UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Aplicación, Evaluación de Metodologías de Estimación en Micro-Proyectos de Software y Desarrollo de un Sistema de Estimación.

T E S I S

PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:
Yesica Adriana Cabrera Bello

DIRECTOR DE TESIS:
Dr. MOISES HOMERO SÁNCHEZ LÓPEZ

CODIRECTOR:
ING. DAVID MARTÍNEZ TORRES

Huajuapan de León, Oaxaca. Abril, 2016

Contenido

Índice de Tablas

Índice de Figuras

Acrónimos

PHU	**Puntos de Historias de Usuario**
PBI	Product Backlog Item
SEI	Software Engineering Institute
CMMI	Capability Maturity Model Integration
ISO	International Organization for Standardization
LOC	Line Of Code
COCOMO	COnstructive COst MOdel
COSMIC- FFP	Common Software Measurement Internacional Consortium Full Function Points
PROBE	Proxy-based estimating
FP	Function Points
SLIM	Software Life-cycle Model)
SEER-SEM	Software Evaluation and Estimation of Resources - Software Estimating Model
CART	Classification And Regression Trees
CBR	Case-Based Reasoning
SCRUM	Marco de trabajo ágil

Capítulo 1. Introducción

En este capítulo se presenta una breve historia de la Ingeniería del Software y la crisis del software, la cual fue la causa de investigación y desarrollo de nuevas técnicas para desarrollar software de calidad y sobre todo métodos de estimación de esfuerzo para que los Proyectos de Desarrollo de Software pudieran ser entregados en tiempo y forma. Se explicarán algunas de las principales razones por las que los proyectos de desarrollo de software fallan. Como objeto central de este trabajo de tesis se optó por los proyectos denominados micro o de corta duración, se explicará su relevancia y un problema relacionado a estos, por lo cual son motivo de estudio. El resto del capítulo proporciona las bases sobre las cuales estará guiado este trabajo de investigación, es decir, se definirán la hipótesis y preguntas de investigación, el objetivo general y objetivos específicos, así como también el alcance y limitaciones del presente trabajo de tesis. La última parte del capítulo describe de forma breve la estructura general de este documento.

1.1 Antecedentes

Actualmente todos los países dependen de complejos sistemas basados en computadoras. Infraestructuras nacionales confían en sistemas basados en cómputo y más productos eléctricos incluyen una computadora y control de software. La industria manufacturera y de distribución es completamente computarizada, así como el sistema financiero. Por lo tanto la producción y mantenimiento de software rentable es esencial para el funcionamiento de la economía nacional e internacional (Pressman, 2001) (Sommerville, 2007).

La Ingeniería del Software es una disciplina ingenieril la cual se enfoca en el desarrollo rentable de sistemas de software de alta calidad (Pressman, 2001). El software es abstracto e intangible, no es limitado por materiales, o gobernado por leyes físicas o por procesos de manufactura. De alguna manera, esto simplifica la Ingeniería del Software, al no haber limitaciones físicas sobre el potencial del software. Sin embargo, esta falta de limitaciones naturales significa que el software puede fácilmente llegar a ser extremadamente complejo y de ahí muy difícil de entender (Sommerville, 2007).

La noción de *Ingeniería del Software* fue propuesta por primera vez en 1968 en una conferencia celebrada para discutir lo que fue después llamado la "crisis del software" (Palacio & Ruata, 2009). Esta crisis del software resultó directamente de la introducción de nuevo hardware de computadora basado en circuitos integrados. Su poder hizo que aplicaciones informáticas propuestas hasta ese entonces irrealizables fueran factibles ahora. El software resultante fue de magnitudes más grandes y más complejas que los sistemas previos (Sommerville, 2007).

Las primeras experiencias construyendo estos sistemas mostraron que el desarrollo informal de software no fue suficientemente bueno. La mayoría de los proyectos tuvieron algunas veces años de retraso. El costo de software fue mucho mayor de lo previsto, el software fue poco confiable, difícil de mantener y con pobre rendimiento, el desarrollo del software estuvo en crisis (Pressman, 2001). Los costos del hardware estuvieron cayendo mientras que los costos del software iban en aumento rápidamente (Piattini Velthuis & Calvo-Manzano Villalón, 2004). Nuevas técnicas y métodos fueron necesarios para controlar la complejidad inherente en grandes sistemas de software.

La IEEE Computer Society define la Ingeniería del Software como (Abran & Moore, 2004):

"1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software; que es la aplicación de la ingeniería al software".

La Ingeniería del Software es organizada en 10 Áreas del Conocimiento (Abran & Moore, 2004):

- Requerimientos de software
- Diseño de software
- Construcción de software
- Prueba de software

- Mantenimiento de software
- Administración de la configuración de software
- Administración de la Ingeniería de Software
- Proceso de Ingeniería de Software
- Instrumentos y Métodos de la Ingeniería de Software
- Calidad del software

Requerimientos de software

Esta área de conocimiento tiene que ver con todo lo relacionado a la recopilación de las necesidades del usuario, las cuales se verán reflejadas en los requerimientos del software, especificándolos de manera concisa y clara, para ser analizados y así determinar su costo de desarrollo, además de sus vulnerabilidades con el propósito de evitar problemas a lo largo del desarrollo.

Diseño de software

En este proceso se analizan los requerimientos de software, que son escritos en lenguaje de alto nivel, como resultado del análisis se obtiene una descomposición de todas las partes que constituirán al producto final, su organización y la comunicación entre cada componente. El nivel de detalle que deben tener cada parte o componente, debe ser suficiente para poder implementarlo. El diseño del software es un paso muy importante durante el desarrollo de un proyecto de software, ya que es aquí donde se construye la solución a los requerimientos, seleccionando el modelo más apropiado para definir las actividades subsecuentes en el proceso de desarrollo.

Construcción de software

La construcción de software, está íntimamente ligada con el diseño de software, ya que esta área del conocimiento toma como entrada el análisis realizado previamente en el diseño de software, para refinarlo mejor y llevarlo a un nivel mucho más detallado para dar como resultado software operativo, por medio de la codificación. Esta área también está relacionada con las pruebas del software, ya que durante la construcción se realizan pruebas unitarias y pruebas de integración. Al realizar las pruebas es evidente que se marcan parámetros de configuración con los que el software cuenta, por tal razón la construcción también está relacionada con la administración de la configuración del software.

Pruebas del software

Las pruebas se realizan para verificar la calidad del software desarrollado, las actividades de pruebas deben ser diseñadas para garantizar que el producto resultante satisface los requerimientos del usuario y que el software tiene el comportamiento esperado en casos de prueba específicos. Las pruebas del software han evolucionado y hoy en día se consideran parte del proceso de construcción y no solo un paso posterior a la finalización de la codificación para detectar errores. Poco a poco la planificación de las pruebas está comenzando a incluirse desde etapas tempranas del proyecto, para identificar debilidades potenciales.

Mantenimiento de software

El mantenimiento de software se refiere a las modificaciones que se realizan al software después de que se ha liberado y está siendo usado por los clientes, los cuales encuentran fallas que requieren ser corregidas. Las modificaciones se realizan al código y a la documentación. También las modificaciones se realizan no con el propósito de corregir fallos, sino para mejorar el software, por ejemplo, mejorar el rendimiento del sistema o agregar algunas nuevas tecnologías. Todas las modificaciones que se realicen deben ser documentadas y deben llevar un seguimiento para mantener un control sobre el software.

Administración de la configuración de software

Esta área se enfoca en determinar la configuración de un sistema, para que su funcionamiento se adapte a alguna situación determinada. Se debe llevar una relación entre el tipo de configuración de un sistema y los resultados esperados en el comportamiento del mismo al aplicar dicha configuración, así como también una

bitácora de cambios en la configuración, con el objetivo de mantener la consistencia, estabilidad y seguridad del sistema.

Administración de la Ingeniería de Software

Esta área se encarga de la planeación, organización, ejecución y control de las actividades de Ingeniería de Software. En otras palabras, se encarga de administrar las cuestiones relacionadas con los proyectos de software tales como la relación con los clientes, los procesos de ingeniería usados durante el proyecto, los resultados esperados, mantener el balance entre creatividad y disciplina , el grado de novedad y complejidad del software, mantenerse al tanto de los cambios en la tecnología, de relaciones con el personal, establecimiento de políticas y procedimientos internos, gestión de la comunicación, medición del avance del desarrollo, entre otros.

Proceso de Ingeniería de Software

Esta área del conocimiento se subdivide en dos niveles, el primero engloba las actividades técnicas y de gestión dentro del ciclo de vida del software. Este nivel es cubierto por las áreas mencionadas anteriormente, donde las actividades se siguen para llevar a cabo el proyecto.

El segundo se le llama meta-nivel, que se refiere a la definición, implementación, medición, administración de los cambios y mejoras de los mismos procesos del ciclo de vida del software. Esta área de conocimientos se encarga propiamente de este nivel. Es aquí donde las actividades que se siguen en un proceso determinado, son definidas.

Instrumentos y métodos de la Ingeniería de Software

Esta área se encarga de estudiar propiamente los instrumentos que pueden ser utilizados durante el desarrollo del software para automatizar tareas, reduciendo la carga cognoscitiva de los desarrolladores, con el fin de que estos puedan concentrarse en los aspectos creativos que son necesarios para encontrar las soluciones del proyecto. Respecto a los métodos, es aquí donde se proponen las notaciones y el vocabulario, como parte de una guía sistemática de actividades que se realizan durante el ciclo de vida del proyecto.

Calidad del software

Esta área del conocimiento se concentra en alcanzar la satisfacción total del cliente. La calidad del software define los modelos y criterios para evaluar la calidad de los procesos, que se siguen para desarrollar un producto de software, y los productos resultantes de estos procesos.

Así como se han diseñado procesos para el desarrollo de todo un proyecto de software en los últimos años, también cada una de las actividades fundamentales se ha especializado y se han propuestos técnicas para cada propósito específico, tal es el caso de la *especificación de software*. La cual es la primera actividad de un proceso desarrollo de software, siendo la base sobre la cual girará todo lo concerniente al proyecto. Al ser la base se espera que sea lo suficientemente sólida y estable para que el proyecto llegue a buen término. Pero realmente esto no ocurre, y más adelante se explica el porqué de esta situación.

Aunque las Tecnologías de Información (TI) están llegando a ser más fiables, rápidas y menos caras, los costos, complejidad y riesgos siguen aumentando en los proyectos de TI. En 1995, la empresa consultora The Standish Group realizó una encuesta a 365 administradores de tecnologías de la información. El reporte ampliamente citado, llamado "CHAOS report" fue sorprendente (Group T. S., 1995), el reporte informó que aunque los Estados Unidos gastaron arriba de $250 billones cada año en proyectos de desarrollo de aplicaciones de TI, el 31% de estos proyectos fueron cancelados antes de ser completados. El 53% fue casi completado, pero estos estuvieron sobre presupuestados, retrasados y/o no reunieron las especificaciones originales. El costo promedio excedido para compañías de talla mediana fue de 182% de la estimación original, mientras que el promedio calendarizado excedido fue de 202%. Es decir, los resultados de la encuesta, sugieren que un proyecto de tamaño mediano originalmente estima los costos por un $1 millón para un año de desarrollo, sin embargo realmente cuesta $1,820,000, y se desarrolla en tan solo un poco más de

dos años, y solamente incluye el 65% de las características y funciones previstas. Tristemente, el 48% de los administradores de TI encuestados cree que hubo más fracasos en los últimos cinco y diez años a la fecha del reporte.

El CHAOS report también provee algunas cuestiones interesantes, como las causas por las que algunos proyectos son exitosos mientras otros fallan. La Tabla 1, muestra las principales razones del fracaso de un proyecto.

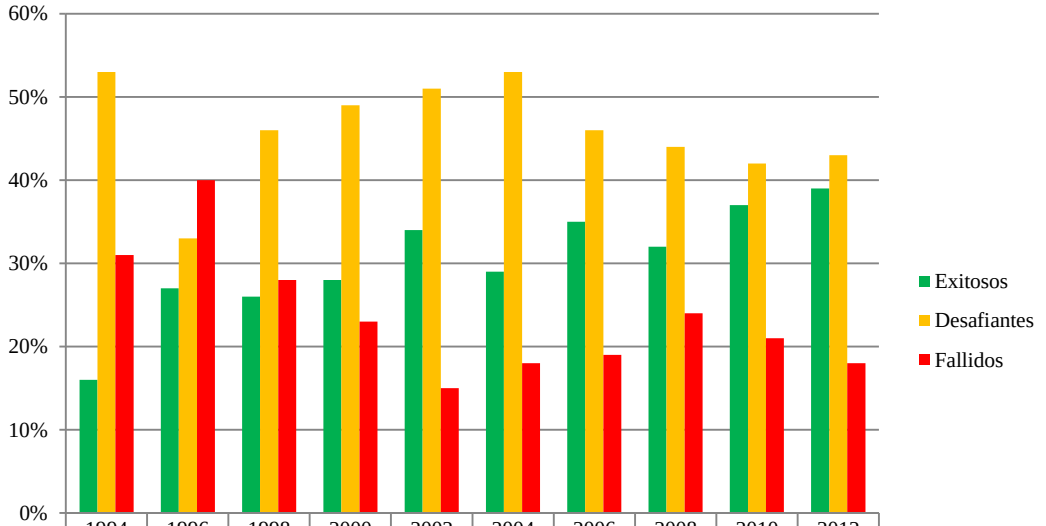
Tabla 1. Razones por las que los proyectos fallan
Fuente: Chaos Reports – The Standish Group, www.standishgroup.com

1.	Requerimientos incompletos	13%
2.	No se incluye al usuario	12.4%
3.	Insuficientes recursos/calendarización	10.6%
4.	Expectativas irrealistas	9.9%
5.	Falta de soporte ejecutivo	9.3%
6.	Cambio de requerimientos	8.7%
7.	Poca planeación	8.1%
8.	No se necesitó más tiempo	7.4%

Cómo se puede observar en la Tabla 1, una de las principales razones del fracaso de los proyectos, y que corresponde a la tercera posición de la tabla, son los insuficientes recursos/calendarización. Es decir, los recursos necesarios para realizar el proyecto no se estimaron adecuadamente.

En la Tabla 2 se puede observar como a través de los años la situación de los desarrollos de software no ha cambiado tanto. Los proyectos fallidos son aquellos que fueron cancelados antes de terminarse, los proyectos desafiantes son aquellos que fueron terminados pero fuera de tiempo, con sobre costos y algunas veces no contenían la funcionalidad especificada por los usuarios. Los proyectos exitosos terminaron en el tiempo acordado, sin sobre pasar el presupuesto acordado y con toda la funcionalidad requerida.

Tabla 2. Status de los desarrollos de software de 1994 a 2012
Fuente: (Group T. S., 1995) (Group T. S., 2009) (Group T. S., 2010) (Group T. S., 2011) (Group T. S., 2012) (Group T. S., 2013) **-The Standish Group, www.standishgroup.com**



	1994	1996	1998	2000	2002	2004	2006	2008	2010	2012
Exitosos	16%	27%	26%	28%	34%	29%	35%	32%	37%	39%
Desafiantes	53%	33%	46%	49%	51%	53%	46%	44%	42%	43%
Fallidos	31%	40%	28%	23%	15%	18%	19%	24%	21%	18%

Varios reportes de análisis (Group T. S., 1995) (Group T. S., 2009) (Group T. S., 2010) (Group T. S., 2011) (Group T. S., 2012) (Group T. S., 2013) bien documentados subrayan a la mala estimación inicial como uno de los factores clave responsables del fracaso de un proyecto de software. Hay varias razones conocidas para que una estimación sea mala, estas razones incluyen (Parthasarathy, 2007)

- Ignorar datos históricos
- Optimismo y sesgo en la estimación
- Incertidumbre de los requerimientos
- Ninguna estimación
- Administración bajo presión
- Estimadores no calificados
- Presupuesto restringido

Cada una de estas razones, individualmente o en conjunto, causan una mala estimación. A través de capacitaciones formales y tutorías se puede ayudar a realizar mejores estimaciones, pero la capacitación necesita ser complementada con un proceso bien definido para formalmente recopilar métricas de datos pasados. Si el proceso es implementado, el equipo de desarrollo podrá manejar los riesgos exitosamente, derivados algunas veces por la administración en forma de limitaciones presupuestarias y un irracional calendario de entrega.

Por mencionar unos ejemplos, las organizaciones ISO y SEI tienen procesos bien definidos para recolectar métricas, analizarlas y sugerir mejoras en la ejecución de las actividades del proceso. La estimación juega un rol significativo en la definición y captura de métricas, como por ejemplo las recomendaciones ISO y SEI/CMMI. Las organizaciones que están dispuestas a implementar efectivamente los procesos ISO y SEI/CMMI encuentran estos criterios de medición de inmenso valor. Ellos ayudan a medir varios parámetros de ejecución del proyecto a través de una amplia variedad de plataformas tecnológicas, funciones de negocio e incluso competencia de equipos de desarrollo en una situación de igualdad. Aquí hay algunos de los beneficios clave del uso de un proceso de recolección de métricas en una organización para la estimación de recursos (Parthasarathy, 2007):

- Métricas de ejecución de proyectos anteriores ayudan a evaluar y analizar las capacidades, fortalezas y debilidades de los procesos, dominio y habilidades tecnológicas, también como los métodos de ejecución del proyecto desplegado a través de la organización.
- Experiencias pasadas también muestran que como la ejecución del proyecto es una de las etapas finales, las pruebas y las actividades de corrección de errores incrementan.
- Las métricas de ejecución de proyectos actuales ayudan en la medición, tutoría y monitoreo de los proyectos en marcha en diferentes etapas del ciclo de vida de la ejecución. Comparándolas con métricas pasadas ayuda en correcciones rápidas y puesta a punto, así se mejora la probabilidad de entregar el proyecto a tiempo, dentro de los costos y con parámetros de calidad.

A futuro se necesita ese conjunto de objetivos a alcanzar en los siguientes 6 o 12 meses. Estos objetivos podrían incluir mejoras de la productividad en un 5% o una reducción en la densidad de defectos del 10%.

En muchos casos los proyectos de software son subestimados y esto es percibido al finalizar el desarrollo del proyecto, otros problemas concurrentes son las entregas del proyecto fuera del tiempo acordado, falta de control sobre la productividad de los programadores y dificultad para mantener el producto final. Lo anterior ejemplifica las características que tiene un proyecto que no terminó exitosamente.
Este tipo de problemas también se presentan en los "Proyectos de Software de Corta Duración", a este tipo de proyectos los denominaremos en este trabajo indistintamente "Proyectos Cortos", o "Micro-Proyectos". Un micro-proyecto será todo aquel desarrollo de software que cumpla las siguientes tres características (Sánchez López & et al., 2012):

- El tiempo de desarrollo proyectado sea menor a 1.5 meses.
- El presupuesto estimado no rebase los 50,000 pesos.
- El tamaño del equipo (integrantes) sea como máximo de dos personas. (En cualquier rol)

Las empresas desarrollan un gran porcentaje de proyectos de corta duración a lo largo de su historia, constituyendo 81% de sus proyectos totales, 11% pequeños y 8% medianos y grandes (Aguilar, et al., 2014).

Entre las metodologías que pueden ayudar a micro y pequeñas empresas en la estimación de costos de desarrollo de proyectos de software, encontramos la metodología de Estimación de Puntos de Función, la cual brinda un margen de costos monetarios y temporales, para el desarrollo de software. Esta metodología es sencilla y fácil de aplicar (Longstreet, August 2002).

1.2 Planteamiento del problema

Como parte de un proyecto desarrollado por el grupo de investigación de Ingeniería del Software de la Universidad Tecnológica de la Mixteca, se realizó una fase de análisis, que consistió en aplicar la metodología de Puntos de Función + COCOMO a una muestra de 7 micro-proyectos realizados anteriormente por empresas.
Durante el análisis se detectó, que el 100% de los proyectos de la muestra tenían una diferencia considerable entre los resultados obtenidos a través del método de Puntos de Función + COCOMO, y los costos reales erogados por las empresas que los realizaron. Ver Anexo K, donde se incluyen los resultados de la aplicación del Método Puntos de Función + COCOMO a 7 micro-proyectos, como actividades de un Proyecto de Investigación formal.

Las diferencias encontradas entre los costos reales y los costos estimados con las metodologías mencionadas fueron significativas, en la mayoría de los casos de más del triple. Siendo siempre los costos reales menores a los costos estimados. Con dichos resultados identificamos el siguiente problema:
"Las estimaciones realizadas con los métodos de Puntos de Función + COCOMO se alejan en promedio un 1439.99% de los costos reales de los proyectos de software de corta duración en la muestra analizada".

1.3 Hipótesis

Existen una o más metodologías de estimación de software que pueden ser aplicadas a los micro-proyectos de desarrollo de software con un grado de precisión que no rebase el 30% de los costos reales manejados en el mercado mexicano. Y el método más preciso puede ser automatizado mediante el desarrollo de una aplicación informática.

1.4 Preguntas de Investigación

Las preguntas que se planteó resolver con esta investigación fueron:

1. ¿Cuáles son los métodos de estimación más utilizados hoy en día?
2. ¿Cuáles son las características comunes y diferencias entre esos métodos?
3. ¿Existe un método de estimación que por sus características se adecúa más a los proyectos de software de corta duración?
4. ¿Los micro-proyectos de software tienen características especiales que requieren métodos de estimación muy específicos?

1.5 Objetivo General

Analizar los métodos de estimación más utilizados y conocidos para compararlos y aplicarlos en una muestra de proyectos de software de corta duración (micro-proyectos) con el fin de identificar cuál de estos se ajusta a los costos reales del mercado mexicano. Finalmente desarrollar una aplicación informática que implemente el método identificado para estimación de esfuerzo y costo de micro-proyectos de software.

1.6 Objetivos Específicos

- Investigar los métodos existentes para la estimación de software.
- Analizar cada uno de estos métodos de estimación.
- Comparar cada método con los otros para determinar sus ventajas y desventajas.
- Seleccionar cinco métodos para aplicarlos a una muestra de 20 micro-proyectos de software.
- Realizar el análisis para determinar cuál es el método que estima con mayor precisión los costos de desarrollo de los micro-proyectos, comparando los resultados obtenidos con los costos reales.
- Desarrollar una aplicación gráfica para la estimación de micro-proyectos.

1.7 Alcance y Limitaciones

Este trabajo estará constituido medularmente por un estudio analítico el cual permita decidir entre un número de métodos de estimación conocidos y utilizados, cuál de ellos es el que arroja resultados más cercanos a los costos reales de micro-proyectos de software desarrollados por MiPYMES mexicanas.

Para ser más explícitos, el estudio analítico se dividirá en dos partes, la primera consistirá en la revisión de una lista de diferentes métodos de estimación (los más conocidos y utilizados) a través de una revisión sistemática de literatura, para posteriormente en base a las características investigadas de cada uno, seleccionar 5 métodos de estimación.

La segunda partes del estudio analítico, incluye actividades prácticas, consistiendo éstas en la aplicación de los 5 métodos de estimación a una muestra de 20 micro-proyectos de software, los resultados obtenidos serán analizados para seleccionar uno de entre los 5 métodos, el cual ajusta de manera más exacta los costos reales de desarrollo de los micro-proyectos de la muestra.

Se abarcarán 20 proyectos de software como meta de este trabajo de investigación. Y se seleccionaran como máximo 5 métodos para acotar el tiempo de desarrollo de la tesis.

El tipo de proyectos a analizar es el de los proyectos de escritorio y proyectos web., Quedan fuera de este trabajo: Apps p/móviles, embebidos, multimedia, distribuidos.

El criterio de selección mencionado en la hipótesis del 30%, se toma porque en Ingeniería de Software las predicciones realizadas mediante el método estadístico de regresión toman un intervalo de predicción de 70% (Humphrey, A Discipline for Software Enginnering, 1995).

El método identificado que estime de manera más exacta los costos de los micro-proyectos de software, será el aporte significativo del trabajo de tesis. Como complemento se desarrollará una aplicación informática para automatizar el método seleccionado.

1.8 Importancia y Relevancia del Problema

De acuerdo con datos del Instituto Nacional de Estadística y Geografía, en México existen aproximadamente 4 millones 15 mil unidades empresariales, de las cuales 99.8% son MiPYMES que generan 52% del Producto Interno Bruto (PIB) y 73% del empleo en el país (Consejo Mexicano para el Desarrollo Economico y Social, 2013).

En cuanto a la industria del software, se estima que en México alrededor de 300 empresas conforman esta industria y cerca del 20% se encuentran formalmente estructuradas, ya sea como subsidiarias de grandes empresas internacionales, como organizaciones netamente mexicanas, o una combinación de ambas. Un aspecto importante relacionado con este hecho es que más del 90% del universo de este sector lo integran

MiPYMES, la mayoría de las cuales carece de estructura y de un enfoque formal en sus actividades de venta y elaboración de contratos (Peñaloza Báez, 2002).
La perspectiva de estas compañías está enfocada en reunir requerimientos y fijar una fecha límite, sin ofrecer ningún valor agregado (Munive & Trejo, 2003). En muchos casos los proyecto de software son subestimados y esto es percibido al finalizar el desarrollo del proyecto, otros problemas recurrentes son las entregas del proyecto fuera del tiempo acordado, falta de control sobre la productividad de los programadores y dificultad para mantener el producto final. Lo anterior ejemplifica las características que tiene un proyecto que no terminó exitosamente.

Hanna Oktaba (Oktaba & Piattini, 2008) nos presenta la siguiente clasificación, la organización de cooperación y desarrollo económico y la unión europea definen las siguientes categorías para las empresas según su tamaño en términos del número de empleados que las conforman. Una micro-empresa tiene una plantilla de empleados menor a 10, una pequeña está conformada con 50 o menos empleados y una empresa mediana cuenta con 250 empleados o menos.

En México, las empresas catalogadas como micro-empresas son aquellas en las cuales laboran menos de 10 personas. La mayoría de las empresas desarrolladoras de software caen en esta categoría. Podría pensarse que los proyectos que desarrollan estas micro-empresas son para clientes que también son MiPYMES, pero esto no es así, la mayoría son contratados por empresas grandes con más de 500 empleados (Sánchez López & et al., 2012), que tienen necesidades específicas y requieren un proyecto corto con la calidad que sus propias políticas organizacionales requieren, esto representa un reto para las empresas desarrolladoras.
Las empresas desarrollan un gran porcentaje de proyectos de corta duración a lo largo de su historia, constituyendo 81% de sus proyectos totales, 11% pequeños y 8% medianos y grandes (Aguilar, et al., 2014), (Sánchez López & et al., 2012).

El paso inicial que una empresa debe realizar es determinar un presupuesto, esta actividad es uno de los puntos más importantes antes de comenzar el proyecto, ya que se corre el riesgo de tener costos subestimados que pondrían en riesgo la culminación del proyecto.

La precisión de las estimaciones es tan crítica para proyectos de software como lo son para proyectos en fabricación, construcción y profesiones similares.

Malas estimaciones o nulas puede llevar a situaciones donde el éxito del proyecto esté en riesgo. Aquí hay algunos ejemplos:

- Proyectos de software tiene una notoria tendencia a inclinarse hacia el fracaso si no se manejan con suma diligencia. Cada reporte publicado muestra la tasa de proyectos de software completamente fracasados. Una mala estimación está entre las principales causas por las que los proyectos fracasan (Sommerville, 2007),
- El proceso de estimación debería abarcar todas las actividades que requieren esfuerzo y proveen suficiente contingencia para otros factores de riesgo que pueden descarrilar el proyecto. Los factores de riesgo incluyen (Parthasarathy, 2007):
    - Definición del alcance del proyecto incompleta o inconsistente
    - Capacidad del equipo del proyecto
    - Reglas de negocio y algoritmos complejos
    - Cambios inesperados en el ambiente tecnológico
- La administración del proyecto depende profundamente de la anticipación al retraso en la correcta ejecución del proceso del proyecto y la realización de las apropiadas correcciones. El esfuerzo para la estimación para actividades individuales y una constante revisión sobre la desviación en estos esfuerzos es una entrada crítica para buenas prácticas en la administración de proyectos.

Debido a la importancia de realizar una buena estimación de costos y tiempo antes de iniciar el desarrollo de un proyecto de corta duración, se pretende encontrar una metodología para que las estimaciones resultantes se apeguen más a los costos reales de las empresas desarrolladoras de software mexicanas. Con esto se espera que los desarrollos de software de corta duración tengan mayor probabilidad de concluir exitosamente.

Esta investigación se enfoca a los micro-proyectos, ya que constituyen un porcentaje muy elevado, 81%, de los proyectos de trabajo de empresas desarrolladoras de software (resultado obtenido de una muestra de 107 empresas) (Aguilar, et al., 2014). Esta investigación puede tener un impacto significativo para las empresas desarrolladoras de software en México, si se encuentra una metodología que permita aproximarse a los costos y tiempos reales de los proyectos. Al encontrar la metodología con mejor aproximación, será automatizada mediante el desarrollo de una aplicación informática.

1.9 Estructura de la Tesis

La estructura del documento de tesis se detalla a continuación:

El Capítulo 2 presenta un "Marco Teórico", en el cual se explica que es la estimación de proyectos de software, la diferencia entre el concepto de estimación y el de medición, así mismo se presenta una amplia variedad de métodos de estimación recopilados de publicaciones de autores reconocidos en la materia y se habla de las MiPYMES desarrolladoras de software y lo importante que es para estas, realizar buenas estimaciones de software.

El Capítulo 3 llamado "Aproximación de la Solución", describe los métodos que se consultaron para llevar de la mano la investigación. El método guía seleccionado fue EBSE, *Evidence-Based Software Engineering* (Ingeniería del Software Basada en Evidencias) y como auxiliar para los tres puntos principales de EBSE se utilizó el método SLR, *Systematic Literature Review* (Revisión Sistemática de Literatura) para recabar información de calidad. El resto del capítulo se centra en el cuarto paso del proceso EBSE y en él se abordan los Métodos de Estimación más utilizados así como sus características, una vez sintetizada esa información fueron seleccionados 5 métodos de estimación los cuales se aplicaron a una muestra de micro-proyectos, con el fin de comparar los resultados obtenidos por la metodología y los reales, estos datos fueron registrados y se muestran al final del capítulo.

El Capítulo 4 "Aplicación y Resultados", lo constituye el último paso del proceso EBSE, y en este se realizó el análisis y síntesis de los datos obtenidos en el capítulo anterior con el fin de decidir cuál es el método o cuáles son los métodos de estimación más apropiados para los micro-proyectos. Se respondieron las preguntas de investigación y se comprobó la hipótesis. En este capítulo termina el proceso de investigación.

El Capítulo 5 esta exclusivamente dedicado al "Desarrollo de la Aplicación de Software" que implementó el método seleccionado con los resultados más apropiados para estimación de micro-proyectos del capítulo anterior. La información que se muestra en el capítulo es referente al proceso de desarrollo del software: Requerimientos, Diseño, Construcción, Pruebas y Liberación.

En el Capítulo 6 se presentan las conclusiones obtenidas durante el desarrollo de este trabajo y trabajos futuros relacionados.

En la sección de anexos se encuentra información necesaria para complementar el trabajo.

Por último se presentan las referencias bibliográficas utilizadas en el desarrollo de esta tesis.

Capítulo 2. Marco Teórico

El constante retraso en la entrega de los proyectos de software y la sobre carga en los presupuestos acordados entre los clientes y proveedores de software, ha sido un problema identificado y discutido desde 1968, año en el que se acuñó el término *Crisis de Software* para referirse a este fenómeno. A raíz de eso, a lo largo de estos años se han ideado formas para mejorar la estimación de esfuerzo de desarrollo de software y costos del mismo, volviéndose así una actividad crítica durante las etapas tempranas del ciclo de vida del software. En breve se describen los factores que intervienen en el proceso de estimación y la diferencia entre estimar y medir. Ha sido tan relevante la estimación de esfuerzo de desarrollo en etapas tempranas del ciclo de vida de software, que la comunidad dedicada a la investigación en el ámbito de la Ingeniería del Software ha propuesto diversos métodos de estimación, así como clasificaciones de los mismos. En este capítulo se describen algunos de los métodos de estimación, seleccionados respecto a la opinión de diversos autores los cuales coinciden en que estos son de los más reconocidos y utilizados en Ingeniería del Software.

2.1 Estimación de proyectos de software

La estimación es el arte de aproximación y una actividad que es hecha antes de que un producto comience a tomar forma. Es natural que la medición de una actividad nunca sea perfecta al 100%. Dado que la estimación es una actividad dependiente de cuatro factores principales: alcance, ambiente, experiencia y herramientas, si se mejora la confianza en la predicción de los resultados de estos factores, eso ayuda en el aumento de la precisión de la estimación. La contribución de los cuatro ingredientes es igualmente significativa y juega un papel importante en la estimación final del proyecto (Piattini Velthuis & Calvo-Manzano Villalón, 2004) (Parthasarathy, 2007) .

La estimación de software es similar a alguna otra actividad de estimación siempre y cuando se conozcan los parámetros clave requeridos para hacer dicha estimación, en el caso del software tenemos que considerar el alcance, el entorno, la experiencia y las herramientas.

Uno de los puntos en los que nos concentraremos es el alcance de un proyecto de software. Los desarrolladores de software dan cuenta de que es extremadamente difícil capturar exactamente el alcance de un proyecto de software. Una amplia variedad de ingredientes hacen un sistema de software completo (Parthasarathy, 2007):

- Funciones de negocio dirigidas a través de la aplicación del sistema.
- Los variados módulos de la aplicación.
- La plataforma, lenguaje y uso de bases de datos.
- Herramientas usadas como parte de la aplicación.
- Rendimiento y otros atributos de capacidad de ejecución del sistema.
- Interfaz con otros sistemas en el ambiente.

Aunque el número de métodos usados para capturar el alcance de los componentes de un sistema de software ha evolucionado en pocas décadas, casi ninguno de ellos puede medir el alcance con precisión. En términos de software, el alcance se equipara con el tamaño del sistema de software. El tamaño ha sido definido en diferentes unidades de medición, como Líneas de Código (LOC, Lines of Code), Puntos de Función (FP, Function Points), número de programas y número de objetos. Algunos de los modelos de estimación de software populares que han sido desarrollados por expertos incluyen (Parthasarathy, 2007) (Fenton & Pfleeger, 1998):

- Método del Análisis de Puntos de Función (Function Point Analysis Method)
- Mark II Function Points
- COCOMO II Model
- Feature Points
- Object Point

- COSMIC-FFP Method
- Delphi Method
- Use Case Point

También existen algunos otros métodos de estimación descritos en (Humphrey, A Discipline for Software Enginnering, 1995) (Cohn, 2006) (Sommerville, 2007) (Mendes, 2008) (Marchewka, 2013), en algunas ocasiones los autores coinciden en la clasificación:

- Basadas en la opinión de un experto
- Analogías
- Desagregación
- Método Bottom-Up
- Método Top-Down
- COCOMO81
- SLIM
- SEER-SEM
- Método Delphi
- Time-Boxing
- Fuzzy-Logic
- PROBE
- Arboles de Clasificación y Regresión (CARTs)
- Redes Neuronales
- Razonamiento Basado en Casos (CBR)
- Puntos de Historias Usuario

El proceso de estimación en cualquier proyecto de software no es únicamente integral, sino también un componente muy crítico. El éxito o fracaso del proyecto depende profundamente de la precisión del esfuerzo y el esquema de estimación, entre otros.

Aunque el esfuerzo final estimado parece ser una sola entidad, en realidad es un conjunto de un número de componentes individuales que tienen su propia complejidad y variación basada sobre la situación del proyecto y necesidades funcionales. Los administradores de proyectos tienden a centrarse más en la entrega del proyecto según lo especificado, pero ponen menos atención al realizar una cuidadosa estimación de varias actividades y varias etapas de la ejecución del ciclo de vida del proyecto (Munive & Trejo, 2003). Esto ha llevado a serias cuestiones, algunas veces resultando en desechar el proyecto completamente.

El papel de la estimación en proyectos de software puede ser resumido como sigue (Parthasarathy, 2007):

- *Incubación*: provee una amplia guía para evaluar la viabilidad y la probabilidad de terminación exitosa del mismo proyecto.
- *Contrato de software*: la estimación juega un rol crítico para definir el acuerdo contractual principal, incluyendo el costo.
- *Ejecución*: ayuda al equipo del proyecto en el establecimiento de objetivos claros por esfuerzo, esquema y costo. También ayuda al equipo monitor (equipo encargado de monitorear el progreso del desarrollo del proyecto, según lo estimado) y al asesor del progreso del proyecto.
- *Proyectos complejos*: en situaciones donde la ejecución de su proyecto sucede en un entorno existente complejo de tecnologías de la información, debe prever la complejidad de diseño debido a la integración de aplicaciones existentes.

2.2 Estimación y medición

Estimación y medición son dos caras del mismo atributo de una aplicación de software: *tamaño*. Esta explicación puede ser aplicada a otros atributos de proyectos de software que incluyen esfuerzo, programación y parámetros de calidad (Fenton & Pfleeger, 1998).

Estimación: Durante el proceso de contratación hay una necesidad de estimar el tamaño, esfuerzo y costo de un proyecto de software que aún no ha sido desarrollado. En cada hito en las etapas de ejecución del proyecto, el balance de esfuerzo requerido para entregar el proyecto necesita ser estimado. La estimación definitivamente no es hecha al final del proyecto, sino en etapas tempranas del proyecto, una vez que se tiene información del sistema que se desea desarrollar.

Medición: Con la excepción de la fase de contratación y el tiempo precedido por el primer hito la actividad de medición toma lugar en todas las otras situaciones.

El proceso de estimación y medición es usualmente complejo debido a que cada proyecto de software y sus atributos son únicos. En la Figura 1 se muestran muchos factores variables que necesitan ser medidos.

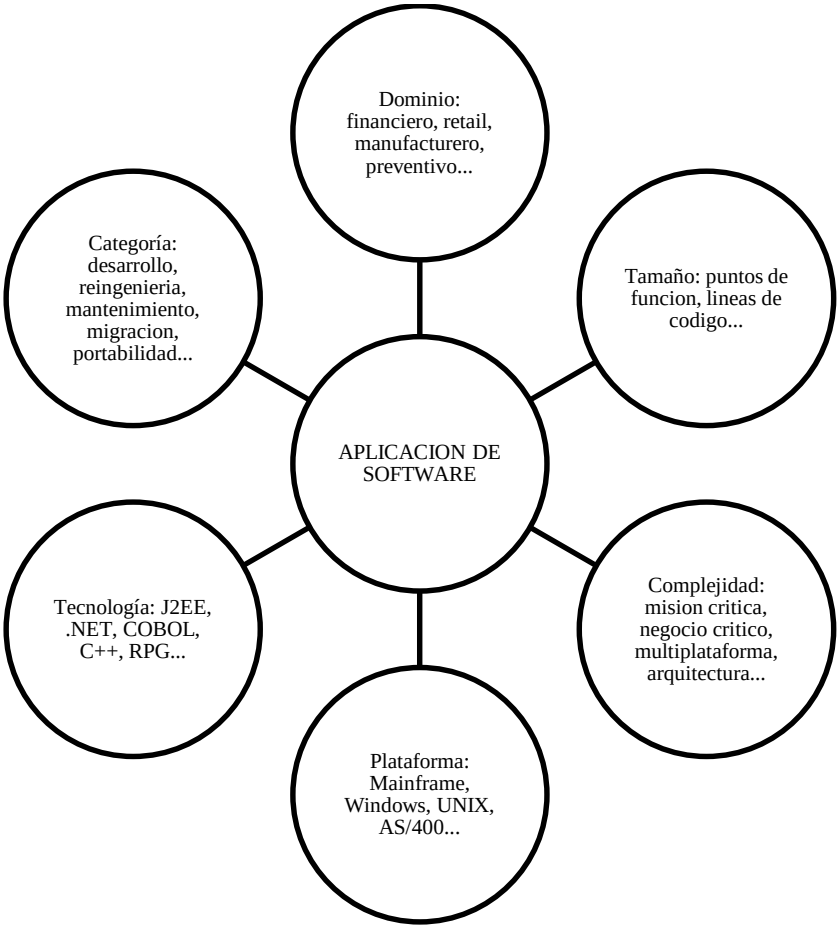


Figura 1. Complejidad de la estimación y la medición (Parthasarathy, 2007)

Una manera que facilita la estimación de software, es utilizar el enfoque divide y vencerás, es decir, en el contexto de la Ingeniería del Software se utiliza la modularización.

La modularización es simplemente un proceso que consiste en identificar los atributos individuales de los componentes de la estimación y la relación entre ellos. Estos componentes pueden ser clasificados generalmente como sigue (Parthasarathy, 2007):

- El *tamaño* de la aplicación.
- La *plataforma* sobre la cual la aplicación es desarrollada.

- El *tipo* de ejecución del proyecto.
- Las *habilidades/capacidades* del equipo del proyecto.
- Otros componentes como la calidad de los procesos y herramientas, etc.

La estimación del esfuerzo total para entregar la aplicación de software dependerá enormemente de la complejidad relacionada a estos componentes, necesariamente en el orden dado (Parthasarathy, 2007).

Tamaño: este atributo es el centro de todos los otros atributos e influye enormemente el esfuerzo total durante el proceso de estimación. El tamaño está directamente relacionado con la funcionalidad. Hay pocas alternativas para medir el tamaño. Puntos de Función es el método más popular, dando el número de Puntos de Función (FP) ajustado, contados en base a la funcionalidad a entregar por la aplicación. El conteo de los Puntos de Función es independiente de la tecnología sobre la cual la aplicación va ser desarrollada. Así mismo, hay estudios donde se relacionan puntos de función frente a LOC dependiendo del lenguaje de programación. Líneas de Código (LOC) es otro método de definición de tamaño de una aplicación. LOC es normalmente contado para aplicaciones que han sido completadas. El LOC está siempre disponible para contar en lenguajes particulares como COBOL, JAVA, C#, etc. Otro método popular para calcular el tamaño de una aplicación está basado en el número de casos de uso (UC) preparados durante la definición del alcance. Seleccionar y escribir los casos de uso en el correcto nivel de granularidad -y también de consistencia- es bastante complicado. Con el incremento del uso de metodologías de desarrollo ágil, otra métrica utilizada para medir el tamaño del software son los puntos de historias de usuario o puntos de historia. Un punto de historia (entender las historias como casos de uso simplificados (Beck , 1999)) es un número entero que representa la agregación de un número de aspectos, cada uno de los cuales contribuye a la "grandeza" potencial de una historia, dicho número estima el tamaño de una historia comparada con otras historias de un tipo similar. Los puntos de historia no tienen unidades y son numéricamente relevantes (esto es, una historia de 2 puntos deberá reflejar que será dos veces más grande que una historia de un punto) (Leffingwell, 2011).

Tecnología/Plataforma: el vehículo para convertir las funciones de negocio dentro de una aplicación de software que es capaz de introducir, sacar, reportar y almacenar datos de manera fácil es la tecnología. La tecnología incluye varios parámetros como estos:

- Lenguaje (COBOL, JAVA, C#...)
- Plataforma (UNIX, MVS, Windows…)
- Archivos/Bases de datos (VSAM, IMS, IDMS, RDBMS, DB2…)
- Arquitectura (2 tiers, 3 tiers…)
- Middleware (IBM MQ, MSMQ…)

Tipo de proceso: dependiendo sobre las necesidades del negocio, un proyecto de software podría ser ejecutado con variantes del proceso. Estos tipos de proyectos incluyen desarrollo, reingeniería, mantenimiento, mejora, migración y portabilidad.

Habilidades/Competencia: denominado de otra manera como "productividad", la velocidad con la cual el software puede ser entregado depende profundamente de las habilidades del equipo de desarrollo.

Otros: un número de otros factores indirectos actúan por encima de la ejecución del proyecto. Esto incluye administración del proyecto, calidad calculada y otras actividades relacionadas. En la mayoría de las situaciones, el esfuerzo para estas actividades es un porcentaje de todo el esfuerzo estimado del proyecto.

En la Tabla 3 se muestran otros tipos de entidades y atributos de un proyecto de software que pueden ser medidos.

Tabla 3. Componentes de medición del software (Fenton & Pfleeger, 1998)

ENTIDADES	ATRIBUTOS	
Productos	***Internos***	***Externos***
Especificaciones	Tamaño, reúso, modularidad, redundancia, funcionalidad, correctes sintáctica…	Comprensibilidad, mantenibilidad…
Diseños	Tamaño, reúso, modularidad, acoplamiento, cohesión, funcionalidad…	Calidad, complejidad, mantenibilidad…
Código	Tamaño, reúso, modularidad, acoplamiento, funcionalidad, complejidad algorítmica, estructuralidad del control de flujo…	Confiabilidad, usabilidad, mantenibilidad…
Datos de prueba	Tamaño, nivel de cobertura…	Calidad…
Procesos	***Internos***	***Externos***
Construcción de la especificación	Tiempo, esfuerzo, numero de cambios en los requerimientos…	Calidad, costo, estabilidad…
Diseño detallado	Tiempo, esfuerzo, numero de defectos del especificación encontrados…	Costo, rentabilidad…
Pruebas	Tiempo, esfuerzo, numero de defectos de código encontrados…	Costo, rentabilidad, estabilidad…
Recursos	***Internos***	***Externos***
Personal	Edad, pago…	Productividad, experiencia, inteligencia…
Equipos	Tamaño, nivel de comunicación, estructuralidad…	Productividad, calidad…
Software	Precio, tamaño…	Usabilidad, confiabilidad
Hardware	Precio, velocidad, tamaño de memoria…	Confiabilidad…
Oficinas	Tamaño, temperatura, iluminación…	Comodidad, calidad…

2.3 Métodos de estimación

Hay una fundamental dificultad en calcular la precisión de diferentes métodos de estimación de costo. Como ya se ha mencionado la estimación es usada para definir el presupuesto del proyecto. También se ha dicho que las organizaciones necesitan hacer estimaciones de costo y esfuerzo del software. Para hacer esto son necesarios métodos. A continuación se listan varios métodos y su clasificación, desde el punto de vista de diferentes autores.

Ian Sommerville en su libro (Sommerville, 2007) clasifica los métodos de estimación como se muestra en la Tabla 4

Tabla 4. Métodos de estimación según Ian Sommerville

MÉTODO	DESCRIPCIÓN
Modelado algorítmico de costos	Un modelo es desarrollado usando información de costos históricos que relaciona alguna métrica de software (usualmente su tamaño) al costo del proyecto. Una estimación está hecha de esa métrica y el modelo predice el esfuerzo requerido.
Juicio experto	Varios expertos sobre las técnicas de desarrollo de software propuestas y el dominio de aplicación son consultados. Cada uno de ellos estima el costo del proyecto. Estas estimaciones son comparadas y discutidas. El proceso de estimación itera hasta que una estimación en común es alcanzada.
Estimación por analogía	Esta técnica es aplicable cuando otros proyectos en el mismo dominio de aplicación han sido completados. El costo de un nuevo proyecto es estimado por analogía con estos proyectos completados. Myers (Myers, 1989) dio una descripción muy clara de este enfoque.
Ley de Parkinson	Ley de Parkinson afirma que el trabajo se expande hasta llenar el tiempo disponible. El costo está determinado por los recursos disponibles y no por evaluación objetiva. Si el software tiene que ser entregado en 12 meses y 5 personas están disponibles, se estima que el esfuerzo requerido es de 60 meses-persona (12 meses por 5 personas= 60 meses-persona).
Ofertando para ganar	El costo del software es estimado para hacer que como sea el cliente esté dispuesto a gastar en el proyecto. El esfuerzo estimado depende del presupuesto del cliente y no de la funcionalidad del sistema.

Dentro del desarrollo ágil, Mike Cohn en su libro (Cohn, 2006) describe los tres métodos más comunes para estimación:

- Opinión de un experto
- Analogía

- Desagregación

Las técnicas pueden ser usadas de manera independiente, pero estas deberían ser combinadas para mejores resultados, la combinación de estas tres técnicas se le llama *planificación de poker* (Grenning 2002), ver Tabla 5.

Tabla 5. Métodos de estimación para desarrollo ágil

MÉTODO	DESCRIPCIÓN
Opinión de un experto	Esta técnica consiste en preguntar a un experto. Al preguntarle a un experto se gasta menos tiempo durante la estimación. Este método es menos usado en proyectos ágiles, en comparación con proyectos tradicionales.
Analogía	Se comparan los requerimientos actuales con uno o más desarrollados anteriormente, para estimar su costo de desarrollo.
Desagregación	Esta técnica consiste en subdividir un requerimiento grande, en sub requerimientos más pequeños, y cada parte estimarla por separado, la estimación total será la suma de las estimaciones individuales.
Puntos de Historias de Usuario	Combina las tres anteriores. Se realiza en grupo.

Desde otro punto de vista M. A. Parthasarathy en (Parthasarathy, 2007) clasifica los métodos de estimación de software bajo dos enfoques:

- Heurísticos
- Paramétricos

Métodos heurísticos: los enfoques heurísticos, por definición, son las prácticas a través de las cuales los profesionales experimentan y encuentran soluciones a los problemas frecuentes. Las personas que utilizan los enfoques heurísticos han experimentado con varios patrones para resolver el problema de la estimación de proyectos de software. Entre los enfoques heurísticos se encuentran los siguientes métodos de estimación. Tabla 6.

Tabla 6. Métodos de estimación con un enfoque heurístico.

MÉTODO	DESCRIPCIÓN
Basado en expertos	Para realizar la estimación los expertos en base a su experiencia determinan los costos.
Método de analogía	Usa la experiencia de proyectos pasados, comparándolos con el proyecto a desarrollar.
Método Bottom-Up	En este método primero se identifican cada uno de todos los componentes, se estiman por separado y para obtener la estimación total se combinan las estimaciones individuales.
Método Top-Down	En este método se trabaja sobre módulos principales, después en sub-módulos y las funciones individuales.
Método algorítmico	Basado en patrones observados por expertos, dichos patrones son representados por fórmulas matemáticas.

Métodos paramétricos: la estimación paramétrica aproxima el volumen del software entregado un pronóstico que puede ser más fácilmente determinado en fases tempranas del ciclo de vida del software, llamado una métrica, ver Tabla 7.

Tabla 7. Métodos de estimación con un enfoque paramétrico.

