UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

APLICACIÓN DE COSMIC EN CONJUNTO CON LA METODOLOGÍA SCRUM

TESIS
QUE PARA OPTAR POR EL GRADO DE
MAESTRA EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA
ERIKA LIZBETH HERNÁNDEZ MORALES

TUTOR O TUTORES PRINCIPALES:
M. en C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁlEZ
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
Co Tutor: FRANCISCO VALDÉS SOUTO
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

JURADO ASIGNADO:

- Presidente M. EN C. GUSTAVO ARTURO MÁRQUEZ FLORES
  Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
- Secretario DR. FRANCISCO VALDÉS SOUTO
  Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
- Vocal M. EN C. MARÍA GUADALUPE IBARGÜENGOITIA GONZÁLEZ
  Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
- 1er. Suplente DRA. ANA YURI RAMÍREZ MOLINA
  Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
- 2do. Suplente DR. ISRAEL ORTEGA CUEVAS
  Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

CIUDAD DE MÉXICO, NOVIEMBRE 2022

Incluso en los peores momentos encontré luz
y a pesar de la frustración se logró,
para aquellos que creyeron en mí
sólo me resta decir gracias.

Agradecimientos

A toda mi familia la cual me apoya y siempre tienen palabras de aliento y motivación para los malos momentos, los más oscuros y lúgubres, así como los momentos más alegres y coloridos me dan sus aliento y fe.

A mi madre Elvia quien siempre ha creído en mi y su cariño es inconmensurable, mi tía Araceli quien siempre me brinda su mano y un lugar para poder hablar y a mi abuela Sara quien siempre me apoya.

Gracias a Daniel, Alexia, Ricardo y Jazmín cuyas palabras de apoyo fueron una de las piezas más importantes en la culminación de este trabajo, siempre tendrán un lugar especial en mi vida. Daniel con quien he compartido tanto la licenciatura y la maestría. Alexia quien me ha apoyado a lo largo de la licenciatura en la búsqueda por encaminarme a algo de mi agrado. Ricardo quien me ayudo en el tramo final motivándome cada día y escuchándome cuando era necesario.

A mis compañeros de la maestría, amigos y ex compañeros de la licenciatura con los que aún tuve contacto, los cuales me han motivado y ayudado.

A los amigos que hice fuera de las actividades escolares, Pamela, Paula, Ángeles, gracias por las platicas tan profundas, por sostenerme cuando no me encontraba en mi mejor momento.

A la Maestra Guadalupe Elena Ibargüengoitia, la cual me aceptó y sin la que este trabajo no hubiera sido posible. Agradezco su tiempo y paciencia para la culminación de este trabajo.

Al Dr. Francisco Souto él cual me apoyo y animó en todo el proceso de la construcción de este trabajo.

A CONACYT por el apoyo y oportunidad de hacer una maestría.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Problemas u Oportunidad

La estimación de software ha atraído la atención de numerosos investigadores, quienes encontraron, entre muchas cosas que: "La estimación es el centro de un desarrollo de proyecto exitoso"(41).

La estimación puede efectuarse sobre diversos atributos del software como pueden ser: el tamaño, complejidad, arquitectura, esfuerzo, Story Points, étc. Diversas propuestas son estudiadas en el artículo A Review Article on Software Effort Estimation in Agile Methodology(54).

Una estimación correcta en el desarrollo de software evita costos excesivos. Los costos respecto a recursos pueden ser calculados mediante la estimación de esfuerzo(54).

La estimación de costos *cost estimates* o *estimation costs*, se refiere a la estimación del software, no solo costo sino también esfuerzo o tiempo. Por lo que esta tesis cuándo se refiera a estimación de costos, se refiere a cualquier costo posible de estimar o requerido en un proyecto. También se puede mencionar específicamente alguno en particular, por ejemplo estimación de esfuerzo.

Las técnicas de estimación que se han utilizado han ido cambiando, siendo la más utilizada el Juicio de experto(56)(19)(54), sin embargo otras técnicas como Planning Poker, o combinaciones de métodos que involucran algoritmos como Machine Learning han tomado poco a poco popularidad. El problema del Juicio de Experto es que utiliza datos cualitativos que están sujetos a la opinión de los llamados “expertos”, impidiendo replicar los resultados y dejando documentación imprecisa sobre las decisiones tomadas.

En 2002 fue aceptado el estándar 19761 de medición de tamaño funcional COSMIC, él cuál permite una medición estandarizada del software gracias a diversas pautas. Este estándar también provee de pautas para estimar software. Actualmente hay un incremento en la búsqueda de cómo poder realizar estimaciones de una manera más formal, por lo que estimar utilizando estándares resulta un enfoque apropiado..

Por otra parte, desde 2001 fecha en que se publicó el Manifiesto Ágil, han surgido diversas las metodologías ágiles las cuales tienen como bases la rápida respuesta al cambio e iteraciones (sprints) cortas, siendo SCRUM una de las más utilizadas(56).

En este tipo de metodologías, si bien se requieren estimaciones, su particular forma de trabajar, agrega un componente de complejidad importante, deben de realizarse rápido, lo cuál ha impulsado más el hecho de estimar bajo el juicio de experto.

En esta tesis se propone una forma de integrar el estándar de medición 19761 COSMIC junto con la metodología ágil SCRUM como una propuesta de realizar estimaciones de tamaño funcional, respetando las etapas de desarrollo de SCRUM, demostrando que son completamente compatibles.

1.2. Objetivo

El objetivo de esta tesis consiste en hacer una propuesta de cómo integrar mediciones formales y estandarizadas como el método COSMIC a las metodologías ágiles para realizar estimaciones, en particular en la metodología SCRUM, que proveerá los siguientes beneficios:

1. Eliminar el juicio de experto y la utilización de métricas ad hoc.
2. Replicar resultados independientemente de la persona que la ejecute.
3. Permitir comparar tanto equipos como proyectos distintos.
4. Una pauta para poder integrar a otra metodología ágil el estándar de medición de tamaño funcional COSMIC (ISO / IEC 19761).

1.3. Preguntas de Investigación

Las preguntas que se pretenden responder son:

1. ¿Cuál es el estado actual de la estimación en el desarrollo de software con metodologías ágiles?
2. ¿Cómo puede ayudar a mejorar un método de estimación basado en COSMIC las estimaciones?
3. ¿Cuáles son las principales ventajas de utilizar el método COSMIC como base para las estimaciones?
4. ¿Cómo lograr estimar el esfuerzo basado en el estándar 19761 COSMIC en una metodología ágil, la cual está enfocada en una rápida respuesta al cambio y tiene iteraciones muy cortas?

1.4. Contribuciones

La estimación de esfuerzo es un área de oportunidad debido a la falta de aplicación de métodos formales para ella. Cabe señalar que a partir del esfuerzo se pueden determinar los costos o incluso tiempos de un proyecto. Con esta tesis se pretende dar una pauta para lograr estimaciones formales utilizando el método estándar de medición de tamaño funcional COSMIC que provee:

- Mediciones rápidas. Siendo SCRUM una metodología ágil donde se valoran los sprints cortos respecto al tiempo, COSMIC establece mecanismos que permiten dimensionar el tamaño del software de una forma rápida, lo que permite hacerlo en tiempos muy cortos.
- Una forma cuantitativa y formal de estimar usando métricas estandarizadas.
- Integración de métodos formales de medición en metodologías ágiles, mejorando las estimaciones y habilitando la comparación de las métricas entre proyectos.
- Una métrica estandarizada que permitirá generar datos comparables entre empresas.

1.5. Estructura de la Tesis

El presente trabajo está dividido en cinco capítulos, descritos brevemente a continuación:

- **Capítulo 2 Marco Teórico:**

  En este capítulo se presentan los conceptos necesarios para entender el tema tratado en la tesis. Se comenzó con una introducción a las metodologías ágiles, la cuál plantea una de las bases en las que se trabajó. Adicionalmente se mencionan los elementos más importantes de SCRUM (la metodología ágil elegida en esta tesis) así como su forma de trabajo. Posteriormente se trabajó el tema principal

de esta tesis los fundamentos de la estimación, dando los conceptos básicos de esta: definiciones, proceso de estimación, modelos de productividad y confiabilidad. Finalmente se habló de la segunda base para esta tesis, los estándares de medición de tamaño funcional, mencionando qué son, y una breve historia de su evolución, para terminar mencionando al estándar 19761 COSMIC, el cual es el elegido en esta tesis por ser el único método de segunda generación que resuelve los problemas de los métodos de primera generación. Se incluyó un glosario de términos que serán utilizados, así como los procesos de medición, aproximación y estimación, los cuales serán utilizados para la propuesta en el capítulo 4.

- **Capítulo 3 Estimación de esfuerzo en metodologías ágiles en los últimos años:**

  En este capítulo se escogieron y desglosaron 3 artículos Effort Estimation in Agile Software Development: A Systematic Literature Review (56), An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review(19) y A Review Article on Software Effort Estimation in Agile Methodology(54), los cuales muestran revisiones sistemáticas sobre artículos publicados referentes al tema de la estimación de esfuerzo en el desarrollo de software con metodologías ágiles. Este capítulo justifica la creación de esta tesis ya que se encontró que hasta el año 2020 (2001 - 2020 período donde se realizó la revisión sistemática), el 26 % de artículos reportaban utilizar juicio de experto, 38 % utilizó una combinación de métodos que implicaban la implementación de algún algoritmo con juicio de experto y 21 % utilizó métodos con algoritmos o métodos basados en estándares. De estos sólo 7 artículos utilizaron COSMIC, de los cuales algunos artículos lo utilizaron para estimar(27), algunos otros lo utilizaron en comparaciones con Use Case Points o diversas técnicas(5)(28); sin embargo ninguno dio una pauta para su integración en SCRUM.

- **Capítulo 4 Propuesta de Integración entre COSMIC y SCRUM:**

  En este capítulo se presenta la propuesta de integración de COSMIC junto a la metodología ágil SCRUM para estimar esfuerzo. Para ello se utilizó el Proceso de Medición Rápida (Aproximación) que proporciona COSMIC, utilizándolo en dos de cinco fases de SCRUM, la fase de Inicio y la fase de Planificación y Estimación. En la fase de Inicio la propuesta permite generar una aproximación a la primera versión de los requerimientos que se tengan, estos requerimientos tienden a estar descritos en un alto nivel y no estar completos. En la fase de Estimación y Planeación que se considera el inicio de cada Sprint, se espera que los requerimientos se vayan refinando o incluso pueden aparecer nuevos, por lo que las estimaciones deben ser más precisas; sin embargo, de acuerdo a como se encuentren los requerimientos se proponen métodos de medición rápida.

- **Capítulo 5 Conclusiones:**

  En este capítulo se presentan las conclusiones de la tesis, seguido de las respuestas a las preguntas de investigación planteadas en este capítulo para terminar con los temas de investigación que pudieran ser de interés para futuros trabajos.

Capítulo 2

Marco Teórico

2.1. Metodologías Ágiles

La Ingeniería de Software se define como la aplicación de un enfoque disciplinado cuantificable sistemático, para el desarrollo, operación y mantenimiento de software(22), a lo largo de los años ha tenido una maduración, diversas metodologías han surgido para satisfacer las demandas en el desarrollo de los proyectos de software. Estas metodologías de desarrollo de software son vistas como un conjunto de técnicas y métodos organizativos usados para diseñar soluciones de software informático, con el objetivo de organizar y guiar al equipo de desarrollo a desempeñar tareas relacionadas a la construcción del software de la mejor manera posible.

2.1.1. Manifiesto Ágil

En el año 2001 se publicó el Manifiesto Ágil (38) que dio inicio formalmente a las metodologías ágiles, conocidas hasta el momento como "Metodologías de Desarrollo de Software de peso liviano", comprendidas en los años 90's como fueron SCRUM, KANBAN, XP, LEAN.

Las metodologías ágiles se consideran como un paradigma definido por valores y guiado por principios los cuales son encontrados en el Manifiesto Ágil(30). Cuatro valores principales son los puntos claves para el Manifiesto Ágil, los cuales son:

- **Individuos e interacciones** sobre procesos y servicios ya que consideran al personal como el "activo" más importante que tiene cualquier organización, ya que aportan creatividad y capacidad de innovación. En muchas ocasiones los equipos tienen la posibilidad de auto-organizarse para poder explotar esas cualidades y agregar valor al software resultante.
- **Software funcionando** sobre documentación extensiva. A diferencia de los antiguos métodos que se priorizaba la documentación al momento de entregar un proyecto donde el usuario final podía encontrar un manual para el uso del producto final, en las metodologías ágiles se valora un producto funcional que cumpla con los requisitos iniciales y que sea intuitivo(50).

- **Colaboración con el cliente** sobre negociación contractual. Lo cual permite un acercamiento al cliente, ayudando a la comunicación y el intercambio de ideas y avances del proyecto, favoreciendo la retroalimentación constante, esto viene de la idea del siguiente valor.
- **Respuesta ante el cambio** sobre seguir un plan. Estas metodologías están basadas en un desarrollo incremental, así el proyecto es dividido en ciclos y en cada ciclo se agregan nuevas funcionalidades; estos ciclos suelen ser cortos, lo que permite la adición de nuevas funcionalidades.

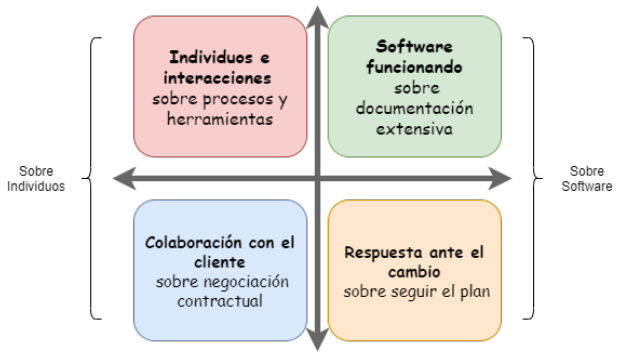

Figura 2.1: Valores de la Metodología Ágil (15)

Estos valores pueden resumirse en la Figura 2.1 donde enfocarse en los valores mostrados en negritas no significa renunciar a los otros, ofreciendo a los equipos procesos de desarrollos menos rígidos y más rápidos. Junto a los cuatro valores principales se encuentran los doce principios del Manifiesto Ágil(38) , mencionados a continuación:

1. **Satisfacción del cliente:** La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor(38).
2. **Apertura al cambio:** Aceptar que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente(38).
3. **Entregas tempranas y constantes:** Entregar software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible(38).
4. **Trabajo colaborativo:** Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto(38).
5. **Equipo motivado:** Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo(38).
6. **Comunicación efectiva:** El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros, es la conversación cara a cara(38).
7. **Software funcionando como medida de progreso:** El software funcionando es la medida principal de progreso(38).

8. **Desarrollo Sostenible** Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida(38).
9. **Atención continua:** La atención continua a la excelencia técnica y al buen diseño mejora la agilidad(38).
10. **Minimizar el gasto:** La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial(38).
11. **Equipos auto-organizados:** Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados(38).
12. **Reflexión y mejora:** A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia(38).

De los métodos más populares en las metodologías ágiles se encuentra SCRUM (56)(19), el cual se usará como ejemplo a lo largo de esta tesis y en la siguiente sección se abordará a modo de introducción.

2.2. SCRUM

Como se mencionó en la sección anterior,en los años 1980 se buscaba optimizar los proyectos de software ya que se observaban proyectos con altos costos, una calidad deplorable e inflexibles, así en el año 1986 Hirotaka Takeuchi e Ikujiro Nonaka publicaron el artículo "The New Product Development Game". En este artículo se hizo una evaluación en diversas empresas como Canon, HP, étc., empresas que se consideraban a la vanguardia en desarrollo de proyectos para entender el ciclo de vida de sus productos, este ciclo estaba a cargo de equipos multidisciplinarios que partían de requisitos generales y generaban productos novedosos en tiempos cortos. El principal punto de comparación fue la forma de trabajar de un equipo de Rugby y de una formación del equipo llamada SCRUM. Posteriormente en 1995 Jeff Sutherland y Ken Schwaber presentan las prácticas donde se formalizaba SCRUM y poder incluirlo en la lista de *Agile Alliance*.[1]

Así nació SCRUM como un marco de trabajo colaborativo entre equipos, donde se motiva al equipo a aprender a través de sus propias experiencias en el proyecto, auto organizarse y reflexionar de su trabajo para una mejora continua (23). SCRUM cuenta con cinco valores fundamentales descritos a continuación:

- **Foco:** Todo el equipo está centrado en el trabajo asignado en ese momento (SCRUM lo denomina como Sprint), así como de alcanzar los objetivos establecidos para ese trabajo especifico.
- **Apertura:** Tanto el equipo de desarrollo como los interesados en el proyectos acuerdan una apertura con el trabajo que se está llevando a cabo y los desafíos que existen para la realización de éste.
- **Respeto:** Los miembros del equipo se respetan mutuamente, lo que les permite mantener una comunicación óptima para lograr sus objetivos.

[1]Organización miembro sin fines de lucro dedicada a promover los conceptos de desarrollo de software Agile. Con más de 60.000 miembros y suscriptores en el mundo.

- **Valor:** Los miembros del equipo poseen el coraje de hacer lo correcto por el bien del proyecto.
- **Compromiso:** Los miembros del equipo están comprometidos personalmente a alcanzar los objetivos trabajando en sus problemas incluso si estos son considerados difíciles.

