Ingeniería en Desarrollo de software
Semestre 5

Programa de la asignatura:
Métricas de desarrollo de software (PSP)

Unidad 2.
Planeación: Introducción, medición y estimación

Clave:
15143529

Universidad Abierta y a Distancia de México

Índice

UNIDAD 2. PLANEACIÓN: INTRODUCCIÓN, MEDICIÓN Y ESTIMACIÓN 3
Presentación de la unidad 3
Propósito 4
Competencia específica 4
2.1. Introducción a la planeación 4
2.1.1. ¿Qué es un plan? 5
2.1.2. ¿Por qué hacer planes? 5
2.1.3. Contenido del plan de un *software* 6
2.1.4. Planeando un proyecto de *software* 8
2.1.5. Producir un plan de calidad 9
2.1.6. Etapas de la planeación 10
2.2. Medición del tamaño del *software* 11
2.2.1. Medición del tamaño 12
2.2.2. Establecer un conteo estándar 14
2.2.3. Contadores de LOC y tipos 15
2.2.4. Consideraciones del re-uso 16
2.2.5. Conteo de líneas de código 16
2.2.6. Calcular la productividad 18
2.2.7. PSP 0.1 19
2.3. Estimación del tamaño del *software* 19
2.3.1. Contexto 20
2.3.2. Métodos de estimación 20
2.3.3. Proxy 21
2.3.4. PROBE 22
2.3.5. PSP 1 23
Para saber más 24
Fuentes de consulta 24

Unidad 2. Planeación: Introducción, medición y estimación

En esta unidad se tratarán los temas relacionados con la medición del tamaño de un proyecto de *software*. A su vez, esta medición proporcionará una base inicial para poder estimar el tamaño nuevos proyectos con la finalidad de poder realizar planes lo más cercano a la realidad.

En la mayoría de los proyectos, existen recursos limitados (dinero, personas, equipos, tecnología). Este factor conlleva a que los proyectos tengan que planearse y gestionarse de la forma más efectiva y asertiva posible, con la finalidad de que el costo total del proyecto no exceda la cantidad de recursos planeados que son destinados a ese proyecto. Sin embargo, para poder planear algo, primero se debe ser capaz de estimar cuánto tiempo, costo y personal que se requiere dentro del proyecto. Para poder estimar, se requiere medir el proyecto, la medición es una tarea fundamental en la mayoría de las disciplinas donde se emprenden proyectos. Dependiendo del contexto, se utiliza una métrica o forma de medir. Por ejemplo, para la construcción, es común medir en áreas lo que se va a construir utilizando alguna unidad como: m^2. De igual forma, para estimar el tamaño un nuevo proyecto de *software* primero se debe tener una forma o métrica de medición enfocada en la medición de un programa.

Presentación de la unidad

En esta unidad conocerás lo importante que es realizar una planeación previa al desarrollo de un nuevo proyecto de *software*.

Aprenderás por qué es necesario realizar planes, así como el contenido de una planeación. Entenderás por qué mientras más detallados sean tus planes, más precisas serán tus estimaciones de tiempo y costos antes de comenzar el desarrollo de un nuevo proyecto. Así mismo, tendrás elementos y bases para gestionar cambios en los requerimientos planeados, para un nuevo proyecto de *software* y cómo deben ser manejados dichos cambios antes de su implementación en el proyecto.

Finalmente, conocerás algunas técnicas para estimar el tamaño de un proyecto de *software* y como éste, está relacionado con el tiempo que tomará desarrollarlo.

Propósito

En esta unidad lograrás:

- Comprender lo que es un plan para un proyecto de *software* y lo que debe contener.
- Conocerás algunas técnicas para estimar el tamaño de un proyecto de *software*.
- Conocerás algunas técnicas para estimar el tiempo que tomará desarrollar un nuevo proyecto de *software*.

Competencia específica

Aplicar la medición y estimación para planear un programa, tomando en cuenta el plan, sus etapas, criterios y estándares de medición, así como diferentes métodos de estimación.

2.1. Introducción a la planeación

En esta unidad se analizará la importancia de planear un proyecto de *software* antes de realizar cualquier actividad relacionada con el desarrollo del mismo. Así mismo, se analizará de forma detallada, cómo realizar la planeación de un nuevo proyecto de *software* y como, un plan bien realizado, brinda una mayor confiabilidad para el éxito en la realización de un nuevo proyecto de *software*.

Por naturaleza, los seres humanos, al emprender nuevos proyectos necesitamos tener una guía o conocer los procesos, acciones o medidas que deben ser seguidas para lograr realizar ese proyecto hasta su finalización. A lo largo de toda su vida, el ser humano emprende proyectos: aprobar una materia de la escuela, construir un prototipo que ayuda a resolver un problema, concursar y ganar una competencia de deportes, etc.

Comúnmente, aquellos proyectos en los que mejores resultados se obtienen son aquellos para los cuáles teníamos una mejor guía de cómo realizarlos. En otras palabras, teníamos un plan lo suficientemente sustentado para lograr resultados efectivos al término del proyecto.

En la ingeniería de *software*, la planeación de los proyectos enfocados en el desarrollo de nuevas tecnologías de información es una de las etapas más importantes para el éxito de dicho proceso, pues como veremos más adelante, la calidad del producto final está dada por la calidad en cada uno de los procesos llevados a cabo. En este sentido, la planeación es el primer proceso llevado a cabo al desarrollar un nuevo producto de *software*.

Una vez que se ha brindado un panorama general sobre la planeación de proyectos incluyendo proyectos de *software*, es necesario conocer que es realmente un plan. Esto será explicado en el siguiente tópico.

2.1.1. ¿Qué es un plan?

En este tópico se explicará y comprenderá lo que es un plan y como está estrechamente relacionado con la culminación exitosa de un nuevo proyecto.
Un plan no es más que una guía que nos muestra los pasos, procedimientos o medidas a seguir para lograr un objetivo o meta determinada. La mayoría de las veces, el éxito que se tiene en un proyecto está en función de la calidad del plan que se realiza antes de comenzar el desarrollo de ese proyecto. A su vez, la calidad de un plan está dada por el sustento y validez de las bases sobre cuáles se apoya dicho plan.
Para comprender lo anterior, veamos el siguiente ejemplo:
Pedro se entera, a mitad de su periodo escolar, que existe un concurso de proyectos finales para la materia de desarrollo de aplicaciones. Al equipo o desarrollador del proyecto ganador se le otorgará un premio que consiste en un par de equipos de cómputo portátiles y dispositivos de cómputo móviles de última generación.
Al pensar Pedro en su proyecto, se da cuenta que anda un poco retrasado en el avance del mismo. Sin embargo, cree que no puede dejar pasar esa oportunidad ya que con algunos ajustes, su proyecto tendría altas posibilidades de ser el mejor.
¿Qué podría hacer Pedro para poder tener las mayores posibilidades de ganar el concurso?
Sin otro conocimiento acerca del problema, lo primero que podríamos pensar en realizar es el conocer el estado actual del proyecto de Pedro, analizar y delimitar un conjunto de mejoras de acuerdo al conocimiento general que se tiene de los otros proyectos. Después habría que definir un conjunto de actividades por realizar así como un tiempo estimado para concluir cada actividad con la finalidad de lograr obtener un proyecto con altas posibilidades de ser exitoso.
Una vez que se ha entendido y comprendido lo que es un plan así como su utilización en el desarrollo de nuevos proyectos, se analizarán algunas situaciones que remarcan la necesidad de realizar planes previos al desarrollo de nuevos proyectos.

