UNIVERSIDAD POLITÉCNICA DE TULANCINGO

U

P

T

“Identificación de variables para la obtención de un modelo de estimación de costos para coadyuvar a equipos de desarrolladores novatos”

Por:

Ing. Carlos Alberto Marquez Sosa.

Tesis

Maestría en Ingeniería con Especialidad en Ingeniería de Software

Asesores:

Dr. Carlos Enríquez Ramírez.

Tulancingo de Bravo, Hidalgo

Dedicatoria

Dedico este trabajo a mi amada esposa Josefina, por su apoyo y ánimo que me brinda día con día para alcanzar nuevas metas, tanto profesionales como personales.

A mis adorados hijos Leah Ferananda, Carlos Jael, a quienes siempre cuidaré para verlos hechos personas capaces y que puedan valerse por sí mismos.

A mi mamá Blanca, quien es mi guía y que siempre a creído en mi desde mi infancia.

A mi hermano Luis, que siempre ha sido mi compañero y amigo.

A mis compañeros de trabajo, a quienes agradezco el apoyo y regaños por aprovechar un poco de tiempo del trabajo para elaborar la tesis.

Agradecimientos

Agradezco a todos los que estuvieron presentes en el trascurso de la realización de este gran proyecto.

Agradezco a todos los integrantes de mi familia, que ha sido el motor que me impulsa para alcanzar todos los objetivos.

Agradezco al Dr. Carlos Enríquez Ramírez por su guía para realizar mi trabajo con calidad y veracidad. Por su apoyo en todo momento.

A la Universidad Politécnica de Tulancingo por ser la casa de estudios que me permitió realizar mis estudios de nivel posgrado. Así como a los maestros que me impartieron catedra y colaboraron en mi formación académica.

Índice

Índice de figuras

Índice de tablas

1 Propósito y Organización

Debido a la gran dinámica de hoy en día en el uso de los modelos de desarrollo de software, se cuenta con diversas formas de llevar la gestión de los proyectos de software, con ello se contempla que en los espacios como son las universidades se prepare a los estudiantes en la planeación económica de dicho desarrollo.
Con la finalidad de que los costos no excedan o en el peor de los escenarios el desarrollador pierda en la contratación de los proyectos, se debe contemplar un "ganar-ganar". Se propone adoptar un método para la estimación de costos en el desarrollo software para pequeños equipos de desarrolladores principiantes, es decir, aquellas personas que se contratan por primera vez en el desarrollo de software como actividad de generación de ganancias.

Una de las tareas más importantes en la gestión de proyectos de software consiste en la estimación del esfuerzo, costo y duración que demandará el mismo. La falta de precisión en la estimación puede ocasionar en las empresas de software incumplimiento de plazos, entrega de productos incompletos, incremento del precio final y pérdida de competitividad. Las pequeñas y medianas empresas de software que inician proyectos de mejora de sus procesos, se enfrentan con el desafío de seleccionar, por un lado, el método de estimación más apropiado a su contexto y, por otro, adoptar mejores prácticas que permitan ir consolidando una base histórica de estimación con el propósito de disminuir cada vez más la desviación entre los valores estimados y reales.

En el primer capítulo se encuentra el fundamento teórico de la propuesta del trabajo de tesis, en la sección dos, se hace revisión de los aspectos que se manejan como elementos a tratar en la implementación de los conceptos en la tesis. En el apartado tres se realiza una descripción metodológica de cómo se lleva a cabo la adopción de los procesos para la estimación de costos en un desarrollo de software dentro de un entorno real y cuáles datos son obtenidos y que servirán para dar un referente de cómo adoptar el modelo propuesto en este trabajo.

1.1 Introducción.

El desarrollo de software se ha convertido en una actividad importante para muchas organizaciones modernas. De hecho, la calidad, el costo, y oportunidad de los programas desarrollados a menudo son críticos y determinantes del éxito de una organización. Hay significativas implicaciones financieras y estratégicas para el desarrollo de proyectos en términos de programación y la estimación de costos.

La estimación de costos de software es una de las tareas más importantes en la gestión de proyectos de software. Los costos de desarrollo tienden a aumentar con la complejidad del proyecto. Si bien, es una de las primeras tareas, posterior a la colocación de requerimientos que se elabora a medida que el proyecto avanza para obtener una mayor precisión de la estimación. Una estimación es una predicción de cuánto tiempo durará o costará un proyecto, constituye la base para la planificación de los proyectos. El desarrollo del software requiere de la estimación para controlar y administrar los recursos que se necesitan utilizar antes y durante el proyecto. Son numerosas las variables, relacionadas con los recursos humanos, el contexto y las políticas que intervienen en el proceso de desarrollo, que pueden afectar los resultados finales.

Para las empresas de software una estimación deficiente del esfuerzo y duración que conlleva un proyecto puede ocasionar incumplimiento de plazos en la entrega de productos.

Un problema importante de la estimación de costos de software es la obtención de atributos precisos de interés en el desarrollo de software. Sin embargo, la actividad más investigada ha sido la estimación de esfuerzo, ya que establece la mayor parte de beneficio para la gestión de proyectos. Las estimaciones de esfuerzo de software son difíciles para la estimación de la cantidad de mano de obra necesaria para el proyecto. Esta estimación determina la asignación de personal y programación para un proyecto de software. Dado que el esfuerzo humano es de los principales impulsores de costos en un proyecto de software, la apreciación del esfuerzo determina el presupuesto del proyecto calculado por diversos modelos de estimación que se enfocan a un conjunto de medidas que describen los procesos

de desarrollo de software, productos y recursos como experiencia del desarrollador, tamaño y complejidad del sistema y las características del entorno de desarrollo.

Hay muchos modelos y herramientas que se utilizan en la fase de costo-estimación de software que proporcionan información muy valiosa sobre los esfuerzos y gastos para la gestión de una oferta o propuesta económica para un proyecto. Los métodos más utilizados para predecir el esfuerzo de desarrollo de software se han basado en lineal por mínimos cuadrados de regresión como COCOMO, un modelo de costo constructivo desarrollado por Barry W. Boehn en 1984, enfocado en tres variantes, básico, intermedio y detallado. El proceso se basa en el tamaño del proyecto, donde se representa por líneas de código (LOC) o de miles de líneas de código (KLOC), basándose en ecuaciones matemáticas para estimar el esfuerzo, el tiempo de desarrollo y el esfuerzo del mantenimiento, definiendo como ecuación matemática el modelo presentado a continuación.

$$\mathbf{E = A \times (KLOC)^B}$$

COCOMO

La estimación de costos tiene dos usos, durante la etapa de planteamiento y la etapa de control de procesos del proyecto; el primero permite decir el recurso humano necesario para llevar a cabo el proyecto y establecer un cronograma, el segundo nos permite evaluar si el proyecto va evolucionando de acuerdo con el cronograma y tomar decisiones correctivas si en su caso fueran necesarias.

Durante muchos años, la industria y las comunidades de investigación han estudiado la mejora de procesos de software, dado que en las grandes organizaciones que dependen de la tecnología de información es de gran importancia determinar la productividad, calidad y tiempo de desarrollo. Como es derivado, muchos invierten tiempo y recursos de mejoramiento de sus productos de software, adaptando modelos o marcos de referencia, como los son CMMI, ISO/IEC entre otras, sin embargo, estos modelos no son fácilmente usados por

pequeñas y muy pequeñas organizaciones de desarrollo de software, ya que dichos modelos son demasiado complicados y caros para poner en práctica.

La transición de ser un desarrollador novato a un experto requiere de una instrucción especializada, un soporte eficaz y años de experiencia (Galván, 2017). Existen varios estudios orientados al trabajo con desarrolladores novatos, como se detallará en el estado del arte, dónde se da un seguimiento desde diferentes puntos como son: codificación, depuración de código, uso de métricas de software, la colaboración en equipos y el uso de procesos. Sin embargo, la resistencia a la adopción de procesos en los principiantes es frecuente, debiéndose a falta de conocimientos en situaciones reales de desarrollo de software, así como la inexperiencia. El seguimiento de un conjunto de procesos en ocasiones es un tanto desorganizado, esto se debe a la dificultad de encontrar un punto de partida en el desarrollo, la falta de experiencia en el área de ingeniería de requerimientos, el diseño de sistemas, la interpretación de un lenguaje de modelado, la codificación en algún lenguaje de computación y en saber integrar los conceptos vertidos en ellos a lo largo de la formación académica.

Los desarrolladores novatos deben conocer los métodos y herramientas que le permitan agilizar sus trabajos de desarrollo de software. Las metodologías ágiles y tradicionales permiten realizar estimación de costos, entregas en tiempo y forma de manera iterativa e incremental, por lo que es conveniente tener en cuenta las tecnologías propias para este tipo de filosofías de desarrollo.

La adopción de procesos para el desarrollo de software ha sido tema que varias investigaciones en el área de la ingeniería de software, como se podrá revisar en los capítulos más adelante. Un aspecto a considerar como importante es el seguimiento de los procesos en el desarrollo de software, lo que proporciona certeza de las actividades que se están realizando, desde el cumplimiento de los requerimientos hasta su aceptación con el cliente.

Existen varias, desde nacionales e internacionales, de alternativas propuestas en el ambiente del desarrollo de software que se refieren al seguimiento de procesos, éstas se enfocan principalmente a las empresas ya sean medianas o grandes, dado

que existen ciertos acercamientos pero que atacan de manera individual las fases del mismo, tal como pueden ser los procesos de desarrollo particulares propuestos por Watts S. Humphrey.

1.1 Objetivo de Tesis

Identificar variables de los proyectos de desarrollo de software basado en modelos previamente reportados y utilizados en la industria de software que permitan determinar un modelo para la estimación de costos en pequeños equipos de desarrollo de software y que se ajusten al caso de uso: desarrolladores novatos de la Universidad Politécnica de Tulancingo.

1.2 Objetivos Específicos

Identificar las variables que influyen en la estimación de costos de desarrollo y que permiten la mayor precisión posible al costo real asociado a la construcción de un determinado desarrollo de software.

Relacionar las variables que influyen en la estimación de costos de desarrollo de un proyecto de software mediante la propuesta de un modelo de estimación de costos para pequeños equipos de desarrolladores novatos.

1.3 Problemática

La transformación del desarrollador de software principiante al experto difiere en aspectos donde se tiene que conjuntar todo el conocimiento y experiencia en el desarrollo de software para lograr eficazmente la especialización.

Muchos de los egresados de la ingeniería en sistemas computacionales carecen de un proceso inicial de desarrollo de aplicaciones básicas, normalmente tienden a desarrollos pequeños, con limitaciones económicas de equipo físico y humano. En ocasiones la falta de experiencia se ve reflejada en el impacto económico, es decir, no cuentan con los elementos para realizar un cobro de su trabajo de manera independiente y establecer un acuerdo oficial que ampare su trabajo. Por pequeños grupos de desarrollo de software se hace referencia de tres y hasta quince desarrolladores, muchos de los cuales llevan a cabo diversos roles en el proyecto, donde se pretende que cada uno de los integrantes cumpla con al menos uno de los procesos (Schwaber & Beedle, 2002).

Una de las complejidades para los desarrolladores principiantes es como determinar el costo del primer desarrollo de software, así como también coordinar

roles y tareas a cada integrante del equipo. Prever los costos y tiempos de entrega se lleva a cabo mediante la experiencia de trabajos previos, o mediante un buen diseño arquitectónico donde se muestren los componentes, relaciones y acciones que tendrán cada uno de ellos.

1.4 Justificación

Actualmente se tiene que los pequeños equipos desarrolladores principiantes desconocen los puntos de inicio de los procesos de desarrollo de software, así como de igual manera excluyen el costo de estimación. Además de carecer de la cultura y disciplina del seguimiento de procesos lo que origina desarrollos desorganizados y productos de baja calidad. De esta manera se deteriora el esfuerzo al ver el poco impacto de sus desarrollos. El no conocer los tiempos, esfuerzos, el no hacer uso de nuevas tecnologías y capacidades que pueden desarrollar afectan el proceso de estimación de costo del producto. Por lo que se busca una serie de procesos sistemáticos que permita guiar, medir, estimar los desarrollos de software a los integrantes que inicien esta actividad.

El hacer uso de una metodología ágil les permite a los integrantes de un desarrollo de software tener ciertas ventajas en sus actividades tales como la simplificación de sobre carga de procesos, calidad y mejora, una evaluación de riesgos y un mejor perfil de productividad en el desarrollo, esto se enfoca en un punto de flexibilidad durante todo el planteamiento del proyecto. Una de las estimaciones precisamente se basa en la búsqueda de un experto que basado en su experiencia determine cuánto puede demorar y el costo del sistema que se pretende desarrollar. Sin embargo, esta estimación no es la más adecuada debido a las imprecisiones de la experiencia humana o por no contar con un experto para realizarla. Una de las opciones está en las estimaciones tempranas, como es el caso de estimaciones por puntos de función, que permiten obtener estos valores a partir del levantamiento de requisitos. Esta estimación adolece de una dificultad, al hacerse en estudios tempranos del proyecto puede tener resultados no muy exactos. Otra de las estimaciones más utilizadas es COCOMO II como modelo algorítmico en el cual se pueden realizar estos cálculos en tres niveles. Estos modelos pueden combinarse para ofrecer estimaciones más exactas en la medida que el proyecto avanza y se necesiten realizar ajustes. También son necesarios en procesos iterativos e

incrementales para cada una de las iteraciones que se van a ejecutar en el proyecto. Una de las problemáticas que tienen estos modelos de estimación es la gran cantidad de cálculos que deben realizar los estudiantes y la múltiple cantidad de criterios que deben ser tomados en cuenta. Otro de los aspectos que hace ser poco interesante para ellos se encuentra en la pobre comprensión de la importancia de la estimación de proyectos a partir de la tendencia a la ejecución detectada en ellos.

2 Marco Teórico

2.1 Proceso Unificado de Educación (UPEDU)

UPEDU es un conjunto habilitado para la web, orientado para agilizar las actividades de desarrollo de software en equipo, con el enfoque de asignación de tareas y responsabilidades dentro de una organización de desarrollo, su objetivo es asegurar la producción de software de alta calidad, que satisfaga las necesidades de los usuarios finales dentro de un tiempo predecible y propuesto (Pierre, 2014).

El objetivo de UPEDU es definir un proceso de software que se adapte al tamaño del proyecto en orden para facilitar su aceptación y mejorar su utilidad. Solo las actividades y artefactos más relevantes permiten a los estudiantes a centrarse en los componentes esenciales del proceso de software. UPEDU se basa en el concepto de los procesos de software que constituyen en la colaboración entre roles, en las que las operaciones llaman actividades y se realizan en entidades físicas llamadas artefactos.

El proceso unificado para la educación (UPEDU) es un proceso basado en RUP diseñado para instruir los procesos de software en la ingeniería de software y en los programas informáticos. Este proceso se basa en actividades centradas con contenido idóneo significativo y un conjunto de artefactos esenciales que deben ser necesarios para proyectos pequeños. El proceso también define roles básicos tales como, “Diseñador”, “Implementador”, “Tester”, “Gestor de proyectos” y “Crítico” que son fácilmente comprensibles por los desarrolladores que no tienen experiencia. Los roles, actividades y artefactos se agrupan en disciplinas que se clasifican como en la ingeniería o en la gestión.

Fases e Hitos

Primera Fase (Inicial)

En la primera fase se tiene como objetivo lograr un conjunto de todos los integrantes a conformar el desarrollo de software durante el ciclo de vida del proyecto, con la finalidad de establecer un alcance, una visión de los criterios de aceptación de funcionamiento, logrando una muestra al menos de una

arquitectura candidata con algunos escenarios principales, así como también realizar una estimación del costo y definir un cronograma estimado del proyecto (Pierre, 2014).

Las actividades que se presentan en la primera fase para el proceso de desarrollo de software son:

- Formular el alcance del proyecto (contexto de Requerimientos).
- Establecer criterios de aceptación.
- Planeación del caso de negocio.
- Gestión de Riesgos.
- Definir un cronograma.
- Un prototipo de la arquitectura.
- Estimar un costo.

Dentro de las mismas actividades se determina la evaluación del proyecto y la organización y la selección de las herramientas a ser utilizadas.

Segunda Fase (Elaboración)

En la segunda fase se establece la arquitectura base del sistema para proporcionar el diseño y el esfuerzo de implementación, se desarrolla a partir de la consideración de los requerimientos más importantes (las que tienen mayor impacto en la arquitectura de software) y la evaluación de riesgo.
Los objetivos de la fase elaboración garantizan que la arquitectura, requerimientos y planes sean lo suficiente estables y los riesgos sean menores a ser capaz de determinar un costo predecible y un calendario, normalmente los puntos principales a reducir su riego son:

- Diseño / Requerimientos.
- Reutilización de Componentes.
- Viabilidad del producto o demostraciones para los clientes y usuarios finales.

Tercera Fase (Construcción)

En la fase de construcción la finalidad es alcanzar que todos los componentes y requerimientos deben ser implementados, integrados y probados obteniendo una versión aceptable del producto, donde se hace referencia a la gestión de los recursos y el control de las operaciones para optimizar los costos, horarios y calidad.

Los objetivos principales hacen referencia a la reducción al mínimo de los costos de desarrollo mediante la optimización de los recursos, para lograr una calidad, completar el análisis, diseño, desarrollo y pruebas de funcionalidad requerida.

Cuarta Fase (Transición)

En la fase de transición la finalidad es asegurar que el software esté disponible para los usuarios finales, por lo que se incluye las pruebas del producto en preparación para el lanzamiento, complementar la documentación, hacer pequeños ajustes en base a la retroalimentación de los usuarios, entrenar al usuario en el manejo del producto y en general tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto.

Fases de Planeación

Un ciclo de proceso inicial para un desarrollo de software mediano debe anticipar la siguiente distribución entre el esfuerzo y el tiempo, que puede ser representada gráficamente como se muestra en la *figura [1]:*

	Inicial	**Elaboración**	**Construcción**	**Transición**
Esfuerzo	5%	20%	65%	10%
Tiempo	10%	30%	50%	10%

Figura 1. Representativa de valores porcentaje de tiempo y esfuerzo de un ciclo de desarrollo de software

.

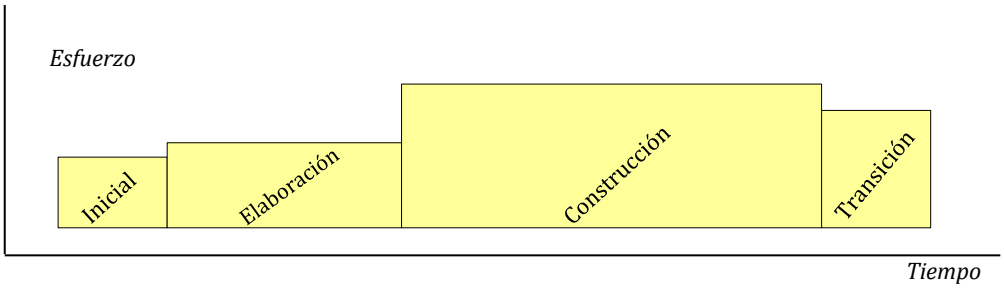

Figura 2. Representación gráfica de tiempo y esfuerzo en el desarrollo de software.

Disciplinas de Desarrollo de Software

Requerimientos

La finalidad al levantar requerimientos es que se puedan transformar en un sistema operacional, a través de la recopilación, análisis y verificación de las necesidades del cliente, de la misma también se determine una estimación del costo y tiempo que llevara el proceso de desarrollo de software (S. Pressman., 2002).

Requerimientos funcionales especifican acciones que un sistema debe ser capaz de realizar, sin tomar en consideración las limitaciones físicas. Estos describen un modelo de casos de uso. Por lo tanto, los requisitos funcionales especifican el comportamiento de entrada y salida de un sistema.

Requerimientos no funcionales, sólo describen atributos del sistema o atributos del entorno del sistema. Aunque algunos de estos pueden ser capturados en los casos de uso.

Existen diferentes tipos de necesidades o requerimientos. Una forma de clasificarlas es en base al modelo FURPS.

Funcionalidad *- (Functionality).* Especifican acciones de que un sistema debe ser capaz de realizar, sin tomar en consideración las limitaciones físicas.

Usabilidad *- (Usability)***.** Especifican los requerimientos de usabilidad que incluyen, factores humanos, estética, coherencia en la interfaz de usuario, documentación del usuario y materiales de capacitación.

Confiablidad *- (Reliability)***.** Consideran la gravedad de la insuficiencia, recuperación, precisión y tiempo medio entre fallas.

Rendimiento *- (Performarce)***.** Determina los parámetros de rendimiento de: velocidad, eficiencia, disponibilidad, rendimiento, tiempo de respuesta, tiempo de recuperación y uso de recursos.

Soporte *- (Supportability)***.** Requerimientos de compatibilidad que puedan incluir la capacidad de prueba, adaptabilidad, utilidad y configuración.
Requerimientos-Flujo de Trabajo

Cada detalle del flujo de trabajo representa una habilidad clave que debe ser aplicado para realizar la gestión de requerimientos eficaces. Analizar y comprender el problema que se enfocó durante el comienzo de la fase de un proyecto.