2.2.1. Elementos de SCRUM

SCRUM definió sus propios elementos para ayudar a explicar el funcionamiento del marco de trabajo de SCRUM. Estas definiciones se encuentran plasmadas en "The SCRUM Guide"(26).

1. **El equipo SCRUM *(SCRUM Team).*** Considerado como la unidad fundamental de SCRUM, este equipo de personas debe ser pequeño, entre los miembros del equipo debe existir al menos una persona:

    - **Dueño del producto *(Product Owner).***[1] Responsable de maximizar el valor del producto resultante aludiendo a diversas técnicas de acuerdo a las necesidades de cada equipo de trabajo SCRUM, también es el encargado de gestionar los diversos artefactos generados en SCRUM como es el *Product Backlog*, el cual se definirá más adelante.
    - **El facilitador de SCRUM *(SCRUM Master).***[2] Responsable de establecer y seguir la metodología SCRUM según la *Guía Oficial de SCRUM* ayudando a comprender los elementos de la teoría y la práctica a los miembros del equipo y a la organización. Considerado como el líder que mejor conoce la metodología dentro del equipo. El crecimiento de los miembros restantes del equipo SCRUM y la gestión del resto del equipo son responsabilidades que debe manejar.
    - **Desarrolladores *(Developers).*** Son considerados las personas dentro del equipo de SCRUM que están comprometidas a crear el producto final en cada Sprint. La cantidad de desarrolladores varía de acuerdo al proyecto; sin embargo se debe seguir la idea de equipos pequeños. Cada desarrollador es responsable de su trabajo siguiendo los valores de SCRUM, adaptando sus metas u objetivos al proyecto y apoyándose en el *SCRUM Master* de su equipo.

2. **Eventos de SCRUM *(SCRUM Events).*** Una vez definido el trabajo, SCRUM divide al proyecto en iteraciones, estos son llamados **Sprints**, en cada Sprint ciertas actividades son asignadas, permitiendo una oportunidad formal de inspeccionar y adaptar los artefactos de SCRUM y transformarlos en un objeto de valor. Con una duración fija de un mes o menos, los sprints comienzan cuando el último sprint es terminado, en ellos podemos encontrar:

    a) **Planificación del Sprint *(Sprint Planning).*** Cuando da comienzo un nuevo Sprint se debe establecer un plan de trabajo, este plan es creado en colaboración con todo el Equipo SCRUM, con el *Product Owner* encargado de incluir a los miembros del equipo; adicionalmente personas ajenas al equipo pueden ser incluidas en la

[1] Se utilizará a lo largo de la tesis el término Product Owner
[2] Se utilizará a lo largo de la tesis el término SCRUM Master

planeación del sprint si se considera necesaria su asesoramiento u opinión. Las preguntas que deben ser resueltas en la planeación deben ser: ¿Por qué el Sprint es valioso?, ¿Cuáles son las metas del Sprint?; es decir, ¿Qué actividades deben ser realizadas en ese Sprint? y finalmente ¿Cómo se resolverán las metas fijadas en el Sprint?

b) **Revisión del Sprint *(Sprint Review).*** Una vez que el período del Sprint es completado se hace una revisión de los resultados obtenidos para poder determinar futuras adaptaciones. El equipo SCRUM debe presentar sus logros y dificultades encontradas, así de acuerdo a la información obtenida se puede alimentar al siguiente *Sprint Planing* para saber que se deberá hacer a continuación. El *Sprint Review* sólo debe durar una sesión con un máximo recomendado de cuatro horas si el Sprint es de un mes; sin embargo, esto no es estricto(26).

c) **Retrospectiva del Sprint *(Sprint Retrospective).*** El propósito del *Sprint Retrospective* es planificar formas de aumentar la calidad y eficacia en los productos que se desean lograr. De acuerdo al *Sprint Review* del último Sprint el equipo identificará los procesos, herramientas o interacciones que pudieron afectar al desarrollo del último Sprint para poder modificarlos o resolverlos, además de identificar los procesos que se ejecutaron de forma correcta. El *Sprint Retrospective* concluye formalmente el Sprint y tienen un límite de tiempo recomendado de máximo tres horas para un Sprint de un mes(26).

d) **Reuniones Diarias *(Daily SCRUM).*** [1] En todo el desarrollo del Sprint se tendrán reuniones diarias con le objetivo de inspeccionar el progreso hacia las metas establecidas para el Sprint y adaptar el *Sprint Backlog* según sea necesario para ajustar el trabajo próximo. Las sesiones deben durar un máximo recomendado de 15 minutos, en ellas deben asistir los desarrolladores del equipo, de preferencia deben ser realizadas a la misma hora. Estas sesiones permiten mejorar la comunicación entre el equipo y promueven la toma de decisiones rápidas de acuerdo al desarrollo del proyecto, además de propiciar la flexibilidad a cambios en el proyecto(26).

e) **Sprint.** El momento donde las ideas se convierten en valor. Tienen una duración fija determinada por el SCRUM Master, se recomiendan períodos de un mes o menos(26)

.

3. **Artefactos de SCRUM *(SCRUM Artifacts).*** Los artefactos creados para SCRUM ayudan a representar el trabajo que se está realizando, así como su valor, están diseñados para maximizar la transparencia de la información clave en el proyecto, además de ayudar al desarrollo de futuros sprints o la evaluación de estos.

   - **Lista de Producto *(Product Backlog).*** [2] Es una lista emergente y priorizada de las actividades necesarias para mejorar/ realizar el producto, la primera vez que es creada en la Reunión de Planeación y Especificación de Requerimientos y va modificándose por el equipo en los Daily Scrum o en el Sprint Planning, el equipo es el encargado de llevar las actividades a cabo. El objetivo de llevar la lista priorizada es trazar pequeños objetivos con el fin de alcanzar el producto final *(Product Goal)*; esta meta debe encontrarse en el Product Backlog.

[1] Se utilizará a lo largo de la tesis el término Daily SCRUM
[2] Se utilizará a lo largo de la tesis el término Product Backlog

- **Lista del Sprint *(Sprint Backlog).***[1] Compuesto por el Sprint Goal y el conjunto de actividades del Product Backlog seleccionados para el sprint, así como un plan propuesto para realizar las actividades seleccionadas que puedan ser ejecutadas por los desarrolladores. Es creado durante el Sprint Planning, al obtener nuevas tareas del Daily Scrum, el Sprint Backlog está en constante cambio. El objetivo de llevar la lista es seleccionar las actividades del Sprint en curso con el fin de alcanzar la meta del Sprint *(Sprint Goal)* permitiendo una flexibilidad en el desarrollo y apoyando el trabajo colaborativo del equipo.

- **Incremento *(Increment).*** Un incremento es un pedazo de actividad realizada y terminada que formará parte del producto final. Cada incremento se suma a los incrementos anteriores una vez que pasa una minuciosa verificación para garantizar su integración con los demás, es la definición de *Actividad completada* que maneja SCRUM. Cada Sprint debe generar al menos un incremento; sin embargo un Sprint puede generar más de un incremento. Sólo los incrementos podrán presentarse en el Sprint Review.

2.2.2. Ciclo de vida de un Proyecto en SCRUM

Conociendo los elementos de SCRUM, se verá cómo se desarrolla un proyecto de acuerdo a las 5 fases de SCRUM, cada etapa de SCRUM forma parte de una meta en común para lograr satisfacer el Proudct Backlog. En la Figura 2.2 se muestra el diagrama de cómo se representan las cinco fases, además de los elementos de SCRUM presentes en cada fase, la flecha curva nos muestra que el proceso de un Sprint contempla las fases de Planeación y estimación, Implementación y Revisión y retrospectiva.

Inicio: En esta fase se estudia y analiza el proyecto para determinar cuáles son las metas a lograr Product Goal, cuales serán los elementos y tareas generales para la realización y culminación del proyecto, así como las fechas para los documentos entregables. Finalmente en esta fase se da la conformación del equipo SCRUM que deberá llevar a cabo el proyecto.

Planeación y Estimación: Considerada como una fase de vital importancia, en esta fase es donde se considera iniciado un sprint, por lo que se construye el sprint Backlog de acuerdo a las actividades seleccionadas para el Sprint, además de estimar los tiempos para la ejecución de las tareas. La persona a cargo es el SCRUM Master.

Implementación: En esta fase toda la planeación hecha en la fase anterior se lleva a cabo, por lo que los Daily SCRUM deben llevarse a cabo todos los días para mantener informado a todo el equipo sobre los avances en el proyecto, además de adecuar el Sprint Backlog en caso de ser necesario.

Revisión y retrospectiva: Cuando todas las actividades sean finalizadas se deben someter a una revisión para comprobar la calidad y que los nuevos artefactos puedan ser implementados como parte del proyecto final. Esta etapa marca la finalización del sprint con el Sprint Review.

[1] Se utilizará a lo largo de la tesis el término Sprint Backlog

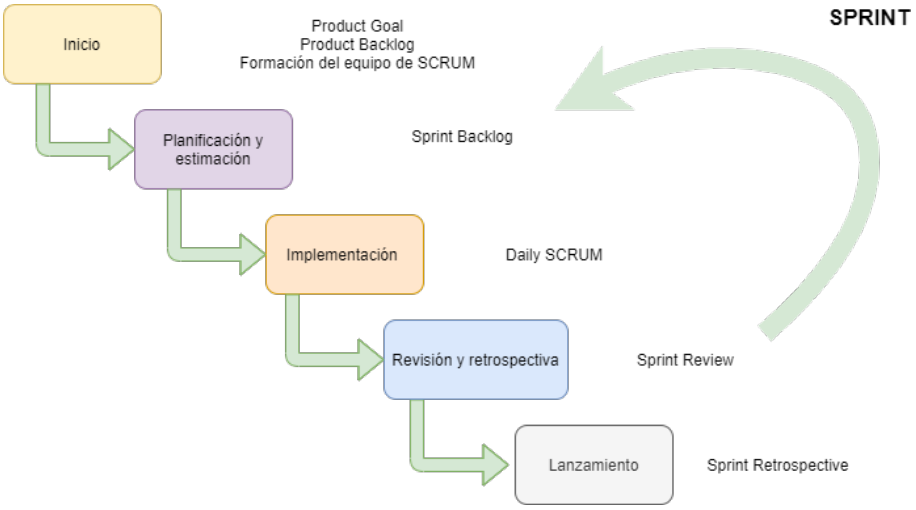

Figura 2.2: Fases del desarrollo en SCRUM

Lanzamiento: Una vez que se ejecutó el último sprint, el producto finaliza y debe entregarse al cliente, se deben enviar los entregables y una retrospectiva del proyecto para futuros proyectos. Cuando esta etapa finaliza el software deberá estar terminado.

2.3. Fundamentos de la estimación

En 1968 Rubey publicó un artículo sobre la complejidad al construir software(53), en el señaló un total de 57 atributos de software distribuidos en siete categorías, además definir y discutir sus métricas realizadas a través de fórmulas y explicadas a detalle. En 1980 Curtis señaló que se deben aplicar procedimientos científicos rigurosos para estudiar el desarrollo de sistemas de software para transformar la programación en un disciplina de ingeniería. En el centro de estos procedimientos está el desarrollo de técnicas de medición, la determinación de relaciones causa-efecto. Los acontecimientos anteriores fueron preliminares importantes ocurridos entre los años 60 y 70 para poder dar pie al desarrollo de la medición de software(16).

Actualmente las empresas buscan lograr un óptimo desarrollo que lleve a un crecimiento constante y mejora de los productos o servicios que ofertan, este crecimiento y mejora puede verse reflejado en costos y ganancias, junto a la reputación de la empresa, la mejor forma de hacerlo es que las empresas sean capaces de conocer su productividad. Conocer la productividad de una empresa requiere un estudio de medición de la cantidad de productos producidos por la empresa.

Def: Se le conoce como **medición** al proceso de asignación de números o símbolos (previamente definidos) a entidades que se encuentran en el mundo real, entiéndase tangible(20).

La definición anterior provee un concepto útil en nuestra vida cotidiana; sin embargo, el software muchas veces se entiende como algo intangible y la forma de medirlo no sólo varia entre personas, también ha cambiado a lo largo de los años. Esto nos conduce a problemas

cuando queremos comparar resultados o incluso generar resultados nuevos hechos por personal diferente al que realizó las mediciones anteriores. Dentro de la definición general de medir, podemos encontrar dos tipos de medición(17):

- **Medición Directa:** Aquella que para medir un atributo del objeto en cuestión no necesita conocer la medida de otro atributo de ese mismo objeto.
- **Medición Indirecta:** Aquella que para conocer la medida de un atributo necesita conocer la medida de otros atributos.

Teniendo en cuenta los tipos de medición, podemos observar que la **medición de software** necesita de una medición indirecta; por otro lado, si queremos medir sólo el tamaño en bytes del software basta con saber el peso del archivo por lo que la medición sería directa.

La definición de **medición de software** ha variado a lo largo de los años y los diversos autores, algunos ejemplos de definiciones encontrados en artículos y libros son:

- **Def:** Disciplina basada en métricas como un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo software y los proyectos de mantenimiento(9).
- **Def:** Etapa de la base del ciclo de vida del software que ayuda a controlar los errores y carencias dentro del desarrollo y mantenimiento del software facilitando la toma de decisiones, convirtiéndose en un aspecto fundamental de la Ingeniería del Software(45).
- **Def:** Proceso del método de medición de modelo de productividad identifica los distintos pasos involucrados, desde el diseño de un método de medición hasta la explotación de los resultados de la medición en modelos posteriores como modelos de calidad y estimación(16).

2.3.1. Estimación

Def: Una **estimación de software** es un cálculo preliminar del costo del proyecto(46).

Este cálculo tentativo o preliminar se refiere a los recursos que deberán ser utilizados para la elaboración del proyecto, estos pueden ser costo monetario, cantidad de personas, étc. Sin embargo, se debe saber que una estimación no es un plan o un objetivo, las empresas de desarrollo trazan constantemente metas, si las metas no están bien fundamentadas estás no serán realizables, la estimación nos da una idea de cuántos recursos (tiempo, dinero, personas, esfuerzo, étc.) se necesita en el proyecto o si las metas que se buscan lograr son alcanzables, lo anterior permite trazar un plan para lograr alcanzar la meta trazada previamente(44).

Lo anterior deja ver la importancia de la estimación ya que determina los recursos necesarios convirtiéndose en la base para lograr metas realizables.

Al ser la estimación un cálculo debe tener un procedimiento o método que permite que sea repetible en los distintos proyectos, así como conocer su precisión, este proceso es conocido como **Proceso de estimación.**

2.3.2. Proceso de estimación

Diversos autores han descrito el proceso de estimación de manera semejante con pequeños cambios, para esta sección se tomó el proceso de acuerdo al autor Alain Abran(1). El proceso de estimación consta a grandes rasgos de los siguientes puntos:

1. **Recolección de entradas:** La primera fase del proceso incluye el poder identificar los requerimientos del proyecto, los recursos que serán utilizados en el proyecto (aquellos que puedan ser de bajo costo o puedan elevar el costo del proyecto) así como los límites del alcance del proyecto. En la Figura 2.3 se puede apreciar en amarillo los elementos de entrada para el proceso de estimación. En la parte superior en amarillo se está considerado la confiabilidad de estos datos, ya que de introducir datos no confiables, el proceso se verá afectado produciendo una estimación de dudosa confiabilidad.
2. **Ejecutar el modelo de productividad** Está fase tiene como entrada los elementos de la fase anterior y está compuesta por dos pasos:

   a) Los datos de la fase anterior son utilizados sin tomar en cuenta sus rangos de incertidumbre y son utilizados para construir al modelo de productividad [1]. Este modelo de productividad tendrá un desempeño, el cual será utilizado para especificar un rango de variación. En la Figura 2.3 se pueden apreciar estos elementos en la nube del Modelo de productividad y el repositorio de datos.

   b) Teniendo en cuenta los rangos de incertidumbre que fueron ignorados en el paso anterior se hace un ajuste a los rangos estimados de variación. En la Figura 2.3 se pueden apreciar este elementos en el cuadro que sale de la nube del Modelo de Productividad.

   Análogamente al paso anterior, se debe tomar en cuenta la confiabilidad en el Modelo de Productividad.
3. **Ajustes:** Ya que en el paso anterior se hizo un ajuste a los rangos de variación y de tener los factores de incertidumbre identificados, además de los datos en el proceso de estimación, se deben tomar en cuenta en los modelos de productividad por las siguientes razones:

   - El modelo de productividad está hecho con número limitado de variables, estas variables deben ser independientes en las ecuaciones matemáticas.
   - El no tener datos históricos o no identificar prontamente los factores de riesgo puedo impactar el ciclo de vida del proyecto, en está etapa se espera tener un panorama más amplio de los factores de riesgos para su pronta integración al modelo.
   - Inclusión de la siguiente información: generadores de costos y elementos de incertidumbre que no fueron detectados en la primera fase, identificación de riesgos y probabilidades de ocurrencia, además de puntos clave del proyecto. En está parte del proceso muchos equipos suelen incluir al juicio de experto para la identificación de riesgos y probabilidades de ocurrencia.

   Una vez concluido este proceso se obtiene un conjunto de valores usualmente representados por rangos. En la Figura 2.3 se observa en morado los elementos que toma el proceso de ajuste.