2.1.2. ¿Por qué hacer planes?

Como se vio anteriormente, la planeación es una etapa que tiene como finalidad aumentar las posibilidades de éxito que tendrá un proyecto. Sin embargo, existen factores subyacentes al éxito del proyecto los cuáles describiremos en este tópico y que sustentan la importancia que tiene el realizar planes al emprender nuevos proyectos que involucran el desarrollo de productos de software.

Al proponer una solución a un problema relativo a la falta de información vigente y confiable, generalmente se involucra en dicha solución, un sistema de información el cual, la mayoría de las veces, no existe a la medida de la persona o empresa que tiene el problema descrito anteriormente. Esto conlleva a desarrollar un *software* que satisfaga las necesidades de información de la empresa o persona en particular.

Al desarrollar productos de *software* para un cliente -que puede ser algún directivo para la empresa para la cual se trabaja o bien un cliente externo-; en el caso de tener una empresa que se dedica al desarrollo de proyectos de *software*, es común que se realicen dos cuestionamientos:

- ¿Cuánto tiempo tomará desarrollar el sistema?
- ¿Cuánto va a costar?

Si el cliente cuenta con varias opciones para el desarrollo de su programa, quizás opte por el proyecto que consuma menos tiempo en realizarse y que sea más barato. Sin embargo, esta elección no siempre es la más adecuada como veremos en seguida.

Cuando se responde a un cliente las dos preguntas anteriores, ¿sobre qué bases es respondido el tiempo que tomará desarrollar el sistema?

Cuando no se lleva una metodología que involucre una sólida planeación en el desarrollo de nuevos proyectos, se suele responder en base a la experiencia propia. Sin embargo, ¿qué tan acertada es una respuesta de este tipo? Generalmente, existe poca asertividad y generalmente se planea menos tiempo del que realmente se requiere para desarrollar el proyecto. Un factor adicional frecuente, que lleva a realizar una mala estimación, es la competencia para ser elegidos por el cliente para el desarrollo de dicho proyecto. Esto conlleva a subestimar el esfuerzo real que requiere el proyecto y se propone un tiempo relativamente corto para la realización del mismo. Al final, el proyecto podría tardarse 2 ó 3 veces más el tiempo estimado. Por eso, se mencionó anteriormente que la elección del tiempo más corto no siempre es la más adecuada. Sin dejar de lado, que el costo de un proyecto también está relacionado con el tiempo en que se desarrolla.

Con base en la información anterior, podemos deducir lo importante que es el realizar una planeación de los nuevos proyectos de desarrollo de software para lograr las mayores posibilidades de éxito. (Zapata, J., García, J., Cerrada, J. 2001. Pág. 43-55).

2.1.3. Contenido del plan de un *software*

Para que un plan, previo al desarrollo de un nuevo proyecto de *software*, sea efectivo, debe contener ciertos elementos relativos a las tareas que deberán realizarse, el tamaño

del proyecto, la forma de medir el status del proyecto en cualquier momento, etc. Todo esto será analizado y discutido en este tópico.

El contenido de un plan depende de las necesidades de las personas que utilizarán dicho plan y de lo que requieran hacer con él. En PSP la planeación involucra a dos tipos de persona: el desarrollador y sus clientes.

Una planeación en PSP involucra definir y conocer los siguientes 4 factores:

1. Tamaño del producto. ¿Qué tan grande es el proyecto?
2. Definición de las tareas que tienen que realizarse y como se realizarán.
3. Estatus del trabajo que se está realizando. Esto significa saber en qué etapa estamos, qué falta por hacer. Esto nos permite responder una pregunta crítica muy importante: ¿Se terminará el producto en el tiempo estimado con los costos planeados?
4. Evaluación. Este factor se refiere a la evaluación de la planeación realizada y responde a las preguntas: ¿Qué tan efectiva fue la planeación? ¿Se cometieron errores obvios de detectar? ¿Qué errores se pueden evitar en el futuro? ¿Qué se requiere ajustar o modificar para realizar una mejor planeación?

   De lo anteriormente expuesto, Humphrey, W. (1995), menciona que el factor 4 es de vital importancia pues es la clave para mejorar la efectividad de las planeaciones que se realizarán posteriormente para nuevos proyectos.

Es necesario mencionar que al utilizar PSP, todo lo que realizamos es un trabajo completamente individual. En este contexto, el contenido de las planeaciones debe permitir saber:

1. ¿Qué se tiene que entregar?, ¿Cuándo?, ¿Cuánto costará?
2. El proyecto que se planea desarrollar, ¿es realmente lo que el cliente quiere?
3. ¿Existe algún mecanismo que permita medir, monitorear o conocer el avance del proyecto en cualquier momento?

En proyectos pequeños, la administración y control de los aspectos anteriores probablemente no sean complejos; sin embargo, se debe ser consciente de lo siguiente:

- La planeación debe estar basada en tareas bien definidas.
- Cada tarea debe estar claramente definida y debe ser posible medirla.
- El cliente debe conocer el plan y estar de acuerdo con él antes de comenzar cualquier tarea.
- Se debe reportar el avance del proyecto cada determinado tiempo. Este reporte generalmente se debe presentar a los administradores y clientes para que conozcan cómo va evolucionando el desarrollo del mismo.

Siempre hay que tener en cuenta que: cuando se planea el trabajo personal, el objetivo es calcular el tiempo que tardará en desarrollarse un determinado proyecto, el costo que tendrá y qué fechas se tienen planeadas para terminar cada una de las tareas a realizar (Humphrey, W. 1995. pp. 57-68).

En el siguiente tópico se comprenderá como comenzar la planeación de un nuevo proyecto así como los pasos necesarios para poder obtener un plan lo más acertado posible conforme a la realidad.

2.1.4. Planeando un proyecto de *software*

En este tópico se analizarán los pasos necesarios para realizar el plan de desarrollo de nuevos proyecto de *software*. Se aprenderá en primera instancia la secuencia ordenada de actividades a realizar, para poder obtener un plan de calidad previo al desarrollo de un proyecto de *software*.