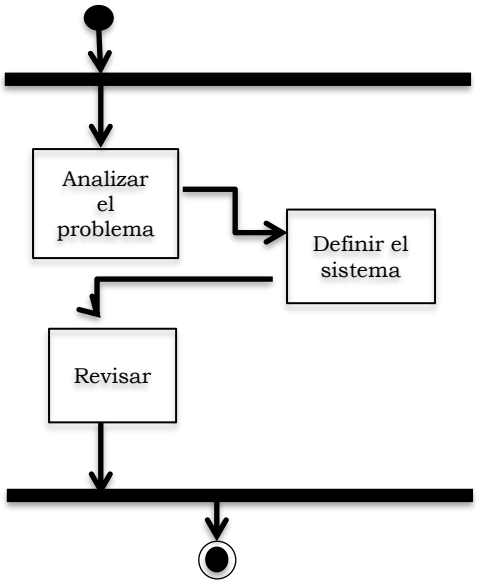

Figura 3. Representación de Flujo de Trabajo de Requerimientos.

Análisis y Diseño

El análisis y el diseño determinan y transforman los requerimientos en un diseño que se enfoca en una arquitectura firme y clara, de la misma manera se adapta al diseño para que ajuste con el entorno del sistema.

La visión de implementación es uno de los tres puntos de la arquitectura de un sistema, el propósito es capturar las decisiones del diseño realizadas para el desarrollo, estas son conformadas por:

- Una enumeración de todos los subsistemas en el modelo de implementación.
- Diagramas de componentes que ilustran cómo se organizan los subsistemas en capas jerárquicas.

Implementación

El enfoque implementación define la organización del plan de integración, la ejecución de componentes y al mismo tiempo realizar pruebas a los módulos desarrollados para llegar a tener un sistema integrado.

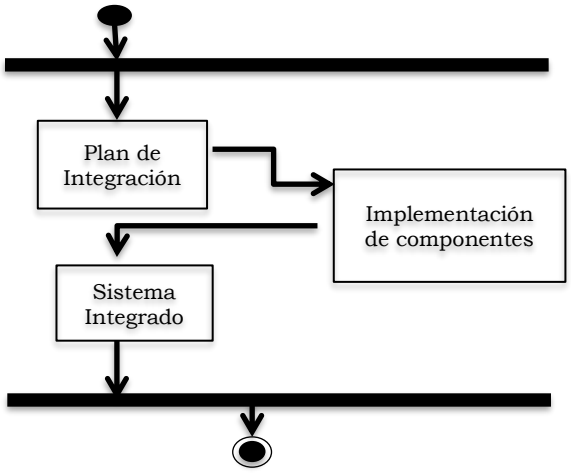

Figura 4. Implementación e integración de un sistema

Prueba

Se centra en la evaluación o valoración de la calidad del producto basando en los diferentes puntos:

- Encontrar y documentar errores en la calidad del software
- Demostración de la validez del diseño y las especificaciones de los requerimientos.
- Validación de las funciones del producto de software.
- Validar los requerimientos que sean implementado adecuadamente.

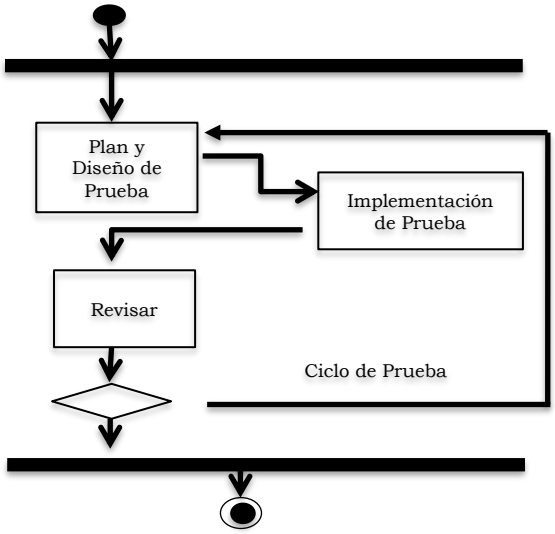

Figura 5. Flujo de pruebas para la calidad del producto.

Este flujo de Trabajo puede requerir variaciones en base a las necesidades específicas de cada proceso.

Gestión de la Configuración y cambio

La Configuración y administración de cambio de control (CM y CCM) implica la identificación de los elementos de configuración, restringir los cambios a los artículos, auditar cambios realizados sobre dichos elementos y definir y administrar las configuraciones de esos artículos. Cada método, procesos y

herramientas para proporcionar el cambio de configuración se puede considerar como un sistema CM.

Un sistema CM es esencial para controlar los numerosos artefactos producidos por muchos integrantes que colaboran en un mismo proyecto en común. Esto ayuda a evitar confusiones que pueden ser costosas, y asegura que el artefacto resultante no esté en conflicto.

Gestión de Proyectos

La gestión de proyectos equilibra los objetivos, administra el riesgo, y alcanza las restricciones del proyecto a entregar, que satisface las necesidades del cliente y usuario. Con la gestión del proyecto se logra una mejor entrega exitosa de un desarrollo de software donde su propósito consiste en:

- Administrar proyectos de software intensivos.
- Planear, dirigir personal, ejecutar acciones y supervisar proyectos.
- Administrar el riesgo.

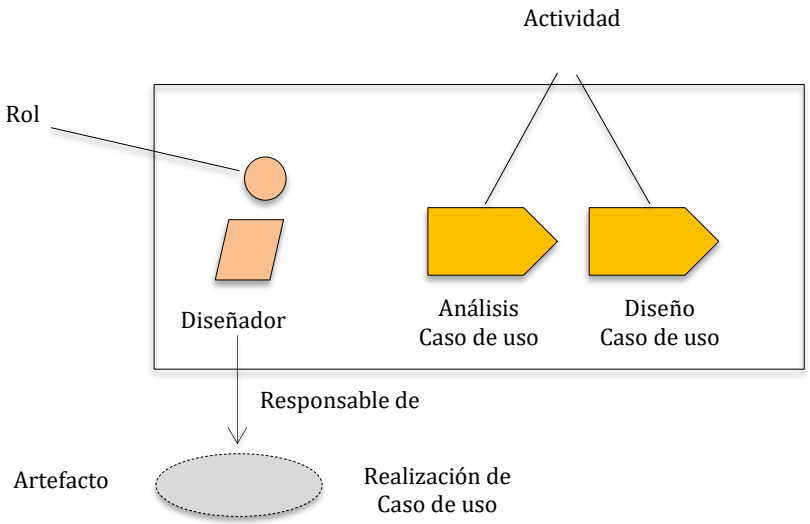

Figura 6. Relación entre roles, actividades, artefactos.

Roles y Actividades

En un proceso de desarrollo de software se define quién hace qué, como, y cuando, *Ian Sommerville* define cuatro roles que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los productos, que responden pregunta ¿Qué? y los flujos de trabajo que responden la pregunta ¿Cuándo?

En el desarrollo de software se llevan a cabo diferentes roles y actividades.

- Analista
- Diseñador
- Implementador
- Integrador
- Tester
- Jefe de control de cambios
- Jefe de Configuración
- Gerente del proyecto
- Crítico
- Stakeholder

2.2 Estado del Arte

Según Pressman (2002) indica que antes de que el proyecto pueda dar inicio, el equipo de desarrollo de software debe considerar el trabajo a realizar al igual que los recursos que se requerían, como el tiempo que transcurrirá de principio a fin. Una vez definidas dichas actividades, el equipo de desarrollo de software debe plantear un cronograma donde defina las tareas y roles de cada integrante, especificando entre tareas que puedan imponer una fuerte demora en el avance del desarrollo.

La planeación es un compromiso inicial donde se proporciona un marco conceptual que permita al gerente realizar estimaciones de recursos, costo y calendario. Además, se puede generar un grado de validación donde el equipo de software se apega a un plan de trabajo establecido, de modo conforme avanza el proyecto se debe adaptarse y actualizarse al proceso de la planificación de trabajo.

La estimación de costo y esfuerzo se apega a demasiadas variables (humanas, técnicas, ambientales, políticas) que pueden afectar el costo final del desarrollo. Sin embargo, la estimación del proyecto de software se puede manipular por una serie de pasos sistemáticos que determinen estimaciones con un índice de aceptabilidad, con base al riesgo que se pueda generar, al no definir las variables convenientes en la planeación del desarrollo. Par lograr una estimación que garantice un grado de 100 por ciento precisa se debe realizar conforme avanza el proyecto, desafortunadamente una estimación debe proporcionarse por anticipo, si bien otra opción sería basarse en proyectos similares en el cual se reflejen esfuerzos anteriores, corriendo el riesgo donde la experiencia pasada no siempre es un indicador de resultados futuros.

El uso de técnicas de descomposición y modelos empíricos para generar estimaciones de costo y esfuerzo son opciones de enfoques viables para la estimación del proyecto de software. Las técnicas de descomposición tienen un enfoque de “divide y vencerás”. Al fragmentar un proyecto en funciones y actividades la estimación y esfuerzo se puede realizar en forma escalonada.

Las habilidades para tomar una decisión en base al diseño de desarrollo de software se ha observado que en la educación es generalmente inadecuada, a la hora de instruir a los desarrolladores principiantes (Robillard, 1999), esto se refleja cuándo un desarrollador novato determina que es competente para desempeñarse en la industria y descarta que diversas de las habilidades que se llevan en los procesos de desarrollo de software se adquieren a través de la experiencia (Boulangera & Smith, 2001).

Existen estudios de expectativas de la industria que reflejan deficiencias en la preparación de los estudiantes de ingeniería de software (R. Wright, 2012), entre algunas como:

- La habilidad para entender sistemas de software como diferentes programas de un solo usuario.
- La habilidad para visualizar diferentes perspectivas o puntos de vista en sistemas de software.
- La habilidad para pensar de manera crítica y reflexiva.
- Análisis de sistemas y habilidades de diseño.
- Resolución de problemas y habilidades de investigación.
- La forma de enfrentar el primer contrato de desarrollo.
- Como determinar el costo y tiempo de un desarrollo

A medida que se hace uno más dependiente de los sistemas apoyados en computadoras, desarrolladores y diseñadores de software se deben adoptar nuevas habilidades y estrategias para determinar mejores decisiones que sean eficaces y que muestren resultados definidos. Las estrategias para la toma de decisiones comunes que se enseñan en ciencias de la computación e ingeniería de software empleadas por los desarrolladores de software principiantes y diseñadores, no son eficaces cuando se aplican a grandes problemas del mundo real (Carpersen & Kölling, New York, NY, USA, 2006). Los desarrolladores novatos tienden a pensar linealmente y concentrarse sólo en el problema en cuestión a utilizar estrategias de ensayo y error y la falta de confianza en sus decisiones de desarrollo (Ahmed, Wallace, & Blessing, 2003).

Mientras que estas habilidades pueden ser suficientes para la traducción de los diseños minuciosamente documentados, son inadecuados para el razonamiento sobre los grandes y complejos sistemas, donde el mundo moderno depende de que los estudiantes de diseño de software y de ingeniería necesitan apoyarse de herramientas para poder lograr sus desarrollos aplicando estrategias de diseño de expertos como referencia para madurar y convertirse en profesionales del diseño de software (Dewar & Astrachan, 2009).

Cuando se realizan estudios de aprendizaje de ingeniería de software en un aula de clases, los investigadores necesitan una estrategia o guía que les permita manejar un tiempo idóneo, para agilizar e implementar los procesos de desarrollo a los desarrolladores novatos y así, permitir que los resultados de enseñanza se puedan emplear en la industria del desarrollo de software (Carver, Shull, & Basilli, 2006).

Se manejan Técnicas que demuestran en el proceso de observación durante un desarrollo que un desarrollador novato, pueda incrementar su experiencia como los son:

- Dividir a los integrantes en parejas para satisfacer las limitaciones.
- Capacitar a los estudiantes en la formación de nuevas tecnologías.
- Un integrante de cada pareja aplica la tecnología mientras el otro observa. La persona que está aplicando la tecnología se refiere como el ejecutor, mientras que el sujeto que observa la aplicación de la tecnología es llamado observador.
- Cambio de papeles dentro de las parejas. Este paso permite que el sujeto que observo ahora realice la aplicación de la tecnología.
- Medición y análisis**.** Los resultados obtenidos de este paso deberían ser más precisos debido a que se ha abordado aún más la curva del aprendizaje.

El enfoque formulado anteriormente permite que los desarrolladores novatos adquieran experiencia por medio de la inspección (Carver, Shull, & Basilli, 2006).

Las empresas que se dedican al desarrollo de software comienzan a emprender iniciativas de mejora de los procesos de software (SPI) para alcanzar una madurez y obtener un grado de calidad más exacto en cada uno de sus métodos (Sulayman, 2010). La inversión a la mejora de procesos ha tenido varias ventajas comerciales es decir mejorar la calidad, reducir el tiempo de construcción, causar la mejor productividad, aumentar la flexibilidad organizativa y la satisfacción del cliente.

En años recientes las empresas de desarrollo de software pequeñas y medianas han surgido muy rápidamente y son generalmente desarrolladores en el dominio de desarrollo web (Tortorella & Visaggio, 1999). De tal forma los desarrolladores de software inician su preparación en dos fases, una es dónde comienzan a nivel superior y la segunda es cuando promueven su primer trabajo como profesional en la industria. Los informáticos una vez que finalicen su primera etapa deben alcanzar los conocimientos como programar, diseñar y probar el software, en la siguiente etapa en base a la experiencia que se va adquiriendo deben aprender a corregir, eliminar y crear código (Begel & Simon, 2014).

Un modelo de un sistema debe ser identificado, clasificado y se debe abstraer todos los elementos que constituyen un problema y poder organizarlos en una estructura formal. La abstracción es una de las prácticas que la mente humana se enfrenta con complejidad dándole a una solución comprensible y manejable. De esta manera un modelo de sistema provee un medio de comunicación y negociación entre todos los que conforman para su desarrollo (usuarios, analistas y desarrolladores).

Los estudios de las aplicaciones de la ciencia y la programación comenzaron en la década de 1970. Esta área se ha centrado en estudios empíricos sobre el comportamiento de las personas que programan. Los enfoques previos al estudio de desarrollo han formulado como una tarea de inteligencia orientada realizada por un solo programador, en raras ocasiones se ha considerado como un tema multidisciplinario en la dinámica de grupo. Además, la comprensión de una tarea muy inteligente resuelto por dos personas en un esfuerzo de colaboración también puede implicar problemas fundamentales de la informática moderna, ingeniería de software, el desarrollo y la psicología. Por ejemplo, la programación en par, lo que implica dos desarrolladores que colaboran como un solo individuo en la misma

tarea de programación, se ha demostrado por dos experimentos para ser productivo, y para producir código de mayor calidad (Nosek, 1998 Williams et al, 2000). Esto puede ser un caso en el que nuestro conocimiento del proceso de desarrollo de software puede ser beneficioso para otros campos como la ciencia cognitiva.

En el desarrollo de software utilizado métodos ágiles se encuentra en un proceso de crecimiento debido a la asociación de estas metodologías, además de la adaptación flexible que ha tenido para equipos pequeños. Sin embargo, estas metodologías presentan debilidades en la estimación y gestión de costos de desarrollo. La administración de proyectos de software es una de las partes fundamentales para obtener resultados exitosos según Jones (2004), considerando un proyecto exitoso como el que termina dentro del tiempo, costo y calidad esperada.

En la administración de los métodos ágiles el paso a utilizar para evaluar el rendimiento de un equipo es conocer la velocidad en que se desarrollan los elementos de registro del producto (Prudoct Backlog Items) (Yap, 2006), de igual manera el esfuerzo total es el cálculo del tamaño del proyecto. Respeto a lo anterior, las medidas más comunes para determinar administración de un proyecto en los métodos ágiles son el esfuerzo y el tamaño, sin embargo, esto enfoca a pérdidas económicas por falta de medidas de costos.
El problema con los costos en los métodos ágiles es la falta de una administración y una monitorización que sean efectivas, además de carecer de evidencias y estimaciones de costos, afectan y ocasionan un descontrol con los administradores de proyectos y expertos en métodos ágiles.

La estimación de costos en el desarrollo de proyectos de software es uno de las tareas más cruciales para la administración (Keung, Jeffery, & Kitchenham, 2004). Los factores que son típicos para la estimación de los desarrollos de software son: costo, tamaño, agenda, recursos humanos, calidad, esfuerzo, costo de mantenimiento y la complejidad. Dentro de las técnicas de estimación de costos o modelos que utilizan la colección de datos de proyectos pasados. Estos usualmente requieren factores tales como el tamaño del sistema como entradas para el

funcionamiento del modelo. Por ejemplo, las principales técnicas basadas en este modelo son COCOMO, SLIM, RCA PRICE-S, SEER-SEM and ESTIMACS. La mayoría de las técnicas de estimación de costos y agendas pueden ser agrupados en modelos de clasificación basados en regresión, modelos orientados en aprendizaje, acercamiento basado en el juicio de expertos y finalmente métodos de composición Bayesiana (Conboy, 2006).

Modelos diversos de estimación.

Muchos de los modelos de estimación que están disponibles son técnicas de regresión. Modelos de regresión que tienen fundamentos matemáticos que se basan en la colección de datos de proyectos anteriores. Los modelos orientados en el aprendizaje que se basan en el entrenamiento de diversas experiencias de desarrollo. Otro de ellos es el más usado que es la comparación de proyectos iguales realizados en el pasado, este tipo de estimación es el más recurrido debido a que implica un bajo costo de implementación debido a que solamente se hace la analogía de proyectos realizados en el pasado y se usa la información como una guía de estimación para el proyecto propuesto.

Por ejemplo, la técnica Delphi y el *Work Breakdown Structure* (WBS) caen en este tipo de mecanismos de estimación. Dentro de los modelos se tiene el de (Khalid & Yeoh, 2017) quienes sugieren un modelo difuso para estimar el costo de la eliminación de defectos en las primeras etapas de la planificación de defectos en las primeras etapas de la planificación del proyecto en términos del tamaño funcional del proyecto y la experiencia del equipo de desarrollo. El modelo sugerido en esta investigación mejora la precisión en la estimación del costo del proyecto de software. El modelo propuesto tiene como objetivo estimar el costo de trabajo, en una etapa temprana del proyecto, como un porcentaje del costo de la fase de desarrollo de software. Este documento se centró en el tamaño del software y la experiencia del equipo como las principales variables independientes que afectan la cantidad de reprocesos. (Bhardwaj & Rana, 2015) hacer mención que el tamaño del software es muy importante en estimar otros aspectos del proyecto de software como es el costo, planeación y trabajo. Para determinar el tamaño de un proyecto

de software se adoptó el esquema de la *International Software Benchmarking Standards Group (ISBSG).*

Tamaño relativo de software	*Tamaño de software en puntos de función*
Pequeño	*Menos de 100*
Medio	*De 100 a 999*
Largo	*De 1000 a 3999*
Muy Largo	*Más de 4000*

Tabla 1. Clasificación ISBSG para el tamaño de software

Por otro lado, se espera que, a medida que aumente la experiencia del equipo, aumente la calidad de los entregables, lo que resultará en una reducción de la cantidad de reprocesos. Siguiendo la clasificación ISBSG, los niveles de la experiencia promedio se muestran en la Tabla 2.

Nivel de Experiencia	*Promedio de experiencia en años*
Bajo	*Menos de 1*
Medio	*De 1 a 3*
Alto	*De 3 a 9*
Muy Alto	*Más de 9*

Tabla 2. Clasificación ISBSG para el tamaño de software

La estimación del esfuerzo de desarrollo de software juega un papel vital en el éxito de la gestión de proyectos de software. Tiene una influencia considerable en cuestiones de gestión, como planificación de proyectos, programación y ofertas. Una de las técnicas para la estimación es la Estimación Basada en Analogía (ABE), como método de estimación ampliamente aceptado, sufre mucho del problema de los conjuntos de datos incoherentes y no normales porque es un método basado en la comparación y la calidad de las mismas y depende en gran medida de la consistencia de proyectos. Para superar este problema, estudios previos sugirieron el uso de métodos de ponderación, técnicas de eliminación de datos atípicos y

varios tipos de métodos de *soft computing*. Dado que una estimación de esfuerzo incorrecta puede conducir a resultados imprevistos en proyectos de software, los investigadores han intentado proponer métodos de estimación precisos y confiables en este campo. Los métodos sugeridos diseñados para estimar el esfuerzo de desarrollo de los proyectos de software se pueden dividir en seis grupos principales (Boehm, Abts, & Chulani, 2000) de la siguiente manera: Modelos paramétricos que incluyen a COCOMO (Boehm B. , 1980), SLIM (Putman, 1978), y SEER-SEM (Jensen, 1983); técnicas Delphi que contiene la evaluación de expertos (Dalkey & Helmer, 1963); Métodos de aprendizaje automático y estimación basada en analogías ABE (por sus siglas en inglés) (Shepperd & Schofield, 1997). Métodos basados en la regresión que incluyen regresión cuadrática mínima ordinaria, regresión por pasos y regresión múltiple (Costagliola, Ferrucci, Tortora, & Vitiello) también árbol de clasificación y regresión (Breiman, Friedman, & Olshen, 1984); modelos basados en dinámica (Madachy, 1994) y finalmente métodos compuestos (Vishal, 2010); (Reddy, 2011); (Benala, Dehuri, Satapathy, & Madhurakshara, 2012).

La principal idea detrás de ABE es la comparación de nuevos proyectos con proyectos completados en el pasado. De hecho, para estimar el esfuerzo, este método se basa en encontrar proyectos completos que sean similares al proyecto objetivo. La comparación de proyectos se realiza en función de los atributos del proyecto, como el tamaño, el tipo de desarrollo, el lenguaje de programación, el tamaño del equipo, la plataforma de desarrollo, etc. Debido a varios problemas, como la falta de información en las primeras etapas de un proyecto y la incertidumbre de los atributos del proyecto, la estimación del esfuerzo es un proceso complicado y difícil. Para producir estimaciones precisas, los modelos paramétricos requieren datos suficientes y precisos porque estiman el esfuerzo usando ecuaciones matemáticas y estadísticas. Por lo tanto, los modelos paramétricos no pueden superar la complejidad, la incertidumbre y la falta de información que existen en los proyectos de software.