MÉTODO	DESCRIPCIÓN
Modelo del ciclo de vida de Larry Putnam´s (SLIM)	Este método está basado en el modelo Rayleigh, que describe el personal necesario al desarrollar proyectos complejos. Fue desarrollada para estimar los costes de los grandes proyectos de software utilizando una ecuación, mejor conocida como la ecuación del software. Para proyectos pequeños la ecuación se debe ajustar. La ecuación básica es: $T = C \times E^{1/3} \times t_d^{4/3}$ Donde T es el tamaño en LDC, C es un factor que depende del entorno, E es el esfuerzo en personas-año, y t_d es el tiempo para completar el proyecto, medido en años (Piattini Velthuis & Calvo-Manzano Villalón, 2004).
SEER-SEM	Es una herramienta de estimación desarrollado por Galorath, Inc. basada en el Modelo Matemático de Estimación de Software desarrollado por Dr. Randall W. Jensen. SEER-SEM usa un algoritmo propietario.
Estimador SELECT	Basado en el modelo ObjectMetrix desarrollado por Object Factory. ObjectMetrix funciona midiendo el tamaño de un proyecto contando y clasificando los elementos software inherentes al mismo. Comienzan con una métrica básica del esfuerzo en personas-días requeridos típicamente para desarrollar un elemento de un proyecto. Este esfuerzo asume todas las actividades de un proceso normal de desarrollo de software, pero no tiene en cuenta nada relativo a las características del proyecto o la tecnología empleada. El estimador SELECT adapta la estimación ObjectMetrix refinando los calificadores y los factores de tecnología.
COCOMO II	COCOMO 81 y COCOMO II (actualización de COCOMO 81) son modelos algorítmicos de costos, lo que significa que utilizan fórmulas matemáticas para predecir el costo del proyecto, estimando el tamaño del proyecto, el número de ingenieros de software y otros factores de productividad y de proceso, este modelo fue propuesto por Barry Boehm en 1997 y en año 2000 fue reajustado.
COSMIC-FFP	Basado en el modelo COSMIC (noviembre 1999) de estimación de costes basados en la funcionalidad. Desde que los puntos función son considerados más útiles y problemáticos en el dominio del software en tiempo real, se han realizado nuevos esfuerzos en la estimación basada en la funcionalidad, fruto de ellos es la técnica de los Puntos Función Completos (FFP) que son una medida específicamente adaptada al software integrado y en tiempo real.
Puntos de Función	Desarrollado por Allan Albrecht. Mide la funcionalidad que será entregada a un usuario final, con base en características funcionales bien definidas de un sistema de software. Estas características son: *entradas externas*, información que es entregada al sistema (cajas de texto, botones, transacciones lógicas); *salidas externas*, información que es procesada por el sistema y liberada (reportes); *consultas externa*, información mostrada que no fue procesada (reportes); *archivos lógicos internos*, información que es procesada y almacenada en el sistema (información de un grupo de usuarios) y *archivo de interface externo*, información que es mantenida fuera del sistema pero es necesaria para satisfacer un requerimiento en particular de un proceso (interfaces con otros sistemas) (Garmus & Herron, 2001). A dichas características se les asigna una puntuación ya definida de un conjunto de tablas (Parthasarathy, 2007). Al realizar las operaciones determinadas por la metodología obtenemos la estimación de costo.
Plan de conocimiento	De investigación de la productividad del software.
Puntos de Casos de Uso	Fue desarrollado en 1993 por Gustav Karner, bajo la supervisión de Ivar Jacobson (creador de los casos de uso). El método utiliza los actores y casos de uso. A los casos de uso se les asigna una complejidad basada en transacciones, entendidas como una interacción entre el usuario y el sistema, mientras que a los actores se les asigna una complejidad basada en su tipo, es decir, si son interfaces con usuarios u otros sistemas. También se utilizan factores de entorno y de complejidad técnica para ajustar el resultado, para el cálculo se procede de forma similar a Puntos de Función, (Gómez, 2013).

También Jack T. Marchewka en (Marchewka, 2013) realiza una clasificación de los métodos de estimación, los métodos de la Tabla 8, son llamados *Métodos Tradicionales de Estimación*. Y en la Tabla 9 se encuentran *los Métodos de estimación del esfuerzo de desarrollo de software*:

Tabla 8. Métodos Tradicionales de Estimación, clasificación de Jack T. Marchewka

MÉTODO	DESCRIPCIÓN
Suposición (guesstimating)	Se realizan estimaciones al aire, basadas en presentimientos y no en datos específicos. Administradores de proyectos sin experiencia, tienden a hacer esto ya que es más fácil y rápido. Como sabemos este tipo de prácticas en lugar de beneficiar el desarrollo del proyecto lo pone en gran riesgo.
Técnica Delphi	Esta técnica consiste en realizar sesiones round-robin entre múltiples expertos y una vez que han llegado a un consenso de manera anónima se exponen los resultados a los que llegaron.
Time-Boxing	Consiste en asignar una cantidad (box) de tiempo específico a alguna tarea en particular.
Estimación Top-Down	En este método se trabaja sobre módulos principales, después en sub-módulos y las funciones individuales.
Estimación Bottom-Up	En este método primero se identifican cada uno de todos los componentes, se estiman por separado y para obtener la estimación total se combinan las estimaciones individuales.

Tabla 9. Métodos de estimación del esfuerzo de desarrollo de software, clasificación de Jack T. Marchewka

MÉTODO	DESCRIPCIÓN
Líneas de Código	Trata de estimar la cantidad de líneas de código que serán necesarias para completar el proyecto, los LOC brindan una idea del tamaño, pero estos no consideran la complejidad, restricciones o influencias.
Puntos de Función	Como ya mencionamos anteriormente fueron propuestos por Allan Albrecht en 1979. Estos relacionan el tamaño de un proyecto con su funcionalidad. Los puntos de función son muy usados ya que son independientes de la tecnología o lenguaje de programación, esto permite que un sistema pueda ser comparado con otro.
COCOMO	Constructive COst MOdel estima el esfuerzo (persona/mes) y duración, necesarios para desarrollar un sistema de software utilizando una ecuación. Este método subdivide a los sistemas en tres categorías, orgánicos, embebidos y semi-desconectados.
Heurísticos	Este tipo de métodos se basan en la premisa de que las mismas actividades se estarán repitiendo en más proyectos, por lo tanto la estimación se hace en base a la experiencia.

Otro autor reconocido por ser el creador del proceso personal de software PSP, Watts S. Humphrey en su libro (Humphrey, A Discipline for Software Enginnering, 1995) habla sobre los cuatro métodos de estimación de software más populares por su utilidad y en los cuales está fundamentado el método de estimación basado en proxy, método PROBE, ver Tabla 10.

Tabla 10. Métodos de estimación más populares descritos por Watts S. Humphrey

MÉTODO	DESCRIPCIÓN
Método Wideband-Delphi	Fue originado por Rand Corporation y refinado por Boehm. Varios ingenieros producen estimaciones individuales y después usan la técnica Delphi para llegar a un consenso en la estimación.
Método Fuzzy-Logic	Este método fue definido por Putnam. El proceso de estimación del tamaño está basado en datos históricos de proyectos anteriores.
Método Standard-Component	También este método es descrito por Putnam, utiliza la organización de datos históricos para hacer progresivamente estimaciones de tamaño más refinadas.
Puntos de Función	Nuevamente se hace mención de esta metodología, que relaciona los requerimientos funcionales con el tamaño del sistema.
Método PROBE	Este método fue desarrollado por Watts S. Humphrey y es usado en PSP para la estimación de los proyectos. Utiliza datos históricos. Un proxy ayuda a relacionar la funcionalidad con el tamaño.

Emilia Mendes explica que dentro de la estimación de proyectos web, los cuales cada vez son más comunes, podemos encontrar 3 principales clasificaciones (Mendes, 2008):

- Estimación del esfuerzo basado en expertos
- Modelos algorítmicos
- Técnicas de inteligencia artificial

Ya se ha hablado que los *métodos de estimación basado en expertos*, determina el costo de estimación basado en experiencia y conocimientos adquiridos en proyectos previos.

Los *métodos algorítmicos*, construyen un modelo matemático el cual relaciona el esfuerzo con uno o más características del proyecto se asume que el tamaño del proyecto es el mayor contribuyente al esfuerzo. Generalmente la relación entre el tamaño y el esfuerzo es mapeado a una ecuación. En la Tabla 11, se muestra el método más popular de estimación en desarrollos web (Mendes, 2008).

Tabla 11. Método algorítmico de estimación más utilizado en desarrollos web

MÉTODO	DESCRIPCIÓN
COCOMO	Como ya se ha mencionado en las tablas anteriores. Este método también es muy utilizado en desarrollos web debido a su alta adaptabilidad a las necesidades de los proyectos. Su primera edición fue en 1981 y se ha ido ajustando a las tendencias de desarrollo de software, COCOMO II (2000) es la versión más reciente y actualizada de este modelo.

Técnicas de inteligencia artificial: en la última década, han sido usadas como complemento o como alternativa a las dos categorías anteriores (Mendes, 2008), en la Tabla 12 se muestran ejemplos de este tipo de métodos.

Tabla 12. Métodos de estimación que utilizan técnicas de inteligencia artificial

MÉTODO	DESCRIPCIÓN
Fuzzy Logic	Mencionado anteriormente.
Clasificación y Arboles de Regresión (CARTs)	Utiliza diferentes variables para construir arboles binarios, cada nodo representa la categoría que hay que estimar o que ya ha sido estimada. Para hacer la estimación se recorre el árbol.
Redes Neuronales	Tratan de automatizar las mejoras en el proceso de estimación construyendo modelos que "aprenden" de la experiencia previa.
Razonamiento Basado en Casos (CBR)	Se basa en la premisa de que problemas similares proveen soluciones similares. Se compara el proyecto actual contra información histórica.

A lo largo de este capítulo, se describieron los cuatro principales factores que intervienen en el proceso de estimación: alcance, ambiente, experiencia y herramientas. La contribución de los cuatro factores es igualmente significativa y juega un papel importante en la estimación final del proyecto. También se habló sobre la diferencia entre estimar y medir, estimar es una actividad que se realiza antes de que un producto comience a tomar forma, en el contexto del desarrollo de software, la estimación se realiza una vez que se tiene información del proyecto que se va a realizar. Por otra parte la medición se realiza en las etapas siguientes de desarrollo, en las cuales el proyecto ya cuenta con atributos que son susceptibles de ser medidos como el tamaño de la aplicación, tipo de ejecución del proyecto, las habilidades/capacidades del equipo de desarrollo, la calidad de los procesos y herramientas, la plataforma sobre la cual la aplicación es desarrollada, entre muchos otros más. El resto del capítulo se centró en describir diferentes métodos de estimación y las clasificaciones de estos, en la opinión de diversos autores, entre los métodos descritos tenemos: Método del Análisis de Puntos de Función, Puntos de Función Mark II, COCOMO, COSMIC-FFP, Puntos de Casos de Uso, Basadas en la opinión de un experto, Analogías, Desagregación, Método Bottom-Up, Método Top-Down, SLIM, SEER-SEM, Método Delphi, Time-Boxing, Fuzzy-Logic, PROBE, Clasificación y Arboles de Regresión (CARTs), Razonamiento Basado en Casos (CBR) y Puntos de Historias de Usuario.

Todos los métodos descritos anteriormente serán el punto de referencia del cual se iniciará el proceso de búsqueda para encontrar entre esos métodos uno o algunos que por sus características ayuden a estimar el esfuerzo de desarrollo de los micro-proyectos de software, obteniendo con esto resultados más cercanos al esfuerzo real de desarrollo.

Capítulo 3. Aproximación de la solución

En el presente capítulo se describen los métodos que se utilizaron para llevar de la mano la investigación. El método guía seleccionado fue EBSE, el cual básicamente consiste de cinco pasos. En los tres primeros pasos de EBSE se utilizó como auxiliar el método SLR, para recabar información de calidad. El resto del capítulo se centra en el cuarto paso del proceso EBSE y en él se abordan los Métodos de Estimación más utilizados así como sus características, una vez sintetizada esa información fueron seleccionados 5 métodos de estimación los cuales se aplicaron a una muestra de micro-proyectos, estos datos fueron registrados y se muestran al final del capítulo. La comparación de los resultados obtenidos por los Métodos de Estimación elegidos y los datos de desarrollo reales de cada proyecto, se realizó en el Capítulo 4.

Para desarrollar este trabajo se ha elegido la metodología EBSE, *Evidence-Based Software Engineering* (Ingeniería de Software Basada en Evidencias). El propósito de EBSE es mejorar las decisiones tomadas relacionadas con el desarrollo del software y su mantenimiento por recolección y evaluación de las mejores evidencias desde estudios de investigación y experiencia basada en la práctica.

El objetivo principal de EBSE es ser para los profesionales una guía para proveer soluciones de Ingeniera del Software apropiadas en contextos específicos.

Los principales pasos en EBSE son los siguientes (Kitchenham, et al., 2009):

1. Convertir un problema relevante o necesidad de información a una pregunta respondible.
2. Buscar la literatura de la mejor evidencia disponible para responder la pregunta.
3. Evaluar críticamente la evidencia para su validación, impacto y aplicabilidad.
4. Integrar la evidencia evaluada con la experiencia práctica y la valoración de los clientes y circunstancias para hacer decisiones sobre la práctica.
5. Evaluar la interpretación en comparación con interpretaciones previas y buscar formas de mejorarla.

Fue seleccionado el método EBSE porque se utiliza para tomar decisiones en el contexto de desarrollo del software. Para este trabajo de investigación fue necesario elegir el método de estimación que mejor se ajusta a los micro-proyectos de software, ese tipo de decisión tiene que ver con cuestiones de desarrollo del software. Para el desarrollo de los tres primeros pasos se realizó una SLR, *Systematic Literature Review* (Revisión Sistemática de Literatura) (Kitchenham, et al., 2007).

Las series de Revisiones Sistemáticas de Literatura sobre estimación de costos, demuestran el valor potencial de EBSE para sintetizar evidencia y hacer disponibles los resultados para los profesionales (Kitchenham, et al., 2009).

3.1 Convertir un problema relevante o necesidad de información a una pregunta respondible.

Dentro del capítulo introductorio de este documento fue realizado este paso y las preguntas de investigación formuladas fueron:

1. ¿Cuáles son los métodos de estimación más utilizados hoy en día?
2. ¿Cuáles son las características comunes y diferencias entre esos métodos?
3. ¿Existe un método de estimación que por sus características se adecúa más a los proyectos de software de corta duración?

4. ¿Los micro-proyectos de software tienen características especiales que requieren métodos de estimación muy específicos?

3.2 Buscar la literatura de la mejor evidencia disponible para responder las preguntas.

Consiste en realizar una selección de la literatura a ser consultada durante la investigación. Para este segundo paso del proceso EBSE existe una guía de SLR (Kitchenham, et al., 2007) la cual fue seguida por ser uno de los principales métodos de síntesis para el proceso de investigación (Kitchenham, et al., 2009). Una SLR aporta preguntas que nos ayudan a guiarnos durante la investigación. Ayuda en el análisis de datos, captura de información y divulgación de los resultados, siendo el reporte final de SLR en dos formatos: un capítulo de tesis o un artículo.

Las ventajas de las revisiones sistemáticas de literatura (SLR, por sus siglas en inglés) son (Kitchenham, et al., 2007):

- La metodología bien definida la hace ser menos propensa a que los resultados de la literatura estén sesgados, aunque no protege a los estudios primarios de publicaciones dudosas.
- Ellas pueden proveer información acerca de los efectos de algún fenómeno a través de un amplio rango de configuraciones y métodos empíricos. Si los estudios dan resultados consistentes, la revisión sistemática provee evidencia de que el fenómeno es robusto y transferible. Si los estudios dan resultados inconsistentes, las fuentes de variación pueden ser estudiadas.
- En el caso de estudios cuantitativos, es posible combinar datos usando técnicas de meta-análisis. Esto incrementa la posibilidad de detectar efectos reales que estudios individuales más pequeños insuficientes para detectarlos.

La mayor desventaja de las revisiones sistemáticas de literatura es que ellas requieren un considerable mayor esfuerzo que las revisiones tradicionales de literatura. Además, el incremento de esfuerzo para el meta-análisis puede también ser una desventaja, debido que es posible detectar pequeños sesgos.

Etapas SLR

Kitchenham resume las etapas de la revisión sistemática en tres: Planear la Revisión, Conducir la Revisión y Reportar la Revisión. Las partes básicas que contiene una SLR son (Kitchenham, et al., 2007):

- Antecedentes o introducción
- Preguntas de investigación
- Procesos de búsqueda
    - Fuentes de información
    - Criterio de inclusión
    - Criterios de exclusión
- Proceso de selección de estudios primarios
- Calidad de la evaluación de la fuentes de información
- Recolección de datos
- Análisis de datos
- Difusión

3.2.1 Antecedentes y preguntas de investigación

En el capítulo de introducción se abarcaron las dos primeras partes de la SLR antecedentes o introducción, y preguntas de investigación. Las preguntas de investigación fueron mencionadas nuevamente en el primer paso del proceso EBSE. Esta sección corresponde al segundo paso del proceso EBSE y abarca el Proceso de búsqueda de la metodología SLR.

3.2.2 Proceso de búsqueda

De las preguntas que fueron formuladas anteriormente se identificaron las siguientes palabras clave:

- *Importance of the Estimation (Importancia de la Estimación)*
- *Estimation Methods (Métodos de Estimación)*
- *Software´s Metrics*
- *Cost Estimation*
- *Micro-software projects (Micro-proyectos de software)*
- *Small software projects*
- *Software small- to medium-sized enterprises*
- *Cost of software projects (Costos de los proyectos de Software)*
- *MiPYMES of software development (MiPYMES que desarrollan software)*
- *Software Engineering (Ingeniería del Software)*

La información fue seleccionada con base en las palabras claves anteriores, las cuales fueron la pauta para seleccionar los libros, artículos, conferencias, etc. que contenían dichos términos.

3.2.2.1 Fuentes de información

El proceso SLR recomienda la búsqueda en diversas fuentes, por lo tanto para realizar la investigación se utilizaron las bases de datos

- ACM Digital Library
- IEEE Xplore
- Spring Verlag
- Google Scholar
- ScienceDirect
- Metapress
- Wiley IntersSience

De las bases de datos mencionadas anteriormente se buscaron libros, artículos, reportes, etc. que contuvieran las palabras claves seleccionadas. Además de búsquedas electrónicas se realizaron búsquedas en medios impresos.

3.2.*2.2 Criterios de inclusión*

Los artículos, libros, revistas que se incluyeron en el estudio fueron aquellos que datan del año 2000 al año actual, ya que se deseó información reciente. Los criterios que se tomaron en cuenta fueron:

- La fuente debe presentar información sobre los métodos de estimación más utilizados.
- Las características de los métodos de estimación.
- Información comparativa entre metodologías de estimación.
- Presentar información estadística sobre la efectividad de las metodologías de estimación.
- Respondan directamente a una o más de las preguntas de investigación.

- Contenga información de los problemas o necesidades de las MiPYMES desarrolladoras de software.

3.2.2.3 Criterios de exclusión

Las fuentes excluidas fueron:

- Literatura informal, como diapositivas, talleres, opiniones, reporte y ensayos no publicados.

3.3 Evaluar críticamente la evidencia para su validación, impacto y aplicabilidad

En este paso de EBSE se realizaron las siguientes fases de SLR: Proceso de selección de estudios primarios, Calidad de la evaluación, Recolección de datos, Análisis de datos y la definición de la Difusión.

3.3.1 Proceso de selección de estudios primarios

Los primeros filtros de selección fueron las palabras clave identificadas en la sección de palabras clave del documento, estas deben estar presentes en el título y resumen o introducción del artículo, libro, etc.

La selección preliminar de los estudios primarios se basó inicialmente en la revisión del título, resumen y palabras clave. Esta búsqueda posteriormente se amplió para incluir la revisión del contenido de la introducción y conclusiones en los casos donde el título, resumen y palabras clave no proveían suficiente información.

Los estudios no seleccionados bajo los criterios anteriores, se mantuvieron en una lista de posibles candidatos.

3.3.2 Calidad de la evaluación

La evaluación de la calidad de la información fue sujeta a las siguientes preguntas:

- ¿Los criterios de la revisión se describen y son apropiados?
- ¿La literatura investigada cubre aproximadamente todos los estudios relevantes?
- ¿Los estudios básicos incluidos fueron validados?
- ¿Los estudios fueron adecuadamente descritos?

3.3.3 Recolección de datos

La extracción de los datos para cada fuente de información fue:

- La fuente
- El año de publicación
- Autor y organización
- Hechos o preguntas de investigación sobre la estimación de desarrollos de software
- Métodos de estimación presentados

- Características de los métodos de estimación
- Información comparativa entre métodos de estimación
- Estadísticas sobre el uso de los métodos de estimación
- Información estadísticas sobre MiPYMES desarrolladoras de software
- Información sobre la relevancia de buenas estimaciones

3.3.4 Análisis de datos

Los datos son mostrados más adelante en una tabla comparativa sobre los métodos de estimación encontrados. La tabla muestra **las características comunes y diferentes** de los métodos analizados y las características de cada uno de ellos que se tomaron en cuenta para aplicarse.

3.3.5 Difusión

Los resultados del estudio serán de interés para la comunidad de Ingeniería del Software interesados por la estimación de micro-proyectos de software, los resultados se difundirán a través de un artículo escrito para un congreso internacional y/o su publicación en revistas científicas del área.

3.4 Integrar la evidencia evaluada con la experiencia práctica y la valoración de los clientes y circunstancias para hacer decisiones sobre la práctica

Una vez terminada la revisión sistemática de la literatura, SLR, se contó con las fuentes de información más relevantes e indispensables con las cuales se abordaron los siguientes puntos.

1. Métodos de Estimación
2. Métodos de Estimación más comunes y utilizados
3. Aspectos que influyen en la selección de un método
4. Selección de los Métodos de Estimación a estudiar
5. Preparación de la muestra
6. Aplicación de los Métodos
7. Recopilación de datos

La inclusión de la experiencia y la práctica se vieron reflejadas del punto 4 al 7.

3.5 Evaluar la interpretación en comparación con interpretaciones previas y buscar formas de mejorarla

Este es el último paso del proceso EBSE y las actividades que se realizaron fueron:

1. Síntesis de los datos
2. Determinación de objetivos alcanzados
3. Comprobación de la hipótesis

Como se mencionó al inicio, una vez explicados todos los pasos de la metodología de investigación, a partir de este punto, el resto del capítulo se centra en el cuarto paso del proceso EBSE, y en él se abordan los Métodos de Estimación más utilizados así como sus características. Una vez sintetizada esa información se

seleccionaron 5 métodos de estimación, los cuales se aplicaron a una muestra de micro-proyectos, estos datos fueron registrados y se muestran al final del capítulo.

Recordemos que el cuarto paso del procesos EBSE indica que se debe integrar la evidencia evaluada con la experiencia práctica y la valoración de los clientes y circunstancias, para tomar decisiones sobre la práctica. Respecto a lo anterior, para proseguir fue necesario terminar la revisión sistemática de literatura, SLR, la cual nos proporcionó las fuentes de información más relevantes e indispensables para abordar los siguientes puntos.

1. Métodos de Estimación
2. Métodos de Estimación más comunes y utilizados
3. Aspectos que influyen en la selección de un método
4. Selección de los Métodos de Estimación a estudiar
5. Preparación de la muestra
6. Aplicación de los Métodos
7. Recopilación de datos

La inclusión de la experiencia y la práctica se ven reflejados del punto 4 al 7.

3.6 Resultados de la SLR

Al seguir los pasos marcados por la metodología SLR se reunió un compendio de artículos y libros, electrónicos e impresos, así como también sitios de internet, los cuales cumplieron los criterios de inclusión especificados, para posteriormente ser utilizados como fuente de información para abordar los puntos del 1 al 4 de lo anterior, dicho compendio se describe a continuación.

- Se seleccionaron 22 artículos en formato electrónico provenientes de la IEEE
- 28 artículos de ScienceDirect en formato electrónico
- 10 artículos en formato electrónico de JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION
- 16 artículos electrónicos de diversas fuentes
- 5 reportes de Standish Group, en formato electrónico
- SWEBOK 2004 y 2014, en formato electrónico
- 9 libros electrónicos
- 4 libros impresos
- 6 sitios de internet

En total se recabaron 102 fuentes bibliográficas diferentes. Recordemos que de todas las fuentes seleccionadas, solo algunas proporcionaron información relevante y sustancial para la investigación, por tal motivo únicamente esas se verán citadas en la bibliografía de este documento de tesis.

Tras un arduo proceso de revisión y análisis del compendio de fuentes de información, se sintetizaron en una tabla los métodos de estimación de desarrollo de software más conocidos. Esa información ayudó a preparar la muestra de micro-proyectos que se analizó y contribuyó en la selección de los 5 métodos que se aplicaron a la muestra.

3.7 Análisis de métodos de estimación

Esta sección presenta de manera tabular las características de los métodos de estimación de esfuerzo de desarrollo de software más conocidos, cuya información fue encontrada en la bibliografía obtenida como resultados de una SLR, la Tabla 13 es el resultado y describe a los métodos de manera resumida, para mayor detalle del proceso pueden consultarse los Anexos A, B, C , D, E, F, G, H e I correspondientes a cada método, hay que aclarar que existen métodos cuya complejidad es baja, es decir, no requieren tantos pasos, por tal motivo no se incluye algún Anexo, ya que se considera que los pasos mostrados deben ser suficientes para poner en práctica el método.

La Tabla 13 descompone cada método de estimación, en la métrica que utiliza, su información de entrada, información de salida y el proceso o pasos del método.

Del análisis y síntesis de las características de los diferentes métodos de estimación conocidos, se derivó información sobre las ventajas y desventajas entre métodos, dicha información se muestra en la Tabla 14.

Tabla 13. Análisis de métodos de estimación

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
Análisis de Puntos de Función	Puntos de función	Interfaces gráficas , requerimientos de usuario, reportes, archivos o tablas de bases de datos	Puntos de función ajustados	1. Cálculo de los puntos de función sin ajustar (UFC, Unadjusted Function point Count). 2. Calculo del factor técnico de complejidad (TCF, Technical Complexity Factor) 3. Resultado de los puntos de función ajustados FPA=UFC*TCF Ver Anexo A (Longstreet, August 2002).
Wideband-Delphi	LOC	Especificaciones del programa y un formulario de estimación.	Estimación de tamaño en LOCs	La idea fundamental es usar varios expertos que hacen estimaciones independientes y luego hacerlos converger hacia una estimación única. 1. Cada experto recibe las especificaciones del programa y un formulario de estimación. 2. Se reúnen a conversar sobre suposiciones, dudas, etc. 3. Cada uno lista las tareas y produce una estimación. 4. Las estimaciones son recogidas por un moderador quien tabula los resultados y los devuelve a los expertos (estimaciones, promedio, mediana, etc.). 5. Los expertos se reúnen nuevamente y discuten las tareas. 6. Se vuelve a la tercera etapa (nueva estimación). La discusión entre los expertos a menudo clarifica aspectos y producen cambios en las estimaciones para la etapa siguiente. El método produce estimaciones bastante precisas (Humphrey, A Discipline for Software Enginnering, 1995).

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
Lógica Difusa (Fuzzy-Logic)	LOC	Datos históricos de productos previos	Tamaño en LOCs	1. Se construye una tabla (escala de tipo logarítmica) con rangos y sub-rangos de tamaño de proyectos previos. 2. Por ejemplo si el programa más pequeño es de 104 LOC y el más grande de 17,243 LOC. ***Tamaño/Complejidad*** ***bajo*** ***medio*** ***alto*** *muy pequeño* *104* *173* *288* *pequeño* *288* *481* *802* *mediano* *802* *1338* *2230* *grande* *2230* *3719* *6202* *muy grande* *6202* *10341* *17243* 3. Las filas indican el tamaño (en LOC), y las columnas la complejidad de los programas (o sistemas). 4. El estimador selecciona una de las categorías y subcategorías, comparando el nuevo proyecto con los proyectos anteriores dentro del rango considerado. 5. Luego toma como referencia (principalmente) aquellos proyectos previos que están en esa categoría. (Humphrey, A Discipline for Software Enginnering, 1995)
Método de componentes estándar	LOC	Tamaño en SLOC de componentes usados en proyectos previos, en varios niveles de abstracción: subsistemas completos, módulos, interfaces de usuario, archivos, instrucciones objeto, reportes, programas interactivos, programas por lotes, etc.	Tamaño en LOCs	1. Se estima cuántos de cada uno de los componentes habrá en el nuevo proyecto (estimado, máximo y mínimo). 2. Se combina esto, ponderando 4 veces el más probable (M) y una vez los máximos (S) y mínimos (L): X= (S + 4*M + L)/6. (Humphrey, A Discipline for Software Enginnering, 1995)
COCOMO	KDSI(miles de instrucciones fuente liberadas), Meses-persona y Meses	Tamaño medido en KDSIs. Modo de sistema (orgánico, semi-empotrado o empotrado). Modelo de COCOMO (básico, intermedio o detallado)	Esfuerzo en meses-persona. Duración en meses.	El modelo COCOMO nos permite calcular el esfuerzo requerido para desarrollar el proyecto en persona-mes y también permite calcular la duración del mismo en meses. **COCOMO: Esfuerzo** COCOMO se divide en tres modelos: básico, intermedio y detallado. COCOMO también hace distinción entre el modo de desarrollo de sistemas. Sistema Orgánico, sistema Empotrado y sistema semi-empotrado. En base a lo anterior se seleccionan los valores de cada parámetro de la fórmula para calcular el esfuerzo. Los tres utilizan la misma fórmula (El modelo COCOMO, 2013) (Fenton & Pfleeger, 1998): $$E = aS^b F$$ *E*- esfuerzo en persona-mese *S*- tamaño medido en KDSI *F*- factor de ajuste, Anexo B (tabla de manejadores de costos) *a* y *b*- dependen del tipo de sistema, Anexo B (tablas de valores) **COCOMO: Duración**

				Para calcular la duración de un proyecto se utiliza la siguiente fórmula (El modelo COCOMO, 2013) (Fenton & Pfleeger, 1998): $D = aE^b$ *E*- esfuerzo en persona-mes *a* y *b*- dependen del tipo de sistema, Anexo B (tabla de valores)

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
COCOMOII	KDSI(miles de instrucciones fuente liberadas), Meses-persona y Meses	Tamaño medido en KDSIs. Sub-modelo de desarrollo (aplicación-composición, diseño temprano, reúso o post arquitectura)	Esfuerzo en meses-persona. Duración en meses	El modelo COCOMO II nos permite calcular el esfuerzo requerido para desarrollar el proyecto en meses-persona y también permite calcular la duración del mismo en meses. 1. Entrada del desarrollo de software (requerimientos y sub-modelo COCOMO II a utilizar) 2. Calcular el valor del factor de ajuste del esfuerzo M (ver Tabla de factores de ajuste) 3. Calcular el exponente de esfuerzo (B) y el exponente de calendarización (D) (ver tabla de factores de escala) 4. Encontrar el tamaño del software KSLOC (miles de líneas de código fuente) 5. Calcular el esfuerzo en meses-persona, con la ecuación de esfuerzo $Esfuerzo = 2.94 * Tamaño^B * M$ 6. Calcular la duración en meses, con la ecuación de duración $Tiempo = 3.67 * Esfuerzo^D$ 7. Calcular el personal promedio necesario para el proyecto *Personal-Promedio= Esfuerzo / Tiempo* Ver Anexo C.
SLIM	LOC, persona-años, años	Tamaño en LOCs	Esfuerzo total y tiempo de entrega	Este modelo se expresa en dos ecuaciones que describen la relación entre el esfuerzo de desarrollo y el tiempo de entrega. La primera ecuación, llamada *"ecuación del software"*, relaciona el tamaño, S (en líneas de código) con varias variables: un factor tecnológico, C, el esfuerzo total medido en persona-años, K, y el tiempo transcurrido hasta la entrega, T medido en años. La relación se expresa como sigue: $Tamaño = CK^{\frac{1}{3}}T^{\frac{4}{3}}$ El método incluye otra ecuación $D_0 = \frac{K}{T^3}$ D_0 es una constante llamada **aceleración de mano de obra,** este valor dependerá del tipo de proyecto. Usando estas dos ecuaciones, se puede obtener el esfuerzo o duración (Parthasarathy, 2007). $K = \left(\frac{S}{C}\right)^{\frac{9}{7}} D_0^{\frac{4}{7}}$
Método COSMIC-FFP	Puntos de Función COSMIC, Movimientos de Datos	Requerimientos Diseño del sistema dividido en niveles	Tamaño funcional del software en puntos de función cosmic	En este método el tamaño funcional del software se obtiene como un valor proporcional al número de movimientos de datos de dicho software. Para ello el software se entiende que está dividido en distintos Niveles (Layer) de forma jerárquica. Cada nivel sólo puede comunicarse con sus niveles inmediatamente superior o inferior. Dentro de cada nivel el software se agrupa en componentes llamados Pares (Peers) teniendo todos ellos el mismo nivel jerárquico. Los Procesos Funcionales se recogen dentro de los distintos Pares. El intercambio de información entre distintos niveles se realiza a través de sus respectivos Procesos Funcionales. Las entidades de una aplicación se asocian en Grupos de Datos. Todo Proceso Funcional debe contemplar al menos dos movimientos de datos: • una Entrada (Entry). • una Salida (Exit) o una Escritura (Write). Un Movimiento de Datos tiene un valor asignado de 1 CFP (Cosmic Function Point). El Tamaño de un Proceso Funcional viene determinado por la suma del tamaño de los movimientos de datos que lo forman. El tamaño de un Módulo de Software en un determinado ámbito se obtiene sumando los tamaños funcionales de los procesos funcionales de los cuales consta dicho módulo de software. La suma total nos dará el tamaño funcional del software que estamos midiendo. (Parthasarathy, 2007) (Gómez, 2013)Ver Anexo D.

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
Método Mark II FPA	Puntos de Función	Requerimientos	Tamaño funcional	En el método de Mark II FPA se miden dos componentes del software: • **Componente funcional.** Es la suma de los tamaños de todas las transacciones lógicas que se identifiquen dentro del alcance. • **Componente técnico.** Que consiste en la evaluación de las características generales de la aplicación (de forma similar al método de Puntos de Función). Esta parte es opcional y generalmente no se utiliza en las mediciones. El tamaño de una **Transacción Lógica** es igual a la suma de los elementos de los componentes que la forman: elementos de entrada, elementos de proceso y elementos de salida. (Gómez, 2013). Ver Anexo E.
Puntos de Caso de Uso	Puntos de caso de uso, Horas/persona	Diagramas de Casos de Uso	Puntos de Casos de Uso Ajustados	Para el cálculo se procede de forma similar a Puntos de Función: se calcula una cuenta no ajustada de Puntos de Casos de Uso (UAUCP), asignando una complejidad a los actores y a los casos de uso. Esta complejidad será ponderada con un Factor de Ajuste Técnico y por un Factor de Ajuste Relativo al entorno de implantación, obteniendo tras ello una cuenta de Puntos de Casos de Uso Ajustados (Gómez, 2013). 1) Clasificar cada interacción entre actor y caso de uso según su complejidad y asignar un peso en función de ésta. 2) Calcular la complejidad de cada caso de uso según el número de transacciones o pasos del mismo. 3) Calcular los Puntos de Casos de Uso No Ajustados (UUCP) del sistema. 4) Calcular los Factores Técnicos (TCF). 5) Calcular los Factores de Entorno. Ver Anexo F.
Analogías	La métrica dependerá de las utilizadas en los proyectos anteriores (LOC, Puntos de Función, Puntos de Caso de Uso, etc.)	Requerimientos del proyecto. Información de proyectos anteriores	Esfuerzo necesario para desarrollar el proyecto.	Para poder realizar una estimación sobre un proyecto de software utilizando la analogía, es necesario en primer lugar determinar cuál o cuáles de los proyectos ya completados y almacenados en una base de datos son más parecidos al que queremos estimar. Para realizar esta tarea, debe determinarse el criterio para establecer las distancias entre proyectos. Existen multitud de medidas de similitud entre ejemplares, siendo las más usadas las distancias de Euclides, de Manhattan y de Minkowski. Al calcular las distancias podemos obtener el conjunto de proyectos más parecidos al que está bajo examen. Lógicamente, este grupo estará formado por los ejemplares con la menor distancia al proyecto en cuestión. Una vez obtenido el conjunto de proyectos similares, sólo falta ajustar los valores de esfuerzo de dichos proyectos para poder estimar el del proyecto en cuestión, entre las técnicas más frecuentes para esta tarea se conoce el Ajuste mediante Algoritmos Genéticos y Ajuste por "regresión hacia la media" (Álvarez Sáez, 2013).
Desagregación (Bottom-Up)	LOC, Puntos de Función, Puntos de Caso de Uso, etc.	Requerimientos del software	Duración, costo, esfuerzo y riesgo.	Esta filosofía de estimación divide el proyecto en pequeñas partes, aplicándole a cada una de estas una evaluación directa del esfuerzo, con alguna técnica de las mencionadas anteriormente. El valor obtenido se normaliza en función del peso y la importancia de cada parte. Posteriormente se irán uniendo estas partes en subsistemas. La principal ventaja que aporta este método, es que las personas que realizan la estimación son las mismas que van a hacer el desarrollo. Sin embargo, tiene el inconveniente de que normalmente no se cuenta el tiempo de estimación, que suele ser importante. Otra desventaja sería que si se descubren errores en las partes finales del proceso, habría que rechazar prácticamente toda la estimación hecha hasta el momento. (Fermín, 2006) (Sommerville, 2007)
Método Top-Down	LOC, Puntos de Función, Puntos de Caso de Uso, etc.	Requerimientos del software	Duración, costo, esfuerzo y riesgo.	En primer lugar se estiman los costos a nivel de sistema, de una manera general, para posteriormente ir obteniendo los costos de cada uno de los diferentes subsistemas identificados. (Sommerville, 2007) (Pressman, Software engineering: a practitioner´s approach, 2001)

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
SEER-SEM	Se (tamaño efectivo)	Tamaño en LOC o tamaño basado en la funcionalidad (Puntos de Función, etc.)	Esfuerzo y duración	SEER-SEM (The System Evaluation and Estimation of Resources - Software Estimating Model) soporta tamaño en líneas de código o en métricas de tamaño basadas en función, como los Puntos de Función. El tamaño en estas métricas se normaliza a Se (tamaño efectivo) la fórmula genérica para hacer esto es: Se=TamañoNuevo + TamañoExistente*(0.4*Rediseño + 0.25*Reimplemetación + 0.35*Retest) En caso de que el tamaño este dado en alguna métrica basada en función la fórmula para obtener el tamaño efectivo es: $Se=Lx * (AdjFactor * UFP)^{(Entropía/1.2)}$ Lx- factor de expansión dependiente del lenguaje de programación. AdjFactor- es el resultado al evaluar otros factores como la fase de estimación, el entorno operativo, tipo de aplicación, y la complejidad de la aplicación. Entropía- rango de 1.04 a 1.2dependiendo del tipo de software desarrollado. UFP- puntos de función sin ajustar. (Fischman, McRitchie, & Galor, 2005) **Esfuerzo** La fórmula utilizada para el cálculo del esfuerzo es la siguiente: $K= D^{0.4}(Se/Cte)^{1.2}$ Se- tamaño efectivo, calculado anteriormente. Cte- tecnología efectiva, una métrica compuesta que captura los factores relativos a la eficiencia o la productividad con la que el desarrollo se puede llevar a cabo. D- complejidad de personal, una calificación de dificultad propia del proyecto en términos de la velocidad a la que el personal se suma a un proyecto. **Duración** La duración puede ser calculada con la fórmula que a continuación se muestra. $t_d = D^{-0.2} (Se/Cte)^{0.4}$
Time-Boxing	Horas	Requerimientos del producto	Tiempo específico en el que se debe terminar el producto.	Este método es muy utilizado en desarrollos ágiles como SCRUM. El proceso comienza con elaborar una lista de tareas o actividades que se tiene que realizar en la semana. Una vez que se tiene la lista de tareas que se desea llevar a cabo en el transcurso de la semana, es necesario estimar el tiempo requerido para cada una de ellas a modo de enmarcarlas en algún espacio de tiempo, para llevar a cabo las actividades clave que acercarán el cumplimiento de las metas y además ayudará a estimar los tiempos requeridos para ello. (Díaz, 2013) Una vez fijados los tiempos de cada actividad se trata de cumplirlas cabalmente evitando las interrupciones. (Scrum Bok, 2013)
PROBE	LOC Proxy	Requerimientos del software	Tamaño del programa en LOC, duración en horas	Este método utiliza datos históricos de proyectos anteriores para realizar la estimación del nuevo proyecto. Inicialmente se realiza una estimación inicial, utilizando el diseño conceptual, del tamaño de los módulos o funciones en las que se dividirá todo el proyecto. Para realizar la estimación de tamaño el método PROBE hace uso de proxies los cuales relacionan la funcionalidad con el tamaño del software. Una vez que se cuenta con los requerimientos del producto, obtenemos datos históricos de tamaño de proyectos anteriores, se realiza un diseño conceptual para dividir el producto en partes; cada parte se compara con partes similares de los datos históricos, si dicha parte no tiene semejanza se vuelve a subdividir. Cuando se seleccionan las partes más semejantes, almacenadas en la base de datos, a las partes del nuevo diseño, se prosigue estimando el tamaño relativo de cada parte utilizando los proxies, para estimar el tamaño estimado de todo el producto se suman los tamaños estimados de cada parte. Mediante regresión lineal utilizando los datos históricos y el tamaño estimado se calcula el tiempo que tomará realizar el producto.

				El método PROBE tiene 4 variantes D, C, B y A, su utilización depende del número de datos históricos con los que se cuente y del valor de unos parámetros que utiliza PROBE A y B (Humphrey, A Discipline for Software Enginnering, 1995), ver Anexo I.

MÉTODO	MÉTRICA	ENTRADA	SALIDA	PROCESO
Clasificación y Arboles de Regresión (CART)	Predictores numéricos o categóricos, Horas-persona	Datos históricos, Requisitos del nuevo proyecto	Esfuerzo horas-persona	CART usa variables independientes (predictores) para construir un árbol binario donde cada nodo hoja representa ya sea una categoría a la que pertenece una estimación o un valor para una estimación. La primera situación se produce con los árboles de clasificación y el segundo se produce con los árboles de regresión, es decir, cada vez que predictores son categóricos, el árbol se denomina un árbol de clasificación, y siempre que los predictores sean numéricos, el árbol se denomina un árbol de regresión. Con el fin de obtener una estimación, uno tiene que atravesar los nodos del árbol de la raíz a la hoja mediante la selección de los nodos que representan la categoría o valores de las variables independientes asociados con el caso a ser estimado (Mendes, 2008). Ver Anexo G.
Razonamiento Basado en Casos (CBR)	Horas-persona	Datos históricos, Requerimientos del nuevo proyecto	Esfuerzo	CBR proporciona estimaciones de esfuerzo para los nuevos proyectos mediante la comparación de las características del proyecto actual a estimar con una biblioteca de datos históricos de proyectos terminados con un esfuerzo conocido (caso base) (Mendes, 2008). 1. Caracterización de un nuevo proyecto *p* para el que se requiere un esfuerzo estimado, con variables (características) comunes a los proyectos terminados almacenados en el caso base. 2. La caracterización se usa como base para la búsqueda de proyectos similares (análogos) completados de los cuales se conoce el esfuerzo. Este proceso se puede lograr mediante la medición de la distancia entre dos proyectos, el proyecto que presenta la menor distancia total es el proyecto más similar a proyectar *p*. Para medir la distancia entre el proyecto *p* y los proyectos del caso base son muy utilizados los algoritmos del vecino más cercano utilizando la medida de la distancia Euclides Ponderada. 3. Generación de una estimación de esfuerzo para el proyecto *p* basado en el esfuerzo para los proyectos terminados que son similares a *p*. El número de proyectos similares a tener en cuenta para obtener una estimación esfuerzo dependerá del tamaño del caso base. Hay varias opciones para calcular esfuerzo estimado, como la siguiente: 3.1. Utilizar el mismo valor de esfuerzo del vecino más cercano. 3.2. Utilizar el esfuerzo promedio de dos o más vecinos, más cercanos. 3.3. Utilizar el esfuerzo mediano entre dos o más vecinos, más cercanos. 3.4. Utilizar el rango inverso medio ponderado, que permite a los vecinos de mayor puntuación tener más influencia que los más bajos.
Estimación por Puntos de Historia mediante Planning Poker	Puntos de Historia	Historias de usuario previas realizadas por el equipo, Nueva historia de usuario	Complejidad de las historias de usuario con respecto a otras.	Como primer paso del proceso, se selecciona una historia de usuario para asignarle una complejidad nominal que servirá de referencia para catalogar al resto de historias de usuario. Los valores que se utilizan para representar la complejidad no tienen un valor absoluto sino que su valor es función de su posición en escala. Se comenzó utilizando la serie de Fibonacci: 1,2,3,5,8,13,21, …, aunque para evitar que se pensara que hay una precisión matemática en los valores a partir de cierto número se sustituyeron por otros aproximados: 3,5,8,13,40,100,… (el 1 y el 2 no se recomienda utilizarlos al no incluir mucha diferencia con respecto al 3). También se puede utilizar los siguientes valores: Extra Small, Small, Medium, Large, Extra Large. La complejidad de las historias, los puntos de historia, no se pueden comparar a horas de esfuerzo ya que el sentido que tienen es catalogar la dificultad de la tarea. El número de horas que nos lleve realizarlas dependerá de la capacitación y/o capacidad de la persona que la lleve a cabo, la carga de trabajo del equipo, etc. y por ello variará dicho valor dependiendo de la situación (Gómez, 2013) (Cohn, 2006) ver Anexo H.

3.8 Ventajas y desventajas entre algunos métodos de estimación

Las ventajas/desventajas de los métodos: puntos de función, COCOMO, COSMIC-FFP y Wideband Delphi, fueron tomadas directamente de (Parthasarathy, 2007):

Tabla 14. Ventajas y desventajas entre algunos métodos de estimación

MÉTODO DE ESTIMACIÓN	VENTAJAS	DESVENTAJAS
Puntos de Función	El nivel de confianza de estimación es alto. Estimación basada en la perspectiva que el usuario tiene del sistema. Los puntos de función son independientes del lenguaje, metodología o herramientas usadas para la implementación.	Dependen ampliamente de la perspectiva subjetiva del estimador. Es necesario entrenar a las personas para hacer este tipo de estimaciones. El conteo de los puntos de función es una operación larga y manual. Medir reglas y la lógica del negocio es más complicado.
COCOMO	El modelo COCOMO no solamente utiliza líneas de código fuente (SLOC), también puede usar puntos de función sin ajustar como métrica del cálculo del tamaño en SLOC de un proyecto. El COCOMO básico es bueno para un rápido, temprano y un orden aproximado de la magnitud estimada de los costos de software. Desde su lanzamiento se ha actualizado, la última versión del modelo es COCOMO II (2000). Puede ser calibrado para que se adapte a la situación de la organización y mejore la exactitud de las estimaciones.	En la práctica, pocas organizaciones han recolectado suficiente información de proyectos pasados en una forma que soporten la calibración del modelo. En la fase temprana del ciclo de vida del sistema, el tamaño es estimado con un gran valor de incertidumbre. Su precisión es necesariamente limitada por la falta de factores que tienen una influencia significativa sobre el costo del software.
COSMIC-FFP	Se puede usar en aplicaciones MIS también como en aplicaciones de tiempo real. Simple de usar. Determinación de elementos no subjetiva.	Información de Benchmark actualmente no hay disponible. Aceptación internacional es limitada pero está en crecimiento.
Wideband Delphi	Útil en ausencia de cuantificada información empírica. Puede tener en cuenta las diferencias entre las experiencias de proyectos anteriores y los requisitos del proyecto propuesto. Tiene un alto valor de personalización específica a la organización.	La estimación es solamente tan buena como lo es la opinión del experto. Requiere múltiples expertos para hacer la estimación. Difícil de documentar los factores utilizados por los expertos y no define un proceso por individuos a seguir cuando se estima. Es un proceso que consume tiempo.
PROBE	A medida que los desarrolladores realizan más estimaciones estas se hacen más precisas. Cuando los datos históricos son suficientes las estimaciones caen dentro de un intervalo de predicción más pequeño. Utiliza técnicas estadísticas de estimación. La estimación es realizada por las personas que harán el trabajo. Puede estimar tamaño y esfuerzo.	Necesita varios datos históricos para que las estimaciones sean más precisas. Necesita un proceso disciplinado y riguroso de registro de tiempo de desarrollo, tamaño de programa, número de errores, etc. por parte de los desarrolladores.
Puntos de Casos de Uso	Rápida adaptación a empresas que ya estén utilizando la técnica de Casos de Uso. El uso de los casos de uso es promovido como un artefacto de UML.	Se puede comenzar la estimación una vez diseñados los casos de uso. El esfuerzo calculado solo abarca hasta la codificación de los casos de uso, para calcular el esfuerzo total del proyecto habría que estimar el esfuerzo en realizar el resto de actividades del proyecto y sumarlas al esfuerzo obtenido por el método.
Puntos de historia	Emplean muy poco tiempo para realizar la estimación. La estimación es realizada por las personas que harán el trabajo.	La estimación es muy subjetiva al equipo de desarrollo, es decir, una misma historia de usuario puede tener puntuaciones diferentes al ser estimada por equipos diferentes.

3.9 Estadísticas sobre métricas y métodos de estimación

David Consulting Group (Group D. C., 2012) realizó en el 2012 un informe para determinar las tendencias en medición del software que existen hoy en día en la industria del desarrollo (Gómez, 2013). Las empresas participantes en la serie de encuestas realizadas por la consultoría, son pertenecientes a varios sectores como: *aeroespacial*, *educación*, *finanzas*, *gobierno*, *sanidad*, *manufactura*, *servicios*, *software y telecomunicaciones*, distribuidas geográficamente en América del Norte, Sudamérica, Europa, Asia, India y Australia. El 50% de las organizaciones participantes llevan midiendo el software durante más de 3 años.

Los resultados más importantes de este informe indican que el método más utilizado a lo largo de la industria es el de Puntos de Función IFPUG. Las líneas de código es una métrica que se sigue utilizando, siendo el 2° método más utilizado, en el 3° se encuentran los Puntos de Casos de Uso seguido, en 4° lugar por los Puntos de Historia, los resultados se pueden visualizar mejor en la Figura 2. El 99.99% de los participantes asegura haber realizado mediciones del software alguna vez (Gómez, 2013).

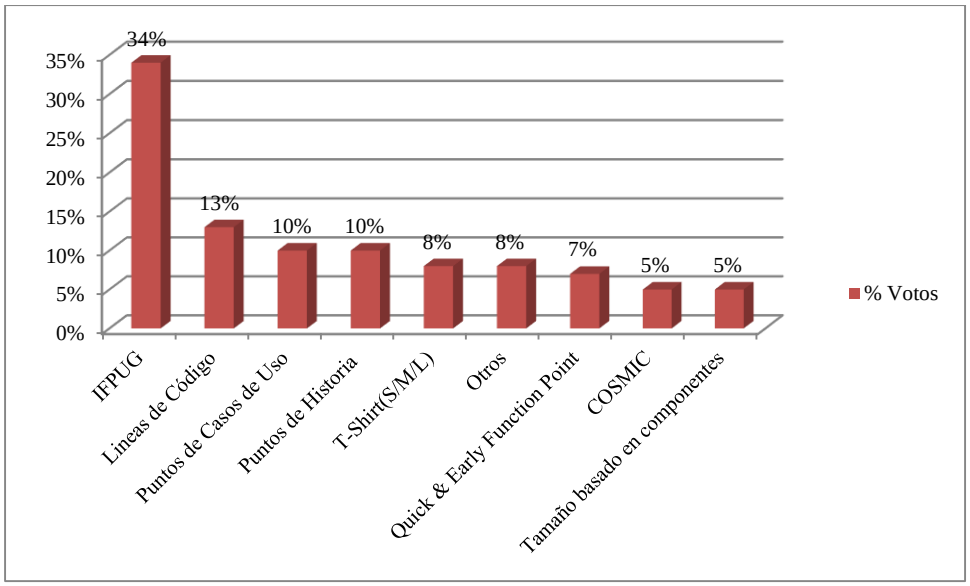


Figura 2. Informe de Métodos de Medición Utilizados en 2012

Dicho informe también muestra información sobre los métodos de medición utilizados por zona geográfica, ver Figura 3.

