UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 9

"Estrategia para aplicar Proceso Personal de Software en los proyectos del polo productivo de video y sonido digital"

TRABAJO DE DIPLOMA PARA OPTAR POR EL

TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: Carlos Yosvel Alayón Morejón

TUTOR: Ing. Mabel Guerra Saumell

Ciudad de la Habana, julio del 2008
"Año 50 de la Revolución"

AGRADECIMIENTOS

La tesis es un punto culminante en el mundo académico de cualquier estudiante, pero también es cierto que no deja de ser un eslabón más en la cadena de la vida, sin duda alguna esta tesis representa un punto crucial de una etapa muy enriquecedora. Todo lo que soy se lo debo agradecer a:

A mis padres, hermana, tías y abuelas por compartir y dedicar gran parte de sus vidas conmigo y por darme aliento para la ardua tarea que es estudiar.

A mi esposa Giselle por haberme ayudado infinitamente, ayudándome en todo momento incondicionalmente.

A mis amigos con quienes he compartido momentos increíbles que siempre llevaré en mi corazón, quienes han enriquecido mi vida con su cariño y alegría. Gracias por recordarme que hay personas valiosas en el mundo.

A mis profesores, que compartieron conmigo sus conocimientos y su amor por la informática.

A mi querida tutora Mabel por el apoyo tan grande y la amistad que me brindada.

A la Revolución cubana por haber inculcado tantos principios y valores en nosotros los jóvenes y haber creado esta hermosa ciudad universitaria.

DEDICATORIA

Se la dedico a mi familia, amigos y especialmente a mi hijo que esta por nacer.

RESUMEN

Para desarrollar la industria cubana de software es preciso contar con procesos bien definidos y un personal competente, entrenado en una disciplina personal y de trabajo en equipo. El Proceso de Software Personal (PSP) contribuye a alcanzar disciplina en el trabajo individual. Este trabajo propone un conjunto de consideraciones para introducir las prácticas de PSP, de forma paulatina e incremental en la formación de los profesionales en los proyectos del polo productivo de video y sonido digital de la facultad 9 en la UCI; con el objetivo de crear las bases para introducir métodos disciplinados de trabajo para cada persona y cumplir con los requerimientos de la industria. Además se propone la herramienta Process Dashboard, para soportar las prácticas de disciplina personal.

El PSP esta formado por 7 niveles, los cuales pasan por diferentes fases (planificación, diseño, codificación, compilación, prueba y postmortem) del desarrollo de proyectos, generando artefactos (Registro de Tiempo, Registro de Defectos, Formulario Propuesta de Mejora del Proceso, Resumen del Plan de Proyecto, etc.); que guardan datos necesarios para mejorar la calidad del producto y el rendimiento de cada integrante de un equipo de desarrollo de software.

PALABRAS CLAVES

Software, Proceso de Software Personal, proyecto MENPET, métricas, formularios, artefactos, proyecto UCI_TV, calidad del software.

ÍNDICE

ÍNDICE DE TABLA

ÍNDICE DE FIGURA

INTRODUCCIÓN

El desarrollo de software se ha convertido en un tema crítico en la sociedad moderna mundial. Todos parecen necesitar mejores software en el menor tiempo posible y a menor costo. Los métodos intuitivos de desarrollo de software que se usan actualmente, son básicamente aquellos que los propios individuos "artesanalmente" siguen, los cuales sólo servirán mientras la sociedad pueda tolerar la falta de predicción que ellos acarrean, es por ello que se hace cada vez más imponente el tema de la calidad de los productos de software que se producen en todo el mundo.

Existe una tendencia internacional a la producción de software teniendo en cuenta la aplicación de estrategias, pero esta práctica se puede decir que no está generalizada en todos los países. En el caso de Cuba, atendiendo al nivel de desarrollo propio, se trabaja en una fase "artesanal" donde aún no se definen procedimientos para medir los estándares de calidad.

La coyuntura económica de Cuba favorece en este momento el desarrollo de sistemas, no solamente para consumo interno de una organización sino también para su comercialización a terceros e incluso para la exportación. La evolución permanente de la tecnología informática, y consecuentemente, de los principios y técnicas de desarrollo de sistemas, impacta constantemente la formación de los profesionales en informática, entre ellos, quienes construyen software. En Cuba se ha construido la Universidad de las Ciencias Informáticas (UCI) la cual tiene la tarea de ser la vanguardia del desarrollo del software.

En la UCI es necesario obtener software de calidad, utilizando metodologías o procedimientos, estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. La aplicación del Proceso de Software Personal (PSP) es una de las necesidades de la UCI para lograr proyectos con la calidad requerida. En la UCI se cuentan con los proyectos del polo productivo de video y sonido digital de la facultad 9 los que presentan una **situación** crítica debido a:

1. Entregas tardías de los productos.
2. Una planificación de trabajo deficiente.
3. Poco aprovechamiento de los recursos.

4. Lentitud en el desarrollo de los proyectos.
5. Ineficiencia en el desarrollo de los proyectos.

En los proyectos del polo productivo de video y sonido digital se aplica de forma intuitiva el PSP, no se tienen datos históricos para verificar la eficiencia de los integrantes de los equipos y se entregan las actividades pero sin llevar un registro de quien las realiza, ejemplo el proyecto UCI TV. Todos estos problemas traen consigo clientes insatisfechos. Estas condiciones generan la necesidad de aplicar el PSP de forma eficiente en este polo productivo.

Después de analizar lo planteado anteriormente la **problemática** a la que se le dará solución en esta investigación es ¿Cómo aplicar el PSP en los proyectos del polo productivo de video y sonido digital de la facultad 9?

El **objeto de estudio** de esta investigación es el PSP en el desarrollo del software y el **objetivo general** que se quiere lograr en la investigación es plantear una estrategia que permita aplicar PSP en los proyectos del polo productivo de video y sonido digital de la facultad 9. Los **objetivos específicos** son los siguientes:

Analizar la factibilidad del uso de los niveles del PSP en el desarrollo de software.
Determinar cuáles son los niveles de PSP a aplicar en los proyectos del polo productivo de video y sonido digital de la facultad 9.

Hacer una propuesta de adaptación de métodos por niveles del PSP en los proyectos del polo productivo de video y sonido digital de la facultad 9.

El **campo de acción** de la investigación son los proyectos del polo productivo de video y sonido digital de la facultad 9 en la UCI.

Al finalizar la investigación se esperan los **resultados** siguientes:

1. Estrategia para aplicar el PSP en los proyectos del polo productivo de video y sonido digital.
2. Documentación sobre el PSP en el desarrollo de software en los proyectos del polo productivo de video y sonido digital.

Después de analizar todo lo anterior se puede plantear la **hipótesis de la investigación:** Si se lleva a cabo un profundo estudio del Proceso de Software Personal y un análisis del desarrollo de software en los proyectos del polo productivo de video y sonido digital de la facultad 9, se logrará definir una estrategia que permita aplicar PSP en los proyectos del polo productivo de video y sonido digital.

La variable independiente de la investigación es el estudio del Proceso de Software Personal y un análisis del desarrollo de software en los proyectos del polo productivo de video y sonido digital de la facultad 9, de ella depende la definición de la estrategia que permita aplicar PSP en los proyectos del polo productivo de video y sonido digital.

Para realizar la investigación y obtener los resultados esperados se deben cumplir estas **tareas de la investigación**:

1. Localizar la existencia del problema en el uso de los niveles de PSP en el desarrollo de software.
2. Investigar sobre los métodos de PSP utilizados en el desarrollo de software en los proyectos del polo productivo de video y sonido digital.
3. Investigar sobre el estado del arte sobre la aplicación de PSP en el desarrollo de software.
4. Estudiar el PSP en el polo productivo de video y sonido digital de la facultad 9.
5. Medir los resultados de la investigación.

Métodos teóricos:

Histórico-lógico: En la primera fase de la investigación se desarrolla un estudio del estado del arte de la problemática analizada; revisando de forma crítica cada uno de los documentos para poder resaltar la importancia de las diferentes estrategias de desarrollo de software que se llevan a cabo en la actualidad en el mundo, así como la eficiencia y/o deficiencias de cada una de estas.

Hipotético deductivo: Permite a partir del problema concreto plantear los objetivos específicos e hipótesis que en el transcurso de la investigación son resueltos siguiendo métodos científicamente bien fundamentados.

Analítico Sistémico: Plantea el problema como un todo, estudio de metodologías, esquemas del uso de las mismas en el desarrollo de software y cada uno de los métodos y herramientas que se utilizan.

Métodos Empíricos:

Entrevista: La entrevista se aplicará a conocedores del tema, personal capacitado en la muestra, o sea los profesores de la asignatura Gestión de Software en la facultad 9, para recopilar información sumamente importante para la investigación.

Con el objetivo de lograr resultados que le dieran credibilidad a esta investigación fue necesario seleccionar una población, de la cual se extrajo una muestra que fue analizada. De los resultados obtenidos a partir de este análisis se pudo inferir como se iba a comportar dicha población. A continuación se presentan:

Población: Los profesores de la asignatura Gestión de Software en la facultad 9.

Muestra: 2 profesores de la asignatura gestión de software en la facultad 9.

Unidad de estudio: Estrategia para aplicar lo planteado por el PSP.

Para la selección de la muestra fue utilizada la técnica de muestreo no probabilística, puesto que estas no garantizan que cada elemento de la población se haya incluido en la muestra, pero dentro de ella se utilizó la técnica del muestreo intencional, esta no ofrece resultados confiables, pero se realizó intencionalmente con datos iníciales y comprobaciones externas previas en la muestra seleccionada, por tanto se garantizó que los resultados fueran confiables.

Población: El proyecto MENPET del polo productivo de video y sonido digital de la facultad 9.

Muestra: 4 integrantes del proyecto MENPET.

Unidad de estudio: Desarrollo de software en los proyectos del polo productivo de video y sonido digital de la facultad 9.

Método de experto o método Delphi.

Es un método que permite obtener el consenso del grupo, maximizando las ventajas que presentan los métodos basados en grupos de expertos y minimizando sus inconvenientes.

Tareas previas antes de su aplicación:

- Delimitar el contexto y el horizonte temporal en el que se desea realizar la previsión sobre el tema en estudio.
- Seleccionar el panel de expertos.
- Explicar a los expertos en qué consiste el método.

Pasos del Método Delphi.

1. Selección de los expertos. Se escogen de tres (3) a seis (6) especialistas de acuerdo con lo que se quiere determinar.
2. Desarrollo de la primera ronda. A cada experto del grupo se le entrega una hoja de papel en la que deben responder sin comentarios cierta pregunta. Es de notar que en este método se parte, precisamente, de buscar las diferentes alternativas que se pretenden valorar. Luego de haberlas determinado, es que se procede con el proceso de consenso y selección de la mejor alternativa; lo cual se detalla a partir del paso número 4.
3. Lista final. Una vez determinadas las posibles acciones y expuestas en una pizarra se indaga entre los expertos de manera oral si consideran que esas son las que conformarán la lista final, o en caso contrario, si existen desacuerdos al respecto. Esto se toma como la segunda ronda.
4. Tercera ronda. Se entrega nuevamente a cada experto una hoja de papel donde deberá responder: "¿Qué ponderación o peso le daría a cada una de las alternativas, con el objetivo de ordenarlas según su influencia en le aspecto tratado, o según su importancia?".
5. Cuarta ronda. Se procede de manera verbal a indagar entre los expertos acerca de su desacuerdo o no con los resultados obtenidos para así lograr una discriminación correcta de los criterios y las acciones.
6. Acabado. A partir de aquí se toman las decisiones pertinentes de acuerdo con los resultados que en definitiva tuvieron lugar dado el desarrollo del método. Teóricamente, ya este habría terminado. Sin embargo, si no se hubiese llegado a un consenso, existiendo posturas muy

distantes, el moderador debería incluso habiéndose terminado todas las rondas confrontar los distintos argumentos para averiguar si se ha cometido algún error en el proceso.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción.

La meta de la ingeniería de software es construir productos de software, o mejorar los existentes; en ingeniería de procesos, desarrollando o mejorar procesos. Un proceso de desarrollo de software es un conjunto de personas, estructuras de organización, reglas, políticas, actividades y sus procedimientos, componentes de software, metodologías, y herramientas utilizadas o creadas específicamente para definir, desarrollar, ofrecer un servicio, innovar y extender un producto de software.

Para producir software es necesario que se tenga en cuenta una serie de procedimientos que sin duda ayudan a que los productos sean terminados en tiempo y con una mayor calidad.

Son precisamente los procedimientos de desarrollo de software el objetivo esencial sobre el cual gira la investigación realizada para el desarrollo del Capítulo 1. Pues constituye la base para el análisis y entendimiento del objeto de estudio que rige la misma, el cual se centra en los procedimientos desarrollo que plantea el PSP para aplicarlo al polo productivo de video y sonido digital.

1.2. Calidad del Software.

Los problemas fundamentales que han tenido que enfrentar los desarrolladores de software en la industria de la computación son los de la calidad del software. Durante los años sesenta, cada vez que se quería afrontar la mejora del software todo se dirigía a la mejora del código dejando olvidado todo lo que le rodeaba, ya en la década del setenta, este tema fue motivo de preocupación para especialistas, ingenieros, investigadores y comercializadores de software, los que fueron mejorando los procesos que se habían definido hasta el momento con el objetivo de mejorar la calidad que seguía siendo aún un tema crítico en el mercado del software, ya en los años ochenta se comenzaron a tener en cuenta los aspectos de especificaciones, diseño, evaluación y gestión del software. Pese a estos cambios realizados, en la actualidad persiste la problemática, ya que el nivel de complejidad del software va aumentando y no se han alcanzado los niveles de calidad deseados.

"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" [1]

"El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas."[2]

Para nadie es un secreto que la calidad es sinónimo de eficiencia, flexibilidad, corrección, portabilidad, usabilidad, seguridad, integridad, etc. La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Es posible medir los principales atributos que forman o caracterizan a un software de buena calidad. La idea es que la calidad del software se caracteriza por ciertos atributos: fiabilidad, flexibilidad, robustez, comprensión, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia, y sin lugar a dudas es posible medir cada uno de ellos, y por consiguiente, caracterizar o medir la calidad del software en cuestión. Para cada atributo a medir, hay una medición confiable (objetiva) que puede llevarse a cabo. La idea es que, dado que el atributo deseable es difícil de medir, se mide otro atributo que está correlacionado con el primero, por ejemplo:

1. La fiabilidad (software confiable, pocos errores) se mide a través del número de mensajes de error, mientras más mensajes existan, contiene entonces menos errores este software.
2. La flexibilidad (capacidad de adaptación a diferentes tipos de uso, a diferentes ambientes de uso) o adaptabilidad.
3. La robustez (pocas fallas catastróficas, el sistema "no se cae") se mide a través de pruebas y uso prolongado.
4. La comprensión (capacidad de entender lo que el sistema hace) se mide viendo la cantidad de comentarios que posee el software, y la extensión de sus manuales de usuarios.
5. El tamaño se mide en bytes, o sea el espacio que ocupa en memoria, (esta medición no tiene objeción, se mide lo que se quiere medir).
6. La eficiencia de un programa se mide en segundos, es la rapidez en su ejecución (esta medición no tiene objeción, se mide lo que se quiere medir).
7. La modularidad de un programa se mide contando el número de módulos que lo conforman.
8. La complejidad de un programa se mide contando el número de anidaciones en expresiones o postulados, (es lo que se le llama complejidad ciclomática).

9. La portabilidad es la facilidad con que se eche a andar en otro sistema operativo distinto al que fue creado. Se mide preguntando a usuarios que han hecho estos trabajos.
10. La usabilidad de un programa es alta cuando el programa aporta gran valor agregado al trabajo. "Es indispensable contar con él". Se mide viendo qué porcentaje de las necesidades cubre ese programa.
11. La reutilización de un programa se mide por la cantidad de veces que partes del mismo se han reutilizado en otros proyectos de desarrollo de software.
12. La facilidad de uso (ergonomía) caracteriza a los programas que no cuesta trabajo aprender, que se amoldan al modo intuitivo de hacer las cosas. Se mide observando la cantidad de pantallas que interaccionan con el usuario, y su sofisticación.

Pueden llegar a la conclusión que en vez de medir la calidad del producto, pueden medir la calidad del proceso. Tener un buen proceso implica producir software de buena calidad. El problema es que no se sabe cómo deducir cuál proceso producirá buena calidad en el software. En ocasiones se recurre a procesos que suenan o se ven razonables, o que han sido ensayados en otros lados con éxito, o que están dados por algún estándar o comité internacional, sin embargo mientras el proceso que se lleve a cabo sea adaptable a el proyecto y sea totalmente fiable entonces se obtendrá un software de calidad.

La obtención de un software con calidad implica la utilización de metodologías o procedimientos, estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. La innovación tecnológica y la adecuación a las nuevas tecnologías plantean el gran reto futuro. Pese a lo arriesgado de hacer predicciones, está claro que la medición se plantea como uno de los principales campos de investigación en Calidad del Software.

La especificación y evaluación integral de la calidad de los productos de software es un factor clave para asegurar que la calidad sea la adecuada. Esto se puede lograr definiendo de manera apropiada las características de calidad, teniendo en cuenta el propósito del uso del producto de software. Es importante especificar y evaluar cada característica relevante de la calidad de los productos de software, mientras esto sea posible, utilizando mediciones validadas o de amplia aceptación, que hagan técnicamente transparente esta actividad. [2]

1.3. Modelos de Calidad del Producto.

En el mundo existe infinidades de modelos de calidad por los cuales se rigen todas las empresas productoras de software para poder vender sus productos, en Cuba las empresas que no están certificadas para poder utilizar estas normas y vender los productos de software en el mercado mundial, pero muchas empresas del mundo las utilizan y obtienen resultados satisfactorios.

A fines de la década del ochenta e inicios de la década del noventa se ha puesto mucho énfasis en los conceptos de calidad de producto y satisfacción del usuario desde diversos enfoques, y particularmente a la valoración y certificación de la calidad de procesos con los bien conocidos CMMI y SPICE (entre otros), sin embargo, también es conocido que los modelos de calidad ya eran reconocidos en la comunidad científica a fines de la década del 70 como los descriptos por McCall y Boehm.

Estos modelos describen a la calidad del producto usando un enfoque de descomposición. El modelo de McCall, por ejemplo fue originalmente desarrollado para la Fuerza Aérea de los EE.UU. y se promocionó su uso para evaluar la calidad del software dentro del DoD (Departamento de defensa de los EE.UU.). En estos modelos, los evaluadores se concentran en los atributos de calidad claves para el producto de software, en consideración de un punto de vista de usuario.

Dada la naturaleza lógica del producto, se asume que la calidad de un sistema de software depende sobremanera de la calidad del proceso usado para desarrollarlo. Los modelos de evaluación y mejora de procesos y su estandarización, han tomado un papel determinante en la identificación, integración, medición y optimización de las buenas prácticas existentes en la organización y desarrollo del software.

Han surgido continuamente con el desarrollo de la informática y las comunicaciones una serie de herramientas, técnicas y modelos que facilitan a las organizaciones, encargadas de las tecnologías de la información, generar productos que cumplan las expectativas del cliente e incluso las rebasen, herramientas que prometen ser la solución a los problemas de calidad, costo y tiempos de desarrollo.