Básicamente, ABE o razonamiento basado en casos (CBR), consta de cuatro componentes principales (Kolodner 1993):

(i) Conjunto de datos histórico
(ii) Función de similitud
(iii) Las reglas de recuperación asociadas
(iv) Función de solución

La estimación de ABE contiene los siguientes pasos:

1. Recopilación de datos de proyectos anteriores y producción de un conjunto de datos históricos.
2. Elegir los parámetros de medición adecuados, como puntos de función y líneas de código.
3. Recuperar proyectos anteriores y calcular las similitudes entre el proyecto objetivo y los proyectos anteriores.
4. Estimar el esfuerzo del proyecto objetivo.

El conjunto de datos históricos contiene aquellos proyectos completados en el pasado. Para estimar el esfuerzo, un nuevo proyecto se puede comparar con los proyectos que existen en el conjunto de datos históricos.
Las reglas de recuperación se definen para determinar el método de selección del proyecto del conjunto de datos históricos. Las reglas pueden aplicar limitaciones (por ejemplo, el número de proyectos seleccionados) al procedimiento de recuperación de proyectos. Debido al volumen de detalles, las funciones de similitud y solución se describen por separado en las siguientes secciones.

Las técnicas de optimización pueden ser útiles para ajustar los pesos de los atributos en la función de similitud ABE. El algoritmo genético, como el método de optimización más común, se ha utilizado para determinar los pesos de los atributos en el método ABE.
El enfoque bayesiano es un proceso de estimación semi-informal que combina las ventajas y experiencias basadas en los métodos de regresión (Ferens, 1988). Este esquema ha sido usado en muchas disciplinas y fue usado en el desarrollo del modelo COCOMO II.

En términos del desarrollo ágil, el proceso de estimación es un proceso iterativo mediante el cual las historias de usuario en XP representan piezas de funcionalidad

que ser estimado y esto se hace cada 2 semanas. Un tiempo total esperado para cada una de estas historias es estimada por los desarrolladores y los clientes luego priorizar las historias basadas en estas estimaciones iniciales y en el valor de negocio de cada uno (Lovaasen, 2001). De acuerdo a Highsmith (2003), la naturaleza de los métodos ágiles a menudo son el resultan en presupuestos fijos de programar y el alcance del proyecto se mantiene flexibles. por otra parte, Ceschi, et al. (2005) informan que las empresas que utilizan métodos ágiles por lo general se inclinan hacia "contratos flexibles en lugar de fijar funciones definidas en precio y tiempo".

Aunque los proyectos se caracterizan por cambios en el alcance y sus requisitos, el impacto de estos cambios puede variar tremendamente dependiendo del momento en que suceda este cambio. Los métodos ágiles tienen a reducir el costo de los cambios a lo largo del desarrollo del sistema, pero no necesariamente en la ocurrencia de los cambios.

Las técnicas usadas para estimar los proyectos de desarrollo ágil han sido típicamente basadas en la experiencia, en donde los desarrolladores observaron proyectos del pasado o iteraciones y usando su experiencia generaron la estimación para las historias de usuario.

El cálculo de la estimación de cada iteración se realiza en la sesión de planeación, donde se produce la estimación en esa fase temprana del proyecto. Se puede estimar los proyectos desde dos perspectivas diferentes en serie cuando existan proyectos sensibles al costo, mientras cuando existan proyectos a la sensibilidad de los requerimientos cambiantes se deberá establecer un desarrollo concurrente.

La estimación que se realiza en los proyectos ágiles se puede enunciar como fácil debido a la frecuencia que se hacen las mismas en cada iteración. Manteniendo un grado de exactitud, además de perfeccionar las técnicas de estimación entre los miembros del equipo de desarrollo.

La estimación del costo del software se considera comúnmente como hacer estimaciones del esfuerzo para completar el software para un proyecto o actividad de desarrollo del sistema. El esfuerzo es una medida del costo de la mano de obra y se informa en unidades tales como persona-mes o persona-año. Los propósitos para los cuales se requieren estimaciones de costos de software son variados. Algunos de ellos son la planificación y el control de proyectos, la evaluación de la

viabilidad del desarrollo del sistema y la preparación de presupuestos u ofertas para su presentación a los clientes. La planificación juega un papel clave en la gestión exitosa de las actividades de desarrollo del sistema.

Los propósitos para los cuales se requieren estimaciones de costos de software son variados. Algunos de ellos son la planificación y el control de proyectos, la evaluación de la viabilidad del desarrollo del sistema y la preparación de presupuestos u ofertas para su presentación a los clientes. La planificación juega un papel clave en la gestión exitosa de las actividades de desarrollo del sistema. Las estimaciones de costos de software son típicamente inexactas, y no hay evidencia de que la comunidad de desarrollo de software esté mejorando su capacidad de hacer estimaciones precisas. A pesar del esfuerzo de investigación que se ha dirigido a desarrollar modelos para la estimación de costos de software, parece que la mayoría de las estimaciones se realizan de manera informal, los modelos de costos se usan con poca frecuencia y, cuando se usan, los modelos de costos no están asociados con una precisión significativamente mayor.

Una medida es una representación numérica de un atributo específico de una entidad (Fenton, 1991). Las medidas para el desarrollo del sistema representan los atributos de las entradas, salidas o el proceso en sí. Las medidas se pueden clasificar como directas o indirectas. Un recuento de líneas entregadas de código es un ejemplo de medida directa. Su valor puede determinarse aplicando una técnica de medición directamente al producto de interés. Las medidas indirectas se basan en mediciones de uno o más atributos diferentes. Las medidas de productividad, que se basan en las tasas de completar un producto del proceso de desarrollo, como líneas de código por día hombre, son ejemplos de medidas indirectas.

En este ejemplo, la relación entre las medidas está determinada por la definición de la medida de productividad. Una medida del esfuerzo de desarrollo que se calcula a partir de una medida como las líneas de código es otro ejemplo de medida indirecta. Esta medida se basa en una relación empírica entre el atributo del proceso de desarrollo, el esfuerzo y un atributo de un resultado del proceso, la longitud del software producido. Esta relación empírica se formaliza a través de un modelo numérico.

Las predicciones generalmente se basan en medidas indirectas, porque el atributo de interés no se puede medir directamente en el momento en que se realiza la predicción. Por ejemplo, el esfuerzo para implementar el software para un sistema puede predecirse utilizando un modelo que relaciona este esfuerzo con un atributo del diseño del sistema, como su longitud medida por líneas de texto. Las horas reales de implementación del software se pueden medir directamente. Después, la predicción del esfuerzo, realizada a través de la medida indirecta, se puede comparar con la medida directa del esfuerzo para evaluar la precisión de la predicción.

Se cuenta con cuatro clases de métodos de predicción:

- Empírico
- Analógico
- Teórico
- Heurístico

Los métodos empíricos analizan los datos para establecer un modelo numérico de la relación entre las medidas de los atributos en el modelo empírico. El análisis de regresión es un ejemplo de un método empírico.

Los métodos de predicción analógica usan medidas de los atributos del modelo empírico para caracterizar el caso actual, para lo cual se debe hacer la predicción. Los valores conocidos de las medidas para el caso actual se utilizan para buscar un conjunto de datos para casos análogos. La predicción se realiza interpolando desde uno o más casos análogos hasta el caso actual. Los métodos de predicción teórica proponen un modelo numérico basado en el modelo empírico. El modelo teórico debe validarse empíricamente, en comparación con los datos reales de las medidas. Los métodos heurísticos se usan como extensiones de los otros métodos.

Las heurísticas son reglas empíricas, desarrolladas a través de la experiencia, que capturan el conocimiento sobre las relaciones entre los atributos del modelo empírico.

El propósito para el cual se requiere una predicción es la influencia central sobre qué medidas se van a predecir. Define la razón para hacer la predicción. Ejemplos de propósitos para los que se requieren predicciones incluyen los siguientes:

- Explorando la viabilidad de desarrollar o comprar un nuevo sistema
- Explorando el impacto de cambiar las funciones de un sistema existente
- Planificando cómo asignar personal a un proyecto de desarrollo de software
- Citando un precio o programa para un nuevo sistema.

El propósito de una predicción tiene la mayor influencia en las decisiones sobre qué atributos del sistema son de interés. Estos atributos determinan el tipo de medidas que son apropiadas. Las medidas para predecir pueden ser inmediatamente evidentes a partir del propósito. En nuestro escenario de desarrollo de sistemas, presentado en la sección anterior, la empresa debe preparar un presupuesto para la licitación. El costo del sistema en moneda local o extranjera es la medida obvia. Este costo podría dividirse entre los costos de software y hardware. En este caso, el costo del hardware para el sistema debe predecirse para su inclusión en la cotización. El costo del hardware incluye el costo del hardware para un servidor en el que reside la base de datos que soporta el servicio de información, así como los costos para el hardware y periféricos del cliente.

Antes de acercarse a un proveedor de hardware para solicitar un presupuesto para el servidor, el equipo de ofertas debe predecir cuánta capacidad de almacenamiento se requiere, qué ancho de banda deben soportar las interfaces de comunicaciones y qué capacidad de procesamiento se requiere para manejar la demanda del usuario. A partir de una medida inicial del costo total, la cantidad de medidas a predecir aumenta a medida que el problema se descompone.

Se cuenta con una gran variedad de procesos de estimación de costos de software en los que se pueden observar: (Bailey & Basilio, 1981) propusieron el " modelo Bailey-Basili" como un proceso de estimación del esfuerzo que puede usarse en una organización particular. (Boehm B. W., 1981) propone un proceso de siete pasos para la estimación del costo del software. (DeMarco, 1982) propone un proceso para desarrollar y aplicar modelos de costos basados en datos recopilados

localmente. Aboga por el uso de modelos de costos de un solo factor derivados por regresión lineal. Un modelo de costo de factor único predice el esfuerzo para todo o parte de un proyecto, basado en una sola variable independiente. La estimación de esfuerzo obtenida del modelo de costo de factor único se debe ajustar aplicando un factor de corrección. Heemstra (1992) describe un proceso general de estimación de costos de software. Su proceso asume el uso de modelos de esfuerzo que dependen del tamaño, y el uso de controladores de costos junto con estos modelos para hacer ajustes de productividad. (Humphrey W. S., 1995) describe un proceso de "estimación basada en proxy" (PROBE) como parte de su proceso de software personal. El método PROBE aboga por el uso de medidas de selección personal y modelos de regresión basados en datos personales para estimar el tamaño de un producto de software. Un proxy es una medida de tamaño que se puede usar para estimar la longitud de un producto en líneas de código. Como ejemplo, PROBE utiliza un recuento de objetos como una medida de tamaño de proxy. La estimación de las líneas de código se usa para predecir el esfuerzo de una persona para producir el producto. Es posible mejorar la precisión de una estimación combinando el juicio y las estimaciones hechas por un número de personas. (Boehm B. W., 1981) describe la técnica *Wideband* Delphi para combinar las opiniones de los expertos para hacer una estimación de tamaño. Cada experto recibe una especificación y un formulario de estimación. En una reunión de grupo, los expertos discuten los problemas de estimación. Cada experto completa el formulario de estimación de forma anónima. Se presenta un resumen de las estimaciones de esta ronda a cada experto, y otra reunión grupal se lleva a cabo para discutir las estimaciones de las rondas anteriores.

Se realizan rondas adicionales hasta que se estima que las estimaciones han convergido satisfactoriamente. Taff et al. (1991) describe otro proceso para el grupo de estimación de AT&T, "*Estimmeting*". Produce una estimación del esfuerzo para desarrollar futuras estimaciones de un sistema. De los anteriores trabajos se observa que en los procesos de (DeMarco, 1982), (Heemstra, 1992) e (Arifoglu, 1993) incluyen explícitamente un paso para estimar la duración o el tiempo transcurrido para el proyecto de software.

El proceso de predicción no prescribe qué medidas deben estimarse, ya que se supone que las medidas apropiadas dependen del contexto en el que se realizan las predicciones.

Como la estimación del costo del software a menudo se utiliza para apoyar la planificación del proyecto, es probable que a menudo se busque una estimación de la duración junto con una estimación del esfuerzo. Sin embargo, en situaciones donde la duración es una restricción en el proyecto de software, puede ser más efectivo predecir cuánto se puede lograr en el tiempo disponible, usando la duración como una entrada a la predicción.

Una vez que se ha realizado la estimación del esfuerzo, (Heemstra, 1992) e (Arifoglu, 1993) incluyen un paso que distribuye el esfuerzo a lo largo de las fases del proyecto o ciclo de vida. Estos dos procesos son ejemplos de un enfoque descendente de la estimación. Una estimación de costos de software de arriba hacia abajo es una estimación del esfuerzo total o duración, en función de las características generales de un proyecto.

Un enfoque de abajo hacia arriba para la estimación del costo del software descompone el desarrollo del sistema en una serie de actividades o entregables del sistema y estima el esfuerzo para cada uno de estos individualmente, y luego combina estas estimaciones para una estimación total del esfuerzo. Tanto los enfoques ascendentes como descendentes se pueden aplicar a cada uno de los métodos de predicción discutidos en el marco de selección.

El proceso de predicción no supone que uno u otro sea preferible. Esto dependerá de las circunstancias en las que se realizan las predicciones. (Boehm B. W., 1981)incluye un paso para precisar los requisitos del software, lo que enfatiza la necesidad de definir un conjunto de requisitos de costos posibles antes de intentar hacer estimaciones basadas en ellos. La importancia de tener requisitos bien definidos sobre los cuales basar las estimaciones se confirma mediante la encuesta de (Lederer & Prasad, 1993), en la que los participantes consideraron los cambios y malentendidos sobre los requisitos como las causas clave de las estimaciones inexactas. Sin embargo, en la práctica, a menudo se requieren estimaciones en circunstancias en las que no hay una definición clara de los requisitos disponible

o se pueden preparar como una base para la estimación, por ejemplo, en la licitación de un contrato. En estas circunstancias, es importante que se documenten los supuestos en los que se basa una estimación, como se sugiere en el proceso de predicción.

(Heemstra, 1992) incluye el análisis de riesgos como parte del proceso de estimación del costo del software. El análisis de riesgos y la estimación del costo del software son actividades de gestión de proyectos interrelacionadas. Los riesgos del proyecto se pueden evaluar en términos de su costo potencial y los impactos del cronograma (por ejemplo, (Fairley, 1994) Cualquiera de los supuestos en los que se basa una estimación presenta un riesgo para un proyecto, según la probabilidad de que la suposición resulte incorrecta y el impacto de que sea incorrecta en la estimación. Las suposiciones documentadas como parte del proceso de predicción son entradas para el análisis de riesgo.

Los procesos del equipo descritos en esta sección sirven como ejemplos de cómo combinar las estimaciones generadas por los miembros del equipo. Podrían usarse junto con cualquiera de los otros procesos de estimación de costos de software descritos, o el proceso de predicción.

Modelos de estimación de costos de software

Esta sección revisa una variedad de modelos de estimación de costos de software. Los modelos de estimación de costos de software han recibido más atención de los investigadores que los procesos o estudios de la práctica de estimación de costos. El marco de selección clasifica los métodos de predicción como empíricos, analógicos, teóricos y heurísticos. Clasificamos cada modelo presentado en esta sección según el método de predicción en el que se basa principalmente, ya que un modelo se puede basar en más de uno de estos métodos. Los modelos basados en cada método se presentan en subsecciones separadas.

La clasificación de los modelos empíricos se amplía en esta sección, clasificando los modelos como paramétricos o no paramétricos. Un modelo paramétrico tiene una forma funcional explícita, que relaciona una variable dependiente con una o más variables independientes, por ejemplo, el modelo COCOMO (Boehm B. W.,

1981). Un modelo no paramétrico no tiene una forma funcional explícita, por ejemplo, un modelo desarrollado utilizando una técnica de aprendizaje automático, como una red neuronal artificial.

Los modelos más comunes son modelos paramétricos empíricos. Donde se ha hecho el esfuerzo, estos modelos se han extendido, en algunos casos, mediante el uso de controladores de costos. Las ventajas y desventajas de los controladores de costos se discuten en este contexto. En contraste con la mayoría de los modelos presentados, qué predictores se han desarrollado un par de modelos para predecir el tiempo transcurrido para un proyecto. Estos se presentan brevemente en la subsección sobre modelos paramétricos empíricos.

Modelos paramétricos empíricos.

Modelos de esfuerzo.

La forma más simple de un modelo de esfuerzo paramétrico empírico es una función que se relaciona con el esfuerzo de desarrollar un sistema o programa para una medida de tamaño. En este contexto, una medida de tamaño es un recuento de algunas características de un producto del proceso de desarrollo, por ejemplo, un recuento de la cantidad de líneas de código en un programa. El esfuerzo a menudo se mide en persona-días o persona-mes. Los modelos se desarrollan ajustando la función de un par de valores de tamaño y esfuerzo, utilizando técnicas de regresión. Los modelos con relaciones lineales y exponenciales entre el esfuerzo y el tamaño son los más comúnmente explorados. El modelo de (Albrecht & Gaffney, 1983) se basa en la relación lineal entre el esfuerzo y los puntos de función.

Visto con ligereza, el desarrollo de un modelo paramétrico empírico es un ejercicio de ajuste de curvas. Esta visión enfatiza algunas de las trampas inherentes al desarrollo de estos modelos. (Courtney & Gustafson, 1981), que es uno de los ejemplos más importantes de relaciones empíricas. Las posibilidades de descubrir un modelo "bueno" con este enfoque son altas con conjuntos de datos pequeños.

También hay algunos escollos conspicuos en el desarrollo de modelos paramétricos empíricos. Se propone un modelo, la cantidad de observaciones en el conjunto de

datos utilizado en el modelo. El modelo también podría invalidarse si los parámetros de entrada no son mutuamente independientes, por lo que es necesario probar los supuestos de su independencia. Las técnicas estadísticas que se utilizan para derivar estos modelos también suponen que el conjunto de datos tiene ciertas propiedades y que consta de muestras de la misma población subyacente. Esta última suposición puede ser difícil de justificar donde se tienen en cuenta los resultados.

Práctica de estimación de costos de software

(Lederer., y otros, 1990) informan sobre el proceso de estimación de costos seguido por una organización con un departamento interno de sistemas de información (ISD). Se han comparado estimaciones y valores reales para una serie de proyectos, con un esfuerzo real en el rango de unos pocos días a 120 días. Se informa que el 60% de las estimaciones difieren de los valores reales en un 20% o más.

Los autores señalan que estos son proyectos pequeños, por lo que las consecuencias de estimaciones inexactas no son serias. La organización sigue una estimación de proceso desarrollada localmente. El usuario y la gerencia de ISD requieren una estimación inicial para determinar si un proyecto puede estar justificado por los costos. Esta estimación a menudo es inexacta porque el alcance del proyecto no se conoce en detalle, y en algunos casos lo es.

Si la administración de usuarios elige continuar basándose en la estimación inicial, se prepara una segunda estimación con más cuidado. El estimador identifica módulos y sus funciones y archivos asociados. El estimador evalúa el nivel de complejidad de cada módulo de forma subjetiva. Un análisis de matrices existente y esfuerzo de programación para módulos de diferentes complejidades. El estimador usa esta matriz para estimar el esfuerzo de cada módulo y ajusta la suma en función de la experiencia y la intuición.

La subjetividad y la incertidumbre sobre el rendimiento en estas estimaciones deja espacio para que la política influya en el resultado. El esfuerzo se vuelve a estimar en cada fase, pero se aplica a las nuevas estimaciones de las anteriores. Las estimaciones se obtienen a través de la negociación. Es discutible que se haya

negociado una estimación. Si se modifica una estimación sin alterar las suposiciones en las que se basa, puede agregarse al proyecto.
(Hihn & Habib-agahi, 1991) informaron los resultados de una encuesta de técnicas de estimación de costos en el Jet Propulsion Laboratory (JPL). A los participantes en la encuesta se les pidió que estimaran el tamaño y el esfuerzo de desarrollar una pieza de software, en base a un documento de diseño que describe el software. El tamaño se mide en líneas de código y esfuerzo en días-persona. Se sabe que el tamaño real y el esfuerzo asociado con esta pieza de software se han implementado antes de la encuesta.

La encuesta muestra que la mayoría del personal técnico del JPL utiliza una analogía informal y una partición de alto nivel de los requisitos para estimar los costos del software. Tanto el esfuerzo como el tamaño fueron inexactos, y los valores promedio fueron significativamente más bajos que los valores reales. Las estimaciones de esfuerzo son más precisas que las estimaciones de tamaño. Solo el 16% de las estimaciones presupuestarias cayeron dentro del 20% del tamaño real. El subgrupo de personas con la aplicación de la experiencia es significativamente mejor que el equipo en general al estimar tanto el tamaño como el esfuerzo. En este grupo, el 27% del presupuesto cayó dentro del 20% del tamaño real, y el 46% del esfuerzo cayó dentro del 20% del esfuerzo real. La inexactitud de las estimaciones se atribuye a una falta de énfasis en la organización. Esto, a su vez, fue pensado para ser más eficiente, menos costoso y más probable de lo esperado.

