos de la frecuencia de errores.
- recursos en términos del costo real del proyecto versus el costo planificado.

Las tendencias son importantes, e incluso más importantes de monitorear que cualquier valor absoluto en el tiempo.

1.9. Conclusiones

Hoy en día, la computación juega un rol primario en casi todas las áreas de nuestra vida. La creciente importancia del software pone más requerimientos en él. Por lo tanto es necesario tener un control preciso y repetido sobre los procesos de producción y los productos de software.

Las métricas del software son la base de la realización de las estimaciones y una de las vías más utilizadas para la toma de decisiones. Las métricas deben ser simples, objetivas, fáciles de coleccionar, fáciles de interpretar y difíciles de malinterpretar. Los valores absolutos y las tendencias de las métricas, deben ser usados activamente por el personal administrativo y el personal ingenieril, para comunicar progreso.

Las mediciones de software son usadas para medir atributos específicos de un producto de software o del proceso de desarrollo de software. Se usan fundamentalmente para: obtener las bases para la estimación, seguir el progreso de los proyectos, determinar la complejidad, ayudar a comprender

cuando se ha alcanzado un estado deseado de calidad, analizar los defectos y validar experimentalmente las mejores prácticas. Ayudando a tomar mejores decisiones.

El apoyo de los administradores es esencial para lograr una implementación exitosa de las métricas. Los administradores deben tomar un interés activo y visible en el proceso de medición.

Existen softwares para el control del progreso y otros temas de interés de los proyectos, pero la mayoría son de propósito general, muy útiles para proyectos grandes, son herramientas muy eficientes pero no se ajustan a los problemas reales que existen en la industria de software naciente en la UCI. Una base de estimación tradicional no debe ser aplicada estrictamente a un método productivo diferente como el desarrollado en la UCI, en el cual llevan a cabo la producción profesores y estudiantes como parte del modelo Universidad-Empresa.

En la métrica que se propone se pretende medir el estado actual de desarrollo del software, basado en el cumplimiento de tareas (actividades que definen la implementación) y dividir la medición por casos de uso teniendo en cuenta los factores de complejidad técnica (alta, media, baja). Se mide el proceso y el producto; el progreso se expresa en términos de tamaño, tiempo y complejidad. Crea las bases para desarrollar en un futuro un modelo empírico de medición de software y contribuye a la formación de datos históricos con los que hoy no cuenta la institución.

CAPÍTULO 2: DEFINICIÓN DE LA MÉTRICA

2.1. Introducción

Las mayores desviaciones que se producen en un proyecto son debidas a deficiencias en el control del mismo. Para asegurar la eficacia de la ejecución es necesario que el líder de proyecto realice un buen seguimiento. Si se miden los procesos y los productos con métricas rigurosas, se puede mejorar y controlar el proceso a partir de los resultados en su ejecución. (Febles 2003)

Es tarea del líder de proyecto elaborar los informes de seguimiento de forma periódica, puntual, concisa e impersonal y presentar propuestas de soluciones en el caso de producirse desviaciones. Muchas veces estos informes se ven afectados por la falta de objetividad, debido a que los líderes no poseen mecanismos para apreciar la evolución del trabajo desarrollado por los miembros de su equipo.

Las métricas del software son analizadas y evaluadas por el líder de proyecto. Mediante la medición, el líder puede señalar las tendencias (buenas o malas), realizar mejores estimaciones y llevar a cabo una verdadera mejora sobre el tiempo.

La medición proporciona al líder de proyecto la evidencia cuantificable que este necesita como apoyo a la toma de decisiones, basándose en anotaciones realistas y no en la subjetividad, hace más visible el desarrollo y le permite anticiparse a los problemas, ayudando así, a que este pueda mantener el control. Además, estas observaciones le permitirán al líder en caso de atrasos durante el proceso de desarrollo, trazar las estrategias necesarias para cumplir con la planificación.

Las métricas constituyen una base objetiva para la gestión de proyectos y son fundamentales para poder planificar y controlar de forma realista, razón por la cual los líderes de proyectos de software necesitan utilizar métricas que le permitan cuantificar el avance de sus proyectos y mantener informados a clientes y directivos del estado en que se encuentra el software durante los diferentes flujos de desarrollo.

Por consiguiente, en el proyecto SIGEP, se crea una métrica que permite al líder medir el avance del software durante la etapa de implementación. En el presente capítulo se presenta la propuesta de la métrica.

2.2. ¿En qué consiste la implementación?

Uno de los flujos más controversiales en el proceso de desarrollo de un software debido a su importancia y magnitud dentro del ciclo de desarrollo es precisamente el flujo de implementación. En este flujo son implementadas todas las funcionalidades que debe tener el software, por tanto, es en el cual se construye prácticamente el software en su totalidad.

Para realizar con éxitos la implementación del software es necesario que los implementadores tengan un conocimiento previo de la plataforma, que esté definida la arquitectura del software y que se haya realizado anteriormente una buena captura de requisitos.

Como se muestra en la Figura 2 en el Proceso Unificado la implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos como los encontrados con las distribuciones beta del sistema. (RUP 2003)

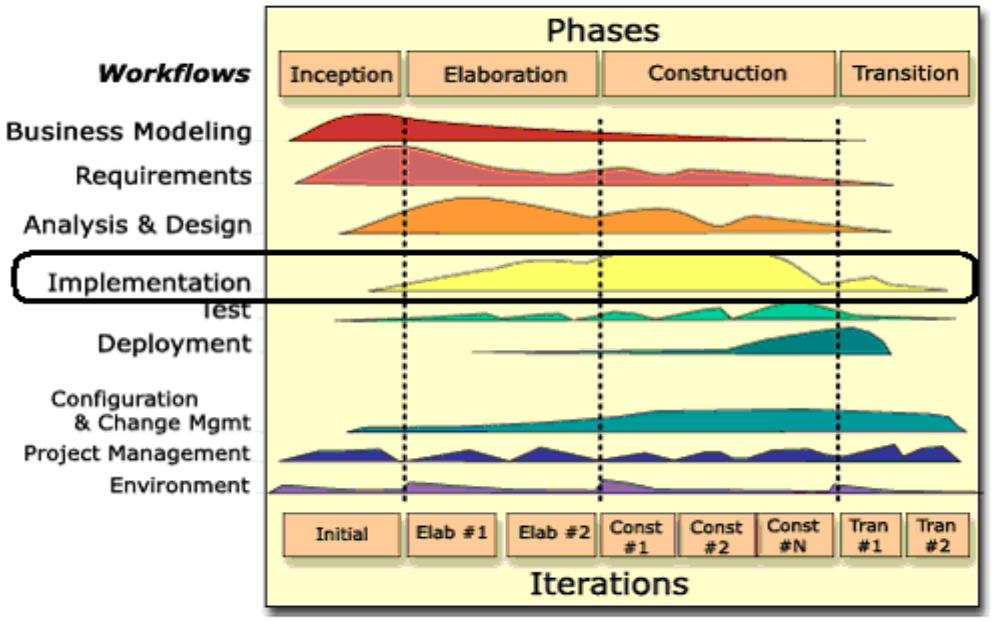

Figura 2 RUP en dos dimensiones

Se definen como propósitos de la implementación:

- Definir la organización del código en capas, en términos de implementación de subsistemas.

- Implementar elementos de diseño en términos de implementación de elementos (códigos fuentes, binarios, ejecutables, y otros).
- Probar los componentes desarrollados como unidades.
- Integrar los resultados producidos por los implementadores individuales (o equipos), en sistemas ejecutables.

2.3. ¿Cómo medir el avance en la etapa de implementación?

Para conocer el estado del software durante el flujo de implementación es necesario medir el avance a lo largo de este flujo. Donde avance se expresa como el por ciento de cumplimiento de las tareas de la implementación. Razón por la cual se divide el avance en dos dimensiones que juntas sean capaces de medir el avance del software.

Dichas dimensiones serían:

- Definición de las tareas del flujo de implementación del SIGEP.
- Determinación de la complejidad de las tareas.

La realización de estas formará parte del diseño de la métrica propuesta.

2.3.1. Definición de las tareas de la implementación

Para lograr un buen monitoreo y control durante el flujo de implementación se hace necesario definir todas las tareas involucradas en este flujo de trabajo, de forma que esto permita saber qué tareas han de ser realizadas y quién es el responsable de ejecutarlas.

Antes de pasar a definir las tareas del flujo de implementación se precisa conocer la estructura del equipo de desarrollo, la arquitectura que rige este flujo, así como el flujo de trabajo de los roles involucrados en la implementación del SIGEP.

2.3.1.1. Estructura del equipo de desarrollo

El equipo de desarrollo de software del proyecto está conformado por cinco áreas fundamentales:

- **Gerencia y Gestión**: Responsable de la dirección del proyecto, la planificación y el control de la ejecución. Se apoya en asesores permanentes legales, de procesos y otros.

- **Base de datos**: Responsable del diseño, configuración, programación y mantenimiento de los modelos de datos y de la base de datos en sí.