Al planear un proyecto de *software*, se debe tomar en cuenta lo siguiente:

1. Se debe tener claramente definido lo que ha de realizarse así como entender perfectamente lo que se busca hacer. Durante el desarrollo del proyecto es frecuente que existan cambios en los requerimientos. Sin embargo, es necesario planear y definir el mayor número de dichos requerimientos tanto como sea posible.
2. Dividir las tareas de desarrollo del proyecto que tomarán varios días en varias tareas más pequeñas que podamos medir y estimar de forma separada. Es importante mencionar que: a mayor detalle en la especificación de las tareas que han de desarrollarse, la planeación será más precisa.
3. Cuando se realiza la estimación de tareas, se deben comparar con otras tareas similares que se hayan realizado en el pasado para conocer el tiempo que se dedicó a esas tareas pasadas y estimar el tiempo que tomará realizar la tarea planeada.
4. Se debe documentar la estimación y al término del proyecto, se deben comparar los datos de la estimación contra los datos reales. Esto nos permite conocer de forma clara y consciente el por qué hubo diferencias entre el tiempo planeado y el real. Este nuevo conocimiento adquirido tendrá una vital importancia al estimar y planear nuevos proyectos de *software*.
5. Si existen cambios en los requerimientos, se debe reajustar la planeación y antes de comenzar la implementación de cualquier cambio en los requerimientos, se debe informar a los jefes inmediatos así como al cliente para concientizarlos de lo que implica en tiempos y costos la implementación de dichos cambios de requerimientos y en base a ello, se tome la decisión de implementar o no dichos cambios.

La Figura. Secuencia de pasos a seguir para elaborar un plan de desarrollo de un nuevo proyecto de software (Humphrey, W. 1995. Pág. 65) muestra los pasos que tienen que realizarse para obtener un plan eficiente para el desarrollo de un nuevo proyecto:

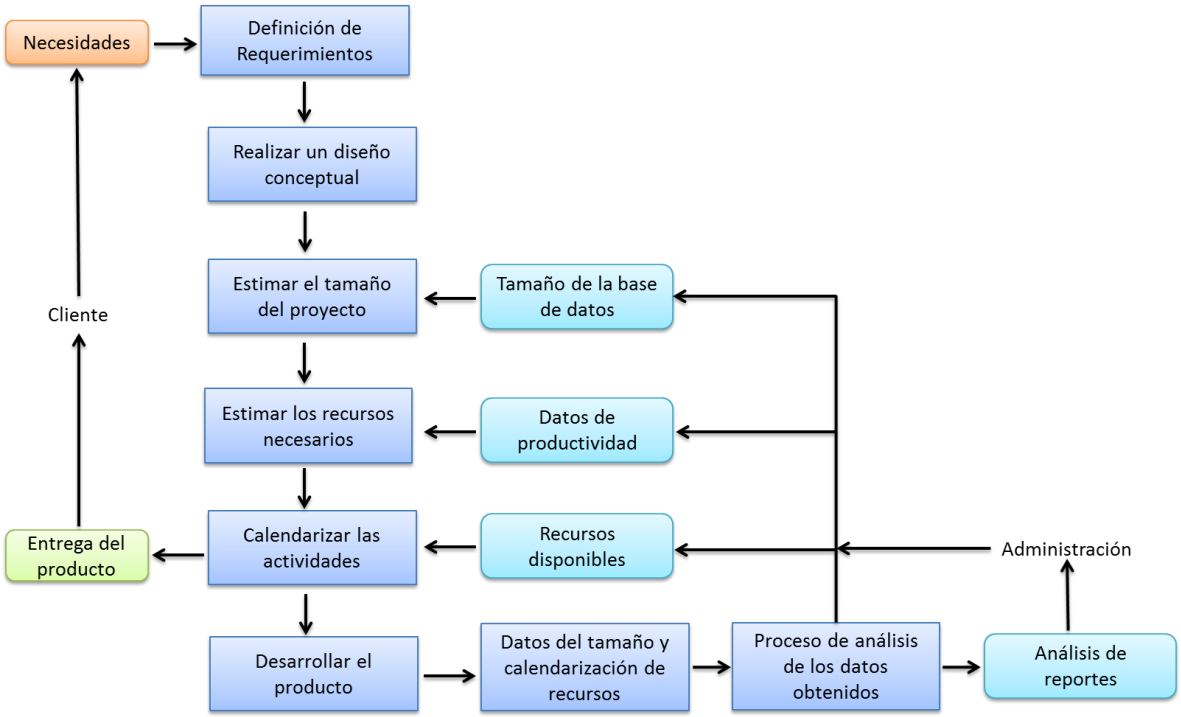

Figura. Secuencia de pasos a seguir para elaborar un plan de desarrollo de un nuevo proyecto de software. (Humphrey, W. 1995. Pág. 65).

Una vez que se conocen los pasos necesarios para realizar la planeación de un proyecto de *software*, es necesario conocer aquellos aspectos que dotan de la calidad necesaria a una planeación. Mientras más alta sea la calidad de un plan, mayores son las posibilidades de éxito al término del desarrollo del proyecto. (Humphrey, W. 1995. Pág. 57-68).

2.1.5. Producir un plan de calidad

En este tópico se discutirá de forma general como producir un plan con la calidad adecuada para obtener éxito en los proyectos planeados. Eso es importante de considerar pues el éxito de un proyecto está dado en gran parte por la calidad del plan que se tiene para su desarrollo.

El objetivo de planear el trabajo es para obtener un plan preciso que indica qué actividades han de realizarse y en qué tiempo. Sin embargo, para poder obtener esta

información se requiere un estudio profundo que nos permita conocer cómo medir, predecir y mejorar las planeaciones que realizamos.

Cuando planeamos nuevos proyectos de *software* lo hacemos de dos formas: imparcial o balanceado. Se considera que una planeación está balanceada si por ejemplo, de diez proyectos, cinco fueron sobreestimados y cinco fueron subestimados. Un proyecto **sobreestimado** es aquel en el cual se planeó más tiempo para su realización que el tiempo real. En otras palabras, se tardó menos tiempo de lo planeado en realizar el proyecto. Un proyecto **subestimado**, al contrario de uno sobreestimado, es aquel en el que se planeó menos tiempo del que realmente se necesitó para llevarlo a cabo. Una planeación imparcial se da cuando no es balanceada, es decir, el número de proyectos sobreestimados no es igual al número de proyectos subestimados. De acuerdo a la experiencia de Watts Humphrey, el autor del PSP, una planeación balanceada permite realizar un mejor ajuste de la estimación de tiempos para nuevos proyectos, pues al haber un equilibrio entre los proyectos sobreestimados y los subestimados, se puede fácilmente obtener un tiempo promedio a través del método PROBE, el cual veremos más adelante.

Para terminar de comprender el proceso de la planeación de un proyecto de software, se analizarán las etapas involucradas en la planeación. (Humphrey, W. 2005. Pág. 65).

2.1.6. Etapas de la planeación

El proceso de planear un proyecto de *software* está conformado por varias etapas que serán descritas en este tópico. Es importante que se sigan cada una de estas etapas sin omitir ninguna con la finalidad de que la planeación final del proyecto sea lo más real posible.