(Heemstra, 1992) informa de un estudio de campo de proyectos de desarrollo de software en 598 organizaciones holandesas. Entre sus hallazgos se encuentra que el 80% de los proyectos van más allá de los presupuestos y duraciones, y que los excesos promedio son del 50%. El estudio de campo también encontró que el 35% de las organizaciones no hacen estimaciones, en cuyo caso el presupuesto y la duración son presumiblemente determinados por factores externos al proyecto.

La estimación por analogía (61%) y el juicio de expertos (26%) son los métodos más utilizados por las organizaciones que realizan estimaciones. Catorce por ciento de estas organizaciones informan que también usan modelos paramétricos.

Las organizaciones encuestadas pueden usar más de un método de estimación de costos. De las organizaciones encuestadas, el 30% dijo que las estimaciones estaban limitadas por el costo o los recursos disponibles.

(Lederer & Prasad, 1993) informan los resultados de una encuesta a gerentes de sistemas de información y analistas de diferentes organizaciones en los Estados Unidos. La encuesta indica que aproximadamente el 63% de los proyectos rebasan significativamente sus estimaciones, mientras que el 14% subestima significativamente sus estimaciones.

Evaluación del método de evaluación del método de evaluación del método de evaluación del método de evaluación. Esto es consistente con los hallazgos de (Heemstra, 1992), que es el método de estimación más común. Solo el 17% de los participantes de la encuesta informan que usan un paquete de software para ayudar a estimar los costos de desarrollo. Este grupo informó una proporción mayor de proyectos que sus estimaciones de que el grupo no usó paquetes de software. Este resultado también es consistente con el hallazgo de (Heemstra, 1992) que los modelos de costos no mejoran la precisión de las estimaciones. (Lederer & Prasad, 1993) examinaron las opiniones de los participantes en la encuesta sobre las causas de las estimaciones inexactas. Cuatro de las cinco causas principales están relacionadas con dificultades en la obtención y la definición. La otra causa de las estimaciones inexactas que aparecieron en los cinco primeros fue tareas olvidadas.

(Vicinanza, Mukhopadhyay, & Prietula, 1991) estudiaron el uso del juicio experto en la estimación de costos. Se solicitó a los expertos estimar el costo de un proyecto basado en COCOMO y modelos de punto de función. Las estimaciones se realizan sin depender de técnicas algorítmicas formales. El mismo conjunto de proyectos se había estimado algorítmicamente en un estudio previo (Kemerer, 1987). Cuando los resultados han sido comparados, las estimaciones de los expertos son más variadas que las estimaciones hechas por los modelos. El éxito de los expertos se atribuye a una mayor sensibilidad a los factores que afectan la productividad en una situación particular. Sin embargo, hay evidencia de otros estudios, que sugiere

que las predicciones hechas por expertos son inferiores a las hechas por modelos lineales simples (Johnson, 1998).

Estos estudios destacan la necesidad de medición, retroalimentación y empaque. Estas son partes clave de nuestro proceso de predicción que no parecen ser frecuentes en la práctica.

La prevalencia de estimaciones informales es inevitable cuando se realiza una estimación. Es probable que la falta de datos históricos adecuados sea uno de los factores que limitan la aplicación de técnicas formales de estimación y modelos de estimación de costos. (Heemstra, 1992) informa que solo se ha registrado el 50% de los datos. Es poco probable que muchas de las organizaciones encuestadas hayan hecho demasiado hincapié en la importancia de la medición (por ejemplo, (DeMarco, 1982), (Fenton, 1991)). El análisis e interpretación de la investigación en este artículo identifica una serie de limitaciones y dificultades en la investigación y práctica de estimación de costos de software y ofrece algunas explicaciones de por qué han surgido. La falta de medición dentro de la industria de desarrollo de software está limitando el progreso en el campo de la estimación de costos de software. Este es un problema tanto con investigadores como con profesionales. Los investigadores necesitan datos para explorar y desarrollar nuevos modelos y métodos para la estimación de costos de software. Los practicantes necesitan usar métodos y modelos existentes con éxito.

Los modelos para predecir el esfuerzo total de desarrollo del sistema son importantes y han sido el centro de muchas investigaciones previas. Si este enfoque continúa, sin embargo, puede ser una limitación, ya que los modelos para predecir el esfuerzo son necesarios para la planificación y el monitoreo.
Hacer estimaciones precisas utilizando modelos de estimación de costos de software es difícil. La incertidumbre es la causa de la variación en el esfuerzo. El desarrollo de modelos que explican mejor esta variación y, por lo tanto, son más precisos, es un gran desafío para los investigadores.

Esto debe complementarse con el esfuerzo de calibrar adecuadamente los modelos. La incertidumbre también se introduce cuando los valores de los parámetros de entrada no se pueden medir. Se podría decir que hay muy pocos modelos que sean

adecuados para la estimación temprana del ciclo de vida, si la capacidad de medir los parámetros de entrada es el criterio para elegirlos.

Una dificultad evidente con la práctica de la estimación de costos de software, en toda la industria, es que la precisión de las estimaciones de costos de software no está mejorando. Una explicación para esto es que es importante hacer un enfoque informado para estimar que es necesario mejorar.

Se necesita investigación sobre las medidas iniciales del ciclo de vida para predecir el esfuerzo, de modo que pueda estimarse directamente al comienzo del desarrollo del sistema. Son inadecuados los modelos basados en líneas de código para este propósito. La investigación de modelos para predecir el esfuerzo total. La investigación en esta área puede ser más precisa si los modelos se utilizan para predecir el futuro. Una de las desventajas de desarrollar modelos para el esfuerzo total de desarrollo del sistema es la variación que es inevitable en la interpretación de esta medida. Sin embargo, el hecho de que a menudo esté disponible un valor para el esfuerzo total es una ventaja para los investigadores.

Los modelos para estimar el esfuerzo de las actividades de desarrollo se acoplarán estrechamente a las definiciones de ciclos de vida y procesos, lo que probablemente dificultará la información, especialmente entre las organizaciones.

Un área desafiante para el desarrollo de modelos accesibles para predecir los efectos del estrés, la duración y el personal para el desarrollo del sistema y las limitaciones en estos temas. Los gerentes de proyecto deben ser capaces de presentar estas soluciones, y estos gerentes serían mejores. La simple vista del esfuerzo total puede deducirse de los atributos del sistema solo, ejemplificado por un modelo que predice el esfuerzo basado en líneas de código, ignora la interacción entre el personal, la fecha de finalización del objetivo y el esfuerzo esperado. El esfuerzo total gastado está determinado por los efectos de estas interacciones.

Desarrollar enfoques para la estimación de costos de software que puedan aprovechar el conocimiento sobre el progreso actual. La reestimación es necesaria debido a las suposiciones sobre las cuales se estiman los cambios. El desafío en esta área es desarrollar enfoques para ayudar a medir el progreso en el proceso de

estimación, de modo que puedan hacerse estimaciones del esfuerzo o tiempo para completar.

La recomendación más importante para la mejora es seguir un proceso para la estimación del costo del software, al menos de manera informal. La retroalimentación de la evaluación de la precisión de las predicciones y el análisis de las causas de la inexactitud es necesaria para la mejora. Estos métodos de predicción y predicción se han propuesto en este artículo mediante el método de prescripción, y el proceso predictivo.

Otra área clave para la mejora es la medición. Un proceso de estimación de costos de software que se adopta sin los beneficios de los datos apropiados para respaldarlo.

Los modelos de estimación de costos de software desarrollados por los investigadores no son una solución inmediata para mejorar las estimaciones. Deben calibrarse para el entorno en el que se aplican, si deben usarse con confianza. Calibración de modelos existentes y desarrollo de modelos locales. La falta de medición dentro de la industria de desarrollo de software está limitando el progreso en el campo de la estimación de costos de software. Los investigadores necesitan datos para explorar y desarrollar nuevos modelos y métodos para la estimación de costos de software. Otra área clave para la mejora es la medición. Un proceso de estimación de costos de software que no cuenta con el respaldo de los datos apropiados fallará. Los datos deben recopilarse y almacenarse para que la estimación del costo del software pueda calibrarse para el entorno en el que se aplican. Sin calibración, los modelos no serán confiables y no se pueden usar con ninguna consecuencia.

En el artículo de Sulaiman y Barton (2006) se define una metodología llamada AgileEVM, que tiene el propósito de monitorear el avance de un proyecto en los atributos de tiempo y costo. AgileEVM se basa en el método de administración del valor ganado (EVM, *Earned Value Management*). EVM se define en Project Management Institute (2003) como: “EVM es un método para la integración del alcance, cronograma y recursos, y para medir el desempeño del proyecto”.

En AgileEVM las métricas de Scrum se relacionan directamente con las métricas de EVM tradicional buscando no agregar burocracia innecesaria al proceso de desarrollo.

El monitoreo del desempeño y del presupuesto del proyecto en Agile EVM se realiza al final de cada iteración, en estas revisiones se recolectan cuatro parámetros: el número de iteración actual (n), los puntos de historia terminados durante la iteración (PC), los puntos de historia agregados o eliminados de la lista de trabajo que se tienen que desarrollar en las siguientes iteraciones (lista backlog) durante la iteración actual (PA) y el costo de la iteración (SC).

Kang y Choi (2010) proponen un modelo de monitorización dinámico, la métrica básica de monitoreo son los puntos de función FP (*Function Points*), los cuales describen la funcionalidad de cada Historia de Usuario. En este modelo el monitoreo es diario, por lo que se tiene una base histórica de información; este modelo usa el filtro de Kalman y el modelado de espacios de estado con la finalidad de hacer proyecciones más precisas del avance en puntos de función que se espera tenga el producto según la velocidad histórica del equipo.
El modelo de espacio de estado se alimenta con la información de los FP faltantes y las variaciones del alcance en el proyecto.

En los métodos ágiles se utilizan prácticas empíricas basadas en el juicio de expertos para estimar el costo de desarrollo para un nuevo producto, en la literatura se identificaron dos prácticas comunes para esta finalidad, la más común involucra como métrica base a los SP (*Story Points*) y la segunda se basa en los FP. La primera manera identificada, que es la más común en métodos ágiles, tiene como métrica principal a los SP. En esta práctica la primera tarea a realizar es la estimación del esfuerzo total necesario para desarrollar el proyecto, esta estimación la realiza el equipo encargada de su desarrollo. En seguida se define una velocidad base adquirida de datos históricos no muy confiables, o de juicio de expertos, para conocer el tiempo aproximado en el que el proyecto terminará.

Conociendo el tiempo necesario para desarrollar el proyecto, el líder del proyecto calcula el costo de las personas que forman parte del equipo para obtener un presupuesto necesario para el recurso humano, a tal estimación además se

agregan otros tipos de gastos que el proyecto necesite y se agrega una suma acumulada de gastos al total del presupuesto estimado (Cohn, 2005).

Otra manera de obtener la estimación de costos se propuso en Kang y Choi (2010), donde la métrica base para realizar estas estimaciones es utilizar FP, lo que involucra más esfuerzo en la planeación del producto, pero los autores aseguran que su modelo es más preciso ya que los FP son valores absolutos y no relativos como los SP.

La programación en par recientemente ha tenido mucha atención debido al gran impacto que ha tenido la metodología programación extrema (extreme Programming). De 1988 a 2001 los estudiantes aprendices de las ciencias de la computación realizaron experimentos controlados para evaluar la eficiencia de programación en parejas. Estos experimentos se dividieron en un grupo de programadores en par y un grupo solitario de programación, a los dos grupos se les pidió desarrollar un mismo programa para que los resultados fueran comparados directamente reflejando como resultados diferentes e incompatibles.

Los estudios identifican que la programación en parejas trata el proceso de programación en parejas como un cuadro negro (Williams, R. Kessler, & Cunningham, 200). Tal como el caso que se experimentó con 41 estudiantes. En cuanto a la identificación los programas en parejas se manejan una automatización en el post desarrollo de casos de pruebas. La demostración de los datos en pares tiene una variación más pequeña en comparación con la muestra de los individuales.

La evaluación del costo de desarrollo mostró que, después de un período de ajuste inicial en la que las parejas gastan alrededor del 60 por ciento más de horas de programador en la realización de las tareas, la sobrecarga de trabajo se redujo a 15 por ciento. La programación en parejas en comparación con la programación en solitario con 15 programadores profesionales. La tarea experimento fue implementar un control de coherencia de base de datos. Todos los pares superaron a los individuales. Aunque el promedio de tiempo para la finalización era más de 12 minutos más para las personas, la diferencia no fue estadísticamente significativa.

2.3 Modelos de Mejora para la estimación de Proyectos

En la actualidad es muy común encontrar proyectos de ingeniería de software donde la gestión de proyectos se lleva a cabo débilmente. Muchas de las veces los líderes de los proyectos se encuentran con dificultades intentando cubrir los plazos fijados de entrega, como final diseccionan al usuario final y acaban dedicando el tiempo restante al mantenimiento del sistema. Estos son aspectos de problemas técnicos y de gestión, y derivado de esto, los líderes del proyecto se cuestionan:

- ¿Por qué lleva tanto tiempo terminar un proyecto?
- ¿Por qué es tan elevado el costo?
- ¿Por qué no se pueden encontrar todos los errores durante y antes de que se entregar el proyecto?
- ¿Por qué no se puede entregar a tiempo el producto?

El responder estas preguntas y empezar a resolver los problemas de gestión de proyectos le corresponde al área de ingeniería de software, el punto determinante es que se pueda llevar al proceso de desarrollo de software hacia una disciplina que nos permita tener un control de los aspectos a considerar durante la gestión del proyecto.

Descripción del Proceso del desarrollo del software.

En primero lo que se debe considerar es entender el contexto del problema dentro de la realidad, con el fin de obtener un modelo que le corresponda al proyecto. En la ingeniería de software se implementan una variedad de diagramas para poder modelar los componentes del sistema, estos modelos representan las interrelaciones entre los distintos elementos (procesos) para el desarrollo de sistemas y nos permita comprender y estudiar su comportamiento.

El proceso un desarrollo de software se deriva tres tipos de áreas de responsabilidad (componentes de proceso) con sus correspondientes actividades

asociadas (Humphrey W. , 1989) Gestión de proyecto, Desarrollo y Calidad. Tal como se muestra en la figura.

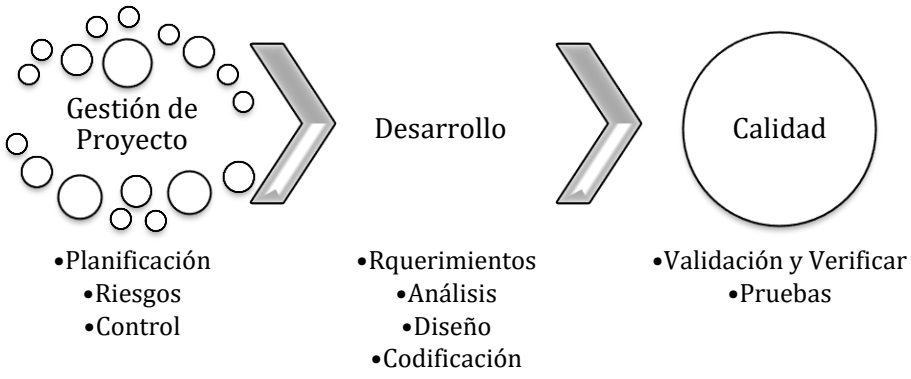

Figura 7. Componentes del desarrollo de Software.

En el componente de desarrollo se ocupan los métodos, técnicas y herramientas requeridas para la elaboración de los productos de software, donde se incluyen las actividades de *requerimientos, diseño y codificación.* El componente de calidad garantiza la validez de las actividades que se generan de un proyecto. Para garantizar un alto nivel de calidad de un producto de software no basta con verificar y validar si no debe de apegarse a las actividades intermedias como los documentos de requerimientos y de diseño, además se deberá preservar la integridad de los fragmentos que componen un desarrollo de software. El componente de Gestión de Proyectos se enfoca en cuantificar la probabilidad de éxito del proyecto al igual definir las tareas a realizar, la calendarización, la estimación de costos y la asignación de recursos humanos.

Proceso de estimación de costos.

En los procesos de estimación de costos se miden una serie de factores (tiempo, presupuesto, información histórica del proyecto y recursos humanos) que llegan a formar parte de los parámetros en los modelos de toda una metodología. Estos factores son determinantes en los resultados de cualquier modelo de estimación de costos.

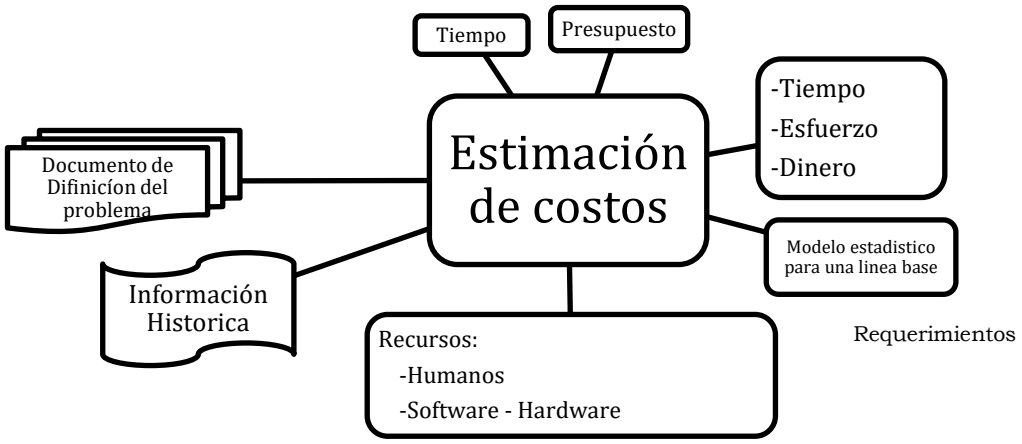

Figura 8. Proceso para calcular el esfuerzo y el tiempo de un desarrollo de software.

Fundamentos Teóricos de la metodología

La noción precedente que necesita un líder de proyecto para estimar un proyecto de desarrollo de software no solo se basa de su experiencia sino de tener en cuenta técnicas, métodos y métodos determinen la estimación de costos. Los elementos que se manejan en la planeación de proyectos son bastante extensos pero los más empleados son el método de la ruta crítica, las gráficas de Gantt, la descomposición del trabajo (WBS) y las gráficas de Pert. Estos conocimientos teóricos son un término necesario para aplicarlos en cualquier metodología de estimación de costos.

Dentro de la estimación de costos basándose en los fundamentos teóricos se abordan las siguientes preguntas.

- ¿Cuál es la arquitectura utilizada en la planeación de proyectos?
- ¿Cuáles son las técnicas utilizadas para tener una mejor estimación?

- ¿Cuál es el eje básico de la metodología de estimación de costos?
- ¿Cuál es la arquitectura de la metodología de estimación de costos?

Para darle seguimiento lógico a la metodología, se trazaron las siguientes secciones de interés.

- *Arquitectura para la planeación de proyectos* (Reglas básicas para definir proyectos de software)
- *Eje de la metodología de estimación de costos* (Especificación de técnicas y métodos que se usaran en cada paso)
- *Fundamentos* (Pasos que se deben seguir para diseñar)

Arquitectura para la planeación de proyectos

En la estimación de un proyecto se deben de conocer antes las reglas que se deben utilizar en la planeación de proyecto. Una arquitectura está enfocada a darle un panorama al usuario ver lo que esta fuera de su alcance en diferentes formas de detalle. Donde los niveles de abstracción representan un nivel de detalle, tal como se muestra en la siguiente figura.

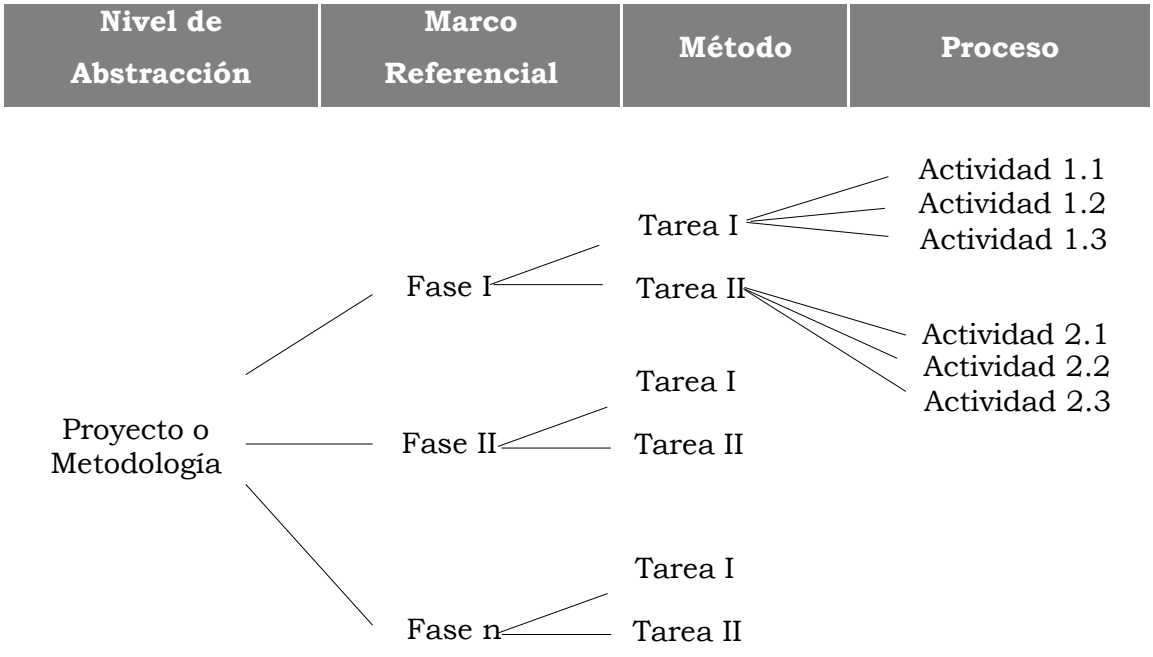

Figura 9. Unidades lógicas de trabajo por nivel de abstracción

Nivel de Abstracción Fase. El Gerente es el interesado en este nivel ya que se emplean los siguientes aspectos.