[1] Véase la siguiente sección para una explicación de los modelos de productividad

4. **Presupuesto y contingencias** Esta fase se selecciona un valor específico o conjunto de valores (sobre esfuerzo y duración) de los rangos propuestos en la fase anterior de ajuste para tomar una decisión sobre el presupuesto de asignación del proyecto. La selección de el valor o el conjunto de valores dependerá de la persona a cargo de tomar decisiones sobre el proyecto; en esta parte puede utilizarse el juicio de experto o algún método con sustento matemático. Alain Abran(1) menciona 3 tipos de personas en la toma de decisiones:

   a) Quien evita el riesgo selecciona un valor en el rango superior, siendo este escenario pesimista.

   b) Quien toma el riesgo selecciona un valor en el rango inferior, siendo este escenario optimista.

   c) Quien elige un punto intermedio analizando los rangos y sus probabilidades para poder seleccionar un presupuesto para el proyecto, reservan un fondo de contingencia en el caso de haber seleccionado un valor muy por debajo.

   En la Figura 2.3 los elementos de esta fase se encuentran en color azul. El círculo representa la persona a cargo del proyecto que debe tomar la decisión sobre el Presupuesto y los Fondos de Contingencia que se encuentran representados en la aparte de abajo. Los Fondos de Contingencia están señalados con dos colores, azul y naranja debido a que esta parte también es definida en la siguiente fase.
   Este proceso no puede ser más confiable que la confiabilidad de cada sub proceso y componente, análogamente es tan débil como su componente más débil(1).

5. **Re- estimación** El monitoreo de manera constante para verificar su progreso respecto a la estimación de tiempos, al presupuesto, cronograma y calidad esperada por el cliente, ayuda al proyecto a hacer frente respecto a la incertidumbre que se genera en el ciclo de vida del software. Poder mostrar resultados importantes al cliente de forma constante y continua genera tranquilidad y confianza en el proyecto. En la Figura 2.3 se observa en naranja los elementos de la fase de re-estimación.

6. **Mejoras en el proceso de estimación** La inclusión de esta fase en el proceso de estimación depende del autor, debido a que la fase no está directamente relacionada al administrador del proyecto; es decir, esta fase suele ser ejecutada por los administradores de la empresa, no por el administrador del proyecto. En la Figura 2.3 se observa en rojo los elementos de la fase de mejora de la estimación, como esta fase no siempre es considerada, podemos notar dos flechas más pronunciadas, si esta fase no es considerada, los elementos en rojo no existen y la fase de re-estimación conecta directamente al Proceso de Ajuste.

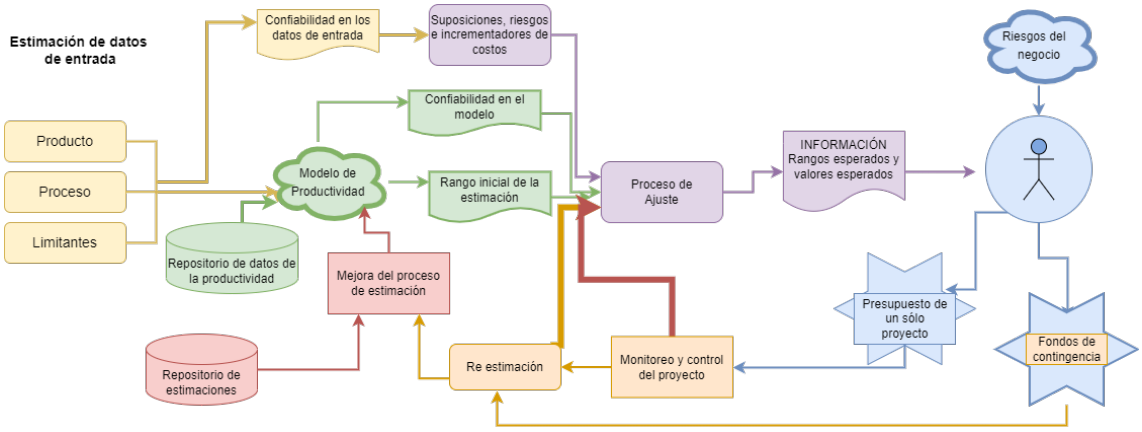

Figura 2.3: Fase del proceso de estimación adaptada de Alain Abrain (1)

En el capítulo 4 de la presente tesis se hará la propuesta de integración de este proceso de estimación (utilizando las fase 1 a la 4) junto con COSMIC en la metodología ágil SCRUM.

2.3.3. Modelos de Productividad

El concepto de **productividad** es definido como la relación de factores de productividad (bienes físicos iniciales y los recursos para construirlos o transformar los iniciales en finales), el término anteriormente se refería a la producción de recursos físicos siendo los bienes iniciales elementos físicos y el recurso obtenidos son representados como unidades monetarias, una relación producto-insumo(35).

Def: Un **modelo matemático** es una representación simplificada que utiliza ecuaciones, funciones o fórmulas matemáticas, de un fenómeno o de la relación entre dos o más variables(3).

Un modelo de productividad permite relacionar las variables dependientes de interés en el proyecto como pueden ser esfuerzo, costo, personas del proyecto o duración del proyecto con las variables independientes como son el tamaño del producto o las características del producto(1). En un modelo de productividad podemos identificar las siguientes partes:

- **Variables independientes:** Estás variables pueden ser incorporadas sin que afecten a la ecuación, más bien, según cuales sean sus valores, las variables dependientes de ellas serán afectadas.
- **Variables dependientes:** Estás variables son calculadas de acuerdo al valor de variable independiente a la que están asociadas, es decir, cambian respecto del valor de las variables independientes.
- **Representación de los datos históricos:** Representados como puntos en la gráfica de dispersión (x,y), se consideran los datos de proyectos pasados para generar un modelo de correlación, ya que se conocen sus variables independientes y dependientes.

- **Ecuación matemática del modelo:** Esta ecuación sirve para relacionar los variables independientes con las dependientes a partir de los datos históricos que se tengan en ese momento. Está ecuación no es estática; es decir, puede ir cambiando conforme se introduzcan más datos históricos al modelo. Esta ecuación es conocida como **Modelo de productividad**(1).

En la Figura 2.4 se aprecia un modelo de productividad con una variable independiente. El tamaño del software es utilizada como variable independiente representada en el eje de las abscisas y el esfuerzo es utilizado como variable dependiente al tamaño, representado en el eje de las ordenadas. Los datos históricos son representados mediante puntos azules en la imagen, cada uno es un par ordenado ($tamaño\ funcional\ del\ proyecto$, $esfuerzo\ realizado$). Finalmente la recta que se observa representa a la ecuación matemática que relaciona el tamaño del software con el esfuerzo requerido, está recta es representada por la función $y = 1.2483x + 0.6421$.

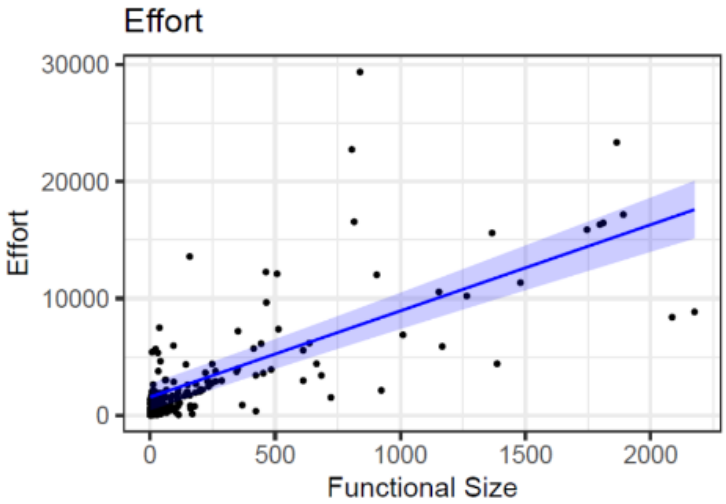

Figura 2.4: Ejemplo de un Modelo de productividad adaptado del artículo Improving the Software Estimation Models Based on Functional Size (57)

Algunas de las ventajas que tienen los modelos de productividad frente al "Juicio de experto" son:

- Presentación de un enfoque formal de la estimación contra el enfoque informal del juicio de experto.
- Documentación clara y precisa sobre las variables usados en la creación de los modelos de productividad y cómo estas variables se relacionan.
- Basados en la selección de variables cuantitativas medidas con precisión en función de lo que se ha implementado en el software; es decir, en información cuantitativa con parámetros bien definidos. Esto permite la replicación de resultados para futuras consultas.

El poder medir dentro del software nos permitirá hacer mejores estimaciones, proporcionando mayor confianza en las estimaciones ya que se tienen datos cuantitativos, generar una aproximación de los recursos necesarios para el proyecto evitando costos excesivos.

2.3.4. ¿Las estimaciones son confiables?

Diversos expertos en el tema han calificado a las estimaciones generadas a través de los años como buenas o malas, cada uno ha utilizado diferentes medidas para considerar la efectividad de cada medición, incluso se puede encontrar diversas definiciones de una buena definición como son:

- En 1998 Carpers Jones planteó que una buena estimación es posible posible con una precisión del 10 %, esto sólo cuando se tienen proyectos bien controlados.
- Un buen enfoque de estimación debe proveer estimaciones estén dentro del 25 % de los resultados reales el 75 % del tiempo(12).
- Una buena estimación es una estimación que provee una visión lo suficientemente clara de la realidad del proyecto permitiendo al líder del proyecto tomar buenas decisiones para el control del proyectos y alcanzar sus objetivos(44).

Samuel Daniel Conte, en el libro Software engineering metrics and models(12) expuso criterios que proporcionan información sobre el desempeño de un modelo de productividad, estos son:

1. **Coeficiente de determinación** (R^2) Determina en porcentaje (o con valores entre 0 y 1) si existe una relación fuerte entre las variables independientes y dependientes. Si el valor es más cercano a 1, la relación entre las variables es fuerte y ponemos decir que el modelo se ajusta a lo ocurrido en la realidad proporcionando una mayor confiabilidad. Caso contrario si el valor es más cercano a 0.
2. **Error de estimación (E):** es el valor obtenido de restar el valor actual al valor estimado de la variable dependiente, esto en cada proyecto. Se observa que es el criterio más simple donde se está comparando que tan alejada fue la estimación del valor obtenido en un tiempo especifico, denominado como tiempo actual en la formula
   $Error = E = Valor_{actual} - Valor_{estimado}$
3. **Error Relativo (RE):** similar al criterio anterior, se está dividiendo el error de estimación entre el valor actual para encontrar la proporción entre los valores estimados por el modelo y los valores reales, el RE puede ser positivo lo cual indicaría una subestimación, caso contrario si es negativo indicaría una sobrestimación.
   $ErrorRelativo$: $RE = \frac{valor_{actual} - valor_{estimado}}{valor_{actual}}$
4. **Magnitud del Error Relativo (MRE):** la magnitud del error relativo indica que tanto difiere los valores estimados por el modelo de los valores reales obtenidos,esto con relación al valor real, expresado en forma de porcentaje. Por otro parte en la magnitud media del error relativo se toma en cuenta el valor de cada proyecto que exista en los datos históricos del modelo.
   $Magnitud\ del\ error\ relativo : MRE = |RE| = |\frac{valor_{actual} - valor_{estimado}}{valor_{actual}}|$
   $Magnitud\ media\ del\ error\ relativo : MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i$

Al pasar de los años se han considerado más criterios generalmente derivados de la estadística como el RMS(61), P-valores(25), Balance del Error Relativo (BRE)(60), Magnitud mediana del error relativo (MdMRE)(7), Predicción a nivel (N), (PRED(N)) (39), por mencionar algunos ejemplos.

2.4. Estándares de medición de tamaño funcional

El software es medido por una razón especifica, lo que se deseé medir corresponde a una meta o necesidad que se tiene, estas necesidades o metas varían de acuerdo a la persona que requiera la información, por ejemplo, los desarrolladores buscan cómo mejorar su manera de crear código o saber qué tan bueno es su código, esto difiere con los gerentes de proyecto que buscan saber qué tan productivo es el equipo de trabajo que tienen a cargo, cuáles son los costos del proyecto.

2.4.1. Qué se mide en el software

El termino métricas de software engloba distintas actividades como: colecciones de datos, modelos de calidad y medidas, costo y estimación de esfuerzo en los modelos y medidas, evaluación de métodos y herramientas, gestión por métricas, étc(18).

En diversos artículos la definición de métrica y medida eran intercambiables pero a partir del estándar IEEE 610.12, provee la siguiente definición que es la que se usará en la presente tesis:

Def: Una **métrica del software** es una medida cuantitativa del grado en el que un sistema, componente o proceso dispone de un atributo dado(51).

Roger S. Pressman(51) expresa una diferencia entre métricas de software y métricas del proyecto, refiriéndose a estas últimas como tácticas que permiten adaptar el flujo del proyecto y las actividades técnicas que requiere, además propuso diferentes grupos para clasificar a las métricas de acuerdo a ciertos criterios, algunos de los grupos son:

1. **Métricas técnicas:** Basadas en normalizar las características del software, es importante que puede ofrecer el software; sin embargo se deja de lado todo el proceso de desarrollo de software.
2. **Métricas de calidad:** Estas métricas se refieren tanto a la calidad del producto las cuales son obtenidas gracias a los requerimientos no funcionales solicitados por el cliente; así como a la calidad en el ciclo de vida del software.
3. **Métricas de productividad:** Estas métricas están enfocadas al rendimiento en el ciclo de vida del software, qué tan eficiente es el esfuerzo que se está aplicando para lograr cumplir en tiempo y forma su ciclo de vida previamente planeado.
4. **Métricas orientadas a la persona:** Obtenidas con base al grupo de personas que está trabajando en el desarrollo del software, cómo es su relación respecto al proyecto (entregas, esfuerzo, errores, étc.), se pretende evaluar a cada integrante con el fin de poder mejor individualmente, lo cual llevará a una mejora en el equipo de desarrollo.
5. **Métricas orientadas al tamaño:** Obtenidas gracias al proceso de normalizar las medidas de calidad y/o productividad; es decir, al tener una colección de datos de proyectos pasados se pueden crear pautas con base a ellos para poder medir el tamaño del software.

6. **Métricas orientadas a la función:** Obtenidas gracias al proceso de normalizar la medida de la funcionalidad que tenga la aplicación final. La medida anterior que mas fue usada fueron los "Puntos de Función (Function Points)". Contrario al grupo anterior de métricas orientadas al tamaño, este grupo pretende la abstracción de los lenguajes de programación para sólo concentrarse en la funcionalidad.

Algunos ejemplos de métricas que han sido utilizadas son:

- **Cantidad de lineas de código**(8), siendo la métrica más usada cuando los lenguajes de programación como COBOL estaban en auge, permitía saber la carga de ejecución de un programa gracias a la cantidad de lineas de código.
- **Datos de proyectos pasados**(37), Keaveney y Conboy en 2006 utilizaron está técnica donde el objetivo consiste únicamente en un análisis exhaustivo de los proyectos pasados.
- **Encuestas de 13 preguntas**(4), propuesta por Andreas Schmietendorf y et. al. en 2008, estás preguntas engloban aspectos fundamentales del proyecto.
- **Story Points**(10), propuesto por Coelho and Basu en 2012, propusieron un nuevo concepto que además permite destacar el área que se necesite.

Def: Los **estándares** son objetos o pautas de comparación que definen o representan la magnitud de una unidad. Caracterización que establece límites, tolerancias o restricciones para categorías de objetos medibles(29).

La organización IEEE (Institute of Electrical and Electronics Engineers) provee estándares para que la comunidad pueda desarrollar software de calidad. En su sección Testing, Instrumentation and Measurement, and Metric Practice para las Tecnologías de la información, cuenta con 204 estándares publicados.

Por otro lado existe la organización ISO (International Standarization Organization) fundada en 1946 junto con la comisión IEC (the International Electrotechnical Commission) forman un sistema especializado de estandarización a nivel mundial, permitiendo mejorar procesos de las empresas, gestión de calidad en los servicios o productos ofrecidos por estas empresas, además de que las empresas al tener una certificación avalada mundialmente, les otorga confianza a sus clientes. Cuentan con más de 23 000 estándares enfocados en diversas áreas de gestión, tecnologías y procesos de producción.

Estos estándares permiten saber cuales son los atributos del software que son deseables para obtener un software de calidad y sobre los cuales podemos aplicar las métricas que permiten evaluar el desempeño de estos. A diferencia de las métricas, son pautas que permiten homogeneizar (estandarizar) el desarrollo de software, pretendiendo facilitar su mantenimiento, crecimiento y actualización.

La relación entre las métricas y los estándares es la siguiente: Al querer medir un objeto, debemos saber cuales son los atributos de este objeto para saber sobre cuál atributo vamos a medir, así con una métrica, que es una medida cuantitativa, podremos medir el atributo seleccionado, y le podremos expresar mediante una unidad de medida, la cual es una determinada cantidad definida y adoptadas por convenio (estándar), por lo tanto es comparable(29). Esto se puede apreciar en la Figura 2.5.