Planear un proyecto de *software* implica las siguientes etapas:

1. Conocer las necesidades del cliente. Este punto inicial es el más importante, pues de este, se deriva prácticamente todo el contenido y alcance del proyecto a desarrollar. En esta etapa se debe decidir si las necesidades del cliente realmente involucran el desarrollo de un proyecto de software o por el contrario, existen otras herramientas previamente desarrolladas que pueden ayudar a resolver dichas necesidades.
2. Definición de Requerimientos. En esta etapa, una vez que se ha identificado que las necesidades del cliente realmente involucran el desarrollo de un proyecto, se deben identificar dichas necesidades de forma muy particular. Al término de esta etapa lo que se obtiene es un documento con toda la especificación de requerimientos y funcionalidades con las que deberá contar el producto final. Tener claros los requerimientos del proyecto ayuda a

proyectar una primera aproximación sobre qué tan grande puede ser el proyecto.

3. Realizar un Diseño Conceptual. En esta etapa se debe producir un diseño preliminar del proyecto a desarrollar. En esta etapa, se suelen identificar y planear los módulos que conformarán el producto final así como la interacción que tendrán dichos módulos.
4. Estimar el tamaño del proyecto. En esta etapa, se toman datos históricos de otros proyectos, se busca información de módulos previamente desarrollados que tengan el mayor parecido con las especificaciones del nuevo proyecto para poder tener una aproximación lo más certera posible del tamaño del proyecto.
5. Planeación de los recursos. Una vez que se tiene una estimación del tamaño del proyecto, se deben planear los recursos que intervendrán en el desarrollo del proyecto.
6. Desarrollo del Proyecto. Una vez que se tiene un plan claro que indica las fechas de avance y el personal asignado a cada actividad del desarrollo del proyecto, se puede pasar a la etapa de desarrollo. En esta etapa se llevan a cabo todas las actividades involucradas en el desarrollo del proyecto. Por ejemplo: diseño de interfaces, diseño de la arquitectura, diseño e implementación de la base de datos, etc.
7. Generación de nuevos datos. Durante el desarrollo del proyecto, cada actividad debe ser medida para generar datos históricos del proceso de desarrollo.
8. Análisis de Datos. Cuando se concluye el desarrollo del proyecto, es importante analizar los datos generados, pues estos datos jugarán un papel esencial al planear nuevos proyectos y estimar su tamaño.

Ya que se han comprendido las etapas que involucran la planeación de un proyecto de *software*, se debe comprender el aspecto principal: medir el tamaño del proyecto. Esta es quizás la tarea más difícil y crítica en la planeación de un proyecto. Por lo tanto, el siguiente tema estará completamente enfocado en describir el proceso de medición del tamaño de un nuevo proyecto de *software*. Pero, primero realiza la siguiente actividad.

2.2. Medición del tamaño del *software*

Cuando planeamos nuevos proyectos de *software*, inevitablemente tenemos que estimar o tener un conocimiento preliminar sobre el tiempo que nos tomará realizar dicho proyecto. Esta etapa cobra una vital importancia, pues de los datos que se generen depende en gran medida el costo que tendrá el proyecto para el cliente.

Sin embargo, para poder estimar o predecir cuánto tiempo se requiere para desarrollar el proyecto, es necesario conocer o tener una medida del mismo. Y un par de preguntas que surgen son: ¿cómo medimos un proyecto de software? ¿Qué medida es la más precisa?

En este capítulo se analizarán algunas técnicas para medir y estimar el tamaño de nuevos proyectos de *software*, pues a través de los años, el estudio e investigación sobre PSP, por parte de Watts Humphrey, ha recopilado una serie de datos y buenas prácticas que ahorrarán en gran medida la investigación y aprendizaje en este tema. (Zapata, J., García, J., Cerrada, J. 2001. Pág. 58-71).

2.2.1. Medición del tamaño

Para comprender las métricas para el tamaño de un producto de *software* conviene responder la siguiente pregunta: ¿por qué medir el tamaño?

Como se mencionó en la introducción a este capítulo, conocer el tamaño de algo, en este caso, de un proyecto de *software*, es fundamental para poder estimar cuánto tiempo tomará desarrollar dicho proyecto. Una vez que se conoce la importancia de medir el tamaño, la siguiente pregunta que se hace es: ¿cómo se puede medir el tamaño de un proyecto de *software*?

Responder a la pregunta anterior ha sido uno de los retos más controversiales que ha tenido la ingeniería de *software* desde que los proyectos tuvieron la necesidad de planearse, en la década de 1970. A lo largo de ese tiempo, los lenguajes y paradigmas de la programación han estado en constante cambio. Este factor ha sido determinante para hacer aún más complejo el obtener una métrica o forma de medir el tamaño de un proyecto de *software* de forma precisa en un cien por ciento. Por lo tanto, no existe una métrica que sea exacta por completo, sino por el contrario, todas las métricas que se han utilizado hasta ahora son aproximaciones, sin embargo, si existen diferencias considerables en la exactitud y eficacia de cada una de ellas.

Para que una métrica, en cualquier contexto y escenario, sea útil, debe cumplir lo siguiente (Humphrey, W. 1995.):

- Debe servir para un propósito específico. Por ejemplo, la unidad de medida litro sirve específicamente para medir volúmenes. Querer medir el peso con una métrica o instrumento que mide volumen sería muy difícil a la vez que los resultados serían inadecuados y poco útiles. En el contexto de la ingeniería de *software*, medir el tamaño de un producto debe servir a un propósito muy específico: conocer de forma puntual que tan grande es un proyecto de *software*.

- Debe estar bien definida. Esto significa que las unidades que se utilicen para medir deben estar claramente definidas de tal forma que no presenten ambigüedades. Por ejemplo, la unidad de medida metro está bien definida y cualquier cinta métrica o regla que utilicemos para medir por ejemplo, el largo de una mesa, nos dará la misma medida para esa misma mesa. Aplicando este ejemplo en la ingeniería de *software*, se requiere de una unidad de medida lo más exacta posible y que no tenga ambigüedades cada vez que se aplique en la medición de proyectos de *software*.
- Debe ser adecuadamente administrada. Generalmente, conocer la medida de algo tiene implicaciones para tomar una decisión o hacer algo con esa medida conocida. Por ejemplo, conocer el área que ocupa una mesa es podría ser de utilidad para escoger el mantel que mejor la cubra. De igual forma, las métricas de desarrollo de *software* deben aportar utilidad para administrar y controlar el proyecto.
- Debe ser adecuadamente utilizada. Los datos obtenidos con una métrica determinada deben ser adecuadamente utilizados. Por ejemplo, medir el área de un cuarto sería poco útil si buscamos elegir el color de piso que mejor combine con el diseño de dicho cuarto. En todo caso, es mejor utilizar otra unidad de medida, como por ejemplo, el color más representativo o la luminosidad del cuarto. En el contexto de la ingeniería de *software*, las métricas para medir el tamaño de un proyecto de *software* deben utilizarse para tal fin. Sería ilógico utilizar una métrica de tamaño de *software* para elegir el tamaño del monitor ideal para la computadora donde se instalará el sistema a desarrollar en ese proyecto.