1. Las fases de desarrollo que se cubrirán en cada proyecto.
2. El tiempo total del proyecto.
3. La fecha de inicio estimada de cada fase.
4. La fecha final estimada de cada fase.
5. El tiempo de cada fase.

Nivel de abstracción Tarea. El líder de proyecto es quien emplea los siguientes aspectos en el nivel tarea.

1. Las tareas (bien definidas) que se cubrirán en cada fase.
2. El tiempo total de la tarea.
3. La fecha de inicio estimada de cada tarea.
4. La fecha final estimada de cada tarea.
5. El tiempo sumado por cada tarea.

Nivel de abstracción Actividad. El programador es el interesado en este nivel dado que es donde se requiere tener programadas las actividades para llevar a cabo una tarea.

Eje de la metodología de estimación de costos

El manejo de una metodología de estimación de costos parte desde un eje central donde se integran técnicas y métodos que se utilizan independientemente del ciclo de vida del proyecto, como paso inicial se conforman los requerimientos donde estos se clasifican por aspectos que se integran de una categorización de requerimientos, una estimación del tamaño de los requerimientos, un estimando de esfuerzo de cada requerimiento, un estimando de tiempo por requerimiento, un ajuste de plan de trabajo y comprimir tiempos y una estimación de costos.

Categorías de requerimientos

La identificación por categoría nos permite dividir el trabajo e identificar las fases, tareas y actividades que se desarrollan en el proyecto. El objetivo es identificar a que categoría pertenece un requerimiento del cliente.

Un requerimiento se define por las siguientes categorías.

a) Requerimientos que necesitan desplegar gráficas estadísticas
b) Requerimientos que dan mantenimiento a la base de datos
c) Requerimientos que generan Reportes
d) Requerimientos de consulta de información
e) Requerimientos que incrustan Objetos gráficos desde otras aplicaciones
f) Requerimientos que consultan y dan mantenimiento a la base de datos

Tamaño de requerimientos

El tamaño de los requerimientos se determina a través de la aplicación de una métrica, de acuerdo a la categoría que pertenece el requerimiento que se estima. Un requerimiento siempre finaliza siendo un programa ejecutable, el cual contiene funciones y algoritmos que implementa el servicio que requiere el cliente. Esto nos puede ayudar a determinar el tamaño, pero nos faltaría determinar la complejidad.

La complejidad la calculamos con base a la persona que realizará la tarea y la complejidad de requerimiento. Para esto se tienen dos estrategias. En la primera, se conoce quien realizará la tarea. Si esto sucede se le pregunta la complejidad de del requerimiento. En la segunda estrategia, no se conoce quien es la persona que realizará la tarea. En este caso el líder calcula la complejidad de la tarea con base en su experiencia y defiendo si es complejo, medio o complicado el requerimiento.

Los modelos de mejora de procesos son de vital importancia en las empresas desarrolladoras de software que se fundamentan en la capacidad del manejo de sus modelos de procesos para generar software. En ocasiones son utilizados de una manera inconsciente, pero el propósito fundamental de los modelos es guiar a las empresas a la madurez del proceso actual para obtener una mejora en el desarrollo del software. Según Gómez-Gil (2007), “Los modelos de procesos son guías que presentan las mejores prácticas para desarrollo del producto en cuestión. Su

propósito es guiar a las organizaciones en la selección de estrategias de mejora, determinando la madurez del proceso actual, e identificando puntos importantes a atacar para mejorar tanto el proceso como la calidad de software." La mejora de procesos es una cultura que se aplica para mejorar la calidad del desarrollo. Ésta puede ser vista de manera internacional o nacional como pueden ser los siguientes casos: SW-CMM, CMMi, PSP, TSP, ISO/IEC 15504, PMBOK y SWEBOK (Ruvalcaba, 2005). Algunas propuestas de investigación como la de Centeno y Gómez-Gil (2010) proponen una manera original de incorporar la mejora de procesos para las PyMEs dedicadas al desarrollo de software, para solventar las problemáticas, las cuales pueden ser reflejadas en la mala calidad de los productos, entregas fuera de calendario y presupuestos rebasados todo ello por no contar con procesos maduros. La mejora de procesos de software se requiere para incrementar la productividad en las compañías. En general, el objetivo es aumentar la calidad del software producido y mantenerlo en un presupuesto y el tiempo. Los modelos de calidad de procesos de software fueron desarrollados en el contexto de las grandes organizaciones y empresas multinacionales. De hecho, existe una tendencia generalizada a resaltar que el éxito de la mejora de procesos de software sólo es posible para las grandes empresas que disponen de recursos suficientes para hacer frente a estos tipos de prácticas. Esta percepción se basa en el hecho de que los programas de mejora de procesos simplemente no son viables para las PyMEs, debido a su estructura organizativa y los altos costos que esto involucra (Hareton, 2001). Sin embargo, la industria del software en la mayoría de países se compone principalmente de PyMEs, que favorecen el crecimiento de las economías nacionales (Conradi y Fuggetta 2002); en algunos países de latino América, la mayoría de las organizaciones son pequeñas, con un comportamiento indisciplinado, con procesos ad-hoc y falta de concentración en el proceso. Es un hecho que las PyMEs se centran principalmente en el producto más que en el proceso, lo que lleva en ocasiones al desconocimiento de los modelos de desarrollo.

MoproSoft

Los intentos de una mejora en la industria del software en México se han reflejado en el desarrollo del "Modelo de Procesos para la Industria de Software" conocido como MoproSoft (Oktaba, 2006). Este modelo presenta una estructura para definir nuevos procesos, la documentación de elementos, enfocándose a las pequeñas y medianas empresas o a pequeños grupos de desarrollo de software que formen parte de una organización más grande (Astorga, Espinoza y Flores 2006).

Se tienen diversos casos de estudio de la aplicación del modelo. Por ejemplo, en (Flores et.ál., 2008), se siguió la siguiente metodología para su implementación: 1) identificación de los factores de cambio que permiten la concientización del grupo de trabajo de la empresa; 2) el diagnóstico del estado real de la empresa, 3) aplicación de los 9 procesos obligatorios requeridos por el modelo, y 4) la evaluación oficial del cumplimiento de la capacidad de Moprosoft. E n este proceso se identifican dos fases: primero una autoevaluación detectándose áreas de oportunidad a cubrir, como fueron la mejora de los procesos en la gestión del negocio, la asignación de responsabilidades en los 9 niveles que marca Moprosoft y la segunda etapa el ser evaluados por NYCE de acuerdo a la norma NMX-I-059-NYCE-2005).

El interés de las PyMEs en la adquisición de esta certificación se enfoca en mejorar sus procesos de software como una estrategia para asegurar la calidad de sus productos. Además, Moprosoft les es de utilidad como un instrumento para asegurar la auditoria de otros estándares, como puede ser en el caso de ISO 9000:2000 que cubre en un 92% y un alcance de 77% de las especificaciones generales y prácticas de CMMI en un nivel 2 (Rivera y Montero, 2004). Oktaba y colaboradores (2006) llevaron un registro de la mejora en aquellas empresas mexicanas donde se ha aplicado el modelo, evaluando sus procesos en un rango de 0 a 1 la primera vez, antes de adoptar a Moprosoft, después del entrenamiento en el mismo.

Cuando las empresas se evaluaron por segunda vez, éstas lograron un aumento promedio de 1.08 en el nivel de capacidad de todos sus procesos. La última

columna representa el promedio de la mejora del proceso, enfatizando la relación entre el esfuerzo por persona y el promedio de la mejora del proceso.

Algunas empresas han adoptado algunos modelos propuestos por el SEI (SEI, 2006), el CMMI o los de ISO (ISO, 2004); (ISO, 2007). Sin embargo, estos no han podido adaptarse a la cultura de las empresas y tienden a no funcionar. Por otro lado, la implantación de este tipo de modelos es costosa, por lo que para empresas donde se cuenta con 10 o menos programadores no es viable.

Compañía	Empleados	Total, de esfuerzo (Horas)	Esfuerzo por persona (Horas)	Promedio de Mejora del proceso
A	17	479	28.18	1.00
B	8	199	24.88	1.00
C	17	628	36.94	1.56
D	29	221	7.62	0.78
Promedio	18	383	21.28	1.08

Tabla 3. Experiencia de Mejora usando Moprosoft (Oktaba, 2007).

La incorporación de Moprosoft con métodos agiles han dado resultados a nivel Latino América ya que las pequeñas y medias compañías tienen problemas con la ausencia de un modelo de desarrollo. Entre estos trabajos podemos encontrar a Competisoft (Oktaba, 2007) que tiene como objetivo el incrementar el nivel de competitividad de las PyMEs Iberoamericanas productoras de software, mediante la creación y difusión de un marco metodológico común. Ajustado a las necesidades específicas de la región, Competisoft pueda llegar a ser la base sobre la que establecer un mecanismo de evaluación y certificación de la industria del software reconocido en toda Iberoamérica. La novedad en este modelo es la incorporación de la filosofía de métodos agiles para la entrega de los productos al cliente, principalmente hace uso de Scrum que permite la evaluación interna de los procesos, basándose en un modelo iterativo e incremental. Siguiendo esta lógica y tomando el siguiente párrafo para justificar el uso de los métodos agiles y modelos de mejora de procesos para el aseguramiento de la calidad en el desarrollo de

software se tiene que: “Usando los principios de Agiles a la hora de diseñar y seleccionar las prácticas CMMI pueden crear actividades en los procesos más aceptables y una adecuados. Además, la incorporación de los objetivos de CMMI en las actividades del proyecto de los equipos Agiles pueden ayudar a que estos equipos sean maduros y/o capaces de gestionar la continuidad de los proyectos. La aplicación de prácticas CMMI en una organización que utiliza métodos ágiles, sin dejar de ser fieles a los principios Agiles, deben mantener un alto nivel de confianza en el conjunto de actividades suficientes para el desarrollo del producto” (Glacer, 2008).

3 Metodología.

La estimación es un proceso orientado al futuro pues se trata de determinar el costo, el tiempo y el esfuerzo en el desarrollo de un sistema antes de su ejecución. Para (Pressman, 2010), la estimación es el proceso de medición anticipada de la duración, esfuerzos y costes necesarios para realizar todas las actividades y obtener todos los productos asociados a un proyecto. Es necesario tener en cuenta numerosos aspectos que afectan a la estimación como la complejidad del proyecto, su estructuración, el tamaño, los recursos involucrados y los riesgos asociados. Para ello es necesario poseer una considerable experiencia en el desarrollo de sistemas.

3.1 Alcance y enfoque de la investigación.

Para el desarrollo de la investigación se pretende desarrollar en un ambiente escolar, teniendo como objeto de estudio a grupos de individuos que construyan software. Ya que una buena práctica para la investigación en el desarrollo del software es aplicar en individuos que lleven a cabo tareas o alguna de las mismas del ciclo de vida del desarrollo de software. Aplicando la clasificación paramétrica y no paramétrica de los métodos de estimación.

3.2 Hipótesis.

A partir de los referentes teóricos y el estudio del estado del arte definidos en las secciones anteriores, se define la hipótesis de investigación y de la metodología a seguir para su comprobación.

Mediante la implementación de un framework de procesos e implementación de metodologías ágiles se logra una propuesta para determinar costos en el desarrollo de software para los desarrolladores novatos.

4 Desarrollo del modelo

El proceso de desarrollo de software está sujeto a la aplicación de procesos propios de la ingeniería del software (levantamientos de requerimientos, análisis, diseño de construcción, pruebas e implementación) que se apegan de manera a la generación de una herramienta informática que cumpla con las particularidades deseadas por el cliente. El validar otros factores que influyan en el desarrollo del mismo, que dependen de la gestión del gerente del proyecto y de su equipo, de cómo serían los tiempos de entrega, alcance del proyecto y tema principal de este estudio el cual sita a que variables se pueden considerar para la estimación de costos asociados al proyecto pues este factor se considera fundamental en el éxito o fracaso del proyecto.

4.1 Planificación de proyecto de software

La estimación comienza con una descripción del ámbito del problema. Luego este se descompone y cada un conjunto de problemas más pequeños y cada uno de estos se estima, usando como guías datos históricos y experiencia.
La complejidad y el riesgo se consideran antes de realizar una estimación final.

1. Formular alcance y complejidad del proyecto
   1.1 Objetivo del negocio
   1.2 Modelo de negocio
   1.3 Objetivos específicos
   1.4 Identificación de participantes
   1.5 Roles
   1.6 Flujo de Proceso
2. Recabar Requerimientos
   2.1 Determinación de Requerimientos
      2.1.1 Requerimientos funcionales
      2.1.2 Requerimientos no Funcionales
      2.1.3 Restricciones del proyecto
3. Modelado del negocio
4. Estimación
   4.1 Cosmic

1. Formulación de Alcance y complejidad del proyecto

Dentro del alcance y la complejidad del proyecto se centra en buscar una definición precisa donde obtendremos la ejecución del proyecto y en ella se den considerar puntos como, la justificación, las características de los resultados esperados, las condiciones a cumplir, se deben considerar los objetivos específicos en concreto, exclusión del proyecto, determinar delimitaciones.

Estos puntos confirman que la definición del alcance del proyecto no se limita a definir su ámbito de aplicación, sino que de igual manera sustrae importantes detalles acerca de los objetivos y los posibles resultados.

Internamente en la planificación del proyecto aplicando el modelo de casos de uso para formular el alcance y complejidad del proyecto se contemplan.

- Objetivos de negocio
- Modelo de negocio
- Objetivos específicos
- Identificación de participantes
- Roles
- Flujo de proceso

1.1 Objetivos de negocio

Los objetivos de negocio determinan las prioridades que se tienden a implementar, para construir un buen objetivo, se debe iniciar por entender el concepto en general del proyecto, para así definir lo que el cliente necesita o lo que quiere lograr.

1.2 Modelo de negocio

Un modelo de negocio se debe describir bajo el contexto del proyecto, donde se tiene que entender el negocio para así direccionar sobre lo que se debe hacer en el desarrollo de software y cómo puede ser utilizado lo mejor posible por la comunidad del negocio, esto con el propósito de generar objetivos que alcancen de manera clara y precisa lo que se quiere plasmar.

1.3 Objetivos específicos

Los objetivos específicos se refieren a los propósitos por los cuales se hace la investigación y cuáles van hacer los alcances de trabajo, así dar claridad a lo necesario que se tenga bien definido el campo de investigación, objeto de estudio y las restricciones.

1.4 Identificación de Participantes

El equipo de desarrollo de software puede ser tan pequeño o tan grande como se requiera. En su estructura más básica, el equipo deberá contar con un mínimo dos integrantes para desarrollos de software de magnitud pequeña los cuales realizarán actividades correspondientes a diferentes roles (Gerente, líder de proyecto, Analista de sistemas, diseñador, Ingeniero de software, responsable de calidad, responsable de pruebas, administrador de la configuración del proyecto) y contemplando de igual al cliente como un integrante del equipo del proyecto.

1.5 Roles

Un rol es determinado en el desarrollo de software para cumplir con responsabilidades y habilidades sobre cómo realizar actividades específicas y desarrollos de artefactos. Los integrantes de un equipo de trabajo generalmente cubren varios roles; sin embargo; el rol describe cómo es el comportamiento del integrante en el negocio y qué responsabilidades tienen.
Los roles pueden ser clasificados por lo general:

- Cliente: El cliente es factor importante en el éxito de un proyecto, ya que su colaboración en el proyecto precisa
- Analistas
- Equipo de Desarrollo
- Probadores
- Directivos
- Otros

1.6 Flujo de Proceso

Dentro de la ingeniería de software se debe definir un proceso de acciones y tareas que encuentran dentro de una estructura o modelo que define su analogía entre sí como en el proceso. Una estructura en la ingeniería de software está compuesta por actividades (planeación, comunicación, modelado, construcción y despliegue), asimismo se lleva un control del proyecto, administración de riesgos y un control de calidad, dichas actividades se organizan a través de flujos de procesos.

- Flujo de proceso lineal
- Flujo de proceso iterativo
- Flujo de proceso evolutivo
- Flujo de proceso paralelo

2. Requerimientos

En el desarrollo de un proyecto de software durante su planificación el proceso de recopilar y verificar las necesidades del cliente es llamado ingeniería de software, con el propósito de entregar una recopilación de requerimientos que apunten a la comprensión de lo que se requiere en el proyecto a desarrollar. Por otra parte, una mala definición de los requerimientos tiende a un alto precio, definiendo requerimientos incompletos o contradictorios entre los que se pueden destacar:

- Problemas de entendimiento
- Costos erróneos
- Mala Calidad
- Retraso en la entrega
- Integrantes del equipo agotados
- Problemas de alcance
- Problema de volatilidad
- Clientes insatisfechos

Los requerimientos se deben de caracterizar por su combinación compleja de diferentes personas que pertenecen a la organización y al entorno donde se va a utilizar el software. Deben ser lo más claros que se pueda y cuantificables en medida de lo posible, indicando la funcionalidad y que características va a tener el sistema resultante.

2.1.1 Determinación de los Requerimientos

Los requerimientos tienden a clasificarse en requerimientos funcionales y requerimientos no funcionales.

2.1.2 Requerimientos Funcionales

Los requerimientos funcionales describen las funciones que el software debe realizar para sus usuarios: aceptar, verificar y registrar datos, transfórmalos, presentarlos y ejecutarlos, clasificándolos en requerimientos de usuario y requerimientos de sistema, este primero haciendo énfasis al comportamiento del

sistema que generalmente se determinan en casos de uso, y los requerimientos de sistema establecen con detalle los servicios y restricciones del sistema.

2.1.3 Requerimientos no Funcionales

Los requerimientos no funcionales consisten en restricciones impuestas por el entorno y la tecnología, especificación de respuesta, requisitos en cuanto a la interfaz, facilidad del mantenimiento, extensibilidad, calidad, rendimiento disponibilidad.

(Sommerville, 2006) desglosa los requerimientos no funcionales en tres grupos:

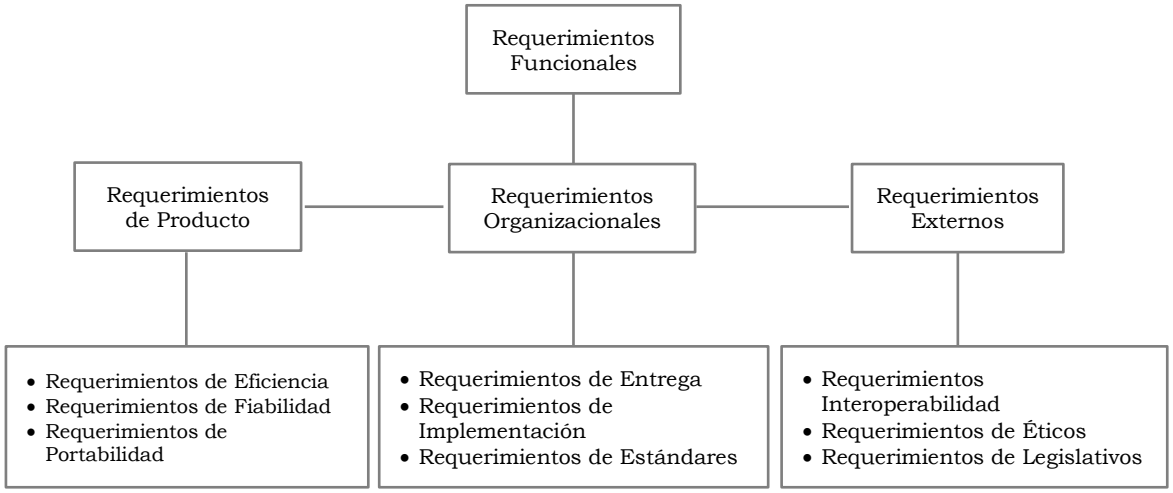

Figura 10. Tipos de requerimientos no funcionales

- Requerimientos de Producto: Especifican el comportamiento del producto *(rapidez de la ejecución, capacidad de memoria, fiabilidad).*
- Requerimientos Organizacionales: Derivan de políticas y procedimientos existentes en la organización del cliente *(Estándares de procesos, métodos de diseño, lenguajes de programación, métodos de entrega).*
- Requerimientos externos: Se derivan de factores externos al sistema y de sus procesos *(Requisitos de interoperabilidad, legislativos, éticos).*

Para obtener información de los requerimientos que debe cumplir el software es necesario recurrir a: entrevistas, documentación sobre el sistema actualmente existente, a usuarios que intervienen en la organización y a sistemas parecidos que se encuentren en el mercado. Dando en seguimiento a un proceso para definir y validar los requerimientos, que cumplan las condiciones o especificaciones solicitadas por el cliente.