Otras normas, directivas, modelos o estándares más usados en la actualidad son básicamente las siguientes:

Familia ISO 9000

ISO 9000 es un conjunto de estándares internacionales para sistemas de calidad. Diseñado para la gestión y aseguramiento de la calidad, especifica los requisitos básicos para el desarrollo, producción, instalación y servicio a nivel de sistema y a nivel de producto. Esta serie de normas pueden aplicarse a cualquier industria, producto o servicio, y consta de requisitos y directrices para establecer sistemas de calidad dentro de una organización, permitiéndole efectuar transacciones con cualquier organización en el mundo, con menor riesgo y mayor confianza, son normas prácticas burocráticas que buscan el logro de la calidad .[2]

Las normas ISO 9000 tiene tres componentes básicos: administración, sistema de calidad y aseguramiento de la calidad.

- Administración

ISO 9000 provee un sistema para alcanzar el progreso de la organización mediante la realización de metas estratégicas, comprensión de las necesidades de los usuarios, productividad, etc., por medio de acciones correctivas y preventivas.

- Sistema de calidad

ISO 9000 requiere que la organización documente los procedimientos y los ponga en práctica, de tal forma que si se realiza un cambio, también se registre por escrito. Es necesario contar con una base documental que se ajuste a la realidad al cien por ciento.

- Aseguramiento de la calidad

ISO 9000 es dinámico, ya que se envuelve en muchas facetas de la organización, como por ejemplo, el establecimiento y documentación de sistemas de ventas, de compras, de producción, de almacenamiento, de embarcación e ingeniería, etc.

Objetivos de las normas ISO 9000

- Establecer sistemas de aseguramiento de la calidad, que garanticen el buen funcionamiento de la empresa y satisfacción de sus clientes.

- Definir el sistema de administración de las actividades que pueden influenciar la calidad de un producto.

- Ayudar a desarrollar: un sistema de calidad a nivel mundial, productos de calidad consistente y buena relación con los clientes.

CMMI

El modelo CMMI constituye un marco de referencia de la capacidad de las organizaciones de desarrollo de software en el desempeño de sus diferentes procesos, proporcionando una base para la evaluación de la madurez de las mismas y una guía para implementar una estrategia para la mejora continua de los mismos.

SPICE

SPICE es un emergente estándar internacional de evaluación y determinación de la capacidad y mejora continua de procesos de ingeniería del software, con la filosofía de desarrollar un conjunto de medidas de capacidad estructuradas para todos los procesos del ciclo de vida y para todos los participantes. Es el resultado de un esfuerzo internacional de trabajo y colaboración y tiene la innovación, en comparación con otros modelos, del proceso paralelo de evaluación empírica del resultado. De acuerdo con la investigación realizada se puede apreciar que la familia ISO y CMMI son los modelos más populares en el mundo de la ingeniería del software, seguido por SPICE. [2]

La arquitectura del modelo organiza las prácticas en números de categorías usando diferentes tipos de aproximaciones. La arquitectura distingue entre:

- Prácticas base, son las actividades esenciales de un proceso especifico, agrupado por categorías de procedimientos y procesos de acuerdo al tipo de actividad que direccionan.
- Prácticas genéricas, aplicables a cualquier proceso, que representa las actividades necesarias para administrar el "proceso" y mejorar su potencialidad.

El modelo agrupa a los procesos en cinco categorías:

- Procesos Cliente - Proveedor (Customer - Supplier) esta categoría consiste en los procesos que directamente impactan al cliente, al soporte de desarrollo y a la transición del software al cliente.
- Procesos de Ingeniería (Engineering) esta categoría consiste, a los procesos que directamente especifican, implementa, y mantienen un sistema, un producto de software y la documentación del usuario.
- Procesos de Proyecto (Project) esta categoría consiste en los procesos establecidos dentro del proyecto, coordinación y administración de los recursos para producir un producto o proveer un servicio para satisfacer al cliente.
- Procesos de Soporte (Support) esta categoría consiste en los procedimientos que establecen y soportan el desempeño de los otros procesos del proyecto.
- Procesos de la Organización (Organization) esta categoría consiste en los procesos que establecen las metas de negocio de la organización, los procesos de desarrollo y recursos que ayudan a la organización alcanzar dichas metas.

1.3.1 El modelo de madurez de las capacidades (CMM).

"Los procesos son como los hábitos: difíciles de establecer e incluso mucho más difíciles de romper" [3].Los procesos para desarrollar software a gran escala, pueden ser muy grandes y complejos. Pueden ser difíciles de definir e incluso mucho más difíciles de introducir. Por esta razón fue que el Software Engineer Institute (SEI) de la universidad Carnegie - Mellon desarrolló un ambiente de trabajo (framework) de madurez de procesos de software [4]. Este framework es una forma ordenada para las organizaciones de determinar las capacidades de sus procesos actuales y establecer prioridades en su mejora. Se hace a través del establecimiento de 5 niveles de prioridad progresivos logrando procesos de capacidades más maduros [5]. Por cada nivel se han definido los principios elementales o KPAs (Key Process Areas) que proveen las metas y ejemplifican las prácticas a llevar a cabo (figura 1).
CMM ha sido revisado y refinado por varios especialistas y representan el mejor juicio actual y el método más efectivo para conseguir los objetivos de cada nivel de madurez.

	Características	KPAs	Resultado
5 Optimizado	Bases cualitativas para una inversión capital continua en la automatización y mejoramiento de la retroalimentación de los procesos.	-Prevención de defectos -Administración de la tecnología de cambio -Administración del proceso de cambio	Productividad y calidad
4 Administrad	(cuantitativo) -Procesos medidos -Control estadístico razonable sobre la calidad del producto	-Medición y análisis de los procesos -Administración de la calidad	
3 Definido	(cuantitativo) -Costo y planificación confiables -Procesos definidos e institucionalizados -Rendimiento de la calidad mejorado pero impredecible	-Definición y mejoramiento de los -procesos organizacionales -Programas de entrenamiento -Administración integrada de software -Coordinación intergrupal -Inspecciones entre compañeros -Ingeniería del producto de software	
2 Repetible	(intuitivo) -Procesos dependientes de los individuos - Costo y calidad altamente variables - Métodos y procedimientos informales y "ad hocs"	-Administración de los requerimientos -Planificación y vigilancia de los proyectos de software -Administración de la subcontratación de software -Aseguramiento de la calidad del software -Administración de la configuración de software	
1 Inicial	-Ad hoc -Caótico -costos, planificación y rendimiento de la calidad impredecibles		

Figura 1: Representación esquemática de CMM. [6]

1.4. Medidas de Calidad.

Para elevar la calidad de un producto, la misma debe ser medida de alguna manera. La interrogante principal es la siguiente: ¿qué se mide y cómo se utilizan los datos resultantes? Antes de que se puedan usar los datos, se tiene que decidir sobre las medidas de calidad. Los defectos corregidos en un producto terminado indican la efectividad del proceso de eliminación de defectos. Para manejar la calidad del trabajo, se tiene que medir todo el trabajo, no solamente el producto obtenido Las medidas de trabajo disponible son: rendimiento en la eliminación de defectos, costo de la calidad (COQ) tasa de tiempo por fase y el índice de calidad del proceso. [7]

1.4.1 Calidad del rendimiento.

La eliminación de defectos puede ser vista como una secuencia de filtros que progresivamente los van eliminando y que pueden ir definiendo medidas útiles de eliminación de defectos. El rendimiento mide la eficiencia de cada fase de filtrado en la eliminación de defectos. El rendimiento de una fase es el porciento de defectos del producto que son eliminados en esa fase. Por ejemplo, si un producto contiene 20 defectos al comienzo de una prueba de unidad y siete son encontrados durante la prueba, el rendimiento de la prueba de unidad sería de 100*7/20 = 35%. El rendimiento, sin embargo también considera los defectos introducidos, tanto como los eliminados en cada fase. Por tanto, si se comete un error cuando se están corrigiendo los primeros 6 defectos y se introduce un séptimo defecto, el cual es encontrado, se habrían encontrado 6 de los 20 defectos presentes en la entrada de la fase y se habría introducido y corregido uno más. El rendimiento de la fase estaría entonces basado en encontrar 7 de 21 posibles defectos para un 33% de rendimiento de la fase. [7]

El rendimiento de la fase puede ser calculado para cualquier fase. De esta manera, existe el rendimiento de pruebas de unidad, de revisiones de código, de compilación y otras. Existe también el rendimiento del proceso que se refiere al por ciento de defectos que fue eliminado antes de la primera compilación. Si 100 defectos fueron introducidos durante la fase de requerimiento, diseño y codificación, y si 68 de ellos fueron encontrados en todas las revisiones e inspecciones sin incluir la fase de compilación, el rendimiento del proceso sería de 68%.

El rendimiento del proceso puede ser calculado para otras fases, de manera que si los 100 defectos fueron introducidos en todas las fases antes de las pruebas del sistema y 28 más fueron encontrados durante la compilación, pruebas de unidad y pruebas de integración, entonces 68 + 28 = 96 defectos habrían sido encontrados antes de comenzar las pruebas del sistema. El rendimiento de proceso antes de las pruebas del sistema sería entonces de 96%. Cuando se usa el rendimiento del proceso sin tener en cuenta el nombre de una fase este se refiere al rendimiento antes de la compilación. Si el ambiente de desarrollo no tiene una fase de compilación entonces el rendimiento del proceso será medido sin incluir las pruebas de unidad. Por otra parte, cuando se incluye el nombre de una fase, el rendimiento de la fase puede ser calculado para cualquier fase. [7]

1.4.2 Coste de Calidad (COQ).

El coste de calidad incluye todos los costes acarreados en la búsqueda de la calidad o en las actividades relacionadas en la obtención de la calidad. Se realizan estudios sobre el coste de calidad para proporcionar una línea base del coste actual de calidad, para identificar oportunidades de reducir este coste, y para proporcionar una base normalizada de comparación. La base normalizada siempre tiene un precio. Una vez que se han normalizado los costes de calidad sobre un precio base, se tienen los datos necesarios para evaluar el lugar en donde hay oportunidades de mejorar los procesos. Es más, se puede evaluar cómo afectan los cambios en términos de dinero. [7]

Los costes de calidad se pueden dividir en costes asociados a la prevención, la evaluación y los fallos.

Entre los costes de prevención se incluyen:

1. Planificación de la calidad,
2. Revisiones técnicas formales,
3. Equipos de pruebas,
4. Formación

Entre los costes de evaluación se incluyen actividades para tener una visión más profunda de la condición del producto "la primera vez a través de" cada proceso. Algunos ejemplos de costes de evaluación:

- Inspección en el proceso y entre procesos,
- Calibrado y mantenimiento del equipo,
- Pruebas.

Los costes de fallo son los costes que desaparecerían sin no surgieran defectos antes del envío de un producto a los clientes. Estos costes se pueden dividir en coste de fallos internos y costes de fallos externos.

Los internos se producen cuando se detecta un error en el producto antes de su envío (retrabajo – revisión, reparación, análisis de las modalidades de fallos.) [7]

Los costes de fallos externos son los que se asocian a los defectos encontrados una vez enviado el producto al cliente (resolución de quejas, devolución y sustitución de productos, soporte de línea de ayuda, trabajo de garantía). [7]

Como es de esperar, los costes relativos para encontrar y reparar un defecto aumentan dramáticamente a medida que se cambia de prevención a detección y desde el fallo interno al externo.

1.4.3 Tasas de revisión.

Aunque el rendimiento y las métricas de COQ son útiles, ellas miden qué se hizo, no que se está haciendo. Para realizar un trabajo con calidad, se necesita medidas que guíen, qué hacen mientras lo estén desarrollando. En fases de diseño o revisiones de código, el factor principal que controla el rendimiento es el tiempo y el cuidado que el desarrollador tiene cuando está haciendo la revisión. La tasa de revisión y la medida de proporción de la fase provee una forma de seguir y controlar el tiempo de revisión. La métrica tasa de revisión es principalmente usada para revisiones de código e inspecciones y mide las líneas de código (LOC) o páginas revisadas por hora. Si, por ejemplo, se pasan 20 minutos revisando un programa de 100 LOC, la tasa de revisión sería de 300 LOC por hora. No existen tasas definitivas sobre cuál rendimiento de revisión es incorrecto. Sin embargo, Los datos de PSP muestran que altos rendimientos están asociados con bajas tasas de revisiones y que bajos rendimientos están relacionados con altas tasas de revisión. [7]

1.4.4 Tasa de tiempo por fase.

Un conjunto de medidas de calidad útiles es la proporción del tiempo utilizado en dos fases del proceso. Por ejemplo, la proporción del tiempo utilizado en la revisión del diseño contra el tiempo utilizado en el diseño. Para medir la calidad del proceso, PSP usa la proporción del diseño contra el tiempo de codificación, revisión del diseño contra tiempo de diseño y revisión de código contra tiempo de codificación. Se esperaría que el incremento en el tiempo de diseño tienda a incrementar la calidad del producto en correspondencia con la reducción del tiempo de codificación.

Aunque no hay un punto definido sobre cuándo se tiene una calidad pobre, una proporción de 1 a 1 en el diseño contra el tiempo de codificación parece ser un límite inferior razonable, con una proporción de

1.5 como la más óptima. Este punto óptimo varía considerablemente por los individuos y tipos de programas, pero una regla útil es que el tiempo de diseño debe ser, como mínimo, igual al tiempo de codificación. Si esto no sucede, se está haciendo probablemente una significativa cantidad de diseño mientras que se está codificando.

Otra proporción útil es el tiempo de revisión contra el tiempo de desarrollo. En el PSP, la pauta general es que se debe utilizar como mínimo la mitad del tiempo que se usa en la codificación para hacer las revisiones. Esto significa que por cada hora de codificación se debe utilizar como mínimo la mitad de una hora haciendo revisiones del código. La misma proporción sirve para el diseño contra el tiempo de revisión del mismo, requerimientos y tiempo de revisión de estos, etc. [7]

1.4.5 Índice de calidad del proceso (PQI)

Suponga que se siguen todas estas pautas, que se utiliza mucho tiempo diseñando y codificando y que se utiliza el 50% del tiempo en las revisiones de diseño y codificación. ¿Garantiza esto una alta calidad? Desafortunadamente no. Todo depende en cómo se utilice el tiempo.

Todo esto implica la necesidad de otras dos métricas. La primera es evaluar la calidad de las revisiones de código. Si se sigue el proceso de revisión de código y se usa una lista de chequeo para las revisiones de código se tendrá una revisión con un mayor rendimiento. El mejor indicador de esto es encontrar muy pocos defectos durante la compilación. En el PSP, es considerado como poco, encontrar 10 o menos defectos por KLOC durante la compilación. Sin una buena revisión de código, es bien difícil obtener ese número. Desafortunadamente, si su ambiente de desarrollo no tiene una fase de compilación, estas medidas no estarán disponibles. En este caso, se pueden usar los datos de los tipos de defectos de codificación encontrados en las pruebas de unidad y, en un equipo de TSP, los tipos de defectos de codificación encontrados durante las inspecciones y las pruebas de unidad. [7]

Las métricas en las revisiones de diseño son algo más complejas. Aunque el número de defectos encontrados en las pruebas de unidad puede ser una medida útil de la calidad del diseño, también incluye calidad del código. Si, sin embargo, los defectos por KLOC en la compilación estuvieran por debajo de 10, entonces la calidad es probablemente mejor y los defectos de las pruebas de unidad pueden medir la calidad del diseño. De esta manera, sin los datos de defectos durante la compilación no se puede interpretar las medidas de defectos encontrados durante las pruebas de unidad. Las

medidas sugeridas por PSP especifican que si los defectos por KLOC en la fase de compilación son menos de 10, una cantidad de 5 defectos por KLOC en las pruebas de unidad indican un diseño con calidad. Los criterios de PQI [7] son los siguientes:

1. Diseño/Tiempo de codificación = tiempo de diseño/tiempo de codificación, con rango desde 0.0 a 1.0.
2. Tiempo de revisión del diseño = 2*tiempo de revisión del diseño/tiempo de revisión, con un rango entre 0.0 y 1.0
3. Tiempo de revisión de código = 2*tiempo de revisión de código/tiempo de codificación, con un rango entre 0.0 y 1.0
4. Defectos de compilación/KLOC = 20/(10+defectos de compilación/KLOC), con un rango entre 0.0 y 1.0
5. Defectos de pruebas de unidad/KLOC = 10/(5+defectos de pruebas de unidad/KLOC), con un rango entre 0.0 y 1.0

El índice de calidad del proceso es obtenido multiplicando estos 5 valores entre si.

1.5. Control de la Calidad.

El control de la calidad es una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso del software para asegurar que cada producto cumple con los requisitos que le han sido asignados. El control de la calidad incluye un bucle de realimentación (feedback) del proceso que creó el producto. La combinación de medición y realimentación permite afinar el proceso cuando los productos de trabajo creados fallan al cumplir sus especificaciones. Este enfoque ve el control de la calidad como parte del proceso de fabricación.

Las actividades de control de la calidad pueden ser manuales, completamente automáticas o una combinación de herramientas automáticas e interacción humana. Un concepto clave del control de la calidad es que se hayan definido todos los productos y las especificaciones mensurables en las que se puedan comparar los resultados en cada proceso. El bucle de realimentación es esencial para reducir los defectos producidos.

Se puede ver el proceso de un software como la combinación de dos procesos competitivos: la introducción de defectos y la eliminación. La satisfacción de la eliminación de defectos en los

productos terminados viene de la diferencia entre estos dos procesos. Cambios relativamente pequeños en el rendimiento del proceso pueden causar grandes cambios en la satisfacción con la eliminación de defectos en los productos terminados.

Desde un punto de vista de control del rendimiento, el número final de defectos es el residuo del rendimiento de todas las fases del proceso. Si una fase sencilla para cualquier componente de un producto tiene bajo rendimiento, algún porciento de defectos que se olvide impactará en la fase subsiguiente y degradará el producto final. Esta degradación también impactará en los costos de mantenimiento, soporte y uso del producto, tanto como el costo de todas las mejoras subsecuentes del producto. Un trabajo de baja calidad tiene un efecto final, por tanto es importante que cada desarrollador maneje la calidad de su trabajo personal. La calidad en el trabajo no solo ahorrará tiempo y esfuerzo, sino que producirá ahorros durante el ciclo de vida del producto. [8]

1.5.1 La Garantía de la Calidad.

La garantía de la calidad consiste en la auditoría y las funciones de información de la gestión. El objetivo de la garantía de la calidad es proporcionar la gestión para informar los datos necesarios sobre la calidad del producto, por lo que se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos. Por supuesto, si los datos proporcionados mediante la garantía de calidad identifican problemas, es responsabilidad de la gestión afrontar los problemas y aplicar los recursos necesarios para resolver aspectos de calidad. [8]

1.5.2 Gestión de la Calidad del Software.

Algunos desarrolladores de software continúan creyendo que la calidad del software es algo en lo que empiezan a preocuparse una vez que se ha generado el código. Nada más lejos de la realidad. La garantía de calidad del software (Software Quality Assurance (SQA), Gestión de la calidad del Software) es una actividad de protección que se aplica a lo largo de todo el proceso de software.