Por lo tanto, medimos para:

- Entender y saber administrar los cambios
- Planear el futuro
- Comparar un producto, organización o proceso con otro
- Determinar si estamos apegados a estándares
- Contar con bases para controlar

Se tiene que ser consciente qué las medidas solo producen números. Para que una medida sea de utilidad, debe:

- Estar relacionada con los objetivos que se plantean alcanzar. Si lo que medimos no está relacionado con los objetivos que deseamos lograr, no hay razón para invertir esfuerzos en medir aquello.
- Ser adecuadamente interpretada. Una vez que obtenemos una medida, debemos entender lo que esa medida significa y lo que no. Por ejemplo, conocer el número de tablas que tendrá la base de datos no significa conocer cuántas de esas tablas serán catálogos que impliquen su propia interface para altas, bajas y cambios.

- Permitir tomar acciones apropiadas. El conocer la medida de algo, nos debe ser utilidad para tomar acciones apropiadas ya sean de carácter preventivo, correctivo o las acciones de curso normal para lograr los objetivos planteados al inicio del desarrollo del proyecto así como para efectuar una mejor planeación de nuevos proyectos.

A lo largo del tiempo, se han propuesto varias técnicas para medir el tamaño de un proyecto de *software*. A continuación se mencionan algunas de ellas:

- Número de archivos que tendrá el programa
- Número de módulos que tendrá el programa
- Número de tablas en la base de datos
- Número de pantallas que tendrá la aplicación
- Puntos de función
- Líneas de código

De las técnicas mencionadas anteriormente, la que más efectiva ha sido en el PSP es la métrica por líneas de código. Como su nombre lo indica, esta técnica se basa en contar las líneas de código que tiene o tendrá el proyecto de *software*. Esto será tratado con mayor detalle en los siguientes tópicos. (Humphrey, W. 1995. Pág. 69-90).

2.2.2. Establecer un conteo estándar

Al elegir una métrica para calcular el tamaño o la medida de algo, se debe responder lo siguiente: ¿si dos personas miden la misma cosa, obtienen el mismo resultado?

Responder a la pregunta anterior es fundamental, pues de ello depende la confiabilidad en la precisión de la métrica que estamos aplicando para medir algo. Si cada vez que medimos la misma cosa en las mismas condiciones obtenemos resultados distintos, inevitablemente deberemos tomar algún otro criterio o métrica más estable que nos permita obtener confiabilidad en las medidas que nos arroja.
Para poder homogenizar una métrica y aumentar su confiabilidad, es necesario realizar un estándar de conteo.

Un estándar de conteo es un documento que expresa de forma clara y sin ambigüedad qué es algo contable y qué no lo es, de acuerdo con una métrica determinada. En el caso de la métrica de líneas de código, en dicho estándar se debe definir lo que sí debe ser contado como una línea de código y lo que no. Por ejemplo, los comentarios en el código, ¿deben ser considerados como líneas de código o no? Decidir si las líneas de un comentario en el código fuente de un programa o módulo serán contadas depende de

quien realiza el estándar de conteo de líneas de código y del criterio que aplique para tomar esta decisión.

2.2.3. Contadores de LOC y tipos

Llevar un conteo de líneas de código de forma manual es muy difícil e impreciso. Por tal motivo, existen distintas herramientas que ayudan a los ingenieros de *software* con esta tarea. Sin embargo, se tiene que considerar que los contadores de líneas de código automatizados solamente trabajarán para las características que se han definido en dicho contador. En otras palabras, al elegir un programa contador de líneas de código, debemos asegurarnos que dicho programa se apega al estándar de conteo definido con anterioridad. De otra forma, sería impreciso y poco útil el utilizar esa métrica si no se apega al estándar de conteo definido previamente. Una práctica común para salvaguardar este escenario es el de tomar con estándar de conteo de líneas el propio estándar que ya trae por definición el programa contador de líneas de código elegido (Humphrey, W. 2005. Pág. 35-55).

De las diversas soluciones que existen, algunas son de pago y otras gratuitas. La diferencia entre unos y otros son básicamente las prestaciones que brindan así como los lenguajes y plataformas que soportan.

A continuación se mencionan algunos contadores de líneas de código y sus características:

Programa	Gratuito	Página de la compañía	Plataformas y lenguajes soportados
Code Counter Pro	No	http://www.geronesoft.com/	Varios
Microsoft LOC Counter	Si	http://archive.msdn.microsoft.com/LOCCounter	Lenguajes de la plataforma .Net 4.0
Code Line Counter Pro	No	http://www.codelinecounter.com/	Varios. Se compra por separado para cada lenguaje distinto.

Tabla. Software para contar líneas de código.

No importa cuál sea la herramienta que utilices, si tu equipo no tiene correctamente definidos los estándares de conteo habrá discrepancia en la forma de contar de cada programador. Por ejemplo el código de re-uso no debería contarse a menos que hagas mejoras. En el siguiente tema se explica en que consiste este tipo de código y su utilidad para las empresas de desarrollo.

2.2.4. Consideraciones del re-uso

Generalmente, al desarrollar nuevos proyectos, se intenta reducir el esfuerzo requerido en el desarrollo. Una de las prácticas comunes es buscar código existente de proyectos previos para reutilizarlo en los nuevos evitando invertir esfuerzos para obtener un código que hace la misma función de uno que previamente ya se había desarrollado. También, es común en pensar en modificar un código existente y simplemente adaptarlo antes que crear uno completamente. Sin embargo, debe evaluarse si el esfuerzo requerido para modificar un código existente realmente es menor que el necesario para crear ese código fuente desde el principio.

En el mejor de los casos, lo más conveniente es utilizar librerías y módulos completos previamente desarrollados evitando de esta forma el mayor esfuerzo posible. Sin embargo, poder llegar a este tipo de buenas prácticas implica que para cada proyecto desarrollado deben crearse módulos genéricos con la menor dependencia posible del proyecto. Lo cual, es una tarea que se da por la experiencia obtenida con el progresivo desarrollo de proyectos. (Humphrey, W. 1995. Pág. 84).

2.2.5. Conteo de líneas de código

En el capítulo 2.2.3. se analizó de forma introductoria lo que significa el conteo de líneas de código de un programa y cómo esta métrica puede ser utilizada en PSP para obtener el tamaño de un proyecto de software basado en el número de líneas de código del mismo.