Figura 11. Proceso de Gestión de requerimientos

Elicitación: En esta parte del proceso se identifican, se entienden los requerimientos del sistema a desarrollar. El descubrimiento de los requerimientos implica entender el dominio de la aplicación, los problemas que va a resolver y las necesidades de los usuarios de la aplicación. El trabajo que se realiza en la elicitación de requerimientos se derivan los siguientes puntos:

- Establecimiento de Objetivos *(Análisis de los objetivos de los negocios, Formulación de problemas, Establecimientos de las restricciones).*
- Entendimiento del dominio
- Organización del conocimiento
- Recolección de Requisitos

Análisis de Requerimientos: Es proceso por el cual se tiene una comprensión precisa de los requerimientos, donde se analizan por parte del cliente de tal forma que se obtiene el documento de definición de los requerimientos validado.

El análisis de requerimientos comprende una serie de actividades

- Analizar los requerimientos funcionales (RF)
- Agrupar y clasificar los requerimientos funcionales
- Determinar de los requerimientos clasificados: los que no son necesarios, no son factibles y los que están repetidos.
- Aprobar lista de requerimientos funcionales definitivos por parte del experto.
- Estructurar el documento de definición de requerimientos (DDR).
- Elaborar el documento de definición de requerimientos con la lista de los requerimientos funcionales, el cual debe estar aprobado por el cliente.

Especificación de requisitos: Proceso que hace referencia a definir y documentar los requisitos funcionales y no funcionales, identificar atributos de calidad, requerimientos importantes y restricciones.

En esta fase se elaboran tres tipos de documentos:

- Documento de definición del sistema: Define los requisitos del sistema de alto nivel, además se incluye información de los objetivos del sistema, declaración de limitaciones y los requisitos no funcionales.
- Documento de requisitos del sistema: En este documento se manifiestan lo que requieren los desarrolladores del sistema.
- Documentos de requisitos de software: contiene una descripción completa de las necesidades y funcionalidades del sistema que se va a desarrollar a demás determina el alcance del sistema.

Validación de los requisitos: Los requisitos deben ser validados para asegurar que el equipo de desarrollo haya entendido los requisitos, el proceso de validación implica la revisión de requisitos, prototipo, validación del modelo, pruebas de aceptación, verificación y validez, verificación de consistencia de integridad, calidad.

3. Modelado de Negocio

Crear modelos tiene como finalidad entender mejor la entidad real de lo que se va a construir, debe ser capaz de representar la información que el software transforma, la arquitectura y las funciones que permitan que esto ocurra. Los modelos deben cumplir los objetivos los objetivos en diferentes niveles de abstracción, en primer lugar, con la ilustración del software desde el punto vista del cliente y posterior a un nivel más técnico.

Derivar los requisitos funcionales a partir del modelo del negocio, en la realización de casos de uso, se obtienen las actividades como se refleja en el esquema de flujo de trabajo de requisitos.

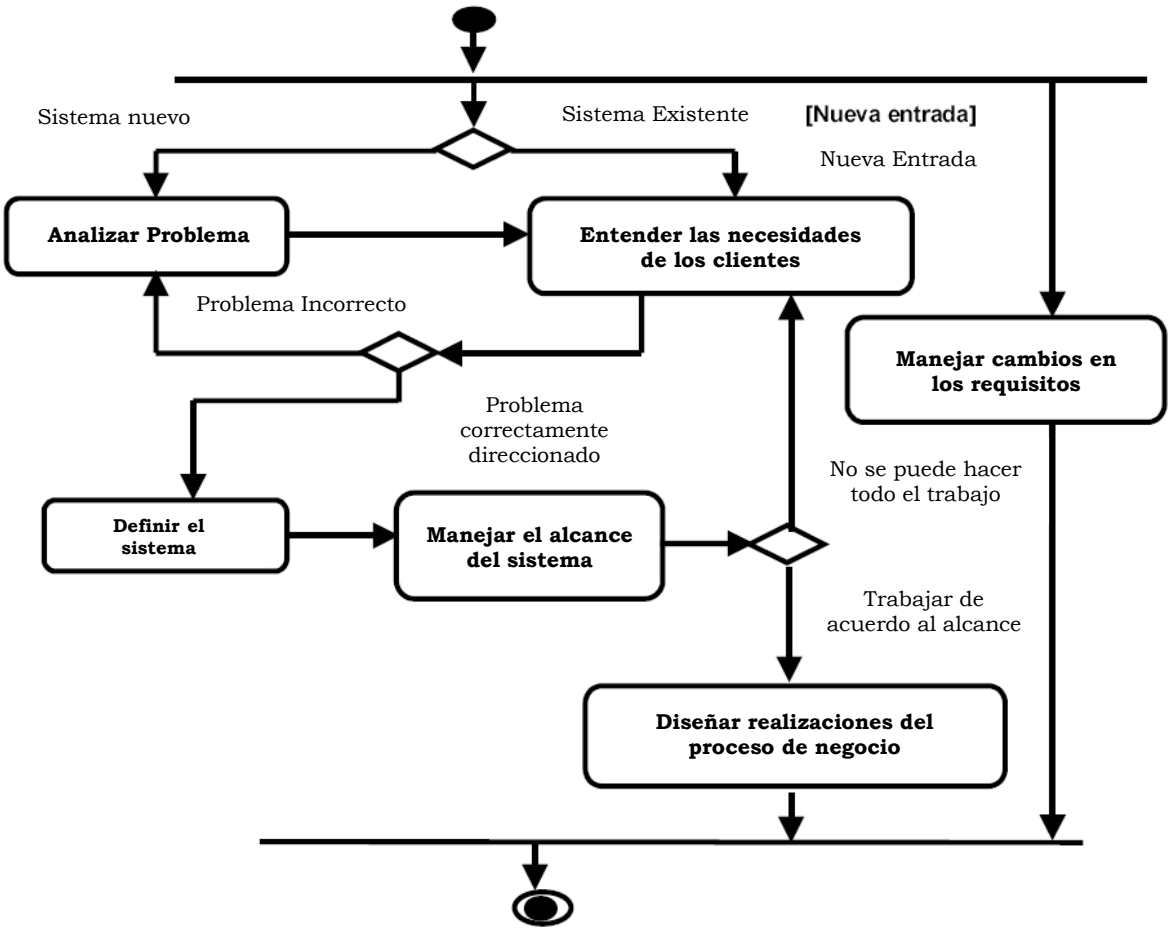

Figura 12. Esquema de flujo de trabajo de requisitos.

4. Estimación

La estimación de los costos es un proceso iterativo, el resultado obtenido de la estimación se incrementa a medida que transcurre el proyecto. Generalmente el estimado de costos de un proyecto es expresado en unidades monetarias, en algunos casos se pueden utilizar otras unidades de medición como las horas y los días de trabajo, estos estimados de derivan a través del tipo de costo que se requiera reflejar como lo son:

- *Costos directos:* aquellos que se identifican por medio de la definición de un objetivo, bien sea producto, servicio o proyecto.
- *Costos indirectos:* son aquellos que la organización tiene que recurrir como operaciones rutinarias.
- *Costos fijos:* estos se generan de manera independiente a nivel de producción o carga de trabajo.
- *Costos Variables:* son costos que dependen del nivel de producción.

Identificar el tipo de desarrollo al que se orientara la aplicación en este caso se orientara al desarrollo de aplicaciones móviles o web. En la fase temprana determinar un estimado de costo el cual como todo proyecto de software puede modificarse debido al incremento de requerimientos no previstos por el usuario. La UIT (Unión Internacional de Telecomunicaciones) estima que hay más de 6 (seis) mil millones de clientes de telefonía móvil en todo el mundo. Según Gartner, 1.750 millones de personas con teléfonos móviles con capacidades avanzadas; también prevé un mayor crecimiento de esta tecnología en los próximos años [4]. Existe una tendencia general hacia el aumento de usuarios conectados a la red a través de dispositivos móviles que, en consecuencia, crearán una demanda creciente de información, aplicaciones y contenido para dichos equipos.

El hecho de que este es un nuevo escenario que está surgiendo con nuevos requisitos y restricciones requiere una revisión del conocimiento actual de los procesos de planificación y construcción de sistemas de software. Estos nuevos sistemas tienen diferentes características y, por lo tanto, un área en particular que exige dicha adaptación es la estimación de software. Los procesos de estimación, en general, se basan en las características de los sistemas, tratando de cuantificar

la complejidad de su implementación. Por esta razón, es importante analizar los métodos actualmente propuestos para proyectos de software y evaluar su aplicabilidad a este nuevo contexto de informática móvil.

En los últimos años, la complejidad y el tamaño de las aplicaciones móviles han aumentado y el desarrollo de aplicaciones móviles de alta calidad requiere un enfoque de ingeniería sistemático y la identificación de herramientas de gestión específicas. La estimación del esfuerzo es una actividad clave de gestión del proyecto necesaria para la planificación del proyecto, la estimación de los recursos del personal, la estimación del costo, el control de calidad y la evaluación comparativa (Sommerville, 2006).

Para el software tradicional, se han definido varios enfoques para respaldar esta tarea que se pueden dividir en dos categorías principales, paramétricos y no paramétricos.

En términos generales, los métodos no paramétricos implican el juicio de expertos humanos, que proporcionan una predicción basada en su experiencia previa. Por otro lado, los enfoques paramétricos se basan en la definición de un conjunto de factores de costo utilizados como variables independientes en modelos de predicción destinados a estimar una variable numérica, por ejemplo, el número de horas / hombre o el esfuerzo requerido para desarrollar / mantener un software.

Una de las ventajas de los enfoques paramétricos es que son más replicables. Sin embargo, dependen críticamente de la identificación y evaluación de los factores de costo. Se pueden diseñar diferentes enfoques basados en los factores de costo empleados y cada uno se puede aplicar en una fase diferente del proceso de desarrollo, una vez que la información para evaluar los controladores de costos requeridos esté disponible.

Actualmente, no se ha dedicado mucho trabajo para identificar los enfoques adecuados para la estimación del esfuerzo de las aplicaciones móviles, lo que puede ser particularmente difícil para los directores de proyectos debido a los nuevos enfoques de desarrollo y programación adoptados (Wasserman, 2010). En el caso del trabajo de (Catalino, Salza, Gravino, & Ferrucci, 2017) es llenar la brecha

mediante la propuesta de un conjunto de factores de coste que se emplean para el modelo basado en la estimación del esfuerzo de aplicaciones para móviles. La metodología adoptada para obtener la propuesta se inspiró en un trabajo similar llevado a cabo por Mendes et al. en el contexto de las aplicaciones web (Mendes, Mosley, & Counsell, 2003) Los factores de costo reunidos por Mendes et al. determinó la creación de un conjunto de datos, llamado TUKUTUKU, que se ha empleado en varias investigaciones (Kitchenham & Mendes, 2004)- (Corazza, y otros, 2013). De manera similar a Mendes et al., El primer paso de esta metodología consistió en analizar los formularios de cotizaciones en línea puestos a disposición por las compañías de software con el fin de extraer un conjunto inicial de métricas. Luego, involucramos a cuatro gerentes de proyecto que tienen una buena experiencia en la administración y desarrollo de aplicaciones móviles con el objetivo de validar el conjunto inicial de métricas derivadas durante la primera fase. En particular, a partir de un conjunto de 48 indicadores, 36 de ellos fueron confirmados como los que tuvieron un mayor impacto en el esfuerzo de un proyecto, mientras que 12 fueron descartados. En cuanto al uso de FPA, el IFPUG propuso una guía que explica cómo adoptar este método en el contexto de las aplicaciones móviles (Preuss, 2013). Mientras que estudios recientes propusieron un conjunto de pautas para un tamaño aproximado y rápido de aplicaciones móviles en términos de COSMIC (D'Avanzo, Ferrucci, Gravino, & Salza, 2015); (Heeringen & Gorp, 2014) y (Sellami, Haoues, Rusli, & Ibrahim, 2014). En particular, Sellami et al. [15] definió el "tipo de acción", una forma de simplificar el recuento del número de Puntos de Función COSMIC mediante la asignación de los tipos de acciones que caracterizan los movimientos de datos que se determinarán al aplicar COSMIC [9]. van Heeringen y van Gorp [13] introdujeron un conjunto de suposiciones (por ejemplo, considerar una aplicación como una capa de presentación, sin almacenamiento persistente) relacionada con las características de la aplicación a medir, lo que permite la estimación del tamaño en términos de los Puntos de Función COSMICOS. Sucesivamente, Ferrucci et al. propuso y evaluó (D'Avanzo, Ferrucci, Gravino, & Salza, 2015), un nuevo conjunto de directrices capaz de ayudar a medir aplicaciones de negocios móviles, que contienen almacenamiento persistente como una base de datos interna (es decir, es accesible mediante movimientos de datos de Lectura y Escritura). Otros métodos centraron

su atención en la medición de aplicaciones de juegos (Abdullah, Rusli, & Ibrahim, 2014). El uso de COSMIC para medir aplicaciones complejas compuestas por arquitecturas móviles y basadas en la nube también fue considerado por Cruz et al. y Ferrucci et al. (Ferruci, Gravino, & Pasquale, 2017). La limitación de los enfoques anteriores es que pueden emplearse una vez que los requisitos funcionales del usuario estén bien documentados, por lo tanto, solo después de que se haya completado la fase de ingeniería de requisitos. De manera diferente, nuestro objetivo es identificar e investigar los factores relevantes que se pueden estimar en las primeras fases del desarrollo de software, de forma similar a los controladores de costo TUKUTUKU (Mendes, Mosley, & Counsell, 2003) donde se realizó una encuesta (S1) utilizando un motor de búsqueda para obtener formularios de cotización de proyectos web empleados por compañías web de todo el mundo para proporcionar presupuestos iniciales en proyectos de desarrollo web. Los 133 formularios de cotización de proyectos Web recopilaron datos sobre métricas de tamaño inicial, factores de costo, contingencia y posiblemente métricas de ganancias. Estas métricas se organizaron en categorías y se clasificaron. Los resultados indicaron que las dos métricas de tamaño inicial más comunes utilizadas para la estimación de costos web fueron "número total de páginas web" (70%) y "qué funcionalidad proporcionará a la aplicación" (66%). El objetivo del primer paso del estudio fue el análisis de las cotizaciones en línea disponibles por las empresas en la web, con el propósito de extraer un conjunto inicial de métricas basadas en la información solicitada por las empresas. El contexto del estudio consistía en que cada compañía tuviera un sitio web y proporcionara un formulario en línea para solicitar un presupuesto sobre el desarrollo de una aplicación móvil.

Usamos una herramienta de búsqueda automática, llamada GOOGLE SCRAPER, que está disponible públicamente y de código abierto en GitHub. En particular, la herramienta se comporta como un motor de búsqueda de Google y recibe como entrada la consulta para buscar y una cantidad máxima de enlaces a la mía. Cuando diseñamos la consulta, primero incluimos todos los términos que posiblemente hacen referencia a aplicaciones móviles (por ejemplo, "aplicación Android" o simplemente "aplicación"). Entonces, incluimos los términos relacionados con la presencia de cotizaciones en línea, como el "precio estimado" o

simplemente "cotización". Finalmente, consideramos sinónimos y abreviaturas. La consulta final fue la siguiente:

(“quote” OR “quote form” OR “price estimate”)
(“mobile application” OR ”mobile app” OR
”smartphone app” OR ”smartphone application” OR
”android app” OR ”android application” OR ”ios app”
OR ”ios application” OR ”windows phone app” OR
”windows phone application”)

El resultado de la consulta consistió en una lista de enlaces que validamos manualmente. El objetivo de la validación era filtrar todos los enlaces que no estaban relacionados con las cotizaciones en línea, para tener un conjunto inicial de métricas. Para este objetivo, primero descartamos los enlaces que carecían de cualquier tipo de cita dentro de la página web, mientras que en un segundo paso descartamos los enlaces que presentaban citas genéricas que no proporcionaban ninguna información útil en nuestro contexto (por ejemplo, la información del cliente o la descripción genérica del proyecto). ser desarrollado). El proceso de validación fue realizado y verificado por dos de los autores. El resultado de esta fase consistió en un conjunto de métricas que se emplearon en la fase de validación que se describe en la siguiente subsección.

El objetivo del segundo paso del estudio fue validar el conjunto inicial de métricas por expertos que tenían un buen conocimiento de los métodos de estimación del esfuerzo y las aplicaciones móviles. El objetivo fue explotar a los expertos involucrados para (i) confirmar / refutar la utilidad de los indicadores de métrica / costo para estimar el esfuerzo en la fase inicial de desarrollo de aplicaciones móviles, definidas durante el primer paso del proceso y (ii) posiblemente descubrir nuevos factores que no fueron revelados después del primer análisis. El contexto del estudio fue compuesto por cuatro gerentes de proyecto con más de 4 años de experiencia en la gestión del desarrollo móvil y la estimación del esfuerzo.

Con el objetivo de reunir las opiniones de los participantes y proporcionar una solución conjunta, adoptamos el método Delphi (Linstone & Turoff et al., 1975). El método Delphi es una técnica de comunicación estructurada, originalmente

desarrollada como un método de pronóstico sistemático e interactivo que se basa en un panel de expertos. Los expertos responden cuestionarios en dos o más rondas. Después de cada ronda, un facilitador proporciona un resumen anónimo de todos los juicios. Como consecuencia, se alienta a los expertos a revisar sus respuestas anteriores a la luz de las respuestas de otros miembros. El proceso se detiene después de un criterio de detención predefinido (por ejemplo, número de rondas, logro de consenso, estabilidad de los resultados).

En nuestro caso, primero procedimos con el diseño de un cuestionario en línea utilizando la plataforma Google Form. Una vez que todos los participantes completaron el cuestionario, las diferentes opiniones se han recopilado y agrupado en un solo documento. Finalmente, dicho documento fue enviado a los expertos, quienes tuvieron la oportunidad de expresar opiniones más recientes en base a las respuestas proporcionadas por los otros participantes. Si se encontró una solución común en esta etapa, el proceso finaliza. De lo contrario, el proceso se reiniciaría hasta que se haya encontrado una solución común. El primer autor de este documento ha desempeñado el papel de facilitador. Específicamente, los pasos se detallan a continuación:

Fase 1: inicialmente se contactó a cada participante por correo electrónico que resumía el propósito del trabajo con un documento explicativo que incluía: (i) una breve explicación del objetivo del trabajo y (ii) la lista de métricas con una descripción. Las instrucciones para completar el cuestionario también fueron incluidas. El cuestionario se compone de tres partes:

1) Pre-cuestionario: un pre-cuestionario dirigido a recolectar información general sobre los antecedentes de los participantes;
2) Evaluación de métricas: para cada métrica, se pidió a los participantes que evaluaran el nivel de importancia para la estimación temprana del esfuerzo utilizando una intensidad de escala Likert [20] de 1 ("nada") a 5 ("mucho");
3) Sugerencias: se les pidió a los participantes sugerir posibles adiciones, eliminaciones o cambios en las métricas.

Fase 2: una vez recibidas las respuestas del cuestionario, el facilitador analizó las opiniones surgidas. En cuanto a la segunda parte del cuestionario, es decir, la evaluación de las métricas, calculamos la media, mediana, mínima, máxima y desviación estándar de los puntajes asignados por los participantes a cada métrica. Con respecto a la tercera parte del cuestionario, agregamos las respuestas sobre las opiniones de los gerentes de proyecto considerando el porcentaje de participantes que creían que las métricas contenidas en una categoría determinada (por ejemplo, las "Características") eran significativas. También recopilamos más opiniones sobre la adición / modificación / eliminación de métricas en forma de preguntas abiertas, con el fin de comprender la lógica detrás de las elecciones de los participantes;

Fase 3: Una vez analizado el cuestionario, el facilitador reunió los resultados en un único documento y se lo envió a los expertos, quienes tuvieron la oportunidad de expresar opiniones más recientes en función del juicio proporcionado por los otros participantes. Este paso fue la parte más delicada del estudio porque tuvimos que recopilar y fusionar todas las opiniones diferentes para crear una solución común. Por esta razón, proporcionamos a los participantes un nuevo cuestionario en línea que contiene todas las opiniones recogidas. El objetivo era impulsar a los expertos a centrarse en las opiniones de los demás participantes para llegar a un consenso sobre el conjunto final de métricas.

La industria de TI cambia rápidamente y surgen nuevos tipos de hardware, arquitecturas de software y aplicaciones a un ritmo rápido. Las empresas de desarrollo de software se enfrentan al hecho de que necesitan adaptarse para poder ofrecer nuevos tipos de proyectos de desarrollo de software. Necesitan tener las personas adecuadas (conocimientos y habilidades) en su organización para adaptarse al nuevo mundo, pero también deben ser competitivos y tratar de limitar sus riesgos a la hora de cotizar para proyectos. La estimación del software, basada en el tamaño funcional, sigue siendo una actividad importante, también en el "nuevo mundo" del desarrollo de software.

Aunque las aplicaciones móviles pueden considerarse como un tipo moderno de arquitectura cliente-servidor, parecen ser diferentes de las aplicaciones tradicionales, debido a una serie de razones, algunas de ellas son:

El usuario puede interactuar con las aplicaciones en más formas que en las aplicaciones tradicionales. Por ejemplo, la aplicación puede reaccionar al cambio de posición (alternar) del dispositivo móvil, agitar el dispositivo móvil y algunas aplicaciones pueden aceptar mensajes de voz (por ejemplo, Siri);
Algunas aplicaciones también se comportan en eventos en tiempo real, como el movimiento del dispositivo móvil, el cambio de Wi-Fi a celular y viceversa o cuando la red está fuera de alcance.

Algunos de los datos que la aplicación está utilizando se almacenan en el dispositivo móvil, y algunos pueden estar en la nube o en un sistema de fondo. La arquitectura exacta puede ser muy diferente para cada aplicación;

Las aplicaciones deben tener una funcionalidad que maneje las interrupciones, como una llamada entrante.