La SQA engloba: (1) un enfoque de gestión de calidad; (2) tecnología de ingeniería de software efectiva (métodos y herramientas); (3) revisiones técnicas formales que se aplican durante el proceso del software; (4) una estrategia de pruebas multiescalada; (5) el control de la documentación del

software y de los cambios realizados; (6) un procedimiento que asegure un ajuste a los estándares de desarrollo del software (cuando sea posible), y (7) mecanismos de medición y de generación de informes. [8]

1.6. Actualidad en la calidad de software Cuba y la Universidad de las Ciencias Informáticas.

La industria del software ha emergido, crecido y fortalecido a tal punto que representa actualmente una actividad económica de suma importancia para todos los países del mundo. La industria del software en la mayoría de los países está formada por tejido industrial compuesto en gran parte por desarrolladoras de software que favorecen al crecimiento de las economías nacionales. La mayoría de empresas desarrolladoras de software son pequeñas (Tienen menos de 50 empleados) y desarrollan productos significativos que, para su construcción, necesitan prácticas eficientes de Ingeniería del Software adaptadas a su tamaño y tipo de negocio.

La Industria del Software actual afronta una crisis debido a una serie de factores entre los que se encuentra: insuficiente calidad del producto final, estimaciones de duración de proyectos y asignación de recursos inexactas, retrasos en la entrega de productos terminados, están fuera de control los costos de desarrollo y mantenimiento de productos, escasez de personal calificado en un mercado laboral de alta demanda, y una tendencia al crecimiento del volumen y complejidad de los productos.

Las piedras angulares del proceso de mejora continuo del desarrollo del software son: el personal, el proceso y la tecnología. Existen Modelos de calidad que se establecen a los diferentes niveles de jerarquía en las organizaciones, por ejemplo CMM, que involucra a PSP y TSP para la mejora y adiestramiento de los procesos desarrollados en la empresa a nivel de cada individuo, y a nivel de equipos de trabajos, respectivamente. La calidad del software se puede definir como el "Grado con el cual el cliente o usuario percibe que el software satisface sus expectativas".

La garantía de la calidad del software es una "actividad de protección" que se aplica a cada paso del proceso de software.

Para llevar a cabo adecuadamente una garantía de calidad del software, se deben recopilar, evaluar y distribuir todos los datos relacionados con el proceso de ingeniería del software. La capacidad de garantizar la calidad es la medida de madurez de la disciplina de ingeniería.

Con el objetivo de lograr la informatización de la sociedad cubana, en el país se está trabajando fehacientemente en el desarrollo de la Industria Cubana del Software, que permita proveer de sistemas informáticos, no sólo en beneficio de la sociedad cubana, sino también que se puedan exportar y que sean reconocidos internacionalmente, sin embargo las empresas que producen software hoy requieren aún de un sistema que cree un entorno organizado donde sean aplicados procedimientos de ingeniería de software que guíen el control de los procesos para desarrollar y mantener el software con una total calidad, para lograr evolucionar hacia una cultura de ingeniería de software y de administración de excelencia.

Dentro de esta esfera de la producción de software se enmarca la UCI, la cual fue creada con dos objetivos básicos, la informatización de la sociedad y la exportación. Países como Venezuela se ha beneficiado ya con los logros obtenidos, se puede plantear entonces sin temor a equivocarse que aún se están limando detalles en lo que a calidad de software se refiere.

La universidad de las Ciencias Informáticas está envuelta en proyectos de producción de software, por lo que es lógico deducir la necesidad de llevar un control de la calidad de los mismos. Existe un Grupo de Calidad encargado del control de la calidad de los productos de software de todas las facultades. Durante el desarrollo de los primeros proyectos, dicho grupo comenzó a realizar pruebas a los productos, y se determinaron muchos errores en las aplicaciones; es por eso que se hacía necesario la búsqueda de una solución a ese problema. Por tal motivo se decide incorporar al Grupo de Calidad un mayor número de personas involucradas en el proceso de control de la calidad. Por cada facultad se selecciona un Asesor de la Calidad, preferiblemente con conocimientos de Ingeniería de Software, y un grupo de 30 estudiantes para llevar a cabo las actividades de calidad. Estos grupos tributaban tanto a la producción de las facultades, como a nivel de universidad.

Inicialmente se contaba con un grupo de 30 estudiantes de toda la UCI, que trabajaban en todos los proyectos, después es que surgen los Asesores de Calidad por facultades. A partir de este momento se comienzan a desarrollar cursos de adiestramiento para los estudiantes y Asesores involucrados en la Calidad. Se establecieron además los Lineamientos mínimos de Calidad en la Universidad; así como un manual de cómo se deben comportar o dirigir los proyectos productivos. Estos lineamientos se muestran en el Anexo de este documento.

Actualmente en la Universidad el Grupo de Calidad lleva a cabo las actividades de Pruebas de Caja Negra basadas en la búsqueda del camino básico, para el control de la calidad de los productos de software. Se realizan pruebas de aceptación, integración y funcionalidad.

1.7. Introducción al Proceso Personal de Software (PSP).

El PSP es un proceso de auto mejoramiento diseñado para ayudar a controlar, administrar y mejorar la forma en que se trabaja individualmente. Está estructurado por formularios, guías y procedimientos para desarrollar software. Si es usado apropiadamente, brinda los datos históricos necesarios para trabajar mejor y lograr que los elementos rutinarios del trabajo sean más predecibles y eficientes. [3]

PSP mantiene los siguientes principios:

- La calidad de un sistema de software está determinada por la calidad de sus peores componentes.
- La calidad de un componente de software está gobernada por el individuo que lo desarrolla.
- La calidad de un componente de software está gobernada por la calidad de los procesos usados para desarrollarlo.
- La clave para la calidad son las habilidades individuales del desarrollador, compromiso y disciplina personal de proceso.
- Como un profesional del software, cada persona es responsable de su proceso personal.
- El ingeniero informático debe medir, monitorear y analizar su trabajo.
- El ingeniero informático debe aprender de sus variaciones de desempeño.
- El ingeniero informático debe incorporar lecciones aprendidas en sus prácticas personales.

Una organización disciplinada de ingeniería de software tiene bien definido sus prácticas. Sus profesionales emplean dichas prácticas, monitorean y se esfuerzan para mejorar su desempeño personal y mantienen ellos mismos una responsabilidad por la calidad de los productos que producen. Lo más importante es que ellos poseen los datos y confianza en sí mismos, necesarias para resistir demandas de acuerdos irrazonables.

El único propósito de PSP es ayudar a mejorar las habilidades de ingeniería de software. El mismo constituye una herramienta poderosa que puede ser empleada de muchas maneras. Por ejemplo, lo ayudará a administrar su trabajo, evaluar sus talentos y formar sus habilidades. Contribuye a la realización de mejores planes, para monitorear el desempeño de forma precisa y medir la calidad de los productos. Independientemente de que se diseñen programas, se desarrolle requerimientos, se escriba la documentación o se de mantenimiento al software existente, el PSP ayuda a hacer mejor el trabajo.

En lugar de usar un enfoque para cada trabajo, es necesario un arreglo de herramientas y métodos así como las habilidades expertas para usarlas apropiadamente. El PSP brinda los datos y técnicas de análisis necesarios para determinar que tecnologías y métodos trabajan mejor para cada integrante de un equipo de desarrollo.

El PSP también brinda un marco de trabajo para comprender por qué se cometen errores y cómo es mejor encontrarlos, arreglarlos y prevenirlos. Es posible determinar la calidad de las revisiones, los tipos de defectos que comúnmente los ingenieros informáticos obvian y los métodos de calidad que son más efectivos para cada integrante del equipo de desarrollo. Después de conocer PSP serán capaces de decidir qué métodos emplear y cuándo usarlos. También conocerá cómo definir, medir y analizar su propio proceso. Entonces, a medida que se obtenga experiencia, se reforzarán sus procesos para tomar ventaja de cualquier nueva herramienta y métodos de desarrollo. [3]

El PSP no es la respuesta mágica a todos los problemas de ingeniería de software, pero ayuda a identificar dónde y cómo puede mejorar el desarrollo de software. Sin embargo, los ingenieros informáticos mismo deben hacer las mejoras.

1.8. Niveles del PSP.

Los siete niveles de procesos empleados para introducir PSP se muestran a continuación. Cada nivel se conforma del nivel anterior mediante la incorporación de algunos pasos de procesos para el mismo. Esto minimiza el impacto de los cambios de procesos en el ingeniero, quien necesita solamente adaptar las nuevas técnicas en una línea base de prácticas existentes. Está formado por diferentes componentes entre los que se encuentra:

Guiones: Documentan el Proceso. Contienen los criterios de entrada de cada etapa del proceso, las etapas/pasos y los criterios de salida. Su propósito es guiar al ingeniero para usar el proceso.

Mediciones: Miden el proceso y el producto. Proveen la visibilidad de cómo está trabajando el proceso y del estado actual del trabajo.

Formularios: Proveen un medio consistente y conveniente para recolectar y almacenar datos.

Estándares: Proveen definiciones consistentes que guían el trabajo y la recolección de datos.

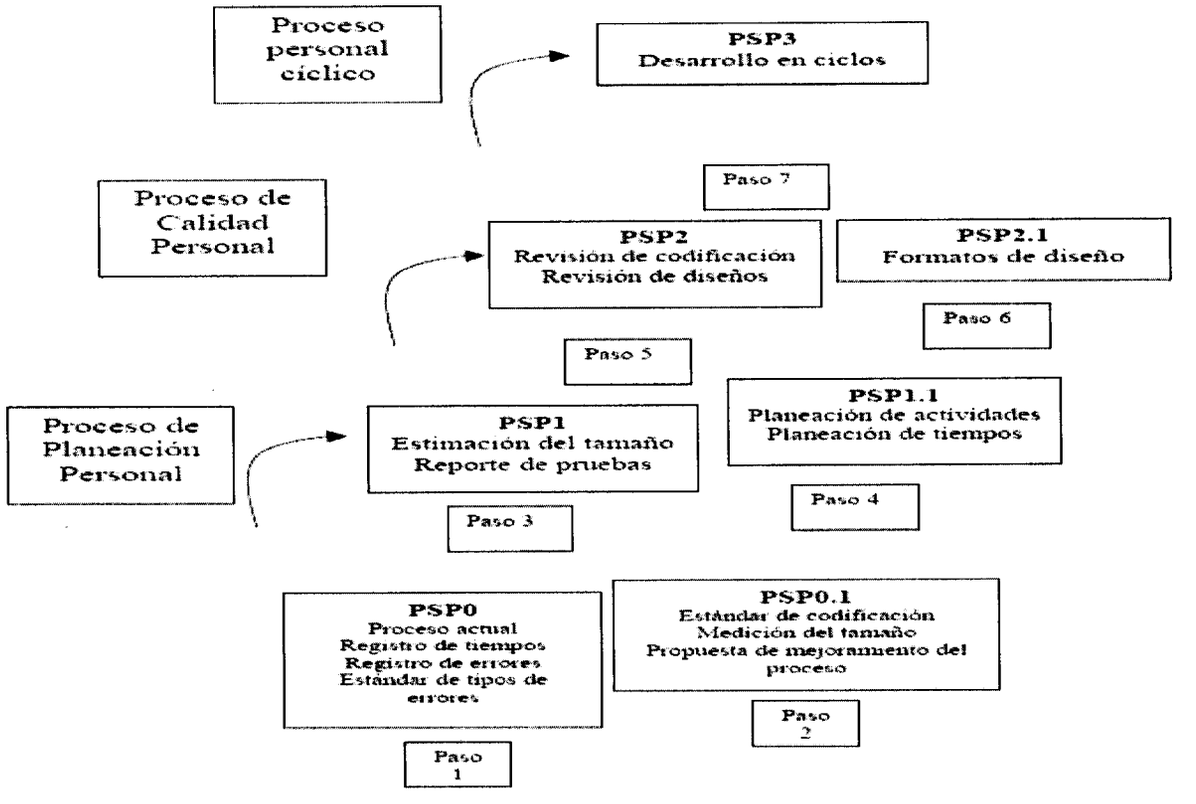

Figura 2: Los niveles de procesos de PSP. [14]

1.8.1 La Línea Base del Proceso Personal – PSP0 y PSP0.1

La Línea Base del Proceso Personal (PSP0 y PSP0.1) brinda una introducción al PSP y establece una base inicial de tamaños históricos, tiempo y datos de defecto. Los ingenieros escriben programas en este nivel. Está permitido que empleen sus métodos actuales pero dentro del marco de trabajo de los 6 pasos de la línea base de procesos que se muestra en la siguiente tabla: [7]

Paso	Fase	Descripción
1	Planificación	Se planifica el trabajo y se documenta el plan.

2	Diseño	Se diseña el programa.
3	Codificación	Se implementa el diseño.
4	Compilación	Se compila el programa, se reparan y registran los defectos encontrados.
5	Prueba	Se prueba el programa, se reparan y registran los defectos encontrados.
6	Postmortem	Se registra el tiempo actual, defectos y tamaño de datos en el plan.

Tabla 1: Línea Base del Proceso Personal (PSP0 y PSP0.1).

PSP0 introduce procesos básicos de medición y planificación. El tiempo de desarrollo, defectos y el tamaño de los programas son medidos y registrados en formularios establecidos. Un formulario simple de resumen del plan es usado para documentar los resultados planificados y actuales. Un formulario para registrar propósitos de mejora de procesos (PIP) es también introducido (PSP0.1). Dicho formulario permite a los ingenieros contar con una vía conveniente para registrar los problemas del proceso y las soluciones propuestas. [7]

El principal objetivo de PSP0, el proceso de línea base, es proporcionar un marco de trabajo para escribir su primer programa PSP y para obtener datos en su trabajo. Los datos de los primeros programas PSP brindan una línea base comparativa para determinar el impacto de los métodos PSP en su trabajo.

El proceso PSP0 proporciona los siguientes beneficios:

- Una estructura conveniente para ejecutar tareas a pequeña escala.
- Un marco de trabajo para la medición de esas tareas.
- Los fundamentos para la mejora de proceso.

Una estructura de trabajo conveniente

Con un proceso definido como PSP0 es posible disminuir el tiempo empleado para la planificación de tareas, debido a que ya se ha determinado cómo hacer las tareas cuando el proceso fue desarrollado. Cuando se ejecutan tareas similares muchas veces, es más eficiente desarrollar y documentar el proceso una sola vez, en lugar de parar para inventar un nuevo proceso cada vez que se inicie un trabajo. [7]

Un marco de trabajo de medición

Un proceso definido establece las medidas del proceso. Esto permite adquirir datos sobre el tiempo que se emplea en cada tarea, los tamaños de los productos que se elaboran y el número de defectos que son insertados y eliminados en cada paso del proceso. Esos datos ayudan a analizar el proceso, para comprender sus fortalezas y debilidades y mejorarlo. Un proceso definido también brinda mediciones con un significado explícito. [7]

Con la línea base del proceso PSP0 algunas de las entradas y criterios de salida no son muy precisos. Esto es porque PSP0 es un proceso simple.

Una fundación para la mejora

Si no se conoce lo que se está haciendo, es difícil mejorar la forma de hacerlo. La mayoría de los profesionales del software defenderían que ellos conocen lo que hacen cuando desarrollan software. Sin embargo, raramente pueden describir sus acciones en detalles. Cuando se sigue un proceso definido, se tiene mayor conocimiento del trabajo que se hace. Es posible observar el comportamiento propio y a menudo se puede observar cómo descomponer un gran paso en elementos más pequeños. PSP0 es un primer paso en la construcción de esta idea más refinada. [7]

Flujo de trabajo cíclico de PSP0

En ocasiones grandes programas o aquellos que no son bien comprendidos pueden requerir un enfoque iterativo. En el siguiente ejemplo el diseño es completado en un solo paso. Dos módulos son identificados durante el diseño, A y B. Entonces cada módulo es separadamente codificado, compilado y probado. [7]

1.8.2 La Gestión de Proyectos Personal – PSP1 y PSP1.1

PSP1 y PSP1.1 se enfocan en técnicas personales de gestión de proyectos, introduciendo tamaño y estimación de esfuerzo, planificación de cronograma y métodos de seguimiento de cronogramas. Los estimados de tamaño y esfuerzo son realizados mediante el método PROBE (PRoxy-Based Estimating). Con este método los ingenieros usan el tamaño relativo de un proxy para hacer sus

estimados iníciales, entonces usan datos históricos para convertir el tamaño relativo del proxy a Líneas de Código (LOC). Ejemplos de proxy para la estimación del tamaño del programa son: objetos, funciones y procedimientos. Otros ejemplos incluyen objetos de pantallas, guiones, reportes y documentos impresos. [7]

PSP usa el método de valor ganado para la planificación de cronogramas y el monitoreo. Dicho método es una técnica estándar de gestión que asigna un valor planificado a cada tarea en un proyecto. Dicho valor está basado en el porciento del esfuerzo total planificado para el proyecto que dicha tarea tomará. A medida que las tareas se completen, el valor planificado de las tareas se convierte en valor ganado para el proyecto. El valor ganado del proyecto se convierte entonces en un indicador del porciento del trabajo completado. Cuando se monitorea semana a semana, el valor ganado del proyecto puede ser comparado con sus valores planificados para determinar el estado, para estimar tasas de progreso y para proyectar la fecha de culminación del proyecto. [7]

Uno de los escalones de la estructura incremental de PSP es específicamente PSP1, el cual introduce la Estimación de Tamaño y el Reporte de Prueba. El objetivo de PSP1 es entablar un procedimiento ordenado y repetido para la estimación de tamaño de software y como los demás niveles de PSP viene acompañado por un conjunto de guiones y formularios.

Los objetivos de PSP 1.1 tienen bastante similitud con los vistos en PSP 1, sin embargo se introducen otros métodos de planeación y los formatos que se han venido utilizando sufren de pequeñas modificaciones. [7]

En el PSP 1.1 se introducen dos nuevas formas de planeación, el plan de tareas y el plan de cronograma. El primer tipo se basa en la actividad a desarrollar, como escribir un programa o un reporte. El segundo tipo está basado en un período determinado de tiempo, como ejemplo se puede tomar cualquier segmento de un calendario (días, semanas, meses o años). Estos planes proporcionan un seguimiento del progreso mientras se está trabajando Ambos tipos dependen uno del otro para que se lleven a cabo.

En PSP 1.1 el primer paso para hacer una planeación de producto es tener una definición clara del producto. Es necesario tomar en cuenta tres puntos importantes:

- El tamaño y las características importantes del producto.

- Un estimado del tiempo requerido para realizar el proyecto.
- Un calendario del proyecto.

Mientras más complejos sean los productos, éstos requerirán de una planeación más sofisticada. Los planes individuales de producción colaboran a cumplir con las fechas y tareas independientemente, con lo que puede revisar los compromisos adquiridos constantemente. [7]
Los ingenieros deben utilizar planes de producción para saber el estado en el que el proyecto se encuentra, si el proyecto va cumpliendo con el calendario, entonces los programadores puntuales pueden prestar su ayuda a los demás integrantes del equipo. Esta es la manera en la que pueden organizar su tiempo y evitar crisis de último minuto, derivando así, en productos de mejor calidad.