- **Arquitectura**: Responsable de definir la línea base de la arquitectura, define las pautas para el diseño y la codificación. Establece el esqueleto sobre el cual se implementarán los casos de uso en los equipos de desarrollo.

- **Equipos de desarrollo**: Para proyectos de desarrollo muy grandes conviene dividir el trabajo en equipos más pequeños que asuman la construcción de sistemas, subsistemas o módulos en dependencia de la complejidad de estos. La cantidad de estos equipos varía según la disponibilidad del personal y las dimensiones del proyecto. Intervienen especialistas funcionales de las áreas particulares para las que se desarrolla el sistema.

- **Aseguramiento de la Calidad**: Asegura la calidad a lo interno del proyecto desde el inicio. Sirve de contrapartida al trabajo de los equipos de desarrollo. Evalúa que se respeten las normas de calidad establecidas por la organización.

- **Análisis**: Responsable de definir qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen. Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.

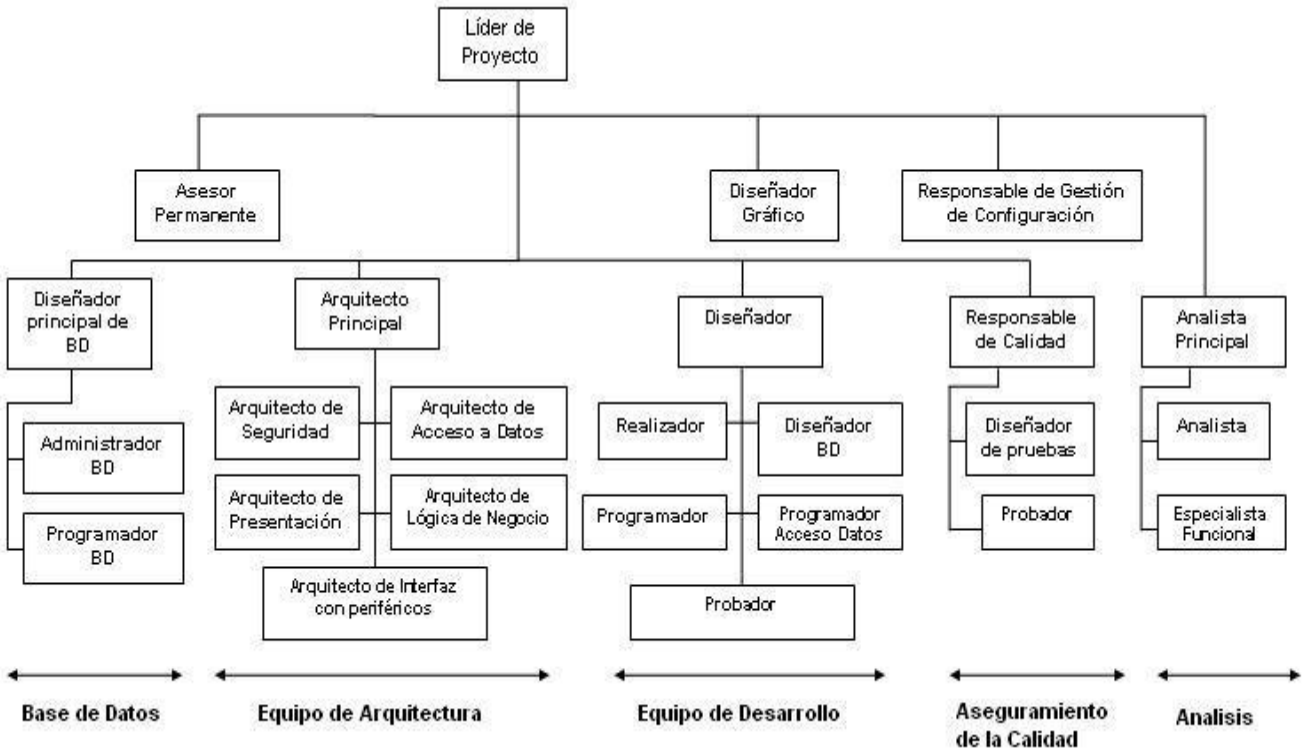

Figura 3 Estructura del Equipo de desarrollo

Como podemos observar el equipo de desarrollo está estructurado por los siguientes roles:

- Diseñador
- Realizador
- Programador de Interfaz de Usuario
- Diseñador de Base de Datos
- Programador de Acceso a Datos
- Probador

Descripción del rol Diseñador:

Interpreta la información resultado del análisis y traduce al lenguaje de los programadores (interfaz, negocio y acceso a datos). Define los elementos de diseño a tener en cuenta para la implementación

de los casos de uso. Diseña la implementación sobre la arquitectura definida, integra los componentes de la solución y define las interfaces, es el responsable de la codificación de los objetos de negocio de la aplicación y dirige el trabajo de los programadores.

Descripción del rol Diseñador de Base de Datos:

Utiliza la información del análisis del negocio para identificar, definir y catalogar todos los datos que la aplicación almacenará en la base de datos. Documenta los datos mediante un diagrama entidad-relación.

Descripción del rol Programador de interfaz de usuario:

Crea el prototipo de interfaz de usuario. Es responsable de la codificación de todos los HTML, Javascript, applet/Swing, JSPs y/o Servlets de la aplicación. En general ejecuta cualquier tarea directamente involucrada con la producción de la interfaz de usuario y colabora con el diseñador gráfico para desarrollar un prototipo funcional.

Descripción del rol Programador de acceso a datos:

Responsable de la codificación de los objetos de acceso a datos y de la programación de la comunicación con sistemas externos.

Descripción del rol Realizador:

Construye estáticamente el prototipo de interfaz de usuario a partir de las pautas definidas para el diseño de la interfaz. Las vistas estáticas que construye, constituyen el punto de partida de los programadores de Interfaz de Usuario.

Descripción del rol Probador:

Responsable de diseñar los casos de prueba[2] y los juegos de datos. Ejecuta las pruebas diseñadas y anota los resultados obtenidos.

[2] Casos de prueba: Un caso de prueba será un conjunto de entradas con datos de prueba, unas condiciones de ejecución, y unos resultados esperados

2.3.1.2. La arquitectura del SIGEP

Es substancial para el flujo de implementación del software, definir la arquitectura en la fase de elaboración, de modo que estructure y soporte la implementación de todo el sistema y solo sean necesarios pequeños ajustes en las restantes etapas. Dada esta situación, surge ArBaWeb (arquitectura base sobre la web), SIGEP es el proyecto en la UCI que más ha usado el framework ArBaWeb debido a que la idea de su creación surgió como necesidad del propio entorno y, por tanto, se aplica desde los inicios gestionando toda la arquitectura.

¿Por qué es importante una arquitectura de software? Independientemente a la variedad de definiciones literarias se converge en la gran importancia que representa la arquitectura en el desarrollo de cualquier tipo de software; pues representa las bases sobre las que se desarrollará el sistema. Brinda una estructura que soporta las soluciones a cada tipo de problema que pueda aparecer en el desarrollo. Define la organización e interacción entre los distintos componentes del mismo. Asegura que los requerimientos más importantes puedan ser evaluados e implementados. Una arquitectura de software también permite flexibilidad en el sistema pues facilita la ejecución de futuros cambios. Promueve la reutilización de componentes existentes como librerías de clases, sistemas legados y de aplicaciones de terceros.

La arquitectura de software provee una descripción de alto nivel de los subsistemas que comprenden la arquitectura del sistema. Está atada a comprender cómo las mayores operaciones del negocio son soportadas. Ayuda a la planificación de recursos y la asignación de tareas, debido a que el trabajo de desarrollo puede ser particionado a través de los subsistemas y los esfuerzos de desarrollo individual pueden proceder en paralelo. Cuando los equipos de desarrollo están geográficamente dispersados puede minimizar el número de interacciones requeridas.

La arquitectura base propuesta por el SIGEP se descompone en cuatro aspectos: el diseño de las capas lógicas, convenciones o estándares de nombres, estructuras de código y mecanismo de colaboración entre los subsistemas y módulos específicos de la aplicación.

Dicha arquitectura plantea: orientar el diseño de las capas lógicas a través de una propuesta de diseño base; organizar la forma de codificar según las propuestas de convenciones o estándares de códigos y recursos; brindar una estructura física para soportar el código, creando así un esqueleto base; y proponer mecanismos de colaboración entre los componentes integrados en ella. Se basa fundamentalmente en los estilos arquitectónicos Cliente-Servidor y Arquitectura en capas.

Capítulo 2

ArbaWeb define un sistema con arquitectura de tres capas, distribuidas cada una según las siguientes especificaciones:

- **Cliente**: Computador de tecnología Intel. La aplicación se ejecuta a través de un navegador de Internet instalado sobre cualquier sistema operativo (se sugiere alguna distribución Linux y Windows XP solo para las estaciones de trabajo a las que se conectarán los dispositivos externos: escáner de huellas y de documentos, cámara digital). Máquina Virtual de Java, versión 1.5 para aplicaciones clientes que la necesiten.