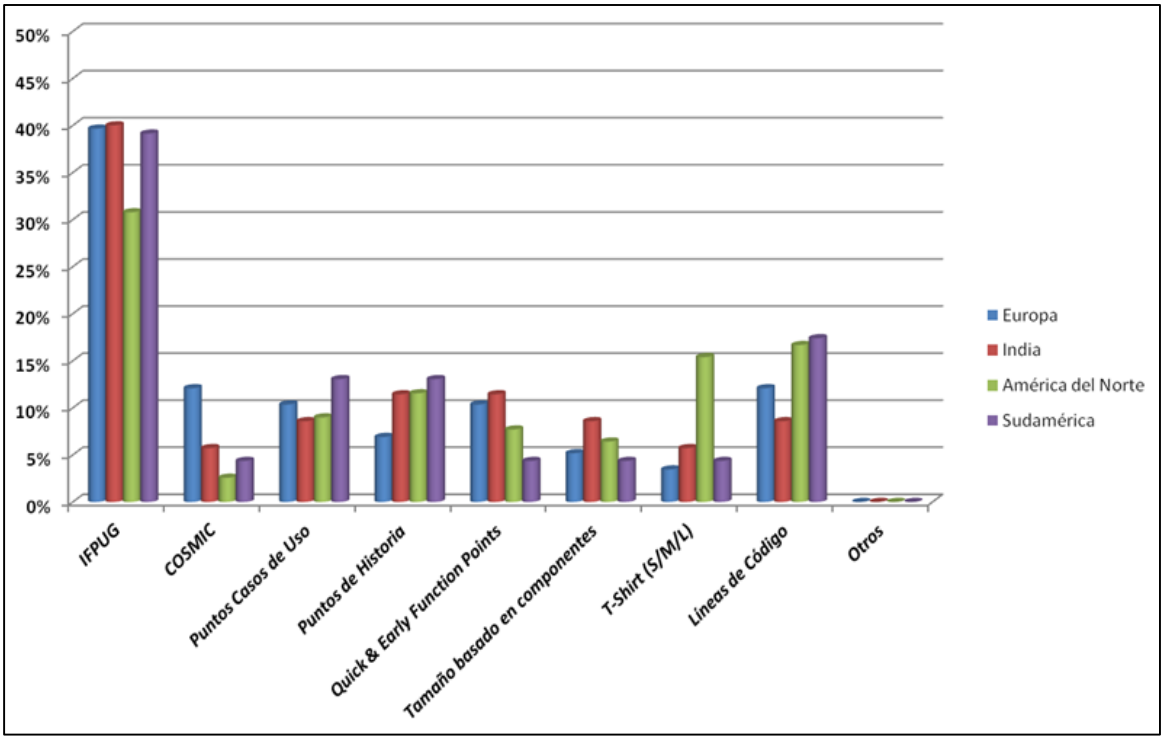


Figura 3. Informe de Métodos de Medición por Zona Geográfica 2012

Como podemos ver, los resultados globales se mantienen en general por cada zona geográfica.

Respecto a resultados del 2013, en el mes de marzo un blog en internet llamado Laboratorio de TI (Gómez, 2013), realizó una encuesta sobre cuáles son los métodos de medición utilizados habitualmente, los participantes fueron alrededor de 505 repartidos en más de 26 países, contabilizándose 685 votos. Las zonas de mayor participación fueron Europa y Latinoamérica. En la Figura 4, se pueden ver los resultados globales de la encuesta.

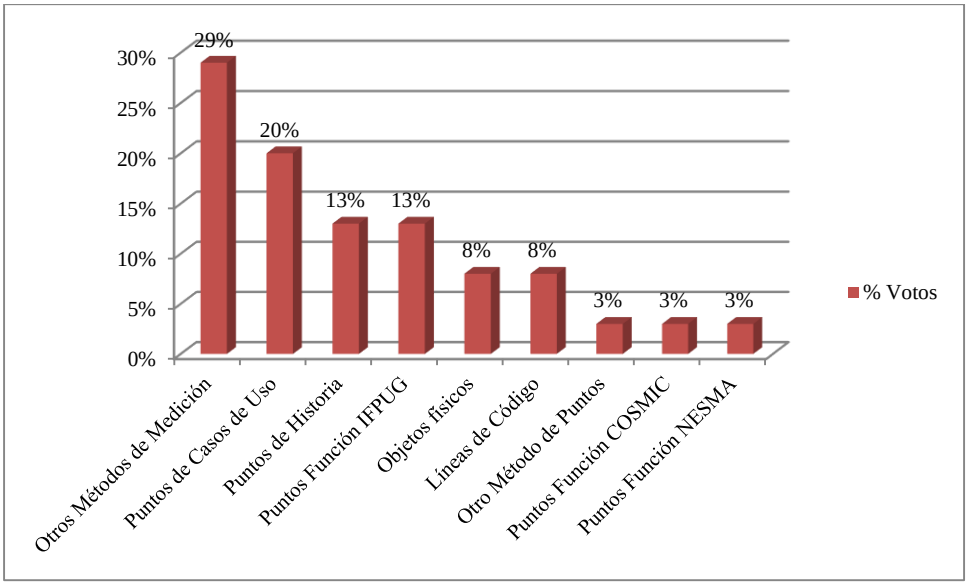


Figura 4. Métodos de medición utilizados habitualmente

Como podemos ver en la imagen, los métodos funcionales son los que tienen una mayor representación con un 42% de los votos, donde predomina el método de Puntos de Casos de Uso con un 20% y los Puntos Función IFPUG con un 13%.

Con un empate del 13% de los votos, se ubica el método de Puntos de Historia, indicando que las metodologías ágiles tienen una influencia importante en el desarrollo del software.

Los métodos técnicos también siguen utilizándose de manera destacable como podemos ver por el 16% sobre el total que suman a partes iguales los métodos de Objetos físicos y las líneas de código.

La información anterior se refiere a los resultados globales de la encuesta realizada por el Laboratorio de TI, en el mismo sitio se muestra información más detallada sobre Europa y Latinoamérica, en este trabajo nos enfocaremos en la región de Latinoamérica, ya que México es parte de esta zona geográfica.

En Latinoamérica (LatAm) se han realizado un total de 409 votos sobre un total de 299 votantes únicos, Figura 5.

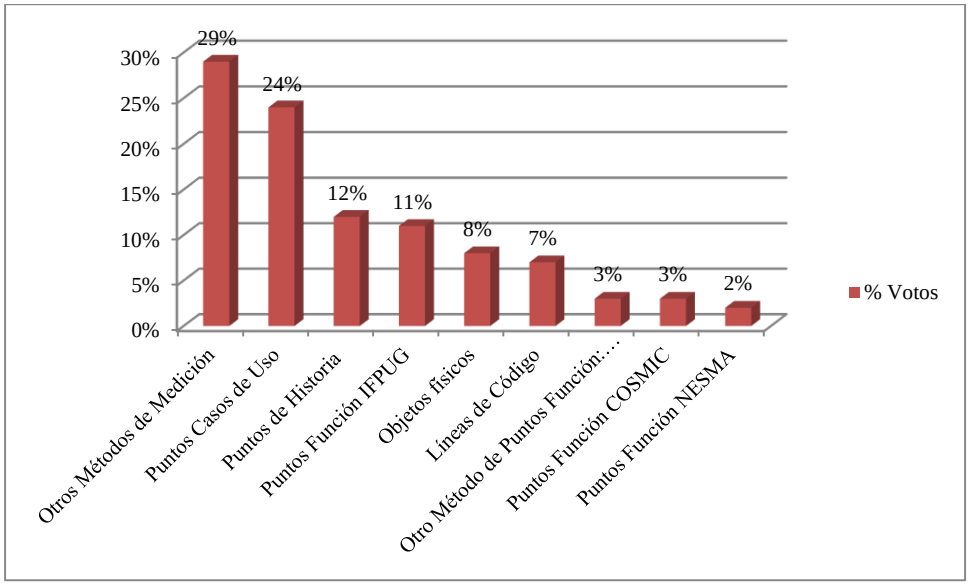


Figura 5. Métodos de medición utilizados habitualmente en Latinoamérica

Los métodos funcionales vuelven a obtener el mayor porcentaje de utilización con un 43%.

Aquí el método de Puntos de Casos de Uso es el más utilizado con una amplia mayoría de más del doble que el siguiente método funcional (24%): los Puntos Función IFPUG con un 11%.

Los puntos de historia mantienen su posición de tercer método utilizado con un 12%.

Las metodologías técnicas mantienen su influencia con un 15%, predominando ligeramente los objetos físicos con un 8% a las líneas de código con un 7%.

Como podemos notar, realizando la comparación con los resultados de 2012, el cambio más significativo para la zona de Latinoamérica es el hecho de que los Puntos de Función IFPUG han sido desplazados por Puntos de Casos de Uso, Puntos de Usuario (utilizado en desarrollos ágiles) y otros métodos de estimación. Hay que aclarar que los resultados de dicha encuesta no son concluyentes, ya que la muestra fue más pequeña en comparación con la muestra del reporte del año 2012 por David Consulting Group, pero si nos dan una pauta importante para identificar las tendencias que el desarrollo de software y la estimación del mismo están teniendo.

Debido a que este trabajo está girando en torno a los costos del mercado mexicano es importante revisar los resultados de la encuesta que el Laboratorio de TI obtuvo, ver Figura 6.

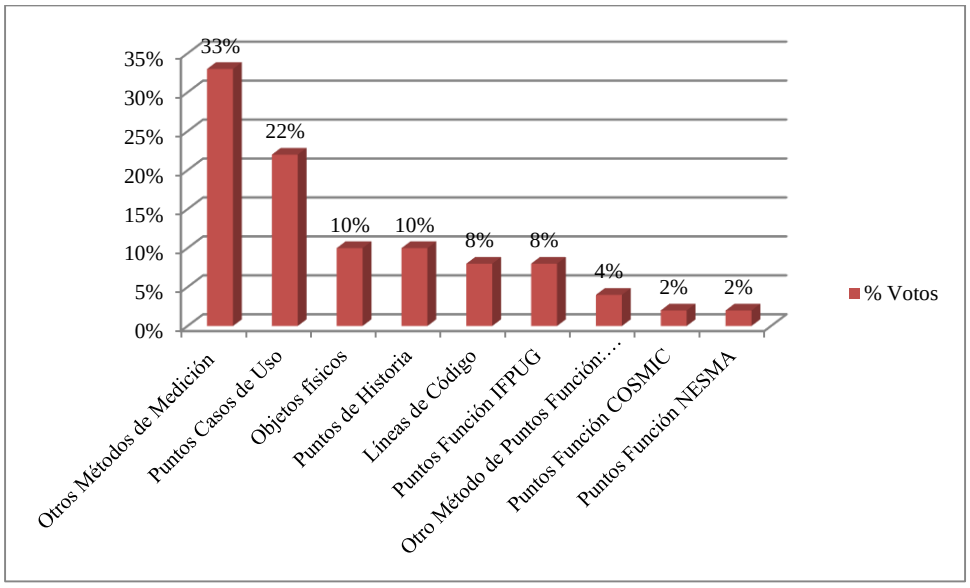


Figura 6. Métodos de medición utilizados habitualmente en México.

El total de votos ha sido 49 y el número de votantes únicos ha sido de 33.

El Método de Puntos Casos de Uso se confirma como el método nominal más utilizado con un 22% de los votos.

La opción de otros métodos de medición alcanza un valor bastante alto con un 33% de los votos

Las metodologías técnicas tienen una amplia difusión en México ya que el método de Objetos Físicos y el de Líneas de Código están situados en tercera y quinta posición, eso sí empatados respectivamente con el método de Puntos de Historia y con el método de Puntos Función IFPUG con un 10% y un 8% de los votos cada uno de ellos.

Los Puntos de Historia se siguen manteniendo como una constante en utilización con un 10% de los votos dando idea de la difusión de las metodologías ágiles de desarrollo.

Mencionar también que las metodologías de Puntos de Función COSMIC y Puntos Función NESMA se ven superadas por otros métodos de medición en Puntos de Función que han recibido el mismo porcentaje de votos que estas dos opciones juntas, 4%.

El Laboratorio de TI comparte las siguientes conclusiones sobre la encuesta que realizó, en los siguientes puntos (Gómez, 2013):

- El método de medición de Puntos de Casos de Uso es el método más ampliamente utilizado en todos los países.
- Las metodologías de conteo en Puntos de Función tienen un seguimiento desigual en cada uno de los países, dominando el método de Puntos Función IFPUG, pero se mantienen como una de las prácticas más utilizadas en cada uno de ellos, aunque difieran en el método utilizado.

- Los Puntos de Historia se mantienen en un porcentaje muy similar de votos en todos los países participantes lo que nos da una idea de la amplia extensión de las metodologías ágiles de desarrollo a nivel internacional.
- Las metodologías funcionales son mucho más utilizadas que las metodologías técnicas.

Como se observó en la Figura 6, correspondiente a los resultados de México, un alto porcentaje de los votos cayeron en la categoría "Otro método de medición", lo cual abre la interrogante, sobre ¿cuál será ese método de medición?

Con base en reportes del Software Engineering Institute (SEI) resulta que el 50% de acreditados en PSP (Personal Software Process) en el mundo son ingenieros mexicanos, permitiendo a este país ubicarse en el primer lugar con el mayor número de Desarrolladores Certificados PSP, seguido por Estados Unidos con el 34% (Avantare, 2013).

Las estadísticas anteriores indican que existe la probabilidad de que cierto porcentaje de la categoría "Otros métodos de medición" este ocupado por el método de estimación de PSP, llamado PROBE, cuya forma de medir el software es a través de proxies y líneas de código.

La información estadística que fue mostrada de la Figura 2 a la Figura 6, hace referencia a los métodos de medición más utilizados, no confundir con los métodos de estimación. Como se explicó en el Capítulo 2, medir y estimar no es lo mismo. Estos datos son útiles ya que ayudan a determinar según el tipo de métrica, cuales son los métodos de estimación que se están utilizando en la industria del desarrollo de software.

En la Figura 6 que muestra las métricas utilizadas para medir, podemos inferir los métodos de estimación. Si se utilizan puntos de caso de uso para medir, implica que se estima con el método puntos de casos de uso (Issa , Odeh , & Coward , 2006).

Si se utilizan puntos de historia para medir, se puede estimar con métodos ágiles como planning poker que utiliza puntos de historia. Los resultados que evidencian el porcentaje de usuarios que utilizan puntos de historia de la Figura 6, se pueden observar en la Figura 7, (Landry, 2013) (resultado del reporte de 2009 realizado por Forrester Research una empresa de consultoría), esa información refleja que ha habido un incremento en la adopción de métodos ágiles de desarrollo. Información más reciente revela los niveles actuales de adopción ágil, en una encuesta que se realizó de junio a principios de septiembre de 2012 y hubo 113 encuestados (Ambler, Agility at Scale Survey:Results from the Summer 2012 DDJ State of the IT Union Survey, 2013). La encuesta fue anunciada en junio 2012 en el artículo Disciplined Agile Change Management (Ambler, Dr.Dobb´s: the world of software development , 2012), escrito por Scott W. Ambler. En este caso el 71% de los encuestados indicaron que trabajan en organizaciones que han tenido éxito en la metodología ágil y un 15% en las organizaciones que han intentado métodos ágiles, pero aún no han triunfado en ella, ver Figura 8.

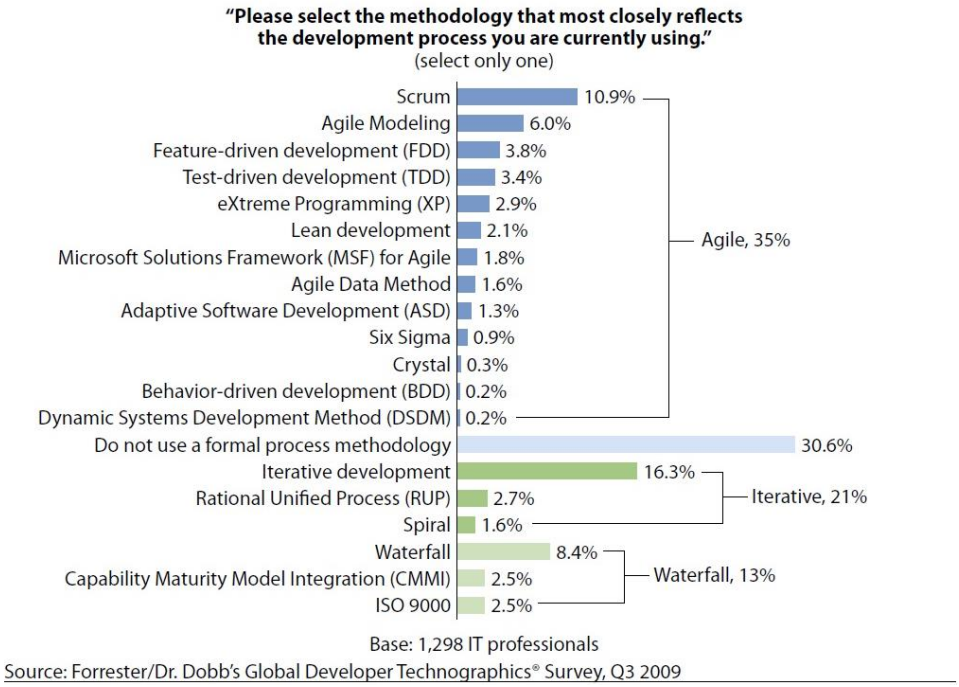


Figura 7. Adopción de métodos ágiles en 2009

Figura 8. Adopción de métodos agiles en 2012

También esta encuesta reporta resultados que tienen que ver con el éxito que los equipos de desarrollo han tenido al adoptar metodologías ágiles, pero este éxito está estrechamente relacionado al número de integrantes que conforman el equipo, ver Figura 9. Para nuestra investigación, esta información es relevante ya que significa que las metodologías ágiles se adaptan muy bien a equipos muy pequeños de desarrollo y como recordaremos estamos abordando los micro-proyectos de software, los cuales son desarrollados por equipos que van de un integrante a lo más tres. Por toda la información anterior se concluye que las estimaciones con métodos ágiles (como Planning Poker) que utilizan puntos de historia como métrica de medición, deben ser consideradas para nuestro análisis práctico de métodos de estimación, y más específicamente Planning Poker ya que es el método de estimación utilizando en SCRUM la cual es una de las metodologías de desarrollo ágil más utilizadas, ver Figura 7.

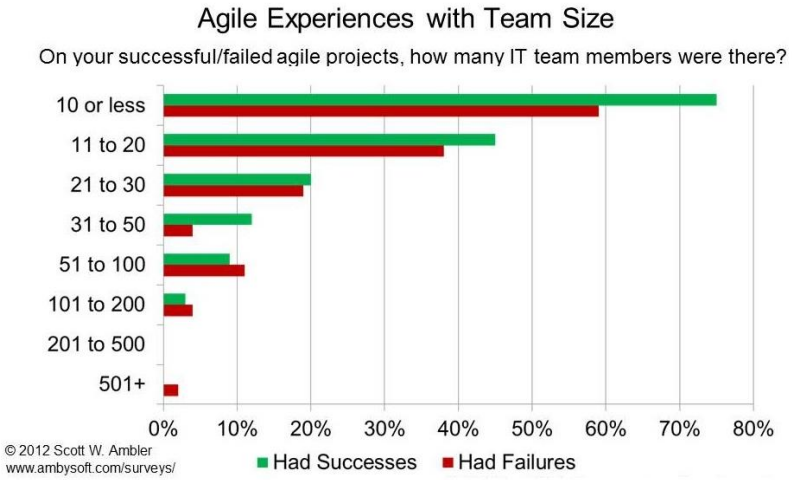


Figura 9. Efectividad de adopción de métodos ágiles entre los equipos de desarrollo

Volviendo a la Figura 6 se observa que la métrica de puntos de función es muy utilizada, implicando que los métodos de estimación empleados pueden ser puntos de función: IFPUG, Mark II, FiSMA, COSMIC o NESMA (Gómez, 2013). En cuanto al uso de líneas de código para medir, hace que se amplíe más el rango de métodos de estimación que se podrían estar utilizando, entre los más comunes COCOMO, SLIM, analogías, juicio experto, etc.

Más específicamente estudios sobre métodos de estimación utilizados por las empresas mexicanas desarrolladoras de software, se cuenta con uno realizado en la Universidad Tecnológica de la Mixteca, Oaxaca, México en el año 2012; realizado por un grupo de investigación, el cual muestra que la gran mayoría de las empresas encuestadas no utilizan métodos de estimación, ya que no los conocen o han ideado propias formas de estimación, ver Anexo J.

Otro punto importante en la selección de los métodos de estimación a aplicar, es la muestra de micro-proyectos con que se cuenta, la investigación previa que dio pie a este trabajo de tesis (Anexo J y Anexo K), inició con una muestra de 10 proyectos, analizando las características de cada uno de ellos, se encontró que solo 4 cumplían la definición de micro-proyecto de software, para alcanzar el objetivo de analizar 20 fue necesario recabar más, con la contribución de empresas desarrolladoras de software se reunieron en total 7 micro-proyectos, no fue posible reunir más ya que no se tenía relación con otras empresas desarrolladoras de software. La información de cada micro-proyecto proporcionada por las empresas, consiste en los requerimientos que debe cumplir el software, algunos detalles técnicos como lenguaje de programación y versiones de bases de datos utilizadas, el tiempo de desarrollo acordado con sus clientes, el tiempo real y el costo del proyecto, así como manuales de usuario con pantallas del sistema. Únicamente de 4 proyectos fue proporcionado el código fuente.

Todos los métodos de estimación tienen como prerrequisito conocer los requerimientos que el sistema debe cumplir, ya sea de manera general o preferentemente de forma más detallada, pero recordemos que la estimación se realiza en etapas tempranas de desarrollo por lo tanto no se conocen muchos detalles del sistema. Con la información disponible de los proyectos se pueden seleccionar aquellos métodos que no necesitan tantos detalles en los requerimientos como Puntos de Historia y PROBE. Ya que se cuenta con manuales de usuario que contienen interfaces graficas se puede aplicar un análisis de Puntos de Función, recordemos que puntos de función es una métrica de tamaño, por esa razón para el cálculo del esfuerzo se complementará con COCOMO II y no con SEER-SEM, ya que COCOMO II es más aceptado y conocido, sin mencionar que esta versión del modelo está especialmente adaptada para desarrollos actuales, su último ajuste fue realizado en el año 2000. Con la información disponible de los micro-proyectos, el quinto método que puede ser aplicado a la muestra, es el de estimación por Puntos de Caso de Uso, evidentemente este método requiere que se tengan listos los Casos de Uso del proyecto, aunque los proyectos de la muestra no contienen explícitamente los casos de uso, estos pueden ser construidos con la información disponible, y así aplicar el método. Caso contrario con los métodos que requieren datos históricos de proyectos de software anteriores,

estos difícilmente pueden ser considerados, ya que no se cuenta con ese tipo de información. PROBE puede ser una excepción a lo anterior, ya que aunque utiliza datos históricos para mejorar las estimaciones, proporciona una variante del método en caso de que no se cuente con información histórica, además de que este método no desecha por completo el juicio ingenieril del desarrollador para hacer la estimación.

En base a las estadísticas y tomando en cuenta la información disponible de los proyectos por los que está constituida la muestra a analizar, los métodos de estimación seleccionados para su aplicación fueron:

- Análisis de Puntos de Función
- COCOMO II
- Puntos de Casos de Uso
- PROBE
- Estimación de Puntos de Historia con Planning Poker

3.10 Preparación de la muestra de micro-proyectos, aplicación de los métodos de estimación seleccionados y recolección de datos.

Para la parte práctica de este trabajo de tesis, se recabaron en total 7 micro-proyectos de software, proporcionados por tres empresas desarrolladoras de software (Arquitectos de Software, Kadasoftware y Luminisoft), los cuales constituyen la muestra a analizar con los 5 métodos de estimación seleccionados. La información de cada micro-proyecto proporcionada por las empresas, consiste en los requerimientos que debe cumplir el software, algunos detalles técnicos como lenguaje de programación y versiones de bases de datos utilizadas, el tiempo de desarrollo acordado con sus clientes, el tiempo real y el costo del proyecto, así como manuales de usuario con pantallas del sistema. Únicamente de 4 proyectos fue proporcionado el código fuente, en la Tabla 15 se desglosa la información de la muestra de micro-proyectos. Respecto a información sobre la metodología de desarrollo que se utilizó en cada micro-proyecto, no fue proporcionada. Para cada método de estimación la información relacionada con los requerimientos del sistema se presentará en un formato específico, según los datos de entrada que el método necesite. La Tabla 16 describe claramente lo anterior.

Tabla 15. Información de la muestra de micro-proyectos a analizar

Nombre	Tiempo de desarrollo	Costo	# programadores	Lenguaje de programación	Año de desarrollo	Información disponible
Proyecto 1	2 meses	$25,520.00	1	Visual Basic	2010	Requerimientos. Manual de usuario. Código fuente.
Proyecto 2	4 meses	$46,400.00	1	Visual Basic	2010	Requerimientos. Manual de usuario. Código fuente.
Proyecto 3	3 meses	$100,000.00	1	ASP.NET	2007	Requerimientos. Manual de usuario. Código fuente.
Proyecto 4	3 meses	$6,000.00	1	Visual Basic	2001	Requerimientos. Manual de usuario. Código fuente.
Proyecto 5	95 días	$19,210.00	2	Ruby 3.1.3	2013	Requerimientos. Manual de usuario.
Proyecto 6	45 días	$53,824.00	2	Ruby 3.1.3	2013	Requerimientos. Manual de usuario.
Proyecto 7	57 días	$50,956.00	2	VisualBasic	2012	Requerimientos. Manual de usuario.

Tabla 16. Información de entrada para cada método de estimación

Método de estimación	Formato de entrada de la información
Puntos de Función	Se requieren interfaces gráficas y requerimientos del sistema.
COCOMO II	Se requiere el tamaño estimado del software en líneas de código y los requerimientos del software. El sub-modelo COCOMO II seleccionado para el análisis es el de Diseño Temprano, porque se supone que esta estimación de esfuerzo se hace antes de comenzar el diseño del sistema.
Puntos de Casos de Uso	Son necesarios los Casos de Uso, realizados a partir de los requerimientos del sistema.
PROBE	El método necesita conocer los requerimientos del sistema, a partir de estos se realiza un Diseño Conceptual y un Diseño Detallado.
Puntos de Historia	Los requerimientos de los usuarios deben estar escritos en forma de Historias de Usuario.

Como última parte de este capítulo, una vez que la información de los micro-proyectos fue organizada en el formato requerido dependiendo del método de estimación, por cada proyecto se realizaron cinco análisis, para estimar el esfuerzo de desarrollo necesario para llevar a cabo cada proyecto. Los resultados se muestran en la Tabla 17.

Tabla 17. Datos recabados del análisis de la muestra de micro-proyectos con cinco métodos de estimación.

MICRO-PROYECTO	ESFUERZO			
	Puntos de Función-COCOMOII	**Puntos de Casos de Uso**	**PROBE**	**Puntos de Historia**
Proyecto 1	36 meses-persona	24 meses-persona	5 meses-persona	2 meses-persona
Proyecto 2	13 meses-persona	9 meses-persona	1 meses-persona	1 meses-persona
Proyecto 3	65 meses-persona	32 meses-persona	1 meses-persona	1 meses-persona
Proyecto 4	5 meses-persona	4 meses-persona	1 meses-persona	1 meses-persona
Proyecto 5	13 meses-persona	29 meses-persona	2 meses-persona	1 meses-persona
Proyecto 6	16 meses-persona	50 meses-persona	3 meses-persona	3 meses-persona
Proyecto 7	24 meses-persona	19 meses-persona	1 meses-persona	1 meses-persona

El análisis de los datos obtenidos después de aplicar los cinco métodos de estimación a la muestra de micro-proyectos, comparándolos con los datos reales de desarrollo se realiza en el Capítulo 4. Este tercer capítulo solo abarcó la explicación de la metodología de investigación que se utilizó para obtener la literatura necesaria y más importante para determinar cuáles son los métodos de estimación más conocidos, sus características y estadísticas sobre su uso en la industria del desarrollo del software, con el fin de ayudar a elegir cinco de ellos. El nombre de la metodología es EBSE y se auxilia de otra llamada SLR, para búsqueda de literatura. El capítulo no solamente explica cómo se realizó la búsqueda de literatura y sus resultados, sino que también explica la parte práctica del proceso de investigación EBSE, el cual fue constituido por la aplicación de los cinco métodos de estimación de esfuerzo elegidos, a una muestra de 7 micro-proyectos de software, es decir, por cada proyecto de la muestra se realizaron cinco estimaciones con métodos diferentes y cuyos resultados se mostraron en forma de tabla al final del capítulo.

Capítulo 4. Resultados

Este capítulo lo constituye el último paso del proceso EBSE, y en este se realizó el análisis y síntesis de los datos obtenidos en el capítulo anterior con el fin de decidir cuál es el método o cuáles son los métodos de estimación más apropiados para los micro-proyectos. Se responderán las preguntas de investigación, se comprobará la hipótesis y se realizara una revisión de los objetivos planteados al inicio de la investigación para corroborar que se haya cumplido. En este capítulo termina el proceso de investigación.

En el Capítulo 3 fueron seleccionados cinco métodos de estimación de esfuerzo: Análisis de Puntos de Función-COCOMO II, Puntos de Casos de Uso, Puntos de Historias de Usuario con Planning Poker y PROBE. Estos métodos fueron aplicados a una muestra de 7 micro-proyectos de software, los resultados de la estimación del esfuerzo en meses-persona se muestran en la Tabla 18. La unidad de medida del esfuerzo de desarrollo de software que se utilizará es meses-persona[1]. El esfuerzo real utilizado como punto de comparación entre resultados de las estimaciones, fue calculado a partir de la información proporcionada por las empresas que brindaron los micro-proyectos, fue necesario hacer el cálculo ya que no se contaba con ese dato, dicho resultado se obtuvo al multiplicar el tiempo total de desarrollo por el número de programadores del proyecto, y en el caso de ser necesario se convirtió a meses-persona.

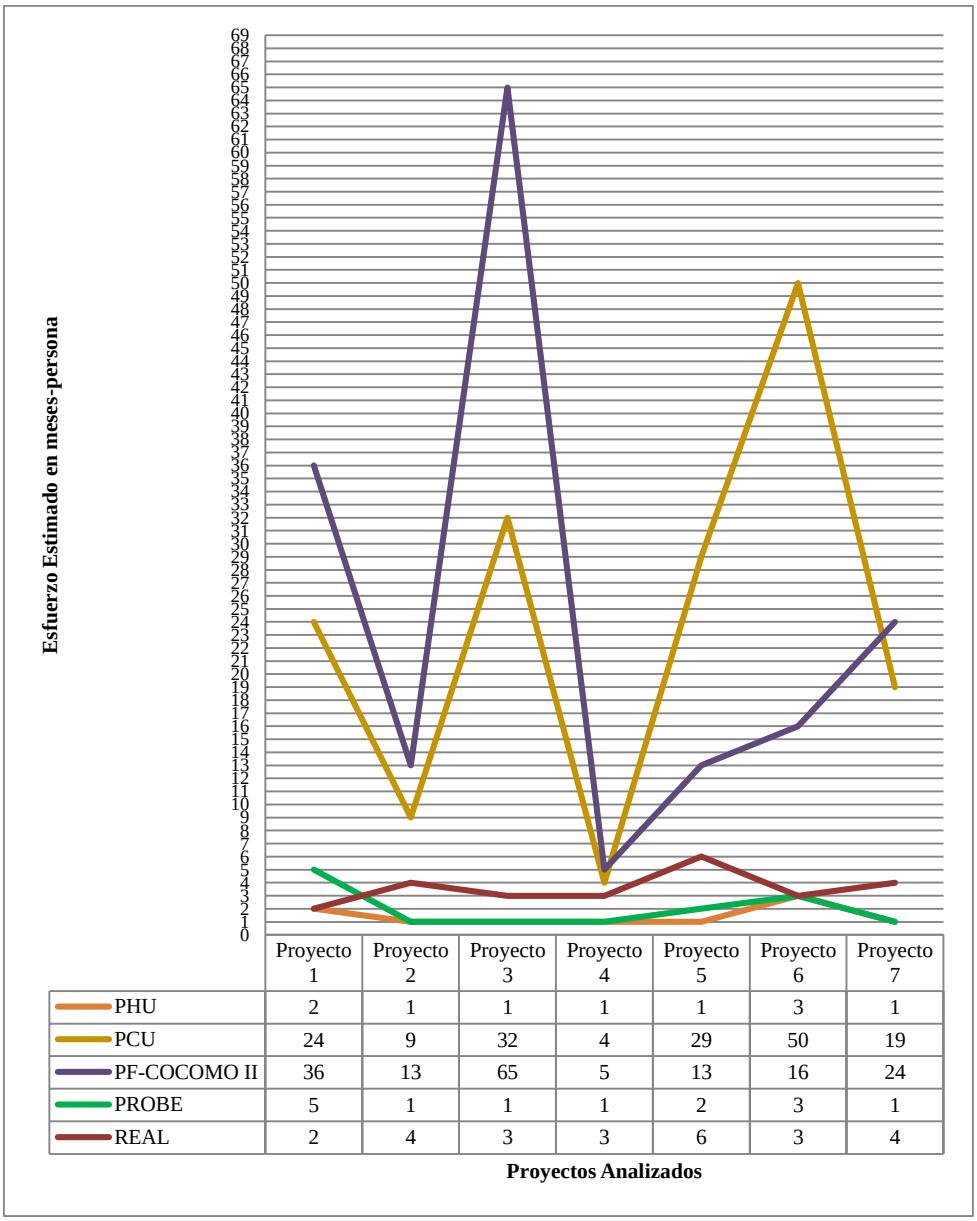
Los valores en la columna Puntos de Función-COCOMO II son el resultado de combinar ambos métodos, ya que por sí solos no calculan el esfuerzo requerido (Puntos de Función estima tamaño, COCOMO II parte de la [1] premisa de que el tamaño ya fue estimado con otro método y posteriormente calcula el esfuerzo de desarrollo).

En la Figura 10 se presenta una gráfica de resultados.

Tabla 18. Resultados del análisis

MICRO-PROYECTO	MÉTODO UTILIZADO				
	Puntos de Historia	**Puntos de Casos de Uso**	**Puntos de Función-COCOMO II**	**PROBE**	**Real**
	ESFUERZO (meses-persona)				
Proyecto 1	2	24	36	5	2
Proyecto 2	1	9	13	1	4
Proyecto 3	1	32	65	1	3
Proyecto 4	1	4	5	1	3
Proyecto 5	1	29	13	2	6
Proyecto 6	3	50	16	3	3
Proyecto 7	1	19	24	1	4

[1] Un mes-persona es la cantidad de tiempo que una persona gasta trabajando en el desarrollo de un proyecto de software por un mes (Boehm B. , 2000)



	Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4	Proyecto 5	Proyecto 6	Proyecto 7
PHU	2	1	1	1	1	3	1
PCU	24	9	32	4	29	50	19
PF-COCOMO II	36	13	65	5	13	16	24
PROBE	5	1	1	1	2	3	1
REAL	2	4	3	3	6	3	4

Figura 10. Gráfica comparativa de resultados del análisis

Las estimaciones realizadas con los diferentes métodos contaron con la misma información, entonces ¿a qué se deben las diferencias entre los resultados? Algunas de las posibles razones planteadas por los investigadores (Parthasarathy, 2007) son:

- Los datos de entrada que se recopilan para la aplicación de cada método son de naturaleza distinta dependiendo del método de estimación y pueden ser más o menos efectivos dependiendo de la naturaleza de los proyectos.
- Los valores constantes utilizados en los métodos no son los adecuados para micro-proyectos, ya que se obtuvieron a partir de muestras de proyectos grandes.
- Participación del cliente para responder dudas sobre lo que necesita que haga el software.
- Experiencia por parte de los desarrolladores.
- Entendimiento del método de estimación.
- Incertidumbre de los datos.

En la Tabla 19 se describen las limitaciones encontradas para cada método al evaluar el esfuerzo en este trabajo de investigación.

Tabla 19. Limitaciones encontradas para la realización del trabajo de investigación

Método de Estimación	Descripción
Puntos de historias de usuario con Planning Poker	Depende mucho de la participación del dueño del producto, para que el equipo de desarrollo entienda el alcance del proyecto y sus implicaciones. Ya no se tuvo disponible al Dueño del Producto de cada proyecto para validar información respecto a la funcionalidad de las aplicaciones. Las estimaciones dependen de la experiencia del equipo de desarrollo.
Puntos de casos de uso	Este método utiliza algunas constantes para realizar la estimación, los valores constantes fueron establecidos en base a proyectos con atributos que superan en magnitud a los micro-proyectos. Como este método utiliza casos de uso, los estimadores pueden entender mejor la interacción del sistema con el usuario. Pero no es suficiente entender la interacción, si se contara con información más detallada sobre cuestiones técnicas como el funcionamiento interno del sistema y el entorno en el cual se desarrolló el proyecto, la estimación sería más precisa, ya que el método evalúa factores técnicos y factores de entorno por separado, esos factores tienen un impacto significativo en la estimación total.
Puntos de Función-COCOMO II	Se contó con las interfaces de usuario e información de las bases de datos, lo cual permitió realizar el conteo de Puntos de Función sin Ajustar correctamente, pero faltó información más detallada para evaluar los factores técnicos de complejidad y así calcular los Puntos de Función Ajustados. Para el método COCOMO II faltaron detalles acerca del entorno de desarrollo, para la evaluación de los multiplicadores de esfuerzo y factores de escala. Ambos métodos utilizan valores constantes para realizar la estimación. Estos valores fueron establecidos al analizar proyectos cuyos atributos superan a los de los micro-proyectos.
PROBE	El método es muy riguroso. No se contaba con datos históricos que apoyen la estimación.

Para determinar cuál es el método de estimación cuyos resultados son más exactos respecto al esfuerzo real de desarrollo de software, se realizó una comparación entre el esfuerzo estimado y el esfuerzo real para cada uno de los proyectos. En la Tabla 20, se muestra el porcentaje de diferencia que existe entre el resultado estimado y el real. En la Tabla 21, se muestra una valoración de cuál fue el mejor y el peor método de estimación de esfuerzo para cada proyecto, basándose en los resultados de la Tabla 20. La escala va de 1 a 4, donde 1 es el que tiene menor diferencia con respecto al esfuerzo real y 4 es el que más difiere con respecto al esfuerzo real de desarrollo.

Tabla 20. Diferencia porcentual entre el esfuerzo real y el esfuerzo estimado por cada método sobre cada uno de los proyectos.

	Proyecto 1	**Proyecto 2**	**Proyecto 3**	**Proyecto 4**	**Proyecto 5**	**Proyecto 6**	**Proyecto 7**
	% Diferencia	% Diferencia	% Diferencia	% Diferencia	% Diferencia	% Diferencia	% Diferencia
REAL	0	0	0	0	0	0	0
PF-COCOMO II	1700	225	2066.7	66.7	116.7	433.3	500
PHU	0	-75	-66.7	-66.7	-83.3	0	-75
PCU	1100	125	966.7	33.3	383.3	1566.7	375
PROBE	150	-75	-66.7	-66.7	-66.7	0	-75

Tabla 21. Valoración del mejor método y el peor para la estimación de micro-proyectos: 1 es el más cercano y 4 es el más lejano.

	Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4	Proyecto 5	Proyecto 6	Proyecto 7
PF-COCOMO II	4	4	4	2	3	3	4
PHU	1	1	1	4	2	1	1
PCU	3	3	3	1	4	4	3
PROBE	2	1	1	2	1	1	1

Los resultados mostrados en la Tabla 20 y la Tabla 21, se visualizan mejor en la Figura 11 y en la Figura 12.

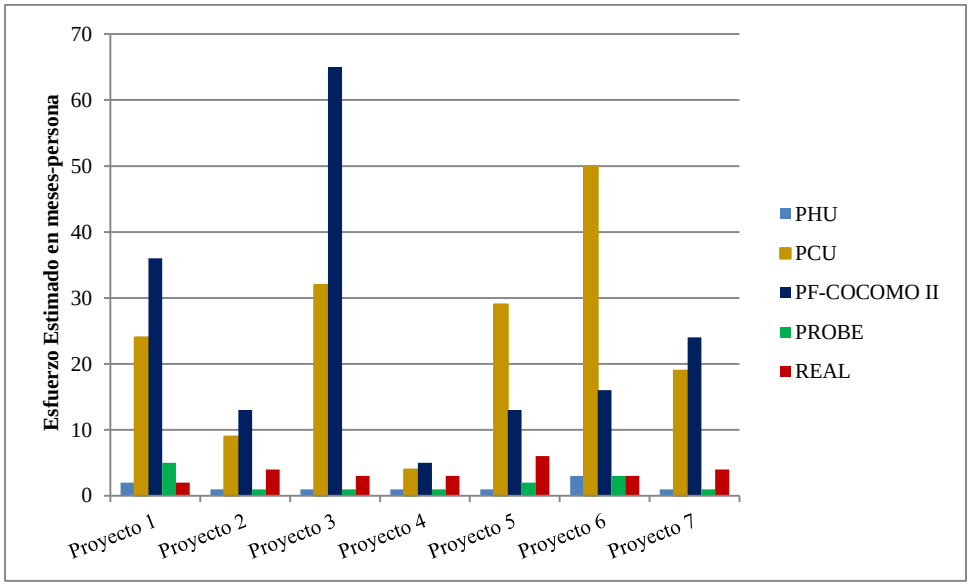


Figura 11. Gráfica comparativa de resultados obtenidos al estimar con PHU, PCU, PF-COCOMO II y PROBE

Como se puede observar en la Figura 11, las estimaciones realizadas con el método Puntos de Función-COCOMO II y Puntos de Casos de Uso fueron las más alejadas del esfuerzo real, quiere decir que sobreestimaron el esfuerzo de desarrollo.

Una de las repercusiones más significativas de sobre-estimar el esfuerzo de desarrollo de software, es que el costo que se presupueste sea elevado, etc. El tiempo de desarrollo estimado también es más alto, por lo que el cliente puede desanimarse debido a que necesite el proyecto lo antes posible. Los dos puntos anteriores son determinantes para que el cliente decida si contrata los servicios de la empresa desarrolladora de software. Una ventaja de la sobreestimación es que da más tiempo de holgura para terminar el proyecto en el tiempo acordado pero la holgura debe ser determinada por el equipo de desarrollo en base a las circunstancias, no por el método de estimación. Pero en los resultados obtenidos, mostrados en la Figura 11, es excesiva la sobreestimación y poco conveniente.

En la Figura 12 se muestran solo los resultados obtenidos con los métodos Puntos de Historias de Usuario y PROBE que fueron los más cercanos al esfuerzo real en la muestra analizada. Nótese que en 5 de los 7 proyectos analizados el método de Puntos de Historias de Usuario obtuvo los mismos resultados que el método PROBE, esto significa que ambos podrían ser igualmente buenos para estimar el esfuerzo necesario para el desarrollo de los micro-proyectos de software.

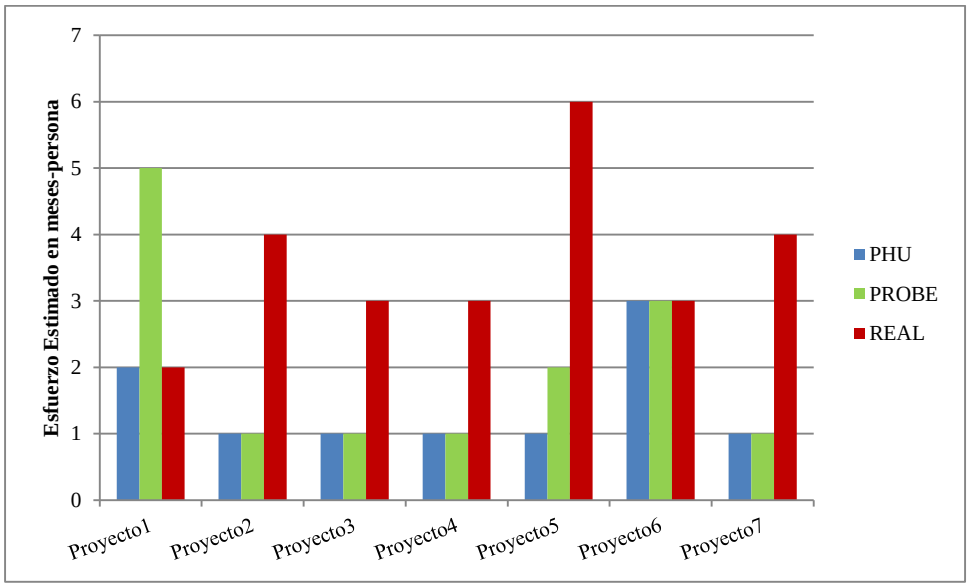


Figura 12. Análisis de los resultados de los métodos: Puntos de Historias de Usuario (PHU) y PROBE

Como se observa en la Figura 12, los resultados estimados con Puntos de Historias de Usuario y PROBE, son en promedio los más cercanos al esfuerzo real de desarrollo, pero están por debajo de él, eso supone un problema ya que se está subestimando el esfuerzo, esto podrá desembocar en que el proyecto termine excediendo el tiempo estimado y con costos extras.

Dado que los Puntos de Historias de Usuario se ocupan en metodologías ágiles, no se establecen plazos de entrega y costos fijos, lo que significa que la fecha y el costo se van definiendo conforme avanzan las iteraciones. Por lo tanto la estimación es un punto de referencia inicial que indica aproximadamente cuanto tiempo tomará el desarrollo y aproximadamente cuánto costará, si este se alarga más de lo previsto no causará tantos conflictos entre el dueño del producto y el equipo de desarrollo. Si se tratase de un método que defina plazo de desarrollo fijo y costo fijo, una subestimación sería catastrófica para el término exitoso del proyecto y la buena relación entre el cliente y el proveedor.

Respecto a los resultados obtenidos con PROBE estos son cercanos al esfuerzo real, como se vio en la Figura 12, pero la mayoría de las estimaciones son menores que el esfuerzo real, por lo tanto la estimación corre un riesgo alto de ser menor (subestimación). PROBE es utilizado en la metodología de PSP/TSP la cual trabaja bajo un esquema de tiempo y costo fijo, por lo cual no es sano que incurra en subestimaciones.

Dos resultados importantes se han obtenido, si el proveedor de software trabaja dentro del margen de desarrollo ágil las estimaciones con Puntos de Historias de Usuario son adecuadas para los micro-proyectos. En el caso contrario donde el proveedor trabaja bajo un régimen de mejora de procesos como PSP/TSP, la estimación con el método PROBE, obtendrá resultados muy cercanos al esfuerzo real de desarrollo de los micro-proyectos.

Con los resultados obtenidos fue posible calcular la varianza y desviación estándar de los conjuntos de datos estimados más el esfuerzo real de cada proyecto, ver Tabla 22.

Tabla 22. Desviación estándar de los resultados estimados por proyecto

	Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4	Proyecto 5	Proyecto 6	Proyecto 7
	Esfuerzo	Esfuerzo	Esfuerzo	Esfuerzo	Esfuerzo	Esfuerzo	Esfuerzo
REAL	2	4	3	3	6	3	4
PF-COCOMO II	36	13	65	5	13	16	24
PHU	2	1	1	1	1	3	1
PCU	24	9	32	4	29	50	19
PROBE	5	1	1	1	2	3	1
Varianza	238.2	27.8	794.8	3.2	132.7	414.5	118.7
Desviación Estándar	15.43	5.27	28.19	1.78	11.51	20.35	10.89

La desviación estándar (**s**) de un conjunto de datos es una herramienta matemática que mide cuánto se dispersan los datos con relación a la media, la desviación estándar es la raíz cuadrada de la varianza.

La varianza (s^2) se define como la media de las diferencias con la media elevada al cuadrado, la fórmula es la siguiente:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

En el contexto de la investigación, esta herramienta determina si hubo una gran diferencia entre las estimaciones obtenidas por cada método al aplicarlos a un proyecto en común. Entre más pequeña sea la desviación estándar indicará que las estimaciones de cada método fueron muy parecidas entre sí. En caso contrario, significa que las estimaciones de cada método sobre un mismo proyecto fueron muy diferentes.

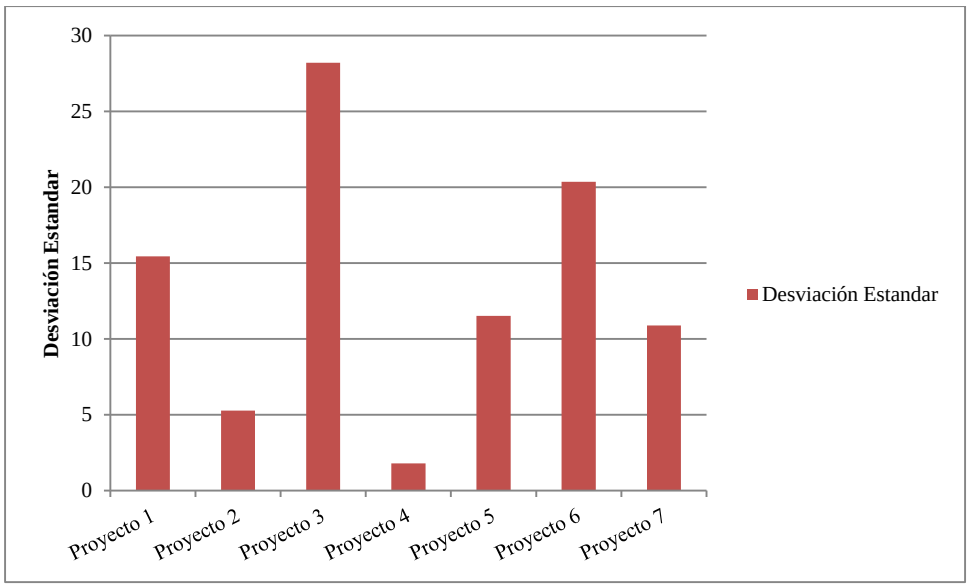


Figura 13. Diferencia entre resultados obtenidos por cada método de estimación aplicado a un proyecto en común.

De manera general podemos apreciar en la Figura 13 que las estimaciones hechas del Proyecto 4 son las más parecida entre sí, ya que la desviación estándar es la más pequeña de todas. Por el contrario la desviación

estándar del conjunto de datos perteneciente al Proyecto 3, muestra que las estimaciones realizadas fueron diferentes entre sí.

Entre los posibles factores que pudieron haber afectado las estimaciones de los micro-proyectos, provocando la variación en los resultados (en mayor medida del Proyecto 3) se identificaron los siguientes:

- Experiencia del estimador.
- Requerimientos poco claros y/o incompletos.
- Falta de comunicación con el dueño de cada proyecto para resolver dudas.
- Falta de información contextual de los proyectos.
- Falta de datos históricos.
- Uso de valores constantes obtenidos tras el análisis de proyectos cuyos atributos superan a los de los micro-proyectos.

Con la información obtenida del análisis es posible contestar las preguntas de investigación planteadas para este trabajo de Tesis. A continuación se describen las respuestas a cada una de las preguntas de investigación.

P1. ¿Cuáles son los métodos de estimación más utilizados hoy en día?