Asimismo se requieren de muchos tipos de información tales como acuerdos o asignaciones de responsabilidades, planes de apoyo, especificaciones del producto o del proceso, dependencias de otros grupos o pruebas especiales. Es importante realizar un plan que sea apropiado a la magnitud y complejidad del trabajo que se realizará. Por medio de la comparación de datos en proyectos pasados, se podrá predecir el tiempo aproximado que llevará realizar el presente proyecto.

PSP 1.1 hace notar que para realizar planeaciones será necesario definir los siguientes términos: [7]

- Producto.- Es algo que se produce junto con un colaborador, proveedor o un cliente.
- Proyecto.- Un proyecto produce siempre un producto. Es algo que se planea.
- Tarea.- Es un elemento definido de un trabajo.
- Proceso.- Define la manera de realizar proyectos.
- Planes.- Describe la forma de cómo un proyecto específico se debe realizar, ¿Cómo?- ¿Cuándo?- ¿A qué precio?
- Trabajo.- Es algo que se lleva a cabo, sea un proyecto o una tarea.

1.8.3 La Gestión Personal de la Calidad - PSP2 y PSP2.1

PSP2 y PSP2.1 incorporan métodos de gestión de calidad para el PSP: diseño personal y revisiones de código, una notación para el diseño, plantillas de diseño, técnicas de verificación del diseño y medidas para la administración de procesos y calidad de producto. [7]

La meta de la gestión de la calidad en PSP es encontrar y remover todos los defectos antes de la primera compilación. La medida asociada con esta meta es el rendimiento. El rendimiento se define como el porciento de los defectos insertados antes de compilar que fueron eliminados después de compilar. Un rendimiento de un 100% ocurre cuando todos los defectos insertados antes de compilar son eliminados después de compilar. [7]

Dos nuevos pasos del proceso, la revisión del diseño y la revisión del código, están incluidas en PSP2 para ayudar a los ingenieros a alcanzar un 100% de rendimiento. Las mismas constituyen revisiones personales conducidas por un ingeniero en su propio diseño o código. Son revisiones estructuradas y orientadas a los datos que se guían por listas personales de chequeo para la revisión derivadas de los datos históricos de defectos de los ingenieros.

Comenzando con PSP2, los ingenieros también comenzarán usando datos históricos para planificar la calidad y el control de calidad durante su desarrollo. La meta de los ingenieros es remover todos los defectos insertados antes de la primera compilación. Durante la planificación, estiman el número de defectos que serán insertados y eliminados en cada fase. Entonces se usa la correlación histórica entre tasas de revisión y el rendimiento para planificar revisiones efectivas y eficientes. Durante el desarrollo se controla calidad mediante el monitoreo de los defectos actuales insertados y eliminados contra el plan, así como mediante la comparación de las tasas actuales de revisión para establecer límites. [7]

Las revisiones son muy efectivas para la eliminación de la mayoría de los defectos encontrados en la compilación y muchos de los defectos encontrados en las pruebas. Pero para sustancialmente reducir los defectos de prueba, se necesitan mejores diseños de calidad. PSP2.1 cubre esta necesidad mediante la incorporación de una notación de diseño, cuatro plantillas de diseño y métodos de verificación de diseño para el PSP. El propósito es no introducir un nuevo método de diseño, pero para asegurar que el diseñador examine y documente el diseño desde diferentes perspectivas. Esto mejora el proceso de diseño y hace más efectiva la verificación y la revisión del diseño. Las plantillas de diseño en PSP brindan 4 perspectivas en el diseño: una especificación operacional, una especificación funcional, una especificación de estado y una especificación lógica.

En el nivel 2 de PSP se introduce la gestión de la calidad como uno de los principales aspecto que debe estar presente en cada etapa del desarrollo de un proyecto.

La definición de calidad está centrada en las necesidades de los usuarios y su conformidad con los requerimientos. Es por esto que la idea central, cuando se construye un software, debe partir de: ¿quiénes son los usuarios?, ¿qué es importante para ellos?, ¿cómo se relacionan sus prioridades con respecto a la forma de construir y agrupar soporte al software?

Si un producto tiene muchos defectos que no garantizan una ejecución consistente, el usuario no apreciará el resto de las cualidades o atributos que este pueda poseer. Esto no significa que los defectos son siempre la máxima prioridad pero si que son una parte muy importante en el desarrollo: si el mínimo nivel de calidad no se logra, más nada es importante. [7]

Por la importancia de un software sin defectos, las organizaciones consagran la mayor parte de su tiempo a encontrarlos y corregirlos y por el hecho de que la magnitud de este proceso de corrección con frecuencia es subvalorada, la mayoría de los proyectos se preocupan por la corrección del defecto e ignoran otras preocupaciones importantes de los usuarios. Cuando un equipo de proyecto está inmerso en la tarea de arreglar los defectos en la fase de pruebas, generalmente ya está escaso de tiempo y las presiones para la entrega son tan intensas que se olvidan otras preocupaciones. Sin embargo, la corrección de estos defectos críticos solo garantiza que el producto alcance el mínimo nivel de calidad deseado. En ese momento: ¿Qué se ha hecho para hacer el producto usable e instalable? ¿Qué hay de la compatibilidad, ejecución, seguridad? ¿Alguien ha revisado la documentación y si se ha logrado un diseño conveniente para una futura mejora del producto? Por el excesivo tiempo de corrección de los defectos no se ha dedicado ni siquiera un mínimo de tiempo a pensar en los problemas que últimamente son más importantes para los usuarios, sin embargo, con pocas pruebas de defectos, los proyectos pueden dirigirse a los aspectos de calidad que el usuario siente que son más importantes. [7]

Los defectos no son la prioridad máxima, pero un manejo adecuado de los mismos es esencial para el manejo de costo, planificación y otros aspectos de la calidad de un producto. Los defectos son el resultado de errores individuales, por tanto para manejarlos se debe manejar el comportamiento personal. Los ingenieros informáticos son la fuente de los errores en los productos por tanto son los únicos que pueden prevenirlos y los más indicados para encontrarlos y corregirlos.

PSP 2 tiene como objetivos introducir revisiones de diseño, de código y métodos para evaluar y mejorar la calidad de las revisiones individuales. Aunque la detección y corrección de defectos es un punto de crítica importancia, esto tiene una estrategia defensiva inherente. Para los niveles de calidad

requeridos en la actualidad, se tienen que identificar las causas de los defectos, definir pasos para prevenirlos y así se ganará en tiempo y en productividad. [7]
EL proceso PSP toma ventaja del hecho de que los errores de las personas son predecibles. Esto no quiere decir que sean incompetentes, sino que son humanos y como humanos se tiende a repetir los errores. Los principios de los procesos de revisiones personales son los siguientes:

1. Revise personalmente todo el trabajo propio antes de moverse a la próxima fase de desarrollo.
2. Esfuércese para corregir todos los defectos antes de darle el producto a alguien más.
3. Use una lista de chequeo personal y siga un proceso de revisión estructurado.
4. Siga las prácticas probadas de revisión: revise en pequeños incrementos, haga revisiones en papel y hágalo cuando está fresco y descansado.
5. Mida el tiempo de revisión, el tamaño de los productos revisados y el número y tipos de defectos que encontró.
6. Use estos datos para mejorar el proceso personal de revisión.
7. Diseñe e implemente sus productos de manera que sean fáciles de revisar.
8. Revise sus datos para identificar formas de prevenir los defectos.

1.8.4 Proceso Personal Cíclico – PSP3

Desarrollo cíclico.

Se le llama desarrollo cíclico cuando se desarrolla una y otra vez. Desarrollos que se repiten periódicamente. Este desarrollo en ciclo actualmente es muy común verlo ya que con el desarrollo de las veces anteriores sirve de base para los desarrollos posteriores. [9]

¿Qué es PSP3?

Con PSP3 finalmente el último nivel de PSP es alcanzado. Con este nuevo nivel se introduce una nueva fase, la fase de realizar el proceso personal creado de una manera cíclica y uniforme. Esto quiere decir que a estas alturas del proceso, el programador tiene una manera de programar única y bien definida, es la firma que cada programador debe de poseer.
Obviamente el proceso personal que el programador crea, es un proceso eficaz y aplicable a cada programa que quiera desarrollar. Este nivel ayuda al desarrollador a realizar programas más largos en

poco tiempo y con menos errores. Es un ejemplo del proceso personal a larga escala porque puede ampliar los métodos del PSP a proyectos mayores, muy adecuado para programas con mas de 1000 LOC. [9]

El Proceso Personal Cíclico, PSP3, centra las necesidades para escalar eficientemente el PSP para proyectos de mayor tamaño sin sacrificar calidad o productividad. En la clase los ingenieros aprenden que su productividad es mayor entre un rango mínimo y máximo de tamaño. Por debajo de este rango la productividad declina debido a los costos fijos superiores. Por encima de este rango, la productividad declina porque el límite de escalabilidad del proceso ha sido alcanzado. PSP3 localiza dicho límite de escalabilidad mediante la introducción de una estrategia cíclica de desarrollo donde grandes programas son descompuestos en partes para el desarrollo y entonces se integran. Esta estrategia asegura que los ingenieros están trabajando a su máxima productividad y los niveles de calidad del producto aumentan, de forma incremental, no exponencial, para grandes proyectos. [9]

Para soportar este enfoque de desarrollo PSP3 introduce el diseño de alto nivel, la revisión de diseño de alto nivel, la planificación del ciclo y ciclos de desarrollo basados en los procesos PSP2.1. Dos nuevos formularios son también introducidos: un resumen del ciclo para resumir tamaño, tiempo de desarrollo y defectos para cada ciclo; y un registro de seguimiento de aspectos para documentar aspectos que pueden afectar ciclos futuros o ciclos completados. Usando PSP3, los ingenieros descomponen su proyecto en una serie de ciclos PSP2.1, entonces integran y prueban la salida de cada ciclo. Debido a que los programas que se producen con PSP2.1 son de alta calidad, los costos de integración y prueba son minimizados. [9]

Estrategia que se sigue con PSP3.

En la tendencia cíclica de PSP3, el programa se divide en segmentos que se solucionan con PSP2.1. El producto de un ciclo sirve de base para el próximo ciclo.

Los objetivos de PSP 3 son los mismos que se detallan en PSP2.1, las únicas adiciones que se hacen es que el programador debe de ser capaz de desarrollar programas de hasta miles de LOC y para esto se introducen nuevos guiones, formularios y plantillas que son los procesos finales que el programador debe de dominar.
Este proceso comienza con los requerimientos, luego se produce un diseño conceptual del programa entero donde se estima el tamaño y se planifica el desarrollo.

A partir de este diseño de alto nivel, se divide el producto y se divisa la estrategia cíclica. Cada ciclo se ejecuta con PSP2.1: codificar, compilar, revisar y probar. Depende de cada programador realizar la división de los módulos y la correcta integración de éstos al producto final. Después de cada ciclo hay que reevaluar el plan, elaborar un reporte de ciclo indicando problemas y desviaciones. [9]

1.9. Medidas para contabilizar el tamaño de un software.

Existen varias formas de medir el tamaño del software, la cual debe ser útil para la estimación del tiempo de desarrollo por esto esta medición debe ser precisa, específica y automáticamente contable.

Precisa: da el valor exacto para el tamaño de un producto.
Especifica: está basada en las propiedades definidas de un producto.
Contable: porque para grandes programas es impracticable el conteo manual.

La teoría de Humphrey de basa en la estimación por criterio de expertos, o sea, donde se parte de una experiencia en el desarrollo de un determinado proceso y lo utilizo para realizar una estimación de un nuevo proceso. [7]

Las medidas de tamaño se introducen para ayudar a estimar el tamaño del producto. En muchas ocasiones se utiliza como medidas de tamaño las líneas de código (LOC) de un programa pero en la práctica existen otras unidades que pueden ser utilizadas. A continuación se presentan dos ejemplos clásicos para contabilizar el tamaño de un software: [7]

Estableciendo el conteo estándar de las BD

Producir una BD contar campo, tablas.
Producir programas para usar una BD contar sentencias, elementos GUI, tablas o líneas de código
Usar una herramienta GUI contar botones, cajas de dialogo, labels, etc.
Estableciendo un conteo de estándar de líneas de código

Las líneas de códigos (LOC) es una medida útil porque se correlaciona bien con esfuerzo de desarrollo.
En PSP se usan otras medidas también: elementos de BD, páginas de documentos, formas.

El primer paso para definir LOC como conteo estándar es establecer una estrategia de conteo:

- Contar las líneas.
- Contar palabras reservadas
- Contar las funciones.

Contabilización del tamaño

Cuando se trabaja con un equipo de trabajo que produce múltiples versiones, para seguir todos los cambios y adiciones hechas al programa y recolectar los datos necesarios para la estimación se debe usar un sistema de contabilización del tamaño. [7]

Usando datos del tamaño.

Las formas principales de usar los datos del tamaño son en la planificación, la administración de la calidad y el análisis de procesos.

Usando medidas del tamaño para la planificación

Para la planificación si se tiene una medida de tamaño definida y datos históricos de tamaño y tiempo se puede estimar el tamaño de un nuevo producto.

El esfuerzo para adicionar o modificar código será a grandes rasgos el mismo, mientras que el esfuerzo requerido para eliminar una línea o incluir una previamente desarrollada será mucho menor. Para otro tipo de trabajo se tendrán que hacer diferentes selecciones.

En los trabajos de mantenimiento, por ejemplo el esfuerzo requerido para eliminar una línea puede tomar tanto o mas tiempo del requerido para adicionar o modificar.

Determinando la calidad del programa

Dividiendo el número de defectos en una fase por el tamaño del programa da la densidad de defectos en esa fase Con datos de programas similares, se puede usar estimaciones de la densidad de defecto y tamaño para estimar las pruebas mantenimiento y costo de servicios. Para el cálculo de la densidad de defecto contar solo los códigos adicionados y modificados durante el desarrollo. [7]

Para la calidad de los programas, la densidad de defecto es generalmente medida por LOC.

Para la calidad de los programas, la densidad de defectos es generalmente medida en defectos por 1000 LOC o KLOC. Para productos terminados la medida más conveniente es 1 000 000 LOC o MLOC. Los trabajos de BD, documentos y otros productos son contados por 100 o 1000 elementos.

Calculando productividad

La productividad es medida generalmente como las horas de labor requeridas para hacer una unidad de trabajo. Si se estima el tamaño de un trabajo nuevo y entonces se usa un factor global de productividad para determinar el total de horas requeridas se obtendrá generalmente un resultado erróneo. La productividad se calcula dividiendo el tamaño del producto hecho por las horas empleadas en producirlo. [7]

Contadores de Tamaño.

Contar el tamaño de los programas, aunque sean pequeños consume tiempo y se puede incurrir en errores. Para programas grandes no es práctico, por lo que se requieren esencialmente contadores de tamaño automatizados.

Los contadores pueden destinarse para contar el número de prácticamente cada elemento del producto, así como la definición del elemento sea precisa y específica. Por ejemplo: los contadores LOC pueden contar líneas físicas o lógicas. También contar automáticamente elementos de bases de datos puede ser un poco mas complejo; si un modelo de objeto es accesible, puedes contar seleccionando conjuntos de elementos de la base de datos. [7]

Contadores físicos de LOC:

Es el tipo más simple de contadores de tamaños. Cuenta todas las líneas del programa excepto los comentarios y líneas en blanco. Una línea de texto que tiene código y comentarios también es contada. [7]

Contadores Lógicos:

Los contadores lógicos funcionan parecidos a los contadores físicos, excepto que el paso de contar las líneas es más complejo. Aunque algunos puedan pensar que contar el principio y final de las declaraciones no es una buena idea, es una cuestión de preferencias personal. Se pudiera demostrar, sin embargo, que el tiempo requerido para desarrollar un programa tiene mas alta correlación con LOC cuando los pares comienzo-fin fueron contados que cuando no, esa seria una buena razón para usarlos. Esto seria también la manera apropiada para resolver cualquier pregunta sobre conteo del tamaño. [7]

Actualmente no existe ninguna evidencia que soporte ningún método de conteo sobre ningún otro. Con buenos datos, sin embargo, un estudio de las distintas alternativas, mostraría rápidamente, si alguno, será el mejor para su trabajo. Podría demostrarse que diferentes acercamientos producen aproximadamente el mismo resultado. En ese caso la selección es arbitraria. El hecho que sea arbitrario, sin embargo, no significa que cada cual deba hacerlo diferente. Si todos los proyectos de una organización usan el mismo conteo estándar, ellos pudieran desarrollar un mismo contador automático con calidad para todos. Tendrían también un largo volumen de datos para usar en las planificaciones y los análisis de calidad. [7]

Contar los elementos del programa:

Siempre es deseable medir el tamaño de varias partes de programas largos. Por ejemplo, si deseas rehusar algunas clases o procedimientos nuevamente, te gustaría medir el tamaño de cada uno. De manera similar los contadores ayudan en la estimación.

Para mantener contadores separados para cada clase o procedimiento, debes determinar cuando comienza y termina cada uno. Debido a que la vía de hacerlo depende del lenguaje de programación, no existe una guía general. Con un poco de estudios puedes llegar a determinar los indicadores de principio y final de los métodos, procedimientos, clases u otros elementos del lenguaje que uses. Puedes usar la definición para escanear los textos del programa e iniciar y parar el contador de tamaño en los puntos apropiados. [7]

Estándar de Codificación:

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. [7]
Unas de las ventajas que brinda el uso de estándares de codificaciones es que reduce la cantidad de errores, garantiza la obtención de un código comprensible, garantiza buena comunicación entre los integrantes del equipo además de facilitar el mantenimiento, rehúso y revisión. [7]

Otras mediciones de tamaño.

La medición por líneas de código (LOC) es solo una opción para la medición de tamaño. Como se pudo observar anteriormente, dependiendo del tipo de trabajo que se hace, elementos de bases de datos, correcciones de errores, o páginas pueden ser más apropiadas. Otra medición útil y mucho más general es conocida como puntos de función. La medición de puntos por función fue desarrollada a finales de los 70 y ha llegado a ser ampliamente utilizada. Actualmente existe un grupo de estándares, el grupo internacional de usuarios de puntos de función que utiliza puntos de función, además de la disponibilidad de una cantidad considerable de literatura. [7]

Un punto a recordar es que no hay una forma de medición que sea la mejor para cada situación. Si prefiere LOC, elementos de bases de datos, o puntos de función, la mejor estrategia es obtener datos en la estimación, el tamaño actual y las mediciones de tiempo. Realizar un análisis de estos datos para observar cuales mediciones y métodos, se ajustan más en cada situación. Independientemente de lo que una persona pueda decir, si sus datos indican que un tipo de medición es mejor para una persona, probablemente este en lo cierto.

1.10. Métodos de estimación de software.

Las estimaciones son hechas, comparando el trabajo planeado con trabajos desarrollados anteriormente. Descomponiendo los productos en piezas más pequeñas y comparando cada parte con datos sobre las partes similares de productos anteriores, se puede determinar el tamaño del nuevo producto. Esta estrategia funciona bien para estimar casi cualquier tipo de trabajo de desarrollo. Sin embargo, requiere datos sobre los productos que se han desarrollado antes y el trabajo requerido para

desarrollarlos. También se necesita un método que utilice datos históricos para hacer las estimaciones. [7]

Diseño Conceptual.