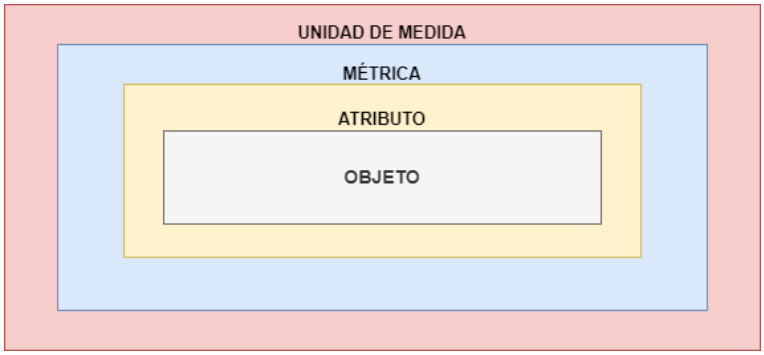

Figura 2.5: Relación de las métricas con los estándares

2.4.2. Historia de los Estándares de Medición Funcional de Software

A lo largo de la historia mientras la Ingeniería de Software se ha ido desarrollando y ha ido buscando el título de ingeniería, se buscan diversas formas de medir los alcances que se puedan generar. Horst Zuse en su libro A Framework of Software Measurement estima que desde el año 1980 se tienen registros sobre discusiones en el ámbito de la medición de software, esto principalmente fue motivado por el aspecto económico. A lo largo de los años diversos métodos han sido creados para satisfacer la necesidad de medir al software(24) (62). En este momento uno de los recursos para estimar era contar la cantidad de líneas del código fuente para estimar el tamaño del programa(8).

El primer método parecido a un estándar de medición funcional del software fue creado a finales de los años 70 por Allan J. Albrecht conocida como Function Points (IFPUG Method)(49), el método se basa en crear "puntos" llamados de función al software que se desarrolla dependiendo a la complejidad de datos y procesos que el usuario podrá realizar, además de considerar la funcionalidad que va a ser implementada calculando el tamaño y costo; también sirve para estimar proyectos de desarrollo y mantenimiento de software. En algunos casos se puede usar estimando la funcionalidad recibida del usuario para corroborar si el estimado de costo y tamaño son correctos. Uno de los principales puntos a favor que tenía era que el cálculo era independiente de la tecnología usada en el desarrollo(49).

En *1982* Tom DeMarco desarrolla el método conocido como DeMarco Bang Metrics que clasifica a los sistemas de software en dos grupos: los orientados a funciones y los orientados a datos. De acuerdo al grupo donde el software a evaluar estuviera, se estimaba el tamaño del software gracias a la descripción de la estructura visualizada durante el proceso de levantamiento de requerimientos con la ayuda de un algoritmo. Este método al estar enfocado en el usuario es independiente de la tecnología usada en el desarrollo del software(52).

En *1986* surge el método de Feature Points (Puntos característicos) propuesto por Capers Jones, este método es considerado como una extensión del método de Albrecht, el método utiliza un algoritmo adicional para cambiar los pesos de los componentes en el método Function

Points original. Originalmente fue una alternativa para utilizar el método de puntos de función en software científico y de ingeniería. Este método tuvo problemas con la estandarización del algoritmo auxiliar por lo que su eficiencia fue puesta en duda(36).

Charles Symons crea en *1988* el método creado Mark II FPA, en 1991 cuando pasó a ser un método público. Consiste en medir el tamaño funcional a través de "transacciones lógicas"que son realizadas entre componentes de entrada, salida y de procesos. Por esta razón, no es recomendable en software de carácter científico que suele contener algoritmos complejos, pero fue diseñado para software de negocios donde se requiere consultas y almacenamientos constantes de datos. Finalmente a finales de los 90 el método fue certificado como parte del estándar ISO/IEC 20926(33).

1989 el método Data Points es creado por Harry Sneed, basado completamente en Feature Points con la única diferencia de que este está enfocado en el paradigma de la programación orientada a objetos(24).

1990 Se desarrolla el método conocido como NESMA FPA por la asociación holandesa Netherlands Software Metrics Users Association, está basado en el conteo de puntos de función en el software, como en el FPA utilizando los elementos: entradas externas, salidas externas, consultas externas, archivos lógicos internos y archivos de interfaz externa. La diferencia con el método FPA es que está certificado por ISO como el estándar ISO/IEC 24570(34) (6).

1992 3D Function Points fue el método diseñado por Scott A. Whitmire, originalmente comenzó a ser desarrollado en 1989 pero se dio a conocer oficialmente en 1992. A diferencia de los métodos anteriormente mencionados, tiene como propósito medir el Tamaño Funcional en aplicaciones de negocio y tiempo real. Está basado en en Function Points por lo que también es independiente de la tecnología usada en el desarrollo del software. Se incorporan dos conceptos nuevos que son “transformaciones” y “transiciones”. El punto en contra que tiene, es que se debe tener un alto conocimiento sobre el sistema (algoritmos que se implementaron, flujo de control, datos que interactúan en él), por lo que no es viable de usar en las primeras fases del ciclo de vida del software; por lo tanto, no es posible usarlo para estimar(42).

1993 se desarrolla el método Use Case Points UCP por Gustav Karner, originalmente es desarrollado para medir proyectos en sus fases iniciales. Consiste en evaluar la complejidad de los actores que interactúan con el sistema y asignarles un número 0,1,2 llamado factor de acuerdo a si el usuario interactúa con una interfaz de usuario o no. Fue diseñado para sistemas con el paradigma orientado a objetos.

1994 diversas mezclas de métodos surgen, muchas de estas mezclas toman como base el modelo de FPA(49) de Albretch como fue le caso de Matson, Barrett y Mellichamp quienes propusieron un método caracterizado por una combinación lineal de 5 componentes en el software: entradas, salidas, archivos claves, interfaces y recursos(43).

1997 Full Functions Points (FFP) es diseñado por la Universidad de Quebec junto con el Laboratorio de Ingeniería de Software para Aplicación de Metricas, fue uno de los primeros métodos para medir sistemas embebidos, también conocidos como empotrados, además también podía estimar software en tiempo real. La principal diferencia que se encuentra en este método es que no sólo observa los procesos, sino también toma en consideración los sub-procesos.

Este estándar fue el primero en establecer las metas para la Medición del Tamaño Funcional (FSM). Una vez que el estándar fue publicado, diversos métodos se adecuaron a él.

1999 Georges Toeloglou desarrolla el método Predictive Object Points, enfocado a software hecho con el paradigma orientado a objetos. El método se caracteriza por la estimación en la herencia de clases dentro del software, evaluación de sus comportamientos entre clases y efectos que puedan provocar sobre otras, dando le un peso a cada método existente en cada clase. Los métodos eran clasificados en 5 tipos: Constructores, Deconstructores, Selectores, Modificadores e Iteradores(55).

Todos los métodos anteriores son considerados como métodos de primera generación, creados a partir técnicas estadísticas de proyectos anteriores.

En *1997* fue publicado el primer borrador sobre Medida de Tamaño Funcional considerado de carácter oficial que formaba parte de los estándares ISO con la clave ISO/IEC 14143 (31). Una vez definido el estándar ISO/IEC 14143, cuatro métodos de primera generación se adecuaron al nuevo estándar.

Finalmente en Diciembre de *2002* se acepta un nuevo método de medición de software de tamaño funcional, el estándar ISO/IEC 19761 o método COSMIC, siendo creado en un consorcio internacional Common Software Mwasurement Consortium, que nos abre camino a la segunda generación de métodos de medición de tamaño funcional. ISO/IEC fue escrito bajo las bases del estándar ISO/IEC 14143 además del conocimiento generado por los antiguos métodos de medición de software, permitiendo lograr un mayor alcance en la aplicación de los métodos de medición, como es el caso de software de mayor complejidad, incluso algunos tipos de software científico permitiendo extender el dominio de su aplicación y una escala de medición más homogénea, lo cual es una ventajas sobre los métodos de primera generación(32).

2.5. COSMIC (ISO/IEC 19761)

COSMIC es un método que consiste en aplicar un conjunto de modelos y principios, definido por reglas y procesos para medir **Requisitos Funcionales del usuario** (abreviados como FUR[1]) de una determinada pieza de software, resultando un valor numérico que representa el tamaño funcional de una pieza de software según el método COSMIC(2). Fue desarrollado en y por la industria para abstraerse de patrones de software, lenguajes y sólo centrarse en lo que el software puede hacer. Al ser un método de segunda generación puede ser aplicado en un más amplio espectro de tipos de software como: multi-capa, de tiempo real, herramientas de usuario e incluso algunos tipos de software científico. Además resuelve los problemas de los métodos de la primera generación que son:

1. Tiene una unidad de medida homogénea y comparable.
2. No se basa en datos estadísticos sino en la representación del software.
3. Cumple las reglas matemáticas para manejo de escalas de los elementos medidos.

[1]En toda esta sección y a lo largo de la tesis presente, se utilizará la abreviatura FUR

2.5.1. Glosario COSMIC

Se definirán unos conceptos que la metodología COSMIC maneja a continuación, estos conceptos serán utilizados a lo largo de esta tesis, los cuales fueron obtenidos del Manual Oficial de COSMIC *COSMIC Measurement Manual*(14).

- **Aplicación:** Software para la recopilación, el almacenamiento, el procesamiento y la presentación de datos a través de un ordenador(14).
- **Frontera:** Interfaz conceptual entre el software que se está midiendo y sus usuarios funcionales(14).
- **Componente:** Cualquier parte de un sistema de software que sea independiente por razones de la arquitectura de software, y que se especificó, diseñó o desarrolló por separado(14).
- **Comando de control:** Un comando que permite a los usuarios funcionales humanos controlar el uso del software pero que no implica ningún movimiento de datos sobre un objeto de interés(14).
- **Evento:** Algo que sucede(14).
- **Evento Desencadenante:** En evento reconocido en los FUR del software que se está midiendo, genera que uno o más usuarios funcionales del software que se está midiendo generen uno o más grupos de datos. No puede ser sub-dividido(14).
- **Manipulación de Datos:** Cualquier cosa que le ocurre a los datos que no sea un movimiento de datos dentro o fuera de un proceso funcional, o entre un proceso funcional y el almacenamiento persistente(14).
- **Modelo:** Una descripción o analogía utilizada para ayudar a visualizar un concepto que no se puede observar directamente(14).
- **Nivel de descomposición:** Cualquier nivel que resulte de dividir una pieza de software en componentes(14).
- **Nivel de granularidad:** Cualquier nivel de expansión de la descripción de una sola pieza de software(14).
- **Objeto de interés:** Cualquier "*cosa*" en el mundo del usuario funcional que se identifica en los FUR sobre el software que se requiere para procesar datos(14).
- **Patrón de Medición:** Una plantilla estándar que se puede aplicar en la medición de una pieza de software desde un dominio funcional de software, que define los tipos de usuario funcional que pueden interactuar con el software, nivel de descomposición y tipos de movimientos de datos que se puede manejar(14).
- **Pieza de software:** Cualquier elemento de software en cualquier nivel de descomposición desde el nivel de un sistema de software(14).

COSMIC fue diseñado basado en 17 principios de la ingeniería de software, los cuales están divididos generando dos modelos, los cuales fueron obtenidos del Manual Oficial de COSMIC *(COSMIC Measurement Manual)*(14):

- **Modelo de Contexto de Software:** Permite a la persona a cargo de medir el software la definición de la pieza de software a ser medida así como su tamaño, permitiendo la identificación de los Requisitos Funcionales a usar. Este modelo contiene los principios:

1. Una aplicación de software se estructura en capas.
2. Una capa puede contener una o más piezas de software, estás pueden ser independientes.
3. Cada pieza de software a medir debe ser descrita mediante sus Requisitos Funcionales del Usuario.
4. Los Requisitos Funcionales del Usuario se expresan en un nivel de granularidad suficiente para exponer sus procesos funcionales, además de utilizar el mismo nivel de granularidad para cada pieza de software.
5. Una pieza de software ofrece funcionalidad a sus usuarios funcionales como se identifica en los Requisitos Funcionales del Usuario.
6. Las piezas de software que serán medidas se definen de acuerdo al alcance de la medición del tamaño funcional, este alcance además está delimitado dentro de una sola capa.
7. El alcance de la Medición del Tamaño Funcional define los procesos que serán medidos y depende del propósito de la medición.

- **Modelo de Software Genérico:** Define como los Requisitos Funcionales del Usuario que se consideran en el alcance de la medición se deben modelar para poder ser medidos por el método COSMIC.

1. Cualquier pieza de software interactúa con sus usuarios funcionales a través de un límite con almacenamiento persistente dentro de ese límite.
2. Todo proceso funcional consiste en subprocesos denominados movimientos de datos.
3. Existen cuatro tipos de movimientos de datos: Entrada, Salida, Escritura y Lectura. Además, un subtipo de movimiento de datos incluye una manipulación de datos asociada.
4. Un movimiento de datos mueve un único grupo de datos.
5. Un grupo de datos consta de un conjunto único de atributos de datos que describen a un único objeto de interés.
6. Cada proceso funcional es iniciado por un evento desencadenante, detectado por un usuario funcional y que a su vez inicia un movimiento de datos llamado Entrada Desescadenante.
7. El tamaño funcional se basa en los tipos de movimientos utilizados para la medición, no en el número de ocurrencias.
8. El tamaño de un proceso funcional es igual al número de sus movimientos de datos donde cada movimiento de datos tiene un tamaño de 1 COSMIC Function Point CFP.
9. El tamaño de una pieza de software es la suma de los tamaños de los procesos funcionales dentro del alcance del FUR.
10. El tamaño de una pieza de software modificada es la suma de los tamaños de los procesos funcionales modificados, agregados y eliminados.

2.5.2. Proceso de Medición con COSMIC

El proceso de medición de COSMIC está dividido en 3 fases: Fase de estrategia de Medición, Fase de Representación y Fase de Medición. En la Figura 2.6 se observa la interacción entre las 3 fases, el proceso fue obtenido del Manual Oficial de COSMIC(14).

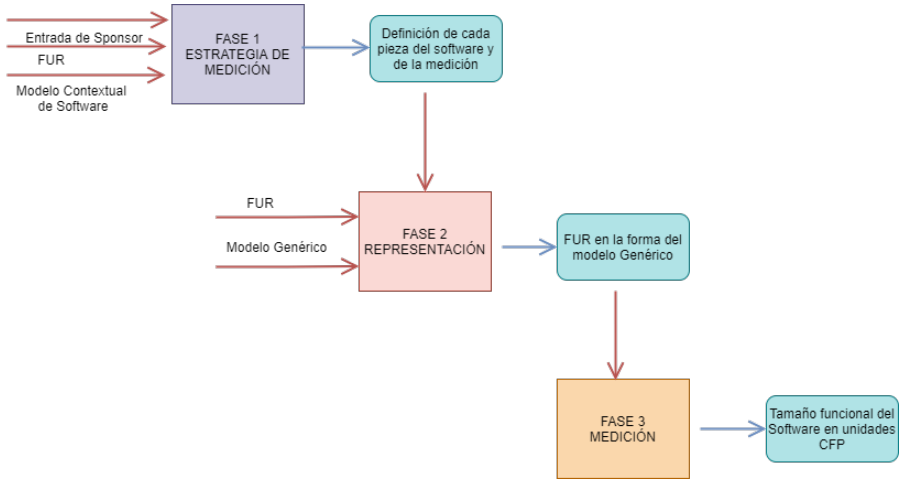

Figura 2.6: Proceso Estimación en COSMIC adaptado del Manual Oficial de COSMIC

En cada fase los procesos son diferentes y se explicarán a continuación:

1. **Fase de estrategia de Medición.** Al ser la primera fase en el proceso de medición de COSMIC, esta fase se alimenta del *Patrocinador de la medición*, donde contestar la pregunta ¿Para qué se va a utilizar la medición? proporciona el propósito de la medición y permite determinar el alcance de la medición; es decir, los objetos que serán necesarios para la medición, además de las piezas de software que serán medidas junto con el nivel de precisión necesaria. Identificación de los usuarios funcionales que interactuarán con las piezas de software seleccionadas, momento en que el proyecto se encuentra dentro del ciclo de vida del software.

   La *Identificación de los usuarios funcionales* también nos permitirá la *Identificación de los Requisitos Funcionales* permite conocer junto con el propósito las piezas de software que serán medidas. Los requisitos funcionales de un proyecto se conocen antes de que el software sea implementado, por lo que el tamaño funcional del software puede ser medido incluso si aún no se encuentran implementadas las piezas de software. Aquellos requisitos que no son funcionales a pesar de aportar un valor significativo al proyecto no contribuyen al tamaño funcional del software por lo que COSMIC no los toma en cuenta para su medición.