En este apartado analizaremos de forma detallada cómo se realiza un conteo de líneas de código de acuerdo al PSP. Aun cuando la métrica basada en líneas de código, en adelante denominadas LOC's, pareciera ser sencilla, en la práctica resulta difícil llevar un rastreo de las mismas si no es propiamente definido y utilizado el concepto de línea de código. De acuerdo con la metodología de PSP, si al iniciar la realización de un nuevo proyecto de *software* iniciamos con 5000 líneas de código que tomamos de otro programa y nosotros escribimos 500 líneas de código adicionales, el resultado debería ser 5500 líneas de código. Por lo tanto, si nosotros sólo escribimos 500 líneas de código el resultado debería ser un programa de 500 líneas de código. Aun cuando este ejemplo parece obvio, en la realidad, al crecer y haber cambios en los proyectos de *software*, obtenemos un resultado diferente. A esto se debe la necesidad de entender y utilizar apropiadamente el tamaño de un programa basado en sus líneas de código.

Cuando desarrollamos un nuevo programa, generalmente intervienen 4 tipos distintos de líneas de código (Humphrey, W. 2005. Pág.40-47):

- **Líneas Base**. Corresponden a las líneas que se piensan reutilizar en la construcción de un nuevo programa. En otras palabras, son líneas de código de otro programa que se creen que pueden ser útiles en alguna parte del nuevo programa.
- **Líneas Agregadas**. Corresponden a las líneas nuevas que el desarrollador escribe en su programa.
- **Líneas Modificadas**. Corresponden a aquellas líneas base que requieren ser modificadas para el buen funcionamiento del nuevo programa.
- **Líneas Borradas**. Corresponden a aquellas líneas base que no son de utilidad alguna en el nuevo programa y requieren ser eliminadas.
- **Líneas Reutilizadas**. Corresponden a las líneas base que no fueron modificadas ni eliminadas, es decir, a aquellas líneas que no requieren modificación alguna para el buen funcionamiento del nuevo programa.
- **Líneas Agregadas y Modificadas**. En muchos proyectos de *software*, el esfuerzo requerido para modificar una línea de código ya existente y el de crear una nueva es muy similar. Por lo tanto, suelen sumarse y obtener una medida de esfuerzo considerando ambos tipos de línea pero sin dejar de identificar cada una de ellas.
- **Líneas Nuevas Reutilizables**. Los nuevos paradigmas de programación, tales como la programación orientada a objetos, permiten construir las aplicaciones basadas en módulos y librerías con funcionalidades muy específicas. De tal forma que se piensa en que un determinado módulo o librería pueda servir sin ser modificado, en el mejor de los casos, en nuevos proyectos. A las líneas de código pertenecientes a estas librerías o módulos se les conoce como líneas nuevas reutilizables. Cabe mencionar que no necesariamente tienen que ser líneas pertenecientes a una librería o módulo, pudiendo ser también líneas correspondientes a una clase, un método o procedimiento dependiendo del lenguaje y plataforma de programación en el cual se está desarrollando el proyecto.
- **Líneas Totales**. Se refiere al número total de líneas que tiene el programa al final de su desarrollo.

La razón por la cual es necesario llevar un rastreo de las líneas base, agregadas, modificadas y borradas se debe a que por ejemplo, si comenzamos el desarrollo del programa en la versión 0, contamos 400 líneas de código base y después agregamos 200, esperaríamos tener una primer versión con 600 líneas de código. Sin embargo, al contar las líneas de código de la versión 1 el tamaño total es de 550 líneas de código. ¿Qué sucedió con las 50 líneas de código faltantes?

2.2.6. Calcular la productividad

La productividad es un factor que relaciona el tiempo requerido para una tarea determinada y el producto derivado de realizar dicha tarea. Este factor es utilizado en muchas de las áreas productivas dentro de las actividades humanas y es un factor clave para determinar el tiempo total que requerirá concluir una tarea específica por una persona en particular, de la cual se conoce su productividad. En este apartado se comprenderá el concepto de la productividad relacionada con el desarrollo de productos de *software* y como se calcula dicha productividad.

Como se ha visto anteriormente, el desarrollo de un producto de *software* involucra una planeación previa al desarrollo del proyecto. Por otra parte, planear el desarrollo de dicho proyecto involucra conocer el tamaño del proyecto. A su vez, conocer el tamaño del proyecto es un dato vital para poder estimar y planear el tiempo en que ha de ser desarrollado. Sin embargo, para calcular el tiempo requerido para el desarrollo del proyecto, es necesario un dato adicional: ¿con qué rapidez se desarrollan las tareas por parte de cada integrante del equipo de desarrollo contemplado para dicho proyecto?

Calcular la productividad en el desarrollo de *software* no es una tarea sencilla y a lo largo del tiempo se han propuesto varias métricas para poder lograr este objetivo. Por ejemplo, se ha propuesto medir la productividad de un desarrollador basado en las horas promedio que le toma construir un archivo de texto. Otras variantes involucran el tiempo que toma producir un archivo de código en algún lenguaje de script, una pantalla de usuario, etc. Para el caso de PSP, el autor propone una métrica de productividad basada en las líneas de código nuevas que un desarrollador realiza en una hora. Si bien, esta métrica pudiera no ser la mejor, ha demostrado ser la más asertiva para medir la productividad de un ingeniero de *software*. Por lo tanto, si la medición del tamaño de un producto de *software* está basada en sus líneas de código y, la productividad de un desarrollador está basada en las líneas de código que produce en una hora, la estimación del tiempo requerido para desarrollar el producto se puede calcular fácilmente dividiendo el tamaño estimado del producto entre la productividad del desarrollador:

Para concluir con este apartado, sólo resta mencionar que la productividad de un desarrollador no es constante. A lo largo del tiempo y después de la experiencia en el desarrollo de varios proyectos, los desarrolladores van aumentando su productividad a la vez que haciendo más eficientemente sus tareas. Por eso es importante ir recolectando datos de la productividad en cada proyecto. Esto permitirá a su vez, obtener información sobre la productividad de los equipos de desarrollo así como información que servirá de base para poder estimar el tiempo de desarrollo de futuros proyectos. (Humphrey, W. 1995. Pág. 88).

2.2.7. PSP 0.1

En PSP 0.1 se tienen los siguientes objetivos:

- Medir el tamaño de los programas que se realizan.
- Realizar el conteo de tamaño para los programas realizados.
- Desarrollar métricas de tamaño asertivas y precisas.

En PSP 0.1 se utilizan todos los documentos vistos en PSP0 y se agregan los siguientes tres:

- PIP, (por sus siglas en inglés, Process Improvement Proposal), significa Propuesta de Mejora del Proceso. Este formato es un documento libre donde el ingeniero de software, por cada programa que realice a partir del nivel 0.1 de PSP, deberá emitir una propuesta de mejora al PSP mediante este documento. Es importante mencionar que en una PIP no basta con describir alguna problemática que se tenga con el PSP, sino que además se tiene que proponer alguna alternativa que ayude a mejorar o erradicar dicha problemática.
- Formato para el conteo del tamaño del programa.
- Estándar de codificación.