Según PSP, se debe primero estimar tamaño y luego hacer la estimación referente al tiempo de desarrollo. Para que la estimación sea certera debe estar basada en un diseño conceptual inicial donde este refleje el modo de planear y construir el producto. El diseño conceptual define un diseño preliminar con el nombre de cada una de las partes que componente el producto y sus funciones. Ahora, no se debe producir un diseño conceptual completo durante la planeación, solamente deben definirse las partes y sus funciones que integraran el producto. [7]

Para lograr estimaciones confiables tienen que refinar el diseño conceptual a un nivel donde se conozcan las partes que se van a construir. Luego, se chequea en los datos históricos de proyectos desarrollados anteriormente si existen partes similares a las del diseño. Si las partes no coinciden se debe entonces refinar el diseño conceptual a un nivel apropiado.

El Método de estimación basado en Proxy (PROBE)

El método PROBE se desarrolló con estos fines, determinar el valor estimado del tamaño total de líneas de código (LOC) de un programa y el tiempo que tomó para ello. Esto, se logra por el método de regresión lineal, su fórmula matemática se describe a continuación: [7]

Tiempo de Desarrollo = $\beta 0 + \beta 1 * \text{Tamaño Estimado (E)}$

Estos valores de regresión β son calculados específicamente por el método PROBE y que además se tratan posteriormente en este capitulo. Este método ha sido automatizado en la herramienta Process DashBoard.

Valor Ganado (EV).

Cuando se planea un proyecto de muchas tareas es necesario seguir y reportar el progreso, fundamentalmente cuando las tareas son completadas en orden diferente al que se planeó originalmente. Si solo tuvieran unas pocas tareas o estas se desarrollaran en iguales intervalos de

tiempo, esto no sería difícil, sin embargo los proyectos generalmente tienen muchas tareas de diferentes tipos y tamaños. [7]

A cada tarea se le debe asignar un tiempo (valor planificado o PV) y cuando la tarea termina el valor planificado se convierte en valor ganado. Este valor se agregará al valor ganado de todo el proyecto, si una tarea es muy grande debe dividirse en subtareas para poder medir el progreso. [7]

La medida del valor ganado provee una vía conveniente para dirigir el seguimiento del progreso y reportar los problemas. Siempre que se complete una tarea se obtendrá una cantidad de EV, para juzgar el progreso contra el plan se debe simplemente comparar el EV de cada semana con el valor planeado para esa semana, si el EV acumulado es igual o excede el PV acumulado entonces no habrá retraso.

1.11. Plantillas de Diseño de PSP.

PSP propone 4 plantillas para representar el diseño de un programa. Estas forman parte de los formularios de PSP 2.1. Las mismas son usadas para describir las propiedades esenciales y la estructura de los módulos de los programas son utilizadas para minimizar los duplicados. Cada elemento es guardado en un solo lugar y su localización es referenciada cuando se necesario. Esto ahora tiempo, reduce la probabilidad de error y proporciona referencias confiables. Un buen diseño debe tener el mínimo de redundancia. [7]

Aunque muchas de las propiedades de los sistemas de software no están cubiertas por estas plantillas, estas propiedades deben ser reflejadas en el diseño o el código de uno o más módulos del programa. Estas plantillas ayudan a asegurar que estas propiedades son implementadas completa y apropiadamente en el nivel del módulo. [7]

Plantilla de especificación operacional (OST)

La OST es una forma simplificada de usar casos que son usados para describir el comportamiento operacional de un programa. Ayuda a visualizar como el programa debe reaccionar bajo varios escenarios de uso. Cuando se enfrenta una decisión de diseño que involucra un actor, esto produce un escenario de ensayo para ver cómo cada acción puede aparecer par el actor. Esto puede ayudar a visualizar el comportamiento del programa para hacer mejores selecciones del diseño. [7]

Plantilla de especificación funcional (FST)

Esta plantilla proporciona una forma simple de documentar muchos de los materiales de los diagramas de clase de UML. El FST describe una parte, incluido los métodos, sus relaciones y sus restricciones. La parte puede ser una clase, un módulo de programa o incluso un programa grande o un sistema. La parte o nombre de la clase se lista en la parte de arriba, junto con la clase u otras partes de las cuales desciende directamente. Los atributos se listan a continuación, con sus declaraciones y descripciones. La tercera sección lista la parte de los métodos o elementos, con sus declaraciones, descripciones y lo que devuelve. [7]

Plantilla de Especificación de Estado (SST).

El comportamiento de estado de una máquina se define por sus actuales entradas y el estado del sistema. La plantilla Especificación de Estado proporciona una forma simple para describir de manera precisa el comportamiento del estado de una parte.

Plantilla de Especificación Lógica (LST).

Esta plantilla muestra cómo usar un lenguaje simple de programación del diseño para describir la lógica interna de una clase. PDL a veces es llamado como seudocódigo.

El LST proporciona una descripción en seudocódigo del programa. Su objetivo es explicar concisa y claramente qué debe hacer el programa y cómo.

Uso de las plantillas de PSP para el diseño.

EL principal intento de las plantillas de PSP para el diseño es guardar el diseño, no ayudar a producir el diseño. Las plantillas definen de manera precisa qué debe hacer el programa y cómo debe ser implementado. Las elecciones del método de diseño son particularmente importantes porque el diseño es un proceso intelectual y es difícil hacer un buen trabajo intelectual en un lenguaje o con métodos que no son fluidos. Por esto, por lo menos hasta que se hayan usado estas plantillas por un tiempo y se esté familiarizado con ellas se debe continuar produciendo diseños como se hayan hecho anteriormente pero guardándolos en las plantillas. Completando las plantillas se identificarán

generalmente problemas de diseño. Cuando se gane experiencia con las plantillas se verá que también estas ayudan a producir el diseño. [7]

Estas plantillas fueron hechas tan genéricas como fue posible de manera que puedan ser usadas con cualquier método de diseño. Si el método de diseño que se está usando también produce la información requerida por una o más plantillas, no es necesario gastar tiempo duplicando ese material. Antes de decidir omitir una o más de estas plantillas, sin embargo, se debe asegurar que el método de diseño usado captura toda la información requerida en los detalles especificados por las plantillas. [7]

1.12. Descripción del dominio del problema.

Durante la investigación fue necesario realizar un estudio profundo del funcionamiento de los proyectos del polo productivo de video y sonido digital en la facultad 9, con el objetivo de conocer el estado actual y los problemas existentes en los mismos. El polo productivo video y sonido digital está dividido en tres proyectos. En estos proyectos no se aplican ningún sistema de planificación para la gestión del software. En los proyectos existen problemas en la estimación de riesgos. Se trabaja sobre estimaciones ficticias, limitando en tiempo tareas que consumen más recursos humanos y técnicos de los que se contemplan en este tipo de planificación. Además del tiempo mismo que requiere el aprendizaje y preparación del personal en las herramientas con las que se trabajan, que no se calcula con valores reales, evidenciando la falta de un patrón o indicador para comparar con los valores estimados.

Después de realizar la entrevista en el polo productivo de video y sonido digital a un jefe de proyecto y a 3 integrantes de este polo plantearon:

Proyecto UCI TV se encarga de realizar prestaciones de servicios. Este proyecto no tiene un tiempo límite, las tareas son orientadas a cualquier persona teniendo en cuenta su carga de trabajo, no documentan ninguna tarea realizada. Realizan una planificación irreal. No tienen ninguna estrategia aplicada de PSP.

Proyecto de Monitoreo de Radio y TV se encarga de la documentación para la aplicación de un proyecto ya realizado en Venezuela.

Proyecto MENPET se encarga de la realización de un software sobre como el utilizado en UCI "Señal 3" para el ministerio de energía y petróleos de Venezuela en software libre, este registra las actividades y el tiempo de culminación pero sus integrantes no aplican PSP. Tienen poco conocimiento sobre el tema. Utilizan como herramienta el DotProject para tener un control de las actividades, así como la asignación de recursos materiales y humanos a las mismas. El proyecto tiene un formulario de gestión de riesgos que es utilizado en el proceso de mejora en el desarrollo de software. Se aplica pero sin tener conocimiento de PSP y sus ventajas en los proyectos.

Es preocupante el descontento general del equipo de desarrollo, luego de probar varias alternativas entre herramientas de diseño, construcción y prueba de software a utilizar, por falta de una buena planificación. Este problema fehaciente se evidencia también cuando se realizan los cortes de proyecto verificando gran parte de incumplimiento de tareas específicas, causando de esta manera principalmente el atraso general del proyecto. Tomando como referencia algunos proyectos de este polo se demuestra la necesidad de un proceso de control del trabajo individual y en equipo para aplicar entre los desarrolladores del mismo, así como de una selección más acertada de las herramientas y metodologías a utilizar para el desarrollo y puesta en práctica del software.

1.13. Conclusiones.

La calidad del software esta estrechamente relacionada con la buena aplicación del Proceso de Software Personal (PSP), ya que logrando una estrategia correcta se logra software con la calidad requerida y en el tiempo establecido. Muchos son los factores que influyen en el resultado final del producto, de ahí la importancia de un correcto, organizado y bien estructurado trabajo personal, constituyendo la primera base en la línea de la arquitectura del software.

Se puede señalar que los proyectos del polo productivo de video y sonido digital de la facultad 9 se encuentran en un estado básico en cuanto a calidad y uso del PSP se refiere, trayendo como consecuencias problemas en la aplicación de sistemas de planificación para la gestión del software, existiendo un descontento general por parte del equipo de desarrollo, causando el atraso del proyecto. Lo antes mencionado, encamina a una ardua investigación para darle solución a estos problemas.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN.

2.1. Introducción

A lo largo de todo el ciclo de vida del producto de software es necesaria la aplicación de PSP pasando por sus niveles los cuales ayudarán a un mejor desarrollo y entrega del producto final. Durante el desarrollo del Capítulo se planteará una solución a la problemática que se genera, al no aplicar PSP de forma eficiente en los proyectos del polo productivo de video y sonido digital de la facultad 9.

Debido a la situación que tiene la organización del trabajo y el cumplimiento del tiempo de entrega de los proyectos; es necesario llevar a cabo un estudio que dará como resultado una estrategia a aplicar, en los proyectos del polo productivo de video y sonido digital.
Precisamente el PSP, puede ser usado por futuros ingenieros de software como guía para un enfoque disciplinado y estructurado en el desarrollo de software.

2.2. Como se aplica el PSP en el desarrollo de software en Cuba

Es necesaria la informatización de la sociedad para estar a la altura del desarrollo actual. En Cuba se han trazado nuevas estrategias apostando por la informática como base para realizar cambios radicales en la economía, y elevar el nivel cultural del pueblo cubano. Este país tiene como meta lograr cambios en varias esferas de la sociedad, se ha comenzado a trabajar arduamente en la informatización de la población. En todos los institutos pedagógicos de la sociedad se están impartiendo clases de computación, la industria cubana ha tenido que informatizarse para lograr un mejor funcionamiento. Todos los municipios del país tienen varios Joven Club de Computación; los cuales tienen la tarea de llevar la computación hasta los lugares más recónditos.

Cuba ha comenzado a tener un desarrollo vertiginoso en la industria del software lo que ha traído consigo que se tenga en cuenta la calidad del software generados por esta industria. Se desarrollan software sin una planeación previa, saltándose las fases de análisis y planeación. Es necesario el estudio y aplicación de buenas técnicas de calidad de software. En el mercado mundial se requiere de productos de calidad y que su entrega sea en el tiempo acordado. En este país se están formando gran cantidad de ingenieros informáticos los cuales tienen la misión de ser los propulsores del desarrollo y calidad del software en Cuba.

En una industria de software como la cubana es necesaria la aplicación del PSP para tener ingenieros productivos y que trabajen de acuerdo a su productividad. Todo proyecto de software y hasta las tareas diarias deben registrarse para tener un control del ingeniero informático y lograr una productividad máxima.

2.3. Cómo se aplica el PSP en el desarrollo de software en la UCI.

La informática está llamada a convertirse en un campo boyante internacionalmente, en una de las ramas más productivas para el país. Para vencer esta meta es inevitablemente lograr el respaldo de un sólido sistema de educación superior donde es trascendental el lugar que ocupa la Universidad de las Ciencias Informáticas (UCI). Sobre el objetivo de este centro Fidel Castro Díaz-Balart planteó:

"El propósito fundamental es lograr un centro de excelencia para la formación masiva de profesionales de nivel superior. Ello debe alcanzarse con la ejecución de ambiciosos programas curriculares y de producción y con la aplicación de las más modernas tecnologías en la docencia." [10]

Durante muchos años en Cuba la computación fue un enigma para casi todos sus habitantes; después de la creación de la UCI con capacidad para diez mil estudiantes y con gran capacidad de generación de software. En la UCI casi todos sus alumnos y profesores están vinculados a proyectos lo que trae consigo la necesidad de aplicar PSP en casi todos sus proyectos. Cuando se quiere calidad hay que distribuir y organizar el trabajo; es necesario tener un control del tiempo que demora un hombre en realizar sus actividades para que se esfuerce según sus posibilidades. La UCI necesita desarrollar informáticos integrales que puedan ser útiles y productivos, con la aplicación del PSP logra en gran parte sus objetivos.

En la mayoría de las profesiones el trabajo competente requiere el uso de prácticas establecidas, planes y procedimientos que traen orden y eficiencia a cualquier trabajo y permite a los trabajadores concentrarse en producir productos de la más alta calidad.

En la UCI ya se han dado los primeros pasos para introducción paulatina de estas buenas prácticas para el desarrollo de software, pero uno de los aspectos que han restado vitalidad a la propuesta ha sido la resistencia al control de los datos necesarios, también en este trabajo se propone la utilización de la herramienta Process Dashboard, para apoyar esta tarea.

2.4. Importancia de la aplicación del PSP en los proyectos de video y sonido digital.

Cuando un proyecto comienza siempre debe tener en cuenta la capacidad de sus trabajadores. En los proyectos del polo productivo de video y sonido digital, principalmente en el proyecto UCI TV, no se aplica ningún método para controlar el rendimiento de sus integrantes El polo productivo no tiene registros de datos históricos del trabajo y las actividades son asignada según la carga de trabajo, de los integrantes de los equipos de desarrollo, esto les limita conocer la productividad de cada persona.

El 70% de los costos en el desarrollo de software se atribuye a costos personales, habilidades, experiencia y los hábitos de cada ingeniero determinan los resultados en el desarrollo de software. Esta relación del ingeniero con los resultados del proceso de desarrollo de software es la premisa en la que se basa el Proceso Software Personal. [9]

2.5. Principales riesgos que tiene la aplicación del PSP en los proyectos.

Todos los trabajos siempre están propensos a tener dificultades como falta de herramientas, problemas metodológicos y muchos más. La aplicación del PSP puede tener diversos riesgos los cuales pueden ser:

1. Los integrantes del equipo de desarrollo no apliquen correctamente el PSP por ser tedioso y pensar que obstruye el cumplimiento de otras tareas del proyecto.
2. Los jefes no controlen la aplicación del PSP por cada integrante de su equipo de trabajo.
3. No realizar un trabajo de mesa profundo para la aplicación del PSP antes de empezar a desarrollar las diferentes fases del proyecto.
4. No aplicar una estrategia de PSP de acuerdo a sus necesidades del proyecto.

2.6. Análisis de tesis y trabajos sobre PSP desarrolladas en la UCI.

Trabajo realizado en la UCI referente a las metodologías utilizadas para el aprendizaje del PSP en carrera del ingeniero informático: [11]

Este trabajo es presentado en la revista UCI se plantea la necesidad de la utilización de PSP por los estudiantes e ingenieros informáticos. La importancia de introducir en el primer año de la carrera el estudio del PSP y los avances que pueden lograr en un proyecto.

En la UCI se cuenta con un grupo de calidad central el cual asigna plantillas para controlar la calidad del desarrollo del proyecto. Esta plantilla se documenta el trabajo y se registran los riesgos a mitigar.

Las plantillas utilizadas por el polo productivo de video y sonido digital son:

- Plantillas de Requisitos.
- Plantillas de Arquitectura y Diseño.
- Plantillas de Implementación y Pruebas.
- Plantillas de Despliegue e Instalación.
- Plantillas de Gestión de Proyecto.
- Plantillas de Gestión de Configuración.

2.7. Herramientas de apoyo al PSP

Las investigaciones en este tema hasta el momento, se dividen en tres generaciones. En la tabla se muestran algunas de sus características [12].

La primera generación hacía uso del PSP como se describe originalmente por su creador en A Discipline for Software Engineering: los usuarios creaban e imprimían tablas o formularios que llenaban de forma manual con gran esfuerzo y cansancio por lo trabajoso del llenado de datos.

En la segunda generación se usaban herramientas automatizadas como Leap, PSP Studio o PSP Dashboard. Estas básicamente tienen las mismas prestaciones: cajas de diálogo donde el usuario registra los datos de esfuerzo, tamaño y defectos, aunque también se pueden mostrar análisis si el usuario lo desea. Esta segunda generación, sin dudas minimiza el trabajo del usuario en la recolección de datos así como en su análisis para la toma de decisiones.

Con el desarrollo de Hackystat en Mayo del 2001, el tema evoluciona hacia la 3ra generación con una herramienta que colecciona automáticamente las métricas mediante sensores adjuntos a las herramientas de desarrollo, los datos son enviados por los sensores al servidor y en dependencia de los análisis pueden ser enviados mensajes de alerta, por ejemplo, a los desarrolladores.

Características	1ra Generación (PSP manual)	2da Generación (Leap, PSP Studio y PSP Dashboard)	3ra generación (Hackystat)
Dificultad en la recopilación de datos	Alta	Media	Nula
Dificultad para el análisis	Alta	Baja	Nula
Cambios en las métricas	Simple	Software	Dependiente de la herramienta

Tabla 2: Evolución de herramienta de apoyo al PSP.

Como se planteó anteriormente, la herramienta recomendada por sus características es el Process Dashboard (tablero de instrumentos de proceso). Fue desarrollado originalmente en 1998 por la fuerza aérea de Estados Unidos, y ha continuado desarrollándose bajo modelo open-source. Está libremente disponible para la descarga bajo condiciones de la GNU Public License (GPL: Licencia Pública de GNU), siendo la última versión la 1.6.5 de diciembre de 2005.

El Process Dashboard (PDB) es un software libre de apoyo a PSP con módulos que incluyen los guiones y formularios que aparecen en el libro de Watts Humphrey "A discipline for Software Engineering", donde se describe el Proceso Personal de Software. El PDB fue desarrollado en Java, y necesita para correr en cualquier sistema de una Máquina Virtual Java. Corre en sistemas Windows, Linux, Unix, y Macintosh. Para acceder a los formularios y guiones se requiere disponer de algún navegador web.

Después de tener instalado el ambiente para correr programas Java, se ejecuta el fichero de instalación nombrado pdash-install-offline-1-6-5.jar y se siguen las indicaciones pertinentes. Lo más importante es ubicar los ficheros de instalación generalmente en C:\Program Files\ PDB y señalar el lugar donde serán guardados los datos históricos.

El PDB en un espacio muy pequeño el tablero de instrumentos da el acceso a una especie de cronómetro como el contador de tiempo para las actividades de la sincronización, un diálogo de la entrada del defecto para la información del defecto que captura, scripts y formas para seguir procesos definidos, un "checkbox práctico de la terminación" que permite una manera rápida y fácil de decir a la herramienta que una fase del desarrollo sea completa, menús que permiten la navegación con la jerarquía de la estructura de la interrupción del trabajo de proyecto, y un menú de configuración que permite el acceso a otras partes de la herramienta.