- **Servidor Web**: En este servidor radica la lógica de negocio de la aplicación. Computador de tecnología Intel. Sistema operativo Linux RedHat 4.0, Advanced Server, para los servidores del centro de datos y alguna distribución de Linux para los servidores locales de los establecimientos penitenciarios. Servidor Web Apache Tomcat versión 5.x.x. Java Runtime Environment (JRE), versión 1.5.

- **Servidor de Base de datos**: Computador de tecnología Intel. Sistema operativo Linux RedHat 4.0, Advanced Server, para los servidores del centro de datos y alguna distribución de Linux para los servidores locales de los establecimientos penitenciarios. Servidor de base de datos Oracle 10g Standard Edition. Espejado de disco.

La aplicación está dividida en tres capas lógicas fundamentales:

Capa de Presentación: En esta capa se encuentran las Vistas y la Lógica de Presentación. En la Lógica de Presentación se maneja todo el flujo web utilizando la implementación del patrón Modelo Vista Controlador que nos brinda Spring MVC. Las Vistas son los recursos que junto al modelo generado por los controladores le permiten al cliente visualizar la información, estos pueden ser páginas HTML, documentos en formato PDF, hojas de cálculo, entre otras.

Capa de Servicios de Negocio: Encapsula toda la lógica de la aplicación en fachadas de negocio que son utilizadas por los controladores en la capa de presentación y se exponen algunos procesos de negocio a través de interfaces de servicios. A estas fachadas de negocio se le aplican la seguridad a nivel de métodos y de objetos de negocio, auditorías, caché, política de transacciones, entre otros.

Capa de Acceso a Datos: Maneja los objetos de acceso a datos abstrayéndolos del mecanismo de persistencia usado; a través de interfaces que exponen las operaciones de persistencia definidas para

cada uno de los DAOs e implementadas utilizando Hibernate, que es un framework de mapeo a objeto-relacional (ORM).

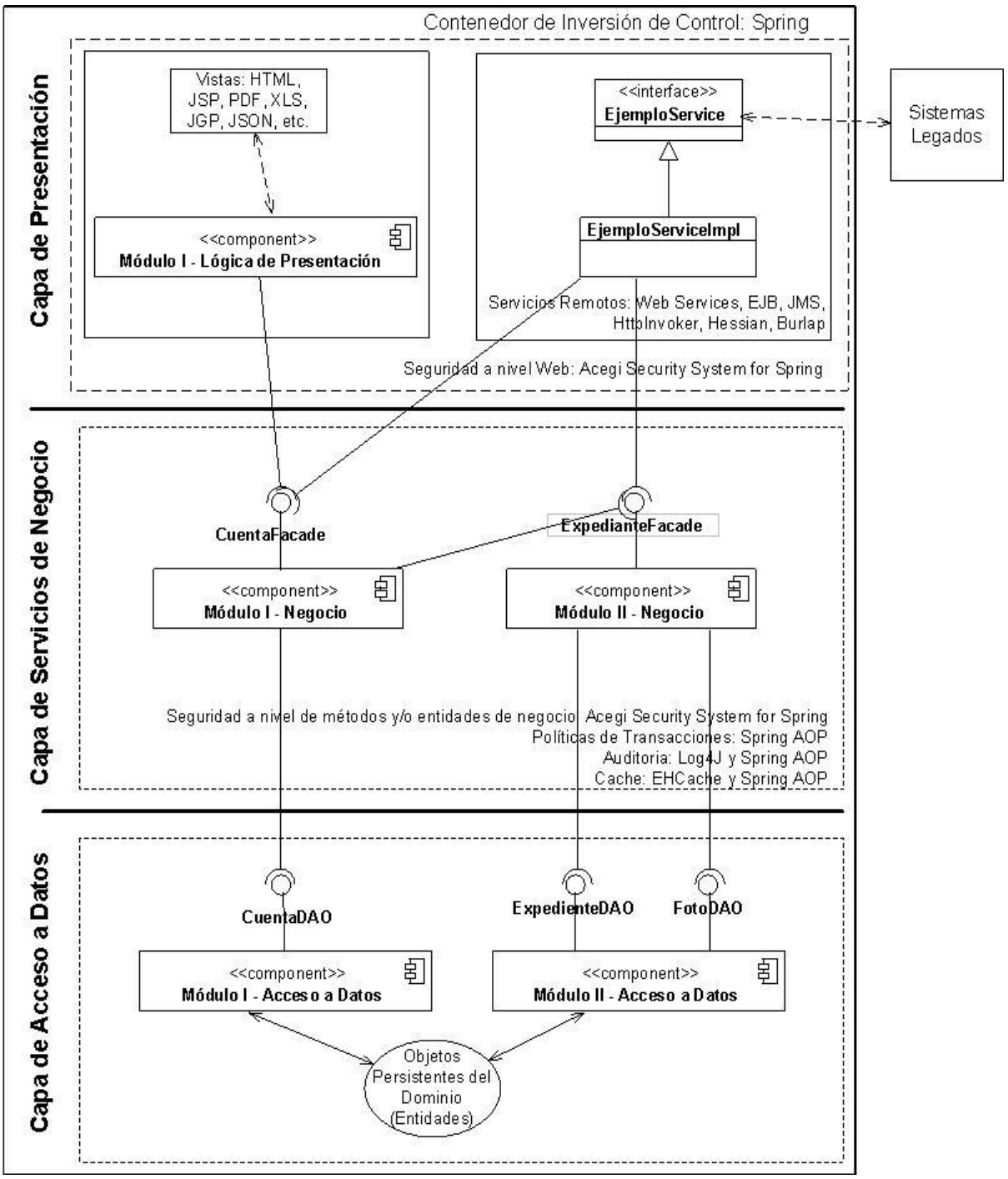

Figura 4 Estructura por capas de la aplicación

2.3.1.3. Flujos de Trabajo Proyecto SIGEP

Para el desarrollo de una aplicación se hace necesario seguir un conjunto de pasos que guíe a los desarrolladores en la elaboración de cada componente. Este flujo de trabajo se enmarca en las funcionalidades mínimas necesarias para que se puedan desarrollar en paralelo las capas y que exista una interacción óptima entre los desarrolladores. ArBaWeb define un rol correspondiente a cada capa lógica de la aplicación: programador de acceso a datos, programador de lógica de negocio y programador de interfaz de usuario. Las actividades que ellos realizan como parte del flujo de trabajo definido por ArBaWeb se exponen a continuación.

Acceso a Datos

- **Crear ficheros de mapeo**: Cuando se comienza a desarrollar un módulo lo primero que debe salir son las entidades de dominio por parte de programador de lógica de negocio, el programador de acceso a datos utiliza estas entidades para comenzar a realizar los ficheros de mapeo con el diagrama de Entidad Relación de la base de datos. Esta actividad se debe realizar antes de realizar las pruebas a las implementaciones de los DAO.

- **Crear implementación falsa de los DAO**: Una vez que el programador de lógica de negocio termine de realizar las interfaces de los DAO el programador de acceso a datos debe comenzar inmediatamente la implementación de los DAO falsos para que las demás capas puedan continuar su trabajo y no depender de que el de acceso a datos termine todo su trabajo.

- **Declarar DAOMock en el dataaccess-context y en el dataaccess-context-moc**k: Una vez implementados los DAO falsos se declara en el dataaccess-context para que el programador de lógica de negocio los inyecte a su fachada sin conocer que son las implementaciones falsas. Se deben declarar también en el dataaccess-mock para que puedan ser utilizados en cualquier momento garantizando siempre el funcionamiento de cada capa de la aplicación ante cualquier problema que esté presente de funcionamiento.

- **Implementar DAO**: Una vez terminados los ficheros de mapeo se debe proceder a realizar la implementación real de los DAO.

- **Realizar pruebas a los mismos**: Terminada la implementación se procede a realizar las pruebas de unidad a los mismos.

- **Poner en el dataaccess-context.xml**: Cuando una prueba de unidad se ha completado completamente a un DAO este ya se encuentra listo para declararse en el dataaccess-context.xml en lugar del DAOMock que existía anteriormente.

Lógica de Negocio

- **Crear Entidades de Dominio**: El programador de lógica de negocio realiza las implementaciones de las entidades de dominio que necesita el módulo.

- **Crear Interfaces de las Fachadas**: Crea las interfaces de las Fachadas donde representa las funcionalidades que se deben realizar en el negocio, conocidas hasta el momento.

- **Crear Interfaces de los DAO**: Crea las interfaces de los DAO donde define las operaciones que se necesitan que se implementen en los DAO.

- **Crear Implementaciones falsas de las fachadas**: El programador de interfaz de usuario necesitará que los métodos definidos en la, o las fachadas, se encuentren funcionales para poder implementar los controladores.

- **Declarar las fachadas falsas en el AppContext de negocio y en el context-mock**: Una vez que se han elaborado las implementaciones falsas de la, o las fachadas, se procede a declararlas en los ficheros de configuración mencionados. En el AppContext de negocio para que el programador de interfaz de usuario pueda continuar su trabajo en los controladores y en el context-mock con el objetivo de tener la capa funcional en caso de ocurrir algún problema de implementación en esta capa.

- **Implementar fachadas**: Se implementa la lógica que corresponde a cada método dentro de la fachada.