Como resultado de esto, el formato del resumen del plan del proyecto también deberá mostrar los datos referentes al tamaño del programa tomando en cuenta que a partir de este nivel, el desarrollo del programa debe ser planeado por cada fase del ciclo de desarrollo. (Echeverría, C.M., Echeverría, C.D. Mera, J.L. 2006. Pág 23-26).

2.3. Estimación del tamaño del *software*

Una vez que se tiene una métrica para medir el tamaño de un producto de *software*, se tiene también una base para poder estimar que tan grande será un nuevo proyecto basado en los datos históricos previos de otros proyectos.

En principio, las estimaciones son realizadas comparando el trabajo planeado con el trabajo realizado en proyectos anteriores. Si se divide el proyecto actual en partes más pequeñas y se comparan con partes más pequeñas de proyectos anteriores, se puede obtener una mejor estimación del tamaño total del proyecto. Esta estrategia funciona bien para la mayoría de los proyectos que se realizan en distintas áreas.

El proceso de estimación del tamaño de un producto de *software* a través de la división de tareas tiene la ventaja de que es fácilmente escalable. Si se es capaz de estimar

proyectos pequeños, se puede ser capaz de estimar proyectos grandes a través de esta técnica.

2.3.1. Contexto

Generalmente, cuando se planea el desarrollo de un nuevo proyecto, tan solo se conocen los requerimientos del cliente. La dificultad de estimar el tamaño del proyecto es precisamente el poder predecir el tamaño del producto final conociendo tan solo los requerimientos iniciales del cliente. Y, dado que nadie conoce en realidad que tan grande o cuanto se tardará realmente en realizarse, la estimación seré siempre un proceso con un cierto grado de incertidumbre. Por lo tanto, siempre será una ventaja el contar con una mejor definición así como el mayor número de requerimientos por parte del cliente. (Humphrey, W. 1995. Pág. 97).

2.3.2. Métodos de estimación

Existen varios métodos que ayudan a estimar el tamaño de un nuevo producto de *software*, así como establecer su costo. Algunos de estos métodos son herramientas comerciales, otras son de implementación personalizada, automatizadas o manuales. Dentro de las comerciales existe una amplia variedad que ayudan a las organizaciones de desarrollo de *software* a generar estimaciones más precisas y controlables. Algunos ejemplos son: COCOMO II, CoStar, CostModeler, CostXpert, KnowledgePlan®, PRICE S, SEER, SLIM y SoftCost.

Las herramientas comerciales están principalmente enfocadas a la estimación del costo del *software* y muchas de ellas comparten algunas características como:

- Especificación de requerimientos.
- Niveles de fase, actividad, tarea, etc.
- Definición del período laboral y vacacional.
- Manejo de salarios.
- Uso de diferentes tipos de proyectos.
- Métricas de puntos de función, líneas de código, etc.

Sin embargo, ningún método de estimación es lo suficientemente preciso para indicar con exactitud los tiempos que cada tarea nos llevará. Una buena práctica de la estimación es que la herramienta que se utilice, ya sea la comercial o propia, se vaya mejorando con cada proyecto y cada vez nos pueda ir dando valores más cercanos a la realidad. En el siguiente tema veremos los dos métodos que se recomiendan en PSP los cuáles son el

Proxy y el PROBE. Y en la unidad 3 verás métodos basados en juicio experto y estadísticos. (Jones, C. 2010. Pág. 1).

2.3.3. Proxy

En este tópico se analizará el método *Proxy*. El método *Proxy* es un método propuesto por Watts Humphrey, creador de PSP y, se verá más adelante, sirve para medir el tamaño que tendrá un producto de *software* basado en la división más elemental de los componentes que integrarán el producto que se piensa desarrollar. A estos elementos se les llama "*partes proxy*" cuya característica principal es que pueden ser comparados con otros elementos *proxy* correspondientes a proyectos desarrollados previamente de los cuales ya se tienen datos históricos.

El siguiente ejemplo muestra en primera instancia las bases de un método de estimación basado en un *proxy*. Si se piensa por ejemplo, en la construcción de una casa tomando en cuenta la cantidad de metros cuadrados que se van a construir, se podría tener una base para estimar el costo de construcción. Sin embargo, algunas personas, pueden pensar en términos de metros cuadrados basados en el número de cuartos y baños que tendrá la casa para realizar la estimación del costo de construcción. La estimación del *software* es un problema similar. Si se pudiera saber el número de tablas y relaciones entre ellas, que tendría la base de datos o, el número de líneas de código que tendrá el programa, se formularía una base para poder estimar su tamaño.

Es muy difícil realizar la estimación del tamaño de un programa basado únicamente en los requerimientos del cliente. Se requiere de algún *proxy* que permita relacionar el tamaño del producto con las funciones que se desean incorporar en el programa. Un *proxy* no es más que un sustituto del cual conocemos su tamaño. Ejemplos de *proxies* son tablas, clases, campos o pantallas.

Existen algunos criterios para seleccionar un *proxy* adecuadamente:

- La medida del *proxy* debe estar altamente relacionada con el esfuerzo requerido para desarrollar el producto.
- El contenido *proxy* de un producto debe ser automáticamente contable.
- El *proxy* debe ser fácil de visualizar al inicio del proyecto.
- El *proxy* debe ser personalizable a las necesidades de cada proyecto y desarrollador.
- El *proxy* debe ser sensible a las variaciones de implementación que afectan los costos de desarrollo o esfuerzo.

El siguiente diagrama muestra el proceso para seleccionar un *proxy* adecuado en el desarrollo de un proyecto.