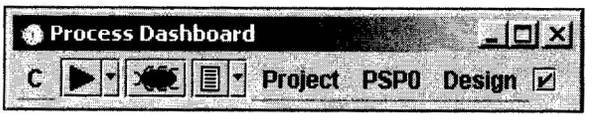

Figura 3: Menú de configuración del Process Dashboard (PDB)

De manera general el PDB es un software que resulta muy conveniente usar para fines docentes en la formación de profesionales con hábitos de disciplina personal y por el personal encargado de enfrentar el proceso de desarrollo de software en las empresas cubanas. Uno de los aspectos fundamentales es que es un software libre, y su uso estaría al margen de las políticas dictadas al respecto en Cuba. En este trabajo solo se describieron las funcionalidades que brinda el software fundamentalmente en el manejo y control, de tiempo y defectos, pero el software brinda otras funcionalidades muy importantes, tal es el caso de posibilitar realizar estimaciones de tiempo y tamaño basadas en datos históricos.

2.8. Solución propuesta para un proyecto del polo productivo de video y sonido digital.

El desarrollo de productos de software implica mucho más que escribir instrucciones de programación juntas y ejecutarlas en un ordenador. Requiere cumplir requisitos del cliente a un costo y planificación acordada [9]

El PSP muestra cómo producir de forma regular software de alta calidad. Utilizando el PSP se obtienen datos que muestran la efectividad del trabajo y se identifican los puntos fuertes y las debilidades, además se practican habilidades y métodos que ingenieros del software van a desarrollar durante muchos años de pruebas y errores.

El PSP enseña a ingenieros y futuros ingenieros, cómo administrar la calidad de sus productos y cómo hacer compromisos que ellos puedan cumplir. Puede ser empleado en muchas fases en el ciclo de desarrollo de programas pequeños, definición de requerimientos, documentación, pruebas y mantenimiento. [13]

El diseño de PSP se basa en los siguientes principios de planeación y de calidad: [3]

1. Cada ingeniero es esencialmente diferente; es decir, los ingenieros deben planear su trabajo y basar sus planes en sus propios datos personales.

2. Para mejorar constantemente su funcionamiento, los ingenieros deben utilizar personalmente procesos bien definidos y medidos.
3. Para desarrollar productos de calidad, los ingenieros deben sentirse personalmente comprometidos con la calidad de sus productos.
4. Para hacer un trabajo de ingeniería de software de la manera correcta, los ingenieros deben planear de la mejor manera su trabajo antes de comenzarlo y deben utilizar un proceso bien definido para realizar de la mejor manera la planeación del trabajo.
5. Para que los desarrolladores lleguen a entender su funcionamiento de manera personal, deben medir el tiempo que pasan en cada proceso, los defectos que inyectan y remueven de cada proyecto y finalmente medir los diferentes tamaños de los productos que llegan a producir.

Durante un exhaustivo análisis de las necesidades principales del proyecto MENPET, se traza la necesidad de utilizar PSP de acuerdo a las necesidades fundamentales del proyecto, ya que aplicar PSP completo resulta muy engorroso para el equipo de desarrollo; pues en los proyectos informáticos de la UCI, los integrantes de los equipos de desarrollo no intervienen en todas las fases del desarrollo de los mismos, los cuales están divididos por roles que intervienen en diferentes fases del proyecto. Esto trae consigo aplicar PSP teniendo en cuenta el rol que cumple cada integrante dentro del equipo de desarrollo.

Para una mejor aplicación del PSP en el proyecto informático se debe dividir el mismo en etapas, las que estarán divididas en fases que generarán artefactos y intervendrán roles. Estas etapas mejorarán el aprendizaje y compresión del PSP.

2.8.1 Primera Etapa (PSP0 y PSP0.1)

En esta etapa se debe asignar a uno de los equipos de desarrollo un primer módulo a desarrollar de tamaño estándar o similar a los que realizan regularmente en el proyecto. En la etapa son generados artefactos importantes para los ingenieros informáticos como son el registro de tiempo y defecto, que controlan los errores cometidos y los tiempos de desarrollo de cada tarea a realizar en el módulo asignado.

Fases de la primera Etapa:

1. Planificación.

2. Diseño.
3. Implementación.
4. Prueba.
5. Postmortem.

Artefactos que se generan en la primera etapa:

1. Registro de Tiempo (LOGT).
2. Registro de Defecto (LOGD).
3. Formulario Mejora del Proceso (PIP).
4. Resumen del Plan de Proyecto.
5. Estándares de Diseño.
6. Estándares de Implementación.
7. Plantillas propuesta por Calidad UCI.

Roles que intervienen en la primera etapa:

1. Jefe de Proyecto.
2. Diseñador.
3. Programador.
4. Probador.

2.8.1.1. Fase de Planificación.

En esta fase se realiza la estimación de tiempo y tamaño para el nuevo módulo a desarrollar. La primera estimación es hecha de manera intuitiva por el Jefe de Proyecto ya que no se cuenta con datos históricos. Se generan los LOGT, LOGD para controlar el tiempo destinado a la realización de las tareas del proyecto en esta fase y registrar los defectos encontrados para que no sean repetidos durante el desarrollo de módulos futuros. Se elaboran los estándares de diseño y de implementación a seguir por el equipo de desarrollo, los cuales son realizados por el Jefe de Proyecto y por los Diseñadores y Programadores. Los estándares elaborados en esta fase son muy importantes para un

mejor desarrollo de las próximas fases, donde guiarán el diseño y la implementación. Se recomienda el uso de los formularios LOGT y LOGD propuestos en el presente capítulo.
Tiempo Planificación=Tiempo de Desarrollo Planificación + Tiempo de Corrección de errores de planificación encontrados en la fase de Prueba.

2.8.1.2. Fase de Diseño.

Esta fase está dirigida al diseño donde normalmente se deben generar las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo del software. Interviene el Diseñador, revisa los requerimientos y elabora un diseño acorde a ellos, debe realizar un buen análisis del programa identificando diagramas de clases del análisis por casos de uso, diagramas de clases de diseño, diagramas de interacción (secuencia y colaboración). El tiempo se va registrando en el LOGT a medida que se desarrolla el diseño, y los defectos encontrados en el LOGD.

Tiempo Diseño =Tiempo de Desarrollo Diseño + Tiempo de Corrección de errores de diseño encontrados en la fase de Prueba.

2.8.1.3. Fase de Implementación.

Durante esta fase interviene el Programador que define la organización del sistema en subsistemas, se implementan los elementos de diseño y se integran los resultados en un sistema ejecutable. Luego se procede a la compilación, que es donde se compila hasta que esté sin errores el programa, se repara y registran los defectos encontrados en LOGD, además del tiempo de la fase en LOGT. Se deben generar las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo de la fase.

Tiempo Implementación =Tiempo de Desarrollo Implementación + Tiempo de Corrección de errores de implementación encontrados en la fase de Prueba.

2.8.1.4. Fase de prueba.

En esta fase se corrigen todos los defectos encontrados, se identifican los métodos de prueba y se realizan los casos de pruebas. Los defectos de esta fase son registrados en el LOGD y se registra el tiempo de duración de la misma en LOGT. Una vez culminada la fase de prueba, el resultado será un programa probado exhaustivamente, y formularios LOGT y LOGD completos. Esta etapa es desarrollada por el probador. Se deben generar las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo de la fase.

Tiempo Prueba =Tiempo de Desarrollo Prueba.

2.8.1.5. Fase de Postmortem.

La fase de postmortem consta de tres tareas fundamentales que son, registrar la cantidad de defectos inyectados por fase, los defectos eliminados, así como el registro de tiempo de demora de cada fase. Estas tareas parten, de un Resumen del Plan de Proyecto con tiempo y defectos estimados, los LOGT y LOGD completos resultantes de cada una de las fases, y de un programa corrido y probado. El Resumen del Plan de Proyecto es realizado por el Jefe de Proyecto. Se recomienda el uso del Resumen del Plan de Proyecto propuesto en el presente capítulo.

En esta etapa también se genera el formulario de Mejora del Proceso (PIP), provee un registro de problemas encontrados y mejoras para el proceso. Describe los problemas encontrados en el proyecto, su impacto sobre el producto o proceso, y se identifica el proceso de mejora, donde es posible, estableciendo prioridades en la planificación y registrando las lecciones aprendidas. El formulario PIP debe ser actualizado por todo los integrantes del equipo de desarrollo. Se recomienda el uso del formulario de Mejora del Proceso (PIP) propuesto en el presente capítulo.

Tiempo Postmortem =Tiempo de Desarrollo Postmortem.

2.8.1.6. Resultados de la etapa.

En esta etapa para cada desarrollador según su rol y la fase en que participan se obtiene el Registro de Tiempo (LOGT), Registro de Defectos (LOGD), los estándares de diseño e implementación, el formulario de Mejora del Proceso (PIP) y la productividad para cada miembro del equipo. El Jefe de Proyecto realiza el Resumen del Plan de Proyecto. Se obtiene la primera generación de datos históricos. Son generadas las plantillas propuestas por Calidad UCI: Plantillas de Requisitos, Plantillas de Arquitectura y Diseño, Plantillas de Implementación y Pruebas, Plantillas de Despliegue e Instalación, Plantillas de Gestión de Proyecto, Plantillas de Gestión de Configuración.

2.8.2 Segunda Etapa (PSP1 y PSP2)

Esta etapa comienza cuando al mismo equipo de desarrollo se le asigna un segundo módulo a ser desarrollado. El Jefe de Proyecto debe orientar a su equipo que revise el LOGD y PIP generados en la etapa anterior para que no se repitan los mismos errores encontrados, en el diseño, implementación y del proceso de desarrollo de software. El equipo de desarrollo debe usar los estándares de diseño e implementación elaborados en la etapa anterior para mantener la misma guía a seguir por los Diseñadores y Programadores del proyecto o confeccionarlos. Se propone elaborar las listas de chequeos o utilizar las propuestas en este capitulo.

Fases de la segunda Etapa:

1. Planificación.
2. Diseño.
3. Revisión de Diseño.
4. Implementación.
5. Revisión de Implementación.
6. Prueba.
7. Postmortem.

Artefactos que se generan en la segunda etapa:

1. Registro de Tiempo (LOGT).

2. Registro de Defecto (LOGD).
3. Formulario Mejora del Proceso (PIP).
4. Resumen del Plan de Proyecto.
5. Estándares de Diseño.
6. Estándares de Implementación.
7. Lista de Chequeo de Diseño.
8. Lista de Chequeo de Codificación.
9. Plantillas propuesta por Calidad UCI.

Roles que intervienen en la segunda etapa:

1. Jefe de Proyecto.
2. Diseñador.
3. Programador.
4. Probador.

2.8.2.1. Fase de Planificación.

En esta fase se estiman los recursos, utilizando el método matemático de estimación PROBE sobre la base de los datos históricos existentes. Esta estimación es realizada por el Jefe de Proyecto. Se generan los LOGT y LOGD propios de esta fase. Se actualizan los estándares de diseño y de implementación a seguir por el equipo de desarrollo en caso necesario, de lo contrario se utilizan los generados en la primera etapa. Se generan las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo del software.

Tiempo Planificación=Tiempo de Desarrollo Planificación + Tiempo de Corrección de errores de planificación encontrados en la fase de Prueba.

2.8.2.2. Fase de Diseño.

En esta fase se desarrollan los mismos artefactos de la primera etapa durante la fase de diseño e interviene el mismo rol, el Diseñador. Es realizada teniendo en cuenta todos los defectos encontrados

en la etapa anterior, con el objetivo de evitar cometer una vez más los mismos. Se generan el LOGT y LOGD para la fase. Se generan las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo del software.

Tiempo Diseño =Tiempo de Desarrollo Diseño + Tiempo de Corrección de errores de diseño encontrados en la fase de Prueba.

2.8.2.3. Fase de Revisión de Diseño.

Esta fase se introduce para minimizar los defectos de diseño encontrados en la fase de Prueba. Es realizada por el diseñador y se genera la Lista de Chequeo de Diseño.
El objetivo de utilizar la lista de chequeo de diseño es guiar una revisión efectiva de diseño, verificar que todos los datos de seguridad son de fuente segura y al final revisar el diseño para ver si está conforme con todos los estándares de diseño aplicados. Se recomienda el uso de la lista de chequeo propuesta en el presente capítulo, en caso contrario debe ser elaborada.

Tiempo Revisión de Diseño =Tiempo de Desarrollo Revisión de Diseño.

2.8.2.4. Fase de Implementación.

En esta fase se desarrollan los mismos artefactos de la primera etapa durante la fase de implementación e interviene el mismo rol, el Programador. Los programadores deben utilizar los estándares de implementación realizados en la primera etapa y revisar el formulario PIP evitando repetir errores ya cometidos. Se generan las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo del software.

Tiempo Implementación =Tiempo de Desarrollo Implementación + Tiempo de Corrección de errores de implementación encontrados en la fase de Prueba.

2.8.2.5. Fase de Revisión de Implementación.

Esta fase se introduce para minimizar los defectos de implementación encontrados en la compilación y en la fase de Prueba. Es realizada por el programador y se genera la Lista de Chequeo de Codificación. Al realizar la primera compilación se debe obtener un programa con menos defectos.
La lista de chequeo de código es recomendable utilizarla para guiar una revisión efectiva del código, verifica que el código cubra todo el diseño. Se asegura que el código este acorde al estándar de codificación donde se verifica que todos los ficheros estén apropiadamente declarados, abiertos y cerrados. Se recomienda el uso de la lista de chequeo de código propuesta en el presente capítulo, en caso contrario debe ser elaborada.

Tiempo Revisión de Implementación =Tiempo de Desarrollo Revisión de Implementación.

2.8.2.6. Fase de Prueba.

La fase de prueba genera los mismos artefactos de la primera etapa, se identifican los métodos de prueba y se realizan los casos de pruebas. Interviene el Probador que corrige todos los defectos inyectados en el diseño e implementación y los registra en el LOGD, el tiempo que se demora en el desarrollo de las pruebas es registrado en el LOGT. Se generan las plantillas que exige Calidad UCI para registrar los artefactos propios del desarrollo del software.

Tiempo Prueba =Tiempo de Desarrollo Prueba.

2.8.2.7. Fase de Postmortem.

En esta fase se desarrollan los mismos artefactos de la primera etapa durante la fase de postmortem e interviene el equipo de desarrollo completo. Se genera los LOGT, LOGD, el formulario PIP y el Resumen del Plan de Proyecto realizado por el Jefe de Proyecto. Se cuenta con los datos históricos de la primera etapa y se deben comparar con los obtenidos en esta, los resultados de esta comparación tienen que ser analizados por el Jefe de Proyecto para conformar un balance de las mejoras introducidas durante la aplicación del PSP.

Tiempo Postmortem =Tiempo de Desarrollo Postmortem.

2.8.2.8. Resultados de la etapa.

En esta etapa para cada desarrollador según su rol se obtiene el Registro de Tiempo (LOGT) y Registro de Defectos (LOGD), el formulario de Mejora del Proceso (PIP) y la productividad para cada miembro del equipo. El Jefe de Proyecto realiza el Resumen del Plan de Proyecto. El diseñador genera la Lista de Chequeo de Diseño y el programador la Lista de Chequeo de Codificación. Se obtiene la segunda generación de datos históricos. Son generadas las plantillas propuestas por Calidad UCI para registrar los artefactos propios del desarrollo del software: Plantillas de Requisitos, Plantillas de Arquitectura y Diseño, Plantillas de Implementación y Pruebas, Plantillas de Despliegue e Instalación, Plantillas de Gestión de Proyecto, Plantillas de Gestión de Configuración.

2.8.3 Tercera Etapa (Métricas de Calidad)

Esta etapa comienza cuando al mismo equipo de desarrollo se le asigna un nuevo módulo a ser desarrollado. El Jefe de Proyecto debe orientar a su equipo que revisen el LOGD y PIP generados en las etapas anteriores para que no se repitan los mismos errores detectados. Durante la aplicación de la primera y segunda etapa el equipo de desarrollo obtiene los conocimientos indispensables sobre la aplicación del PSP, para introducir en esta tercera etapa el cálculo de las métricas de calidad. El objetivo de la etapa es obtener datos precisos de la calidad del trabajo realizado por cada integrante del equipo de desarrollo, y del proceso de desarrollo como un todo.

Fases de la tercera Etapa:

1 Planificación.
2 Diseño.
3 Revisión de Diseño.
4 Implementación.
5 Revisión de Implementación.
6 Prueba.
7 Postmortem.

Artefactos que se generan en la tercera etapa:

1 Registro de Tiempo (LOGT).
2 Registro de Defecto (LOGD).
3 Formulario Mejora del Proceso (PIP).
4 Resumen del Plan de Proyecto con métricas de calidad incluidas.
5 Estándares de Diseño.
6 Estándares de Implementación.
7 Lista de Chequeo de Diseño.
8 Lista de Chequeo de Codificación.
9 Plantillas Propuesta por Calidad UCI.

Roles que intervienen en la tercera etapa:

1 Jefe de Proyecto.
2 Diseñador.
3 Programador.
4 Probador.

En esta tercera etapa se realizan las mismas fases que en la segunda, intervienen los mismos roles y se generan los mismos artefactos. El cambio en esta etapa ocurre en la fase de Postmortem, cuando al Resumen de Plan de Proyecto se le agrega el cálculo de las Métricas de Calidad.

2.8.3.1. Fase de Postmortem.

En esta etapa la fase de postmortem genera los mismos artefactos y roles de la segunda etapa. El formulario PIP debe ser realizado por todo los integrantes del equipo de desarrollo. Al Resumen de Plan de Proyecto se le agrega el cálculo de las Métricas de Calidad para evaluar la calidad del producto y del proceso de desarrollo. Se realiza un análisis de los resultados de las métricas y con los datos obtenidos el Jefe de Proyecto hace el balance de la eficiencia del equipo, en la aplicación del PSP durante el desarrollo del módulo.

Las métricas propuestas a ser calculadas son índice de calidad del proceso y tasa de revisión, debido a que los valores que utilizan son mayormente de tiempo y defectos encontrados durante el desarrollo del producto, y son precisamente este tipo de datos los que son registrados en las etapas propuestas para la aplicación de PSP en el LOGT y LOGD.

Para realizar el cálculo del índice de calidad del proceso hay que obtener cinco métricas: la primera es evaluar la calidad de las revisiones de código. Si se sigue el proceso de revisión de código y se usa una lista de chequeo para las revisiones de código se tendrá una revisión con un mayor rendimiento. El mejor indicador de esto es encontrar muy pocos defectos durante la compilación. En el PSP, es considerado como poco, encontrar 10 o menos defectos por KLOC durante la compilación. Sin una buena revisión de código, es bien difícil obtener ese número. Desafortunadamente, si su ambiente de desarrollo no tiene una fase de compilación, estas medidas no estarán disponibles. En este caso, se pueden usar los datos de los tipos de defectos de codificación encontrados en las pruebas de unidad.