Por lo general, existen requisitos no funcionales importantes que una aplicación debe cumplir, p. seguridad, rendimiento, tráfico de datos mínimo, espacio ocupado en el dispositivo y consumo de energía de la batería.

Existe una publicación disponible que cubre la medición del tamaño funcional de las aplicaciones móviles utilizando el método IFPUG (IFPUG, 2017), publicado en la guía IFPUG para TI y medición de software. Este documento se centra en la forma en que se puede llevar a cabo una medición de tamaño funcional utilizando el método COSMIC y propone un método aproximado para dimensionar aplicaciones móviles de forma rápida y precisa.

Método COSMIC

Sogeti ha desarrollado un método COSMIC aproximado para determinar el tamaño funcional de una aplicación móvil de forma rápida y precisa. Este método se describe en esta sección. El método COSMIC ofrece un método estandarizado de medición de un tamaño funcional del software de los dominios conocidos

comúnmente como 'aplicación de negocios' de software, software en 'tiempo real', software de 'infraestructura' y algunos tipos de científicos / software de ingeniería.

Supuestos básicos

El método COSMIC está diseñado para ser aplicable para medir la funcionalidad del software de los siguientes ámbitos:

Aplicaciones Software de Gestión, las cuales son las típicamente utilizadas para dar soporte a la administración de negocios, tales como banca, seguros, contabilidad, personal, compras, distribución o fabricación. Este software está a menudo caracterizado como "rico en datos", ya que está centrado principalmente en la necesidad de gestionar grandes cantidades de datos acerca de aspectos del mundo real.

Software en tiempo real, cuya tarea es mantener o controlar acontecimientos que están sucediendo en el mundo real. Algunos ejemplos serían el software para centrales telefónicas y de intercambio de mensajes, el software incluido en dispositivos para el control de máquinas tales como los electrodomésticos, ascensores, motores de automóviles y aeronaves, para el control de procesos y la adquisición automática de datos, y software de los sistemas operativos de los ordenadores.

El software de infraestructura en apoyo de lo anterior, tales como componentes reutilizables, controladores de dispositivos. Algunos tipos de software científico / ingeniería.
El método aproximado para medir el tamaño funcional de las aplicaciones móviles se basa en el método COSMIC. Las siguientes suposiciones básicas adicionales y / o explícitas también se aplican a este método:

Se considera que una aplicación móvil es una capa de aplicación que se desarrolla sobre una o más capas de datos (Common Software International Measurement Consortium, The COSMIC Functional Size Measurement Method, version 3.0.1, Measurement Manual, sección 2.2.4). Si tal capa de datos se almacena físicamente en el dispositivo móvil o en un servidor central (o en una combinación de los mismos) no tiene relevancia para la medición; Lógicamente, no se almacenan datos

persistentes en la capa de la aplicación. Almacenar datos en el dispositivo móvil para reducir el tráfico de datos o posibilitar el trabajo con la aplicación cuando ninguna conexión de red está disponible se considera una solución técnica. Como resultado de esto, solo se identifican los movimientos de datos de entrada y salida al aplicar este método;

Dentro de una aplicación móvil, los procesos funcionales pueden usar ciertos datos de proceso que están espontáneamente presentes, p. fecha y hora del sistema, la ubicación actual del GPS, etc. (COSMIC implementation guide for ISO/IEC 19761: 2003, sección 4.1.2, regla de entrada C);

Las aplicaciones móviles se consideran aplicaciones comerciales. Por lo tanto, se debe identificar un movimiento de datos de salida para todos los mensajes de error de la aplicación en cualquier proceso funcional (Common Software International Measurement Consortium, The COSMIC Functional Size Measurement Method, version 3.0, sección 4.4.7);

Como el mensaje de error de la aplicación puede provenir de la capa de datos, se debe identificar un movimiento de datos de entrada para todos los mensajes de error de la aplicación en cualquier proceso funcional (Common Software International Measurement Consortium, The COSMIC Functional Size Measurement Method, version 3.0.1, sección 3.2).

El método de medición COSMIC consiste en la aplicación de un conjunto de modelos, principios, reglas y procesos a los Requisitos Funcionales de los Usuarios (o Functional User Requirements, FUR) de una determinada aplicación software. El resultado es un “valor de una cantidad” numérico (tal y como se define en ISO), que representa el tamaño funcional de la aplicación software de acuerdo con el método COSMIC. Tenga en cuenta que el método COSMIC reconoce que algunos tipos de requerimientos (por ejemplo, la calidad y las restricciones ambientales) pueden expresarse temprano en la vida de un proyecto de software como requerimientos ‘No Funcionales’, de acuerdo a la definición de la ISO. Sin embargo, estos mismos requisitos pueden evolucionar a medida que el proyecto avanza en Requerimientos Funcionales de Usuario.
El tamaño funcional medido por el método COSMIC está diseñado para depender

sólo de los FUR del software que va a ser medido y ser independiente de cualquier requerimiento o restricciones relativas a la aplicación de los FUR. 'Funcionalidad' puede ser vagamente definido como 'el procesamiento de la información que el software debe realizar para sus usuarios'.

El Modelo Genérico de Software

Después de haber identificado y definido los FUR del software a ser medido en términos del "Modelo Contextual del Software", debemos ahora aplicar el Modelo Genérico del Software a los FUR para identificar los componentes de la funcionalidad que se medirán. Este Modelo Genérico de Software asume que los siguientes principios generales son ciertos para cualquier software que puede ser medido con el método.

CFP (Punto Funcional Cosmic, Cosmic Function Point), que es el tamaño de un movimiento de datos.

El tamaño de un proceso funcional es igual al número de sus movimientos de datos.

El tamaño funcional de una pieza de software de alcance definido es igual a la suma de los tamaños de sus procesos funcionales.

El tamaño de los cambios necesarios en una pieza de software se mide de la siguiente manera:

- El tamaño de cualquier movimiento de datos afectado (es decir, que hay que añadir, modificar o borrar) por el cambio requerido se mide por convención como uno CFP.
- El tamaño de los cambios requeridos en una pieza de software es igual al número de movimientos de datos que se ve afectada por los cambios requeridos.
- El tamaño mínimo de un cambio en una pieza de software es 1 CFP.

El proceso de medición COSMIC se compone de tres fases:

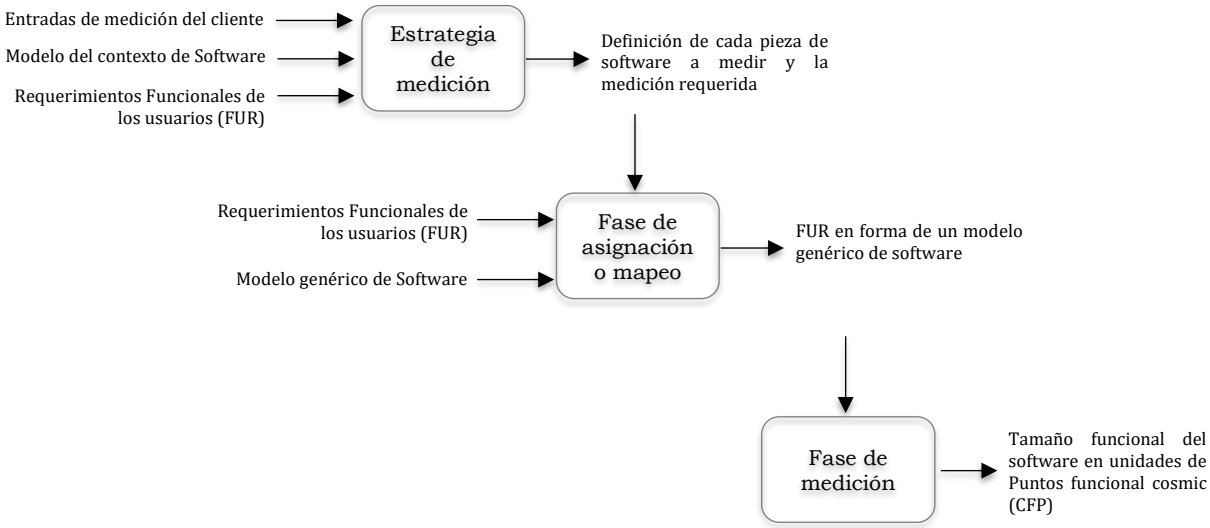

Figura 13. Proceso de medición del método COSMIC.

A. Estrategia de medición.

La fase de estrategia de medición para este método aproximado no difiere, en principio, de la fase de estrategia de medición 'estándar' del método de medición del tamaño funcional COSMIC. Para este método aproximado, también es necesario considerar el propósito y el alcance de la medición e identificar los usuarios funcionales y el nivel de granularidad que se debe medir, como se describe en el capítulo 2 del documento "Common Software International Measurement Consortium, The COSMIC Functional Size Measurement Method, version 3.0.1, Measurement Manual".

Es evidente que el medidor de una pieza de software debe decidir, en función de la finalidad de la medición, cuándo medir (antes, durante o después del desarrollo), qué medir (por ejemplo, todo el software que se entrega en un proyecto, o excluir el software reutilizado) y cuáles artefactos utilizar para derivar los FUR a ser medidos (por ejemplo, una especificación de requisitos o el software instalado).

Los siguientes son los propósitos de medición típicos

- Para medir el tamaño de los FUR a medida que evolucionan, como entrada

a un proceso para estimar el esfuerzo de desarrollo.

- Para medir el tamaño de los cambios a los FUR después de que se hayan establecido
- inicialmente, con el fin de gestionar el alcance 'scope creep'.
- Para medir el tamaño de los FUR del software entregado como entrada para la medición del desempeño de la organización de desarrollo.
- Para medir el tamaño de los FUR del software entregado, y también el tamaño de los FUR del software que fue desarrollado, con el fin de obtener una medida de la reutilización funcional.
- Para medir el tamaño de los FUR del software existente como entrada para la medición del desempeño del grupo responsable de mantener y dar soporte al software.
- Para medir el tamaño de algunos cambios a (los FUR de) un sistema de software existente como una medida del tamaño del trabajo a realizar de un equipo de proyecto de mejora.
- Para medir el tamaño del subconjunto de la funcionalidad total del software que debe ser desarrollado, y que se proporcionará a los usuarios funcionales seres humanos del software.

En resumen, el propósito de la medición debe ser siempre utilizado para determinar (a) qué software es incluido o excluido del alcance global y (b) la forma en que el software incluido puede que sea necesario dividirlo en partes separadas, cada una con su propio alcance, que deben medirse por separado.

La necesidad de un nivel de granularidad estándar en las etapas iniciales de un proyecto de desarrollo de software, los requisitos funcionales de los usuarios (FUR) se especifican en ‘alto nivel’, es decir, se obtiene un esbozo, o con pocos detalles. A medida que el proyecto progresa, los FUR, son refinados, (por ejemplo, a través de las versiones 1, 2, 3, etc.), revelando más detalle a un ‘menor nivel’. Estos diferentes grados de detalle de los requisitos actuales (y por tanto derivados de los FUR), son conocidos como diferentes ‘niveles de granularidad’.

Antes de continuar, es importante asegurar que no haya malentendidos sobre el significado de ‘nivel de granularidad’ en el método COSMIC. Detallar los FUR

implica la ampliación de la descripción del software de un nivel de granularidad 'superior' a uno 'inferior' revelando más detalles, pero sin modificar su alcance. Este proceso no debe confundirse con alguno de los siguientes.

- Detallar un software con el fin de revelar sus componentes, subcomponentes, etc. (en diferentes 'niveles de descomposición'- véase la sección 2.2.3 anterior). Tal definición de detalle puede ser necesario si el propósito de medición requiere que el alcance general de la medición sea sub-dividida siguiendo la estructura física del software.

- La evolución de la descripción de algún tipo de software a medida que avanza a través de su ciclo de desarrollo, por ejemplo, de los requisitos al diseño lógico, al diseño físico, etc. Cualquiera que sea la etapa en el desarrollo de algún tipo de software, sólo estamos interesados en su FUR para fines de medición.

- El concepto de 'nivel de granularidad' se aplica únicamente a los requisitos de los usuarios funcionales de software (FUR).

B. Fase de mapeo.

La fase de mapeo para este método aproximado no difiere, en principio, de la fase de mapeo "estándar" del método de medición del tamaño funcional COSMIC. Para este método aproximado, también es necesario identificar el conjunto de procesos funcionales de la pieza de software a medir y los objetos de interés y los grupos de datos a los que hace referencia la pieza de software que se va a medir, como se describe en el capítulo 3 del documento. [1] Sin embargo, es importante llevar a cabo esta fase teniendo en cuenta los supuestos básicos que se describen arriba.

C. Fase de medición.

La fase de medición de este método aproximado consta de dos pasos:

1. Identificar el tipo de cada proceso funcional;
2. Cuantificación de los parámetros para la identificación del tipo de proceso funcional.

Fase 1: Identificar el tipo de cada proceso funcional.

Funcionalidad de vista: la intención principal de este tipo de proceso funcional es presentar datos al menos uno de los usuarios funcionales;

Manipulación de datos: la intención principal de este tipo de proceso funcional es agregar información, cambiar información o eliminar información del almacenamiento persistente de datos;

Función de consulta: en la sección 4.1.5 del documento [8] se explica que un FUR para actualizar o eliminar datos persistentes requiere dos procesos funcionales por separado. Uno para la actualización / eliminación real (un proceso funcional para la manipulación de datos como se describió anteriormente) y otro que lee y muestra los datos que necesitan ser actualizados / eliminados.

Este segundo proceso funcional se caracteriza como un proceso funcional investigación;

Funcionalidad de soporte del usuario: este tipo de proceso funcional puede describirse como una función que muestra una lista desde la cual un usuario puede hacer una selección (por ejemplo, una pantalla de selección, función de selección, lista de selección, cuadro de lista o función emergente). Cuando este tipo de funcionalidad está disponible en una pantalla, pero el usuario no está obligado a usarla (en otras palabras: el usuario puede llenar todos los campos de la pantalla sin utilizar esta funcionalidad), este tipo de funcionalidad debe considerarse como una función separada proceso funcional.

Cuando el usuario elige usar la pantalla de selección no obligatoria, en realidad desencadena un nuevo proceso funcional. Y este proceso funcional se caracteriza como funcionalidad de soporte del usuario. Nota: si la pantalla de selección es la única forma de completar el campo correspondiente, entonces el uso de la pantalla es obligatorio y, por lo tanto, parte del proceso funcional de manipulación de datos en el que se ofrece;

Funcionalidad especial: hay algunos tipos especiales de funcionalidad además de los tipos genéricos antes mencionados que deben distinguirse para este método

aproximado. Estos tipos son:

- Menús generados dinámicamente
- Funcionalidad de inicio de sesión y cierre de sesión
- Funcionalidad de ayuda
- Invocar funcionalidad externa

Fase 2: Cuantificación de los parámetros involucrados para el identificado tipo de proceso funcional.

Cuando se caracterizan todos los procesos funcionales, los parámetros implicados para cada proceso funcional deben cuantificarse para encontrar el tamaño funcional (aproximado) de la aplicación móvil.

Funcionalidad de Vista.

Un proceso funcional de vista básica tiene un valor de 6 CFP:

E	Inicio de Entrada
X	Pregunta para información a la capa de datos
E	Recibiendo información
E	Recibiendo mensaje de error de aplicación
X	Mostrar datos
X	Mostrar mensaje de error de aplicación

Tabla 4. Identificación de cada proceso funcional

El proceso básico incluye la visualización de atributos de datos que pertenecen a un grupo de datos de un objeto de interés. Se obtienen en recibido de la capa de datos y se muestran al usuario funcional.

La capacidad de cambiar la vista de los datos mostrados (por ejemplo, mediante el filtrado o la clasificación de los datos) no conduce a la identificación de movimientos de datos adicionales. Esto solo implica los llamados comandos de control (ver sección 4.1.10 del documento [1]).

Para cada grupo de datos adicional que se muestra, el tamaño del proceso funcional aumenta con 2 CFP:

E	Recibir datos
X	Mostrar datos

Tabla 5. Representación de tamaño del proceso funcional.

Para cada grupo de datos adicional que se puede ver dentro del mismo proceso funcional, se identifican 2 movimientos de datos adicionales. Un movimiento de datos para recibir los datos de la capa de datos y un movimiento de datos para presentar los datos al usuario funcional.

No es necesario un movimiento de datos separado para la pregunta de información a la capa de datos, la pregunta de información tiene los mismos atributos de datos sin importar cuántos grupos de datos estén involucrados.

Se muestra un grupo de datos adicional cuando:

- Se consulta un objeto de interés diferente.
- Se consulta el mismo objeto de interés, pero para un conjunto diferente de atributos de datos. Esto ocurre, por ejemplo, cuando se usa la funcionalidad de vista, por personas con diferentes perfiles de autorización, donde se muestran los atributos de datos diferentes del perfil.
- No se identifica ningún grupo de datos adicional cuando un proceso funcional ofrece varias opciones de visualización para presentar diferentes conjuntos de atributos de datos del mismo objeto de interés para el mismo usuario, donde este usuario puede alternar entre las opciones de visualización presionando los botones (de desplazamiento). seleccionando diferentes pestañas o inclinando el dispositivo móvil.

Cuando un proceso funcional muestra datos transaccionales en una lista (en modo retrato) y le ofrece al usuario la posibilidad de ver más atributos del mismo objeto de interés inclinando el dispositivo móvil, esto todavía tiene que ver con ver el mismo grupo de datos.

Para cada grupo de datos que muestra datos calculados o derivados, el tamaño del proceso funcional se incrementa con 1 CFP:

X	Mostrar datos

Tabla 6. Tamaño incremental para cada proceso funcional.

Si se muestra un grupo de datos que comprende datos calculados y / o determinados, se identifica un movimiento de datos adicional. Un (sub) total bajo una lista de datos recuperados es un buen ejemplo aquí.

Resumen: para cada proceso funcional con la intención principal de presentar datos o al menos uno de los usuarios funcionales, el tamaño funcional se encuentra usando esta fórmula:

4 + (2 * cantidad de grupos de datos derivados de la capa de datos) + (1 * cantidad de grupos de datos con datos calculados y / o determinados)

Funcionalidad de manipulación de datos

Un proceso funcional de manipulación de datos básicos tiene un valor de 4 CFP:

E	Inicio de Entrada
X	Pregunta para información a la capa de datos
E	Recibiendo información
X	Mostrar mensaje de error de aplicación

Tabla 7. Valor que representan los procesos funcionales.

El proceso básico incluye la adición de atributos de datos, el cambio de atributos de datos en la eliminación de atributos de datos que pertenecen a un grupo de datos de un objeto de interés. Los atributos de datos involucrados se proporcionan a la capa de datos, que se encarga de procesarlos.

Para cada grupo de datos adicional que se manipula, el tamaño del proceso funcional aumenta con 2 CFP:

E	Recibir datos
X	Dar información de la capa de datos

Para cada grupo adicional de atributos de datos que se pueden manipular dentro del mismo proceso funcional, se identifican dos movimientos de datos adicionales. Un movimiento de datos para ingresar los datos por parte del usuario y un movimiento de datos para proporcionar los datos a la capa de datos;

Un grupo de datos adicional se manipula cuando:

- Se manipula o manipula un objeto de interés diferente

- Se manipula el mismo objeto de interés, pero el conjunto de atributos de datos difiere de otro conjunto de atributos de datos del mismo objeto de interés que se manipula dentro del mismo proceso funcional. Esto ocurre, por ejemplo, cuando se usa la funcionalidad de manipulación de datos, por personas con diferentes perfiles de autorización, en donde dependiendo del perfil se pueden manipular los diferentes atributos de datos.

- Se reconoce que no es una regularidad que por cada grupo de datos manipulados siempre existirán tanto una entrada y un movimiento de salida de datos. Sin embargo, para este método aproximado, se supone que este es el caso;

- No hace falta decir que, para la funcionalidad para borrar datos, es posible identificar dos grupos de datos del mismo objeto de interés dentro de un proceso funcional.

Para cada grupo de datos que se muestra al usuario dentro del proceso de manipulación, el tamaño del proceso funcional se incrementa con 3 CFP:

- Si se muestran datos adicionales durante el proceso de manipulación de datos, en función de los datos que se ingresan, deben identificarse

movimientos de datos adicionales para solicitar, recibir y visualizar los datos.

Ejemplo: durante la adición de una dirección, el nombre de la calle y la ciudad se muestran, en función del código postal y el número de casa que se ingresan, esto implica ver el objeto de interés de los datos del código postal en el proceso de agregar la dirección.

- Si la funcionalidad de soporte del usuario obligatorio (una pantalla de selección, función de selección, lista de selección, cuadro de lista o función emergente) se ofrece al usuario dentro del proceso de manipulación de datos, esto también se debe considerar como visualización de datos adicionales. Para la funcionalidad de soporte de usuario no obligatorio, se distingue un tipo diferente de proceso funcional.

En esta sección describimos primero el método de medición de tamaño COSMIC y luego cómo lo aplicamos en aplicaciones móviles.

El tamaño funcional se define como el 'tamaño del software derivado de la cuantificación de los requisitos funcionales del usuario (FUR)' [6]. Los FUR describen qué se espera que haga el software para sus usuarios. Algunos ejemplos son la transferencia de datos, la transformación de datos, el almacenamiento de datos y la recuperación de datos.

COSMIC define una medida estandarizada del tamaño funcional del software expresado en unidades de Puntos de Función COSMIC (CFP). La medición está diseñada para ser dependiente solo de los FUR del software que se medirán e independientemente de los requisitos / restricciones sobre su implementación.