R1. Para responder esta pregunta la investigación comenzó con la búsqueda de los métodos de estimación y clasificaciones de estos descritas por diversos autores, en varias ocasiones los autores coincidían en los nombres y clasificaciones de estos y en otras los llamaban de diferente manera pero se trataba de los mismos métodos, esto para asegurar que los métodos de estimación fueran reconocidos en el ámbito de la Ingeniería del Software. El siguiente paso para determinar los métodos más utilizados, consistió en buscar información estadística sobre el uso de métodos de estimación en la industria del desarrollo del software. Los métodos encontrados fueron los siguientes:

- Análisis de Puntos de Función
- Wideband-Delphi
- Lógica Difusa (Fuzzy-Logic)
- Método de componentes estándar
- COCOMO
- COCOMO II
- SLIM
- Método COSMIC-FFP
- Método Mark II FPA
- Puntos de Caso de Uso
- Analogías
- Desagregación (Bottom-Up)
- Método Top-Down
- SEER-SEM
- Time-Boxing
- PROBE
- Clasificación y Arboles de Regresión (CART)
- Razonamiento Basado en Casos (CBR)
- Estimación por Puntos de Historia mediante Planning Poker

Una descripción más detalla de los resultados obtenidos de la investigación se muestran en la Tabla 13 del Capítulo 3.

P2. ¿Cuáles son las características comunes y diferencias entre esos métodos?

R2. Es difícil comparar todos estos métodos, ya que algunos pertenecen a diferentes clasificaciones, por ejemplo COCOMO es un método paramétrico, Wideband-Delphi utiliza un enfoque heurístico, otros usan técnicas de inteligencia artificial como Lógica Difusa (Fuzzy-Logic), Puntos de Historias de Usuario con Planning Poker tiene un enfoque ágil, etc. En el Capítulo 2 se describe ampliamente las clasificaciones de los métodos así como sus características, y en el Capítulo 3 en la Tabla 14 se realiza una comparación entre algunos de estos métodos de estimación.

P3. ¿Existe un método de estimación que por sus características se adecúa más a los proyectos de software de corta duración?

R3. Se aplicaron cinco diferentes métodos de estimación (Puntos de Función-COCOMO II, Puntos de Casos de Uso, Puntos de Historias de Usuario y PROBE) a una muestra de 7 micro-proyectos de software.
De los resultados obtenidos con cada método de estimación algunos estuvieron por encima del esfuerzo real de desarrollo de software (Puntos de Función – COCOMO II y Puntos de Casos de Uso) y otros por debajo (Puntos de Historias de Usuario y PROBE).
Después de comparar los resultados estimados y el esfuerzo real, ver Tabla 21, se obtuvo que de siete micro proyectos analizados, uno fue estimado mejor con Puntos de Historias de Usuario, uno con el método PROBE y uno con Puntos de Casos de Uso. De esos mismos tres proyectos, los puntos de historias de usuario fueron el segundo mejor método en aquellos proyectos en donde no fueron el primero y equivalentemente PROBE fue el segundo mejor método en aquellos en los que no fue el primero. Respecto a los 4 proyectos restantes, el método de Puntos de Historias de Usuario junto con el método PROBE, estimaron de igual forma cada proyecto, siendo esas estimaciones las más cercanas al esfuerzo real de desarrollo. Con lo anterior tenemos dos métodos que para la muestra fueron los que mejor estimaron el esfuerzo de desarrollo de software: Puntos de Historias de Usuario y PROBE.

Como parte del trabajo de tesis, se ha establecido la implementación de un método de estimación de micro-proyectos de software, pero dado que se tienen dos métodos, cuyas estimaciones son las más cercanas el esfuerzo real, fue necesario seleccionar solamente uno. A continuación se presentan las características que fueron tomadas al seleccionar alguno de los dos métodos para estimar el esfuerzo de micro-proyectos.

El método PROBE es una de las mejores opciones para la estimación de esfuerzo de desarrollo de micro-proyectos. A pesar de que en la Figura 12 se observa que la mayoría de los resultados obtenidos con este método están por debajo del esfuerzo real, implicando que se subestimó el esfuerzo, este método asegura que las estimaciones serán bastante exactas, siempre y cuando se tengan datos históricos y antes de comenzar las estimaciones los requerimientos hayan sido perfectamente definidos y acordados con el cliente, ya que estos no cambiaran a lo largo del desarrollo. La definición de requerimientos toma tiempo ya que debe ser lo más clara y exacta posible. Una vez hecha la estimación con base en los requerimientos y haber acordado la fecha de liberación, el proveedor no ve al cliente hasta la entrega del producto terminado. Si al final el resultado no es el esperado se tomará tiempo extra no estimado para corregir el producto y hacer las debidas pruebas de funcionalidad, alargando la entrega lo cual incrementará el costo.

Si el proveedor de software trabaja bajo un esquema de mejora de procesos como PSP/TSP y cumple con las condiciones para que PROBE obtenga estimaciones más exactas (tener datos históricos y requerimientos bien definidos) previos al inicio del proyecto, este método es recomendable para estimar el esfuerzo de desarrollo de un micro-proyecto de software, si no se cumplen las condiciones se caerán en subestimaciones, dado que este método de estimación es utilizado en PSP/TSP cuyo esquema de trabajo consiste en establecer fecha y costo fijo, los resultados pueden ser catastróficos.

Al igual que el método PROBE, las estimaciones realizadas con el método de Puntos de Historias de Usuario con Planning Poker dieron los resultados más cercanos a los valores reales de esfuerzo, y así como PROBE estos estuvieron por debajo del esfuerzo real. La estimación con Puntos de Historias de Usuario se vuelve cada vez más exacta conforme el equipo de desarrollo, encargado de la estimación, adquiere más experiencia como desarrollador y estimador a traves del tiempo y con práctica. Mientras más experimentado sea el equipo de desarrollo, las estimaciones serán más exactas.

Dado que los Puntos de Historias de Usuario se ocupan en metodologías ágiles, no se establecen plazos de entrega y costos fijos, lo que significa que la fecha y el costo van siendo definidos conforme avanzan las iteraciones. Por lo tanto la estimación es un punto de referencia inicial que indica aproximadamente cuanto tiempo tomará el desarrollo y aproximadamente cuánto costará, si este se alarga más de lo previsto no causará tantos conflictos entre el dueño del producto y el equipo de desarrollo. Si el proveedor de software desarrolla con base en un marco de trabajo bajo los principios del desarrollo ágil, como SCRUM, este método es recomendable para estimar el esfuerzo de desarrollo de un micro-proyecto de software.

Comparando el método PROBE con el método de Puntos de Historias de Usuario, el segundo tiene la ventaja que le da el marco de trabajo ágil, al evitar conflictos con los clientes por incumplimiento de entregar a tiempo y con el costo acordado un proyecto determinado, porque de antemano ambas partes saben que el

tiempo y costo pueden variar. Al contrario con PROBE si llegase a demorarse la liberación del proyecto y el costo es mayor a lo acordado, es muy posible que termine con dificultades o incluso no se termine.

En conclusión tanto el método de Puntos de Historias de Usuario como el método PROBE por sus características son adecuados para estimar el esfuerzo de desarrollo de un proyecto de corta duración (micro-proyecto) y la utilización de uno u otro dependerá del contexto en el cual se desenvuelva el proyecto.

P4. ¿Los micro-proyectos de software tienen características especiales que requieren métodos de estimación muy específicos?

R4. En efecto los micro-proyectos de software deben cumplir tres características especiales:

- El tiempo de desarrollo proyectado sea menor a 1.5 meses.
- El presupuesto estimado no rebase los 50,000 pesos.
- El tamaño del equipo (integrantes) sea como máximo de dos personas (en cualquier rol).

Como ya se ha dicho a lo largo de este documento, los métodos de estimación más utilizados fueron diseñados hace algunos años, basándose en desarrollos de software que eran generalmente llevados a cabo en un plazo de meses, incluso años, con un gran número de programadores y con un presupuesto económico considerable, ya que en ese entonces los clientes que más demandaban herramientas de software eran comúnmente grandes empresas, que tenían el recurso económico y la gran necesidad de la herramienta, por tal razón no les importaba esperar demasiado tiempo ni pagar grandes sumas de dinero, con tal de tener algo que les ayudara a hacer más fácil su trabajo. Los diferentes investigadores interesados en encontrar una forma de estimar con mayor exactitud el esfuerzo de desarrollo de software, utilizaron esos datos de esfuerzo, tiempo y costo para diseñar métodos que se ajustaran a los valores "normales" de su tiempo. Con el transcurso de los últimos años, las metodologías de desarrollo, los lenguajes de programación y la tecnología, han evolucionado para hacer más fácil el desarrollo de software. En la actualizad hacer uso de aplicaciones informáticas es común e incluso rutinario. Al ser tan común el uso de software, las empresas dedicadas al desarrollo han entrado en un círculo competitivo por sacar al mercado productos con más y mejores características (elevando la complejidad de las aplicaciones) que los productos de sus rivales en el menor tiempo posible, caracterizando así ese tiempo de desarrollos como micro-proyectos o de corta duración.

Las metodologías de estimación desarrolladas hace tiempo y las cuales son más conocidas no han sido muy eficaces para estimar este tipo de proyectos. Retomando la respuesta **R3** de la pregunta **P3**, los métodos de estimación seleccionados que mejor se adecúan a los proyectos de software de corta duración son el método de estimación de Puntos de Historias de Usuario (PHU) y el método PROBE, ambos son relativamente recientes, implicando que seguramente consideran los problemas actuales del desarrollo del software, problemas que también afectan a los micro-proyectos.

En concreto para responder si los micro-proyectos por sus características requieren métodos muy específicos para estimar el esfuerzo de desarrollo que necesitan, se ha llegado a la conclusión de que no es así, es decir, no es necesario un método muy específico, porque ya se encontraron dos métodos que ayudan a estimarlos y estos no fueron especialmente diseñados para los micro-proyectos, sino fueron pensados en la creciente necesidad de entregar software de calidad y en poco tiempo, objetivo que se alinea a las características de los micro-proyectos.

Recordemos que fue posible contestar las preguntas de investigación, en base a los resultados obtenidos del análisis de una muestra de 7 micro-proyectos de software, por tal razón para esta muestra se cumple todo lo afirmado anteriormente.

Con los resultados obtenidos y presentados, y después de responder cada una de las preguntas de investigación, se puede confirmar la hipótesis de investigación, la cual establece:

"Existen una o más metodologías de estimación de software que pueden ser aplicadas a los micro-proyectos de desarrollo de software con un grado de precisión que no rebase el 30% de los costos reales manejados en el

mercado mexicano. Y el método más preciso puede ser automatizado mediante el desarrollo de una aplicación informática."

Efectivamente al menos existen dos métodos, los cuales estimaron de manera más exacta los costos reales de desarrollo de micro-proyectos de una muestra de siete, llevados a cabo por empresas mexicanas de desarrollo de software, estos fueron el método PROBE y el método de Estimación de Puntos de Historias de Usuario con Planning Poker, ambos proporcionan estimaciones muy cercanas al esfuerzo real de desarrollo, se utilizan en un marco de trabajo orientado a la mejora de procesos como PSP/TSP (PROBE) o en un marco de trabajo orientado al desarrollo ágil como SCRUM (Puntos de Historias de Usuario). También se cumple que los esfuerzos estimados obtenidos no rebasaron el 30% de los resultados reales de desarrollo, al contrario los resultados fueron menores al esfuerzo promedio de desarrollo, como se explicó anteriormente.

En el caso de PROBE, las estimaciones se vuelven más exactas si se cuenta con datos históricos y requerimientos bien definidos. En el caso de Puntos de Historias de Usuario, se requiere que el equipo de desarrollo tenga mayor experiencia en el desarrollo de software. Como sea ambas condiciones se cumplen con el transcurso del tiempo.

La hipótesis de investigación también plantea que se automatizaría el método de estimación seleccionado mediante una aplicación informática. Debido a que en nuestro caso se encontraron dos métodos (PROBE y Puntos de Historias de Usuario) fue necesario seleccionar uno. Dado que ambos métodos pueden ser utilizados para la estimación de esfuerzo de desarrollo de micro-proyectos, el criterio de selección será el porcentaje de uso de Puntos de Historias de Usuario (usados en el método de estimación Planning Poker) comparado con el porcentaje de uso de Líneas de Código (PROBE mide el software en líneas de código) para medir el software en México, ver Figura 6 en el Capítulo 3. Las estadísticas indican que los Puntos de Historias son utilizados un 13% para medir el software y las Líneas de Código son utilizadas en un 8%; esto quiere decir que las metodologías ágiles se están haciendo más populares para el desarrollo del software, ver Figura 7. Ante estos resultados el método seleccionado para ser implementado fue el de estimación de Puntos de Historias de Usuario con Planning Poker, ya que se consideró importante comenzar a familiarizarse con la tendencia de desarrollo de software ágil.

El método de estimación de Puntos de Historias de Usuario con Planning Poker, se basa en la experiencia y conocimientos del equipo de desarrollo, ya que son los miembros del equipo los que realizan las estimaciones, Por lo tanto sí se puede realizar una aplicación informática que asista al equipo de desarrollo durante el proceso de estimación. El desarrollo de esta aplicación se describirá en el siguiente capítulo.

A continuación se presenta una revisión de los objetivos planteados antes de comenzar la investigación, para corroborar que se han alcanzado. Fueron planteados un objetivo general y seis objetivos específicos los cuales dieron dirección y motivación al presente trabajo de Tesis.

Objetivo General

Analizar los métodos de estimación más utilizados y conocidos para compararlos y aplicarlos en una muestra de proyectos de software de corta duración con el fin de identificar cuál de estos se ajusta a los costos reales del mercado mexicano. Finalmente desarrollar una aplicación informática que implemente el método identificado para estimación de esfuerzo y costo de micro-proyectos de software.

El objetivo general debió cumplirse al alcanzarse los seis objetivos específicos que se describen a continuación, por lo tanto si los objetivos específicos fueron alcanzados el objetivo general también se logró. A continuación se revisa cada uno de estos objetivos.

Objetivos Específicos

- *Investigar los métodos existentes para la estimación de software.*
- *Analizar cada uno de estos métodos de estimación.*

- *Comparar cada método con los otros para determinar sus ventajas y desventajas.*
- *Seleccionar cinco métodos para aplicarlos a una muestra de 20 micro-proyectos de software.*
- *Realizar el análisis para determinar cuál es el método que estima con mayor precisión los costos de desarrollo de los micro-proyectos, comparando los resultados obtenidos con los costos reales.*
- *Desarrollar una aplicación gráfica para la estimación de micro-proyectos.*

Investigar los métodos existentes para la estimación de software: En la práctica es tal vez imposible identificar y analizar todos los métodos existentes, por lo tanto lo que realmente se hizo fue recopilar los métodos más conocidos, utilizados y sobre todo mencionados en la literatura dentro del ámbito de la Ingeniería del Software. Como se explicó en el Capítulo 3 para este proceso de búsqueda y selección de métodos se realizó un Revisión Sistemática de Literatura. La búsqueda utilizó una serie de parámetros de inclusión y exclusión de información, con la finalidad de encontrar artículos, libros, estadísticas, etc. con información substancial relacionada a los métodos de estimación de esfuerzo de desarrollo de software y que ayudaron a la selección.

Analizar cada uno de estos métodos de estimación: una vez recopilados los métodos más conocidos y utilizados, estos fueron descompuestos en partes: métricas, entradas, salidas y proceso para identificar las diferencias entre métodos, ventajas y desventajas.

Comparar cada método con los otros para determinar sus ventajas y desventajas: la comparación entre todos los métodos no fue posible ya que algunos por sus características no tenían parámetros de comparación, tal es el caso de los métodos de estimación heurísticos, por ejemplo Wide Band Delphi, y los métodos de estimación paramétricos, como COCOMO II, por mencionar unos ejemplos. Respecto a las ventajas y desventajas de cada método, en algunos casos no se encontró este tipo de información.

Seleccionar cinco métodos para aplicarlos a una muestra de 20 micro-proyectos de software: este objetivo se puede dividir en dos partes. La primera tiene que ver con el número de métodos seleccionados, el enunciado indica que 5 y estos fueron, Puntos de Función, COCOMO II, Puntos de Casos de Uso, Puntos de Historias de Usuario y PROBE, la selección fue realizada con base a estadísticas del porcentaje de uso de las métricas de medición utilizadas por cada método de estimación. En efecto son 5 diferentes, pero para efectos prácticos se contabilizan 4 ya que para estimar esfuerzo de desarrollo se combinan el método Puntos de Función y el método COCOMO II. Puntos de Función estima el tamaño del software y COCOMO II estima el esfuerzo necesario para el desarrollo. La segunda parte del objetivo, tiene que ver con el tamaño de la muestra de micro-proyectos a analizar, al inicio se especificó que sería de 20, pero solo se reunieron 7 micro-proyectos; los cuales fueron proporcionados por tres empresas desarrolladoras de software mexicanas. A pesar de la insistencia con otras empresas, no fue posible obtener más micro-proyectos muestra.

Realizar el análisis para determinar cuál es el método que estima con mayor precisión los costos de desarrollo de los micro-proyectos, comparando los resultados obtenidos con los costos reales: en esta parte a cada proyecto de la muestra se le aplicaron los cinco métodos de estimación de esfuerzo seleccionados: Puntos de Función- COCOMO II, Puntos de Historias de Usuario con Planning Poker, Puntos de Casos de Uso y PROBE. Como los proyectos ya habían sido concluidos se contaba con el esfuerzo real de desarrollo de cada uno, lo que permitió comparar los resultados de las estimaciones con el valor real, y de esta forma determinar que método arrojaba resultados más cercanos al real. Como resultado del análisis se llegó a la conclusión de que tanto el método de estimación con Puntos de Historias de Usuario y PROBE son adecuados para estimar micro-proyectos.

Desarrollar una aplicación gráfica para la estimación de micro-proyectos: este es el último objetivo a alcanzar del trabajo de Tesis. Como se mencionó anteriormente se concluyó que existen dos métodos que son apropiados para estimar los micro-proyectos, pero para cumplir este objetivo, fue necesario seleccionar uno de los dos métodos. Estimación de Puntos de Historias de Usuario utilizando Planning Poker fue el método seleccionado para ser implementado. Dadas las características del método, no fue posible implementarlo al 100%, ya que sus resultados dependen de la experiencia de las personas que realizan las estimaciones, por tal razón se desarrolló una aplicación que asiste a los estimadores durante el proceso, permitiendo llevar un registro de los resultados de la estimación. Este método de estimación es utilizado dentro del marco de trabajo

ágil llamado SCRUM, la aplicación además del método de estimación Planning Poker, implementa el proceso básico del seguimiento del desarrollo de software que especifica SCRUM.

Una vez revisados cada uno de los objetivos específicos, es momento de revisar si el objetivo general de este trabajo de Tesis se ha cumplido.

Como indica el objetivo general se analizaron los métodos de estimación más utilizados y conocidos en la literatura en el ámbito de la Ingeniería del Software. El análisis de los métodos encontrados consistió en identificar las métricas que utiliza el método, sus entradas, salidas y pasos para realizar la estimación. Para seleccionar los 4 métodos de estimación que se aplicaron a la muestra de micro-proyectos se analizaron estadísticas de frecuencia de uso de métricas para medir el software, esas métricas son utilizadas en los métodos de estimación de esfuerzo seleccionados. La etapa de evaluación de métodos de estimación sobre una muestra de micro-proyectos se realizó completamente, analizando cada proyecto con los cinco métodos de estimación seleccionados y comparando los resultados con los valores reales del esfuerzo de desarrollo. El objetivo indica que como resultado de la investigación y el análisis se debía encontrar un método que estimara mejor los micro-proyectos, pero se determinó que existen dos: Estimación con Puntos de Historias de Usuario con Planning Poker y PROBE. Para el último punto del objetivo general se seleccionó el método de Puntos de Historias con Planning Poker, para su implementación. Como se mencionó con anterioridad el método no se automatizó al 100% como indica el objetivo.

El objetivo del capítulo fue presentar los resultados obtenidos tras el análisis de una muestra de 7 micro-proyectos de software, a la cual se le aplicaron 5 métodos de estimación de esfuerzo diferentes, los resultados obtenidos fueron analizados y sintetizados, para dar respuesta a las preguntas de investigación e hipótesis que fueron la guía para realizar la investigación de este trabajo de Tesis. También se realizó una revisión para corroborar que los objetivos específicos y objetivo general del trabajo de investigación se hayan cumplido.

Capítulo 5. Desarrollo de la Aplicación de Software

En este capítulo se describe la aplicación que se desarrolló para la administración de micro-proyectos de software con el marco de trabajo SCRUM, el capítulo inicia con una breve introducción sobre SCRUM, posteriormente se describe el análisis y diseño realizados para el desarrollo del software. Como evidencias del diseño se presentan diagramas de actividades de las historias de usuario más significativas de la aplicación y para mayor entendimiento del funcionamiento también se muestra el diagrama de clases así como el diagrama entidad relación de la base de datos. El capítulo concluye con el manual de usuario del sistema.

5.1 Introducción

En 1996 Jeff Sutherland y Ken Schwaber presentaron un modelo de desarrollo ágil, iterativo e incremental para desarrollar y mantener sistemas de software. Este modelo de gestión de desarrollo es conocido como SCRUM (Palacio & Ruata, 2009).

SCRUM no es un modelo prescriptivo ya que no indica exactamente lo que los desarrolladores deben hacer. SCRUM es un marco de trabajo, por lo tanto describe buenas prácticas a seguir para la administración del desarrollo de software, bajo los 4 principios básicos descritos en el manifestó ágil (Scrum Bok, 2013), listados a continuación:

- Entregar componentes funcionales sobre documentación.
- Reaccionar a los cambios sobre apegarse a un plan.
- Personas sobre procesos.
- Colaboración con el cliente sobre acuerdos contractuales.

La forma de trabajo al seguir SCRUM es iterativa incremental, las iteraciones son llamadas Sprints. Los Sprints son periodos de tiempo definidos. En ese periodo de tiempo el equipo trabaja para tener como resultado un componente funcional de software que puede ser liberado.

El marco de trabajo SCRUM está fundamentado principalmente por un conjunto de Roles, Reuniones y Artefactos.

Roles

En un Sprint se definen tres roles principales:

- SCRUM Master o Experto SCRUM, la persona con este rol es un experto en SCRUM, entre sus funciones principales se encuentran: verificar que se siga el marco de trabajo, resolver los problemas que impiden al equipo hacer su trabajo y facilitar la comunicación entre el Product Owner y el equipo de desarrollo.
- Product Owner o Dueño del Producto, la función principal de este rol es establecer la funcionalidad del componente de software que será construido por el equipo de desarrollo. Solo este rol puede decidir que funciones se implementan, se quitan o se cambian.
- Team Developer o Equipo de desarrollo, es el encargado de la construcción del software, el equipo es quien decide cómo se reparte el trabajo y es responsable de que la entrega se realice al final del Sprint.

Reuniones

SCRUM define cinco reuniones básicas:

- Release Planning o Planeación de la liberación, esta reunión se divide en dos partes, la primera parte tiene como objetivo definir la pila del producto y en la segunda parte se estima el esfuerzo de desarrollo requerido, el número de iteraciones (Sprints) que tendrá el proyecto y las piezas de software funcionales que se entregarán al final del proyecto.
- Sprint Planning o Planeación del Sprint, esta reunión establece el objetivo del Sprint, el entregable que se desarrollará, las actividades necesarias a realizar y los criterios de aceptación que debe cubrir la pieza funcional de software a liberar definida en el Sprint.
- Daily Meeting o Reunión Diaria, cada equipo de desarrollo se reúne al final del día y cada integrante contesta las siguientes preguntas:

    1. ¿Qué hice hoy?
    2. ¿Qué problemas o complicaciones tengo en la realización de mis actividades?
    3. ¿Qué haré mañana?

    Esta reunión dura como máximo 15 minutos y al final se establecen las estrategias para afrontar las complicaciones expuestas si las hubiera.

- Sprint Review o Revisión del Sprint, esta reunión se realiza para hacer la demostración del software producido durante el Sprint. La demostración es hecha al final del Sprint y se realiza ante el Product Owner y Stateholders (otras personas relacionadas al proyecto) para que validen que el software construido cumple con todos los requerimientos esperados. En caso de que sea necesario realizar modificaciones, en esta reunión el Product Owner identifica los cambios.
- Sprint Retrospective o Retrospectiva del Sprint, esta reunión se sostiene solo por los integrantes del equipo de desarrollo y se analiza con detenimiento que se hizo bien, que se hizo mal y que se puede mejorar para el siguiente Sprint. La reunión de retrospectiva se realiza al siguiente día, después de la revisión del Sprint.

Artefactos

Los artefactos definidos por SCRUM son:

- Product Backlog o Pila del Producto, es una lista priorizada de requisitos, historias o funcionalidades, en otras palabras, son cosas que el cliente quiere, descritas usando su terminología, estos son los elementos de la Pila o Product Backlog Items (PBIs).
- Sprint Backlog o Pila del Sprint, contiene la lista de historias que se implementarán durante un Sprint, las historias son subdivididas en tareas y se les asignan horas estimadas de terminación.
- Sprint Burndown, este artefacto consiste en una gráfica que relaciona el trabajo faltante por realizar en lo que resta del Sprint, la gráfica se actualiza diariamente después de la reunión diaria, sirve para detectar si el equipo se está retrasando o va adelantado.
- Release Burndown, este artefacto también es un gráfica que relaciona el trabajo faltante por realizar en los Sprints restantes, esta gráfica se actualiza al final de cada Sprint.
- SCRUM Board, es utilizado para llevar el seguimiento del trabajo realizado durante un Sprint. Este artefacto muestra de manera gráfica las historias pendientes por realizar, las que están en progreso y las terminadas.

Los elementos básicos del marco de trabajo SCRUM descritos anteriormente, fueron retomados como parte de los requerimientos funcionales de la aplicación de software que permite la estimación de esfuerzo de desarrollo con Planning Poker de micro-proyectos, así como su administración con SCRUM.

La aplicación permite asignar los roles de Product Owner, SCRUM Master y Equipo de Desarrollo en un proyecto específico, construir los artefactos mencionados para definir y administrar el trabajo, así como llevar un registro del avance del desarrollo. Respecto a las reuniones, la aplicación está diseñada para ayudar a realizar actividades propias de cada reunión, por ejemplo, en la reunión de Release Planning, el programa permite crear la Pila del Producto, estimar el esfuerzo de desarrollo y calcular un número aproximado de Sprints necesarios para la liberación del producto. La estimación de esfuerzo de desarrollo con el método Planning Poker es de las actividades principales en esta reunión, así mismo, es una función medular cuyo resultado sirve como punto de referencia para determinar si existen o no retrasos durante el desarrollo. Como por naturaleza el método Planning Poker es un juego de cartas, éste fue implementado como un juego de cartas en una red local.

Para la reunión Sprint Planning, el sistema permite asignar las historias que se realizarán en el tiempo que durará el Sprint, establecer el objetivo del Sprint y asignar tareas a cada miembro del equipo. Respecto a la reunión diaria o Daily Meeting, el sistema permite actualizar el ScrumBoard y la gráfica Sprint Burndown diariamente al terminar la reunión.

Como se explicó anteriormente al final de cada Sprint se realiza la reunión de revisión o Sprint Review, donde se hace la demostración del software construido. Para esta reunión el sistema únicamente actualiza la gráfica Release Burndown y permite visualizar el trabajo pendiente para los Sprints restantes. Por último el sistema facilita el registro de las conclusiones obtenidas por el equipo al final de la reunión de retrospectiva o Sprint Retrospective, para llevar un seguimiento de mejoras en el proceso de desarrollo.

La aplicación que permite la estimación de esfuerzo de desarrollo con Puntos de Historias de Usuario y la Administración del Desarrollo de Micro-Proyectos de software fue gestionado con el marco de trabajo SCRUM, en la sección siguiente se describe el proceso de desarrollo dividido en las etapas de análisis, diseño e implementación, esto es con el fin de describir de manera ordenada el trabajo realizado. Cabe aclarar que estas etapas no son definidas explícitamente por SCRUM, sino que son etapas básicas de cualquier desarrollo de software.

5.2 Análisis

El método de estimación de Puntos de Historias de Usuario con Planning Poker fue implementado siguiendo la metodología de desarrollo ágil SCRUM.

En todo desarrollo de software deben quedar especificadas todas las funcionalidades que el producto final debe cumplir. Esas funcionalidades o requerimientos funcionales, son especificados por el cliente y son recabados durante la fase de análisis dentro del ciclo de vida de un desarrollo de software.

En el proceso ágil de desarrollo para especificar los requerimientos del cliente se utiliza un formato llamado historia de usuario. La historia de usuario es un enunciado que describe quien será la persona <rol> que hará uso de la funcionalidad, una descripción de lo que se quiere hacer con el sistema <actividad> y una descripción del porque se desea esa funcionalidad <valor para el cliente>.

Las historias de usuario definidas para la elaboración del sistema de estimación de esfuerzo de desarrollo y administración de micro-proyectos bajo el marco de trabajo SCRUM están ordenadas por prioridad y se

muestran en la Tabla 23. En esta misma tabla se incluyen los puntos de historia estimados para su desarrollo. Los valores fueron estimados utilizando el método Planning Poker.

Tabla 23. Historias de Usuario del Sistema para la Administración de Proyectos con SCRUM ordenadas por prioridad

ID	HISTORIA DE USUARIO	PRIORIDAD	PHU
1	Yo como Product Owner quiero agregar PBIs al Product Backlog para representar mi funcionalidad deseada	1	13
3	Yo como Product Owner quiero asignar una prioridad a los PBIs, para registrar la importancia que estos tiene para mi	2	5
11	Yo como SCRUM Master quiero registrar la estimación que cada integrante asigna a una historia de usuario, para ejecutar adecuadamente el Planning Poker	3	21
12	Yo como SCRUM Master quiero registrar la estimación final de una historia de usuario, para que ésta quede grabada en el historial.	4	5
14	Yo como SCRUM Master quiero arrastrar tareas a la pila del Sprint, para definir las historias de usuario que se desarrollarán en dicho Sprint.	5	21
15	Yo como SCRUM Master quiero capturar el objetivo del Sprint, para tener claro dicho objetivo.	6	3
20	Yo como SCRUM Master quiero arrastrar los PBIs del Product Backlog hacia diferentes columnas del SRUM Board según sea su status durante el Sprint, para visualizar todo el desarrollo del proyecto	7	13
10	Yo como SCRUM Master quiero dar de alta a mi equipo de trabajo para controlar el número de integrantes y sus tareas.	8	21
2	Yo como Product Owner quiero agrupar mis PBIs en funcionales y tecnológicos para identificarlos mejor	9	8
4	Yo como Product Owner quiero categorizar mis PBIs en módulos funcionales, para clasificarlos en base a funcionalidad	10	8
6	Yo como Product Owner quiero modificar la prioridad, categoría o contenido de cualquier PBI, para actualizar mis requerimientos con base en mis necesidades	11	5
21	Yo como Miembro del Equipo quiero asignar una lista de actividades a cada historia de usuario e indicar la duración en horas de cada actividad, para llevar un control de mis tareas	12	13
7	Yo como Product Owner quiero registrar las pruebas de aceptación de cada PBI, para indicar el criterio de finalización de cada PBI	13	13
5	Yo como Product Owner quiero consultar el status de cada PBI, para saber si están en progreso, pendientes o terminados.	14	5

24	Yo como Miembro del Equipo quiero actualizar el estado de las pruebas de aceptación para actualizar el avance del Sprint	15	8
23	Yo como Miembro del Equipo deseo consultar los registros de los Daily Meeting, para saber el estado de los problemas y sugerencias	16	21
8	Yo como Product Owner quiero revisar el status de las pruebas de aceptación para saber si el PBI ha quedado culminado.	17	5
22	Yo como Miembro del Equipo deseo actualizar el estado de las tareas de cada historia de usuario, para actualizar el avance	18	5
16	Yo como SCRUM Master quiero generar la gráfica Sprint Burndown Chart, para dar seguimiento al avance del Sprint.	19	21
17	Yo como SCRUM Master quiero generar la gráfica Release Burndown Chart, para dar seguimiento a todo el proyecto.	20	21
18	Yo como SCRUM Master quiero registrar todas las conclusiones del Sprint Retrospective, para registrar la retroalimentación obtenida en la reunión	21	21

El esfuerzo de desarrollo estimado para implementar el sistema siguiendo el marco de trabajo SCRUM fue 217 Puntos de Historias de Usuario (PHU) y el tiempo en horas estimado fue 210, aproximadamente 5 semanas y media de desarrollo, en el Anexo L se muestra de manera detallada la división de tareas por historia de usuario y el tiempo estimado para cada una de ellas.

5.3 Diseño

SCRUM no especifica un formato para representar el diseño del proyecto, ya que deja la selección al Equipo de Desarrollo.

A continuación se describe el diseño reflejado en diagramas de actividades de las historias de usuario más importantes y complejas dentro de la aplicación.

Dentro de las funcionalidades más importantes del sistema están el módulo para la construcción de la pila del producto o Product Backlog y el módulo que permite la estimación de Puntos de Historias de Usuario con Planning Poker de la pila del producto.

5.3.1 Construcción del Product Backlog

La construcción de la pila del producto es realizada por el dueño del producto o Product Owner, y su tarea es agregar, modificar y/o quitar historias de usuario que describen la funcionalidad deseada del producto, ver diagrama de actividades en la Figura 14. Una vez construida la pila del producto es analizada por el Equipo, para estimar el esfuerzo de desarrollo.

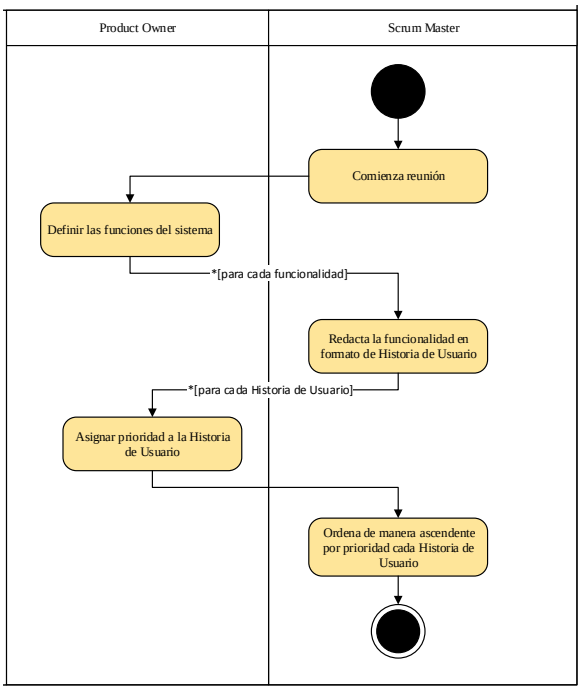
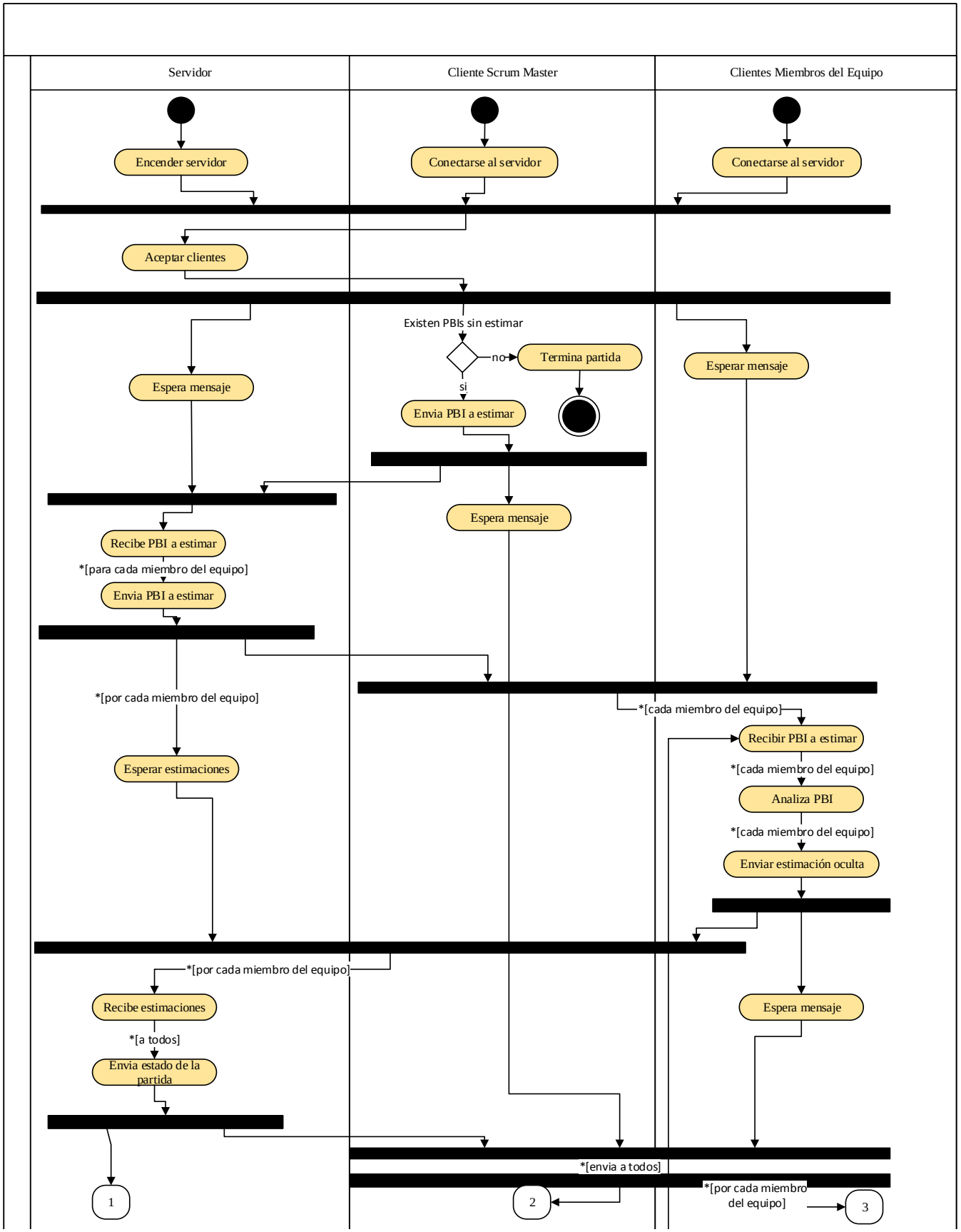


Figura 14. Construcción de la pila del producto (Product Backlog)

5.3.2 Estimación de esfuerzo de desarrollo a Historias de Usuario

La estimación de Historias de Usuario con Planning Poker es una actividad de estimación que se desarrolla en grupo y las estimaciones son el resultado del juicio y experiencia de cada miembro del equipo, la dinámica se desarrolla como un juego de cartas similar al Poker clásico. Esta funcionalidad fue implementada de tal manera que asista a un equipo de desarrollo en el proceso de estimación, se diseñó como un juego de cartas en una red local, que permite a cada miembro del equipo simular una partida de cartas. En la Figura 15 se muestra el diagrama de actividades que se debe seguir para realizar la estimación de esfuerzo, cuya unidad de medida son los Puntos de Historias de Usuario. En la Figura 16 se muestra el proceso a seguir para realizar la estimación del tiempo en horas que tomará terminar cada PBI, la suma total de tiempo, es el tiempo estimado para el desarrollo de todo el proyecto.


Servidor
Cliente Scrum Master
Clientes Miembros del Equipo
Encender servidor
Conectarse al servidor
Conectarse al servidor
Aceptar clientes
Existen PBIs sin estimar
no
Termina partida
si
Envia PBI a estimar
Esperar mensaje
Espera mensaje
Espera mensaje
Recibe PBI a estimar
*[para cada miembro del equipo]
Envia PBI a estimar
*[por cada miembro del equipo]
*[cada miembro del equipo]
Recibir PBI a estimar
*[cada miembro del equipo]
Analiza PBI
*[cada miembro del equipo]
Enviar estimación oculta
Esperar estimaciones
*[por cada miembro del equipo]
Recibe estimaciones
Espera mensaje
*[a todos]
Envia estado de la partida
*[envia a todos]
*[por cada miembro del equipo]
1
2
3

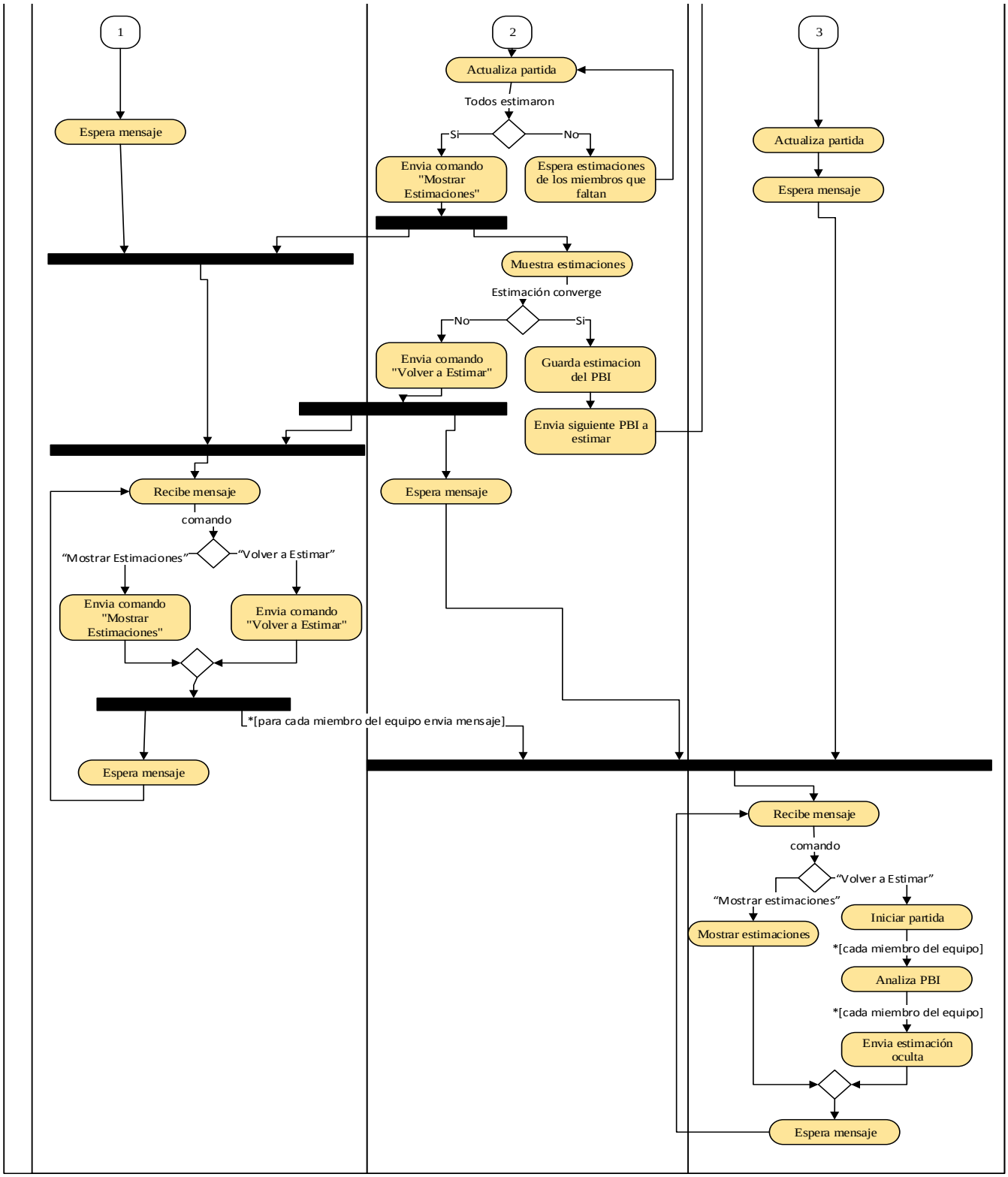


Figura 15. Diagrama de actividades para la estimación de esfuerzo de desarrollo

A continuación se muestra el diagrama de actividades para la estimación del tiempo de desarrollo.

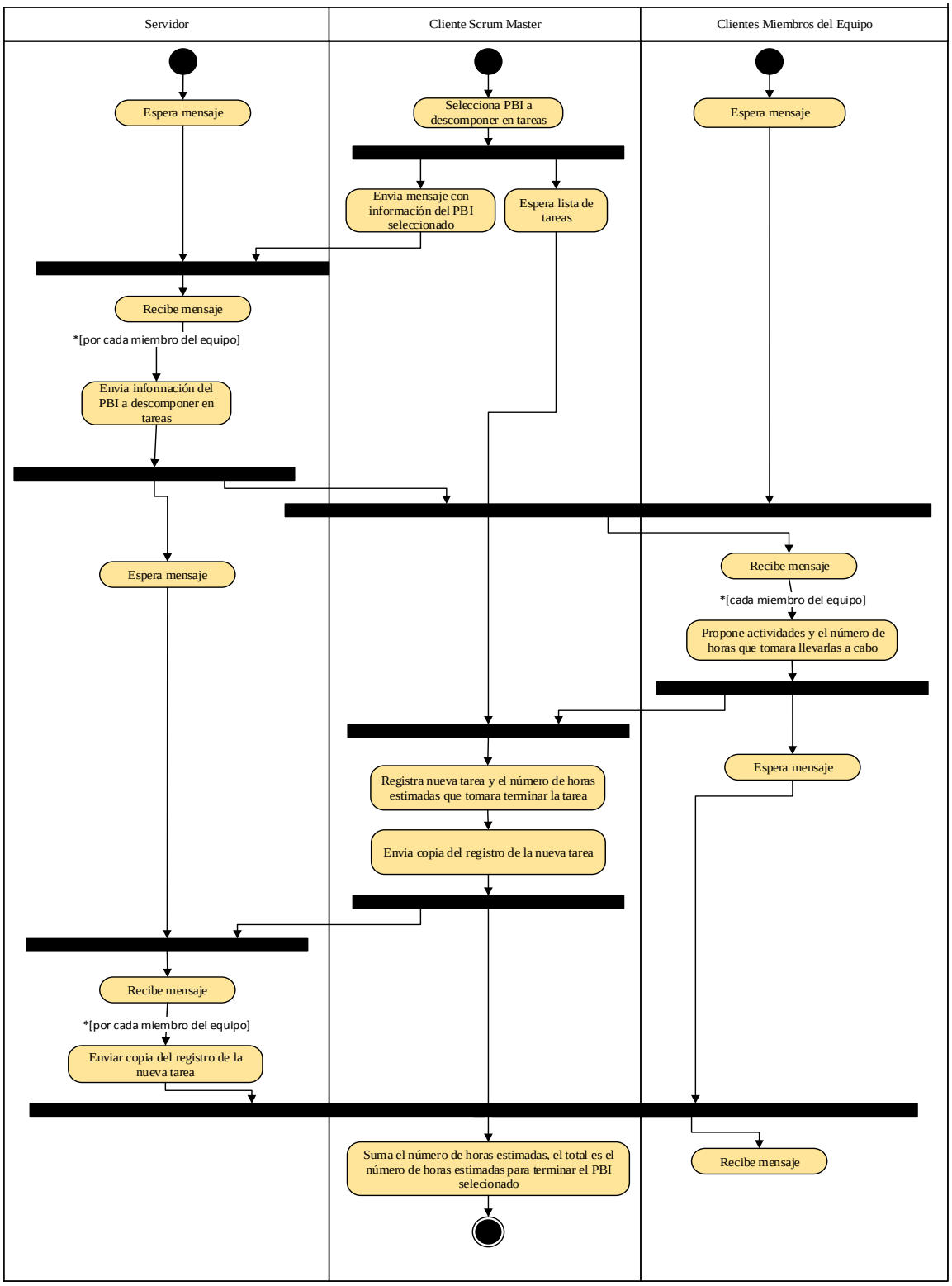


Figura 16. Diagrama de actividades para la estimación de tiempo de desarrollo

Cabe señalar que es importante como muchos otros sistemas, guardar datos e información del proyecto a desarrollarse, por tal razón, fue necesario realizar un diseño de base de datos. El modelo entidad relación diseñado que representa la base de datos es el que se muestra en la Figura 17.

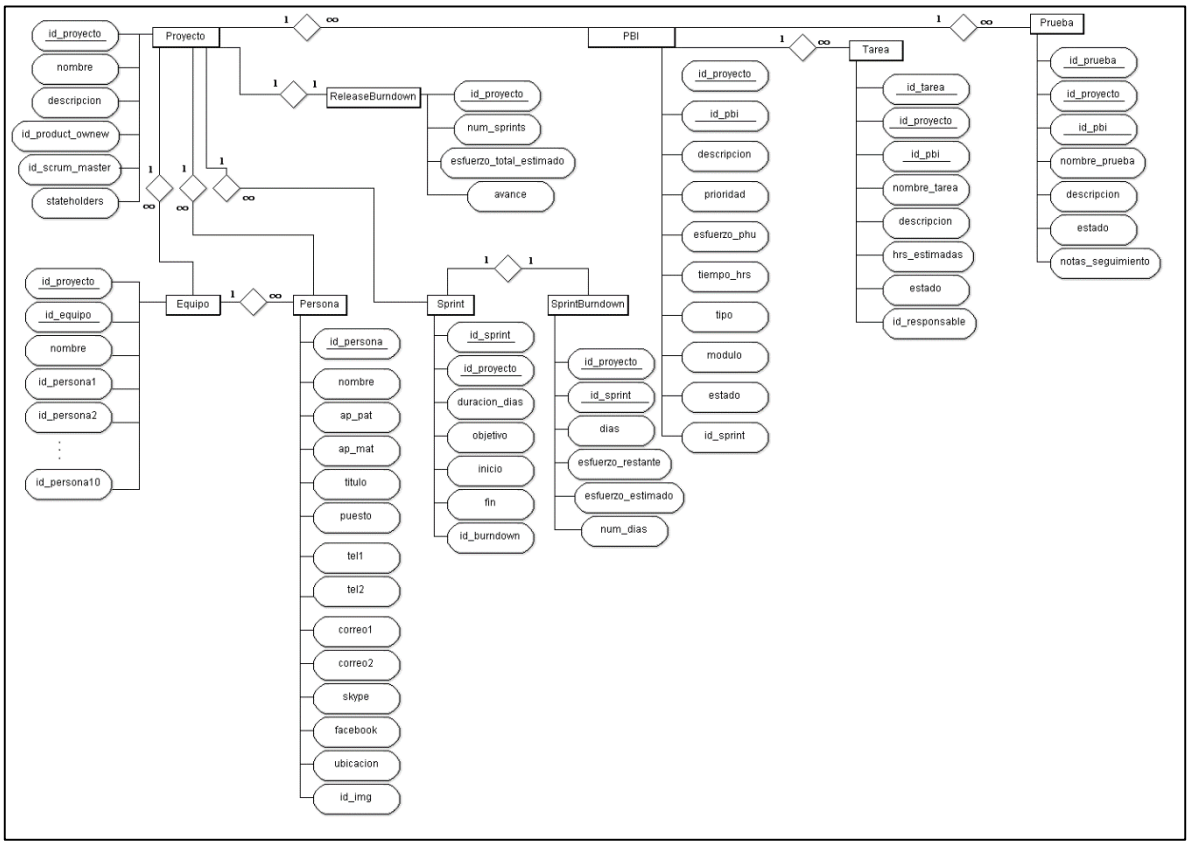


Figura 17. Diagrama entidad-relación del sistema

A continuación se describen las tablas del modelo y la información que contiene cada una de ellas:

Tabla Proyecto

Esta tabla contiene los datos de los proyectos que fueron realizados por la empresa, así como también los proyectos que están en curso. Los campos que contiene son un número único de identificación del proyecto, nombre del proyecto, descripción general del proyecto, un identificador de la persona con el rol de Product Owner, un identificador de la persona con el rol de SCRUM Master y una lista de personas relacionadas con el proyecto (stateholders).