Las métricas en las revisiones de diseño son algo más complejas. Aunque el número de defectos encontrados en las pruebas de unidad puede ser una medida útil de la calidad del diseño, también incluye calidad del código. Si, sin embargo, los defectos por KLOC en la compilación estuvieran por debajo de 10, entonces la calidad es probablemente mejor y los defectos de las pruebas de unidad pueden medir la calidad del diseño. De esta manera, sin los datos de defectos durante la compilación no se puede interpretar las medidas de defectos encontrados durante las pruebas de unidad. Las medidas sugeridas por PSP especifican que si los defectos por KLOC en la fase de compilación son menos de 10, una cantidad de 5 defectos por KLOC en las pruebas de unidad indican un diseño con calidad. Los criterios de PQI son los siguientes:

Factores de calidad:

1. Diseño/Tiempo de codificación = tiempo de diseño/tiempo de codificación, con rango desde 0.0 a 1.0.
2. Tiempo de revisión del diseño = 2*tiempo de revisión del diseño/tiempo de revisión, con un rango entre 0.0 y 1.0
3. Tiempo de revisión de código = 2*tiempo de revisión de código/tiempo de codificación, con un rango entre 0.0 y 1.0

4. Defectos de compilación/KLOC = 20/(10+defectos de compilación/KLOC), con un rango entre 0.0 y 1.0
5. Defectos de pruebas de unidad/KLOC = 10/(5+defectos de pruebas de unidad/KLOC), con un rango entre 0.0 y 1.0
6. Índice Calidad Proceso =multiplicación de los cinco factores de calidad.

La métrica tasa de revisión es principalmente usada para revisiones de código e inspecciones y mide las LOC o páginas revisadas por hora. Si, por ejemplo, se pasan 20 minutos revisando un programa de 100 LOC, la tasa de revisión sería de 300 LOC por hora. No existen tasas definitivas sobre cuál rendimiento de revisión es incorrecto. Sin embargo, los datos de PSP muestran que altos rendimientos están asociados con bajas tasas de revisiones y que bajos rendimientos están relacionados con altas tasas de revisión. La proporción ideal es donde el tiempo de revisión de diseño o código, es la mitad del tiempo de desarrollo del diseño y código.

Tasa Revisión de Diseño =Tiempo Diseño
Tiempo Revisión Diseño

Tasa Revisión de Código=Tiempo Código
Tiempo Revisión Código

Tiempo Postmortem =Tiempo de Desarrollo Postmortem.

2.8.3.2. Resultados de la etapa.

En esta etapa para cada desarrollador según su rol se obtiene el Registro de Tiempo (LOGT) y Registro de Defectos (LOGD), el formulario de Mejora del Proceso (PIP) y la productividad para cada desarrollador. El Jefe de Proyecto realiza el Resumen del Plan de Proyecto incluyendo el cálculo de las Métricas de Calidad. El diseñador genera la Lista de Chequeo de Diseño y el programador la Lista de Chequeo de Codificación. Se obtiene otra generación de datos históricos. Son generadas las plantillas propuestas por Calidad UCI para registrar los artefactos propios del desarrollo del software: Plantillas de Requisitos, Plantillas de Arquitectura y Diseño, Plantillas de Implementación y Pruebas, Plantillas de Despliegue e Instalación, Plantillas de Gestión de Proyecto, Plantillas de Gestión de Configuración.

2.8.3.3. Resumen de la solución propuesta.

Etapas/fases	Artefactos	Roles	Fases
Primera Etapa	Registro de Tiempo (LOGT). Registro de Defecto (LOGD). Formulario Mejora del Proceso (PIP). Resumen del Plan de Proyecto. Estándares de Diseño. Estándares de Implementación. Plantillas propuesta por Calidad UCI.	Jefe de Proyecto. Diseñador. Programador. Probador.	Planificación. Diseño. Implementación. Prueba. Postmortem.
Segunda Etapa	Registro de Tiempo (LOGT). Registro de Defecto (LOGD). Formulario Mejora del Proceso (PIP). Resumen del Plan de Proyecto. Estándares de Diseño. Estándares de Implementación. Lista de Chequeo de Diseño. Lista de Chequeo de Codificación. Plantillas propuesta por Calidad UCI.	Jefe de Proyecto. Diseñador. Programador. Probador.	Planificación. Diseño. Revisión de Diseño. Implementación. Revisión de Implementación. Prueba. Postmortem.
Tercera Etapa	Registro de Tiempo (LOGT). Registro de Defecto (LOGD). Formulario Mejora del Proceso (PIP). Resumen del Plan de Proyecto. Estándares de Diseño. Estándares de Implementación. Lista de Chequeo de Diseño. Lista de Chequeo de Codificación. Plantillas propuesta por Calidad UCI. Calculo de las métricas de calidad.	Jefe de Proyecto. Diseñador. Programador. Probador.	Planificación. Diseño. Revisión de Diseño. Implementación. Revisión de Implementación. Prueba. Postmortem.

Tabla 3: Resumen de la solución propuesta.

2.9. Registro de Tiempo (LOGT).

El objetivo de registrar el tiempo es obtener datos de cómo se trabaja realmente. La forma y el procedimiento utilizados para reunir los datos no son tan importantes mientras los datos sean exactos y completos.

La cantidad típica de tiempo no interrumpido que los ingenieros dedican a sus tareas es generalmente inferior a una hora. Medir el trabajo en unidades de horas no proporcionará el detalle necesario para posteriormente planificar y gestionar el trabajo. Es mucho más fácil controlar el tiempo en minutos que en fracciones de una hora.
En la tabla de registros de tiempo se introduce en cada línea un período de tiempo:

- Fecha: la fecha de realización de alguna actividad, como asistir a clase, analizar o escribir un programa.
- Comienzo
- Fin
- Interrupción: cualquier pérdida de tiempo debida a interrupciones
- Delta tiempo: tiempo dedicado a cada actividad en minutos, entre los tiempos de comienzo y fin, menos el tiempo de interrupción.
- Actividad: nombre descriptivo para la actividad
- Comentarios: descripción más completa de lo que se está haciendo, el tiempo de interrupción o cualquier cosa que pueda ser útil para analizar posteriormente los datos de tiempo.
- C (completado): se marca esta columna cuando se termina una tarea correspondiente a alguna actividad, como finalizar un diagrama, escribir el módulo de un programa, finalizar la lectura de un capítulo.
- U (unidades): el número de unidades de una tarea acabada.

Nombre: _____ **Fecha:** _____
Profesor: _____ **# Programa:** _____

Fecha	Inicio	Fin	Tiempo interrupción	Delta tiempo	Fase	Comentarios	C	U

Tabla 4: Plantilla propuesta del Cuaderno de Registro del Tiempo.

2.10. Diferentes formas de encontrar y corregir defectos.

La primera herramienta que los ingenieros utilizan es un compilador. Este generara código hasta que encuentre algunos caracteres que no puede interpretar. Los compiladores pueden identificar muchos

defectos sintácticos, pero no te pueden decir lo que pretenden, solamente proporcionan síntomas de defectos y debes entender donde y cual es su problema aunque normalmente harás esto rápidamente, en ocasiones puedes necesitar mucha dedicación. El compilador no detectara todos los defectos sintácticos.

Una segunda forma de encontrar defectos, es por medio de las pruebas aunque hay muchas clases de pruebas todas requieren que los examinadores proporcionen datos de pruebas y condiciones de pruebas. La calidad de las pruebas esta gobernada por el grado en que estos escenarios cubren todas las funciones importantes del programa. El examinador ejecuta estos casos de pruebas para ver si el programa proporciona los resultados adecuados Aunque las pruebas pueden utilizarse para comprobar casi cualquier función del programa tienen varias desventajas: solo suponen el 1er paso de corrección de defectos, cada prueba verifica solamente un conjunto de condiciones del programa.

La tercera forma de encontrar los defectos, es la más común de todas. Consisten entregar programas defectuosos y esperar que los usuarios rectifiquen e informen de los defectos. Esta es la estrategia más costosa.

La cuarta y mas efectiva forma de encontrar y corregir defectos es pensar personalmente el código fuente del programa, aunque esto puede parecer una forma difícil de limpiar un programa defectuoso se trata de la forma mas rápida y eficiente.

2.10.1 Importancia de las listas de comprobación para la revisión del código de los programas que se implementan.

Para encontrar y corregir cada defecto en el programa se siguen procedimientos precisos, una lista de comprobación ayuda a realizar mejor estas tareas. Pueden ser una fuente de ideas, seguir una lista de comprobación personal para saber como revisar tu código. Utilizando la lista correctamente, puedes detectar defectos en cada uno de sus pasos. Se debe comparar la lista de comparación con la de otros ingenieros, puede sugerir aproximaciones útiles para la revisión. Encapsula la experiencia personal. Utilizándola con irregularidad y mejorándola, mejoraras en la detección de los defectos de tu programa, ayudara a encontrar estos defectos en menos tiempo o para cualquier diseño por muy sencillo que sea.

2.10.2 Registrar los defectos encontrados en un programa, en el Cuaderno de Registro de Defectos.

En el cuaderno de registro de defecto los programadores tendrán controlado todos los defectos encontrados durante la realización de las actividades. Se registrara la fecha del defecto encontrado, número, el tipo de defecto, la fase en que es introducido, cuando es eliminado, el tiempo que se demora su corrección y los defectos recogidos durante su eliminación.

Nombre: _____ **Fecha:** _____
Profesor: _____ **# Programa:** _____

Fecha	#	F Inyectado	Tipo	F Eliminado	T Corrección	D Corregido	Descripción

Tabla 5: Plantilla propuesta del Cuaderno de Registro de Defectos.

2.11. Resumen del Plan del Proyecto con los datos de defectos introducidos y eliminados en cada una de las fases, a partir del Cuaderno de Registro de Defectos.

En el resumen del plan del proyecto se registra el tiempo (LOC) que se demora en corregir los defectos encontrados durante la realización del Cuaderno de Registro de Defectos. Con este cuaderno se controla el tiempo de eliminación de defectos durante la realización del producto para tener un estimado de tiempo a tener en cuenta en la entrega del producto final.

Nombre: _____ **Fecha:** _____
Profesor: _____ **# Programa:** _____

Resumen	Plan	Real	Hasta la Fecha
LOC/Hora	8,88	9,80	9,23
Defectos/KLOC			
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	44	57	105

Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	13	18	33	4,8
Diseño	11	43	55	8,1
Codificación	130	162	308	45,2
Revisión del código				
Compilación	44	21	70	10,2
Pruebas	82	73	165	24,2
Postmorten	17	32	51	7,5
Total	297	349	682	100,0
Tiempo Máximo	392			
Tiempo Mínimo	203			

Defectos introducidos	*Plan*	*Actual*	*Hasta la Fecha*	*% Hasta la Fecha*	*Def./Hora*
Planificación					
Diseño		2	2	25,0	
Codificación		6	6	75,0	
Revisión del código					
Compilación					
Pruebas					
Total		8	8	100,0	

Defectos eliminados	*Plan*	*Actual*	*Hasta la Fecha*	*% Hasta la Fecha*	*Def./Hora*
Planificación					
Diseño					
Codificación					
Revisión del código					
Compilación		6	6	75,0	
Pruebas		2	2	25,0	
Total		8	8	100,0	

Figura 4: Ejemplo de Plantilla propuesta del Resumen del Plan del Proyecto.

2.12. Propuesta de Mejora del Proceso (PIP).

El formulario de Mejora del Proceso (PIP), provee un registro de problemas encontrados y mejoras para el proceso. Describe los problemas encontrados en cada proyecto. El impacto sobre el producto o proceso y se identifica el proceso a mejorar, donde es posible, así como relacionar la propuesta al problema. Contiene los datos estimados y reales del proyecto en un formato conveniente y legible, el

objetivo fundamental es registrar las ideas sobre las mejoras en el proceso, estableciendo prioridades en la planificación y registrando las lecciones aprendidas.

Este formulario se usa para establecer prioridades en el plan de mejoras, registrar ideas sobre mejoras en el proceso y registrar lecciones aprendidas y soluciones iníciales. Consta de un encabezamiento, en el cual se encuentra el nombre, la fecha, el profesor, el número del programa y el nombre del proceso utilizado.

Existen dos casillas fundamentales la del problema y la de la propuesta, en la del problema se encuentra una descripción detallada del mismo, tan claro como sea posible, se deja registrado el grado de dificultad encontrado y el impacto en el producto, se enumera los problemas en cada formulario y se usa un número de secuencia conveniente comenzando con 1 en cada formulario.

En la casilla propuesta, se describe la mejora del proceso propuesta tan explícita como sea posible, se referencia el elemento específico del proceso y el número del problema, si es importante se debe describir la prioridad.

En la casilla comentarios, las lecciones aprendidas (Lec Aprendidas); se introducen las condiciones que se deban recordar, debido al buen o mal trabajo durante el proceso de desarrollo de proyectos, como comentarios del proceso. Una propuesta al formulario anteriormente descrito es la siguiente:

Nombre: _____ **Nombre del Proceso:** _____
Profesor: _____ **# Programa:** _____ **Fecha:** _____

Problema				**Propuesta**		**Cometarios**
Descripción	**Grado Dificultad**	**#**	**#Secuencia**	**Descripción**	**Prioridad**	**Lec Aprendidas**

Tabla 6: Propuesta del formulario Mejora del Proceso (PIP) de PSP0.1

2.13. Lista de Chequeo de Código PSP2.

La lista de chequeo de código es recomendable utilizarla pues esta guía una revisión efectiva del código, verifica que el código cubra todo el diseño y que los includes estén completos. La lista de chequeo de código verifica el formato de la llamada a funciones y chequea todas las cadenas que

estén identificadas por punteros y terminadas en NULL. Además chequea que cada formato de salida tenga un salto de líneas apropiado. Aquí se verifica el uso apropiado de operadores lógicos y se chequea línea por línea de código según la sintaxis de la instrucción y la puntuación adecuada. Se asegura que el código este acorde al estándar de codificación donde se verifica que todos los ficheros estén apropiadamente declarados, abiertos y cerrados. Una propuesta a la lista anteriormente descrita es la siguiente:

Nombre: _____ **Fecha:** _____
Nombre del Programa: _____ **# Programa:** _____
Profesor: _____ **Lenguaje:** _____

Objetivo	Guía una revisión efectiva del código
General	Una vez que se complete cada paso de la revisión, marcar cada elemento en el cuadro de la derecha Completar la lista de chequeo para una unit del programa antes de comenzar a revisar la próxima.
Completo	Verificar que el código cubra todo el diseño:
Includes	Verificar que los includes estén completos
Inicializaciones	Chequear la inicialización de variables y parámetros: • Al iniciar el programa • Al comenzar cada lazo • A la entrada de cada función /procedimiento
Llamadas	Chequear el formato de la llamada a función: • Punteros • Parámetros • Uso de &
Nombres	Chequear el nombre letra a letra y su uso: • Su consistencia • Si está en el alcance declarado • Hacer que todas las estructuras y las clases usen la referencia "."
Cadenas	Chequear que todas las cadenas estén: • Identificadas por punteros • Terminadas en NULL
Punteros	Chequear que: • Los punteros estén inicializados en NULL • Los punteros sean eliminados solo después de new • Los nuevos punteros sean siempre eliminados después de su uso
Formato de Salida	Chequear en cada formato de salida. • Salto de líneas apropiados • Espacios apropiados
Pares {}	Asegurar que {} están adecuadamente casados

Operadores lógicos	• Verificar el uso apropiado de ==, =, //, y otros • Chequear que cada función lógica tenga su propio ()
Chequeo línea por línea	Chequear para cada LOC de código: • Sintaxis de la instrucción • Puntuación adecuada
Estándares	Asegurase que el código esté acorde a l estándar de codificación
Abrir y Cerrar Fichero	Verificar que todos los ficheros estén: • Apropiadamente declarados • Abiertos • Cerrados

Tabla 7: Propuesta de la Lista de Chequeo de Código de PSP2.

2.14. Lista de Chequeo de Diseño PSP2.

El objetivo de utilizar la lista de chequeo de diseño es guiar una revisión efectiva de diseño, se revisa el programa completo por categoría de lista de chequeo. Se asegura que los requerimientos, especificaciones y diseño de alto nivel estén completamente cubiertos por el diseño donde todas las salidas fueron producidas, todas las entradas necesarias fueron suministradas y todos los includes requeridos fueron colocados. Aquí se verifica que la secuencia del programa sea adecuada y que los tipos de datos abstractos como pilas, listas y colas estén en el orden adecuado, además verifican que todos los lazos estén adecuadamente inicializados, incrementados y terminados, examinando cada sentencia condicional. Asegura que se opere bien con valores vacíos, llenos, mínimos, máximos, negativos o cero para todas las variables. Protege las condiciones fuera de los límites y desbordamiento de búfer y maneja todas las condiciones de entrada incorrectas. Esta lista de chequeo de diseño verifica que todos los datos de seguridad son de fuente segura y al final revisa el diseño para ver si está conforme con todos los estándares de diseño aplicados. Una propuesta a la lista anteriormente descrita es la siguiente:

Nombre: _____ **Fecha:** _____
Nombre del Programa: _____ **# Programa:** _____
Profesor: _____ **Lenguaje:** _____

Objetivo	Guía una revisión efectiva del diseño
General	Revisar el programa completo para cada categoría de la lista de chequeo, una sola a la vez. Una vez que se complete cada paso de la revisión, marcar cada elemento en el cuadro de la derecha Completar la lista de chequeo para un programa o elemento de programa

	antes de comenzar a revisar el próximo.
Completo	Asegurarse que los requerimientos, especificaciones y diseño de alto nivel estén completamente cubiertos por el diseño: Todas las salidas fueron producidas Todas las entradas necesarias fueron suministradas Todos los includes requeridos fueron colocados
Límites externos	Cuando el diseño asume o cuenta con límites externos, determinar si el comportamiento es correcto con valores nominales, en los límites y más allá de estos
Lógica	Verificar que la secuencia del programa sea la adecuada: Pilas, listas y otros estén en el orden adecuado La recursión ocurra apropiadamente Verificar que todos los lazos estén adecuadamente inicializados, incrementados y terminados Examinar cada sentencia condicional y verificar todos los casos
Límites internos	Cuando el diseño asume o cuenta con límite internos, determinar si el comportamiento es correcto con valores nominales, en los límites y más allá de estos
Casos especiales	Chequear todos los casos especiales: Asegurarse que operan bien con valores vacíos, llenos, mínimos, máximos, negativos o cero para todas las variables. Proteger contra condiciones fuera de los límites y desbordamientos del búfer. Asegurar que las condiciones imposibles son absolutamente imposibles Manejar todas las condiciones de entradas incorrectas
Uso funcional	Verificar que todas las funciones, procedimientos u objetos son completamente comprendidos y adecuadamente usados. Verificar que todas las referencias externas son definidas
Consideraciones del sistema	Verificar que el sistema no causa que los límites sean excedidos Verificar que todos los datos de seguridad son de fuentes seguras Verificar que todas las condiciones de seguridad conforman las especificaciones de seguridad.
Nombres	Verificar lo siguiente: Todos los nombres y tipos especiales están claros y están definidos específicamente El alcance de todas las variables y parámetros está evidente y bien definido.

Tabla 8: Propuesta de la Lista de Chequeo de Diseño de PSP2.