Conceptos principales Un proceso funcional es uno de los principales conceptos definidos en COSMIC. Es un conjunto de movimientos de datos que representan una parte elemental de los FUR. Un usuario funcional se define como un (tipo de) usuario que es un remitente y / o un destinatario de datos en el FUR; esto significa que un usuario funcional puede ser un ser humano o, por ejemplo, un dispositivo

externo también. Además, un límite es una interfaz conceptual entre el software que se mide y sus usuarios funcionales. Con estas definiciones, es posible enfocarse en los cuatro diferentes tipos de movimiento de datos: los tipos de Entrada (E) mueven los datos de un usuario funcional a un proceso funcional; Los tipos de salida (X) mueven los datos de un proceso funcional a un usuario funcional; Los tipos de escritura (W) mueven los datos de un proceso funcional a un almacenamiento persistente; Los tipos de lectura (R) mueven datos del almacenamiento persistente a un proceso funcional.

Se asigna 1 unidad CFP por cada movimiento de datos y su suma representa el tamaño de medición. El método COSMIC define un proceso de medición, que consta de tres fases: la Fase de la Estrategia de Medición, la Fase de Mapeo y la Fase de la Medición. Cada uno de ellos se explica a continuación.

Fase de estrategia de medición. Esta es la fase de trabajo preliminar en la que se definen los parámetros clave de la medición. Algunos de ellos son: el propósito, que define para qué se usará el resultado de la medición; el alcance que define qué piezas de software (en términos de FUR) deben medirse; el nivel de granularidad que describe qué tan detallada es la documentación sobre el software (por ejemplo, en términos de la descripción de los requisitos o también la descripción de la estructura).
Todos los parámetros se definen en el Modelo de software de contexto COSMIC y es extremadamente necesario definirlos cuidadosamente. Además, cuando se comparan diferentes tamaños funcionales y luego los propósitos de sus mediciones son iguales, como en nuestro trabajo, es esencial definir esta fase de manera consistente para garantizar que los resultados se comparen de manera segura. COSMIC define varios patrones de estrategia de medición para muchas situaciones comunes en las que el método COSMIC necesita ser consistente para diferentes mediciones [5]. Un patrón de estrategia de medición define una combinación estándar de parámetros a determinar en la fase de estrategia de medición de un proceso de medición COSMIC.

También define los posibles tipos de movimientos de datos y proporciona una plantilla para dibujar el Diagrama de contexto del software que se va a medir [4].

Patrón de estrategia de medición es un concepto introducido en la última versión actual (4.0) de COSMIC [6]. Las versiones anteriores no hacen referencia a ningún patrón de estrategia.

Fase de mapeo. En esta fase, el medidor extrapola los procesos funcionales de los FUR disponibles. Es un trabajo técnico en el que los principios y, sobre todo, las reglas del método COSMIC (informadas en el Modelo de software genérico COSMIC) deben cumplirse cuidadosamente. El medidor identifica los procesos funcionales potenciales dentro de los FUR recordando que cada proceso funcional se inicia con un E desencadenante y comprende al menos dos movimientos de datos: un E más una X o una W. La E activadora es la E del usuario funcional eso inicia el proceso funcional. Un proceso funcional no puede tener más de un desencadenante E.

En algunos casos, no podría haber una relación de uno a uno entre una FUR y los procesos funcionales. Las manipulaciones de datos dentro de un proceso funcional no se cuentan como CFP, por lo tanto, COSMIC no puede dimensionar la manipulación de datos.

En algunos casos, no podría haber una relación de uno a uno entre una FUR y los procesos funcionales. Las manipulaciones de datos dentro de un proceso funcional no se cuentan como CFP [6], por lo tanto, COSMIC no puede dimensionar los sistemas intensivos de manipulación de datos. El objeto de interés se define como cualquier 'cosa' que se identifica desde el punto de vista de los FUR; puede ser cualquier cosa física, así como cualquier objeto conceptual o parte de un objeto conceptual en el mundo del usuario funcional sobre el cual se requiere que el software procese y / o almacene datos. Los objetos de interés no deben coincidir con los términos relacionados con los métodos de ingeniería de software específicos (por ejemplo, orientado a objetos). Cada E, X, R o W es un movimiento del grupo de datos de un solo objeto de interés. Hay solo dos excepciones: el E desencadenante, que puede iniciar un proceso funcional sin movimiento de datos, por ejemplo, en una consulta específica para una lista de elementos; el mensaje de error / confirmación que se define como una X para la atención de un usuario humano que confirma solo que los datos ingresados han sido aceptados, o solo que hay un error en los datos ingresados.

Fase de medición. Define cómo contar los movimientos de datos, que consisten en asociar un CFP a cada movimiento de datos. La cantidad de todos los movimientos de datos representa el valor funcional de la medición. Vale la pena señalar que en casos (a diferencia de nuestro trabajo) de agregar tamaños de medición (software estratificado en diferentes capas) o al medir el tamaño de los cambios de software, esta fase puede volverse más compleja [6].

Pasos de Estimación.

1. Obtener objetivo del negocio

2. Identificar el número de segmentos a desarrollar

3. Obtener el requerimiento funcional y no funcional del segmento en cuestión

    a) Registrar en el anexo de "Matriz del Modelo Genérico de Software"
    b) Calificar el tipo de movimiento de los datos identificados
    c) Según la siguiente convención: "E" para una Entrada, "X" para una Salida, "R" para una Lectura y 'W 'para una Escritura.
    d) Para cualquier proceso funcional, el tamaño funcional de cada movimiento de datos individual debe ser agregado en un único valor de tamaño funcional en unidades de CFP para luego sumar todos juntos.

Tamaño (proceso funcional) = Σ tamaño(Entradas) + Σ tamaño(Salidas) +
Σ tamaño(Lectura) + Σ tamaño(Escrituras)

4. Realizar un proceso de granularidad repetir paso 2, 3 hasta no existir requerimientos extensos.

5. Realizar la sumatoria de cada uno de los segmentos visualizados en el paso 2.

6. El tamaño de CFPTotal es el tamaño total de requerimientos que muestra la unidad de medida del software a desarrollar.

7. Mediante la siguiente expresión lineal obtener el costo del proyecto.

$$Y \text{ (Esfuerzo en horas)} = f(x)=A^* \text{ CFPTotal} + B$$

A = Costo variable = número de horas por CFP (horas/CFP)

B =Costo fijo en horas.

Diagrama de flujo del proceso para estimar los costos en desarrollo de aplicaciones móviles y aplicaciones web.

El proceso de gestión para estimar el costo de un desarrollo de una aplicación móvil y web consiste básicamente en desarrollar y obtener una aproximación de los costos necesarios para completar la ejecución de las actividades del proyecto. La estimación de los costos del proyecto es una predicción en un momento dado, del costo de los recursos que deben ser alinedos a las actividades del proyecto para logar su terminación.

Figura 14. Esquema de flujo de entradas, herramientas, técnicas y salidas para estimar costos.

En el siguiente diagrama se presenta el modelo propuesto para la estimación de costos de un desarrollo de software.

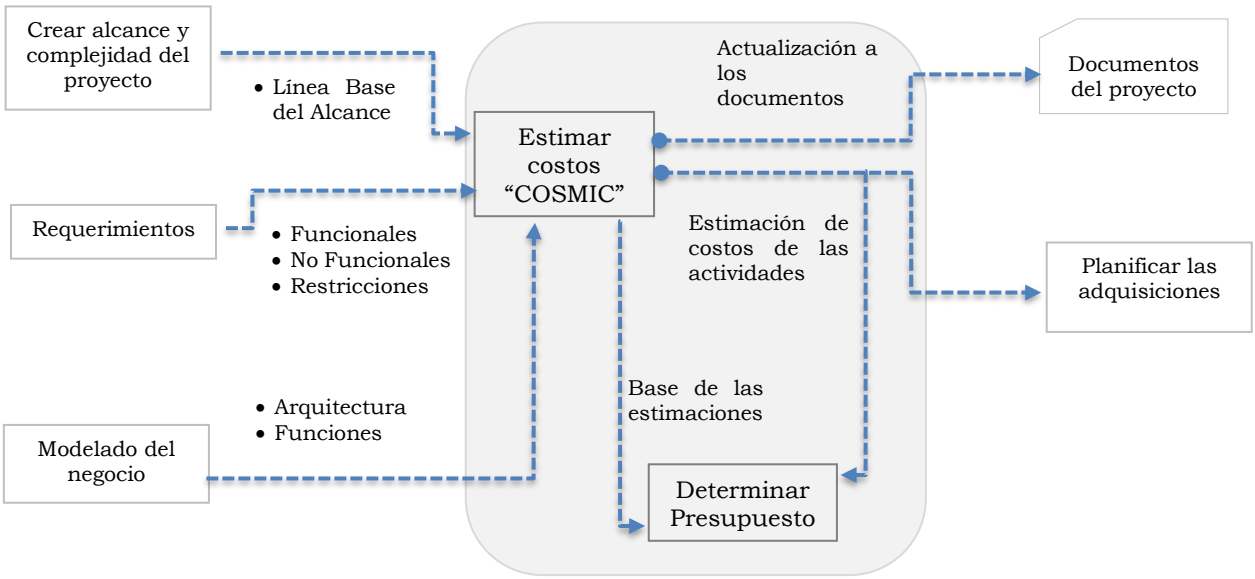

Figura 15. Diagrama de flujo del proceso para estimar los costos en desarrollo de aplicaciones móviles y aplicaciones web.

Conclusión

La estimación de costos es una actividad muy importante en el desarrollo de software. Esta actividad tiende a no ser estable, sino que cambia en diferentes puntos del ciclo de vida del desarrollo.

Se tienen que hacer estimaciones desde diferentes puntos de vista, desde la predicción hasta recopilación del comportamiento del costo, para lo cual las herramientas de estimación nos ayudan a agilizar la tarea, sin duda el poder hacer predicciones con respecto al desarrollo de software, nos da ventajas que acceden al éxito de un proyecto.

El objetivo fundamental de este trabajo de tesis se enfoca en el problema referente a la estimación de costos en el desarrollo de software para aplicaciones móviles, dado que se tiene como presente el problema que egresados de la carrera de sistemas computacionales al estimar un costo de desarrollo de software no interpretan de manera correcta un estimado. Se llega a la conclusión que la identificación de variables que repercuten en el desarrollo de software y el uso de las herramientas como COSMIC o COCOMO permiten se obtenga un estimado de costo del proyecto, esto con la finalidad de generar una perspectiva al desarrollador principiante y no indague en sus estimaciones.

Por otra parte, se plantea la propuesta de un modelo para la implementación de procesos de estimación de costos para pequeños equipos de desarrollo el cual se pretende utilizar cómo guía para el estimado de un proyecto.

Bibliografía

Abdullah, N., Rusli, N., & Ibrahim, M. (2014). Mobile game size estimation: Cosmic fsm rules, uml mapping model and unity3d game engine. *Open Systems (ICOS), 2014 IEEE Conference on. IEEE*, (pp. 42-47).

Ahmed, S., Wallace, K. M., & Blessing, L. M. (2003). Understanding the differences between how. *Research in Engineering Design*, 1-11.

Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Trans. Software Eng.*, 639-648.

app, S. p. (n.d.). *http://en.wikipedia.org/wiki/Siri_(software)*.

Arifoglu, A. (1993). A methodology for software cost estimation. *SIGSOFT Software Eng.* , 96-105.

Auer, M., Trendowicz, A., Grasser, B., Haunschmid, E., & Biffl, S. (2006). Optimal project feature weights in analogybased cost estimation: improvement and limitations. *IEEE Trans Softw Eng, 32*(2), 83-92.

Bailey, J. W., & Basilio, V. R. (1981). A meta-model for software development resource expenditures. *In Proc. 5th International Conference on Software Engineering, Lund, Sweden*, 107-116.

Begel, A., & Simon, B. (2014). Novice Software Developers, All Over Again. pp. 1-2.

Benala, T., Dehuri, S., Satapathy, S., & Madhurakshara, S. (2012). Genetic algorithm for optimizing functional link artificial neural network based software cost estimation. *Proceedings of the International Conference on Information Systems Design and Intelligent Applications*(132), 75-82.

Bhardwaj, M., & Rana, A. (2015). Estimation of Testing and Rework Efforts for Software Development ProjectsAsian Journal of Computer Science and Information Technology. *Asian Journal of Computer Science and Information Technology*, *5*, pp. 33-37.

Boehm, B. (1980). *Software engineering economics.* USA: Prentice Hall PTR.

Boehm, B. W. (1981). Software engineering economics. *Prentice Hall, Englewood Cliffs*.

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches. *A survey. Ann Softw Eng, 10*(4), 177-205.

Boulangera, S., & Smith, I. (2001). Multi-strategy workspace navigation for design education. *Design Studies*, 111-140.

Breiman, L., Friedman, J. H., & Olshen, R. A. (1984). Classification and regression trees. *Wadsworth, Pacific Grove*.

Carpersen, M. E., & Kölling, A. (New York, NY, USA, 2006). A novice's process of object-oriented programming. *In OOPSLA '06: Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 892-900.

Carver, J. C., Shull, F., & Basilli, V. (2006, 01 01). Can observational techniques help novices overcome the software inspection learning curve? An empirical investigation. pp. 5-6.

Catalino, G., Salza, P., Gravino, C., & Ferrucci, F. (2017). A Set of Metrics for the Effort Estimation of Mobile Apps. *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems*, (pp. 194-198).

Ceschi, M., Sillitti, A., Succi, G., & De Pangilis, S. (2005). Project Management in Plan- Based and Agile Companies. *IEEE Software,, 22*, 21-25.

Conboy, K. (2006). Cost estimation in agile development projects. *National University of Ireland, Galway.*

Corazza, A., Martino, S., Ferruci, F., Gravino, C., Sarro, F., & Mendes, E. (2013). Using tabu search to configure support vector regression for effort estimation. *Empirical Software Engineering, 18*(3), 506--546.

Costagliola, G., Ferrucci, F., Tortora, G., & Vitiello, G. (n.d.). Class point: an approach for the size estimation of object-oriented systems. *IEEE Trans Softw Eng, 31*(1), 52-74.

Courtney, R. E., & Gustafson, D. A. (1981). Software Engineering Economics. *Prentice Hall, Englewood Cliffs, NJ.*

Cruz, A. M., & Paiva, S. (2016). Modern software engineering methodologies for mobile and cloud environments. *Information Science Reference*, (pp. 60--87).

Dalkey, N., & Helmer, O. (1963). An experimental application of the Delphi method to the use of experts. *Manag Sci, 9*(3), 458-467.

D'Avanzo, L., Ferrucci, F., Gravino, C., & Salza, P. (2015). COSMIC functional measurement of mobile applications and code size estimation. *Proceedings*

of the 30th Annual ACM Symposium on Applied Computing, (pp. 13 - 17). New York.

DeMarco, T. (1982). Controlling Software Projects. *Yourdon Press, New York*.

Dewar, R., & Astrachan, O. (2009). Point/counterpoint CS education in the U.S.: heading in the wrong. *Communications of the ACM*, 41-45.

Fairley, R. (1994). Risk management for software projects. *IEEE Software*, 57-67.

Fenton, N. E. (1991). Software Metrics: A Rigorous Approach. *Chapman and Hall, London*.

Ferens, D. V. (1988). Software Size Estimation Techniques. *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference.*

Ferruci, F., Gravino, C., & Pasquale, S. (2017). Using cosmic for the functional size measurement of distributed applications in cloud environments. *Software Project Management for Distributed Computing: Life-Cycle Methods for Developing Scalable and Reliable Tools.* Springer.

Genexus. (2014, Julio 15). *Industria del software aporta 0.7% del PIB*. Retrieved Septiembre 26, 2014, from http://eleconomista.com.mx/industrias/2014/07/15/industria-software-aporta-07-pib

Heemstra, F. J. (1992). Software cost estimation. . *Inform. Software Tech*, 627 - 639.

Heeringen, H., & Gorp, E. (2014). Measure the Functional Size of a Mobile App: Using the COSMIC Functional Size Measurement Method . *IWSM-MENSURA '14 Proceedings of the 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, (pp. 11-16). Washington.

Highsmith, J. (2003). Agile Project Management: Principles and Tools. *Cutter Consortium*, *4*, pp. 1-37.

Hihn, J., & Habib-agahi, H. (1991). Cost estimation of software intensive projects: A survey of current practices. *International Conference on Software Engineering.*, 13-16.

Humphrey, W. (1989). Managing the Software Process. *Prentice-Hall*, 8-30.

Humphrey, W. S. (1995). A Discipline for Software Engineering. *Addison-Wesley. Reading, ing, MA*.

IFPUG. (2017, 11 21). Retrieved from International Function Point User Group. Function Point Counting Practises Manual: http://www.ifpug.org

Jensen, R. (1983). An improved macrolevel software development resource estimation model. *In Proceeding of 5th Conference of International S Parametric Analysts*, 88-92.

Johnson, E. J. (1998). Expertise and decision under uncertainty: Performance and process. *The Nature of Expertise*.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. . *Conzniirn. ACM*, 416-429.

Keung, J., Jeffery, R., & Kitchenham, B. (2004). The Challenge of Introducing a New Software Cost Estimation Technology into a Small Software Organisation. *Proceedings of the 2004 Australian Software Engineering Conference.* Australia.

Khalid, T. A., & Yeoh, E. T. (2017). Early Cost Estimation of Software Rework Using Fuzzy Requirement-Based Model. *2017 International Conference on Communication, Control* (pp. 1-5). Sudan: Computing and Electronics Engineering.

Kitchenham, B., & Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. *IEEE Transactions on Software Engineering. 30*, pp. 1023-1035 . NJ: IEEE Press Piscataway .

Lederer, A. L., & Prasad, J. (1993). Information systems software cost estimating: A current assessment. *Inform. Tech.*, 22-23.

Lederer., A. L., Mirani, R., Neo, B. S., Pollar, C., Prasad, J., & Ramamurthy., K. (1990). Information system cost estimating: A management perspective. *MIS Quurterly*.

Letelier, P., & Penadés, M. C. (2006, Junio). Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Departamento de Sistemas Informáticos y Computación (DSIC)*, pp. 4-15.

Linstone, H., & Turoff et al. (1975). The Delphi method: Techniques and applications. *29.* MA: Addison-Wesley Reading.

Lovaasen, G. (2001). Brokering with eXtreme Programming. *Universe 2001. Raleigh, North Carolina.*

Madachy, R. (1994). *A software project dynamics model for process cost, schedule and risk assessment.* Ph.D. Dissertation, University of Southern California.

Man Lui, K., & Chan C., K. (2006, September 9). Pair programming productivity: Novice-novice vs. expert-expert. pp. 915-925.

Mendes, E., Mosley, N., & Counsell, S. (2003). Investigating early web size measures for web cost estimation. *Proceedings of EASE'2003 Conference*, (pp. 1-22.). Keele.

Pierre, N. R. (2014, 01 01). *Software Engineering with the UPEDU*. Retrieved Junio 09, 2014, from http://www.upedu.org/

Preuss, T. (2013). Mobile Applicactions, function points and cost estimating. *International Conference on Cost Estimation and Analysis Association (ICEAA).*

Putman, L. H. (1978). Particle swarm optimization in the fine-tuning of fuzzy software cost estimation models. *Int J Softw Eng Knowl Eng, 1*(2), 12-23.

R. Wright, D. (2012). Inoculating Novice Software Designers with Expert Design Strategies. *American Society for Engineering Education*, 2-51.

Reddy, P. (2011). Particle swarm optimization in the fine-tuning of fuzzy software cost estimation models. *Int J Softw Eng Knowl Eng, 1*(2), 12-23.

Robillard, P. N. (1999). The role of knowledge in software development. *Communications of the ACM*, 87-92.

S. Pressman., R. (2002). *Ingeniería de Software, Un enfoque práctico.* España: Mc Graw Hill.

Sellami, A., Haoues, M., Rusli, N., & Ibrahim, M. (2014). Mobile game size estimation: Cosmic fsm rules, uml mapping model and unity3d game engine. *Open Systems (ICOS), 2014 IEEE Conference on. IEEE*, (pp. 42-47).

Sethumadhavan, G. (2011). Sizing Android Mobile Applications. *International Software Measurement and Analysis.*

Shepperd, M., & Schofield, C. (1997). Estimating software proecteffort using analogies. *IEEE Trans Softw Eng, 23*(11), 736-743.

Sommerville, I. (2006). *Software Engineering: (Update) (8th Edition) (International Computer Science).* Boston, MA, USA: Publishing Co. Inc.

Sulayman, M. (2010). Protocol for Software Process Improvement of Small & Medium Web Development Organizations. p. 3.

Tortorella, M., & Visaggio, G. (1999). Empirical Investigation of Innovation Diffusion in a Software Process. *International Journal of Software Engineering and Knowledge Engineering*, pp. 595-621.

Vicinanza, S., Mukhopadhyay, T., & Prietula, M. (1991). Software effort estimation: An exploratory study of expert performance. *Inform. Sysrems Res*, 243-262.

Vishal, S. (2010). Optimized fuzzy logic based framework for effort estimation in software development. *Int J Comput Sci, 7*(2), 30-38.

Wasserman, A. I. (2010). Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research.* (pp. 397-400). ACM.

Wiki. (2014). *Programación Extrema*. Retrieved Octubre 20, 2014, from Programación Extrema: http://programacion-extrema.wikispaces.com/

Williams, L., R. Kessler, R., & Cunningham, W. (200). Strengthening the Case for Pair Programming. *Process Diversity Focus*, 20- 25 .