Tabla Equipo

Esta tabla contiene la relación de los miembros pertenecientes a un equipo. Un proyecto puede tener más de un equipo de desarrollo por lo tanto el programa permite asignar varios equipos a un mismo proyecto, los equipos trabajan en paralelo. El tamaño de un equipo va de una persona a diez. La información que ésta tabla guarda es: el identificador del proyecto al que pertenece el equipo, un identificador para el equipo, un nombre de equipo y los identificadores de las personas que son miembros del equipo.

Tabla Persona

Esta contiene los datos de todos los desarrolladores de la empresa, los cuales pueden tomar el rol de Product Owner, SCRUM Master o Miembro de un equipo. La información que se almacena es la siguiente: un identificador único por persona, nombre(s), apellido paterno, apellido materno, título (licenciado, ingeniero, maestro, doctor, etc.), su puesto de trabajo, dos números de teléfono, dos cuentas de correo electrónico, cuenta de skype, cuenta de Facebook, su ubicación dentro de la empresa (números de cubículo u oficina), y un identificador para guardar una foto de perfil.

Tabla ReleaseBurndown

La información que contiene esta tabla está relacionada con la gráfica que permite visualizar el trabajo restante (medidos en puntos de historia) por terminar para completar todo el proyecto, es decir, al final de cada sprint se revisa cuanto trabajo falta por realizar y esa información se almacena aquí. La tabla específicamente almacena: el identificador del proyecto al que pertenece la gráfica, el número de Sprints estimados para terminar el proyecto, el esfuerzo total de desarrollo estimado en Puntos de Historias de Usuario y el avance por Sprint. El avance se registra y según sea este la aplicación realiza una diferencia entre el trabajo total estimado, el resultado es el trabajo restante por terminar.

Tabla Sprint

Esta tabla guarda el número del sprint en el que se encuentra el desarrollo, el identificador del proyecto al que pertenece el sprint, la duración en días del sprint, objetivo, fecha de inicio y fecha de fin.

Tabla SprintBurndown

Esta tabla contiene la información para construir la gráfica Sprint Burn-down, la cual sirve para llevar el seguimiento del proyecto durante un sprint. El trabajo restante por realizar durante el sprint es lo que se registra en esta gráfica. La información almacenada en la tabla es la siguiente: identificador del proyecto al que pertenece, identificador del sprint al que pertenece, un campo llamado esfuerzo restante que almacena una cadena los valores correspondientes a cada día del sprint, el valor del esfuerzo estimado para ese sprint y el número de días del sprint.

Tabla PBI

La tabla PBI es la que contiene todas historias de usuario que se ingresaron a la pila del producto. La información que se almacena es la siguiente: el identificador del proyecto al que pertenece, un identificador único para el PBI, descripción de la historia, prioridad, esfuerzo, en PHU, estimado para implementar el PBI, las horas estimadas, el tipo de PBI, módulo al que pertenece, el estado en el que se encuentra y el sprint en el cual se planea implementar el PBI. .

Tabla Tarea

Cada PBI es descompuesto en tareas, a las cuales se les asigna un tiempo en horas para llevarlas a cabo. La tabla guarda un identificador para cada tarea relacionada del PBI, el identificador del proyecto al que pertenece la tarea, identificador del PBI al que pertenece la tarea, el nombre de la tarea, descripción, horas estimadas para terminar la tarea, estado y el identificador de la persona responsable de hacer la tarea.

Tabla Prueba

La tabla almacena la información de cada una de las pruebas que un determinado PBI debe pasar para ser considerado como hecho o terminado. En la tabla se guarda un identificador por prueba, el identificador del proyecto al que pertenece, identificador del PBI asociado, nombre de la prueba, descripción, estado y unas notas de seguimiento de la prueba. El diagrama de clases que describe la implementación del módulo principal del sistema y el módulo cliente-servidor se muestra en la Figura 18.

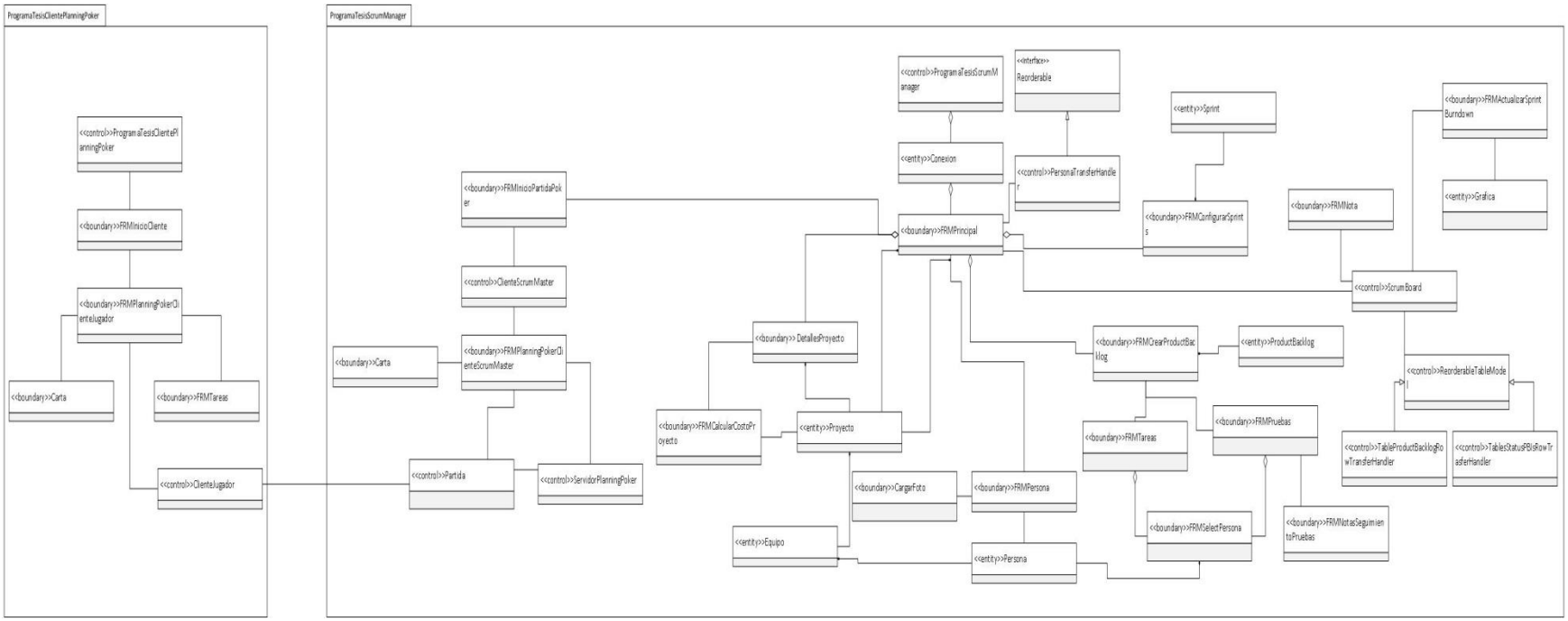


Figura 18. Diagrama General de Clases

5.4 Implementación

La aplicación fue desarrollada en el lenguaje de programación Java 6, el entorno de desarrollo fue NetBeans IDE 7.2.1 y el manejador de bases de datos utilizado fue MySQL Workbench 7.0.

El programa se divide en dos partes, ya que el sistema implementó un módulo cliente-servidor encargado del proceso de estimación de Puntos de Historias de Usuario con Planning Poker y un módulo principal para realizar el seguimiento de los proyectos bajo el marco de trabajo SCRUM.

La implementación de los diagramas de actividades y diagrama de clases, junto con la base de datos, generó como resultado una aplicación capaz de estimar el esfuerzo de desarrollo de micro-proyectos de software, así como su administración dentro del marco de trabajo SCRUM.

A continuación se muestran las pantallas más significativas de la aplicación las cuales corresponden a las funciones de registro de nuevos proyectos Figura 19, pantallas principales del módulo cliente-servidor para la estimación de esfuerzo Figura 20 y Figura 21, y la pantalla que permite el seguimiento de un proyecto con la metodología SCRUM Figura 22. La funcionalidad y modo de empleo de las pantallas se describe en el manual de usuario del Anexo M, así como las demás funcionalidades de la aplicación.

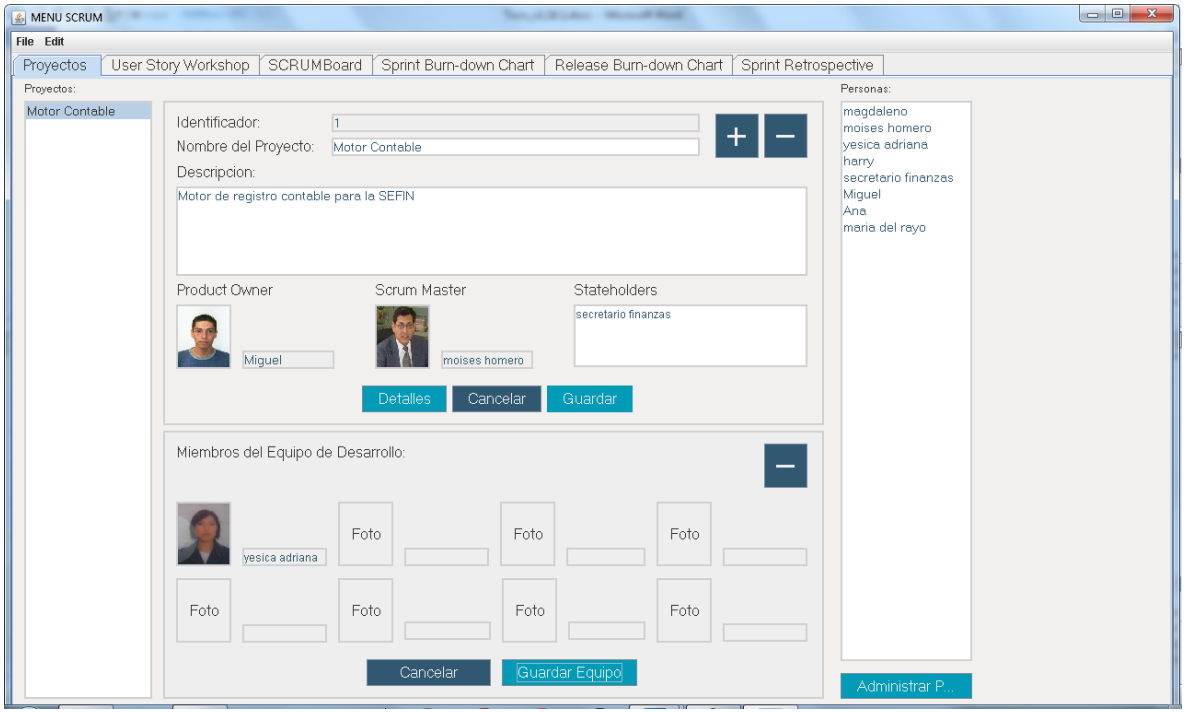


Figura 19. Pantalla para la administración de "Proyectos"

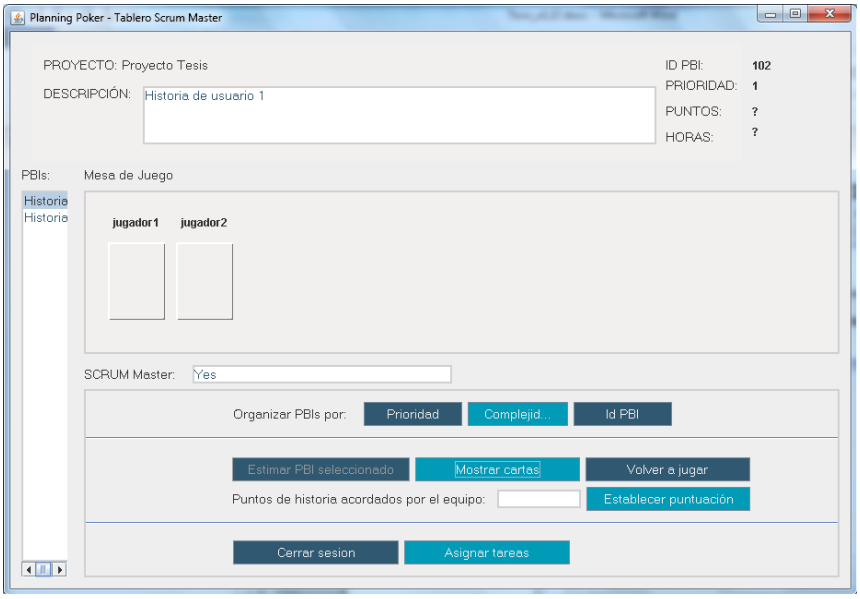


Figura 20. Pantalla Cliente SCRUM Master

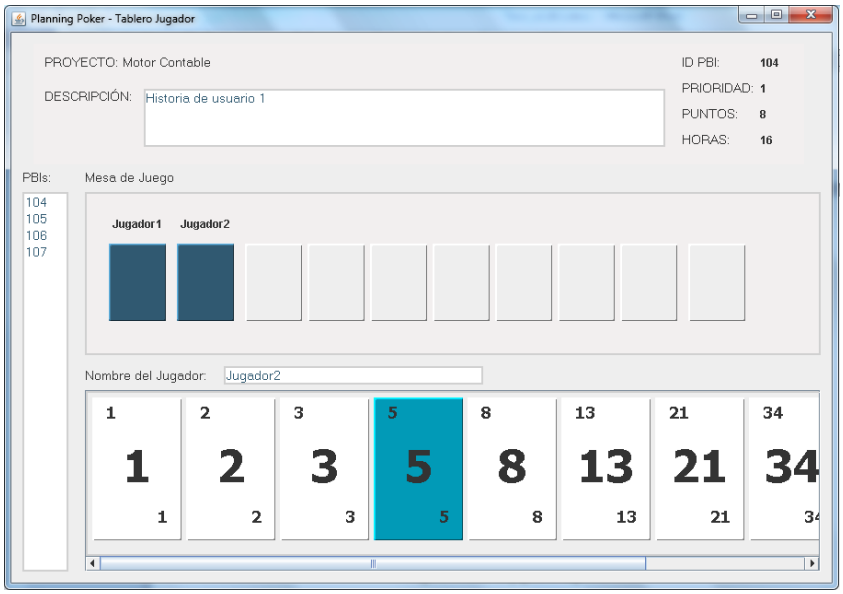


Figura 21. Pantalla de cada Cliente Jugador

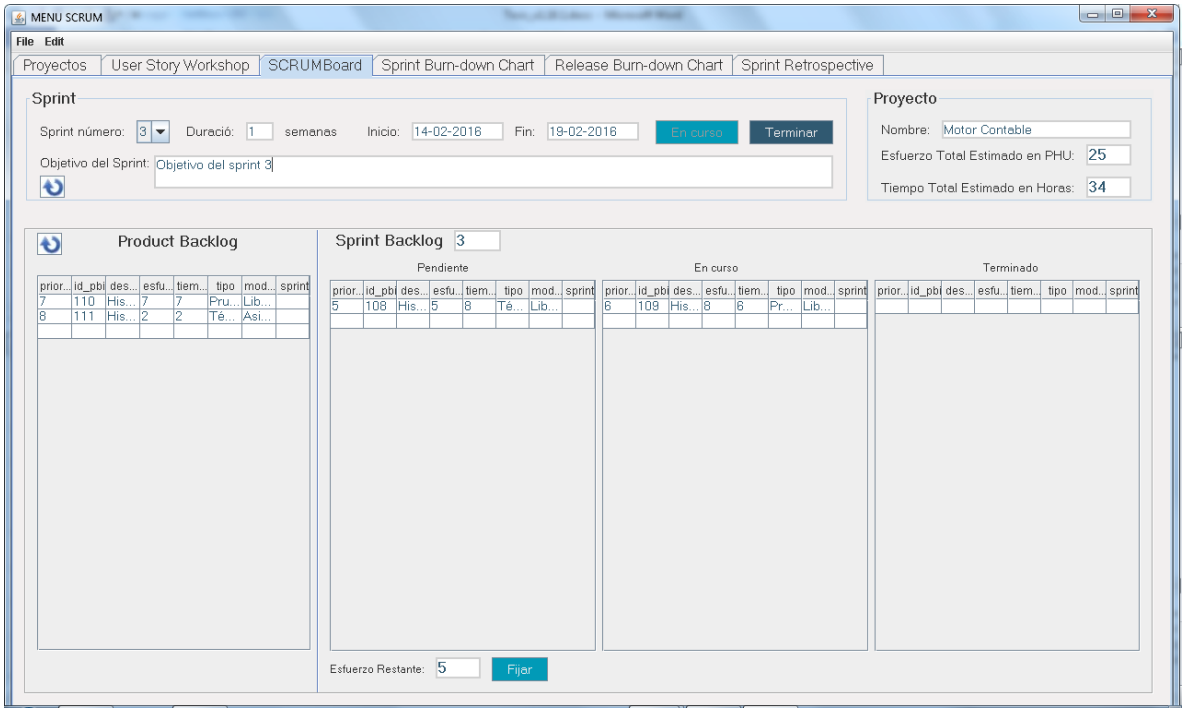


Figura 22. Pantalla SCRUM Board para el seguimiento del proyecto

El objetivo del capítulo fue presentar el análisis, diseño e implementación de la aplicación informática que permite realizar estimaciones de esfuerzo de desarrollo con el método de puntos de historias de usuario, sobre micro-proyectos de software. Así como administrar el proceso mediante el marco de desarrollo ágil SCRUM.

Capítulo 6. Conclusiones

El objetivo general de este documento de tesis fue analizar los métodos de estimación más utilizados y conocidos para compararlos y aplicarlos en una muestra de proyectos de software de corta duración (micro-proyectos) con el fin de identificar cuál de estos se ajustó al esfuerzo real de cada elemento de la muestra. Finalmente se desarrolló una aplicación informática que implemente el método identificado para estimación de esfuerzo y costo de micro-proyectos de software.

Durante la investigación de métodos de estimación de esfuerzo de software se encontró que los 5 métodos más utilizados y conocidos en la industria son los siguientes (ver sección "Estadísticas sobre métricas y métodos de estimación"):

- Puntos de Función
- COCOMO II
- Puntos de Casos de Uso
- Puntos de Historias de Usuario con Planning Poker
- PROBE

Existen dos métodos de estimación de esfuerzo que proporcionan una estimación muy cercana al esfuerzo real: El método "Puntos de Historias de Usuario con Planning Poker" y el método "PROBE". Lo anterior quiere decir que se encontraron opciones diferentes que son igualmente efectivas en la estimación de micro-proyectos de software, y la utilización de una u otra dependerá del contexto de desarrollo dentro del cual se vaya a desarrollar el proyecto.

Si el proveedor de software trabaja dentro del margen de desarrollo ágil las estimaciones con Puntos de Historias de Usuario son adecuadas para los micro-proyectos. En el caso contrario donde el proveedor trabaja bajo un régimen de mejora de procesos como PSP/TSP, la estimación con el método PROBE, obtendrá resultados muy cercanos al esfuerzo real de desarrollo de los micro-proyectos.

Se observa que en los últimos años, en México el uso de la métrica de Puntos de Historias de Usuario ha aumentado y es más utilizada (13%) para medir el software en comparación a las Líneas de Código (9%) (Ver la Figura 6 del Capítulo 3), esto quiere decir que las metodologías ágiles se están haciendo más populares para el desarrollo del software, ver Figura 7.

El método de estimación de Puntos de Historias de Usuario con Planning Poker, no se puede automatizar completamente, debido que éste método se basa en la experiencia y conocimientos del equipo de desarrollo que se encarga de realizar las estimaciones.

Las constantes utilizadas en los métodos de Puntos de Función, COCOMO II y Puntos de Casos de Uso fueron determinadas mediante un estudio aplicado a proyectos muy grandes, por lo que estos métodos resultaron ser los más alejados de los valores de esfuerzo reales en los micro-proyectos.

6.1 Aportaciones

- Las MiPYMES desarrolladoras de software pueden encontrar con esta investigación dos opciones diferentes (probadas) para la estimación de esfuerzo de desarrollo, que les sean de ayuda para estimar específicamente micro-proyectos de software de manera más exacta. Dado que aproximadamente más del 80% de los proyectos que desarrollan las MiPYMES están categorizados como micro-proyectos, estimarlos correctamente es vital para asegurar la estabilidad económica de la empresa.
- Este documento de tesis recopila varios métodos de estimación de esfuerzo de desarrollo de software, por lo tanto puede ser utilizado como recurso académico, en el área de la Ingeniería del Software.
- Se desarrolló una herramienta informática que ayuda a implementar los conceptos básicos y el proceso de la metodología de desarrollo ágil SCRUM. Una de las características más novedosas de esta herramienta es la implementación de la dinámica de Planning Poker como un juego de cartas ejecutándose en una PC

conectada a una red local, para simular la dinámica del método manual. Esta característica no ha sido observada en otras herramientas informáticas administrativas de proyectos ágiles.

- Debido a que la herramienta informática no cuenta con funciones para (clasificar cada actividad según en la etapa del ciclo de vida del software en la que se encuentra, registrar el porcentaje de avance de cada tarea del proyecto o llevar el registro de avance de múltiples equipos trabajando paralelamente en un mismo proyecto) que permitan llevar el registro de un proyecto determinado de forma precisa y detallada, su uso es más efectivo en el ámbito académico, para el aprendizaje de los conceptos y el procedimiento de la metodología SCRUM en el desarrollo de un proyecto de software.

6.2 Trabajo futuro

Al analizar varios métodos de estimación de esfuerzo y al ver que se pueden combinar algunos para mejorar las estimaciones (por ejemplo Puntos de Función – COCOMO II), un tema de investigación futuro es encontrar una combinación de métodos que mejore las estimaciones del esfuerzo de desarrollo en los micro-proyectos.

Dado que las constantes utilizadas en los métodos de Puntos de Función, COCOMO II y Puntos de Casos de Uso fueron determinadas mediante un estudio aplicado a proyectos muy grandes, y estos métodos resultaron ser los más alejados de los valores de esfuerzo reales en los micro-proyectos. Se propone realizar un estudio que repita el procedimiento para encontrar estas constantes pero aplicado a una muestra significativa de micro-proyectos.

En los micro-proyectos analizados se aplicaron diferentes técnicas de recopilación y especificación de los requerimientos. No está determinado si las técnicas de obtención y especificación de los requerimientos ayudan a mejorar la estimación del esfuerzo de desarrollo, por lo que se propone realizar una investigación sobre las técnicas más efectivas de obtención de requerimientos para micro-proyectos.

Los micro-proyectos de software deben ser entregados en plazos muy cortos de tiempo (ver Tabla 15), por lo que se requiere investigar que herramientas, técnicas y métodos de diseño son los más adecuados para determinar la arquitectura de los mismos.

La herramienta desarrollada contiene funcionalidad básica de cómo funciona el marco de trabajo SCRUM, por lo tanto permite que esta se pueda mejorar. Entre las mejoras se encuentran:

1. Migrar la aplicación a un entorno móvil, para que sea mucho más portátil y el principio de agilidad sea más evidente.
2. Aumentar su funcionalidad agregando módulos que permitan llevar el seguimiento del desarrollo de un proyecto de manera más detallada.

Anexo A

Puntos de Función

Desarrollados por Allan Albrecht, estos miden la funcionalidad de lo que será entregado a un usuario final, con base en características funcionales bien definidas de un sistema de software. Éstas características son: *entradas externas, conformadas por* información que es entregada al sistema (cajas de texto, botones, transacciones lógicas); *salidas externas, aquella* información que es procesada por el sistema y liberada (reportes); *consultas externas, la* información mostrada que no fue procesada (reportes); *archivos lógicos internos, integrados por* información que es procesada y almacenada en el sistema (información de un grupo de usuarios) y *archivo de interface externo,* información que es mantenida fuera del sistema, pero que es necesaria para satisfacer un requerimiento en particular de un proceso (interfaces con otros sistemas) (Garmus & Herron, 2001). Ha dichas características se les asigna una puntuación ya definida de un conjunto de tablas (Parthasarathy, 2007). Al realizar las operaciones determinadas por la metodología obtenemos la estimación de costo.

Procedimiento

1. Cálculo de los puntos de función sin ajustar (UFC, Unadjusted Function point Count).
   1.1. Tener un prototipo de los requerimientos funcionales (pantallas de formularios, reportes, etc. y archivos o tablas de base de datos).
   1.2. Cada requerimiento funcional se debe asociar con uno de los 5 componentes funcionales IFPUG (EI, EO, EQ, ILF, EIF).
   1.3. Para cada componente funcional se le determina su complejidad (low, average o high) con su correspondiente valor en puntos de función, ver Tabla 24, Tabla 25, Tabla 26 y Tabla 27.
   1.4. Se suman todos los puntos de función, dando como resultado los puntos de función sin ajustar.
2. Cálculo del factor técnico de complejidad (TCF, Technical Complexity Factor)
   2.1. TCF es una suma de pesos de 14 componentes, ver de la Tabla 28 a la Tabla 41.
   2.2. Cada componente tiene un rango de 0-5
   2.3. El TCF puede ser calculado como: $TCF = 0.65 + 0.01 \sum_{j=1}^{14} F_j$
      2.3.1. $\sum_{j=1}^{14} F_j$ es la suma de los pesos de los 14 componentes.
   2.4. EL TCF varía de 0.65 a 1.35.
3. Resultado de los puntos de función ajustados FPA=UFC*TCF

Determinar la complejidad de un componente funcional

Data Element Type (DET): son campos que no se repiten, reconocibles por el usuario, contenidos en el ILF. Los campos físicamente almacenados en campos múltiples (como números de cuenta, fechas, hora) deberían ser contados como un campo cuando el usuario se refiere a ellos como un elemento.
File Type Referenced (FTR): tiene que ser un archivo lógico (ILF) o un archivo externo (EIF).

External Inputs (EI): Los datos entran en los límites de la aplicación, pueden venir de una pantalla de entrada o de otra aplicación. Los datos son usados para mantener uno o más archivos lógico internos (ILF's). Los datos pueden ser orientados al negocio o información de control. Si los datos son información de control, no se tiene que actualizar un ILF.

External Outputs (EO): Son datos derivados que salen de los límites de la aplicación, tales datos crean reportes o archivos de salida que pueden ser enviados a otras aplicaciones. Estos reportes y archivos son creados de uno o más ILF's o archivos de interface externos (EIF's).
Los datos derivados son datos que se procesan más allá de la edición directa, son usualmente el resultado de algoritmos o cálculos.
Una EO podría mantener uno o más ILF's y/o alterar el comportamiento del sistema.

External Inquiry (EQ): Son todas las combinaciones de entradas/salidas que resultan de la adquisición de datos o información de control de uno o más ILF's o EIF's. Ningún ILF es mantenido durante una EQ, ni tampoco el comportamiento del sistema se ve alterado. El proceso de salida no contiene datos derivados.

Internal Logical File (ILF): Un ILF es un grupo de datos definidos por el usuario que están relacionados lógicamente, residen en su totalidad dentro de los límites de la aplicación y son actualizados a través de entradas externas.

External Interface File (EIF): Un EIF es un grupo de datos definidos por el usuario que están relacionados lógicamente, sólo son usados para propósitos de referencia. Los datos residen enteramente fuera de la aplicación además son mantenidos por otra aplicación. Este archivo(s) es un ILF para otra aplicación.

Tabla 24. Matriz de Complejidad de Entradas de Usuario

Matriz EI			
# EIFs / ILFs	DETs		
	<5	5-14	>15
<2	Low	Low	Average
2	Low	Average	High
>2	Average	High	High

Tabla 25. Matriz de Complejidad de Salidas de Usuario

Matriz EO			
# EIFs / ILFs	DETs		
	<6	6-19	>19
<2	Low	Low	Average
2-3	Low	Average	High
>3	Average	High	High

Tabla 26. Matriz de Complejidad de ILF y EIF

Matriz ILF y EIF			
# RETs	DETs		
	<20	20-50	>50
<2	Low	Low	Average
2-5	Low	Average	High
>5	Average	High	High

Al determinar la complejidad de los ILF/EIF, el conteo de los RETs y DETs puede ser confuso al momento de identificarlos. Para explicar cómo identificarlos, supondremos que la aplicación a analizar utiliza una base de datos con dos entidades y una tabla, la tabla será considerada como ILF, las dos entidades serán cada una un RET y los campos de la tabla los DETs, ver Figura 23. El conteo se realizará como sigue, se tiene 1 ILF con 2 RETs y 5 DETs, si nos remitimos a la Tabla 26 la complejidad correspondiente a este ILF es "Low" y por lo consiguiente se le asignarán 7 puntos de función, como indica la Tabla 27. y esa sería la manera de contar los ILF/EIF.

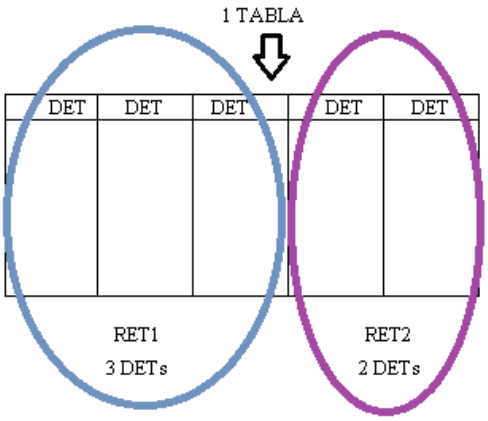


Figura 23. Conteo de RETs y DETs dentro de un ILF/EIF

Hoy en día debido a la normalización de las bases de datos, cada tabla contiene los campos de una entidad única (1 RET), difícilmente en una tabla se tienen campos (DETs) que pertenezcan a más de una entidad, por lo anterior es que se puede generalizar que una tabla se identifica con una complejidad baja o "low", por lo cual se le asignan 7 puntos de función. Para calcular el total de puntos de función de los ILF/EIF solo basta con multiplicar el total de tablas utilizadas por 7.

Tabla 27. Valores de Puntos de Función

Tipo de función	Low	Average	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Factores técnicos de complejidad

Los factores de complejidad son 14, estos ayudarán a determinar el grado de influencia que tiene el medio en el que se desarrollará el micro-proyecto. Para calificar el nivel de influencia en la aplicación se tomarán los siguientes parámetros desde la Tabla 28 a la Tabla 41. Cada uno de los componentes tiene un rango de 0-5:

- 0 = No influencia
- 1 = Incidental (poco, accidental)
- 2 = Moderado
- 3 = Medio
- 4 = Significativo
- 5 = Esencial

1- Comunicación de datos

La información y datos de control utilizados por la aplicación son enviados o recibidos a través de recursos de comunicación de datos, algunos ejemplos de esto son las terminales y estaciones de trabajo. . Todos los dispositivos de comunicación utilizan algún tipo de protocolo de comunicación.

Tabla 28. Comunicación de datos

GRADO	DESCRIPCIÓN
0	Aplicación puramente batch o funciona en una computadora aislada.
1	La aplicación es batch, pero utiliza entrada de datos remota o impresión remota
2	La aplicación es batch, pero utiliza entrada de datos remota e impresión remota.
3	La aplicación incluye entrada de datos on-line vía entrada de video o un procesador front-end para alimentar procesos batch o sistemas de consultas.
4	La aplicación es más que una entrada on-line, y soporta apenas un protocolo de comunicación.
5	La aplicación es más que una entrada on-line y soporta más de un protocolo de comunicación.

2- Procesamiento de datos distribuidos

Los datos y/o procesamientos distribuidos entre varias unidades de procesamiento (CPUs) son características generales que pueden influenciar en la complejidad de la aplicación.
En la Tabla 29 se explica cómo son manejados los datos distribuidos y las funciones de procesamiento.

Tabla 29. Procesamiento de datos distribuidos

GRADO	DESCRIPCIÓN
0	La aplicación no contribuye en la transferencia de datos o funciones entre los procesadores de la empresa.
1	La aplicación prepara datos para el usuario final en otra CPU de la empresa.
2	La aplicación prepara datos para transferencia, los transfiere y entonces son procesados en otro equipamiento de la empresa (no por el usuario final).
3	Procesamiento distribuido y la transferencia de datos son on-line, en apenas una dirección.
4	Procesamiento distribuido y la transferencia de datos son on-line, en ambas direcciones.
5	Las funciones de procesamiento son dinámicamente ejecutadas en el equipamiento más adecuado.

3- Performance

Los objetivos del perfomance del sistema, establecidos y aprobados por el usuario en términos de respuesta, influyen o podrían influir en el proyecto, desarrollo, implementación o soporte de la aplicación. ¿El tiempo de respuesta o el nivel de eficiencia es requerido por el usuario?

Tabla 30. Performance

GRADO	DESCRIPCIÓN
0	Ningún requerimiento especial de perfomance fue solicitado por el usuario.
1	Requerimientos de perfomance y de diseño fueron establecidos y previstos, sin embargo ninguna acción especial fue requerida.
2	El tiempo de respuesta y el volumen de datos son críticos durante horarios pico de procesamiento. Ninguna determinación especial para la utilización del procesador fue establecida. El intervalo de tiempo límite para la disponibilidad de procesamiento es siempre el próximo día hábil.
3	El tiempo de respuesta y volumen de procesamiento son items críticos durante todo el horario comercial. Ninguna determinación especial para la utilización del procesador fue establecida. El tiempo límite necesario para la comunicación con otros sistemas es un aspecto importante.
4	Los requerimientos de perfomance establecidos necesitan tareas de análisis de perfomance en la fase de análisis y diseño de la aplicación.
5	Además de lo descrito en el ítem anterior, herramientas de análisis de perfomance fueron usadas en las fases de diseño, desarrollo y/o implementación para atender los requerimientos de perfomance establecidos por el usuario.

4- Configuración del equipamiento

Esta característica representa la necesidad de realizar consideraciones especiales en el diseño de los sistemas para que la configuración del equipamiento no sea sobrecargada.

Tabla 31. Configuración del equipamiento

GRADO	DESCRIPCIÓN
0	Ninguna restricción operacional explícita o implícita fue incluida.
1	Existen restricciones operacionales leves. No es necesario un esfuerzo especial para resolver estas restricciones.
2	Algunas consideraciones de ajuste de perfomance y seguridad son necesarias.
3	Son necesarias especificaciones especiales de procesador para un módulo específico de la aplicación.
4	Restricciones operacionales requieren cuidados especiales en el procesador central o procesador dedicado.
5	Además de las características del ítem anterior, hay consideraciones especiales en la distribución del sistema y sus componentes.

5- Volumen de transacciones

El nivel de transacciones es alto y tienen influencia en el diseño, desarrollo, implementación y mantenimiento de la aplicación. ¿Qué tan frecuentemente se ejecutan las transacciones al día, semana, mes, etc.?

Tabla 32. Volumen de transacciones

GRADO	DESCRIPCIÓN
0	No están previstos periodos picos de volumen de transacción.
1	Están previstos picos de transacciones mensualmente, trimestralmente, anualmente o en un cierto periodo del año.
2	Se prevén picos semanales.
3	Se prevén picos diariamente.
4	Alto nivel de transacciones fue establecido por el usuario, el tiempo de respuesta necesario exige un nivel alto o suficiente para requerir análisis de perfomance y diseño.
5	Además de lo descrito en el ítem anterior, es necesario utilizar herramientas de análisis de perfomance en las fases de diseño, desarrollo y/o implementación.

6- Entrada de datos on-line

Esta característica cuantifica la entrada de datos on-line proveída por la aplicación. ¿Qué porcentaje de información se captura En Línea?

Tabla 33. Entrada de datos on-line

GRADO	DESCRIPCIÓN
0	Todas las transacciones son procesadas en modo batch.
1	De 1% al 7% de las transacciones son entradas de datos on-line.
2	De 8% al 15% de las transacciones son entradas de datos on-line.
3	De 16% al 23% de las transacciones son entradas de datos on-line.
4	De 24% al 30% de las transacciones son entradas de datos on-line.
5	Más del 30% de las transacciones son entradas de datos on-line.

7- Interface con el usuario

Las funciones on-line del sistema hacen énfasis en la amigabilidad del sistema y su facilidad de uso, buscando aumentar la eficiencia para el usuario final. El sistema posee:

- Ayuda para la navegación (teclas de función, accesos directos y menús dinámicos).
- Menús.
- Documentación y ayuda on-line.
- Movimiento automático del cursor.
- Scrolling vertical y horizontal.
- Impresión remota (a través de transacciones on-line).
- Teclas de función preestablecidas.
- Ejecución de procesos batch a partir de transacciones on-line.
- Selección de datos vía movimiento del cursor en la pantalla.
- Utilización intensa de campos en video reverso, intensificados, subrayados, coloridos y otros indicadores.
- Impresión de la documentación de las transacciones on-line por medio de hard copy.
- Utilización del mouse.
- Menús pop-up.
- El menor número de pantallas posibles para ejecutar las funciones del negocio.
- Soporte bilingüe (el soporte de dos idiomas, cuente como cuatro items).
- Soporte multilingüe (el soporte de más de dos idiomas, cuente como seis items).

Tabla 34. Interface con el usuario

GRADO	DESCRIPCIÓN
0	Ningún de los items descritos.
1	De uno a tres de los items descritos.
2	De cuatro a cinco de los items descritos.
3	Más de cinco de los items descritos, no hay requerimientos específicos del usuario en cuanto a amigabilidad del sistema.
4	Más de cinco de los items descritos, y fueron descritos requerimientos en cuanto a amigabilidad del sistema suficientes para generar actividades específicas incluyendo factores tales como minimización de la digitación.
5	Más de cinco de los items descritos y fueron establecidos requerimientos en cuanto a la amigabilidad suficientes para utilizar herramientas especiales y procesos especiales para demostrar anticipadamente que los objetivos fueron alcanzados.

8- Actualización on-line

La aplicación posibilita la actualización on-line de los archivos lógicos internos.

Tabla 35. Actualización on-line

GRADO	DESCRIPCIÓN
0	Ninguna.
1	Actualización on-line de uno a tres archivos lógicos internos.
2	Actualización on-line de más de tres archivos lógicos internos.
3	Actualización on-line de la mayoría de los archivos lógicos internos.
4	Además del ítem anterior, la protección contra pérdidas de datos es esencial y fue específicamente proyectado y codificado en el sistema.
5	Además del ítem anterior, altos volúmenes influyen en la las consideraciones de costo en el proceso de recuperación. Procesos para automatizar la recuperación fueron incluidos minimizando la intervención del operador.

9- Procesamiento complejo

¿La aplicación tiene mucho procesamiento lógico o matemático? El procesamiento complejo es una de las características de la aplicación, los siguientes componentes están presentes:

- Procesamiento especial de auditoria y/o procesamiento especial de seguridad.
- Procesamiento lógico extensivo.
- Procesamiento matemático extensivo.
- Gran cantidad de procesamiento de excepciones, resultando en transacciones incompletas que deber ser procesadas nuevamente. Por ejemplo, transacciones de datos incompletos interrumpidoas por problemas de comunicación o con datos incompletos.
- Procesamiento complejo para manipular múltiples posibilidades de entrada/salida. Ejemplo: multimedia.

Tabla 36. Procesamiento complejo

GRADO	DESCRIPCIÓN
0	Ninguno de los items descritos.
1	Apenas uno de los items descritos.
2	Dos de los items descritos.
3	Tres de los items descritos.
4	Cuatro de los items descritos.
5	Todos los items descritos.

10- Reusabilidad

¿La aplicación se desarrollará para cumplir una o muchas necesidades del usuario? La aplicación y su código serán proyectados, desarrollados y mantenidos para ser utilizados en otras aplicaciones.

Tabla 37. Reusabilidad

GRADO	DESCRIPCIÓN
0	No presenta código reutilizable.
1	Código reutilizado fue usado solamente dentro de la aplicación.
2	Menos del 10% de la aplicación fue proyectada previendo la utilización posterior del código por otra aplicación.
3	10% o más de la aplicación fue proyectada previendo la utilización posterior del código por otra aplicación.
4	La aplicación fue específicamente proyectada y/o documentada para tener su código fácilmente reutilizable por otra aplicación y la aplicación es configurada por el usuario a nivel de código fuente.
5	La aplicación fue específicamente proyectada y/o documentada para tener su código fácilmente reutilizable por otra aplicación y la aplicación es configurada para uso a través de parámetros que pueden ser alterados por el usuario.

11- Facilidad de implementación

La facilidad de implementación y conversión de datos son características de la aplicación. Un plan de conversión e implementación y/o herramientas de conversión fueron proveídas y probadas durante la fase de prueba de la aplicación.

Tabla 38. Facilidad de implementación

GRADO	DESCRIPCIÓN
0	Ninguna consideración especial fue establecida por el usuario y ningún procedimiento especial fue necesario en la implementación.
1	Ninguna consideración especial fue establecida por el usuario, más procedimientos especiales son requeridos en la implementación.
2	Requerimientos de conversión e implementación fueron establecidos por el usuario y rutinas de conversión e implementación fueron proporcionados y probados. El impacto de conversión en el proyecto no es considerado importante.
3	Requerimientos de conversión e implementación fueron establecidos por el usuario y rutinas de conversión e implementación fueron proporcionados y probados. El impacto de conversión en el proyecto es considerado importante.
4	Además del ítem 2, conversión automática y herramientas de implementación fueron proporcionadas y probadas.
5	Además del ítem 3, conversión automática y herramientas de implementación fueron proveídas.

12- Facilidad de operación

La facilidad de operación es una característica del sistema. Por ello los procedimientos de inicialización, respaldo y recuperación fueron proveídos y probados durante la fase de prueba del sistema. La aplicación minimiza la necesidad de actividades manuales, tales como montaje de cintas magnéticas, manoseo de papel e intervención del operador. ¿Qué tan efectivos y/o automatizados son los procedimientos de inicio, respaldo y recuperación?

Tabla 39. Facilidad de operación

GRADO	DESCRIPCIÓN
0	Ninguna consideración especial de operación, además del proceso normal de respaldo establecido por el usuario.
1 - 4	Verificar cuáles de las siguientes afirmaciones pueden ser identificadas en la aplicación. Cada ítem vale un punto, excepto se defina lo contrario: • Fueron desarrollados procedimientos de inicialización y respaldo, siendo necesaria la intervención del operador. • Se establecieron procesos de inicialización, respaldo y recuperación sin ninguna intervención del operador (contar como 2 items). • La aplicación minimiza la necesidad de montaje de cintas magnéticas. • La aplicación minimiza la necesidad de manoseo de papel.
5	La aplicación fue diseñada para trabajar sin operador, ninguna intervención del operador es necesaria para operar el sistema, excepto ejecutar y cerrar la aplicación. La aplicación posee rutinas automáticas de recuperación en caso de error.

13- Múltiples sitios

La aplicación fue específicamente proyectada, diseñada y mantenida para ser instalada en múltiples locales de una organización o para múltiples organizaciones.

Tabla 40. Múltiples sitios

GRADO	DESCRIPCIÓN
0	Los requerimientos del usuario no consideran la necesidad de instalación de más de un local.
1	La necesidad de múltiples locales fue considerada en el proyecto y la aplicación fue diseñada para operar apenas sobre el mismo ambiente de hardware y software.
2	La necesidad de múltiples locales fue considerada en el proyecto y la aplicación fue diseñada para operar en ambientes similares de software y hardware.
3	La necesidad de múltiples locales fue considerada en el proyecto y la aplicación está separada para trabajar sobre diferentes ambientes de hardware y/o software.
4	Los Planes de mantenimiento y documentación fueron proporcionados y probados para soportar la aplicación en múltiples locales, además los items 1 y 2 caracterizan a la aplicación.
5	Los Planes de documentación y mantenimiento fueron proveídos y probados para soportar la aplicación en múltiples locales, además el ítem 3 caracteriza a la aplicación.

14- Facilidad de cambios

La aplicación fue específicamente proyectada y diseñada con vistas a facilitar su mantenimiento. Las siguientes características pueden ser atribuidas a la aplicación:

- Están disponibles facilidades como consultas e informes flexibles para atender necesidades simples (contar 1 ítem).
- Están disponibles facilidades como consultas e informes flexibles para atender necesidades de complejidad media (contar 2 items).
- Están disponibles facilidades como consultas e informes flexibles para atender necesidades complejas (contar 3 items).
- Los datos de control se almacenan en tablas que son mantenidas por el usuario a través de procesos on-line, pero los cambios se hacen efectivos solamente al día siguiente.
- Los datos de control se almacenan en tablas que son mantenidas por el usuario a través de procesos on-line, pero los cambios se hacen efectivos inmediatamente (contar 2 items).

Tabla 41. Facilidad de cambios

GRADO	DESCRIPCIÓN
0	Ninguno de los items descritos.
1	Apenas uno de los items descritos.
2	Dos de los items descritos.
3	Tres de los items descritos.
4	Cuatro de los items descritos.
5	Todos los items descritos.

Una vez evaluados los 14 factores, el valor del factor técnico de complejidad se calcula con la siguiente formula

$$TCF = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

F_i- es el grado de cada factor

Puntos de Función a Líneas de Código Fuente

Los puntos de función ajustados son el resultado del análisis con el Método de Puntos de Función, para algunos desarrolladores este resultado es suficiente para realizar sus estimaciones de tiempo y esfuerzo con base en su experiencia. Algunos otros desarrolladores utilizan la combinación de este y otro método, como son COCOMO o SEER-SEM para completar la estimación, siendo los puntos de función un parámetro de tamaño perteneciente a la ecuación que utiliza cada método. En el caso de COCOMO el tamaño tiene que estar en KDSI (miles de instrucciones fuente liberadas) o Líneas de Código Fuente. Los Puntos de Función pueden ser convertidos a Líneas

de Código mediante un factor de conversión. Existen organizaciones dedicadas a determinar estos factores, proporcionando tablas de factores de conversión que dependen de la tecnología y leguaje con el que se vaya a codificar el proyecto estimado con Puntos de Función, cabe mencionar que estos factores son calculados en base a grandes cantidades de datos históricos. Una de estas organizaciones es la QSM (Inc., 2013), dedicada a ofrecer servicios de estimación con análisis de puntos de función, cuenta con una base de datos históricos de un poco más de 10,000 proyectos completados, desarrollados aproximadamente en 126 lenguajes diferentes de programación. En la Tabla 42 (obtenida por la QSM) los factores de conversión promedio están dados en Líneas de Código Fuente sobre Puntos de Función [SLOC/FP], para 37 lenguajes diferentes de programación.

Tabla 42. Factores de Conversión Promedio para 37 lenguajes de programación

LENGUAJE	SLOC/FP PROMEDIO
ABAP(SAP)	28
ASP	51
Assembler	119
Brio	14
C	97
C++	50
C#	54
COBOL	61
Cognos Impromptu Scripts	47
Cross System Products (CSP)	20
Cool:Gen/IEF	32
Datastage	71
Excel	209
Focus	43
FoxPro	36
HTML	34
J2EE	46
Java	53
JavaScript	47
JCL	62
LINC II	29
Lotus Notes	23
Natural	40
.NET	57
Oracle	37
Perl	24
PL/1	64
PL/SQL	37
Powerbuilder	26
REXX	77
Sabretalk	70
SAS	38
Siebel	59
SLOGAN	75
SQL	21
VB.NET	52
Visual Basic	42

Para realizar la conversión se selecciona el lenguaje en el que se codificará el proyecto, para conocer el factor de conversión adecuado, ver Tabla 42, una vez seleccionado multiplicar el valor de puntos de función ajustados que se calculó por el factor de conversión. El resultado quedara en SLOCs.

Si el lenguaje de programación a utilizar no está en la tabla, se recomienda seleccionar el que más se parezca, en cuanto a características, al que se pretendía utilizar.

Anexo B

COCOMO

Constructive COst MOdel estima el esfuerzo (meses-persona) y duración en meses, necesarios para desarrollar un sistema de software utilizando una ecuación. Este método subdivide a los sistemas en tres categorías, las cuales son los orgánicos, embebidos y semi-desconectados.

COCOMO: Esfuerzo

COCOMO se divide en tres modelos: básico, intermedio y detallado.

Modelo Básico: puede ser aplicado cuando se conoce poco del proyecto.

Modelo Intermedio: este es aplicable después de que los requerimientos son especificados. (El modelo COCOMO, 2013) (Fenton & Pfleeger, 1998)

Modelo Detallado: es aplicable cuando el diseño está completo.

Los tres utilizan la misma fórmula: $E = aS^bF$

E- esfuerzo en –meses-persona
S- tamaño medido en KDSI
F- factor de ajuste (tabla de manejadores de costos)
a y *b*- dependen del tipo de sistema, Anexo B (tablas de valores)

COCOMO también hace distinción entre el modo de desarrollo de sistemas.

Sistema Orgánico: envuelve procesamiento de datos, tiende a usar bases de datos, se enfoca en transacciones y recuperación de datos.

Sistema Empotrado: contiene software en tiempo real que es parte integral de un sistema basado en hardware más grande.

Sistema semi-empotrado: es algo entre orgánico y empotrado.

Proceso:

1. Identificar el sistema a desarrollar.
2. Identificar el modelo COCOMO a utilizar.
3. Seleccionar los valores de *a* y *b* de la formula, según el tipo de sistema y modelo COCOMO, ver Tabla 44 o Tabla 45.
4. Calcular el factor de ajuste.
   4.1. F=1 para el modelo básico.
   4.2. Para calcular F se tienen que evaluar 15 manejadores de costo, ver
   4.3. Tabla 46 y Tabla 47.
5. Sustituir todos los valores en la formula $E = aS^bF$, como resultado se obtiene el esfuerzo en meses-persona necesario para desarrollar el proyecto.

COCOMO: Duración

Para calcular la duración de un proyecto se utiliza la siguiente fórmula (El modelo COCOMO, 2013) (Fenton & Pfleeger, 1998): $D = aE^b$

E- esfuerzo en meses-persona
a y *b*- dependen del tipo de sistema, (tabla de valores)

Proceso:

1. Calcular previamente el esfuerzo.
2. Seleccionar los valores de *a* y *b* de la formula, según el tipo de sistema.
3. Sustituir todos los valores en la formula $D = aE^b$, ver Tabla 43, como resultado tenemos el tiempo en meses necesario para desarrollar el proyecto.

Tabla 43. Parámetros de duración para los tres modos de COCOMO

MODO	a	B
Orgánico	2.50	0.38
Semi-empotrado	2.50	0.35
Empotrado	2.50	0.32

I. COCOMO Básico

Tabla 44. Parámetros de esfuerzo para los tres modos de desarrollo

MODO	a	B
Orgánico	2.40	1.05
Semi-empotrado	3.00	1.12
Empotrado	3.60	1.20

II.COCOMO Intermedio

Tabla 45. Parámetros de esfuerzo para los tres modos de desarrollo

MODO	a	B
Orgánico	3.20	1.05
Semi-empotrado	3.00	1.12
Empotrado	2.80	1.20

III. COCOMO Detallado

Presenta principalmente dos mejoras respecto al anterior.

Este modelo puede procesar todas las características del proyecto para construir una estimación. Introduce dos características principales (El modelo COCOMO, 2013):

(1) Multiplicadores de esfuerzo sensitivos a la fase. Algunas fases se ven más afectadas que otras por los atributos. El modelo detallado proporciona un conjunto de multiplicadores de esfuerzo para cada atributo. E Para ayudar a determinar la asignación del personal para cada fase del proyecto.

(2) Jerarquía del producto a tres niveles. Se definen tres niveles de producto, como son el módulo, subsistema y sistema. La cuantificación se realiza al nivel apropiado, e es decir el nivel al que es más susceptible la variación.