- **Realizar pruebas a las fachadas**: Se realizan las pruebas de unidad a las fachadas para probar que todas las funcionalidades realicen lo deseado.

- **Declarar las fachadas reales en la aplicación**: Una vez realizadas las pruebas de unidad a las fachadas y comprobado que todo esté correcto se procede a declarar la implementación real en el AppContext de negocio.

Presentación

- **Implementar controladores sin lógica**: Se implementan los controladores sin funcionalidad alguna.

- **Declarar controladores en los AppContext**: Se declaran en el AppContext de la capa de presentación context-servlet pues hasta el momento el programador de lógica de negocio no tiene listas las funcionalidades del negocio listas y se mapean en el HandlerMapping de Spring MVC.

- **Crear Commands, Validators y PropetyEditors**: Se crean los Command para obtener los datos que vienen de la web, en algunos casos se pueden utilizar las mismas entidades del dominio como Command. Se crean también los validadores tanto para los Command como para las entidades de dominio en caso de usarse.

- **Implementar controladores**: Se implementan los controladores utilizando las interfaces de servicios definidas por el programador de lógica de negocio.

- **Probar los controladores**: Se realizan las pruebas a los controladores para verificar que todas las funcionalidades son cubiertas correctamente.

- **Crear y declarar las vistas**: Se crean las vistas que interactúan con el usuario y se declaran en los ViewResolvers de Spring MVC en caso de ser necesario.

- **Programar lógica en el cliente**: Se programa toda la lógica necesaria en el cliente.

- **Realizar prueba a interfaz**: Se realizan pruebas para verificar tanto que en el flujo como en la página se realizan las operaciones que deben realizarse.

2.3.1.4. Definición de las tareas

Luego de conocer la estructura del equipo de desarrollo y la arquitectura sobre la cual se desarrolla el sistema se pasa a definir las tareas involucradas en el diseño de la métrica, las cuales se adaptan a las características anteriormente expuestas.

Ellas son:

- **Análisis de la solución**: Esta actividad es desarrollada por el Diseñador del equipo, consiste en el análisis de la documentación resultante de la captura de requisitos realizada por el

Analista para un conocimiento profundo del sistema, concluye una vez que el diseñador domine el negocio y se sienta capaz de diseñarlo. Es la primera actividad que se realiza antes de comenzar la implementación.

- **Diseño del modelo de datos**: Es desarrollado por el Diseñador de Base de Datos, se realiza un estudio previo de la documentación con el cual se obtiene un modelo parcial que más tarde será validado junto al Diseñador y el Analista del equipo para lograr de conjunto la versión terminada del modelo.

- **Creación del modelo de datos**: Debe estar realizado el modelo de datos para que el Diseñador de Base de Datos pueda pasar a la creación del modelo de datos, actividad que consiste en generar la base de datos de los módulos a desarrollar.

- **Desarrollo de interfaz**: Construcción del prototipo de interfaz de usuario a partir del prototipo resultante de la captura de requisitos realizada por el Analista y basado en las pautas de diseño establecidas. Es responsabilidad del Realizador y constituye el punto de partida para la programación de interfaz.

- **Diseño de la solución**: Responsabilidad del Diseñador. Es considerada una macro actividad que agrupa actividades relacionadas con varios roles y áreas del desarrollo, donde se realizan un grupo de tareas que conjuntamente van a permitir el cumplimiento de ella. Estas actividades son:

    - **Diseño de interfaces de negocio**: Consiste en crear las interfaces de los managers, las cuales encapsulan la lógica de negocio sobre una o varias entidades.

    - **Diseño de entidades de dominio**: Definir las entidades de dominio involucradas en la solución del problema. No concluye hasta tanto no se haya realizado la validación del modelo de datos junto al Diseñador de Base de Datos y el Analista.

    - **Diseño de interfaces de acceso a datos**: Consiste en crear las interfaces de los DAOs, así como los métodos necesarios para responder a las funcionalidades.

    - **Diseño de la capa de presentación**: En esta actividad se definen las vistas y el flujo de navegación, y a partir de esto, las posibles peticiones del usuario y los controladores que las van a atender. El diseño de los controladores generalmente implica diseñar

otros componentes que cumplen funciones en el flujo de Spring-MVC como son validadores (validator), objetos de respaldo (command) y editor de propiedades (property editors). Finalmente se definen componentes del cliente, como son clases JavaScript, documentos HTML, imágenes, etcétera. Depende del diseño de las entidades de dominio.

- **Programación de acceso a datos**: Consiste en implementar la capa de acceso a datos, es responsabilidad del programador de acceso a datos. Tiene que estar creada la base de datos y las entidades de dominio. Se concluye cuando se hayan realizado las siguientes actividades:

    - **Realizar los ficheros de mapeo**: Consiste en crear los ficheros de mapeo del Framework Hibernate. Estos ficheros son XMLs que representan la relación entre las tablas de la base de datos con las entidades persistentes, que en este caso son las entidades de dominio.

    - **Programación de interfaces**: Consiste en crear las clases que implementan las interfaces de los DAOs. Estas clases están relacionadas directamente con el Framework Hibernate.

    - **Realizar pruebas unitarias**: Consiste en crear clases donde se realizan pruebas unitarias a las implementaciones de los DAOs.

- **Programación de lógica de negocio**: Depende del diseño de la solución y es realizada por el Diseñador. Se implementan los métodos definidos para cada una de las interfaces de los managers, ajustándose a las funcionalidades previstas. Cada método implementado tiene que verificar la integridad de los datos e informar a la capa de interfaz cualquier eventualidad a través de las excepciones definidas. Si fuese necesario se implementa la publicación y manipulación de eventos. Los managers implementados se configuran en el fichero de configuración del contexto de Spring correspondiente a la capa de negocio del módulo. Por cada manager se debe programar una prueba de caja blanca que verifique cada método implementado. Para la implementación de la interfaz de la fachada solo tiene que conocerse en cuales managers se encuentran implementadas las funcionalidades necesarias para comunicarlas a la capa de interfaz; de esta forma, la fachada no posee ninguna lógica de negocio, solo es experta del lugar donde radica la información que necesita y así la utiliza. El

uso de fachadas simplifica el acceso de la capa de presentación a la capa de negocio, reduciendo el número de objetos con los que tiene que interactuar esta.

- **Programación de interfaz:** La programación de interfaz depende de las actividades: Desarrollo de interfaz y Programación de lógica de negocio. Consiste en realizar el diagrama de navegación (por páginas, vistas e interacciones cliente-servidor), definir tipos de controladores que se van a usar utilizando un mapa de controladores predefinidos, elaborar tabla (controlador-petición-vista), declarar e implementar controladores sin lógica, declarar los controladores en los ficheros xml de configuración de Spring, crear Commands, Validators y PropetyEditors (posible interacción con el diseñador, necesitan las entidades para utilizarlas como Command o PropertyEditor) y se relacionan los atributos del jsp con los del command en caso de que sea necesario, implementar lógica de los controladores, relacionar vista con modelo, programar lógica en el cliente, realizar pruebas a la interfaz para verificar el correcto funcionamiento e interacción de esta con el supuesto usuario final y elaborar tabla (controlador-petición-vista). Realizar esta actividad es responsabilidad del Programador de Interfaz de Usuario y constituye el punto de inicio para la actividad Ejecución de las Pruebas.

- **Pruebas internas de calidad**: Responsabilidad del Probador del equipo y considerada una macro actividad debido al grupo de actividades que deben desarrollarse para darle cumplimiento. Estas actividades son:

    - **Estudio de la documentación**: Consiste en el estudio de la documentación generada por el analista durante la captura de requisitos, hasta lograr ser un especialista del negocio.

    - **Diseño de Casos de Prueba**: Construcción de todos los posibles caminos de ejecución, o escenarios, de cada caso de uso. Se obtiene como resultado un listado final con los casos de prueba identificados a partir de los posibles escenarios, los resultados esperados para cada caso y las condiciones o valores requeridos para la ejecución de los distintos escenarios.

    - **Diseño de Juego de Datos**: Para cada escenario de caso de prueba se identifican sus valores de prueba, es decir, se precisan los datos de prueba.

- **Ejecución de las Pruebas**: Esta actividad depende de la programación de interfaz. Garantiza la calidad del sistema desarrollado, con el conjunto de escenarios de casos de uso y valores de prueba obtenidos se podrán realizar fácilmente las pruebas del sistema que garanticen que toda la funcionalidad recogida en este caso de uso ha sido correctamente implementada, por tanto, esta actividad necesita que se realicen las actividades anteriores para obtener mejores resultados.

La realización de estas tareas hace posible que se realicen los propósitos de la implementación, desde el punto de vista de los casos de uso, ninguno habrá concluido hasta tanto no haya sido realizada cada una de las actividades anteriores para su caso en particular.

Por este motivo a cada una de las tareas definidas se le otorga, dependiendo del caso de uso al que se refiere, un **Estado de completamiento** entre 0 y 1, 0 si aún no se ha realizado y 1 si ya la tarea se encuentra terminada, un valor decimal que se encuentre entre estos dos números representa el progreso de la tarea dado por el responsable de ejecutarla y basado en las actividades definidas dentro de su flujo de trabajo que aún le restan por terminar.