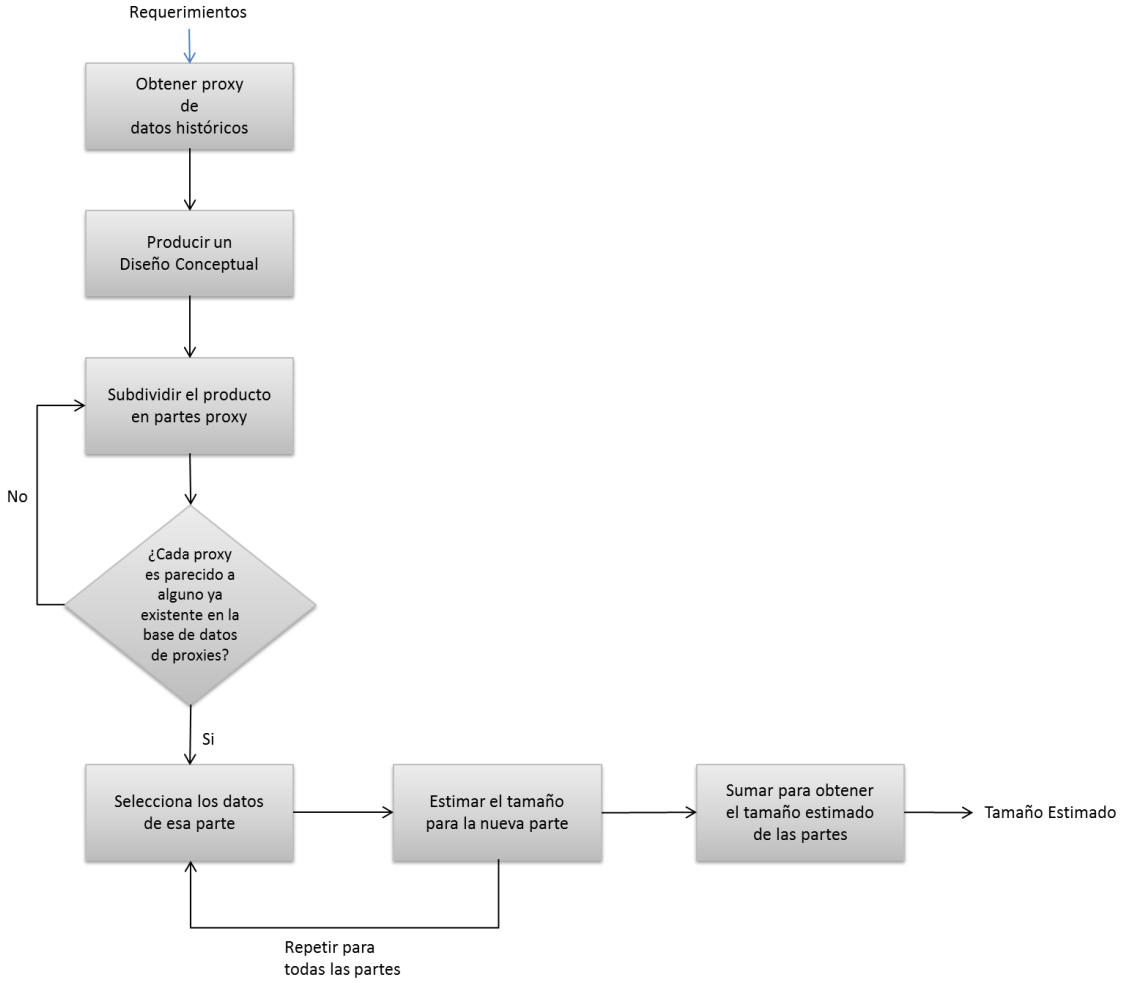

Figura. Diagrama del proceso para estimar proyectos de software utilizando el método Proxy. (Humphrey, W. 1995. Pág. 110).

Una vez que se comprendido el método de estimación por *proxy* se podrá analizar un método más complejo denominado PROBE, el cual será descrito en la siguiente sección. El método PROBE está basado en el método *Proxy*, pero además, permite estimar el tiempo requerido para el desarrollo de cada parte del proyecto. (Humphrey, W. 1995. Pág. 109).

2.3.4. PROBE

En esta sección se analizará el método *PROBE*. Este método permite obtener una estimación del tamaño de cada parte del proyecto (basado en la metodología *Proxy*) y

posteriormente, con estos datos, permite estimar el tiempo requerido para el desarrollo de cada una de las partes del proyecto.

El método *PROBE* utiliza datos históricos para realizar las estimaciones. Por ejemplo, si se estima el trabajo para desarrollar un sistema de consultas en una base de datos, se podría producir inicialmente un diseño conceptual y después dividirlo en partes.

Posteriormente, se podrían estimar el número de elementos en cada parte. Por ejemplo, si se estiman un total de 95 elementos y se sabe que cada elemento se lleva en producirse en promedio, 1.5 horas, el tiempo estimado total de desarrollo serían 142.5 horas.

Para hacer más precisas las estimaciones se requiere además de una base para calcular el tiempo promedio requerido para desarrollar un determinado componente del programa.

Una vez que se han comprendido los conceptos de estimación de tamaño y tiempo de desarrollo, es necesario comprender como son utilizados dentro del proceso PSP, en específico, en el nivel 1 o (también conocido como PSP1), el cual, será descrito en la siguiente sección. (Humphrey, W. 2005. Pág.105).

2.3.5. PSP 1

Durante la Unidad 1, se analizó el PSP 0 el cual es el nivel inicial del proceso personal de *software*. Como se recordará, el objetivo en PSP 0 era estimar el tiempo total que tomaría desarrollar un determinado programa. Una vez que se tiene completado el nivel PSP 0 se pasa al nivel PSP 0.1. En este segundo nivel el objetivo es estimar de forma empírica, basado en la experiencia del desarrollador, el tiempo de desarrollo por cada fase y se estima además, las líneas de código que podría tener el nuevo programa a desarrollar. Finalmente, en PSP 1, el objetivo es el mismo que en PSP 0.1, sólo que, la estimación de las líneas de código se realiza utilizando el método *Proxy* y adicionalmente, utilizando el método PROBE y los datos históricos de PSP 0 y PSP 0.1, se estima de forma automática el tiempo de desarrollo por cada fase.

El objetivo en PSP1 es establecer un procedimiento ordenado y repetible para realizar estimaciones de tamaño y tiempo de desarrollo por cada fase para un nuevo producto de *software*. Este nivel toma en cuenta dos nuevos aspectos:

- Estimación de tamaño y tiempo basado en el método *PROBE*.
- Reporte de pruebas.

La hoja del resumen del plan del proyecto se expande para mostrar además de los datos de PSP0.1, datos de productividad (medida en líneas de código por hora), tamaño planeado para todos los tipos de líneas de código. (Zapata, J., García, J., Cerrada, J. 2001. Pág. 60-61).

Para saber más

Si deseas saber acerca de PSP, TSP o CMMI puedes consultar la siguiente dirección electrónica:

- http://www.ecured.cu/index.php/Proceso_de_Software_Personal

Es una enciclopedia virtual, colaborativa y en español que fue creada para difundir el conocimiento de tecnologías de la información, sus fuentes son confiables. Puedes participar en sus foros y acceder a este sitio por medio de las redes sociales más importantes de la actualidad.

Fuentes de consulta

Bibliografía básica

- Humphrey, W. (1995) A *discipline for software engineering (The complete PSP Book)* United States of America: Addison Wesley.
- Humphrey, W. (2005) *PSP a Self-improvement process for software engineers.* United States of America: Addison Wesley.
- Zapata, J., García, J., Cerrada, J. (2001) *Introducción al proceso software personal*^{SM}. Madrid, España: Addison Wesley.

Bibliografía complementaria

- Echeverría, C.M., Echeverría, C.D. Mera, J.L. (2006). *"Implementación de un Sistema integrado de Control de Costos de Producción, Órdenes de Trabajo, Presupuesto de Obras, Bodega y Control de Inventario utilizando PSP y TSP".* Guayaquil, Ecuador. Escuela superior politécnica del litoral. Recuperado de http://www.dspace.espol.edu.ec/handle/123456789/5006
- Jones, C. (2010). *"Métodos de Estimación de Costos de Software para Grandes Proyectos".* Recuperado de http://www.liderdeproyecto.com/articulos/estimacion_costos_de_software.html