   COSMIC permite la elaboración de diagramas de contexto para explicar visualmente el sistema una vez identificados los puntos anteriores. En la Figura ??.A se observan los elementos para generar el diagrama de contexto y en la Figura ??.B se observa un

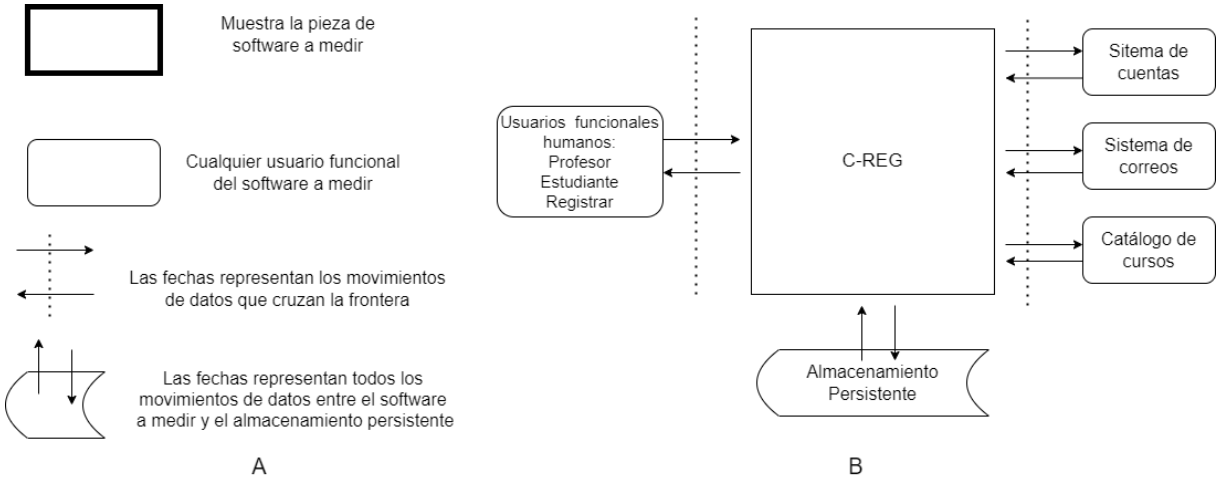

Figura 2.7: Diagramas de Contexto en COSMIC. Del lado izquierda A denota los elementos básicos para entender el Diagrama de Contexto. Del lado derecho B muestra el Diagrama de Contexto

ejemplo de un diagrama de contexto de un software que se desea medir llamado C-REG que almacena información de los profesores de una escuela y los cursos que cada profesor tiene a cargo, que es un sistema de referencia utilizado para varias metodologías como UML y COSMIC. Se identificaron como usuarios funcionales los Profesores, Estudiantes y un sistema llamado Registrar, además de la interacción del sistema C-REG con otros sistemas que son Sistema de Facturación y Sistema de correos.

2. **Fase de Representación.** Al final de la fase anterior se definieron las piezas de software que se utilizarán en la medición, éstas serán las entradas que se reciban para la fase de Representación, ademas de los FUR. El primer paso de esta fase consiste en la *Identificación de Procesos funcionales* para entender como funcionan los movimientos de datos; en la Figura 2.8 se aprecia como están compuestos, inician con una entrada desencadenante que es identificada por los usuarios funcionales que se identificaron en la fase anterior, este evento desencadenante genera un grupo de datos que pasarán a través de la frontera del usuario al software donde serán procesados de acuerdo al proceso funcional solicitado.

   El segundo paso de esta fase consiste en la *Identificación de los grupos de datos*, estos grupos de datos describen a un único objeto de interés, adicionalmente y de manera opcional se puede *Identificar los atributos de los datos* que podrían de ser importantes en caso de requerir cambios en el software; sin embargo la parte esencial de los grupos de datos es para el siguiente paso *Identificación de los movimientos de datos.*

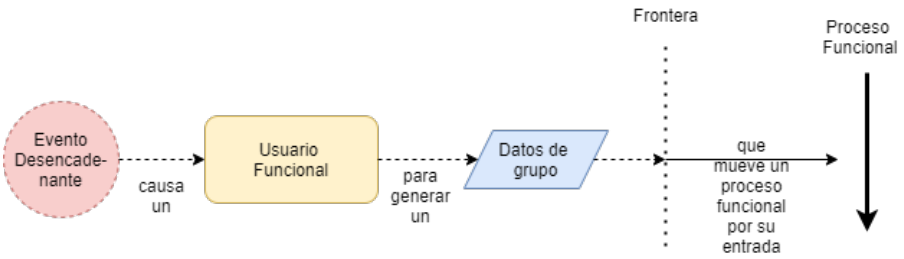

Figura 2.8: Proceso Funcional adaptado del Manual Oficial de COSMIC

Como se mencionó en los 17 principios, existen 4 tipos de movimientos de datos. En la Figura 2.9 se pueden apreciar.

- **Movimientos de Entrada (E):** Mueve un único grupo de datos que describe a un objeto de interés desde el usuario a través de la frontera a el proceso funcional donde se requiere(14).
- **Movimientos de Salida (X):** Mueve un único grupo de datos de un sólo objeto de interés desde un proceso funcional a través de la frontera a el usuario funcional que lo requiere.
  Los mensajes de error o confirmación con texto fijo no serán considerados como salidas a menos que muevan un grupo de datos(14).
- **Movimientos de Lectura (R):** Mueve un único grupo de datos de un sólo objeto de interés desde un almacén persistente al proceso funcional que lo requiere(14).
- **Movimientos de Escritura (W):** Mueve un único grupo de datos a un almacén persistente que se encuentra dentro del proceso funcional.
  Borrar o actualizar un grupo de datos del almacén persistente se medirá como una escritura, sin embargo la creación o actualización de variables o resultados intermedios que son internos al proceso funcional no se consideran como movimientos de escrituras(14).

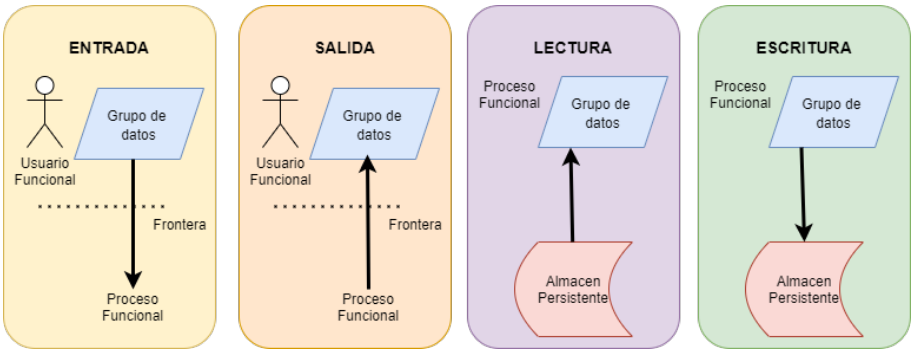

Figura 2.9: Tipos de Movimientos de Datos

3. **Fase de Medición(14).** Finalmente en está fase se aplica la unidad de medición COSMIC 1 CFP por cada movimiento de datos. Así el tamaño funcional de una pieza de software se obtiene de la siguiente forma: Tamaño funcional $= \sum tam(E) + \sum tam(X) +$

$\sum tam(R) + \sum tam(W)$.

COSMIC también permite dimensionar los cambios. Un *Cambio Funcional* del software existente se interpreta como cualquier combinación de sumas de nuevos movimientos de datos, modificaciones o eliminaciones de los movimientos de datos existentes(14).

Los resultados de la medición deben ser expresados como "$xCFP(v)$" en donde: x representa el valor numérico del tamaño funcional, y v representa la identificación de la versión del estándar del método COSMIC. Adicionalmente se deberá identificar la componente del software medido con nombre y versión, identificar las fuentes de información usadas para la identificación de los FUR, documentación del software donde se describa la arquitectura y descripción del software, además de colocar el propósito de la medición.

2.5.3. Proceso de Aproximación con COSMIC

Tener los requerimientos completos de un proyecto permite medirlos, pero si los requisitos no se encuentran completos o no están bien detallados, como sucede en las etapas temprana de cualquier desarrollo de software, no podremos medir; sin embargo, podemos estimar, o juzgar. A esto se le denomina **aproximación del tamaño funcional** en COSMIC.

2.5.3.1. Conceptos Básicos

Antes de que los requisitos funcionales se haya elaborado con el detalle necesario para una medición precisa, se se requiere una estimación del esfuerzo del proyecto. En la documentación oficial de COSMIC existe una guía llamada *Early Software Sizing with COSMIC*(13) que provee una forma de aproximar el tamaño funcional de acuerdo a ciertos criterios que se puedan presentar en los requisitos como pueden ser:

1. Cuando se necesita una medida rápidamente de tamaño y un tamaño aproximado es aceptable si puede realizarse mucho más rápido que con el método estándar. Esto se conoce como ***dimensionamiento rápido (Rapid Sizing)***.
2. En el inicio de la vida de un proyecto, antes de que los requisitos reales hayan sido especificados con algún detalle pero siendo insuficientes los detalles para una medida exacta del tamaño. Esto se conoce como ***dimensionamiento temprano (Early Sizing)***.
3. Cuando la calidad de la documentación de los requisitos reales no es buena o es insuficiente para una medición precisa del tamaño.

Una estimación por aproximación rápida se basa en dos partes: **Escalamiento** y **Calibración**(13).

Escalamiento(13): El principio general de cualquier técnica de escalamiento es encontrar alguna manera de medir el tamaño aproximado de los artefactos definidos de los requisitos reales a un alto nivel de documentación y luego medir los mismos requisitos en unidades de CFP cuando se conocen en el nivel de proceso funcional de la documentación.

Calibración(13): Las técnicas de aproximación están basadas en artefactos no estandarizados, los cuales pueden variar en sus niveles de detalles funcionales, por lo que los factores de escala deben calibrarse localmente; es decir, se deben alinear a los factores de escalamiento previamente definidos, ya que estos factores son representativos del entorno a ser utilizado.

La guía *Early Software Sizing with COSMIC*(13) divide las técnicas de aproximación rápida en 2:

- **Técnicas para la fase de viabilidad(13):** pueden ser usadas cuando la integridad de los requisitos se debe tomar en cuanta por no estar completamente definidos.
    1. Analogía de ICEBERG de software
    2. Aproximación COSMIC temprana y rápida
    3. Aproximación de Puntos de fusión EASY (EARLY AND SPEEDY)
    4. Modelo EPCU o Aproximación Mediante Lógica Difusa
- **Técnicas para la etapa de requisitos(13):** pueden ser usadas cuando los requisitos ya han sido identificados y no existen requisitos desconocidos
    1. Tamaño Medio de Procesos funcionales
    2. Clasificación de tamaño fijo
    3. Bandas de tamaño similar
    4. Promedio de casos de uso
    5. Patrones de Medición de Tamaño funcional

Salvo la técnica de Tamaño Medio de Procesos funcionales que requiere una medición, las demás técnicas requieren de una estimación.

COSMIC posee una definición de **Calidad de los Requerimientos** la cual nos indica el grado detalle en que los requisitos funcionales están escritos. La guía Early Software Sizing with COSMIC propone una clasificación de 6 niveles en los cuales pueden encontrarse los requisitos funcionales, Figura 2.10. Esta clasificación ayuda a saber si es posible estimar o medir dependiendo del nivel donde los requisitos funcionales se encuentren.

Nivel del Proceso Funcional	Definición de Calidad del Proceso Funcional	Medir o estimar
COMPLETAMENTE DEFINIDO	El proceso funcional y sus movimientos de datos están completamente definidos	Se puede medir usando el éstandar 19761 CCSMIC
DOCUMENTADO	El proceso funcional está documentado pero no con el suficiente detalle para identificar los movimientos de datos	Se puede estimar
IDENTIFICADO	El proceso funcional está documentado pero no con el suficiente detalle para identificar los movimientos de datos	Se puede estimar
CONTADO	Se ha dado un conteo de los procesos funcionales, sin conocer más detalles	Se puede estimar
INCLUIDO	El proceso funcional está incluido en los requisitos actuales pero no es mencionado explicitamente.	Se puede estimar
NO MENCIONADO	La existencia del proceso funcional es completamente desconocido en el presente	Se recomienda utilizar datos de proyectos pasado o juicio de experto

Figura 2.10: Niveles en la calidad de los requerimientos. Adaptada de Early Software Sizing with COSMIC Guide.

En las siguiente secciones se explica cada una de las técnicas de la etapa de requisitos o de viabilidad, de acuerdo al orden en que fueron mencionadas anteriormente, estas fueron adaptadas de Early Software Sizing with COSMIC Guide(13).

2.5.3.2. Técnicas para la etapa de viabilidad

Analogía de Iceberg de Software(13)

Descripción: Esta técnica se basa en la **analogía del iceberg** siendo la punta del icerberg la descripción temprana de los requisitos, hasta lograr la visibilidad completa del iceberg debajo de la linea de agua y la *Norma ISO-IEEE29148*, el cual presenta una serie de conceptos relacionados con las fuentes, tipos y niveles de detalle de los requisitos a lo largo del ciclo de vida del sistema y del software.

Consiste en seleccionar un conjunto inicial de requisitos dividido en dos, la primera parte son los requisitos de las partes interesadas del negocio y la segunda parte son los requisitos de otras partes interesadas, lo que conduce a los requisitos de los "sistemas". A partir de los requisitos funcionales del sistema, algunos se asignarán a los requisitos de software (así como a los requisitos de hardware y a veces a los procedimientos operativos manuales). Estas fuentes proporcionan los requisitos contextuales del sistema, incluidos el propósito del sistema, el alcance del sistema y la visión general del sistema. Así se identifican los requisitos funcionales del sistema (algunos de los cuales se asignarán al software) y los requisitos no funcionales. ISO-IEEE 29148 también señala que además de las funciones de software identificadas explícitamente, puede haber interfaces identificadas aún no especificadas, así como requisitos de calidad aún en un alto nivel.

Aproximación Temprana y Rápida COSMIC *Early Quick COSMIC approximation*(13)

Descripción: La técnica se basa en la capacidad del medidor para clasificar una parte de los requisitos reales como perteneciente a una categoría funcional particular. Es una adaptación de la técnica Early Quick Function Points, combinando técnicas de escalado y clasificación. Permite el uso de diferentes niveles de documentación para diferentes ramas del sistema en diferentes niveles de descomposición. Dentro de las fortalezas de la técnica, que es aplicable cuando una parte significativa de los requisitos reales aún no se conoce a un nivel de detalle que permite identificar los procesos funcionales, además de poder manejar diferentes niveles de documentación y descomposición dentro de los requisitos reales; también posee ciertas debilidades como que la asignación de procesos funcionales a una clase de tamaño es un elemento subjetivo, no está diseñado para aproximar mejoras y la incertidumbre del método sólo depende de la capacidad del estimador para identificar la entrada correcta en la tabla para la declaración de requisitos que no está influenciado por la calibración.

Consiste en que cada parte de los requisitos reales debe clasificarse, en orden de magnitud creciente y número de elementos de composición en uno de los cuatro niveles: un **Proceso Funcional (FP)** es el proceso más pequeño, puede ser pequeño, mediano, grande o extragrande, dependiendo de su número estimado de movimientos de datos. Su categorización es similar a la técnica Clasificación de Tamaño Fijo. Un **Proceso Típico (TP)** conjunto de las cuatro operaciones básicas del usuario: crear, recuperar, actualizar y eliminar (CRUD) en datos que describen un objeto de interés determinado. Un **Proceso General (GP)** es un conjunto de Procesos Funcionales medios, puede ser considerado como un subsistema operativo de la aplicación. Un GP puede ser Pequeño, Medio o Grande, basado en el número estimado de Procesos Funcionales que contiene. Finalmente también tenemos al **Macroproceso (MP)**, un conjunto de Procesos Generales medios y puede ser considerado como un subsistema relevante del Sistema de Información general de la organización del usuario. Puede ser Pequeño, Medio o Grande, según el número estimado de Procesos Generales que contiene. La Figura 2.11 muestra una tabla de referencia adecuada permite al medidor asignar un valor medio CFP para ese elemento. La aproximación de tamaño total (que es una estimación de 3 puntos de un tamaño mínimo, más probable y máximo) es la suma de las aproximaciones de tamaño de los componentes individuales.

TIPO	NIVEL	RANGOS/ EQUIVALENTE COSMIC	MIN CFP	LO MAS PROBABLE	MAX CFP
Proceso Funcional	Pequeño	1-5 movimientos de datos	2.0	3.9	5.0
	Medio	5-8 movimientos de datos	5.0	6.9	8.0
	Grande	8-14 movimientos de datos	8.0	10.5	14.0
	Muy Grande	1-5 movimientos de datos	14.0	23.7	30.0
Proceso Típico	Pequeño	CRUD (procesos pequeños/ medianos) CRUD + lista (procesos pequeños)	15.6	20.4	27.6
	Medio	CRUD (procesos medianos/ grandes) CRUD + lista (procesos medianos) CRUD + lista + informe (procesos pequeños)	27.6	32.3	42.0
	Grande	CRUD (procesos grandes) CRUD + lista (procesos meidanos/ grandes) CRUD + lista + informe (procesos medianos)	42.0	48.5	63.0
Proceso General	Pequeño	6 - 10 FP genéricos	20.0	60.0	110.0
	Medio	10 - 15 FP genéricos	40.0	95.0	160.0
	Grande	15 - 20 FP genéricos	60.0	130.0	220.0
Proceso Macro	Pequeño	2 - 4 GP genéricos	120.0	285.0	520.0
	Medio	4 - 6 GP genéricos	240.0	475.0	780.0
	Grande	6 - 10 GP genéricos	360.0	760.0	1300.0

Figura 2.11: Tabla de comparación Quick Early. Adaptada de Early Software Sizing with COSMIC Guide

Aproximación de Puntos de fusión EASY (EARLY AND SPEEDY)(13)

Descripción: Basada en una técnica rápida de aproximación de medición de software (SMART) para dimensionar los requisitos reales difusos actuales. Esta aproximación es válida a lo largo de la evolución de los requisitos reales, ya que su descripción evoluciona en el tiempo. Las fortalezas de esta técnica es que puede ser mezclada con medidas estándar, se puede escalar a diferentes niveles de documentación, además de que funciona para proyectos de mejora; sin embargo, presenta ciertas debilidades como que puede llevar mucho tiempo establecer la calibración y aplicarla a los requisitos reales.