3.1 Estimación del esfuerzo.

A. Fases de desarrollo

El desarrollo del software se lleva a cabo a través de cuatro fases consecutivas: requerimientos/planes, diseño del producto, programación y prueba/integración.

Requerimientos/planes. Es la primera fase del ciclo de desarrollo. Se analiza el requerimiento, se muestra un Plan de Producto y se genera una especificación completa del producto. La fase consume del 6% al 8% del esfuerzo sin el multiplicador F, y dura del 10% al 40% del tiempo de desarrollo. Estos porcentajes dependen del modo y del tamaño (de 2000 LOC a 512000 LOC).

Diseño del producto. La segunda fase del ciclo de desarrollo COCOMO se preocupa de la determinación de la arquitectura del producto y de las especificaciones de los subsistemas. Esta fase requiere del 16% al 18% del esfuerzo sin el multiplicador F, y puede durar del 19% al 38% del tiempo de desarrollo.

Programación. En la tercera fase del ciclo de desarrollo COCOMO se subdivide en dos subfases: diseño detallado y prueba del código. Esta fase requiere del 48% al 68% del esfuerzo sin el multiplicador F, y dura del 24% al 64% del tiempo de desarrollo.

Prueba/Integración. Esta última fase consiste principalmente en unir las diferentes unidades ya probadas. Se utiliza del 16% al 34% del coste nominal y dura del 18% al 34% del tiempo.

B. Principio de estimación del esfuerzo.

B.1. Tamaño equivalente. Como parte del software puede haber sido ya desarrollado, no se requiere entonces un desarrollo completo. En tales casos se estiman las partes de diseño (D%), código (C%) e integración (I%) a ser modificadas. Se calcula un factor de ajuste A.

$$A = 0.4\,D + 0.3\,C + 0.3\,I$$

El tamaño equivalente, Sequ es

$$Sequ = (S \cdot A) / 100.$$

B.2. Cálculo del esfuerzo. El tamaño equivalente se calcula para cada módulo. El esfuerzo asignado al desarrollo de cada módulo se obtiene entonces a través de:

(1) seleccionar los valores apropiados de los atributos de coste para cada fase.

(2) multiplicar los atributos de coste para cada módulo y fase, obteniendo un conjunto de 4 multiplicadores globales.

(3) multiplicar los atributos globales por el esfuerzo nominal en cada fase y sumarlos para obtener el esfuerzo total estimado.

Tabla 46. Manejadores de Costos

ATRIBUTO	TIPO	DESCRIPCIÓN
RELY	Producto	Matriz de Complejidad de Entradas de Usuario.
CPLX	Producto	Representa la complejidad del producto.
DATA	Producto	Tamaño de la base de datos en relación con el tamaño del programa. El valor del modificador se define por la relación: D/K, donde D corresponde al tamaño de la base de datos en bytes y K es el tamaño del programa en cantidad de líneas de código.
TIME	Computadora	Restricción del tiempo de ejecución.
VIRT	Computadora	Volatilidad de la máquina virtual
STOR	Computadora	Restricción del almacenamiento principal.
TURN	Computadora	Tiempo de respuesta del ordenador.
ACAP	Personal	Capacidad del analista del proyecto.
PCAP	Personal	Capacidad del programador.
LEXP	Personal	Experiencia en el lenguaje de programación a usar.
AEXP	Personal	Experiencia del personal en aplicaciones similares.
VEXP	Personal	Experiencia del personal en la máquina virtual.
TOOL	Proyecto	Uso de herramientas de software.
SCED	Proyecto	Compresión del calendario de desarrollo.
MODP	Proyecto	Prácticas de programación modernas.

Cada atributo se cuantifica para un entorno de proyecto. La escala es:

- muy bajo
- bajo
- normal
- alto
- muy alto
- extremadamente alto

En la Tabla 47 se muestran los valores del multiplicador para cada uno de los 15 atributos (Boehm B. W., 1983). Estos 15 valores se multiplican para obtener el facto F y al sustituirlo en la formula proporciona el esfuerzo ajustado al entorno.

Tabla 47. Valores del multiplicador para cada uno de los 15 manejadores de costo

Atributos	Valor					
	Muy bajo	**Bajo**	**Nominal**	**Alto**	**Muy alto**	**Extra alto**
Atributos de software						
Fiabilidad (RELY)	0,75	0,88	1,00	1,15	1,40	
Tamaño de Base de datos (DATA)		0,94	1,00	1,08	1,16	
Complejidad (CPLX)	0,70	0,85	1,00	1,15	1,30	1,65
Atributos de hardware						
Restricciones de tiempo de ejecución (TIME)			1,00	1,11	1,30	1,66
Restricciones de memoria de almacenamiento (STOR)			1,00	1,06	1,21	1,56
Volatilidad de la máquina virtual (VIRT)		0,87	1,00	1,15	1,30	
Tiempo de respuesta (TURN)		0,87	1,00	1,07	1,15	
Atributos de personal						
Capacidad de análisis (ACAP)	1,46	1,19	1,00	0,86	0,71	
Experiencia en la aplicación (AEXP)	1,29	1,13	1,00	0,91	0,82	
Calidad de los programadores (PCAP)	1,42	1,17	1,00	0,86	0,70	
Experiencia del personal en la máquina virtual (VEXP)	1,21	1,10	1,00	0,90		
Experiencia en el lenguaje (LEXP)	1,14	1,07	1,00	0,95		
Atributos del proyecto						
Técnicas actualizadas de programación (MODP)	1,24	1,10	1,00	0,91	0,82	
Utilización de herramientas de software (TOOL)	1,24	1,10	1,00	0,91	0,83	
Restricciones de tiempo de desarrollo (SCED)	1,22	1,08	1,00	1,04	1,10	

Anexo C

COCOMO II

COCOMO 81 y COCOMO II (actualización de COCOMO 81) son modelos algorítmicos de costos, lo que significa que utilizan fórmulas matemáticas para predecir el esfuerzo de desarrollo de un proyecto, estiman el tamaño del mismo, el número de ingenieros de software, también otros factores de productividad y de proceso, este modelo fue propuesto por Barry Boehm en 1997 y en el año 2000 fue reajustado (Faireyl, 2009).

El modelo COCOMO II reconoce diferentes enfoques para el desarrollo de software, como es el prototipado, desarrollo por composición de componentes y uso de programación de bases de datos. COCOMO II soporta un modelo de desarrollo en espiral y varios sub-módulos embebidos que producen estimaciones cada vez más detalladas.

Los cuatro sub-módulos que son parte del modelo COCOMO II son (Sommerville, 2007), algunos autores consideran tres (Pressman, Software engineering: a practitioner´s approach, 2001) (Ismaeel & Jamil, 2007):

1. *Modelo aplicación-composición.* Asume que los sistemas son creados como componentes reusables, scripting o programación de bases de datos. Este sub-modelo es diseñado para hacer estimaciones de prototipos desarrollados. El tamaño del software estimado está basado en puntos de aplicación (puntos de objeto), y una simple fórmula tamaño/producción es usada para estimar el esfuerzo requerido.
2. *Modelo de diseño temprano.* Este modelo es usado durante etapas tempranas del diseño del sistema después de haber establecido los requerimientos. La estimación está basada en puntos de función, los cuales, después son convertidos a líneas de código fuente. El modelo sigue la fórmula estándar mostrada a continuación con un conjunto simplificado de siete multiplicadores de esfuerzo.
3. *Modelo de reúso.* El modelo es usado para calcular el esfuerzo requerido para integrar componentes reusables y/o código de programa que es automáticamente generado por herramientas de diseño o traducción de programas. Usualmente es usado en conjunción con el modelo de post arquitectura.
4. *Modelo post arquitectura.* Una vez que la arquitectura del sistema ha sido diseñada, una estimación más precisa del tamaño del software puede ser hecha, nuevamente este modelo usa la fórmula estándar para estimar el costo. Sin embargo, incluye un conjunto más extenso de 17 multiplicadores que refleja las capacidades personales, características del producto y del proyecto.

En su forma más general, un algoritmo de estimación de costos puede ser expresado como:

$$Esfuerzo = A \times Tamaño^{B} \times M$$

Y el tiempo de desarrollo como:

$$Tiempo = C * Esfuerzo^{D}$$

Donde A y C son constantes, B y D son exponentes calculados al evaluar una serie de *Factores de Escala* que influyen en el cálculo del esfuerzo de desarrollo y M es un factor resultante del producto de factores técnicos, *Multiplicadores de Esfuerzo,* que también influyen en el esfuerzo de desarrollo.

En el sub-modelo COCOMO II de Diseño Temprano, los parámetros de las ecuaciones de esfuerzo y tiempo de desarrollo son:

A=2.94

C=3.67

$M = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCED$, ver Tabla 48.

Exponente de esfuerzo:$B = 0.91 + 0.01 * \sum_{j=1}^{5} SF_j$

Exponente de calendarización:$D = 0.28 + 0.2 * 0.01 * \sum_{j=1}^{5} SF_j$

SF- factores de escala, ver Tabla 49.

Los Factores de Ajuste del Esfuerzo o Multiplicadores de Esfuerzo usados en el modelo de diseño temprano son:

- Fiabilidad y complejidad (RCPX)
- Reusó requerido (RUSE)
- Dificultades de plataforma (PDIF)
- Capacidad personal (PERS)
- Experiencia personal (PREX)
- Facilidad de soporte (FCIL)
- Calendario (SCED).

Los factores de escala son:

- Precedentes (PREC)
- Flexibilidad del proyecto (FLEX)
- Arquitectura/resolución de riesgos (RESL)
- Cohesión del equipo (TEAM)
- Madurez del proceso (PMAT)

Tabla 48. Multiplicadores de Esfuerzo del sub-modelo de Diseño Temprano

	Extra bajo	Muy bajo	Bajo	Normal	Alto	Muy alto	Extra alto
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE			0.95	1.00	1.07	1.15	1.24
PDIF			0.87	1.00	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	1.59	1.33	1.22	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED		1.43	1.14	1.00	1.00	1.00	

Tabla 49. Factores de Escala COCOMO II

		Muy bajo	Bajo	Normal	Alto	Muy alto	Extra alto
Factores de escala (decrementan exponencialmente el esfuerzo)	PREC	6.20	4.96	3.72	2.48	1.24	0.00
	FLEX	5.07	4.05	3.04	2.03	1.01	0.00
	RESL	7.07	5.65	4.24	2.83	1.41	0.00
	TEAM	5.48	4.38	3.29	2.19	1.10	0.00
	PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Procedimiento:

1. Entrada del desarrollo de software (requerimientos y sub-modelo COCOMO II a utilizar)
2. Calcular el valor del factor de ajuste del esfuerzo *M* (ver Tabla 48)
3. Calcular el exponente de esfuerzo *B* y el exponente de calendarización *D* (ver Tabla 49)
    a. Exponente de esfuerzo:

$$B = 0.91 + 0.01 * \sum_{j=1}^{5} SF_j$$

    b. Exponente de calendarización:

$$D = 0.28 + 0.2 * 0.01 * \sum_{j=1}^{5} SF_j$$

4. Encontrar el tamaño del software KSLOC (miles de líneas de código fuente)
5. Calcular el esfuerzo en meses-persona, con la ecuación de esfuerzo

$$Esfuerzo = 2.94 * Tamaño^{B} * M$$

6. Calcular la duración en meses, con la ecuación de duración

$$Tiempo = 3.67 * Esfuerzo^{D}$$

7. Calcular el personal promedio necesario para el proyecto

Personal-Promedio= Esfuerzo / Tiempo

Anexo D

COSMIC FFP

El Método COSMIC FFP. Desarrollado por el Common Software Metrics Internacional Consortium, el denominado método Full Function Points o en general COSMIC-FFP o COSMIC, es considerado como el primer método de medición funcional de segunda generación.

En el método de COSMIC el tamaño funcional del software bajo medición se obtiene como un valor proporcional al número de movimientos de datos de dicho software.

Niveles

Se entiende que el software está dividido en distintos Niveles (Layers) de forma jerárquica. Cada nivel sólo puede comunicarse con sus niveles inmediatamente superior o inmediatamente inferior.
Dentro de cada nivel el software se agrupa en componentes llamados Pares (Peers) teniendo todos ellos el mismo nivel jerárquico.
Los Procesos Funcionales se recogen dentro de los distintos Pares.
El intercambio de información entre distintos niveles se realiza a través de sus respectivos Procesos Funcionales.
Las entidades de una aplicación se asocian en Grupos de Datos.

Movimientos de Datos

Un Proceso Funcional se forma de distintos Movimientos de Datos que son la unidad en la que mide el método. Hay 4 tipos de movimientos de datos básicos los cuales son:

- Entrada – E (Entry). Un dato que cruza la aplicación hasta el proceso funcional.
- Salida – X (Exit). Un dato que cruza la aplicación fuera del proceso funcional.
- Lectura – R (Read). Mueve un único grupo de datos a través de la frontera de la aplicación desde el grupo de datos hasta el proceso funcional.
- Escritura – W (Write). Mueve un único grupo de datos a través de la frontera de la aplicación desde el proceso funcional hasta el medio de almacenamiento persistente.

Todo Proceso Funcional debe contemplar al menos dos movimientos de datos:

- una Entrada (Entry).
- una Salida (Exit) o una Escritura (Write).

Es decir, que como mínimo debe aceptar un dato de entrada y debe mostrar un dato en la salida o debe guardar en el almacenamiento persistente.

Un Movimiento de Datos tiene un valor asignado de 1 CFP (Cosmic Function Point).

Tamaño

El Tamaño de un Proceso Funcional viene determinado por la suma del tamaño de los movimientos de datos que lo forman.
El tamaño de un Módulo de Software en un determinado ámbito se obtiene sumando los tamaños funcionales de los procesos funcionales de los cuales consta dicho módulo de software.
En el caso de que se hable de modificaciones de software existente, el tamaño de la modificación de un proceso funcional es la suma de los tamaños de los movimientos de datos que han sido añadidos, modificados o eliminados de dicho proceso funcional.

De igual forma, el tamaño de los cambios de un Módulo de Software en un determinado ámbito se obtendrá con la agregación de los tamaños de los cambios de los procesos funcionales de los cuales consta dicho módulo de software. La suma total dará como resultado el tamaño funcional del software que se está midiendo.

Anexo E

MARK II FPA

El método Mark II FPA distingue dos tipos de entidades:

- **Entidades Primarias.** Son aquellas por las cuales han diseñado el sistema, para almacenarlas. Por ejemplo, Empleado, Contrato, Factura, etc.
- **Entidades No Principales.** El resto de entidades cuyo objetivo es servir como valores de referencia, validación, etc. y que generalmente suelen constar de un código, descripción y/o conjunto de valores válidos.

Una **Transacción Lógica**, es una combinación de Entrada, Proceso y Salida, correspondientes todos a un proceso único desde un punto de vista lógico, desencadenada por un evento de interés o por una necesidad de usuario.

En el método de Mark II FPA se miden dos componentes del software:

- **Componente funcional.** Es la suma de los tamaños de todas las transacciones lógicas que se identifique dentro del alcance.
- **Componente técnico.** Consiste en la evaluación de las características generales de la aplicación (de forma similar al método de Puntos de Función). Esta parte es opcional y generalmente no se utiliza en las mediciones.

El tamaño de una **Transacción Lógica** es igual a la suma de los elementos de los componentes que la forman:

- **Elementos de Entrada**. Son todos los atributos que recibe como entrada la transacción.
- **Elementos de Proceso.** Todas las entidades principales a las que se referencia: se leen o se actualizan.
  Cada **Entidad Primaria** se contabiliza una única vez sin importar las veces que se referencie, salvo que se relacione consigo misma y las **Entidades No Principales**, no importa el número de ellas al que se acceda, se contabiliza como un acceso a una entidad genérica denominada *Entidad del Sistema*.
- **Elementos de Salida.** Referente a todos los atributos que se muestran en la transacción.

A diferencia del método de IFPUG si un Elemento de Datos aparece en la Entrada y en la Salida, en el método Mark-II FPA deben contarse dos veces.

Para medir el componente técnico deberá evaluar de 0 a 5 las características del sistema que son las siguientes:

- Comunicación de Datos.
- Función Distribuida.
- Rendimiento.
- Configuración fuertemente utilizada.
- Tasa de Transacciones.
- Entradas de datos on-line (interactiva).
- Diseño teniendo en cuenta la Usabilidad.
- Actualización on-line (interactiva).
- Complejidad de Proceso.
- Reusable en otras aplicaciones.
- Facilidad de Instalación.
- Facilidad de Operación.
- Múltiples localizaciones.
- Facilidad de Cambio.
- Requerimientos de otras aplicaciones.
- Seguridad, Privacidad, Auditoria.
- Necesidades de formación de los usuarios.
- Utilización directa por otras empresas.
- Documentación.
- Características definidas por el usuario.

La última de ellas permite añadir a las 19 anteriores, aquellas características que se estiman son importantes para los sistemas.

Anexo F

PUNTOS DE CASO DE USO

Jacobson define el caso de uso como una forma especial de usar el sistema, ejecutando alguna parte de su funcionalidad; cada caso de uso constituye una secuencia completa de eventos disparados por un actor y especifica la interacción que existe entre el actor y el sistema (Jacobson et al., 1992). Por lo tanto, un caso de uso es una secuencia especial de transacciones realizadas por un actor y un sistema, en forma de diálogo. Se define el caso de uso desde la perspectiva del actor. Desde el punto de vista del sistema, se puede decir que un caso de uso es un flujo completo del sistema: es decir, una transacción. Un caso de uso puede tener una o más transacciones, dependiendo del criterio usado por la persona que escribe el caso de uso.

El método de Puntos de Casos de Uso (Use Case Points) fue desarrollado en 1993 por Gustav Karner, bajo la supervisión de Ivar Jacobson (creador de los casos de uso y gran promotor del desarrollo de UML y el Proceso Unificado).

Este método ha sido ampliamente utilizado por la empresa Rational. Su principal ventaja es su rápida adaptación a empresas que ya estén utilizando la técnica de Casos de Uso.

El método utiliza los actores y casos de uso. A los casos de uso se les asigna una complejidad basada en transacciones, entendidas como una interacción entre el usuario y el sistema, mientras que a los actores se les asigna una complejidad basada en su tipo, es decir, si son interfaces con usuarios u otros sistemas. También se utilizan factores de entorno y de complejidad técnica para ajustar el resultado.

Para el cálculo se procede de forma similar a Puntos de Función: se calcula una cuenta no ajustada **Puntos de Casos de Uso (UAUCP**), asignando una complejidad a los actores y a los casos de uso. Esta complejidad será ponderada con un Factor de Ajuste técnico y por un Factor de Ajuste relativo al entorno de implantación, obteniendo tras ello una cuenta de **Puntos de Casos de Uso Ajustados.**

A continuación se observa en detalle los pasos del método (Gómez, 2013):

1) Clasificar cada interacción entre actor y caso de uso según su complejidad y asignar un peso en función de ésta. Para poder clasificar la complejidad de los actores debemos analizar la interacción de éste con el sistema que se va a desarrollar, Tabla 50.

La complejidad de los actores puede corresponderse con una de las tres categorías posibles:

- **Simple.** Representa a otro sistema con una API definida. Se le asigna un peso de valor 1.
- **Medio.** Representa a otro sistema que interactúa a través de un protocolo de comunicaciones. Por ejemplo TCP/IP o a través de un interfaz por línea de comandos. Se le asigna un peso de valor 2.
- **Complejo.** La interacción se realiza a través de una interfaz gráfica. Se le asigna un peso de valor 3.

Tabla 50. Pesos en función de la complejidad de la interacción con los actores

TIPO DE INTERACCIÓN	PESO ASIGNADO
Simple (a través de API)	1
Media (a través de protocolo)	2
Compleja (a través de interfaz gráfica)	3

2) Calcular la complejidad de cada caso de uso según el número de transacciones o pasos del mismo. Para calcular la complejidad de un caso de uso se debe determinar el número de transacciones, incluyendo los caminos alternativos, Tabla 51.

Se entiende por transacción a un conjunto de actividades atómicas, es decir, que se ejecutan todas ellas o ninguna. En función del número de transacciones que posee un caso de uso se clasifica el caso de uso como simple, medio o complejo, siendo la asignación de pesos la que se muestra en la tabla siguiente:

Tabla 51. Pesos en función de la complejidad de los casos de uso

Nº DE TRANSACCIONES DEL CASO DE USO	TIPO	PESO
menor o igual que 3	Simple	5
mayor o igual que 4 y menor que 7	Medio	10
mayor o igual que 7	Complejo	15

3) Calcular los Puntos de Casos de Uso No Ajustados (UAUCP) del sistema. Se obtienen sumando los Puntos de Casos de Uso de todos y cada uno de los actores además de los casos de uso que se han identificado y catalogado en función de su complejidad.

4) Cálculo de los Factores Técnicos (TCF). A cada uno de los Factores Técnicos de la Tabla 52 se le asigna un valor de influencia en el proyecto entre 0 (no tiene influencia) a 5 (esencial), 3 se considera de influencia media. Obtenidos los grados de influencia se multiplican por el peso de cada factor y con la siguiente fórmula se calcula el Factor Técnico que aplica:

$$TCF = 0.6 + \left(0.01 * \sum_{i=1}^{i=13} R_i * influencia \right)$$

Tabla 52. Factores Técnicos para el cálculo del TCF

FACTOR	DESCRIPCIÓN	PESO	INFLUENCIA
R1	Sistema Distribuido	2	0-5
R2	Objetivos de rendimiento	2	0-5
R3	Eficiencia respecto al usuario final	1	0-5
R4	Procesamiento complejo	1	0-5
R5	Código reutilizable	1	0-5
R6	Instalación sencilla	0,5	0-5
R7	Fácil utilización	0,5	0-5
R8	Portabilidad	2	0-5
R9	Fácil de cambios	1	0-5
R10	Uso Concurrente	1	0-5
R11	Características de seguridad	1	0-5
R12	Accesible por terceros	1	0-5
R13	Se requiere formación especial	1	0-5

5) Cálculo de los Factores de Entorno. A cada uno de los Factores de Entorno de la Tabla 53 se le asigna un valor de influencia en el proyecto entre 0 (no tiene influencia) a 5 (esencial), 3 se considera de influencia media.
Obtenidos los grados de influencia se multiplican por el peso de cada factor y con la siguiente fórmula se calcula el Factor de Entorno que aplica:

$$EF = 1.4 + \left(-0.03 * \sum_{i=1}^{i=8} R_i * influencia \right)$$

Tabla 53. Factores de Entorno para el cálculo del EF

FACTOR	DESCRIPCIÓN	PESO	INFLUENCIA
R1	Familiaridad con el modelo de proyecto utilizado	1,5	0-5
R2	Experiencia en la aplicación	0,5	0-5
R3	Experiencia con orientación a objetos	1,0	0-5
R4	Capacidades de análisis	0,5	0-5
R5	Motivación	1,0	0-5
R6	Requisitos estables	2,0	0-5
R7	Trabajadores a tiempo parcial	-1,0	0-5
R8	Lenguaje de programación complejo	-1,0	0-5

6) Obtención de los Puntos de Casos de Uso Ajustados. Una vez calculados los dos factores se calcula el valor ajustado de Puntos Casos de Uso con la siguiente fórmula:

UCP = *UAUCP * TCF * EF*

Después de obtenido el número de Puntos Casos de Uso, si se quiere conocer el esfuerzo necesario para llevarlos a cabo en el método, se provee de un factor de productividad CF.
En este caso el autor propone un valor de **20 horas/persona,** aunque existen distintas propuestas sobre este valor, las cueles han sido mejoradas a lo largo del tiempo. Una forma está basada en los factores ambientales y se calcula de la siguiente manera (Wikipedia, 2014):

1) Contar la cantidad de factores de entorno del R1 al R6 que tienen una puntuación menor a 3, también contar la cantidad de estos mismos del R7 y R8 que son mayores que 3.
2) Para seleccionar el valor del CF evaluar el valor total de factores entre R1 y R6 menores a 3 más la cantidad de factores entre R7 y R8 mayores a 3.
    a. Asignar 20 horas-persona si el valor total es menor o igual a 2.
    b. Asignar 28 horas-persona si el valor total es menor o igual a 4.
    c. Asignar 36 horas-persona si el valor total es mayor o igual a 5.

La fórmula para calcular el esfuerzo en horas-persona viene dado por (Wikipedia, 2014):

$$\mathbf{E} = UCP * CF$$

Este esfuerzo calculado no abarcaría a todas las fases del proyecto sino únicamente a la codificación de los Casos de Uso no estando contemplada otras fases del desarrollo.

Por lo tanto, para calcular el esfuerzo total del proyecto, habría que estimar el esfuerzo en realizar el resto de actividades del proyecto y sumarlas a las obtenidas por el método de Puntos de Casos de Uso.

Anexo G

CLASIFICACIÓN Y ARBOLES DE REGRESIÓN (CART)

CART usa variables independientes (predictores) para construir un árbol binario donde cada nodo hoja representa ya sea una categoría a la que pertenece una estimación o un valor para una estimación. La primera situación se produce con los árboles de clasificación y el segundo se produce con los árboles de regresión, es decir, cada vez que predictores son categóricos, el árbol se denomina un árbol de clasificación, y siempre que los predictores sean numéricos, el árbol se denomina un árbol de regresión.

Con el fin de obtener una estimación, uno tiene que atravesar los nodos del árbol de la raíz a la hoja mediante la selección de los nodos que representan la categoría o valor de las variables independientes asociados con el caso a ser estimado. Por ejemplo, suponiendo que se quiere obtener una estimación de esfuerzo para un nuevo proyecto Web utilizando como base la estructura de árbol de regresión simple se presenta en la Figura 24 (considerado un árbol de regresión porque el esfuerzo es una variable numérica). En este ejemplo, asumiendo que el árbol se generó a partir de datos obtenidos y de las aplicaciones Web realizadas previamente, teniendo en cuenta los valores actuales de esfuerzo y las variables independientes (por ejemplo, las nuevas páginas Web [WP], nuevas imágenes [IM], y las nuevas características / funciones [FN]). Los datos utilizados para construir un modelo CART son llamados "muestra de aprendizaje".

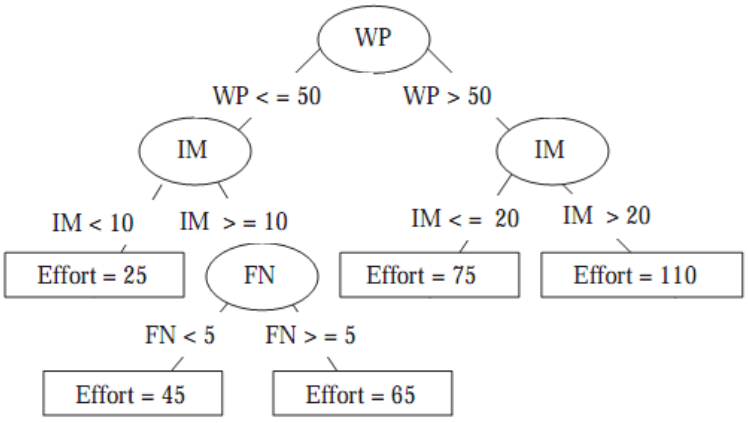


Figura 24. Ejemplo de un árbol de regresión para estimación de esfuerzo web

Una vez que el árbol ha sido construido, puede ser utilizado para estimar el esfuerzo en nuevos proyectos. Por ejemplo y con referencia a la Figura 24, suponiendo que los valores estimados para WP, IM, y FN para un nuevo proyecto Web son 25, 15, y 3, respectivamente, se obtendría un esfuerzo estimado de 45 horas-persona después de navegar el árbol desde la raíz hasta la hoja: Esfuerzo = 45.

De igual forma el uso de la misma figura, si ahora se supone que los valores estimados para WP, mensajería instantánea y FN para un nuevo proyecto Web son 56, 34 y 22, respectivamente, se obtendrías un esfuerzo estimado de 110 horas-persona después de navegar por el árbol desde la raíz hasta la hoja: Esfuerzo = 110.Un ejemplo simple de un árbol de clasificación para la estimación de coste Web se representa en la Figura 25. Utiliza los mismos nombres de variables, como se muestra en la Figura 24, sin embargo, estas variables son ahora todas categóricas, cuando sea posible, las categorías (clases) son Sí y No. El esfuerzo estimado obtenido mediante este árbol de clasificación también es categórico, cuando sea posible, las categorías son alto esfuerzo y poco esfuerzo. Por lo tanto, el árbol presenta en la Figura 25 es un árbol de clasificación en lugar de un árbol de regresión.

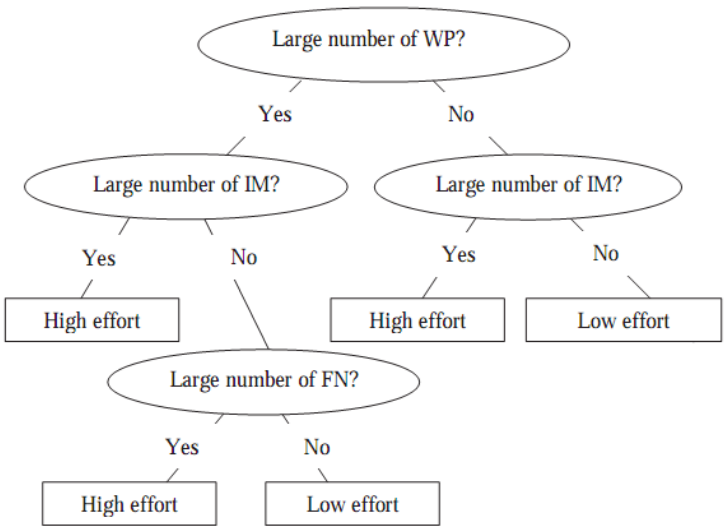


Figura 25. Ejemplo de un árbol de clasificación para estimar el esfuerzo web

El árbol es modelado, construido y luego usado para obtener predicciones para nuevos casos.

Anexo H

Estimación por Puntos de Historias de Usuario mediante Planning Poker

Primero se debe entender cómo es que los equipos de desarrollo estiman el trabajo a realizarse en una iteración.

Afortunadamente, los desarrolladores tienen "objetos" pequeños e independientes por estimar – las historias de usuario – y saben el valor que estas aportan porque de esa manera son diseñadas. Si un equipo puede estimar historias, después este mismo logrará estimar las cosas de valor que estas pueden entregar a un cliente y cuando el equipo esté listo para entregarla.

Hay un número de enfoques para la estimación ágil, entre los métodos más comunes y el recomendado por la mayoría de los instructores ágiles, es la estimación relativa con puntos de historia. Un punto de historia es un número entero que representa la agregación de un número de aspectos, cada uno de los cuales contribuye a la "grandeza" potencial de una historia (Leffingwell, 2011).

- Conocimiento: ¿Se entiende qué es lo que la historia hace?
- Complejidad: ¿Cuán difícil es esta de implementar?
- Volumen: ¿Cuánto de ella es que hay? ¿Cuánto tiempo aproximadamente tomará?
- Incertidumbre: ¿Qué no sabemos y cómo podría afectar la estimación?

Una estimación de puntos de historia combina todos estos aspectos dentro de un número, el cual estima el tamaño de una historia comparada con otras historias de un tipo similar. Los puntos de historia no tienen unidades y son numéricamente relevantes (esto es, una historia de 2 puntos deberá reflejar que será dos veces más grande que una historia de un punto) (Leffingwell, 2011) (Cohn, 2006).

El método de estimación Planning Poker (Cohn, 2006) es uno de los representantes más emblemáticos de los métodos ágiles de estimación, este proceso tiene como primer paso seleccionar una historia de usuario para asignarle una complejidad nominal que servirá de referencia al catalogar al resto de historias de usuario. Los valores que se utilizan para representar la complejidad no tienen un valor absoluto sino que su valor está en función de su posición en escala. Para esto se comenzó utilizando la serie de Fibonacci: 1,2,3,5,8,13,21, …, aunque para evitar que se pensara que hay una precisión matemática en los valores a partir de cierto número se sustituyeron por otros aproximados: 3,5,8,13,21,40,100,… (el 1 y el 2 no se recomienda utilizarlos al no incluir mucha diferencia con respecto al 3). También se puede utilizar los siguientes valores: Extra Small, Small, Medium, Large, Extra Large.

En la Tabla 54 se muestran más claramente las reglas que se siguen al realiza una estimación con Planning Poker (Leffingwell, 2011).

Tabla 54. Reglas para Planning Poker

Reglas para Planning Poker
Los participantes son todos los miembros del equipo de desarrollo (programadores, testers, diseñadores, analistas, etc.)
El dueño del producto (Product Owner) participa pero no estima (el dueño del producto es informado sobre su papel y como puede participar).
A cada estimador se le da un conjunto de cartas con valores 1, 2, 3, 5, 8, 13, 20, 40 y 100. (Nota: algunos instructores realizan un paso opcional de calibraciones: calibrar el tamaño, escogiendo una historia que es acordadamente pequeña y decir que la historia se dice de tamaño 2, después quizás tomar una más, como una más grande que podría ser de tamaño 8)
Por cada historia, el dueño del producto (Product Owner) lee la descripción. Preguntas son formuladas y contestadas. Cada estimador privadamente selecciona una carta que representa su estimación. Los estimadores con la puntuación más alta y la más baja explican su estimación. Después de la discusión, los estimadores reestiman y las cartas son mostradas por segunda vez. Las estimaciones probablemente converjan. Si no, el proceso para esa historia es repetido hasta que se llegue a un acuerdo.
Repetir hasta que todas las historias sean estimadas.

La suma total de los puntos asignados a cada historia de usuario representará la complejidad del proyecto.

Al realizar una estimación del tiempo requerido para el proyecto, las historias de usuario se dividen en actividades, estas se reparten entre los miembros del equipo de desarrollo, el desarrollador responsable de la actividad asigna con

base en su experiencia, la carga de trabajo complejidad de la actividad, un tiempo en horas no mayor a 8 para llevarla a cabo, etc. La suma del tiempo asignado a todas las actividades de todas las historias de usuario, será el tiempo estimado de desarrollo de todo el proyecto.

Para entender mejor el proceso de estimación con los puntos de historia se presenta un ejercicio tomado de (Cohn, 2006) y (Leffingwell, 2011).

A continuación se describe un ejemplo de este método desde la perspectiva de un equipo que está recién entrenado, con el objetivo de que su experiencia y entrenamiento ayude a entender este concepto abstracto en términos más concretos.

Ejercicio: estimación relativa

Durante el entrenamiento, muchos instructores ágiles o de SCRUM corren un ejercicio de ejemplo con sus equipos para que ellos piensen en la relatividad de la "grandeza de las cosas". El ejercicio en sí, junto con los resultados de un equipo, aparece en la Figura 26.

Ejercicio: Asignar "puntos de perro" a cada uno de los siguientes tipos de perros para comparar su "grandeza".

6 Labrador Retriever
~~3~~ 2 Dachshund
7 Great Danes
~~3~~ 4 Terrier
7 German Shepherd
1 Poodle
8 St. Bernard
~~2~~ 3 Bulldog

Figura 26. Un ejercicio de estimación relativa simple, con los resultados de un equipo ágil

En este engañoso y simple ejercicio, los equipos inmediatamente se enfrentan con la ambigüedad.

- ¿Qué quiere decir el instructor con grandeza? ¿Altura, ancho, masa, músculos, mordedura, actitud?
- ¿Qué clase de poodle es? ¿Poodle estándar? ¿Poodle de juguete? "¡eso hace una gran diferencia!"
- ¿Qué escala deberíanusar?

El instructor puede permanecer en silencio durante de este proceso, lo que demuestra al equipo que siempre habrá ambigüedad en el proceso de estimación; o bien actuar como el dueño del producto y responder a cualquier pregunta, lo que demuestra que en caso de duda, hay que preguntar al dueño del producto para aclararla. Con esta experiencia de estimación un tanto trivial, los equipos entonces pueden entender mejor la relatividad de la estimación.

Un punto importante que caracteriza a los métodos ágiles de estimación, es el poco tiempo que invierten en esta actividad. Dean Leffingwell en (Leffingwell, 2011) explica que al gastar mucho tiempo en la estimación no incrementa la precisión de la misma, además de que aburre al equipo. Gastar mucho tiempo estimando es verdaderamente una forma de desperdicio - esfuerzo adicional gastado con resultados muy poco significativos - así que los equipos deben tener cuidado al fijar sus tiempos de estimación y no tratar de convertir esta heurística en una pseudociencia, nos dice este autor.

La complejidad de las historias, los puntos de historia, no se pueden comparar a horas de esfuerzo ya que el sentido que tienen es catalogar la dificultad de la tarea. El número de horas que lleve realizarlas dependerá de la capacitación y/o capacidad de la persona que la realice además de la carga de trabajo del equipo, etc. y por ello variará dicho valor dependiendo de la situación (Gómez, 2013) (Cohn, 2006).

Anexo I

MÉTODO PROBE

La descripción del método que a continuación aparece, es traducción directa de (Humphrey, PSP : a self-irnprovernent process for software engineers , 2005) escrito por el creador del Proceso Personal de Software PSP y por ende del método PROBE, Watts Humphrey, también las figuras y tablas incluidas en este anexo son tomadas de esta referencia bibliográfica. Para más detalles del método PROBE y mejor entendimiento de PSP referirse a (Humphrey, A Discipline for Software Enginnering, 1995).

El método PROBE es una guía en el uso de datos históricos para hacer estimaciones. Por ejemplo, en la estimación del trabajo para desarrollar un sistema de consulta a una base de datos, primero se debería producir el diseño conceptual y después dividirlo en partes. Después se tendría que estimar el número de elementos en cada parte. Por ejemplo, si se estima un total de 80 elementos y se sabe que cada elemento en promedio ha tomado 1.5 horas para desarrollarlo, el tiempo de desarrollo total estimado deberá ser de 120 horas (80 elementos por 1.5 horas).

Sin embargo para hacer estimaciones precisas, se necesita un proceso estadístico formal para determinar el tiempo promedio requerido para desarrollar una parte. El método PROBE indica cómo hacer esto. La Figura 27 muestra una base de datos con información de 21 proyectos. La línea de tendencia para estos datos es calculada con un método llamado **regresión linear.** Este produce una línea que se ajusta exactamente a los datos. La línea de tendencia o línea de regresión, es representada por la siguiente ecuación:

$$Tiempo\ de\ Desarrollo = \beta_0 + \beta_1 * Tamaño\ Estimado$$

$$= 0.17 + 1.53 * 80 = 122.57$$

Aquí β_0 =0.17 y β_1 =1.53, estos valores de β son calculados por el método PROBE descrito en este anexo y ellos son determinados matemáticamente para una mejor representación de la tendencia de los datos. El valor β_1 es el promedio de tiempo de desarrollo requerido por elemento en la base de datos y β_0 es el tiempo extra. En otras palabras esta ecuación indica que, para un proyecto sin elementos, el tiempo de desarrollo deberá ser de 0.17 horas, o cerca de 10 minutos. Para proyectos grandes, después se deberán sumar 1.5 horas por elemento.

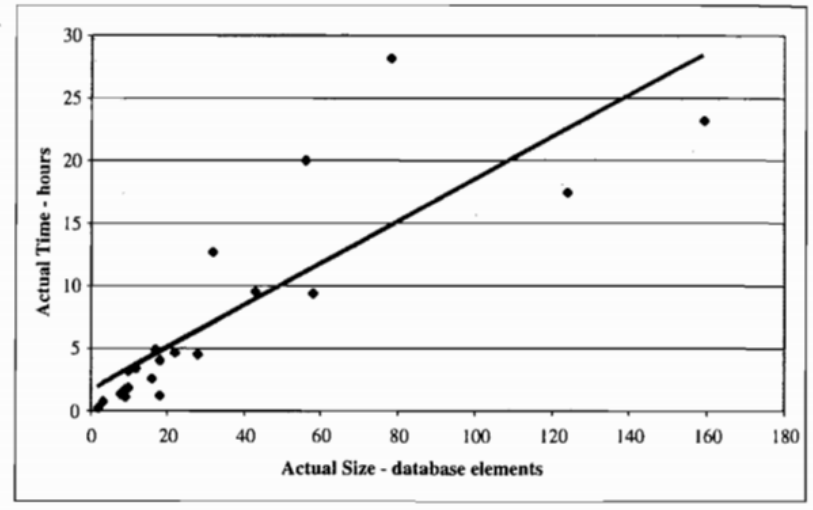


Figura 27. Ejemplo datos de estimación históricos

Estimación basada en proxy

El método PROxy-Based Estimating (PROBE) permite usar cualquier ítem que se seleccione como proxy, siempre y cuando este reúna los criterios para un buen proxy los cuales son descritos en la Tabla 55. Usando los datos de un desarrollador, los pasos básicos del procedimiento para la estimación de tamaño son mostrados en la Figura 28 y descritos en los siguientes párrafos. Recordar que inicialmente, no tendrán suficientes datos para usar el método PROBE completo, por lo cual este tiene varias excepciones, que se describirán posteriormente.

Tabla 55. Criterios de un buen proxy

Criterios para seleccionar un buen proxy
El tamaño medido del proxy debe relacionar el esfuerzo requerido para el desarrollo del producto.
El proxy debe ser automáticamente contable.
El proxy debe ser fácil de visualizar al inicio del proyecto.
El proxy debe ajustarse a las necesidades de cada proyecto y desarrollador.
El proxy debe ser sensible a variaciones de implementación que afectan el costo y esfuerzo del desarrollo.

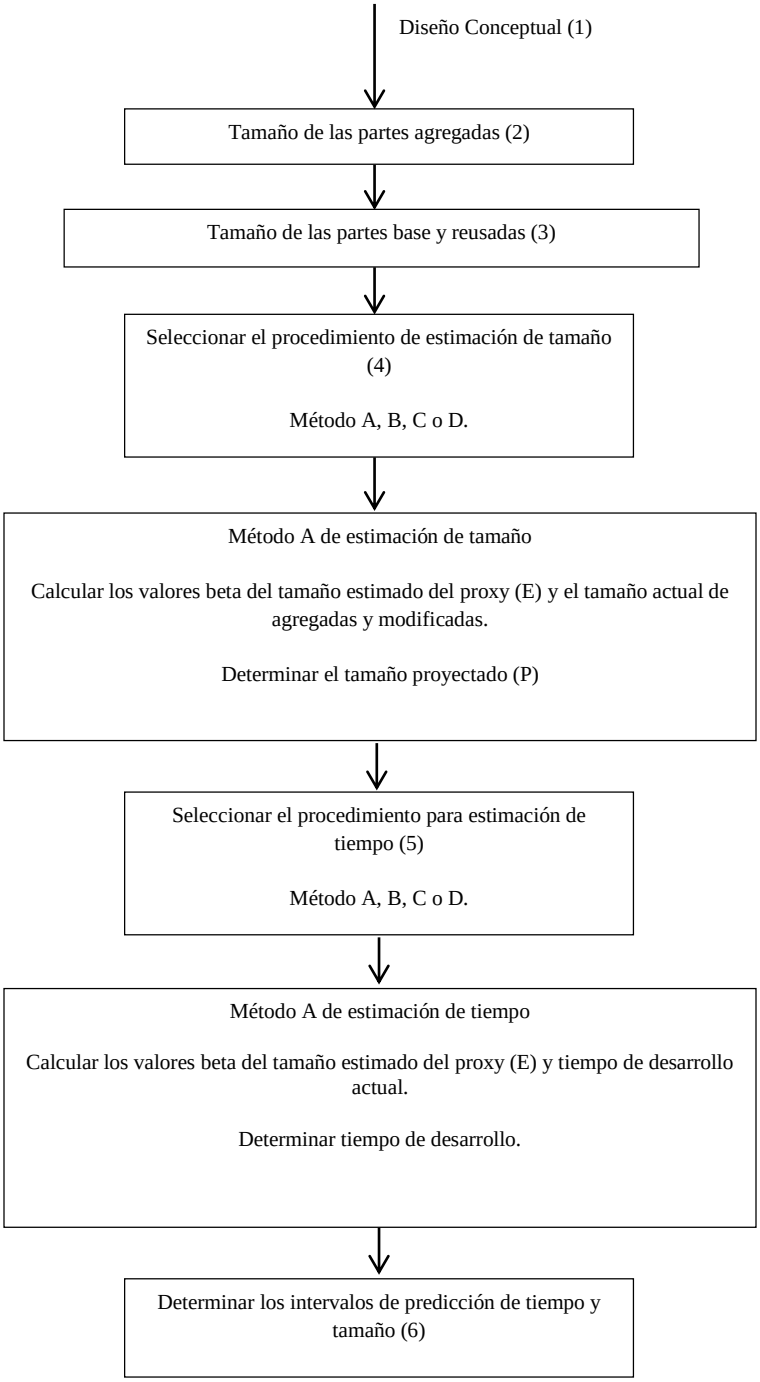


Figura 28. El método de estimación PROBE

PROBE Paso 1: El diseño conceptual

Después de completar el diseño conceptual, se identifican las partes del producto que se planea desarrollar, una vez realizado esto, se está listo para hacer la estimación de tamaño.

PROBE Paso 2: Estimación del tamaño de las partes

Hacer esto para determinar el tipo de parte y después juzgando el número de ítems (métodos) para cada parte y el tamaño relativo de los ítems. Una vez que se conoce si un ítem es muy chico (very small, VS), chico (small, S), mediano (médium, M), grande (large, L) o muy grande (very large, VL), seleccionar el tamaño del ítem de una tabla como la Tabla 56. (no se entiende la idea)

Tabla 56. LOC de partes por item

Tamaño en LOC de clase en C++ por ítem					
Categoría	**Muy chico (VS)**	**Chico (S)**	**Mediano (M)**	**Grande (L)**	**Muy grande (VL)**
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	16.31	16.31	30.09
I/O	9.01	12.06	21.62	21.62	28.93
Logic	7.55	10.98	23.25	23.25	33.83
Set-up	3.88	5.04	8.53	8.53	11.09
Text	3.75	8.00	36.41	36.41	77.66
Tamaño en LOC de objeto en Object Pascal por ítem					
Categoría	**Muy chico (VS)**	**Chico (S)**	**Mediano (M)**	**Grande (L)**	**Muy grande (VL)**
Control	4.24	8.68	17.79	36.46	74.71
Display	4.72	6.35	8.55	11.50	15.46
File	3.30	6.23	11.74	22.14	41.74
Logic	6.41	12.42	24.06	46.60	90.27
Print	3.38	5.86	10.15	17.59	30.49
Text	4.40	8.51	16.47	31.88	61.71

Si una parte tiene ítems de diferentes tipos, estimar cada parte con ítems combinados como una parte separada. Por ejemplo, si una clase A tiene ítems (métodos) de tipo Char, Calc y Print, tratar estas como tres partes: Achar, ACalc y Aprint.

PROBE Paso 3: Estimar el tamaño de las Partes Reusadas y Agregadas a la Base

En el paso 3 PROBE, estimar el tamaño de cualquier otra parte del programa (base, eliminadas, reusadas, etc.). Por ejemplo, si se pudieran encontrar partes que proveen funciones requeridas para el diseño conceptual, podrían reusarlas. Siempre que estas partes trabajen como se pretende y sean de adecuada calidad, reusando partes disponibles se puede ahorrar tiempo considerable. PROBE considera dos tipos de partes reusadas. Primero son las partes tomadas de una librería de reúso. El segundo, las partes nuevas reusables (*new-reusable*), son algunas de las partes agregadas que ya se han estimado.

Si se planea modificar un programa existente, determinar el tamaño del programa base a ser mejorado y el tamaño de cualquier cambio que se espera realizar.

La parte final del paso 3 es sumar todas las partes estimada para obtener el tamaño estimado del proxy (E). E es el número usado por PROBE para hacer la proyección de tiempo y tamaño.

PROBE Paso 4: Procedimiento para la estimación de tamaño

En el paso 4 PROBE, verificar los datos para determinar si se puede usar el método PROBE A o no después de que se ha usado PSP por un tiempo, normalmente se usará el método A para hacer la estimación de tiempo y tamaño. También existen versiones del método PROBE (B, C y D) que se utilizan en caso de tener datos limitados.

Con el tamaño estimado del proxy E, se puede calcular el tamaño proyectado del programa P y tiempo de desarrollo total estimado. Si por ejemplo, los datos históricos mostraron que los programas terminados eran generalmente cerca

del 25% más grandes que lo estimado, se podría sumar un factor de ajuste del 25% a cada estimación. El método PROBE esencialmente hace esto pero de una manera estadística formal. Como se notó anteriormente, el método es llamado regresión lineal. Aunque estos caculos son un poco más complejos que tomar un simple promedio, ellos usan datos históricos para producir una estimación estadística formal. Los parámetros β_0 y β_1 son usados en la siguiente ecuación para calcular el tamaño agregado y modificado proyectado.

$$Tamaño\ agregado\ y\ modificado\ proyectado(P) = \beta_0 + \beta_1 * E$$

Cuando dos conjuntos de datos están fuertemente relacionados, se puede usar el método de regresión lineal para representar la relación, por ejemplo el tamaño de las partes estimadas y el tamaño actual agregado y modificado seguido, están estrechamente relacionados. Esto significa que la regresión lineal es a menudo apropiada. Los parámetros β_0 y β_1 son calculados de datos históricos.

Programas finalizados usualmente contienen más de las partes especificadas en el diseño conceptual. Por ejemplo, ellos probablemente contendrán declaraciones y encabezados de código que no se incluyeron en las partes estimadas. Afortunadamente el método PROBE prevé este factor cuando calcula los parámetros β_0 y β_1.

Por poner un ejemplo, supongamos que β_1=1.3 este indica que los programas terminados de algún programador, históricamente han sido 30% más grandes que el total de las partes estimadas y las modificaciones. Además, el valor de β_o=62, indica que en promedio ha subestimado por 62 LOC (líneas de código). El cálculo de la regresión resulta en un total de 538 LOC agregadas y modificadas proyectadas. En este caso, el tamaño total de los programas es calculado por la suma de las 538 LOC agregadas y modificadas proyectadas más 695 LOC de código base y 169 LOC de código reusado, y la resta de 5 LOC de código modificado. El total estimado es después 1,397 LOC para el programa final. Las líneas de código modificadas son restadas porque de otro modo estas serían contadas dos veces: una vez en las 695 LOC del código base y nuevamente en el tamaño del proxy estimado (E).

PROBE Paso 5: Procedimiento para la estimación de tiempo

En el paso 5 de PROBE, nuevamente revisamos los datos para determinar si se puede usar el método PROBE A para estimar el tiempo o no. Aunque el método A también es el método preferido para estimar tiempo, también se puede usar como alternativa los métodos B, C y D. Una vez que se han obtenido los valores de β para tiempo, se usan junto con el tamaño estimado del proxy (E) para calcular el tiempo de desarrollo estimado.

PROBE Paso 6: El intervalo de predicción

El cálculo final es para el **intervalo de predicción.** El intervalo de predicción es un rango determinado estadísticamente alrededor del tiempo y tamaño estimado dentro del cual el valor actual es probable que caiga. Para un intervalo de predicción del 70%, se esperaría que los valores de tiempo y tamaño actuales caigan afuera de este rango cerca de un 30% de las veces.

Estimando con datos limitados

La Tabla 57 muestra los cuatro métodos PROBE, las condiciones para usarlos y como son usados.