Cada una de las tareas anteriores tiene asignado un peso basado en el tiempo que se tarda la ejecución de dicha tarea durante la implementación. Para establecer dicho peso del modo más real posible, en el proyecto se reunieron un grupo de implementadores y diseñadores, los cuales junto al líder de proyecto, apoyándose en la experiencia de las primeras iteraciones definieron los pesos correspondientes a cada una de ellas. El valor asignado a cada tarea es llamado **Peso de la tarea**.

En la Tabla 1 se muestran los pesos correspondientes a cada una de las tareas.

Tareas	Días que demoró la realización de la tarea	Peso Asignado
Análisis de la solución	**2**	**4%**
Diseño del modelo de datos	**3**	**7%**
Creación del modelo de datos	**1**	**2%**
Desarrollo de interfaz	**5**	**11%**
Diseño de la solución	**4.5**	**10%**

Programación de acceso a datos	**4**	**9%**
Programación de lógica de negocio	**4**	**9%**
Programación de interfaz	**15**	**33%**
Pruebas internas de calidad	**6.5**	**14%**

Tabla 1 Relación de pesos asignados a las tareas

2.3.2. Determinación de la complejidad de las tareas.

La complejidad de las tareas está dada en función de la complejidad del caso de uso. La razón será expuesta a continuación. Empecemos definiendo qué es un caso de uso.

Según el Proceso Unificado un caso de uso se define como: fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores.

Para lograr con éxito un proceso de desarrollo de software es bueno definir casos de uso lo más pequeños posible, por la facilidad que esto permite a la hora de la implementación.

Un caso de uso según su complejidad se clasifica en: alto, medio o bajo. Esta clasificación es concedida por el Diseñador en función de los siguientes aspectos:

- Número de elementos de datos.
- Lógica de negocio.
- Interfaces externas, dígase dispositivos u otro sistema.

Las tareas de la implementación son parte del proceso de solución iniciado con la definición de las funcionalidades o casos de uso definidos para el sistema durante la captura de requisitos. Estos dos aspectos están fuertemente vinculados si se desea conocer el estado de avance de la implementación de un módulo en el cual podemos encontrar casos de uso de diferente complejidad. La existencia de casos de uso de diferente complejidad implica que el tiempo de realización de una tarea no es el mismo para todos los casos, ya que una persona no tarda lo mismo en realizar una tarea cuando se está elaborando un caso de uso de complejidad alta que cuando se trata de un caso de uso de complejidad media o baja.

En un módulo solamente es posible que se presente una de las siguientes combinaciones de complejidad de los casos de uso presentes.

- Existen casos de uso con complejidad alta, media y baja
- Todos los casos de uso tienen complejidad alta
- Existen casos de uso con complejidad alta y media
- Existen casos de uso con complejidad alta y baja
- Todos los casos de uso tienen complejidad media
- Existen casos de uso con complejidad media y baja
- Todos los casos de uso tienen complejidad baja

Como un módulo puede presentar solamente una de las combinaciones antes citadas, para diseñar la solución se hizo necesario asignarle a cada clasificación existente dentro de cada una de las combinaciones un peso único, este valor es llamado **Peso del caso de uso** (Pcu). Este peso resultó para cada caso de la siguiente forma:

Combinaciones posibles	Peso CU complejidad alto (Pcu_{alta})	Peso CU complejidad medio (Pcu_{media})	Peso CU complejidad bajo(Pcu_{baja})
Existen casos de uso con complejidad alta, media y baja	0.5	0.3	0.2
Todos los casos de uso tienen complejidad alta	1	-	-
Existen casos de uso con complejidad alta y media	0.6	0.4	-
Existen casos de uso con complejidad alta y baja	0.8	-	0.2
Todos los casos de uso tienen complejidad media	-	1	-
Existen casos de uso con complejidad media y baja	-	0.6	0.4
Todos los casos de uso tienen complejidad baja	-	-	1

Tabla 2 Relación de los pesos según las combinaciones de casos de uso

2.4. Conclusiones

En el presente capítulo quedaron definidas las tareas o actividades del flujo de implementación y la complejidad de ellas. Quedó explicada la estructura del equipo de desarrollo y las características del equipo, las tareas que se realizan dentro del flujo de trabajo establecido y cómo es que una tarea puede tener mayor complejidad en dependencia del caso de uso en desarrollo. De esta forma quedó explicada la estructura de la solución propuesta.

CAPÍTULO 3: SOLUCIÓN Y RESULTADOS DE LA MÉTRICA

3.1. Introducción

En los inicios del proceso de desarrollo no se contaba con un método que permitiera medir el estado de avance del software. Con el transcurso del tiempo y la puesta en práctica de este método se fueron notando cambios satisfactorios en el desarrollo y avance del proyecto.

En este capítulo se explica el procedimiento que se llevó a cabo para el diseño matemático una vez aplicada la métrica de control de avance. Seguidamente se hace un análisis detallado de los resultados obtenidos en cada una las gráficas resultantes.

3.2. Solución de la métrica

Esta métrica debe ser calculada con periodicidad durante todo el flujo de implementación, en el caso del SIGEP se utiliza para emitir los partes semanales a los directivos interesados en el desarrollo del software.

La métrica propuesta tiene entre sus objetivos establecer el por ciento de avance de la implementación de un módulo. Para calcular este por ciento se divide la medición por casos de uso y se basa en los factores de complejidad técnica (alta, media, baja).

Para conocer el por ciento de avance de la implementación de un módulo, antes es necesario determinar el por ciento de avance de la implementación de cada caso de uso que lo compone.

Para explicar con mayor detalle el cálculo del avance por casos de uso es necesario basarse en la Tabla 3, que muestra un ejemplo del mismo extraído de la aplicación de la métrica en el SIGEP.

		4%	7%	2%	11%	10%	9%	9%	33%	14%			
	Complejidad	Análisis de la solución	Diseño del modelo de	Creación de modelo de	Desarrollo de interfaz	Diseño de la solución	Programación de acceso a	Programación de lógica de	Programación de interfaz	Pruebas internas de	% avance	% avance	
Capacidades												**97%**	alta 0
Gestionar locales del área de reclusión.	Media	1	1	1	1	1	1	1	1	0,8	97%		media 0,6
Gestionar cupos.	Media	1	1	1	1	1	1	1	1	0,8	97%		baja 0,4
Gestionar afectaciones de locales y cupos.	baja	1	1	1	1	1	1	1	1	0,8	97%		
Consultar estado de cupos o locales	baja	1	1	1	1	1	1	1	1	0,8	97%		

Tabla 3 Cálculos realizados durante la aplicación de la métrica en el SIGEP, para el caso del módulo Capacidades

Nótese que en la Tabla 3, no se observa el cálculo del avance del caso de uso, solamente su valor final. Sin embargo el método de calcularlo es muy fácil, el avance por cada caso de uso se podrá calcular por:

Avance por **C**aso de **U**so (ACU) = ∑ Estado de completamiento * peso de la tarea

Donde:

Peso de la tarea corresponde al valor asignado a cada una de las tareas en la Tabla 1.

Para el ejemplo dado en la Tabla 3 para el caso de uso Gestionar cupos, sería:

ACU = (1*4%)+(1*7%)+(1*2%)+(1*11%)+(1*10%)+(1*9%)+(1*9%)+(1*33%)+(0.8*14%) = 97%

Una vez hallado el por ciento de avance de todos los casos de uso correspondientes a un módulo es posible calcular el por ciento de avance de este. Nótese que, hasta el momento, no se ha tenido en cuenta la complejidad de los casos de uso, será en este paso que ella juegue su rol. Durante la implementación toma más tiempo la elaboración de un caso de uso de complejidad alta que uno de complejidad media, así mismo, uno de complejidad media toma más tiempo que uno de complejidad baja. Esta razón permite deducir que para medir el avance de un módulo debemos tener presente este factor, ya que no es posible afirmar que el avance del módulo es el promedio de avance de los casos de usos que este tiene definido. En el caso del ejemplo anterior sería incorrecto pensar que al finalizar los casos de uso Gestionar afectaciones de locales y cupos y Consultar estado de cupos o locales el módulo se encuentra en el 50% de su construcción, pues los dos casos de uso que restan superan a estos en complejidad, por ende, en tiempo de elaboración.

Una vez conocido el Peso de caso de uso correspondiente a cada posible combinación (Ver Tabla 2), se podrá calcular el avance por módulo de la siguiente manera:

Avance por **M**ódulo (AM)= $Pcu_{alto} * PAcu_{alto} + Pcu_{medio} * PAcu_{medio} + Pcu_{bajo} * PAcu_{bajo}$

Donde $PAcu_{alto}$ es el promedio de avance de los caso de uso de complejidad alta, $PAcu_{medio}$ es el promedio de avance de los caso de uso de complejidad media y $PAcu_{bajo}$ es el promedio de avance de los caso de uso de complejidad baja.