Consiste en que en cualquier función, el medidor es libre de asumir una o más 'posibilidades' de valor basadas en la comprensión de los requisitos reales que describen la funcionalidad. La técnica de aproximación EASY proporciona las distribuciones de probabilidad más típicas para que el medidor elija, siempre y cuando tenga datos para generarlas, y permite mezclar tamaños aproximados y tamaños precisos. (Los tamaños precisos, es decir, los tamaños medidos según el método de medición estándar corresponden a tamaños en los que se asigna un valor con una probabilidad cercana al 100 %). A continuación en la Figura 2.12 se muestra una distribución de probabilidades aproximadas que son aceptadas por el estándar COSMIC.

CLASIFICACION DEL FP	NIVEL DE ESPECIFICACIÓN	CFP (MIN)	CFP	CFP (MAX)	CFP APROXIMADO	PROBABILIDAD
FP pequeño	POCO DESCONOCIDO	2 (10%)	3 (75%)	5 (15%)	3.2	> 80%
FP pequeño	DESCONOCIDO (SIN FUR)	2 (15%)	4 (50%)	8 (35%)	5.1	< 50%
FP mediano	POCO DESCONOCIDO	5 (10%)	7 (75%)	10 (15%)	7.25	> 80%
FP mediano	DESCONOCIDO (SIN FUR)	5 (15%)	8 (50%)	12 (35%)	8.95	< 50%
FP grande	POCO DESCONOCIDO	8 (10%)	10 (75%)	12 (15%)	10.1	> 80%
FP mediano	DESCONOCIDO (SIN FUR)	5 (15%)	10 (50%)	15 (35%)	11.45	< 50%
FP complejo	POCO DESCONOCIDO	10 (10%)	15 (75%)	20 (15%)	15.25	> 80%
FP complejo	DESCONOCIDO (SIN FUR)	10 (15%)	18 (50%)	30 (35%)	21	< 50%

Figura 2.12: Distribuciones probabilidades de valores aproximados en el dominio empresarial. Adaptada de Early Software Sizing with COSMIC Guide

Modelo EPCU o Aproximación Mediante Lógica Difusa(13)

Descripción: El modelo toma en cuenta las variables lingüísticas basadas en la experiencia utilizadas por los expertos en estimación en el ámbito de la estimación (aproximación en este caso) y como los expertos combinan estas variables lingüísticas para aproximar el tamaño funcional. Esta técnica se puede aplicar al no conocer en gran nivel una parte significativa de los requisitos. lo que dificulta identificar los procesos funcionales o al tener diferentes niveles de documentación y descomposición dentro de los requisitos reales, además de no necesitar datos históricos

Consiste en 6 pasos: los primeros 3 (identificación de variables de entrada, identificación de variables de salida y generación de reglas de inferencia) sirven para el proceso de estimación el cual se da en el punto 4 Fuzzificación. Finalmente el paso 5 y 6 se obtiene variables de salida y su decodificación en variables lingüísticas.

2.5.3.3. Técnicas para la etapa de requisitos

Tamaño Medio de Procesos Funcionales *Average Size of Functional Processes*(13)

Descripción: Considerado como el proceso más sencillo para obtener un tamaño aproximado de una pieza de software. Utilizada cuando los requisitos reales de una pieza de software son conocidos sólo para el nivel de los procesos funcionales y no en el nivel de los movimientos de datos. Algunas desventajas notorias son el tamaño medio del proceso funcional depende del dominio que se elija y requiere el muestreo y el cálculo de un proceso funcional medio basado en mediciones detalladas.

Consta de 2 partes: La primera es la **determinación del factor de escala**, donde se encuentra una muestra de requisitos con procesos funcionales y movimientos de datos bien

definidos, esta muestra permite medir los tamaños de procesos funcionales utilizando el método estándar de COSMIC, el tamaño medio de todos los procesos funcionales medidos es el factor de escala. La segunda parte es la **aproximación mediante el factor de escala** donde si tomamos un conjunto de requisitos cualquiera, su tamaño funcional se aproxima a ser el número de procesos funcionales multiplicados por el factor de escala.

Clasificación de Tamaño Fijo *Fixed Size classification*(13)

Descripción: Esta técnica es considerada como fácil de usar, es válida si hay una razón suficiente para suponer que la clasificación de tamaño asignada es representativa para el software a medir. En caso de existir reglas locales objetivas deben ser declaradas para ayudar a los medidores a asignar la clasificación correcta. Una desventaja que posee es que la definición de la clasificación de tamaño depende del dominio y la asignación de procesos funcionales a una clase es subjetiva.

Consiste en definir una clasificación de tamaño de los procesos funcionales en la pieza de software a medir. A cada clase se le asigna un tamaño correspondiente, o factor de escala, para todos sus procesos funcionales (ver técnica tamaño medio de procesos funcionales). Se analiza una declaración de los requisitos reales para identificar los procesos funcionales y asignarlos según su tamaño en una de tres o más clases.

Bandas de Tamaño Similar *Equal Size Bands Approximation*(13)

Descripción: Esta técnica se recomienda para el software que tiene una distribución significativamente sesgada del tamaño de los procesos funcionales. Es válida siempre y cuando haya razón suficiente para suponer que la clasificación de tamaño asignada es representativa para el software del que se debe medir el tamaño funcional aproximado, un mayor sesgo, implica mayor será la ventaja de esta técnica para la precisión sobre las técnicas de Tamaño Medio de Procesos Funcionales y Clasificación de Tamaño Fijo. Es una técnica fácil de usar, aplicable al software tanto en la aplicación empresarial como en los dominios incrustados en tiempo real y ayuda a generar una aproximación más precisa.

Consiste en clasificar los procesos funcionales en un número determinado de "bandas de tamaño". Los límites de las bandas se eligen en el proceso de calibración que consiste en adecuar los datos de la tabla junto con los históricos que se tengan en el momento de la calibración. Al final, el número total de los procesos funcionales en cada banda debe ser el mismo para cada banda.

Promedio de casos de uso *Average Use Case*(13)

Descripción: Esta técnica funciona mejor con una entrada de datos simétrica y una desviación estándar significativamente más pequeña que el tamaño medio de caso de uso. De fácil uso si hay un estándar local en lo que es un caso de uso. El problema con esta técnica es que el concepto de Caso de uso es subjetivo, de modo que la cantidad de funcionalidad que está asociada a un caso de uso puede variar ampliamente, la técnica no funcionaría a menos que la organización con los casos de uso adopte algún tipo de estándar para lograr una consistencia en el tamaño funcional. Requiere suficientes datos históricos para la calibración del tamaño de un caso de uso medio, verificando la homogeneidad de los datos históricos también.

El principio de la aproximación es similar a la aproximación media del proceso funcional de la técnica de Tamaño Medio de Proceso Funcionales, pero en un nivel superior de documentación, como conocer el caso de uso.

Patrones de Medición de Tamaño Funciona *Functional Size Measurement (FSM) patterns*(13)

Descripción: Esta técnica es adecuada para el software de tipo empresarial y en tiempo real dentro de dominios bien definidos. Tiene algunas fortalezas como: Contribuir a reducir el esfuerzo de medición, ejecución por usuarios relativamente inexpertos del método COSMIC, proporciona una medición de tamaño más precisa al ayudar a evitar errores de medición comunes, además, los patrones de la estimación temprana del tamaño permiten ser repetibles; sin embargo, también poseen desventajas como su necesitad de ser descritos adecuadamente con el fin de ser utilizados por los medidores inexpertos, su uso aún no han sido evaluados cuantitativamente con los objetivos de solución para COSMIC.

Consiste en identificar a los requisitos funcionales en uno de las siguiente patrones FSM :**Patrón Micrco FSM:** Fragmento de un proceso funcional que implica uno o varios movimientos de datos. **Patrón FSM Básico:** Proceso funcional COSMIC único y completo. **Patrón FSM Compuesto:** Conjunto de patrones FSM básicos que tienen un significado funcional de alto nivel juntos. Combina varios procesos funcionales, por ejemplo el conjunto CRUD (Crear, Leer, Actualizar, Eliminar). **Patrón FSM Multicompuesto:** Conjunto de patrones compuestos y básicos que tienen relaciones funcionales entre ellos. Combina múltiples procesos funcionales que manejan datos que describen varios objetos de interés dentro del software que se está midiendo. Se selecciona si se trabajara con los valores mínimos, máximos o más probables. La elección de estos rangos depende de la calidad de los requerimientos.

2.5.4. Proceso de estimación con COSMIC

En la sección 2.3 Fundamentos de la estimación se dieron las pautas para entender cómo es el proceso de estimación y se mencionaron diversos estándares de medición de tamaño funcional, entre ellos el estándar 19761 o método COSMIC.

A partir de un conjunto de aplicativos medidos y registrados junto con sus esfuerzos, es posible generar estimaciones. Para generar esta estimación se utilizan los pasos del 1 al 4 del Proceso de estimación visto en la sección anterior. Hay 3 aspectos importantes para la estimación con COSMIC:

1. Entradas: Estás entradas deben tener cierto grado de confiabilidad. Para tener referencias de proyectos pasados es posible tomar datos de algunas bases de datos como ISBSG (International Software Bachmarking Standar Group) o de la Asocición Mexicana de Métricas de SOftware (AMMS), la cual contiene diversas variables de proyectos, por lo que se debe seleccionar las variables(dependientes e independientes) que serán estimadas. Las que se usarán en el proyecto son esfuerzo y tamaño o que contengan características similares al proyecto que se busca estimar. Se recuerda que la selección de datos es importante por que es lo que genera el modelo de productividad. El modelo de productividad puede ser un modelo de regresión lineal, un modelo de inferencia estadística y predicción, una correlación exponencial, étc(57)(58).

2. Análisis del modelo de productividad: Una vez obtenido el modelo de productividad, se analiza el modelo con algún criterio de desempeño, en este caso el manual de COSMIC indica utilizar los criterios RE, MRE y MMRE. Esto indicará que tan confiable es nuestra

estimación. Para que un modelo de productividad funcione debe cumplir con ciertos criterios como lo son(59):

a) Análisis Residual: los residuos son estimaciones de la variable de error. Si el modelo dado es incorrecto la gráfica resultante puede mostrar residuos, otras cantidades o desviaciones, lo anterior permite identificar fallas como la no linealidad, la varianza no constante, falta de normalidad o valores atípicos.

b) Valores atípicos y observaciones influyentes: Un valor atípico es un residuo mucho más grande que los demás, estos valores pueden tener distintos orígenes, de ser encontrada una explicación, estos datos pueden analizarse de separada.

c) Multicolinealidad: cuando existen fuertes dependencias lineales entre las variables explicativas, lo que conduce a altas correlaciones entre estas variables, de tal manera que la precisión de algunas estimaciones de las variables correlacionadas, los predictores pueden ser degradados.

3. Resultados de la estimación: COSMIC tienen su propio sistema de medida CFP (COSMIC Function Points). Con base en las mediciones de COSMIC se puede generar una medición de esfuerzo o costo, usualmente es utilizada una fórmula como la siguiente:

$$esfuerzo = (a * tamañofuncional) + b$$

en donde a representa la variable del eje x y b la variable del eje y, recordando que x o y pueden ser el tiempo o tamaño.

Capítulo 3

Estimación de esfuerzo en metodologías ágiles en los últimos años

Este capítulo presenta los trabajos relacionados con el tema de esta tesis. Se analizan los artículos:

1. Estimación de Esfuerzo en Desarrollo de Software Ágil: Revisión Sistemática de la Literatura (Effort Estimation in Agile Software Development: A Systematic Literature Review(56)) el cual dio un primer estudio que presenta una descripción general del estado del arte en la estimación del esfuerzo para ASD hasta el año 2013,
2. Actualización Estimación de Esfuerzo en Desarrollo de Software Ágil: Revisión Sistemática de la Literatura (An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review(19)), el cual busca ampliar el conocimiento del artículo anterior hasta el año 2020 que fue escrito.
3. Un artículo de revisión sobre la estimación del esfuerzo de software en metodología ágil (A Review Article on Software Effort Estimation in Agile Methodology(54)).

Finalmente se expresará el panorama encontrado al analizar los tres artículos.

3.1. Estimación de Esfuerzo en Desarrollo de Software Ágil: Revisión Sistemática de la Literatura

Las Revisiones Sistemáticas de Lectura son creadas por diferentes motivos, en ellas se pretende dar una base teórica para investigaciones posteriores del tema escogido así como conocer el estado de la investigación que se tiene sobre el tema, considerando las posibles limitantes(48). En 2014 se publica Effort Estimation in Agile Software Development: A Systematic Literature

Review(56), considerado como la primera revisión sistemática de la literatura publicada hasta el momento de desarrollo de software con metodologías ágiles, revisando y sintetizando sistemáticamente los modelos y prácticas de estimación de esfuerzo para Agile Software Development (ASD) entre el periodo de 2001 (año en que el Manifiesto Ágil es publicado) hasta el 2013. Fueron encontradas brechas que dieron pie a nuevas oportunidades de estudio en el desarrollo de Software Ágil.

3.1.1. Método empleado

Las metodologías ágiles ejecutan una planificación en tres niveles: inicio, planeación y/o estimación y la implementación diaria. Teniendo esto en cuenta, en el estudio se plantearon las siguientes preguntas de investigación:

1. ¿Qué técnicas se han utilizado para el esfuerzo o estimación de tamaño en el desarrollo de software ágil? | ¿Qué métricas se han utilizado para medir la precisión de estimación en estudios de estimación de esfuerzo para desarrollo Ágil de Software? | ¿Qué nivel de precisión se ha logrado con estas técnicas?
2. ¿Qué predictores de esfuerzo (métricas de tamaño, costo, controladores) se han utilizado en estudios sobre la estimación del esfuerzo para el desarrollo Ágil de Software?
3. ¿Cuáles son las características del conjunto de datos/conocimiento utilizado en estudios sobre estimación de esfuerzo para el desarrollo Ágil de Software?
4. ¿Qué métodos ágiles se han investigado en los estudios sobre la estimación del esfuerzo? | ¿Qué actividades de desarrollo (por ejemplo, codificación, diseño) han sido investigados? | ¿Qué niveles de planificación (inicio, implementación, implementación por día) han sido investigados?

Estas preguntas estaban encaminadas a medir el valor de los sprints que son la parte fundamental del desarrollo en ágil, además de ser la estimación del esfuerzo en desarrollo de software ágil un área de investigación activa y los resultados encontrados serán descritos en la siguiente sección.

Para la selección de artículos publicados, se seleccionaron palabras clave del Desarrollo en Ágil para posteriormente buscar en motores de búsqueda como Scopus, IEEE explore y Science Direct, encontrando 443 resultados y omitiendo duplicados, después de ser filtrados de acuerdo a su resumen, sólo 32 artículos se utilizaron para la revisión sistemática.

Finalmente sobre a cada uno de los 32 artículos se le aplicó el cuestionario sobre calidad de medición que consistía en preguntas sobre la calidad de los datos que utilizaba, calidad en las técnicas, calidad en los resultados obtenidos. A los artículos que obtuvieron un puntaje menor a 3.25, se eliminaron del estudio.

3.1.2. Resultados

En total se revisaron 20 artículos. Los resultados están ordenados de acuerdo al número de la pregunta de investigación del artículo mencionado, que responde.

1. Las técnicas de estimación por tamaño (4 de 20) y por esfuerzo fueron las más utilizadas (13 de 20), mientras que 3 artículos de 20 no reportaron información al respecto. De los artículos anteriores, Planning Poker (7 de 20) junto con Use Case Points fueron las técnicas más reportadas (10 de 20), estas fueron acompañadas con Juicio de Experto, el Juicio de Experto también ha sido utilizando de manera individual (2 de 20). COSMIC FP fue mencionado en 1 artículo, en donde se utilizó una forma de estimar con COSMIC haciendo énfasis en cómo utilizar textos con requisitos escritos de manera informal.

   El nivel de precisión fue estudiado mayormente mediante la *Magnitud de la Media del Error Relativo MMRE* (4 de 20) y la *Magnitud del error relativo MRE* (3 de 20), este último siendo utilizado para las estimaciones de esfuerzo. Por otro lado, 6 artículos de 20 reportaron no utilizar métricas.

   En cuanto a la precisión de estás técnicas, el artículo menciona que hay un vacío respecto a que no todos los artículos presentaban una métrica. Para el Juicio de Experto, presentó un MRE de entre 8 % a 32 %, MMRE de entre 28 % a 38 %

2. Para el predictor de esfuerzo por métricas de tamaño, las técnicas preferidas fueron Story Points (8 de 20 artículos) y Use Case Points (6 de 20 artículos), aunque esta técnicas no son estándares, sus datos no son comparables. COSMIC FP fue utilizado en 1 artículo de los 20 disponibles. Cinco artículos reportaron no usar alguna técnica.

   Respecto al predictor de esfuerzo por costo se encontró que parte del conocimiento no es explícito, las habilidades y la experiencia de los desarrolladores son cruciales en todas las actividades a realizar, esto incluye a la estimación por esfuerzo; sin embargo otros reportaron que podría ser el tamaño del proyecto(2 de 20 artículos), resultados de pruebas de factor de riesgo(4 de 20 artículos), pruebas de factor de eficiencia(4 de 20 artículos), experiencia previa del equipo(3 de 20 artículos), étc (5 de 20 artículos), estos factores podían estar combinados. Finalmente sólo 2 artículos de 20 no describieron nada al respecto.