Tabla 57. Los cuatro métodos alternativos de cálculo PROBE

Método	Datos usados para los valores Beta		Requerimientos de los datos
	Tamaño	**Tiempo**	
A	Tamaño estimado del proxy (E) y tamaño actual de programa.	Tamaño estimado del proxy (E) y tiempo actual de desarrollo.	Los datos deben estar correlacionados con un r>=0.7
B	Tamaño planeado del programa y tamaño actual del programa.	Tamaño planeado del programa y tiempo actual de desarrollo.	Los datos deben estar correlacionados con un r>=0.7
C	Tamaño planeado del programa si están disponibles y tamaño actual del programa. Establecer β_0 =0 y β_1 = tamaño actual a la fecha / tamaño planeado a la fecha. Si no hay datos planeados disponibles, establecer β_1 = 1.0	Tamaño planeado del programa si están disponibles y tiempo actual de desarrollo. Si los datos del tamaño planeado no están disponibles, usar tamaño actual. Establecer β_0 =0 y β_1 = tiempo actual a la fecha / tamaño planeado a la fecha. Si los datos planeados no están disponibles, establecer β_1 = tiempo actual a la fecha / tamaño actual a la fecha.	Existen algunos datos de tamaño y tiempo actual.
D			No hay datos.

Para la selección de uno de los cuatro procedimientos PROBE de estimación de tamaño, antes hay que calcular los parámetros β. El método A debe ser la primera opción pero este requiere al menos tres y preferiblemente cuatro datos de tamaño de proxy estimado (E) y tamaño agregado y modificado actual con una correlación r>=0.7. Si no se puede usar el método A, hay que tratar de usar el método B. Este método usa el tamaño agregado y modificado planeado y el tamaño agregado y modificado actual. Nuevamente al menos se deben tener tres datos y preferiblemente cuatro datos con una correlación r>=0.7. Si los datos no son adecuados para el método A y B, usar el método C si se tienen al menos algunos datos del tamaño agregado y modificado planeado y actual. Si no se tiene ningún dato, se debe usar el método D. Con este método, actualmente no se hacen proyecciones, simplemente se adivina un valor para introducirlo como el tamaño agregado y modificado planeado o el tiempo de desarrollo.

Después de la selección de un método de estimación, calcular los valores de β_0 y β_1, y verificar que ellos son razonables. Si β_0 de tamaño es más grande del 25% que el tamaño esperado del programa planeado, o β_1 no está entre 0.5 y 2.0, usa el método B; si los valores de β aún no están dentro de los rangos de decisión, usar el método C. Para el método C, con datos de tamaño planeado y actual del total de agregados y modificados, establecer β_1 igual al tamaño total actual a la fecha dividido entre el tamaño planeado a la fecha y establecer β_o a cero. Si no se tienen datos del tamaño planeado, establecer β_1 =1.0. Para el método D hay que hacer una mejor suposición.

Para la estimación de tiempo, el procedimiento es muy similar. Por ejemplo, si el tiempo β_o es más grande que el 25% del tiempo de desarrollo esperado para el programa planeado, o β_1 no está entre 0.5 y 2.0 usar el método B; y si los valores de β aún no están dentro del rango de decisión, usar el método C. para el método C, si se cuenta con algunos datos del tamaño total agregado y modificado planeado y el tiempo total actual de desarrollo, establecer β_1 igual al tiempo total actual de desarrollo entre el tamaño total planeado y establecer β_0 a cero. Para el método D hay que hacer una mejor estimación.

El paso final del método PROBE es calcular el valor de P, el tamaño proyectado del programa, y el tiempo de desarrollo estimado.

Anexo J

Resultados de encuesta realizados por el Grupo de Investigación de Ingeniería del Software de la Universidad Tecnológica de la Mixteca

Tabla 58. Conclusiones para micro-proyectos exitosos considerando el 100% de éxito tiempo-costo.

MICRO EMPRESAS	PEQUEÑAS EMPRESAS	EMPRESAS MEDIANAS Y GRANDES
8 micro-empresas es 100%. 3 contestaron que si usan métodos de estimación de costos, lo cual es el 37.5%. De estos 3 ninguno especificó que método utiliza. La respuesta fue la siguiente: horas-hombre, propietario y método propio. NOTA1: Horas-hombre es una medida del esfuerzo, más no es un nombre de un método de estimación de costo Nota: dijo que utiliza un método propietario, pero no indicó cual.	5 pequeñas empresas es el 100%. 1 contestó que si usa métodos de estimación, esto equivale al 20%. Sin embargo este 20% contestó que usa costeo directo, es decir, no usa método de estimación de costo como tal.	0 empresas 100 %. 0 proyectos exitosos en 100%, costo y tiempo por tanto es 0%.

Tabla 59. Conclusiones para micro-proyectos exitosos considerando un rango de 80-100% de éxito tiempo-costo.

MICRO EMPRESAS	PEQUEÑAS EMPRESAS	EMPRESAS MEDIANAS Y GRANDES
18 son las empresas que se encuentran en el rango 80-100%. 5 contestaron que si usan métodos de estimación que son el 28%. De estos 5, dos contestaron método propio, otro contesto la unidad de medida del esfuerzo (horas /hombre) y otro que usan método propietario. En conclusión no indicaron que método usan.	12 es el 100% del rango seleccionado 80-100. 3 respondieron que si usan método de estimación, que corresponde al 25%. De estos respondieron: una empresa usa PSP, otro costeo directo y la última "propio de la empresa". Conclusión, lo toman muy poco en cuenta.	2 representan al 100% del rango seleccionado. 1 contesto que si usa método de estimación que representa el 50. El único que contestó usa el método de puntos de función.

Conclusión general

En todo el análisis (ver Tabla 58 y Tabla 59) las empresas encuestadas prácticamente no utilizan métodos de estimación, lo cual parece razonable debido a que los micro-proyectos tienen poco tiempo-costo para desarrollarlos y no es posible dedicar demasiado tiempo a la estimación.

Anexo K

Resultados del análisis con las metodologías de estimación Puntos de Función y COCOMO a una muestra de micro-proyectos de software, realizado como parte de las actividades de investigación de un proyecto desarrollado en la Universidad Tecnológica de la Mixteca

ANÁLISIS DE PUNTOS DE FUNCIÓN A MICRO-PROYECTOS

DATOS DEL PROYECTO						ANÁLISIS			
Nombre	**Tiempo de desarrollo**	**Costo**	**# programadores**	**Lenguaje**	**Año**	**PF sin Ajustar**	**PF Ajustados**	**SLOC estimadas**	**Observaciones**
Proyecto 1	2 meses	$25,520.00	1	Visual Basic	2010	569	569*0.74=**421.06**	42*421.06=**17685**	Análisis directo de la aplicación
Proyecto 2	4 meses	$46,400.00	1	Visual Basic	2010	184	184*0.75=**138**	42*138=**5796**	Análisis directo de la aplicación
Proyecto 3	3 meses	$100,000.00	1	ASP.NET	2007	220	220*0.95=**209**	57*209=**11913**	El análisis se realizó basándose en las pantallas incluidas en el manual de usuario.
Proyecto 4	3 meses	$6,000.00	1	Visual Basic	2001	58	58*0.72=**41.76**	42*41.76=**1754**	El análisis se realizó basándose en las pantallas incluidas en el manual de usuario.
Proyecto 5	95 días	$19,210.00	2	Ruby 3.1.3	2013	232	232*0.87=**201.84**	24*201.84=**4845**	El análisis se realizó basándose en las pantallas incluidas en el manual de usuario.
Proyecto 6	45 días	$53,824.00	2	Ruby 3.1.3	2013	160	160*0.95=**152**	24*152=**6348**	El análisis se realizó basándose en las pantallas incluidas en el manual de usuario.
Proyecto 7	57 días	$ 50,956.00	2	Visual Basic	2012	101	101*0.77=**77.77**	42*77.77=**3267**	El análisis se realizó basándose en las pantallas incluidas en el manual de usuario.

NOTA: Puntos de Función (PF)

Puntos de función ajustados a líneas de código fuente:

Factor de Conversión [SLOC/FP] * Puntos de Función Ajustados [FP] = Líneas de Código Fuente estimadas [SLOC]

Factores técnicos de complejidad	**Evaluación de factores técnicos de complejidad sobre cada proyecto**						
	Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4	Proyecto 5	Proyecto 6	Proyecto 7
1- Comunicación de datos	0	2	5	0	5	5	0
2- Procesamiento distribuido	0	0	4	0	3	5	0
3- Performance (desempeño)	0	0	2	0	0	0	0
4-Configuración del equipamiento	2	2	2	2	1	2	1
5- Volumen de transacciones	0	0	0	0	0	0	0
6- Entrada de datos on-line	0	0	5	0	5	5	0
7- Interface con el usuario	3	2	3	1	2	3	3
8- Actualización on-line	0	0	4	0	3	3	0
9- Procesamiento complejo	0	0	1	0	0	0	1
10- Reusabilidad	1	1	1	1	0	1	1
11-Facilidad de implementación	0	0	0	0	0	0	3
12- Facilidad de operación	2	2	2	2	2	2	2
13- Múltiples locales	0	0	0	0	0	3	0
14- Facilidad de cambios	1	1	1	1	1	1	1
Nivel de influencia total	**9**	**10**	**30**	**7**	**22**	**30**	**12**
Factor Técnico de Complejidad	0.65+0.01*(9)=**0.74**	0.65+0.01*(10)=**0.75**	0.65+0.01*(30)=**0.95**	0.65+0.01*(7)=**0.72**	0.65+0.01*(22)=**0.87**	0.65+0.01*(30)=**0.95**	0.65+0.01*(12)=**0.77**

ANÁLISIS DE LOS PROYECTOS CON EL MODELO COCOMO

Nombre del proyecto	Modelo COCOMO	Modo de desarrollo del proyecto	Tamaño en KDSI o KSLOC	Factor de ajuste	Parámetros de esfuerzo		Esfuerzo estimado [persona-mes]	Parámetros de duración		Duración estimada [meses]
					a	b		a	b	
Proyecto 1	Intermedio	Orgánico	17.685	0.97197741	3.20	1.05	$3.20 * (17.685)^{1.05} * 0.972 =$ **64**	2.50	0.38	$2.50 * (64)^{0.38}$ =**12.14**
Proyecto 2	Intermedio	Orgánico	5.796	0.94088722	3.20	1.05	$3.20 * (5.796)^{1.05} * 0.940 =$ **19**	2.50	0.38	$2.50 * (19)^{0.38}$ = **7.6**
Proyecto 3	Intermedio	Orgánico	11.913	1.21570891	3.20	1.05	$3.20 * (11.913)^{1.05} * 1.216 =$ **53**	2.50	0.38	$2.50 * (53)^{0.38}$ = **11.3**
Proyecto 4	Intermedio	Orgánico	1.754	0.79133567	3.20	1.05	$3.20 * (1.754)^{1.05} * 0.791 =$ **5**	2.50	0.38	$2.50 * (5)^{0.38}$ = **4.6**
Proyecto 5	Intermedio	Orgánico	4.845	0.94088722	3.20	1.05	$3.20 * (4.845)^{1.05} * 0.940 =$ **16**	2.50	0.38	$2.50 * (16)^{0.38}$ = **7.16**
Proyecto 6	Intermedio	Orgánico	6.348	1.317242108	3.20	1.05	$3.20 * (6.348)^{1.05} * 1.31 =$ **29**	2.50	0.38	$2.50 * (29)^{0.38}$ = **8.9**
Proyecto 7	Intermedio	Orgánico	3.267	0.927187526	3.20	1.05	$3.20 * (3.267)^{1.05} * 0.927 =$ **10**	2.50	0.38	$2.50 * (10)^{0.38}$ = **6**

Tabla de manejadores de costo por proyecto

Atributos	Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4	Proyecto 5	Proyecto 6	Proyecto 7
Fiabilidad (RELY)	1.00	1.15	1.40	1.15	1.00	1.40	1.15
Tamaño de Base de datos (DATA)	1.08	1.00	1.00	0.94	1.00	1.00	0.94
Complejidad (CPLX)	1.00	1.00	1.15	0.85	1.15	1.15	1.15
Restricciones de tiempo de ejecución (TIME)	1.00	1.00	1.11	1.00	1.00	1.00	1.00
Restricciones de memoria de almacenamiento (STOR)	1.00	1.00	1.06	1.00	1.00	1.00	1.06
Volatilidad de la máquina virtual (VIRT)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Tiempo de respuesta (TURN)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Capacidad de análisis (ACAP)	1.00	1.00	1.00	1.00	1.00	1.00	0.86
Experiencia en la aplicación (AEXP)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Calidad de los programadores (PCAP)	1.00	1.00	0.86	1.00	1.00	1.00	1.00
Experiencia del personal en la máquina virtual (VEXP)	1.10	1.00	1.00	1.00	1.00	1.00	1.00
Experiencia en el lenguaje (LEXP)	0.95	0.95	0.95	1.00	0.95	0.95	0.95
Técnicas actualizadas de programación (MODP)	0.91	0.91	0.91	0.91	0.91	0.91	0.91
Utilización de herramientas de software (TOOL)	0.91	0.91	0.83	0.91	0.91	0.91	0.91
Restricciones de tiempo de desarrollo (SCED)	1.04	1.04	1.04	1.04	1.04	1.04	1.04
Factor de Ajuste	**0.97197741**	**0.94088722**	**1.21570891**	**0.79133567**	**0.94088722**	**1.317242108**	**0.927187526**

NOTA: El Factor de Ajuste es el producto de los 15 manejadores de costo

Anexo L

Lista de historias de usuario de la aplicación para la estimación del esfuerzo de desarrollo y administración de micro-proyectos de software

ID	HISTORIA DE USUARIO	PRIOR.	PHU	ACTIVIDADES		HORAS
1	Yo como PO quiero agregar PBIs al Product Backlog para representar mi funcionalidad deseada	1	13	Diseñar base de datos e implementar	5	21
				Implementar clase de conexión y probar la conexión entre la aplicación y la bd.	3	
				Programar módulo "crear PBI"	3	
				Programar módulo "crear proyecto"	5	
				Programar inserción de imágenes	5	
3	Yo como PO quiero asignar una prioridad a los PBIs, para registrar la importancia que estos tiene para mi	2	5	Programar método para asignar prioridades	1	3
				Hacer pruebas	2	
11	Yo como SM quiero registrar la estimación que cada integrante asigna a una historia de usuario, para ejecutar adecuadamente el Planning Poker	3	21	Diseñar protocolo de comunicación cliente-servidor	5	45
				Programar manejo de hilos en la clase servidor	8	
				Programar manejo de hilos y repaint de interface de cliente y servidor	8	
				Programar protocolo de comunicación	3	
				Programar módulo servidor	5	
				Programar módulo cliente	5	
				Programar la comunicación	3	
				Probar la interacción con las pantallas	3	
12	Yo como SM quiero registrar la estimación final de una historia de usuario, para que esta quede grabada en el historial.	4	5	Programar método guardar estimación	2	4
				Probar método	2	
14	Yo como SM quiero arrastrar tareas a la pila del Sprint, para definir las historias de usuario que se desarrollaran en dicho Sprint.	5	21	Diseñar la interface para que permita visualizar los PBIs, darles prioridad al ordenarlos y arrastrar los PBIs	5	23
				Programar método que guarde los PBIs seleccionados para el sprint actual. Para que cuando se inicie el programa se muestre el estado actual del sprint	5	
				Programar la funcion que permita arrastrar de forma gráfica los PBIs	8	
				Hacer pruebas y correcciones	5	
15	Yo como SM quiero capturar el objetivo del Sprint, para tener claro dicho objetivo.	6	3	Método que guarda la información del sprint en la base de datos	2	2

20	Yo como SM quiero arrastrar los PBIs del Product Backlog hacia diferentes columnas del SCRUM Board según sea su status durante el Sprint, para visualizar todo el desarrollo del proyecto	7	13	Diseñar el efecto visual	5	10
				Programar efecto y guardar información en la base de datos	5	
10	Yo como SM quiero dar de alta a mi equipo de trabajo para controlar el número de integrantes y sus tareas.	8	21	Diseñar el efecto visual al asignar las personas al equipo	3	18
				Programar el módulo que carga imágenes	3	
				Módulo para guardar información del equipo	3	
				Investigar como cargar imágenes desde la aplicación	3	
				Programar el efecto visual para arrastrar imágenes	3	
				Pruebas	3	
2	Yo como PO quiero agrupar mis PBIs en funcionales y tecnológicos para identificarlos mejor	9	8	Diseñar la forma de agrupar los PBIs fácilmente	3	6
				Programar y probar	3	
4	Yo como PO quiero categorizar mis PBIs en módulos funcionales, para clasificarlos en base a funcionalidad	10	3	Programar método	1	1
6	Yo como PO quiero modificar la prioridad, categoría o contenido de cualquier PBI, para actualizar mis requerimientos con base en mis necesidades	11	5	Diseñar forma alterna para visualizar fácilmente los PBIs, aparte de la visual en el Scrum Board.	2	3
				Implementar clase de conexión y probar la conexión entre la aplicación y la bd.	1	
21	Yo como ME quiero asignar una lista de actividades a cada historia de usuario e indicar la duración en horas de cada actividad, para llevar un control de mis tareas	12	13	Diseñar clase	3	10
				Diseñar interface	3	
				Implementar clase e interface	4	
7	Yo como PO quiero registrar las pruebas de aceptación de cada PBI, para indicar el criterio de finalización de cada PBI	13	8	Reutilizar clase para insertar tareas y modificar parámetros	3	5
				Probar	2	
5	Yo como PO quiero consultar el status de cada PBI, para saber si están en progreso, pendientes o terminados.	14	5	Diseñar interface y consultas sql para mostrar el estado	2	4
				Implementar	2	
24	Yo como ME quiero actualizar el estado de las pruebas de aceptación para actualizar el avance del Sprint	15	8	Diseñar interface y consultas sql para mostrar el estado	3	5
				Implementar	2	
8	Yo como PO quiero revisar el status de las pruebas de	17	5	Diseñar interface y consultas sql para mostrar el estado	2	3

	aceptación para saber si el PBI ha quedado culminado.			Implementar	1	
22	Yo como ME deseo actualizar el estado de las tareas de cada historia de usuario, para actualizar el avance	18	5	Diseñar interface y consultas sql para mostrar el estado	2	3
				Implementar	1	
16	Yo como SM quiero generar la gráfica Sprint Burndown Chart, para dar seguimiento al avance del Sprint.	19	21	Investigar cómo se grafica en java		18
17	Yo como SM quiero generar la gráfica Release Burndown Chart, para dar seguimiento a todo el proyecto.	20	21			18
18	Yo como SM quiero registrar todas las conclusiones del Sprint Retrospective, para registrar la retroalimentación obtenida en la reunión	21	13			8
		TOTAL PHU	217		TOTAL HRS	210

Anexo M

Manual de Usuario de la Aplicación

Ingresar a la aplicación

Al ejecutar el archivo .jar correspondiente a la aplicación, se muestra un mensaje inicial el cual indica que la aplicación se ha conectado exitosamente a la base de datos, Figura 29.

Figura 29. Mensaje confirmación de conexión exitosa con la base de datos.

Pestaña Proyectos

La pantalla principal es el "MENU SCRUM" y la primera pestaña que se visualiza es la de "Proyectos", Figura 30. Esta pestaña permite al usuario dar de alta un nuevo proyecto, modificar o eliminar existentes. Asignar equipos de trabajo a un determinado proyecto. También muestra en forma de lista todas las personas disponibles para ser asignadas a los diferentes roles que SCRUM maneja.

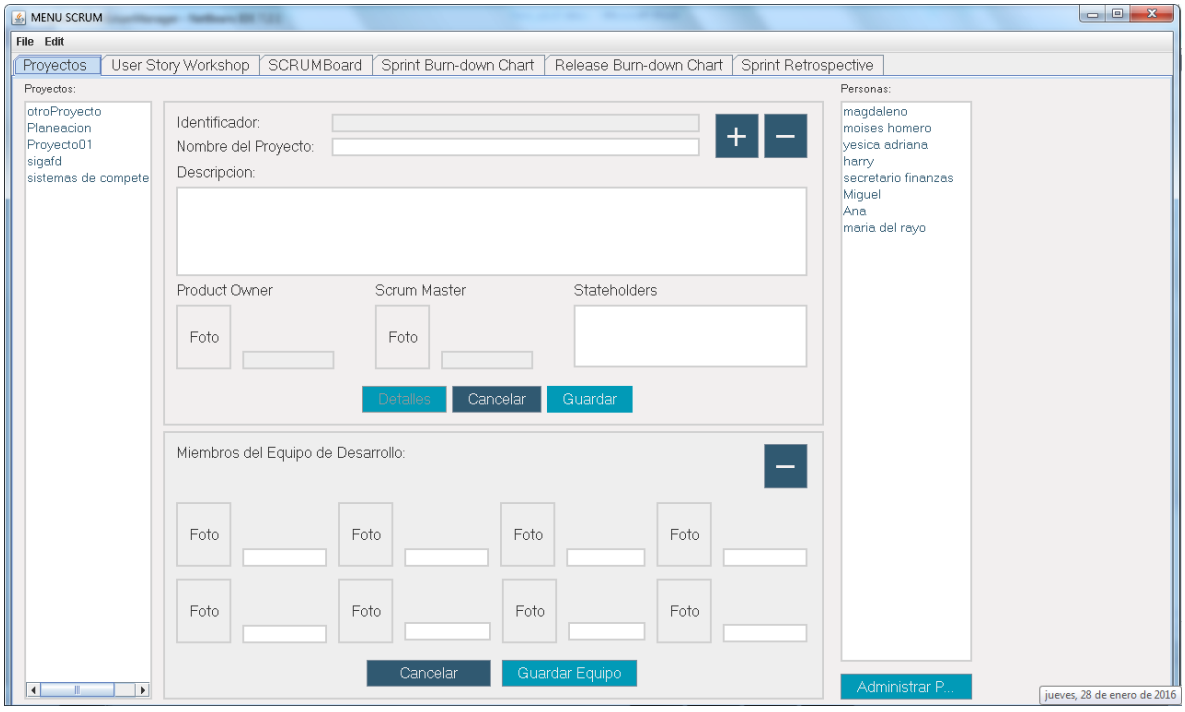


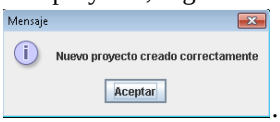
Figura 30. Pantalla principal "MENU SCRUM", pestaña "Proyectos".

Agregar un nuevo proyecto

Para agregar un nuevo proyecto presionar el botón **“Agregar nuevo proyecto”**  y teclear:

- Nombre del proyecto
- Descripción del proyecto
- Una lista de las personas relacionadas con el proyecto, también llamadas stateholders, separadas por punto y coma ‘;’.

Arrastrar de la lista **“Personas”** el nombre de persona que será el Product Owner del proyecto y arrastrar otra persona la cual será el SCRUM Master del proyecto. Al soltar el nombre sobre la caja de texto ubicada debajo de la etiqueta “Foto”, se mostrará la foto de la persona y su nombre.

Presionar el botón **“Guardar”** para guardar el nuevo proyecto, Figura 31. Un mensaje de confirmación se mostrará, indicando que se creó correctamente el proyecto 

.

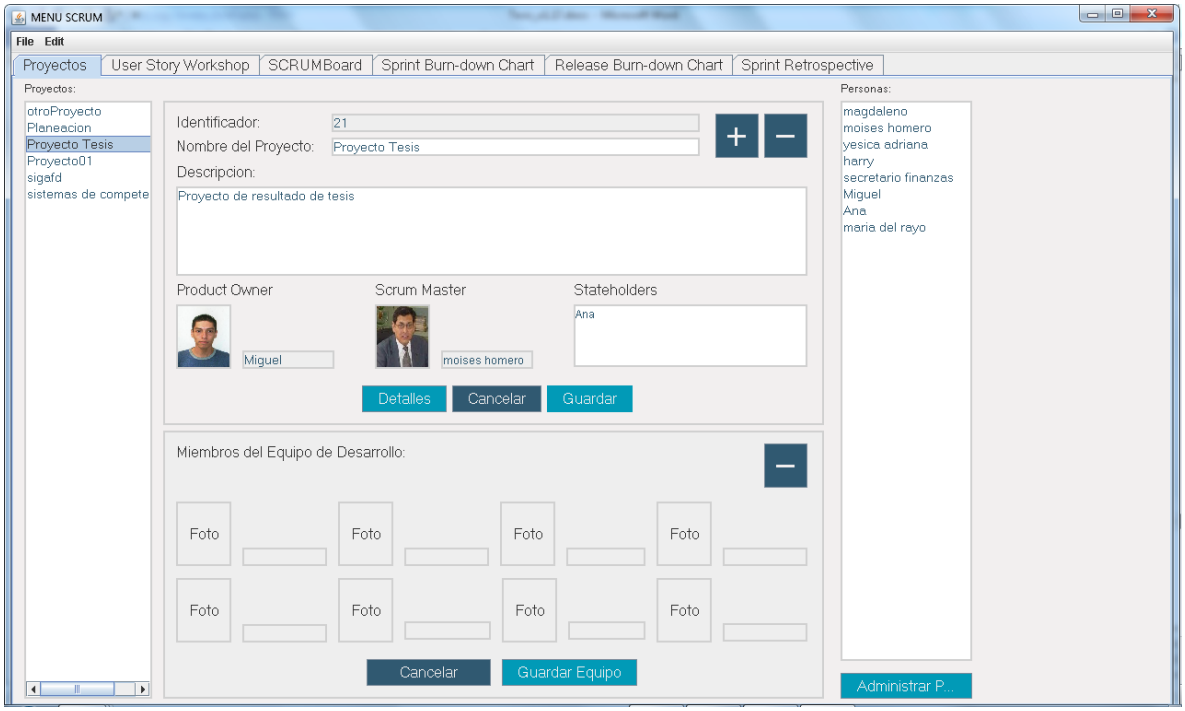


Figura 31. Crear nuevo proyecto

Agregar información detallada del proyecto

Para agregar o modificar la información detallada sobre un proyecto existente, seleccionar uno de la lista **“Proyectos”**. Presionar el botón **“Detalles”**. Teclear los módulos en los cuales se dividirá el proyecto (cada módulo separado por punto y coma ‘;’). Si para el proyecto seleccionado aún no se ha estimado su esfuerzo de desarrollo y

no tiene asignado un tiempo en horas, entonces ingresar de forma manual el costo del proyecto. De lo contrario calcular el costo dando clic sobre el botón **"Calcular"**.

Presionar el botón **"Guardar"**, para que la información agregada o modificada sea guardada dentro del sistema, Figura 32. Enseguida se muestra un mensaje de confirmación.

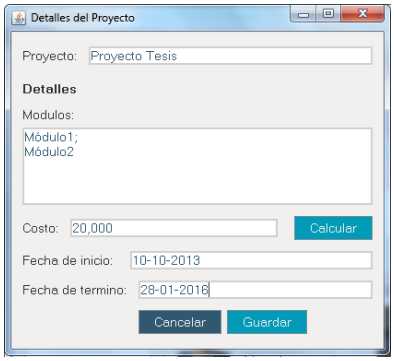


Figura 32. Detalles de un proyecto existente

Calcular el costo de un proyecto

Para utilizar la opción del botón **"Calcular",** el proyecto seleccionado previamente debió haber pasado por la etapa de Estimación de Esfuerzo de Desarrollo con Planning Poker. Al pasar por esa etapa, se conoce el tiempo estimado de desarrollo del proyecto, lo que permite calcular un aproximado del costo con base al tiempo, número de personas en el equipo de desarrollo más el SCRUM Master y sus salarios respectivos, Figura 33.

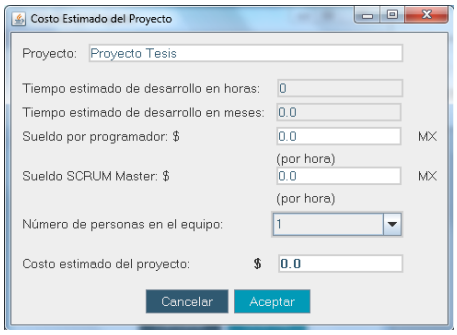


Figura 33. Calcular costo del proyecto

Eliminar un proyecto

Para eliminar un proyecto existente, seleccionar un proyecto de la lista de proyectos y presionar el botón **"Eliminar proyecto seleccionado"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar el proyecto, **"No"** para cancelar la eliminación.

Agregar un equipo de desarrollo a un proyecto

Para agregar un equipo de desarrollo, seleccionar un proyecto de la lista de proyectos. Arrastrar el nombre de una persona de la lista de **"Personas"**, soltar el nombre sobre el campo de texto debajo de la etiqueta **"Foto"**, al hacerlo se mostrará el nombre y fotografía de la persona. Como mínimo se puede agregar una persona al equipo de desarrollo y el tamaño máximo es de ocho. También hay que notar que no es necesario agregar un equipo a un proyecto, ya que como sucede en muchos casos los micro-proyectos de desarrollo solo cuentan con una persona y esa persona puede ser el SCRUM Master únicamente.

Para guardar el equipo creado y asignarlo al proyecto seleccionado, presionar el botón **"Guardar Equipo"**, Figura 34, a continuación se muestra un mensaje de confirmación.

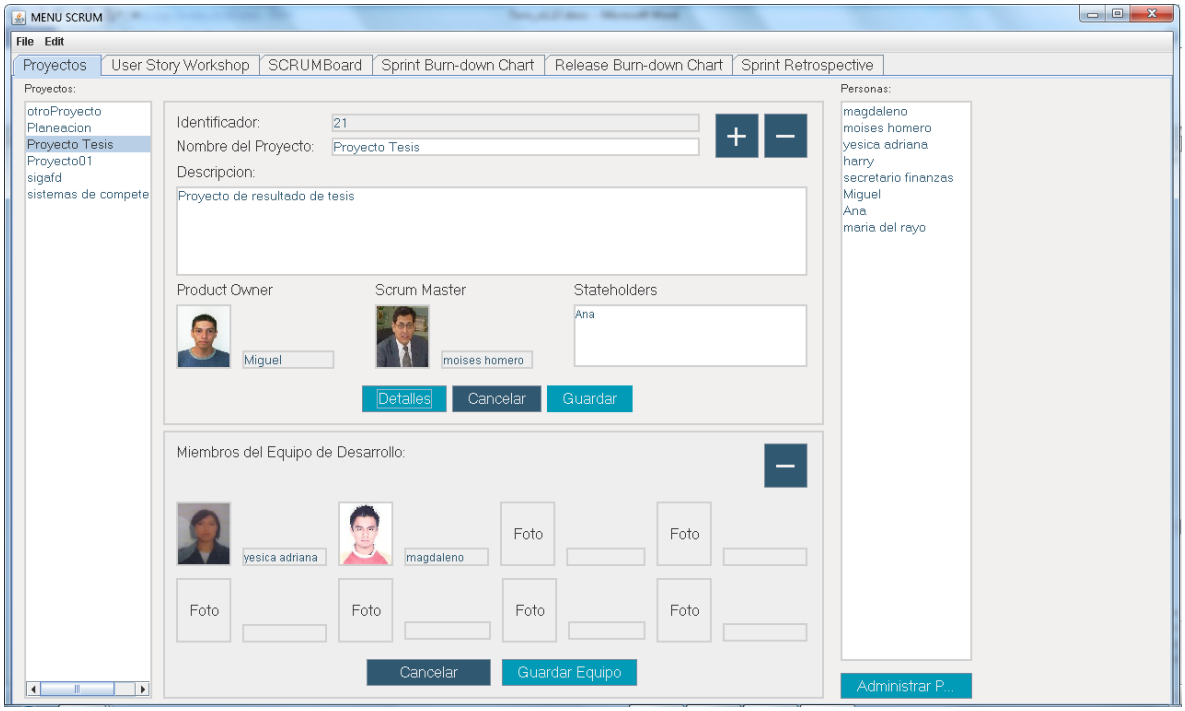


Figura 34. Agregar equipo de desarrollo a un proyecto

Eliminar equipo

Para eliminar algún equipo perteneciente a un proyecto, primero seleccionar un proyecto después presionar el botón **"Eliminar equipo seleccionado"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar el equipo, presionar **"No"** para cancelar la eliminación.

Agregar persona a lista de personas

Estando en la pestaña “Proyectos”, seleccionar el botón “Administrar personas”, a continuación se muestra una pantalla, que permite agregar los datos personales de una nueva persona, ver las personas existentes y su información para ser modificada o eliminada de la lista.

Para agregar una nueva persona dar clic en el botón **“Agregar nueva persona”**  .

Seleccionar una foto de la persona y posteriormente ingresar:

- Nombre
- Apellido paterno
- Apellido materno
- Titulo con el que cuenta (ingeniero, licenciado, maestro, doctor, etc).
- Puesto que ocupa
- Número de celular
- Número fijo
- Dos correos diferentes
- Cuenta de Skype
- Cuenta Facebook
- Twitter
- Ubicación dentro de la organización

Para guardar la información de la nueva persona, dar clic al botón **“Guardar”**, Figura 35. A continuación se mostrará un mensaje de confirmación.

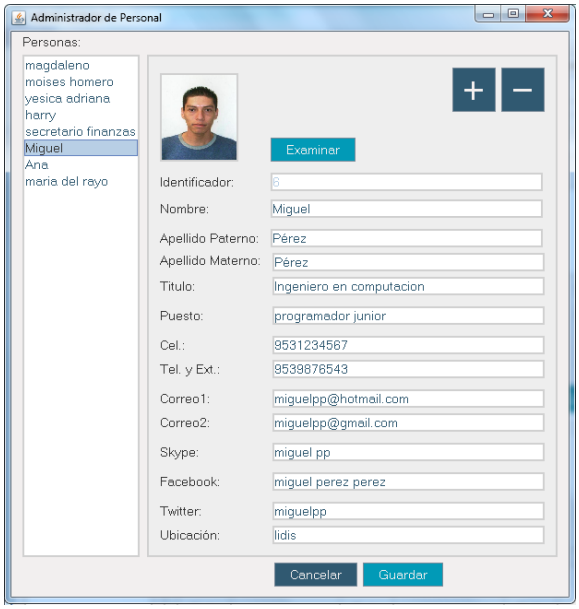


Figura 35. Agregar nueva persona

Eliminar persona

Para eliminar la información de una persona, seleccionar una de la lista "Personas", presionamos el botón **"Eliminar persona seleccionada"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar el equipo, presionar **"No"** para cancelar la eliminación.

Pestaña User Story Workshop

Una de las reuniones fundamentales de SCRUM es en la que se define toda la funcionalidad que el Producto Owner espera del programa y el número de Sprints aproximados que le tomará al equipo implementar esa funcionalidad. Esa reunión se divide en dos partes. En la primera parte el Producto Owner y el SCRUM Master se reúnen sin el equipo de desarrollo. El Product Owner lista la funcionalidad deseada. Cada funcionalidad es redactada por el SCRUM Master como una Historia de Usuario o Product Backlog Item(PBI) y es agregada a la pila del producto. En la segunda parte de la reunión el equipo se reúne con el SCRUM Master (opcional que también participe el Product Owner, eso lo decide el equipo) para realizar una estimación de esfuerzo y tiempo de desarrollo de toda la funcionalidad contenida en el Product Backlog, al contar con esa información el equipo estima el número de Sprints que le tomará terminar el proyecto. La pestaña "User Story Workshop" permite gestionar las actividades de esta reunión, Figura 36.

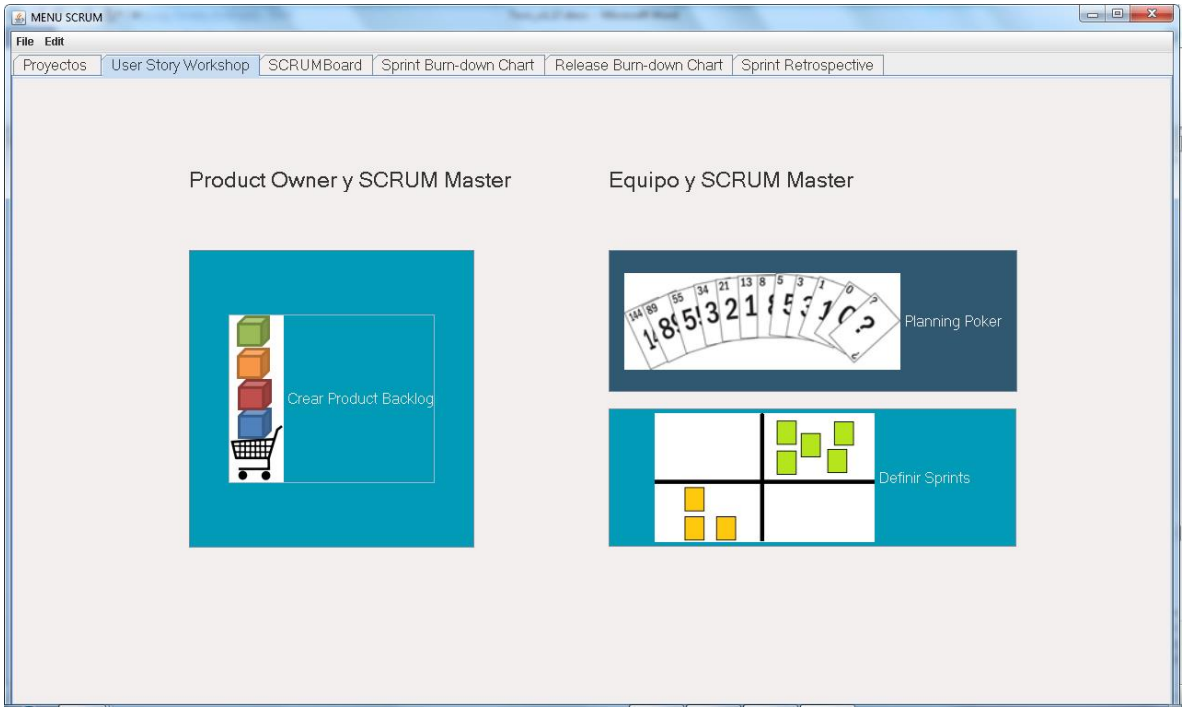


Figura 36. Pestaña User Story Workshop

Crear Product Backlog

En la pestaña **"User Story Workshop"**, dar clic al botón "CREAR PRODUCT BACKLOG". A continuación se muestra la pantalla de la de la Figura 37. Previamente seleccionar un proyecto de la pestaña **"Proyectos"**.

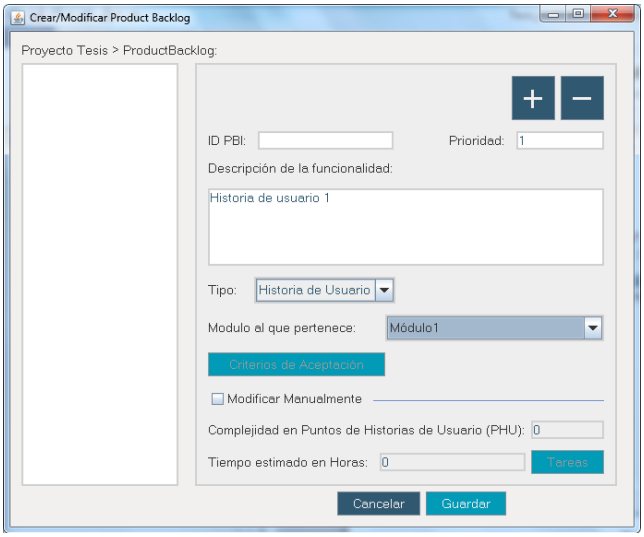


Figura 37. Agregar PBI al Product Backlog

Para agregar un nuevo Product Backlog Item (PBI), presionar el botón **"Agregar nuevo PBI"**  , ingresar la descripción del PBI, seleccionar el tipo de PBI, ya sea historia de usuario (funcional), técnico o de pruebas. Si en los detalles del proyecto se agregaron módulos, seleccionar el módulo al que pertenece el PBI. Para guardar la información presionamos el botón **"Guardar"** y a continuación se muestra un mensaje de confirmación.

El botón **"Criterios de Aceptación"**, permite asignar una lista de condiciones que debe de cumplir el PBI para alcanzar el estado de terminado.

La sección **"Modificar Manualmente"**, permite modificar directamente el esfuerzo de desarrollo del PBI medido en Puntos de Historias de Usuario (PHU) y el tiempo estimado en horas que tomará desarrollar el PBI.

Eliminar un PBI

Para eliminar un PBI existente, seleccionar uno de la lista y presionar el botón **"Eliminar PBI seleccionado"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar el PBI, presionar **"No"** para cancelar la eliminación.

Agregar a un PBI Criterios de Aceptación o Lista de Pruebas

Los Criterios de Aceptación, son una lista de condiciones que debe cumplir el PBI para que pueda ser considerado como "Hecho" o "Terminado". Para asignar a cada PBI una lista de criterios de aceptación, primero seleccionar un PBI de la lista y después dar clic al botón **"Criterios de Aceptación".** A continuación se muestra la pantalla, que permite agregar una lista de pruebas.

Para agregar una nueva prueba, dar clic al botón **"Agregar nueva prueba"**  , ingresar el nombre de la prueba, la descripción de la prueba, el estado es por defecto pendiente.

También seleccionar a la persona encargada de probar que se cumpla la prueba. Para asignar una persona responsable presionar el botón **"Seleccionar"**, a continuación se muestra la lista de personas entre las que se puede elegir. Seleccionar una y presionar "Aceptar", Figura 39.

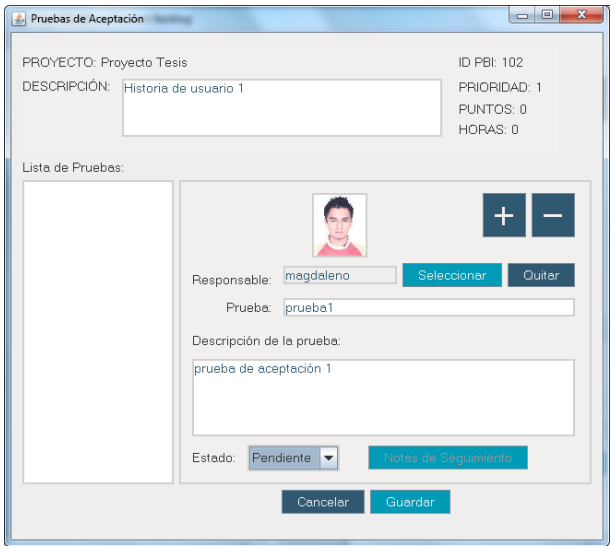


Figura 38. Agregar nueva prueba

Figura 39. Asignar persona

Para guardar la información de la nueva prueba presionar el botón **"Guardar"**, Figura 38.

Agregar notas de seguimiento a una Prueba

Las notas de seguimiento describen con más detalle el estado de la prueba. Para agregar notas de seguimiento a una prueba existente, seleccionar una prueba de la lista de pruebas y presionar el botón **"Notas de seguimiento"**. A continuación se muestra una pantalla, Figura 40, tecleamos las notas referentes al estado de la prueba y presionamos el botón **"Guardar"**.

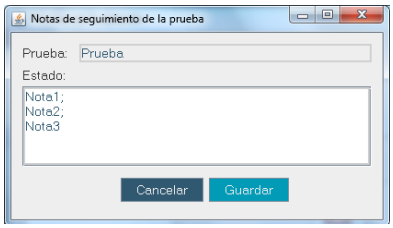


Figura 40. Notas de seguimiento de pruebas

Eliminar una prueba existente

Para eliminar una prueba, seleccionarla de la lista y presionar el botón **"Eliminar prueba seleccionada"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar la prueba, presionar **"No"** para cancelar la eliminación.

Estimar el esfuerzo de desarrollo del Product Backlog con Planning Poker

El siguiente paso una vez construido el Product Backlog, consiste en realizar una estimación del esfuerzo y tiempo de desarrollo necesario para que el equipo diseñe, implemente y pruebe la funcionalidad descrita en todos los PBIs. Como se explicó anteriormente el método de estimación que se utiliza en SCRUM es conocido como Planning Poker. Esta actividad es realizada por el equipo de desarrollo y dirigida por el SCRUM Master (si el equipo de desarrollo está de acuerdo el Product Owner puede estar presente).

Como se ha explicado esta dinámica fue implementada como un juego de cartas entre personas conectadas en una red local. Por lo tanto el SCRUM Master debe presionar el botón **"Planning Poker"**, a continuación se muestra la Figura 41, esta permite levantar el servidor que administrara la partida de cartas y le permite al SCRUM Master conectarse al servidor como un cliente.

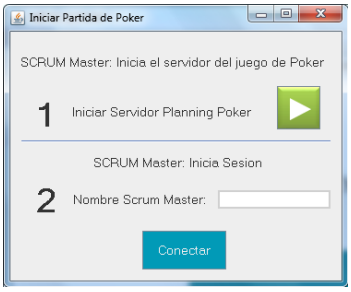


Figura 41. Iniciar Servidor Planning Poker

El SCRUM Master debe presionar el botón verde para iniciar el servidor, a continuación se muestra un mensaje de confirmación. Para conectarse al servidor el SCRUM Master debe ingresar un nombre y presionar el botón **"Conectar"**, Figura 42 . A continuación se muestra la pantalla Figura 43. Una vez conectado el SCRUM Master, los demás jugadores pueden comenzar a conectarse. El número mínimo de jugadores es 1 y el máximo es 10.

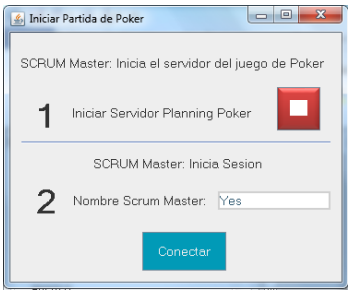


Figura 42. Conectarse al servidor como administrador de la partida

El **"Tablero SCRUM Master"** permite al SCRUM Master llevar el control del juego, seleccionando el orden en que cada PBI se va estimando. Enviar el PBI a estimar a los jugadores para que estos puedan visualizar la información, mostrar las cartas de los jugadores, registrar los puntos de historia asignados a cada PBI, reiniciar la partida y posteriormente descomponer en tareas cada PBI.

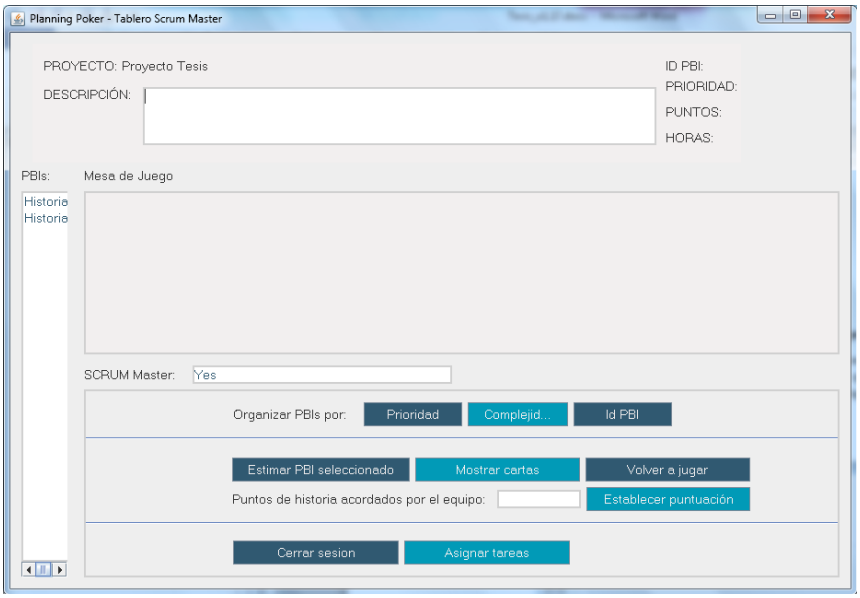


Figura 43. Tablero SCRUM Master

Al ser un juego en red, cada miembro del equipo deberá tener el programa cliente para conectarse al servidor y poder ser parte del juego de poker, para estimar el esfuerzo y tiempo de desarrollo del Product Backlog.

Cada miembro del equipo debe ejecutar el programa cliente, ingresar su nombre e ingresar la dirección IP del servidor. Presionar el botón **"Conectar"**, Figura 44.

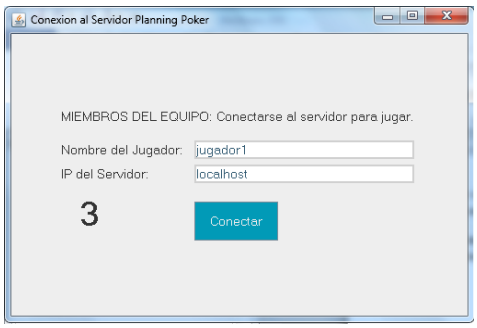


Figura 44. Conexión con el Servidor Planning Poker

Una vez que cada jugador se ha conectado, la pantalla del SCRUM Master se ve como en la Figura 45.

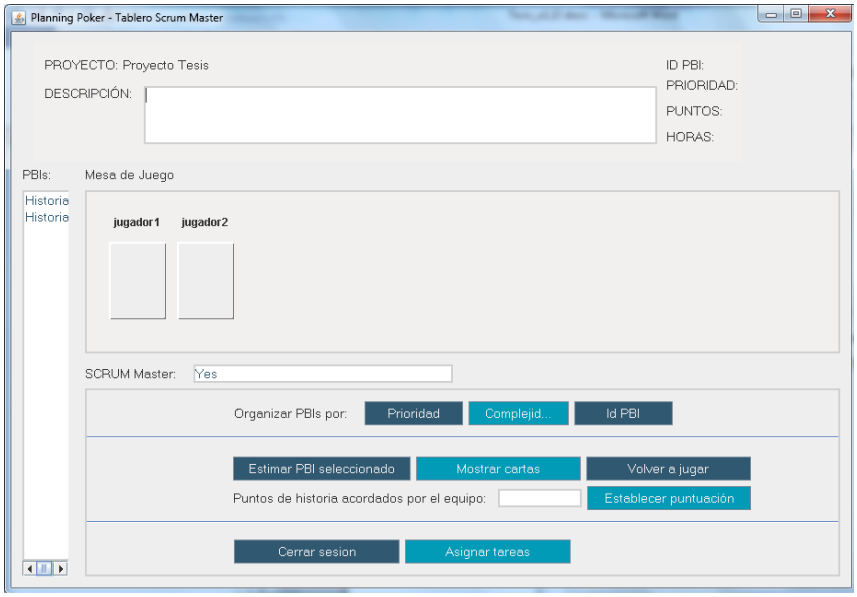


Figura 45. Jugadores conectados

El **"Tablero Jugador"** de cada jugador se visualiza como en la Figura 46, muestra los demás jugadores conectados y un maso de cartas con los puntos de historia que podrán ser asignados al PBI en el proceso de estimación.

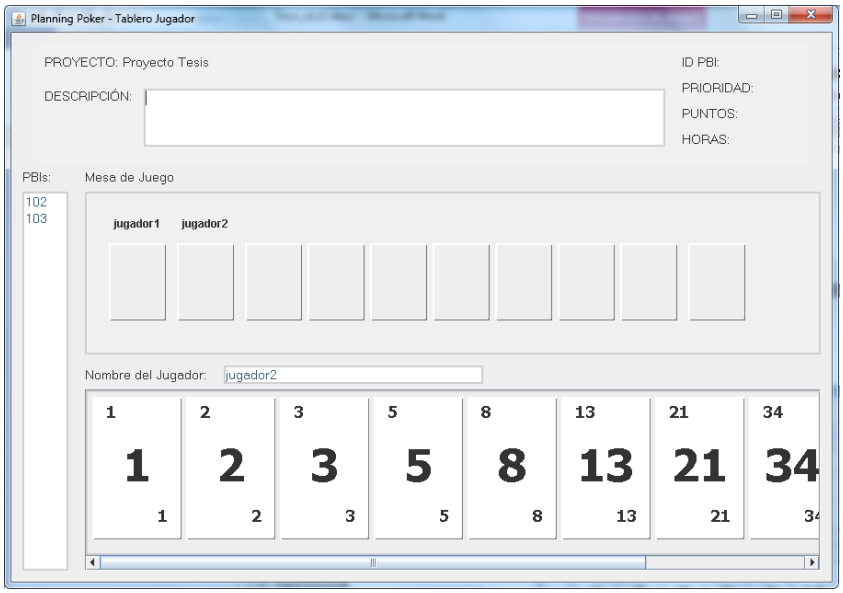


Figura 46. Pantalla de cada jugador

El procedimiento para le estimación de esfuerzo de desarrollo de describe a continuación:

1. El SCRUM Master selecciona el PBI a estimar y presiona el botón **"ESTIMAR PBI SELECCIONADO"**, para enviar a los jugadores la información, Figura 47.