En el caso del módulo de Capacidades el cual se muestra en la tabla anterior, el Avance por Módulo como no existen casos de uso de complejidad alta sería:

AM = (Pcu $_{alto}$ * PAcu $_{alto}$) ^{0} + Pcu $_{medio}$ * PAcu $_{medio}$ + Pcu $_{bajo}$ * PAcu $_{bajo}$

= 0.6 *97% + 0.4*97% = 97%

Hasta el momento se conoce cómo calcular el Avance por Módulo, pero aún falta por conocer cómo se calcula el **A**vance del **P**royecto.

Para explicar con mayor detalle el cálculo del avance del proyecto es necesario basarse en la Tabla 4, que visualiza un ejemplo extraído de la aplicación de la métrica en el SIGEP.

Total de semanas	**9**
Semana actual	**8**
Semanas que quedan	**1**

No	Módulos	Funcionalidades	Avance	Semanas para terminar	Estado del desarrollo
1	Requisas y decomisos	22	98%	0,2	En tiempo
2	Ubicación	3	98%	0,2	En tiempo
3	Pertenencias	7	96%	0,3	En tiempo
4	Novedades	10	97%	0,3	**En tiempo**
5	Medidas disciplinarias	6	90%	0,8	**En tiempo**
6	Capacidades	4	97%	0,3	En tiempo
7	Armamento	17	85%	1,4	**Atrasado**
8	Dotación del interno	5	95%	0,4	En tiempo
9	Educación	12	92%	0,7	En tiempo
10	Trabajo	6	86%	1,3	**Atrasado**
11	Deporte y Cultura	20	91%	0,8	En tiempo
12	Evaluaciones Técnicas	6	97%	0,3	En tiempo
13	Reportes	20	65%	4,4	**Atrasado**
		138	**91%**	**0,8**	En tiempo

Tabla 4 Estado actual de desarrollo

La Tabla 4 muestra un resumen semanal de la situación del proyecto, donde se puede observar un listado de todos los módulos que corresponden a la iteración y seguidamente la cantidad de funcionalidades a implementar en cada uno de ellos.

Nótese que en la tabla, se observa una columna con el nombre de Avance la cual carga automáticamente el Avance por Módulo calculado para cada caso. El Avance del Proyecto se expresa al final de la columna Avance y su cálculo está dado por:

$$\text{Avance del Proyecto} = \frac{\sum \text{Avance del Módulo}}{\text{Cantidad de módulos}}$$

En el caso de la Tabla 4 anterior quedaría:

$$AP = \frac{\sum 98\%+98\%+96\%+97\%+90\%+97\%+85\%+95\%+92\%+86\%+91\%+97\%+65\%}{13} = 91\ \%$$

Se puede observar también una columna que expresa las semanas que necesita el módulo para terminar. Si se hace un análisis de los datos podemos notar que es posible establecer la siguiente razón:

$$\frac{\text{Total Semanas por Terminar}}{100\%} = \frac{\text{Semana Actual}}{\text{Avance por Módulo}}$$

De aquí resulta la cantidad de semanas que se necesitan para implementar el módulo, y si a este valor se le resta la semana actual de desarrollo entonces se pueden obtener las semanas que quedan para terminar. Este cálculo se realiza con la expresión matemática:

$$\text{Semanas por Terminar} = \frac{\text{Semana Actual} * 100\%}{\text{Avance por Módulo}} - \text{Semana Actual}$$

Según la Tabla 4 para el caso del módulo Requisas y Decomisos este cálculo quedaría:

$$\text{Semanas por Terminar} = \frac{8 * 100\%}{98\%} - 8 = 0.2$$

Para finalizar se realiza de igual forma el cálculo general correspondiente a las semanas por terminar el software.

Nótese que existe otro campo cuyo nombre es Estado de Desarrollo, el valor por módulo de este campo depende totalmente de los resultados arrojados en el cálculo anterior. Si el resultado obtenido en Semanas por Terminar supera las semanas que quedan para la fecha de entrega establecida por el cliente, el módulo está **Atrasado**, esto significa que deben tomarse decisiones derivadas de esta información que permitan efectuar la entrega en la fecha acordada. En el caso contrario, es decir, si el resultado obtenido es menor que las semanas que quedan para la fecha de entrega establecida entonces el módulo está **En Tiempo**.

3.3. Análisis de los resultados

3.3.1. Gráfico: Histórico semanal del estado del proyecto

En la Tabla 5 se muestra un histórico por semanas del progreso de los diferentes módulos durante el proceso de implementación del software, donde se puede notar también como resultado del promedio semanal, el avance del software durante la semana.

Esta tabla sirve de análisis al líder de proyecto para conocer el avance por módulo de una semana a otra, e identificar los casos en los que se trabajó, en los que no y en los que el trabajo que se realizó fue poco.

Permite controlar y monitorear el proceso de implementación. De esta forma el líder cuenta con la información del estado del software en cada uno de los momentos medidos, lo cual le permite crear un histórico de su proceso de implementación.

Se pueden definir como propósitos del histórico mostrado en la Tabla 5 los siguientes:

- Mostrar un registro histórico del por ciento de avance semanal por módulos y del software.
- Contribuir al análisis del trabajo en el proyecto.

El histórico semanal luciría de la siguiente forma:

		semana1	semana2	semana3	semana4	semana5	semana6	semana7	semana8	semana 9
No	**Módulos**	**7-3-2008**	**14-3-2008**	**21-3-2008**	**28-3-2008**	**4-4-2008**	**11-4-2008**	**18-4-2008**	**25-4-2008**	**2-5-2008**
1	Requisas y decomisos	24%	34%	57%	62%	81%	89%	92%	92%	98%
2	Ubicación	13%	41%	42%	42%	79%	91%	93%	93%	98%
3	Pertenencias	48%	60%	77%	81%	93%	95%	96%	96%	96%
4	Novedades	44%	46%	53%	57%	73%	77%	85%	86%	97%
5	Medidas disciplinarias	30%	30%	41%	41%	50%	66%	82%	86%	90%
6	Capacidades	25%	47%	67%	75%	84%	92%	93%	93%	97%
7	Armamento	7%	18%	18%	19%	34%	58%	71%	86%	85%
8	Dotación del interno	11%	32%	55%	56%	93%	94%	95%	95%	95%
9	Educación	18%	48%	61%	76%	89%	89%	91%	92%	92%
10	Trabajo	15%	49%	49%	55%	77%	80%	86%	86%	86%
11	Deporte y Cultura	37%	61%	76%	79%	80%	85%	85%	91%	91%
12	Evaluaciones Técnicas	26%	53%	78%	83%	88%	91%	91%	91%	97%
13	Reportes					26%	31%	46%	65%	65%
		25%	**43%**	**56%**	**61%**	**73%**	**80%**	**85%**	**89%**	**91%**

Tabla 5 Histórico semanal del estado de los módulos

3.3.2. Gráfico: Histórico del estado de avance del proyecto

Para analizar el avance durante la implementación de un equipo de proyecto, la métrica más eficiente es graficar el avance total, un ejemplo se muestra en la Figura 4.

Esta gráfica permite observar las tendencias a lo largo del tiempo. Nótese que durante las primeras semanas del desarrollo existe un avance relativo alto. Luego se observa un pequeño avance en la semana 4 respecto a la semana 3, durante la cual se efectuaba en la Universidad los IV Juegos Deportivos, que como es de notar se afectó el proceso de desarrollo debido a que la mayoría de los miembros del equipo son estudiantes de la UCI. Luego se observa nuevamente un avance relativamente alto hasta aproximadamente el 80% del avance en el cual comienza la etapa de pruebas y donde el avance es un poco más lento.

Una buena tendencia sería si se obtuviera una línea recta, pero esto solo es posible bajo condiciones ideales, las cuales son muy difíciles de obtener debido a que los miembros del grupo de desarrollo tienen poca experiencia y no se dedican al trabajo a tiempo completo. Esto permite especificar que el método propuesto no es exacto ya que en él influyen muchos factores y las variables son muy cambiantes.