3. Los artículos fueron divididos de acuerdo a sus características (industriales o académicos), se encontró que 16 artículos basaban sus datos completamente en datos de la industria, lo cual concedía fiabilidad al resultado obtenido, 2 artículos no describieron de donde obtuvieron los datos y 2 artículos utilizaron tanto datos de la academia como de la industria. De los artículos anteriores que utilizaban datos de empresas, sólo 13 utilizaron datos de la empresa donde se desarrollaban, 3 artículos utilizaron datos de distintas empresas, .

4. Los métodos ágiles más utilizados fueron Scrum y XP, cuando se estudió la estimación de esfuerzo en 15 de 20 artículos, los artículos restantes no lo especificaron. Ningún estudio estimó el esfuerzo en actividades de análisis y diseño, sólo en la implementación o en la etapa de pruebas.

   Respecto a la planeación en el contexto ágil se puede realizar en tres niveles: inicio, planificación y estimación e implementación (diaria). Respecto a la implementación, cuando es diaria, ningún artículo reportó hacerlo, para la planificación y estimación de los sprints, 6 de 20 artículos lo reportaron y para la inicio lo reportaron 5 artículos; sin embargo 12 artículos no describieron nada respecto a su planeación.

3.1.3. Conclusiones del artículo

A pesar de que el estudio comprende 13 años, los artículos que fueron utilizados fueron pocos, lo cual implica que no fue un tema de interés para la comunidad científica.

Se utilizaron pocos predictores de esfuerzo, lo más usados son principalmente respecto al tamaño, aunque no utiliza estándares de tamaño sino métricas no formales y de éstos no existe un consenso.

Cada artículo de los 16 que usaban datos de la industria, manejó datos sólo de los proyectos de una industria, por lo que sólo se hizo un análisis de cada industria, no de todo el panorama. Por otro lado, esto también se debió a la poca apertura de las industrias por compartir sus datos al público.

Para el año 2013 ya se habían dado a conocer diversas metodologías ágiles como Kanban(2007), Lean Starup(2009); sin embargo, los artículos analizados reportaron utilizar XP o SCRUM y los artículos que no lo hicieron, no mencionaron la metodología usada.

Un tema que no fue abordado fue el uso de métricas, los pocos artículos que lo mencionaron, reportaban al MMRE y MRE como las más utilizadas.

En los resultados mostrados no se vio que la mayoría de los técnicas alcanzaran una precisión de predicción aceptable, por lo que los valores que se reportaron como estimados distaban considerablemente al contrastarse con los obtenidos al final del proyecto.

En el único artículo que hizo mención a COSMIC, se enfoca en solucionar un problema especifico que es la falta de la expresión formal en los requisitos que es algo común cuando se usa metodologías ágiles y no da una pauta para integrar COSMIC.

3.2. Actualización Estimación de Esfuerzo en Desarrollo de Software Ágil: Revisión Sistemática de la Literatura

Después de la publicación del artículo Effort Estimation in Agile Software Development: A Systematic Literature Review(56), se dejaron pasar años para realizar una nueva revisión sistemática. En 2020 se publica una actualización del artículo anterior llamada: An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review(19) para evaluar el panorama actual que proporcionaba nuevas perspectivas de la Ingeniería de Software con metodología ágil, además de la falta de una revisión sistemática formal.

3.2.1. Método Empleado

El método utilizado fue el mismo que el propuesto en el artículo anterior. Las preguntas de investigación fueron las mismas que en el artículo anterior.

La búsqueda de las bases de datos arrojó 1222 artículos, utilizando un proceso de selección similar al del primer artículo de dos fases (con algunos ajustes en su criterio de búsqueda), se redujo a 75 artículos. Utilizando el mismo cuestionario sobre velocidad de medición elevando el puntaje para descartar los artículos con calificación menor a 6.5, quedando 73 artículos.

El análisis de los artículos fue dividido en 2 partes: análisis biométrico y respuestas a las preguntas de investigación. El análisis biométrico buscó encontrar dónde fueron publicados los artículos originalmente (congresos, revistas, étc.), así como los países que más aportes tuvieron. Para las respuestas a las preguntas de investigación, los resultados aparecen en la siguiente sección.

3.2.2. Resultados

Los resultados se muestran de acuerdo al número de la pregunta de investigación que responde, recordando que las preguntas de investigación que se manejaron en este artículo fueron las mismas que se utilizaron en el artículo Estimación de Esfuerzo en Desarrollo de Software Ágil: Revisión Sistemá- tica de la Literatura (56).

1. Sólo 3 de 75 artículos no mencionaron el método para medir el esfuerzo o tamaño, siendo los métodos que usan el juicio de experto (8 de 75 artículos) los más populares como Planning Poker (16 de 75 artículos) en algunos combinado con otras técnicas como Delphi, Expert Judgement (7 de 75 artículos) y Wideband Delphi con (4 de 75 artículos), seguidos de métodos basados en datos utilizando Maching Learning (13 de 75 artículos), Neural Network (11 de 75 artículos) y Function Size Measurement con 12 artículos, de estos últimos específicamente 7 artículos utilizaron COSMIC, algunos lo utilizaron enfocándose en encontrar equivalencias entre COSMIC, Use Case Points y Function Points en la aplicación de metodologías ágiles(5), establecer los desafíos relacionados a la medición del tamaño funcional en conjunto con metodologías ágiles (28), entender el problema que puede causar la semantica en la granularidad de los requerimientos funcionales a través de casos de estudio(47) o lo utilizaron en la estimación(27) por mencionar algunos de ellos. Finalmente, en métodos combinados 3 de 75 artículos utilizaron Use Case Points, 2 de 75 artículos utilizan modelos de ontologías siendo estos últimos los más populares en esta categoría.

   La métrica de precisión más usada sigue siendo el MRE (32 de 75 artículos), seguidos por el Nivel de Predicción PRED(x) (15 de 75 artículos), el coeficiente de determinación (R^2) (6 de 75 artículos),de estos 6 últimos, en 2 artículos se combinó con otras métricas como MRE o MAE. 22 de 75 artículos no especificaron, o explícitamente mencionaron que no utilizaron ningún método.

   Para la precisión de las técnicas se utilizó el MMRE, MdMRE, PRE(x). Un porcentaje de PRED(25 %) $\geq$ 75 % considera a un modelo como aceptable, lo cual cumplieron 67

de los 75 artículos. Respecto al MMRE, Samuel Conte(12) considera un nivel aceptable a un MMRE ≤ 0.25, de ellos sólo 4 de 75 artículos presentaban niveles altos.

2. Se tomó en cuenta los dos principales predictores de esfuerzo: métricas de tamaño y factores de costo. Para las métricas de tamaño, la métrica más popular es Story Points 45 de 75, de estos 45 algunos la usaron en combinación con otras métricas como COSMIC (4 de los 45 artículos), Quick FP (2 de los 45 artículos) por mencionar algunos. Los Function Points ocupan el segundo lugar (6 de 75 artículos). La mayoría de los estudios que utilizan Planning Poker o Juicio de Experto para la estimación del esfuerzo, utilizó Story Points como la métrica de tamaño (24 de 75).

   Para los factores de costo, se determinó que existen 4 factores importantes para el costo, *los factores de proyecto* como la complejidad del proyecto(5 de 75 artículos), novedad del proyecto (5 de 75 artículos) o la calidad(5 de 75 artículos), *factores de equipo*, siendo la más citada la experiencia de equipo (19 de 75 artículos), seguidos de las habilidades de los desarrolladores (9 de 75 artículos), velocidad del equipo (7 de 75 artículos), *factores técnicos*, como herramientas del software y de desarrollo (6 de 75 artículos) y el impacto en sistemas existentes (5 de 75 artículos), *factores de las historias de usuario* como prioridad de la historia actual (10 de 75 artículos), sprint de la historia actual (4 de 75 artículos).

3. De los datos obtenidos se buscaba su validación con uno o múltiples casos (reales o simulados) para confiabilidad. 51 de 75 artículos ocupan datos de la industria, frente a 6 de 75 artículos que ocupan solamente datos de la academia. Sólo 1 de 75 artículos utilizó datos combinados de la industria y la academia, contra 3 de 75 artículos que utilizan sólo datos simulados. Finalmente, sólo 14 de 75 artículos no reportaron la procedencia de los datos que ocupaban.

   De los datos de la industria, 31 de 51 artículos tomaron artículos sin apegarse a una compañía en especifico (entre ellos los académicos), 20 de 51 artículos utilizaron datos cruzados de diversas compañías.

4. SCRUM sigue siendo la metodología ágil más usada, 49 de 75 artículos reportaron utilizarla, seguida por XP(Extreme Programing) mencionado en 6 de 75 artículos, además en 20 de 75 artículos no fue mencionado claramente la metodología especificada.

   En 22 de 75 artículos se investigó cada una de las etapas del desarrollo, 6 de 75 artículos sólo investigaron la etapa de desarrollo, 1 de 75 artículos solo investigó la etapa de pruebas, 3 de 75 artículos investigaron las etapas de desarrollo y pruebas y los restantes 43 de 7 artículos, no reportaron investigaciones sobre el tema.

   Considerando la planificación en un contexto ágil en tres niveles: inicio, planificación y estimación e implementación (diaria), 33 de 75 artículos reportaron hacer estimaciones de esfuerzo cada implementación, 17 de 75 artículos reportaron hacerlo cada inicio y 7 de 75 artículos lo hicieron cada inicio/implementación, junto a 18 de 75 artículos restantes que no lo especificaron.

3.2.3. Conclusiones del artículo

El número de artículos encontrados (75) es considerablemente mayor que el artículo anterior (20) con 55 artículos más, lo cual muestra un aumento en el interés de los investigadores en años posteriores al primer artículo.

Al aumentar el puntaje el test de calidad en el criterio de selección para los artículos, podemos notar un mejor refinamiento de los artículos seleccionados, garantizando resultados más confiables.

Se encontró una mayor apertura de las empresas a compartir sus datos, lo cuál ayuda a la tendencia a técnicas basadas en datos de los cuales Function Size Measurment forma parte, aunque principalmente se enfocan en el uso de métricas no estandarizadas como story points y use case points o en método de tamaño funcional de primera generación que se ha demostrado que presentan algunos problemas de fundamento(40).

El Desarrollo de Software en Ágil sigue estando basado en la opinión de expertos, siendo Planning Poker el método de estimación más usado para determinar tamaño o esfuerzo; sin embargo, se mostró un incremento en métodos con Inteligencia Artificial. Function Size Measurment aparece en 12 de 75 artículos como una método para medir tamaño (funcional). En particular COSMIC apareció en 7 de 75 artículos a comparación del artículo original donde sólo fue utilizado en 1 de 75 artículos.

Algunos artículos reportaron una mejora significativa si en lugar del juicio de experto utilizaban Planning Poker o Wideband Delphi.

Commeyne(11) mostró que COSMIC proporciona evidencia objetiva de mejores estimaciones frente a Plannign Poker e Story Points.

Tanto en el artículo original como en este artículo los factores más valuados respecto al equipo son la experiencia que ellos tenga individualmente, tanto colectivamente como equipo.

Al tener la apertura de compartir datos entre empresas se puede tener un mejor panorama y tener datos más realistas respecto al mundo real.

Usar factores de equipo y proyecto es más frecuente que la consideración de factores técnicos, A pesar de una mejora significativa en los reportes sobre la medición de tamaño y costos, los resultados no se informan correctamente por lo que las comparaciones no son fiables, los factores no están agrupados por categorías ni estandarizados por lo que está rama aún representa una gran brecha y un posible campo de investigación.

Las estimaciones están realizadas principalmente en la planificación de cada Sprint en un 45 % de las veces o por proyecto/versión 23 %, en ningún caso es realizada diariamente.

3.3. Un artículo de revisión sobre la estimación del esfuerzo de software en metodología ágil

Además del artículo An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review(19), otros artículos en el mismo año 2020 han sido publicados, tratando de brindar un panorama de las metodologías ágiles en 2020 y permitir incursionar para solucionar problemas que se van presentando en la industria o estudiar las áreas de oportunidad que necesitan más estudio, algunos hechos encontrados por el artículo A Review Article on Software Effort Estimation in Agile Methodology(54) que sirvieron como motivación para desarrollar el texto son: La tasa de éxito de Agile tiene dos veces más probabilidades de éxito, y un tercio menos probabilidades de fracasar que los proyectos en cascada, los costos se basan en la estimación del esfuerzo. En resumen, este artículo tiene como principal enfoque la estimación del esfuerzo revisar los métodos y enfoques de la estimación del esfuerzo del software para determinar el método apropiado para el desarrollo ágil y proporcionar una base para un mayor desarrollo de la estimación del esfuerzo, mediante una revisión sistemática, los artículos anteriores utilizaron la revisión sistemática para abordar el panorama en el que se encontraba el desarrollo de software en Ágil en su momento.

3.3.1. Método Empleado

Después de una investigación del panorama e identificación de las variables importantes (proceso que no es descrito ampliamente) se formulan las tres siguientes preguntas:

1. ¿Qué tipo de método se utiliza para la estimación del esfuerzo en Ágil?
2. ¿Cómo funciona implementar el esfuerzo de estimación en Ágil?
3. ¿Cuáles son los atributos involucrados en la estimación del esfuerzo en Ágil?

El proceso se dividió en 3 fases: estrategia de búsqueda, selección de estudios y síntesis de datos. En la estrategia de búsqueda se utilizaron 7 motores de búsqueda (IEEExplore, ACM, Google Scholar por mencionar algunos). Para la selección de estudios, se utilizaron criterios de exclusión y una evaluación de calidad, donde los artículos debían obtener calificación de al menos 1. Finalmente la síntesis de datos se da para responder las preguntas de investigación planteadas anteriormente. Finalmente 38 artículos fueron seleccionados.

3.3.2. Resultados

Los resultados están ordenados de acuerdo al número de la pregunta de investigación que responde.

1. Los métodos de estimación de esfuerzo implementados en ágil son clasificados por el artículo en en cuatro categorías, Juicio de expertos (EJ), Algorítmico (A), Aprendizaje automático (ML) y Estadística (St), de los cuales 6 de 38 artículos no reportaron utilizar ninguno, 14 de 38 utilizan Machine Learning, 10 de 38 juicio de experto siendo Planning Poker el más popular y 8 de 38 algún método algorítmico.

En el juicio de experto factores como la experiencia, tiempo, esfuerzo, prioridad y el valor de las historias de usuario, son las que diferencian a un llamado experto de un principiante; sin embargo suelen ser los principiantes los más interesados en el proyecto, lo cual genera problemas en los equipos. Se ha buscado crear alternativas compuestas combinando otros métodos como redes bayesianas, Naive Bayes, Logistic Model Tree, entre otros; sin embargo, estos nuevos modelos no han sido probados en la industria.

2. El artículo clasifico a COSMIC como un método algorítmico junto a Phase-Wise, Function Point (FP) y Use Case Point (UCP), Phase-wise. Se utilizó para calcular el esfuerzo de estimación, Use Case Points método de estimación para el desarrollo ágil en la etapa inicial al enfatizar los principales factores de complejidad como técnica y medioambiental.

   Los métodos de Aprendizaje Automático (Inteligencia Artificial) se han utilizado para resolver el tamaño de las historias de usuario, La precisión en la estimación de software ágil también se mejora utilizando los modelos ontológicos. Aunque los estudios proporcionaron muchas ventajas, los modelos aún necesitan más conjuntos de datos masivos y otras características ya que las historias de usuario no solo están escritas en inglés, a veces son influenciadas por la demografía de los desarrolladores, la criticidad de la historia y otros sistemas.

   Métodos estadísticos donde se realiza la estimación del esfuerzo a partir del ciclo de vida convencional, el principal fue el *Análisis de Componentes Principales (PCA)* donde se determinan los atributos clave de costo de desarrollo.

3. 32 de 38 artículos explicaron los atributos utilizados en la estimación de esfuerzo contra 6 de 38 artículos que no lo hicieron, algunos de estos atributos fueron: historia de usuario, caso de uso con método de dimensionamiento, casos de uso y puntos de función.

   En los artículos de los últimos años, el atributo más utilizado en Ágil es la Complejidad, seguido de Experiencia, Tamaño, Esfuerzo y Tiempo. El atributo de complejidad se interpreta en diferentes aspectos, la mayoría de los artículos estudiados mostraron interés en el tema. El atributo de experiencia en la mayoría de los estudios se usa para medir la experiencia de implementación del desarrollador o programador.

3.3.3. Conclusiones del artículo

La mayoría de los artículos tienen el objetivo de mejorar la precisión a través del aprendizaje automático, esto incluye a los que utilizan el juicio de experto.

La mitad de los artículos relevantes utilizaron el enfoque no híbrido, el 36.84 % implementó el híbrido, mientras que el resto no mencionó el enfoque utilizado.

El artículo considera un enfoque no híbrido como los métodos de estimación de esfuerzo Planning Póker, Phase Wise, COSMIC, Puntos de función, Modelos de Ontología, étc. Mientras

tanto, el enfoque Híbrido contiene combinaciones de métodos entre Juicio de Experto y Estadística, Juicio de Expertos y Aprendizaje Automático, y Aprendizaje Automático. La mayor parte del enfoque híbrido es una combinación de técnicas de aprendizaje automático.

La volatilidad y cambio en los requisito del cliente en el desarrollo de software ágil (ASD) son atributos que afectan la estimación del esfuerzo.