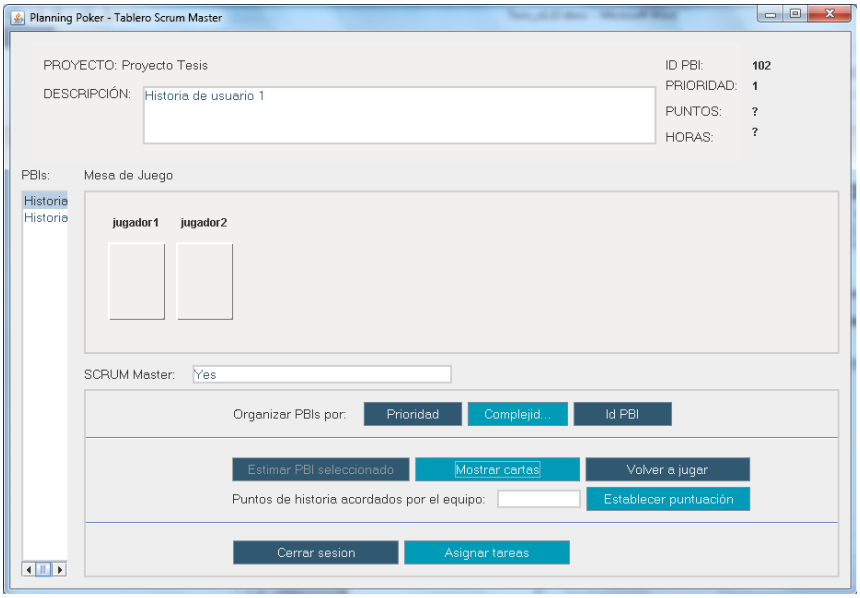


Figura 47. Seleccionar PBI a estimar

2. Los jugadores reciben la información y se visualiza en su tablero, Figura 48.

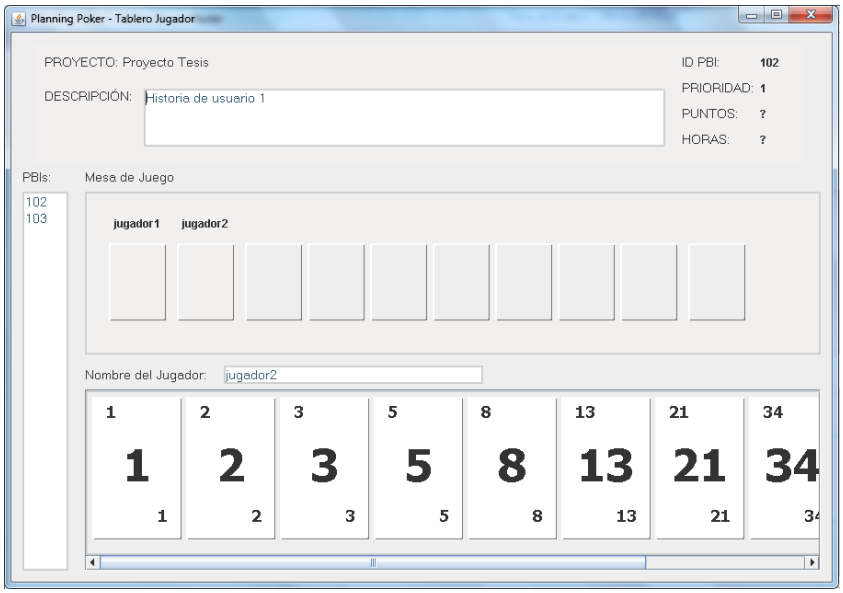


Figura 48. Jugador recibe PBI a estimar

3. Cada jugador selecciona de entre su maso de cartas, el valor del esfuerzo de desarrollo en puntos de historia para el PBI, con base a su experiencia. Esperar a que los demás jugadores seleccionen su carta, Figura 49.

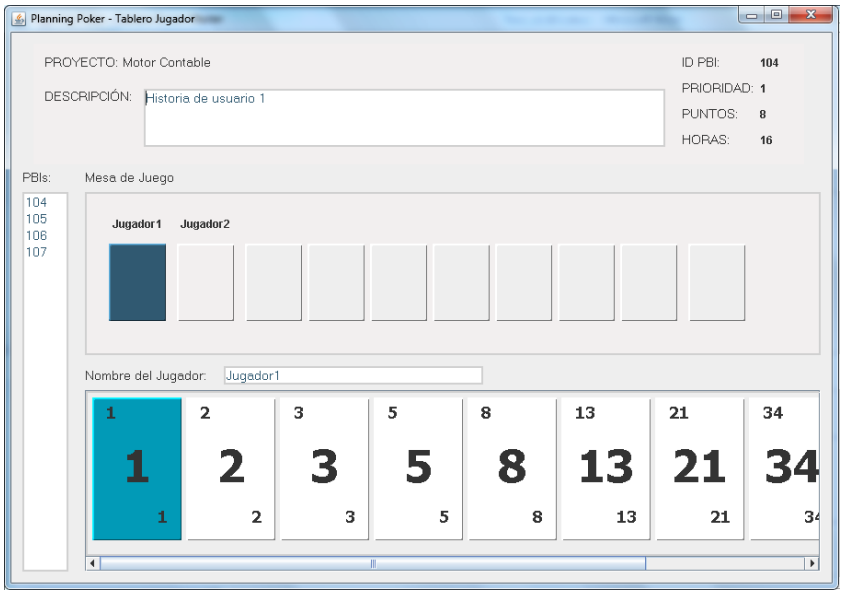


Figura 49. Jugada

4. El SCRUM Master solo observa los jugadores que ya realizaron su jugada y espera a que todos seleccionen su carta, para posteriormente mostrar las cartas, Figura 50.

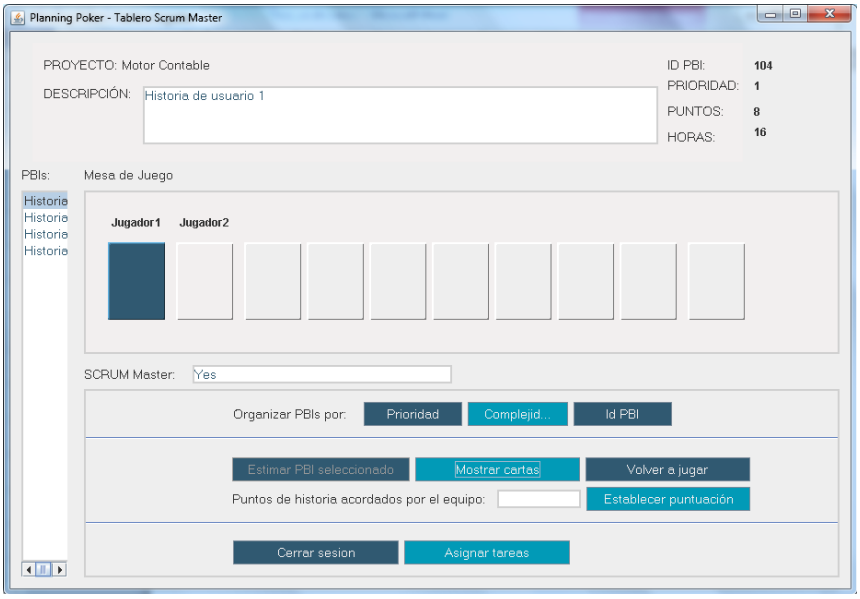


Figura 50. SCRUM Master espera jugadas

5. Todos los jugadores eligen sus cartas, Figura 51.

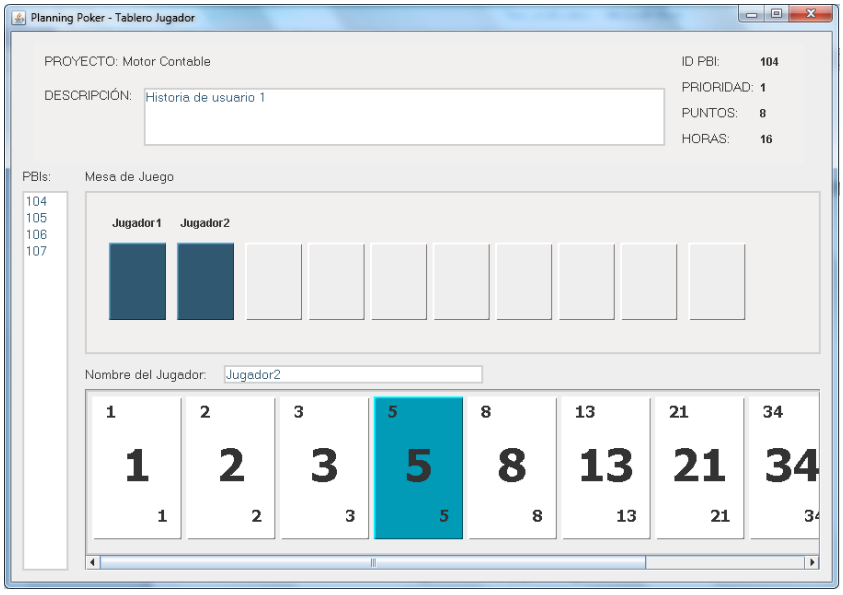


Figura 51. Todos los jugadores eligen sus cartas

6. Una vez que todos los jugadores han realizado sus jugadas, el SCRUM Master muestra las cartas presionando el botón **"MOSTRAR CARTAS"**, Figura 52.

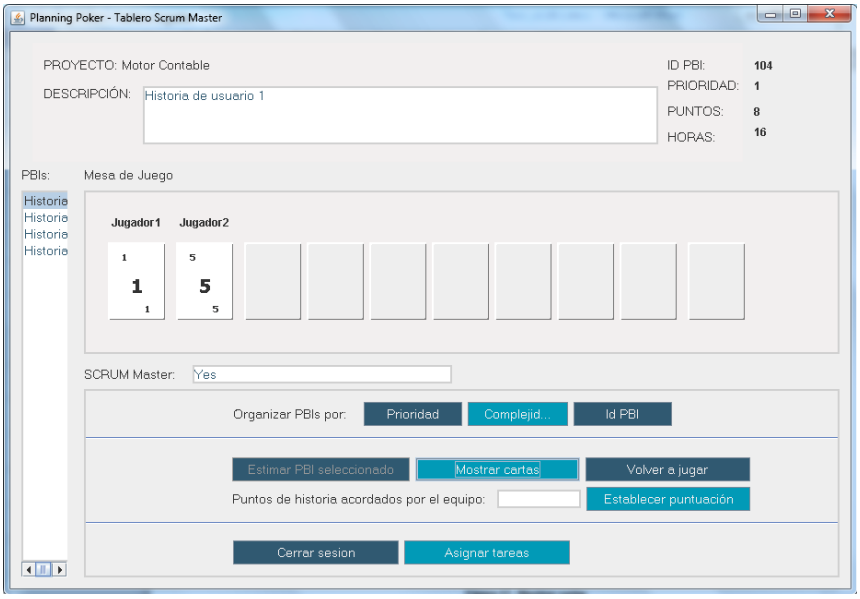


Figura 52. Mostrar cartas

7. Si hay variación en las estimaciones el equipo dialoga sobre las razones por las cuales decidieron asignar la puntuación. Para volver a jugar y seleccionar sus cartas el SCRUM Master limpia el tablero presionando el botón **"VOLVER A JUGAR"**. La dinámica se repite máximo tres veces, si a la tercera vez no hay consenso el SCRUM Master decide la puntuación a asignar al PBI.
8. Para asignar los puntos de historia estimados a un PBI, el SCRUM Master ingresa el valor en el campo de texto **"PUNTOS DE HISTORIA ACORDADOS POR EL EQUIPO"** y presiona el botón **"ESTABLECER PUNTUACION"** para guarda el valor estimado. El valor asignado es enviado a los jugadores para que puedan visualizar la información en su tablero.
9. Se repite el proceso con cada uno de los PBIs del Product Backlog.

Una vez finalizada la estimación de esfuerzo en puntos de historias de usuario con Planning Poker, el equipo debe descomponer en tareas cada uno de los PBIs y asignarles el tiempo en horas que tomaría llevar a cabo cada tarea. Los miembros del equipo dialogan sobre cuál es el tiempo más conveniente para cada tarea, así como por qué y se define quien se encargara de cada tarea. Al realizar esto se estima el tiempo de desarrollo del Product Backlog.

1. Desde su tablero el SCRUM Master selecciona el PBI a descomponer en tareas y lo envía a los jugadores presionando el botón **"ASIGNAR TAREAS"**, Figura 53.

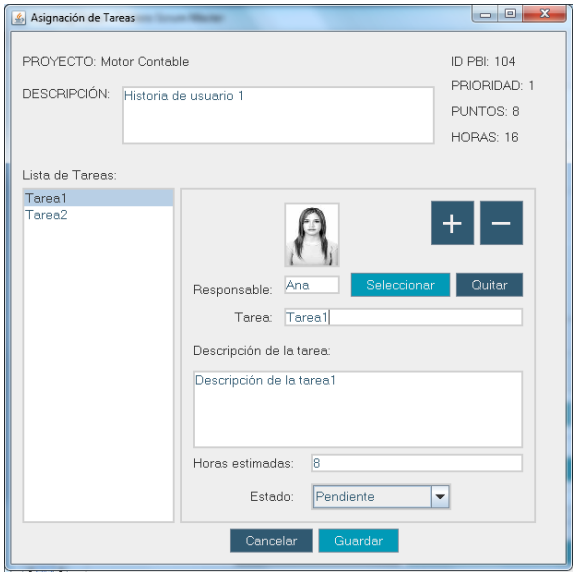


Figura 53. Dividir en tareas un PBI

Para agregar una nueva tarea presionar el botón **"Agregar nueva tarea"**  . Ingresar el nombre de la tarea, la descripción de la tarea, las horas estimadas para terminarla, el estado por defecto es pendiente, definir la persona que se encargara de la tarea y presionar el botón **"Guardar"**. La información de cada tarea es enviada a cada uno de los jugadores. En la parte superior de la pantalla, la cual muestra la información del PBI que se está descomponiendo, automáticamente se muestra el total de horas estimadas para el PBI, resultado de la suma de los tiempos estimados para cada tarea.

Para eliminar una tarea, seleccionarla de la lista y presionar el botón **"Eliminar tarea seleccionada"**  . A continuación se muestra un mensaje de confirmación, presionar **"Sí"** para eliminar la tarea, presionar **"No"** para cancelar la eliminación.

Al finalizar el proceso como resultado se contará con un tiempo estimado en horas para llevar a cabo el Product Backlog.

Una vez estimado el esfuerzo y tiempo de desarrollo, los miembros del equipo deben cerrar el programa cliente y el SCRUM Master terminar la sesión presionando el botón **"CERRAR SESION"**.

Definir Sprints

Cuando se ha estimado el tiempo de desarrollo, se define el número de Sprints necesarios para llevar a cabo el proyecto. En la pestaña **"User Story Workshop"**, presionar la opción **"Definir Sprints"**, a continuación se muestra la Figura 54. La pantalla permite definir la duración de cada Sprint y con base en esa duración se calcula el número de Sprints necesarios, para guardar el número presionar el botón **"Guardar"**.

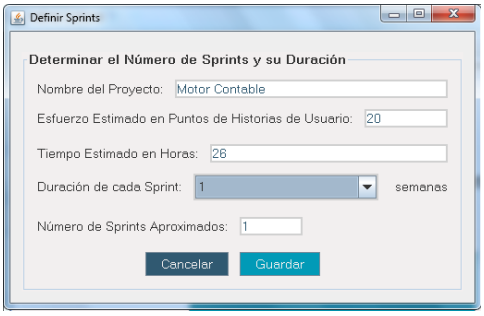


Figura 54. Definir Sprints

Pestaña SCRUM Board

La manera en que en SCRUM se lleva el seguimiento del avance durante un Sprint es a traves del SCRUM Board. En él se visualizan todos los PBIs pertenecientes al Product Backlog, los PBIs seleccionados para implementarse en un Sprint, el estado de cada PBI y la persona encargada de realizarlo, Figura 55.

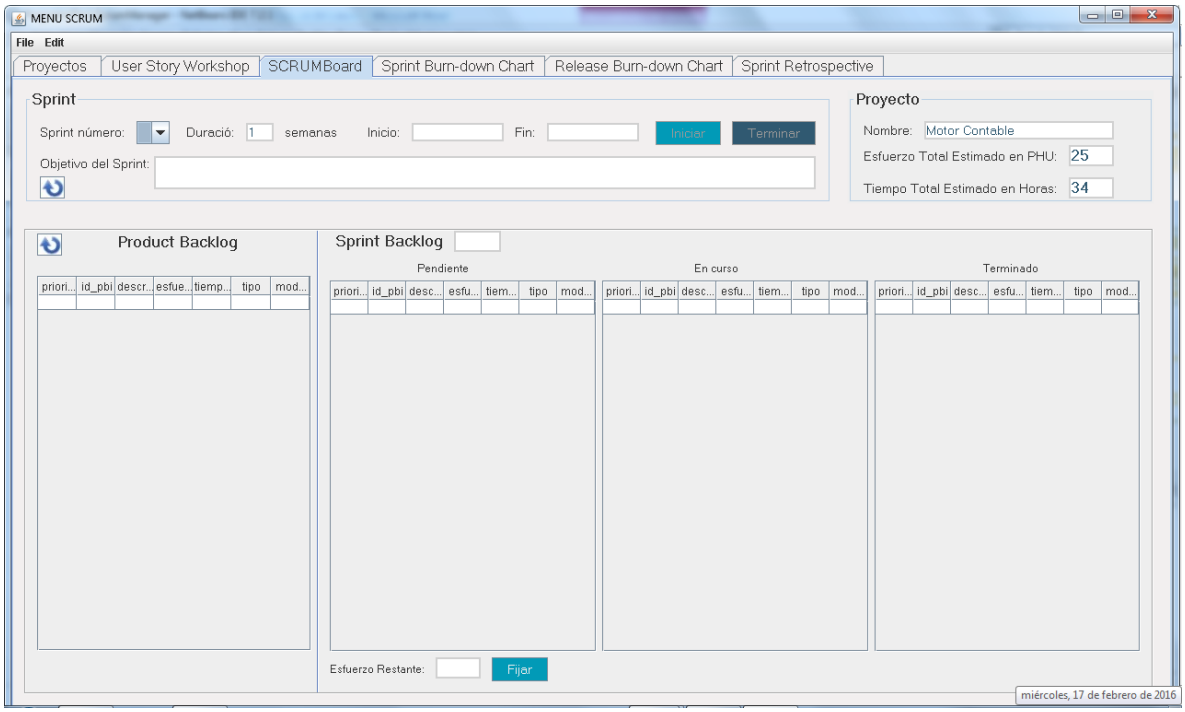


Figura 55. Pestaña SCRUM Board

En la sección **"SPRINT"**, seleccionar el número de Sprint en curso, ingresar la fecha de inicio y de fin, ingresar el objetivo del Sprint y presionar el botón **"Iniciar"**, para posteriormente poder seleccionar los PBIs que se desarrollarán en el sprint seleccionado.

En la sección **"PROYECTO"**, se muestra la información del proyecto que se va a desarrollar.

Para indicar cuales PBIs se trabajarán, seleccionar del Product Backlog uno por uno y arrastrarlo a la columna **"Pendiente"** del Sprint. Debajo de esta columna se muestra un campo que indica cual es el trabajo restante del Sprint (el trabajo es la suma de los puntos de historias de cada PBI en la columna).

Cuando un PBI este en desarrollo moverlo a la columna **“En curso”**, arrastrándolo desde la columna **“Pendiente”**.

Los PBIs terminados moverlos a la columna “Terminado”, arrastrándolos desde la columna **“En curso”.**

Al terminar el Sprint presionar el botón **“Terminar”.** En la Figura 56, se muestra el estado del SCRUM Board al término de un Sprint.

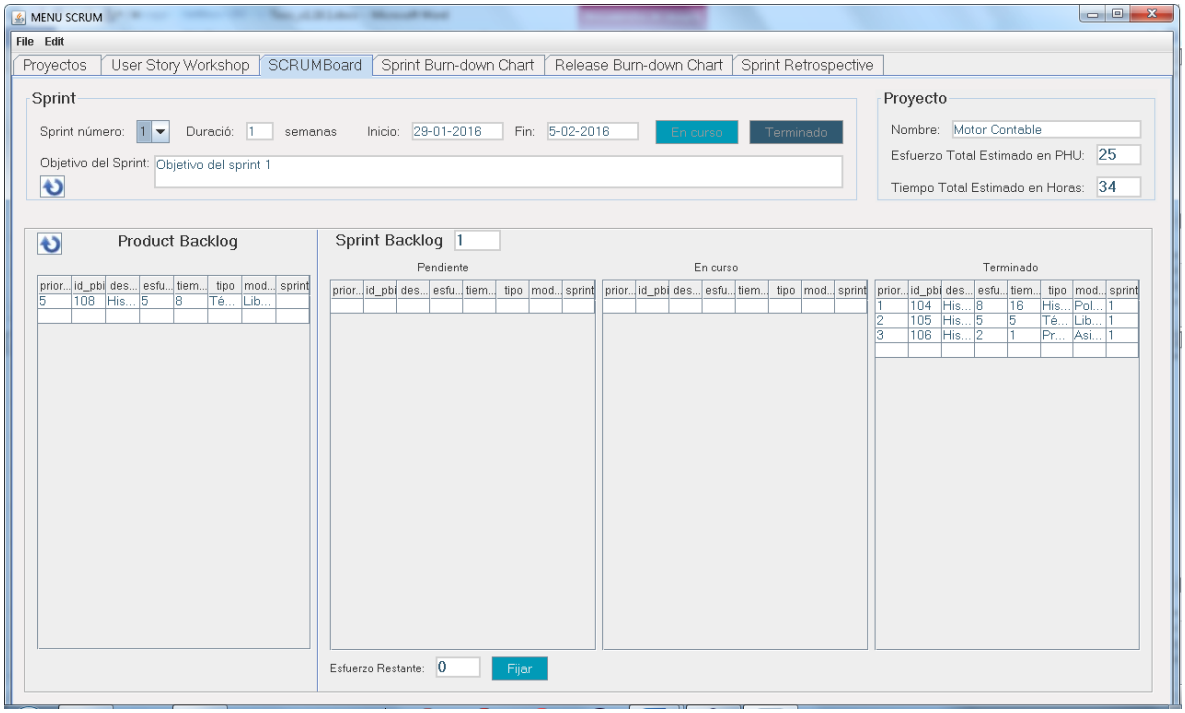


Figura 56. Sprint terminado

El SCRUM Board durante un Sprint, Figura 57.

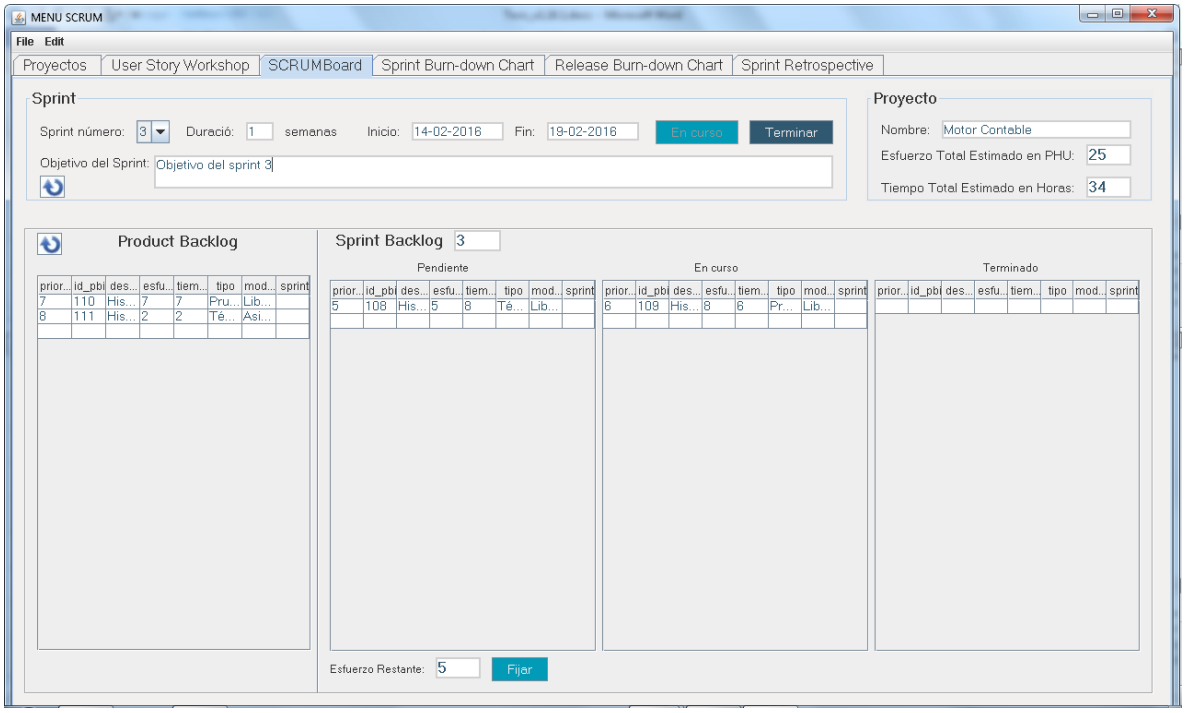


Figura 57. Estado del SCRUM Board durante un Sprint

Cuando tratamos de iniciar un nuevo Sprint sin haber terminado el Sprint anterior, el sistema muestra un mensaje de error, Figura 58.

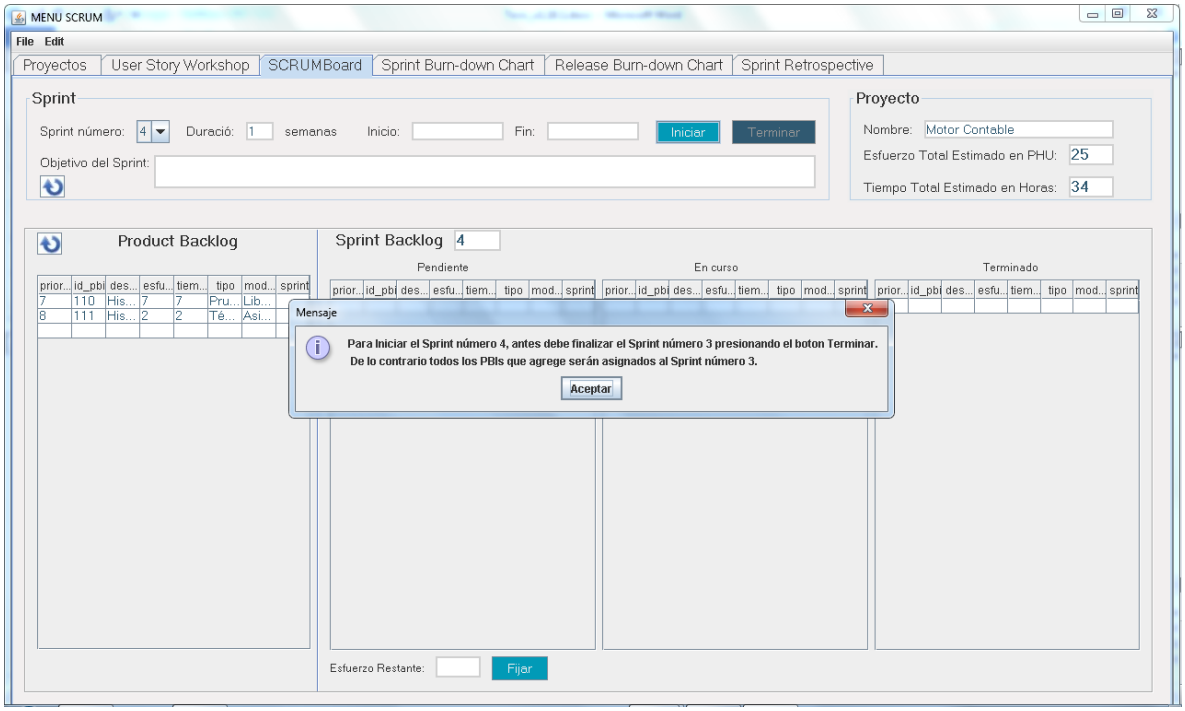


Figura 58. Error al intentar iniciar un nuevo Sprint sin haber terminado el anterior

Como parte de la filosofía de SCRUM, los cambios son bienvenidos, por lo tanto al agregar nuevos PBIs a la pila del Sprint o Sprint Backlog, el sistema muestra un mensaje indicando las afectaciones que se tendrán al agregar nuevos PBIs, Figura 59.

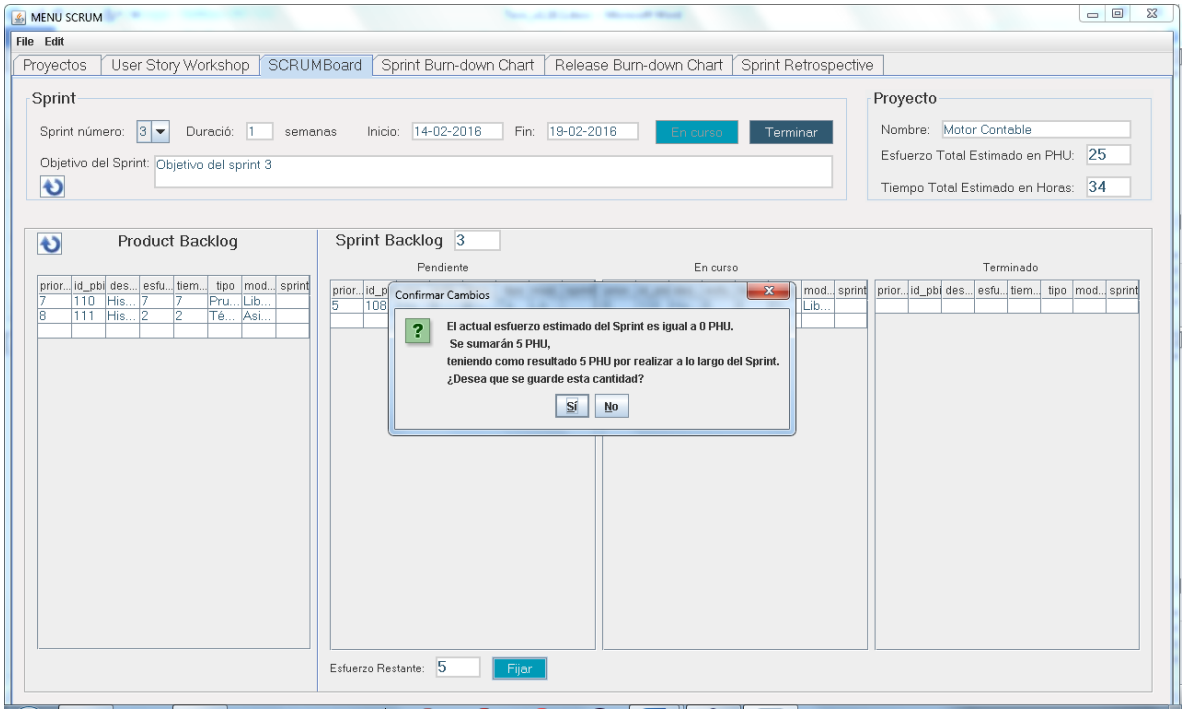


Figura 59. Agregar nuevos PBIs al Sprint Backlog

Para visualizar la información de cada PBI y modificarla, seleccionar en el SCRUM Board un PBI, Figura 60. El cuadro que se muestra permite modificar toda la información del PBI o únicamente modificar lo relacionado a las tareas.

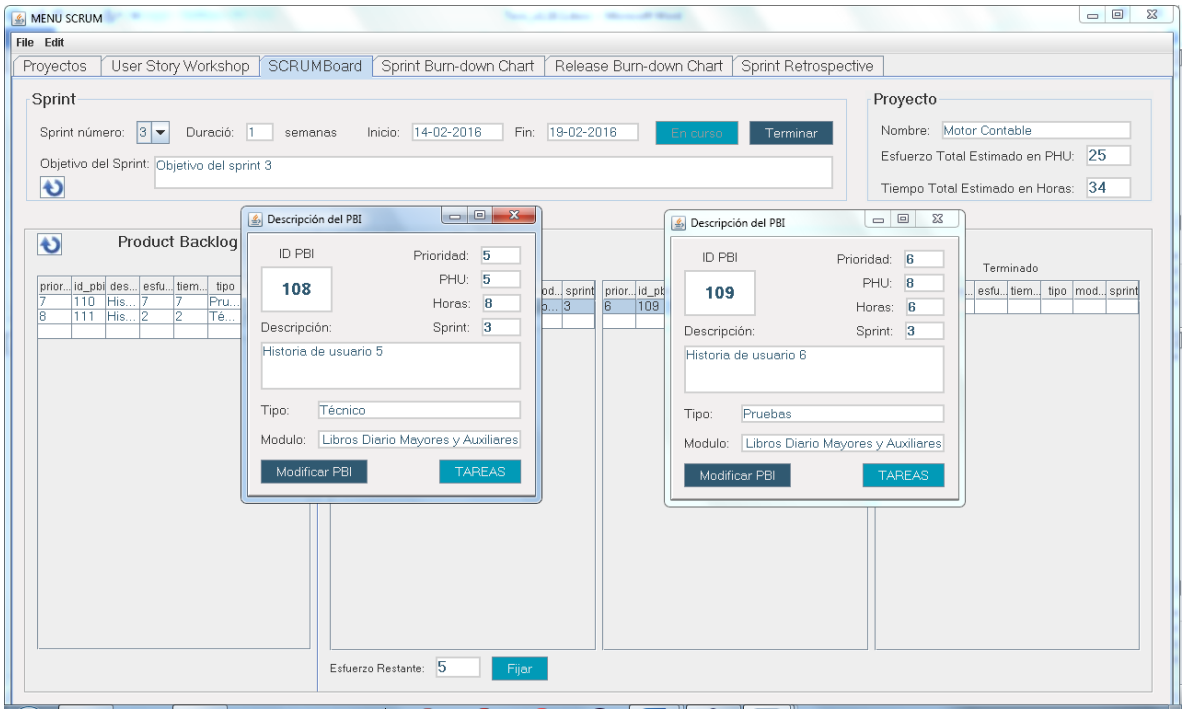


Figura 60. Mostrar detalles de cada PBI

Pestaña Sprint Burn-down Chart

Esta opción permite graficar el trabajo restante por realizar en el Sprint en curso. Esta grafica se actualiza diariamente por el SCRUM Master. Para actualizarla presionar el botón **"Actualizar Gráfica"**, seleccionar el día del Sprint e ingresar el trabajo pendiente por realizar, Figura 61.

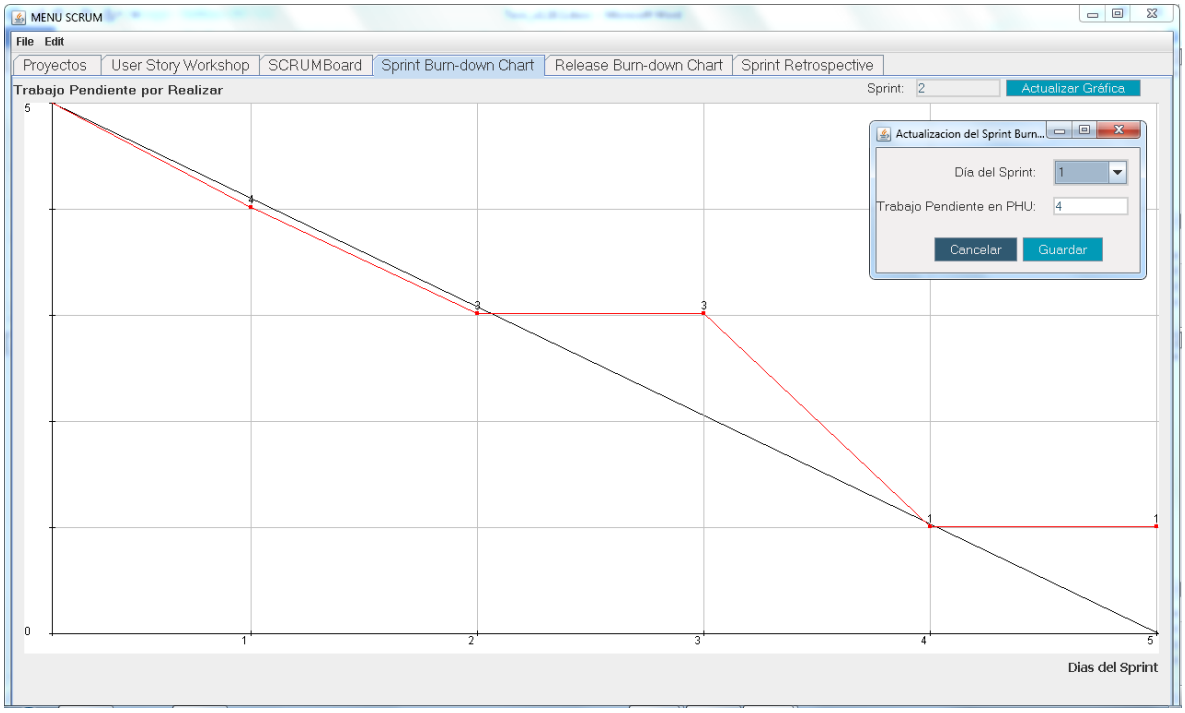


Figura 61. Gráfica Sprint Burn-down

Pestaña Release Burn-down Chart

Esta opción permite graficar el trabajo restante por realizar en todo el proyecto. Esta grafica se actualiza automáticamente, a medida que los Sprints van transcurriendo. La grafica permite visualizar el estado del proyecto y si se ha atrasado o no, Figura 62.

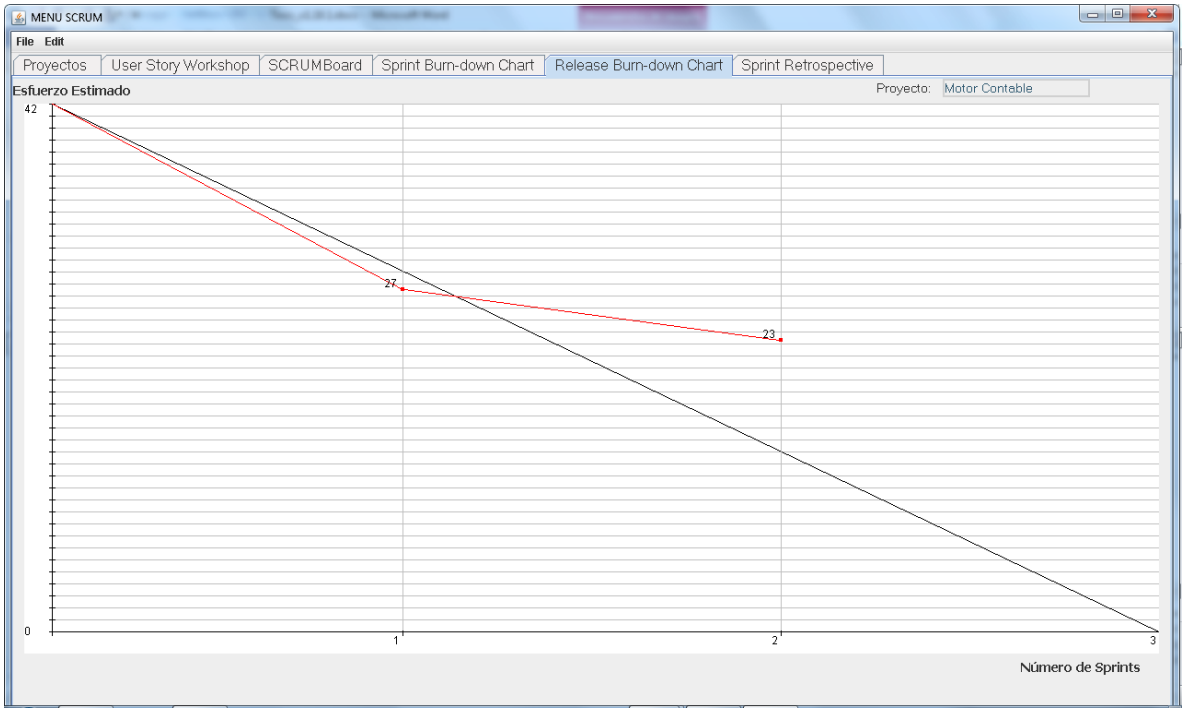


Figura 62. Gráfica Release Burn-down

Pestaña Sprint Retrospective

Una de las reuniones importantes en SCRUM es el Sprint Retrospecitve, la cual se realiza al finalizar cada Sprint, en esta reunión el equipo se reúne junto con el SCRUM Master y responden tres preguntas relacionadas al Sprint, ¿qué se hizo bien?, ¿qué se hizo mal? y ¿qué se puede mejorar para el siguiente Sprint? Esta opción permite llevar una bitácora sobre las cosas que se hicieron bien y mal durante cada Sprint así como también cuales son los puntos a mejorar que fueron identificados por los miembros del equipo para que la calidad del trabajo mejore, Figura 63.

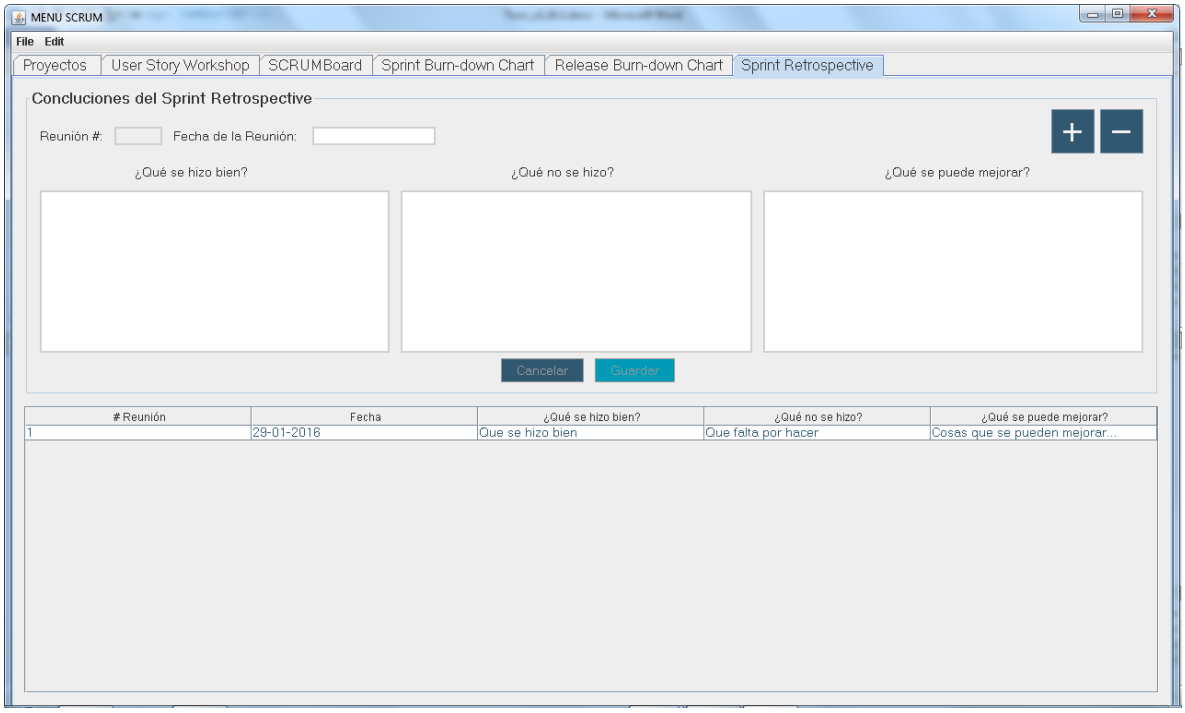


Figura 63. Bitácora del Sprint Retrospective

Bibliografía

Peñaloza Báez, M. (Enero de 2002). *LA INDUSTRIA DEL SOFTWARE,UNA OPORTUNIDAD PARA MÉXICO.* Obtenido de Enterate en línea:Internet, Cómputo y Telecomunicaciones: http://www.enterate.unam.mx/Articulos/2002/enero/software.htm

(Octubre de 2013). Obtenido de El modelo COCOMO: http://www.sc.ehu.es/jiwdocoj/mmis/cocomo.htm

(Octubre de 2013). Obtenido de Scrum Bok: http://www.scrummanager.net

(Octubre de 2013). Obtenido de Avantare: http://www.avantare.com

Abran, A., & Moore, J. W. (2004). *Guide to the software engineering body of knowledge.* The Institute of Electrical and Electronics Engineers, Inc.

Aguilar, J., Sanchez, M. H., Fernandez, C., Rocha, E., Martínez, D., & Figueroa, J. (2014). Software Projects' Size Developed by Mexican Companies. *The 2014 International Conference on Software Engineering Research and Practice*, (pp. 1-7). Las Vegas, Nevada, USA.

Álvarez Sáez, J. (Octubre de 2013). Obtenido de Modelos de Estimación en Proyectos Software: Una Aportación a las Técnicas de Estimación por Analogía Basada en Sistemas de Recomendación con Información Difusa: http://wwwdi.ujaen.es/?q=es/node/348

Ambler, S. W. (2012, junio). Retrieved from Dr.Dobb´s: the world of software development : http://www.drdobbs.com/architecture-and-design/disciplined-agile-change-management/240001474

Ambler, S. W. (2013, Noviembre). Retrieved from Agility at Scale Survey:Results from the Summer 2012 DDJ State of the IT Union Survey: http://www.ambysoft.com/surveys/stateOfITUnion201209.html

Beck , K. (1999). *Extreme Programming Explained: embrace change.* Addison-Wesley.

Boehm, B. (2000). *COCOMO II Model Definition Manual.* Manual, Center for Software Engineering, USC.

Boehm, B. W. (1983). *Software Engineering Economics.* TRW Defense Systems Group, Redondo Beach.

Cohn, M. (2006). *Agile Estimating and Planning.* Prentice Hall.

Consejo Mexicano para el Desarrollo Economico y Social, A. (Septiembre de 2013). Obtenido de Observatorio PYME: http://www.observatoriopyme.org/

Díaz, E. (Octubre de 2013). Obtenido de Manejar su tiempo: http://manejarsutiempo.blogspot.mx

Faireyl, R. E. (2009). *MANAGING AND LEADING SOFTWARE PROJECTS.* Hoboken , New Jersey: John Wiley & Sons, Inc.

Fenton, N. E., & Pfleeger, S. L. (1998). *Software Metrics: A Rigorous and Practical Approach* (2nd ed.). PWS Publishing.

Fermín, J. F. (2006). ESTIMACIÓN DE SOFTWARE POR DESAGREGACIÓN. *X CONGRESO INTERNACIONAL DE INGENIERIA DE PROYECTOS.* Valencia.

Fischman, L., McRitchie, K., & Galor, D. D. (2005, Abril). Inside SEER-SEM. *CROSSTALK The Journal of Defense Software Engineering* , 26-28.

Garmus, D., & Herron, D. (2001). *Function point analysis: measurement practices for successful software projects.* Addison Wesley.

Gómez, J. (Octubre de 2013). Obtenido de El laboratorio de las TI: http://www.laboratorioti.com

Group, D. C. (Octubre de 2012). Obtenido de David Consulting Group: http://www.davidconsultinggroup.com/

Group, T. S. (1995). *Chaos Report* . The Standish Group.

Group, T. S. (2009). *Chaos Summary* . The Standish Group International, Inc.

Group, T. S. (2010). *Chaos Summary.* The Standish Group International, Inc.

Group, T. S. (2011). *Chaos Manifesto.* The Standish Group International, Inc.

Group, T. S. (2012). *Chaos Manifesto.* The Standish Group International, Inc.

Group, T. S. (2013). *CHAOS MANIFESTO* . The Standish Group.

Humphrey, W. S. (1995). *A Discipline for Software Enginnering.* Addison Wesley.

Humphrey, W. S. (2005). *PSP : a self-irnprovernent process for software engineers* . Pearson Education.

Inc., Q. S. (2013, Noviembre). Retrieved from QSM: http://www.qsm.com/resources/function-point-languages-table

Ismaeel, H. R., & Jamil, A. S. (2007). Software Engineering Cost Estimation Using COCOMO II Model. 1-26.

Issa , A., Odeh , M., & Coward , D. (2006). Software Cost Estimation Using Use-Case Models: a Critical Evaluation. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, *27*, 66-71.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering – A systematic. *Information and Software Technology*, *51*, 7-15.

Kitchenham, B., Charters, S., Budgen , D., Brereton, P., Turner , M., Linkman , S., . . . Visaggio, G. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering.* EBSE Technical Report, Software Engineering Group School of Computer Science and Mathematics , Keele, Staffs .

Landry, T. (Noviembre de 2013). Obtenido de klocwork: http://www.klocwork.com/blog/agile-development/agile-adoption-an-update/

Leffingwell, D. (2011). *Agile software requirements : lean requirements practices for teams, programs, and the enterprise.* Pearson Education.

Longstreet, D. (August 2002). *Function Point Analysis: Training Course.* Longstreet Consulting Inc.

Marchewka, J. T. (2013). *Information Technology Project Management: Providing Measurable Organizational Value.* John Wiley & Sons, Inc.

Mendes, E. (2008). *Estimation techniques for web projects.* IGI PUBLISHING.

Munive, E., & Trejo, R. (2003). A Methodology for Self-Diagnosis for Software Quality Assurance in Small and Medium-Sized Industries in Latin America. *The Electronic Journal on Information Systems in Developing Countries*, 2.

Oktaba, H., & Piattini, M. (2008). *Software process improvement for small and medium enterprises: techniques and case studies.* IGI Global.

Palacio, J., & Ruata, C. (2009). *Scrum Manager: Proyectos – apuntes de formación.* Safe Creative.

Parthasarathy, M. A. (2007). *Practical software estimation: function point methods for insourced and outsourced projects.* Addison Wesley.

Piattini Velthuis, M. G., & Calvo-Manzano Villalón, J. (2004). *Análisis y diseño de aplicaciones informáticas de gestión: una perspectiva.* México, D.F.: Alfaomega.

Pressman, R. (2001). *Software engineering: a practitioner´s approach* (5th ed. ed.). McGraw-Hill.

Pressman, R. (2001). *Software engineering: a practitioner´s approach* (5th ed. ed.). McGraw-Hill.

Sánchez López, M. H., & et al. (2012). On the need for optimization of the software development processes in short-term. *Anie, 8*, p. 2.

Sommerville, I. (2007). *Software Engineering.* Pearson Education.

Wikipedia. (Enero de 2014). Obtenido de Puntos de caso de uso-Wikipedia: http://es.wikipedia.org/wiki/Puntos_de_caso_de_uso