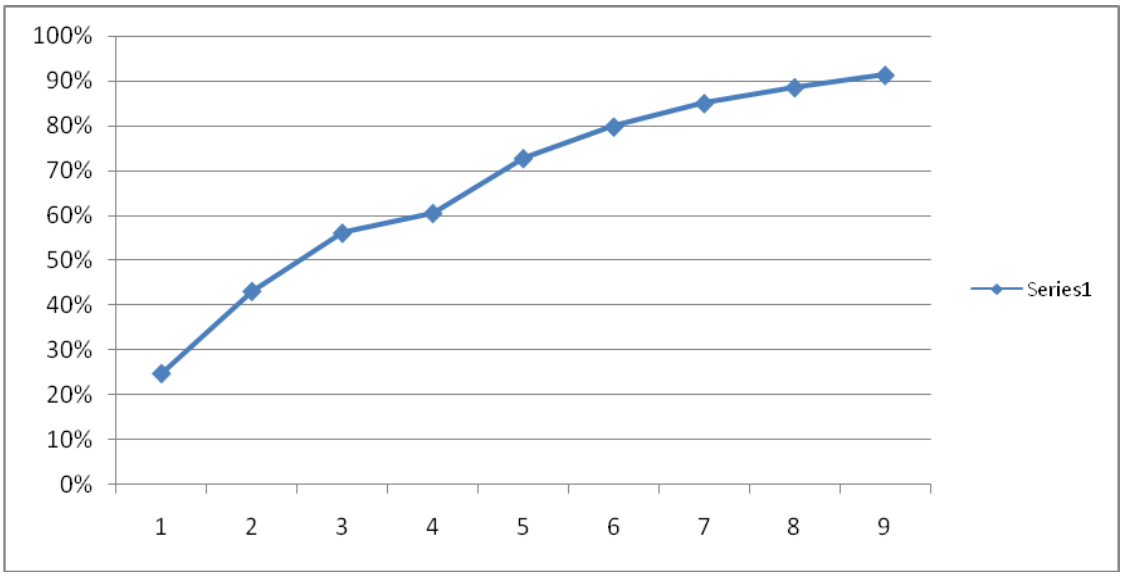

Figura 5 Histórico del estado de avance

3.3.3. Gráfico: Estado de avance por módulos

La forma más cómoda, y visual de lograr un seguimiento del avance por módulos, es a través de los gráficos del análisis del estado de avance por módulos. Un ejemplo puede ser el que se muestra en la Figura 6. En este gráfico se muestra claramente en qué estado de avance se encuentran los módulos y por tanto advierte al jefe del proyecto en cuales módulos tiene que volcar esfuerzos e incluso si es necesario tomar decisiones derivadas de la información brindada.

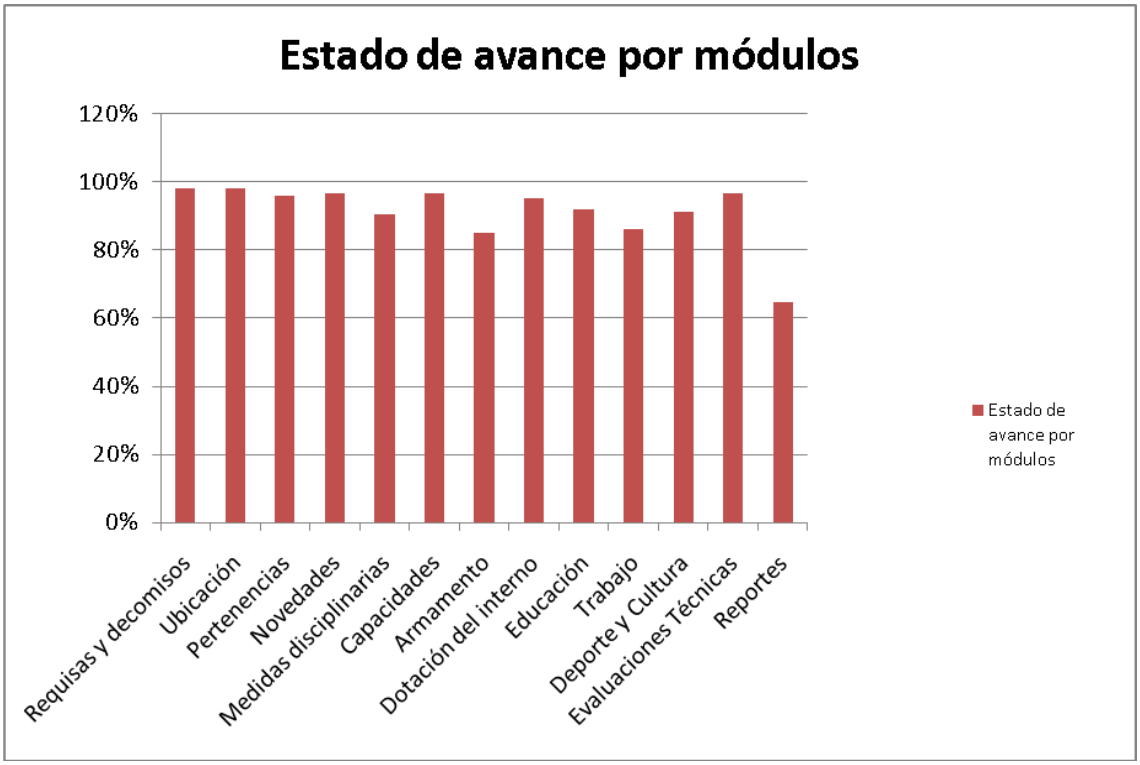

Figura 6 Estado de avance por módulos

Se logra realizar un análisis del estado de los módulos, por ejemplo el gráfico muestra que muchos módulos están prácticamente terminados mientras que existen otros que están más lejos de su culminación.

Este gráfico es muy útil a la hora de decidir en qué módulo debe hacerse un mayor esfuerzo e incluso si es necesario incluir más trabajadores para lograr terminar este en tiempo.

3.3.4. Gráfico: Estado de avance semanal de los módulos del proyecto

Propósito

Mostrar el avance semanal por módulo, en lo que se refiere a:

- Estado de avance del trabajo semanal por módulos.
- Contribuir al análisis semanal del trabajo en el módulo.

El resumen semanal luciría de la siguiente forma:

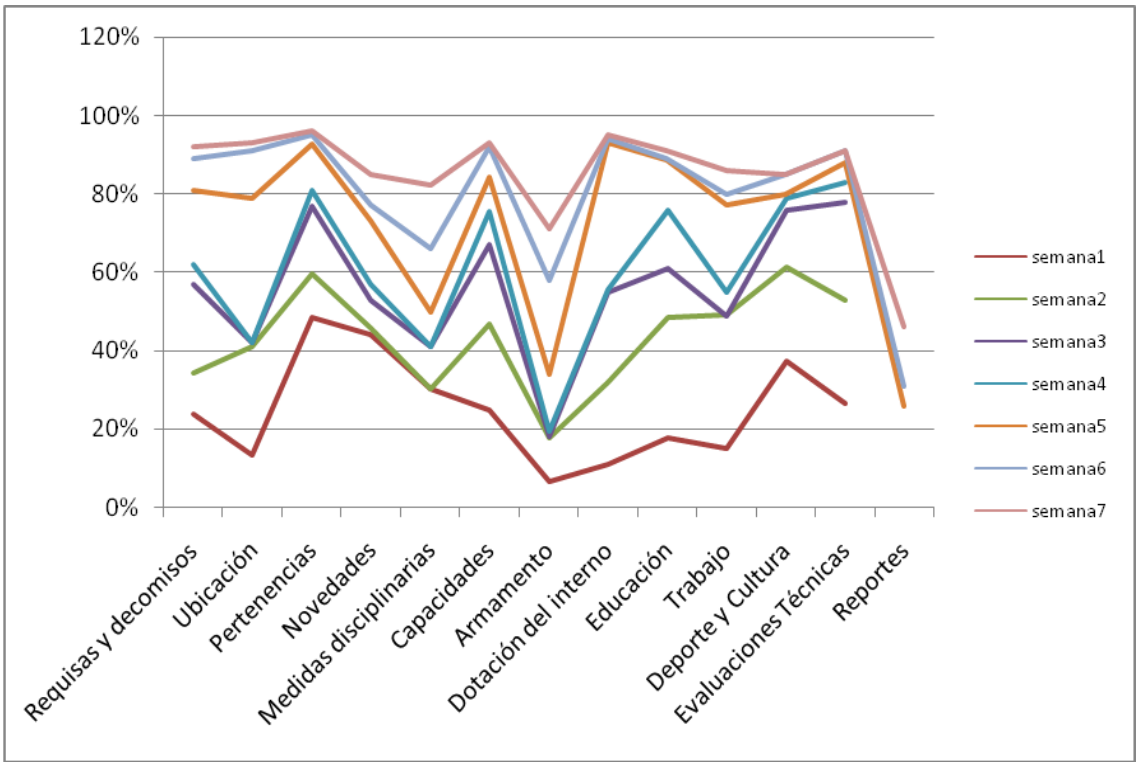

Figura 7 Estado semanal de avance de los módulos del proyecto

Los resúmenes semanales del proyecto proveen una síntesis del estado del proyecto desde el punto de vista del trabajo realizado en la semana.

Es recomendable que estos resúmenes se analicen semana por semana y que se use como punto de partida para el análisis de la semana siguiente, aunque esto no exime que los resúmenes se obtengan cada vez que el usuario los desee. Para poder estar al tanto de la evolución del proyecto se recomienda hacer una reunión semanal en la cual el Jefe de proyecto analiza el estado del proyecto

tomando como base estos resúmenes. También puede ser gratificante para los desarrolladores ver gráficamente el estado de avance de los módulos en los cuales trabaja.

3.4. Conclusiones

Con el desarrollo de este capítulo se expusieron las características fundamentales del diseño matemático de la métrica. Se presentó el análisis de los resultados obtenidos de la aplicación de la métrica en el Proyecto SIGEP. La métrica brinda información clara y precisa obtenida mediante el tratamiento gráfico de los resultados. Su aplicación práctica ha permitido tomar medidas correctivas a tiempo y almacenar los resultados para la toma de decisiones en futuras tareas o proyectos.