2.15. Valoraciones de expertos.

Ing. Enrique Pérez plantea:

La solución planteada por el estudiante Carlos Yosvel Alayón, para aplicar una estrategia del PSP en los proyectos del polo productivo de video y sonido digital, debe ayudar a los integrantes de los equipos de desarrollo del software a entender la importancia de una disciplina personal en el desarrollo de los proyectos. Proporciona fundamentos más rigurosos para la posterior introducción de nuevos formularios, plantillas, fases en el desarrollo de software y cálculos de métricas mejorando los métodos de aplicar PSP por los ingenieros informáticos. Desafortunadamente los ingenieros no cumplen siempre cabalmente las soluciones planteadas para su mejor preparación y deben ser controlados constantemente.

Ing. Yunior Montaner Hernández plantea:

La solución planteada en este trabajo de diploma cumple con las expectativas del equipo de desarrollo. Este trabajo brinda las condiciones necesarias para lograr la eficiencia personal requerida y un mejoramiento paulatino en la utilización del PSP en los proyectos del polo productivo de video y sonido digital.

Ing. Indira Lilled Laurencio Fuentes plantea:

La estrategia planteada del PSP puede ser aplicada, ya que se utilizan los principales formularios y plantillas que genera el PSP. Los proyectos del polo productivo de video y sonido digital de la facultad 9 deben aplicar lo antes posible la solución propuesta, los proyectos en la UCI requieren de una calidad reconocida y la aplicación del PSP es una de las forma de lograrla.

2.16. Conclusiones.

La aplicación del PSP en el proceso de desarrollo de software es, sin lugar a dudas, uno de los elementos fundamentales para tal propósito, por lo que los estudiantes e ingenieros deben comenzar a entenderlo y aplicarlo desde el inicio del desarrollo del software.

Capítulo 2: Propuesta de solución

Ante la tarea de convertir la informática en una de las más importantes ramas productivas del país, la UCI tiene el compromiso de formar ingenieros capaces de asumir y cumplir sus compromisos de trabajo con la más alta calidad.

Con la introducción de PSP desde los primeros años de la carrera Ingeniería Informática y de forma gradual, los futuros ingenieros informáticos del país inferirán la necesidad de saber gestionar correctamente sus tiempos y compromisos, no solo para el trabajo que desempeñarán sino para otras facetas de su vida.

CONCLUSIONES

El nivel actual del mercado mundial de software, requiere de productos de calidad trazada por la comunidad internacional de desarrolladores de software. Para mejorar la calidad del software, debe centrarse en los procesos requeridos para producir consistentemente productos de calidad. Buscar los métodos más efectivos para encontrar defectos es la manera más efectiva de prevenirlos. La aplicación PSP por los equipos de desarrollo, trae consigo el aumento de la calidad del software.

Durante el transcurso de la investigación realizada sobre la aplicación de una estrategia del PSP en los proyectos del polo productivo de video y sonido digital se resuelve problemática planteada; cómo aplicar PSP en los proyectos del polo productivo de video y sonido digital de la facultad 9, la hipótesis es probada por la entrevista y valoraciones planteada por los expertos sobre el tema, lograron los objetivos y las tareas de la investigación. Después de completar su entrenamiento en PSP usted podrá definir y usar sus propios procesos personales para cada aspecto de su trabajo, desde requerimientos hasta pruebas y mantenimiento de productos.

Miles de ingenieros han sido entrenados actualmente en PSP y han encontrado universalmente que esos métodos los ayudan a realizar mucho mejor su trabajo. Por otra parte han encontrado también que usando esos métodos en un equipo TSP constituyen realmente una forma de trabajo reconfortante.

Un proceso definido puede ayudarlo a comprender su desempeño como desarrollador de software y ver dónde y cómo mejorar.

RECOMENDACIONES

Se recomienda llevar a cabo la aplicación inmediata de la solución planteada en el capítulo 2 realizando un estudio previo del PSP, realizar dentro de 5 años una nueva investigación para actualizar lo planteado en este trabajo.

REFERENCIA BIBLIOGRÁFICA

1. R.S. Pressman (1992).
2. ISO 8402 (UNE 66-001-92).
3. Watts S. Humphrey. "A discipline for software engineering". Addison- Wesley, 1995
4. Watts S. Humphrey. "Managing the Software Process". Addison- Wesley, 1989
5. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Carnegie Mellon University and Software Engineering Institute. "Capability Maturity Model, The: Guidelines for Improving the Software Process". Addison-Wesley, 1995.
6. Ailyn Febles Estrada. "Medir el proceso de control de configuración, ¿una utopía para la Industria Nacional de Software?".
7. Watts S. Humphrey, "PSP. A Self – Improvement Process for Software Engineers", 2005.
8. Roger S. Pressman, "Ingeniería del Software. Un enfoque práctico", 2005.
9. Watts S. Humphrey, "Introducción al Proceso Software Personal", 2001.
10. Fidel Castro Díaz-Balart, "Ciencia, Tecnología y Sociedad. Hacia un desarrollo sostenible en la Era de la Globalización". Editorial Científico-Técnica, 2004.
11. Yeleny Zulueta Véliz. "Introducción de técnicas del Personal Software Process desde los primeros años en la formación del ingeniero informático". http://www.inf.udec.cl/revista/ediciones/edicion14/zulueta.pdf
12. HUMPHREY W. A Personal Commitment to Software Quality.1994. http://www.sei.cmu.edu/publications/documents/95.reports/95.ar.psp.qual.html
13. VELASCO, Perla Inés. PSP: Una alternativa para mejorar los procesos del software. http://www.lania.mx/biblioteca/newsletters/2003-primaveraverano/psp.html
14. El modelo de capacidad de Madurez y su enfoque al Proceso Personal de Software (PSP). http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/pelaez_r_jj/

BIBLIOGRAFÍA

1. Humphrey, W., "Introducción al Proceso de Software Personal". 2001.
2. Humphrey, W.S., "PSP. A Self – Improvement Process for Software Engineers". 2005.
3. Roger S. Pressman, "Ingeniería del Software. Un enfoque práctico", 2005.
4. Humphrey, W. "A discipline for software engineering", 1995.
5. Roger S. Pressman "Ingeniería de Software: un enfoque práctico", 5ta. Edición.
6. Teleformación/Bibliografía/Temas/PSP/PSP_for_Eng_Student_V4.1
7. URL: http://hornet.ls.fi.upm.es/DSP/Lecciones.
8. Yeleny Zulueta Véliz. "Introducción de técnicas del Personal Software Process desde los primeros años en la formación del ingeniero informático".
9. Fidel Castro Díaz-Balart, "Ciencia, Tecnología y Sociedad. Hacia un desarrollo sostenible en la Era de la Globalización". Editorial Científico-Técnica, 2004.
10. HUMPHREY W. A Personal Commitment to Software Quality.1994
11. El modelo de capacidad de Madurez y su enfoque al Proceso Personal de Software (PSP).
12. Ailyn Febles Estrada. "Medir el proceso de control de configuración, ¿una utopía para la Industria Nacional de Software?".
13. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Carnegie Mellon University and Software Engineering Institute. "Capability Maturity Model, The: Guidelines for Improving the Software Process". Addison-Wesley, 1995.
14. VELASCO, Perla Inés. PSP: Una alternativa para mejorar los procesos del software.
15. Álvarez S., CMM en el contexto de ISO–9000:2000, 2002

ANEXOS

Anexo1: Resumen Semanal de actividades

Nombre: Alejandro Fecha: 26/12/2005

Tarea	Revisión	Ultras.	Parte	GS(leer)	GS Sem.	CE	Diseño	Total
Fecha								
D 18/12								
L	14	160	25	42				241
M	10	170				45	6	231
Mi			20	50	150			220
J	11	155	25		90	40	6	327
V			25	50			4	79
S								
Totales	35	485	95	142	240	85	16	1098
Tiempos y Medias del Periodo				Número de Semanas(número anterior +1):				2
Resumen de las semanas anteriores								
Totales	63	865	131	122	250	0	9	1440
Med.	63	865	131	122	250	0	9	1440
Máx.	63	865	131	122	250	0	9	1440
Mín.	63	865	131	122	250	0	9	1440
Resumen incluyendo la Ultima semana								
Total								
Med.								
Máx.								
Mín.								

Respuesta:

Resumen incluyendo la Ultima semana								
Total	98	1350	226	264	490	85	25	2538
Med	49	675	113	132	245	42.5	12.5	1269
Máx	63	865	131	142	250	85	16	1440
Mín	35	485	95	122	240	85	9	1098

Anexo2: Propuesta del formulario de Reporte de Prueba de PSP1.

Nombre: _____ Fecha: _____

Profesor: _____ # Programa: _____

Nombre prueba	#	Objetivos	Condiciones	Resultados esperados	Resultados actuales

Anexo3: Propuesta del Plan de Tareas de PSP1.1

Nombre: _____ Fecha: _____

Profesor: _____ # Programa: _____

Tarea		Plan					Real		
#	Nombre	Horas	VP	H Acumuladas	VP Acumulado	Fecha	Fecha	EV	EV Acumulado
Totales									

Anexo4: Propuesta del Plan de Cronograma de PSP1.1

Nombre: _____ Fecha: _____

Profesor: _____ # Programa: _____

		Plan			Real			
# Semana	Fecha	Fecha	HD	HA	HA	HD	Valor Acumulado	Valor Ajuste
Totales								

Anexo5: Propuesta del LOG de seguimiento de problemas PSP3.

Plan: _____ Fecha: _____

Estudiante: _____ Real: _____

Profesor: _____ Lenguaje: _____

# Problema	Fecha:	Fase:
Descripción:		
Resolución:		
Fecha:		
	Fecha:	Fase:
Descripción:		
Resolución:		
Fecha:		
	Fecha:	Fase:
Descripción:		
Resolución:		
Fecha:		

Anexo6: Propuesta del Formulario Resumen del Ciclo PSP3.

Plan: _____ Fecha: _____

Estudiante: _____ Real: _____

Profesor: _____ Lenguaje: _____

Ciclos Tamaño del Programa (LOC)	A la fecha	1	2	3	4	5	Total
Base (B)							
Eliminados (D)							
Modificados (M)							
Adicionados (A)							
Reutilizados (R)							
Total nuevas y cambiadas(N)							
Total LOC (T)							
Total nuevas reutilizables							
Tiempo en fase (min.)							
Diseño							
Revisión de diseño							
Codificación							
Revisión de código							
Compilación							
Pruebas							
Total							
Defectos Inyectados							
Diseño							
Revisión de diseño							
Codificación							
Revisión de código							
Compilación							
Pruebas							
Total							
Defectos Removidos							
Diseño							
Revisión de diseño							
Codificación							
Revisión de código							
Compilación Pruebas							

Anexo7: Propuesta del Formulario Resumen del Plan de Proyecto PSP3.

Estudiante: _____ # Fecha: _____

Programa: _____ # Programa: _____

Profesor: _____ Lenguaje: _____

Resumen	**Plan**	**Real**	**A la fecha**	
LOC/Hora				
Tiempo Planificado				
Tiempo Real				
CPI (Razón de Costo-Planificación)		*Plan/Real*		
% Reusadas				
% Nuevas Reusadas				
Defectos de Pruebas/KLOC				
DefectosTotales/KLOC				
Rendimiento				
_____	_____	_____		

Tamaño Prog (LOC)	**Plan**	**Real**	**A la fecha**	
Base (B)	*Medida*		*Medida*	
Eliminadas (E)	Estimada		Contada	
Modificadas (M)	Estimada		Contada	
Adicionadas (A)	N-M		T-B+D-R	
Reusadas ®	Estimada	Contada		
Total Nuevas-Modific (N)	Estimada	A+M		
Total LOC (T)	N+B-M-D+R	Medida		
Total Nuevas Reusadas				
LOC Objeto Estimado (E)				

Tiempo en fase (min.)	**Plan**	**Real**	**A la fecha**	**% A la fecha**
Planificación				
Diseño de Alto Nivel				
Revisión del Diseño de Alto Nivel				

Defectos inyectados	**Plan**	**Real**	**A la fecha**	**% A la fecha**
Planificación				
Diseño de Alto nivel				
Revisión del Diseño de Alto Nivel				
Diseño Detallado				

Revisión del Diseño detallado				
Codificación				
Revisión del Código				
Codificación				
Compilación				
Prueba				
Total Desarrollo				

Defectos eliminados	**Plan**	**Real**	**A la fecha**	**% A la fecha**
Planificación				
Diseño de Alto nivel				
Revisión del Diseño de Alto Nivel				
Diseño Detallado				
Revisión del Diseño detallado				
Codificación				
Revisión del Código				
Compilación				
Prueba				
Total Desarrollo				

Anexo8: Plantilla de Estimación de Tamaño.

Estudiante_____ Fecha_____
Programa_____ # de Programa_____
Instructor_____ Lenguaje _____

ESTIMADAS				
Partes Base	Base	Eliminadas	Modificadas	Adicionadas
TOTAL	B	D	M	BA (Adiciones a la Base)

ACTUAL				
Partes Base	Base	Eliminadas	Modificadas	Adicionadas
TOTAL	B	D	M	BA (Adiciones a la Base)

ESTIMADO						ACTUAL
Partes Adicionadas	Tipo	Elementos	Tamaño del Objeto	Tamaño	Tamaño	Elementos
TOTAL de partes adicionadas.	PA (Partes Adicionadas)					

ESTIMADO	ACTUAL	
Partes Reusadas	Tamaño Estimado	Tamaño Actual
TOTAL de partes reusadas	R _____	

Anexo9: Hoja de Cálculo de PROBE

		Tamaño	Tiempo
Tamaño	**Tiempo**		
Partes Adicionadas **(A)**	A=BA + PA		
Tamaño de proxy estimado **(E)**	E= BA +PA + M		
Determinar alternativas de PROBE	A,B, C o D		
Correlación	R^2		
Parámetros de Regresión	B0 en Tamaño y Tiempo		
Parámetros de Regresión	B1 en Tamaño y Tiempo		
Tamaño modificado y adicionado planeado**(P)**	P= B0 tamaño+ B1 tamaño * E		
Tiempo de desarrollo total estimado	Tiempo= B0 tiempo + B1 tiempo * E		
Tamaño total estimado **(T)**	T= P+ B – D – M + R		
Total estimado de nuevas reusables	Suma de todos los elementos que tienen asterisco.		
Rango de Predicción	Rango		
Intervalo superior de predicción	UPI= P + RANGO		
Intervalo inferior de predicción	LPI= P - RANGO		
Por ciento de intervalo de predicción			

Anexo10: Formulario Plan de Tarea de PSP1.1

Estudiante: _____ Fecha. _____
Profesor: _____ Programa: _____

Tarea			Plan		Real				
#	Nombre	Horas	Valor Planeado	Horas Acumuladas	Valor Planeado Acumulado	Fecha (Lunes)	Fecha	Valor Ganado	Valor Ganado Acumulado
Totales									

Anexos11: Entrevista a los Especialistas.

1. El PSP debe ser aplicado a los proyectos de la UCI. ¿Por qué?
2. ¿Qué representa para un equipo de desarrollo la aplicación de PSP?
3. ¿Cómo debería aplicarse el PSP a los proyectos informáticos?
4. ¿Es importante la utilización del PSP por los ingenieros informáticos?
5. Cuales son los principales problemas que impiden la aplicación del PSP en la UCI?

Los profesores que se le realizo la entrevista fueron la Ing. Enrique Pérez y Alfonso Chaleco, en un 100% coinciden en la necesidad de la utilización del PSP por los proyectos de la UCI y los ingenieros informáticos en general. Los principales problemas en la aplicación del PSP que plantearon los entrevistados son; el poco conocimiento respecto al tema por parte de los integrantes de los equipos de desarrollo de los proyectos en la UCI. Los entrevistados plantean que la aplicación del PSP es una necesidad de cada ingeniero informático, en busca de la eficiencia personal.

Anexos12: Entrevista a los integrantes del polo productivo de video y sonido digital de la facultad 9.

1. ¿Como es distribuido el trabajo en los proyectos del polo productivo de video y sonido digital de la facultad 9?
2. Las actividades dentro de los proyectos son distribuidas por roles.
3. ¿Como organizan el trabajo de cada integrante del equipo de desarrollo?
4. ¿Como controlan el cumplimiento de las actividades orientadas a los integrantes de los equipo de desarrollo del polo productivo de video y sonido digital?
5. ¿Cuales son las planillas orientadas por el grupo calidad UCI para el desarrollo de los proyectos?
6. ¿Qué representa para un equipo de desarrollo la aplicación de PSP?
7. ¿Cómo debería aplicarse el PSP a los proyectos informáticos?
8. ¿Es importante la utilización del PSP por los ingenieros informáticos?
9. El PSP debe ser aplicado a los proyectos del polo productivo de video y sonido digital de la Facultad 9. ¿Por qué?
10. ¿Cuales son los principales problemas que impiden la aplicación del PSP en los proyectos del polo productivo de video y sonido digital de la Facultad 9?

Los entrevistados fueron:

Bernardo Rey Almaguer: Diseñador.
Jorge Daniel Olivares: Analistas del sistema.
José Andrés Hernández: Analista del sistema.
Ing. Yunior Montaner Hernández: Jefe de proyecto.

Después de realizar la entrevista en el polo productivo de video y sonido digital a un jefe de proyecto y a 3 integrantes de este polo plantearon:
Proyecto UCI TV se encarga de realizar prestaciones de servicios. Este proyecto no tiene un tiempo límite, las tareas son orientadas a cualquier persona teniendo en cuenta su carga de trabajo, no documentan ninguna tarea realizada. Realizan una planificación irreal. No tienen ninguna estrategia aplicada de PSP.

Proyecto de Monitoreo de Radio y TV se encarga de la documentación para la aplicación de un proyecto ya realizado en Venezuela.

Proyecto MENPET se encarga de la realización de un software sobre como el utilizado en UCI "Señal 3" para el ministerio de energía y petróleos de Venezuela en software libre, este registra las actividades y el tiempo de culminación pero sus integrantes no aplican PSP. Tienen poco conocimiento sobre el tema. Utilizan como herramienta el DotProject para tener un control de las actividades, así como la asignación de recursos materiales y humanos a las mismas. El proyecto tiene un formulario de gestión de riesgos que es utilizado en el proceso de mejora en el desarrollo de software. Se aplica pero sin tener conocimiento de PSP y sus ventajas en los proyectos.

GLOSARIO DE TÉRMINOS

Calidad: Calidad de software. Satisfacción de las necesidades de los usuarios.

Camino Independiente: es aquel que introduce por lo menos una sentencia de procesamiento (o condición) que no estaba considerada en el conjunto de caminos independientes calculados hasta ese momento.

Caso de prueba: Conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito. También se puede referir a la documentación en la que se describen las entradas, condiciones y salidas de un caso de prueba.

Defecto: Un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa.

Artefacto: Pieza de información tangible que es creada, modificada y usada por los trabajadores al realizar las actividades; representa un área de responsabilidad y es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento.

Desarrollador: Persona que trabaja directamente en el desarrollo de un proyecto de software, dígase: analista, programador, diseñador, etc.

Fase: Son los pasos en que se descomponen las metodologías. Cada fase puede o no estar subordinada a otra fase, pudiendo existir entre ellas relaciones de dependencia de inicio, fin y paralelismo.

Rol: Papel, cometido o función, que tiene o desempeña un actor.