CONCLUSIONES

Una vez definida y aplicada la métrica se arribaron a las siguientes conclusiones:

1. Quedaron definidas las tareas del flujo de implementación correspondientes a la arquitectura utilizada y a las características del equipo del SIGEP.

2. Se creó y aplicó una métrica que permitió cuantificar el estado de avance del flujo de implementación y mantuvo informado tanto al cliente como a la dirección general del proyecto y al equipo de desarrollo.

3. Con la aplicación práctica de la métrica en la gráfica Histórico del estado de avance del proyecto es visible que a partir del 80% del avance del software, cuando comienza la etapa de pruebas, en la medida que aumenta el tiempo el ritmo de crecimiento es menor y no se alcanza el 100% de avance. Dado estos resultados, se puede concluir que las pruebas requieren más tiempo y por tanto el peso debe ser mayor para esta actividad.

4. Se comprobó que el equipo de desarrollo del SIGEP en 11 semanas de trabajo solamente puede desarrollar de 130 a 140 casos de uso.

5. Se creó un punto de partida para en un futuro desarrollar un método empírico de medición de software.

6. Se sentaron las bases para comenzar a obtener registros históricos de los proyectos desarrollados en la UCI.

RECOMENDACIONES

1. Utilizar el conocimiento asociado y la información recogida para la toma de decisiones en proyectos similares.

2. Evaluar los pesos de las tareas de implementación y valorar los tiempos que se dedican a la etapa de pruebas.

BIBLIOGRAFÍA

ARIAS, O. Y. A. *"Proyecto Técnico de Asesoría Especializada, Colaboración Médica Odontológica, Comunicación Institucional y Solución Tecnológica para apoyar la modernización del Sistema Penitenciario de la República Bolivariana de Venezuela"*. 2006. p.

BRITO, D. R. and H. F. DÍAZ. *Análisis del Método de Estimación empleado para el desarrollo del proyecto SIGEP*. Ciudad de la Habana, Universidad de las Ciencias Informáticas, 2007. p.

CAPUCHINO, A. M. M. S. *"Control y gestión de proyectos software, Unidad 4: Estimación de Proyectos Software"*. 1996. p.

CERRILLO, D. *"ESTIMACIÓN DEL SOFTWARE"*. 1999. p.

ESCORIAL, J. S. *"Calidad de Software: Medidas del Proceso"*. 2006. p.

FEBLES, A. *Un modelo de Referencia para la Gestión de Configuración en la PYME de Software*. Ciudad de la Habana, Instituto Superior Politécnico "José Antonio Echeverría", 2003. p.

HERNÁNDEZ, S. E. B. *"Métricas de estimación de tamaño: Puntos de Caso de Uso"*. 2002. p.

HUMPHREY, W. S. *"A discipline for software engineering"*. Addison-Wesley, 1995. p.

HURTADO, L. L. *"La primera comunidad libre donde aprender y compartir"*. 2007. p.

KAN, S. H. *"Metrics and Models in Software Quality Engineering"*. Addison-Wesley, 2000. p.

KELVIN, L. *"Capacitación de Vanguardia para Desarrolladores"* 2000. p.

MARCELO, J. *"Técnicas de estimación y seguimiento"*. 2006. p.

MARÍA, M. S. C. A. *"Estimación de Proyectos Software"*. 1998. p.

NUSENOFF, R. E. and D. C. BUNDE. *" A Guide Book and Spedsheet Tool for a Corporate Metrics Program"*. Holanda, 1993. p.

PIMENTEL, L. A. and I. P. RIVERO. *ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB.* Ciudad de la Habana, Universidad de las Ciencias Informáticas, 2007. p.

Bibliografía

PRESSMAN, R. S. *"Ingeniería del Software. Un Enfoque Práctico"*. 1998a. p.

RUP. *Rational Unified Process*. 2003. p.

WESTFALL, L. L. *“Software metrics that meet your information needs”*. 1995. p.

ZUSE, H. *“History of Software Measurement”*. 1995. p.

GLOSARIO

Ámbito del proyecto: Se definen los objetivos del proyecto, identifica funciones primordiales que debe llevar a cabo el software e intenta limitar esas funciones de manera cuantitativa.

Analogía: Semejanza existente entre las cosas que se comparan.

Atributo: Cualidad de ser.

Casos de Prueba: Un caso de prueba será un conjunto de entradas con datos de prueba, unas condiciones de ejecución, y unos resultados esperados

Capa de presentación: Generalmente se identifica como la capa web. Esta capa debe ser tan fina como sea posible. Además, debe permitir diferentes capas de presentación, tales como una capa web y/o fachadas de servicios web remotos, sobre una simple y bien diseñada capa de negocio.

Capa de servicios de negocio: Es la responsable de delimitar las transacciones y proveer un punto de entrada para las operaciones sobre el sistema. Esta capa no debería tener conocimiento sobre lo concerniente a la presentación y debería ser reutilizable.

Capa de acceso a datos: Esta capa presenta un conjunto de interfaces independiente de la tecnología de acceso a datos, que son usadas para buscar y persistir los objetos persistentes. Estas son interfaces del patrón de comportamiento definido en los patrones GOF, llamado Estrategia (Strategy) (GRAND). Las implementaciones de estas interfaces usan Hibernate para persistir las entidades persistentes del dominio, aunque estas puedan ser implementadas usando cualquier tecnología de mapeo objeto-relacional (ORM) o la capa de abstracción para JDBC que trae Spring Framework. La capa de acceso a datos no debería contener ningún tipo de lógica de negocio.

Componente: Que forma parte de alguna cosa o de su composición.

Estándar: Que posee el tamaño, la forma o cualquier otra característica que sigue al modelo. Se aplica a lo que se produce en serie. Que sigue una tendencia muy extendida. Aquello que se considera modelo.

Glosario

Estimación: Es un proceso continuo que acompaña a todo el desarrollo del proyecto y comienza usando pocas variables en un nivel alto de abstracción.

Experto: Persona que aporta conocimientos o experiencia específica con respecto a una organización, proceso, actividad o materia que se vaya a auditar.

Fiabilidad: Probabilidad de que algo funcione bien o sea seguro.

Gestión: Actividades coordinadas para dirigir y controlar una organización.

Hito: Suceso o acontecimiento que sirve como punto de referencia.

Implementación: La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos como los encontrados con las distribuciones beta del sistema

Medición: Es el proceso mediante el cual se le asignan números o símbolos a atributos de entidades del mundo real de tal manera que las describan de acuerdo a reglas claramente definidas.

Método heurístico: Método o procedimiento mediante el cual se puede deducir o inducir la verdad.

Métrica: Una forma de medir y una escala, definidas para realizar mediciones de uno o varios atributos

Modelo empírico: Modelo de regresión que relaciona esfuerzo con tamaño o funcionalidad.

Modelo: Cosa que ha de servir de objeto de imitación. Objeto, construcción u otra cosa con un diseño del que se reproduce más iguales. Esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y estudio.

Módulo: Encapsula un conjunto de funciones que debe realizar el sistema, las cuales son agrupadas por tener características muy similares y se definen en la etapa de diseño.

Normas: Modelo, patrón o regla de obligado cumplimiento.

Planificación: Es la actividad fundamental del gestor de proyecto que comprende la formulación de lo que hay que realizar para obtener una finalidad que será precisamente la del sistema que

estamos planificando (Decidir + Hacer).Control: Es inspección, fiscalización, intervención. Esta actividad no se centra solamente en realizar planes, sino controlar su ejecución y puesta en práctica.

Proceso: Cualquier actividad, o conjunto de actividades, que utiliza recursos para transformar entradas en salidas.

Productividad: Relación entre la cantidad de bienes y servicios producidos y la cantidad de recursos utilizados

Proyecto: Proceso único consistente en un conjunto de actividades coordinadas y controladas con fechas de inicio y de finalización, llevadas a cabo para lograr un objetivo conforme con requisitos específicos.

Puntos de caso de uso: Es un método de estimación de esfuerzo de un proyecto de desarrollo de software a partir de los casos de uso.

Puntos de función: Miden la aplicación desde una perspectiva del usuario, dejando de lado los detalles de codificación. Se define como una función comercial del usuario final.

Recurso: Procedimiento o medio del que se dispone para satisfacer una necesidad, llevar a cabo una tarea o conseguir algo. Pueden ser personas o recursos materiales.

Requisito: Condición necesaria para que algo se cumpla.

Revisión: Actividad emprendida para asegurar la idoneidad, la adecuación y eficacia de la materia objeto de la revisión, para alcanzar unos objetivos establecidos.

Riesgo (software): Proximidad a un daño o peligro.

Rol: Define el comportamiento y responsabilidad de un individuo o grupo de individuos.

Sistema: Conjunto de elementos mutuamente relacionados o que actúan entre sí.

Usabilidad: Es la característica de un sistema que pretende ser utilizado por el tipo o tipos específicos de usuario/s, la tarea o tareas que para las cuales el sistema se ha hecho y el contexto en el

que se da la interacción. El "grado de usabilidad" de un sistema es, por su parte, una medida empírica y relativa de la usabilidad del mismo.

Variable: Magnitud que puede tener un valor cualquiera