El aprendizaje automático es el método de estimación de esfuerzo más común utilizado en Ágil, seguido por Juicio de Expertos y Algorítmico. Sin embargo, el aprendizaje automático tiene limitaciones en su implementación porque necesita un conjunto de datos muy grande que proviene de conocimiento del experto.

3.4. Estimación de Software en metodologías ágiles Conclusiones

Después de analizar los 3 artículos anteriores podemos concluir que:

1. En el primer artículo se apreciaba una falta de estudio en el tema de la estimación en metodologías ágiles, esto se puede apreciar en los pocos artículos que cumplieron los criterios impuestos; sin embargo, para el segundo artículo hubo un incremento de artículos encontrados, así como artículos seleccionados (considerando que se aumentaron los puntajes para el criterio de calidad en los artículos), lo cual denota el creciente interés en el tema con el paso de los años.
2. SCRUM se ha mantenido como el marco de trabajo colaborativo más popular a lo largo de los años mencionados por el estudio, por lo cual se seleccionó para trabajar la propuesta en esta tesis.
3. A pesar de un notable interés para transicionar del método de Juicio de Experto a métodos cuantitativos sigue siendo el Juicio de Experto el método más utilizado para la estimación o el método de Planning Poker, el cual también implica juicio de experto. También se encontraron artículos que utilizaban mezclas de distintos métodos para no depender sólo del juicio de experto.
4. La investigación sobre predictores de esfuerzo aún no es un tema estandarizado; sin embargo, diversos autores han identificado que la medición de tamaño funcional es una pieza clave en la estimación, propiciando a una mejor exactitud en los modelos de regresión para los modelos de estimación(59).
5. La falta de la planeación en cualquiera de los tres niveles (Inicio, Planificación y Estimación de o implementación diaria) es notoria debido a la casi nula información proporcionada en los artículos. De la poca información reportada se observó que es más probable que exista una planificación en el Inicio frente a las Planificaciones. Respecto a las planificaciones de la implementación, si el ciclo de trabajo por día es corto y las estimaciones toman un periodo de tiempo largo, no tiene sentido generar estimaciones, a menos que estas pudieran generarse automáticamente.

Capítulo 4

Propuesta de Integración entre COSMIC y SCRUM

4.1. Entorno Previo

4.1.1. Entorno previo

La propuesta que se presenta en esta tesis contempla dos aspectos fundamentales: por un lado respetar el ciclo de vida de Scrum junto a sus valores y propuestas. En segundo lugar, la integración de COSMIC mediante un mecanismo de aproximación con la finalidad de que se puedan tener mejores métricas para la gestión de proyectos ágiles.

4.1.1.1. SCRUM

En el capítulo anterior, el artículo A Effort Estimation in Agile Software Development: A Systematic Literature Review(56) resalta que dentro de las metodologías ágiles, existen 3 niveles importantes en donde es posible estimar, *Inicio, Planificación y estimación del Sprint y Planificación de la implementación diaria*. Dado el diagrama del ciclo de vida de SCRUM mostrado en la Sección 2 de la presente tesis, estos niveles se ven representados por:

- **Inicio 4.1:** En esta etapa es donde se da inicio el proyecto, por lo que se tienen los requisitos proporcionados por el usuario en un alto nivel, por lo que se puede utilizar aproximación de COSMIC para determinar el tamaño y luego estimar. Esto crea el Product Backlog, sobre el cual podemos realizar las estimaciones necesarias, creando una primera estimación al inicio del proyecto, pero no olvidando que conforme los Sprints pasen, el Product Backlog puede irse actualizando. Para esta propuesta sólo utilizaremos el primer Product Backlog creado y las siguientes estimaciones realizadas se harán en la Planificación de la Iteración.

- **Planificación y Estimación 4.1:** Esta etapa que da comienzo a cada Sprint se crea el Sprint Backlog con las actividades que deberán cumplirse en el Sprint, permitiendo realizar estimaciones sobre el contenido del Sprint Backlog. Los Sprints suelen durar un máximo de un mes según *La guía Oficial de SCRUM*. Al tener aproximadamente un mes, las estimaciones son posibles, pero deben ser rápidas de lo contrario no serán de utilidad. Se utilizará el Proceso de Aproximación con COSMIC que nos permitirá generar aproximaciones rápidas de acuerdo al nivel de granularidad que tengamos en los requisitos del Sprint Backlog.
- **Planificación de la implementación Diaria:** Está dentro de la fase de Implementación mostrada en la Figura 4.1. Al ser SCRUM una metodología ágil y orientada a cambios rápidos, el Daily Sprint está establecido en un intervalo de 15 minutos para llevarse a cabo de acuerdo a la guía oficial de SCRUM. Para la propuesta de integración no realizaremos estimaciones en esta etapa, debido a que no se considera viable de acuerdo objetivo del Daily scrum, que es identificar el trabajo que se ha hecho, que falta y que problemas existen; es decir, evaluar el desempeño del equipo, no estimar. Además, los datos que puedan generarse de un día a otro no representan cambios sustanciales que dieran paso a nuevas estimaciones.

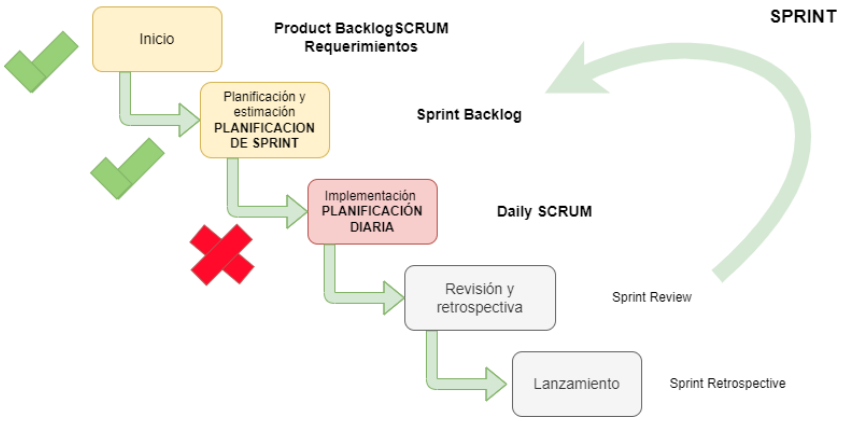

Figura 4.1: Fases del desarrollo en SCRUM con los 3 niveles para estimar

4.2. Propuesta

A manera de lograr un mayor entendimiento de la propuesta, se presenta el Diagrama 4.2, en donde describimos el ciclo de vida de SCRUM y en azul mostramos los momentos donde se utilizará la Aproximación Rápida.

En el diagrama se aprecia las 5 fases de SCRUM Inicio, Planificación y Estimación, Implementación, Revisión y retrospectiva y Lanzamiento en color naranja.

En la parte de arriba de la fase de **Inicio** se muestra que como entrada de la fase están los requisitos de usuario y al final de esta fase se debe obtener:

- El equipo SCRUM que realizará el proyecto
- El Product Goal que indica la meta para conocer el que momento ha sido alcanzado y podemos finalizar el proyecto.
- El Product Backlog, donde estará la lista de requisitos que serán implementados, sobre estos requisitos se utilizará la Aproximación Temprana con una técnica de la Fase de Requisitos descritas en la guía Early Software Sizing with COSMIC.

Una vez que los documentos anteriores sean generados, servirán como entradas de la siguiente fase de SCRUM.

La fase de **Planificación y Estimación** es donde la mayor parte de estimaciones se generan, en esta fase considerada como la fase donde se prepara el Sprint obtenemos el Sprint Backlog, el cual nos indica los requisitos funcionales que deben ser implementados en el Sprint. Sobre este podremos realizar las estimaciones gracias a las técnicas para la fase de viabilidad descritas en la guía Early Software Sizing with COSMIC(13) utilizando las técnicas mostradas en la fase de usabilidad o requerimientos, se escogerá una técnica mediante un algoritmo dado, la elección de cual usar recae en la calidad de los documentos que describen a los procesos funcionales.

En la fase de **Implementación** es donde el Sprint es ejecutado y al término de cada Sprint y antes iniciar uno nuevo, se regresa a la etapa de **Planificación y Estimación**, donde es posible generar una nueva estimación sobre los requisitos faltantes del Product Backlog, actualizar requisitos que vayan a modificarse en el nuevo Sprint o incluso si se generan nuevos requisitos para el producto final que se realizarán en el Sprint puedan ser agregados. Una vez que el Sprint inicie, comienza la fase de Implementación, donde cada día se ejecuta el Daily Sprint para monitorear avances.

Al término de cada Sprint tenemos la fase de **Revisión y Retrospectiva** del Sprint, la cual nos permite conocer el estado del Sprint (Completado o no) permitiendo actualizar el siguiente Sprint Backlog; sin embargo, si es el último Sprint donde el Product Goal es alcanzado, pasamos a la fase de **Lanzamiento**, donde se entrega el producto finalizado.

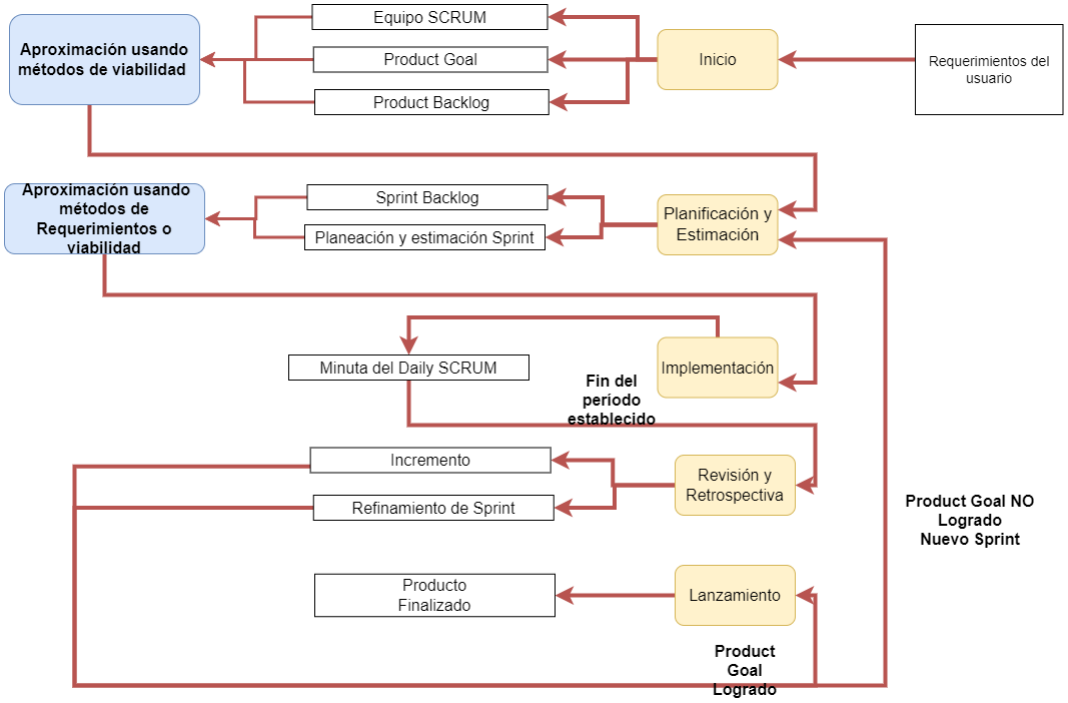

Figura 4.2: Enfoque Propuesto

Una vez explicado los momentos donde se realizará la aproximación rápida, pasamos a mostrar un algoritmo que nos indica como seleccionar la técnica adecuada de Aproximación Rápida de acuerdo a la granularidad de los requerimientos que se tengan al momento.

En la sección 2.5.3 se mostraron técnicas mencionadas por la guía Early Software Sizing with COSMIC(13); sin embargo no todas las técnicas han sido recomendadas para todos los posibles escenarios en que la granularidad de los requerimientos puedan encontrarse e incluso algunas de ellas son utilizadas para el desarrollo de cierto software como software empresarial, donde se tiene un patrón de diseño similar.

Esta propuesta se realizó integrando elementos de SCRUM y COSMIC que han sido validados y actualmente se encuentran en uso, por lo la propuesta está actualizada. Por otro lado se optó por una forma fácil y sencilla donde sólo los conocimientos básicos de cada parte sean necesarios para poder aplicarla y obtener los resultados.

4.3. Ejemplo

Para mostrar detalladamente como se desarrolla la propuesta mencionada, se utilizará la propuesta en un proyecto de desarrollo de software que fue realizado en la clase de Ingeniería de Software de la clase del Posgrado PCIC semestre 2020-1. Se dará una descripción detallada del problema a solucionar, una descripción de como se dio la fase de levantamiento de requisitos, la formación del equipo y el desarrollo. En la siguiente sección una vez que las bases están descritas, se aplicará la propuesta de integración de SCRUM con el método COSMIC siguiendo

la paso a paso hasta obtener el reporte de la aproximación rápida tanto en el fase de inicio, como en la fase de estimación y planeación. Finalmente, con el reporte generado de los resultados obtenidos serán comparados contra los resultados reales que se obtuvieron.

4.3.0.1. Descripción del problema

Se requirió construir una aplicación Web utilizando las siguientes tecnologías:

- Java como lenguaje de desarrollo
- Spring Framework para el apoyo en el desarrollo de la aplicación WEB
- Github utilizado como el controlador de versiones donde todas las entregas serían realizadas.
- Google Drive donde toda la documentación y retroalimentación por parte del cliente (profesora y ayudantes de ella) es notificada.

La aplicación web consistió en una página que mostraba un mapa mundial interactivo que mostraba marcadores, los marcadores contenían información como las coordenadas de localización, un título, una descripción, el nombre del usuario registrado que lo agregó y puede tener comentarios asociados. Estos marcadores son agregados exclusivamente por usuarios que han sido registrados en la página. Por otro lado, los comentarios que los marcadores pueda recibir pueden ser hechos por cualquier persona, además de ser calificados como positivos o negativos. Hay 4 tipos de usuarios, el administrador, informantes, comentaristas y los visitantes.

El administrador es la persona que puede tener acceso a todo el sistema y además es el único que puede dar de alta a los informantes. Los comentaristas sólo pueden calificar una sola vez los marcadores, ya sea como positivo o negativo, aunque pueden poner los comentarios que deseen. Finalmente los visitantes son aquellos que solo pueden navegar por el sitio. La aplicación fue desarrollada bajo una metodología ágil, se adaptó SCRUM a las necesidades de la clase.

4.3.1. Aplicando la propuesta

4.3.1.1. Aproximación en la Fase de Inicio

Antecedentes:
En un inicio se identificaron un total de 22 requisitos funcionales, por lo que el Product Backlog quedó con las siguientes actividades:

1. CRUD cuentas de informantes
2. CRUD comentarios
3. CRUD temas
4. CRUD marcadores
5. Listar temas
6. Buscar temas
7. Ver comentarios
8. Iniciar Sesión
9. Cerrar Sesión
10. Calificar comentarios

Aplicación de la propuesta:
Para la elección del método propuesto se tomó en cuenta la existencia de la lista de procesos funcionales; sin embargo no hay una muestra significativa de los requerimientos por lo que se optó por utilizar la técnica de **Clasificación de Tamaño fijo**.

Se identificó el nivel de granularidad de los requisitos anteriormente mencionados en dicho punto:

Requisitos de bajo nivel:

- Listar temas
- Buscar temas
- Ver comentarios
- Iniciar Sesión
- Cerrar Sesión

Requisitos de alto nivel:

- CRUD cuentas de informantes
- CRUD comentarios
- CRUD temas
- CRUD marcadores
- Calificar comentarios

Se van a manejar 3 clases, estas tres clases tienen las etiquetas de: Chico, mediano y grande. Para la primera clase con etiqueta Chico, serán los procesos funcionales que tenga un CFP entre 0 y 5, para la clase Medio son los procesos que tengan entre 6 y 10 CFP. Finalmente para la clase Grande son aquellos que tengan entre 11 y 15 CFP.

Clasificación	Tamaño en CFP	ENTRADAS (E)	SALIDAS (X)	LECTURAS (R)	ESCRITURAS (W)	MENSAJES DE ERROR
PEQUEÑO	5	1	1	1	1	1
MEDIO	10	2	2	3	2	1
GRANDE	15	3	3	4	4	2

Figura 4.3: Tabla para primera aproximación

Se seleccionaran los requisitos representativos, así tenemos los siguientes:

- CRUD temas. Ya que los 4 procesos funcionales son parte fundamental del propósito de la aplicación web
- CRUD cuentas. Junto al punto anterior las cuentas del administrador y la creación de los informantes son una parte fundamental de la aplicación web, ya que los informantes son los únicos que pueden agregar nuevos temas y marcadores; por otro lado, el administrador es quién puede agregar nuevos informantes.
- Iniciar Sesión La mayor parte de la creación de marcadores y temas se hace a través de las cuentas de los informantes, por lo que estos deben estar autenticados en sus perfiles.
- Buscar temas. Ya que se solicita filtros para facilitar la búsqueda de marcadores y temas haciendo la experiencia del usuario más amigable, se considera como un requerimiento funcional de vital importancia.

Requerimiento Funcional	Clase
Iniciar Sesión	Chico
Buscar Temas	Mediano
Calificar Comentarios	Mediano
CRUD temas	Grande
CRUD cuentas de informantes	Grande

Tabla 4.1: Clasificación de Requerimientos Funcionales por el método Clasificación por Tamaño Fijo

Requerimiento Funcional	Clase	CFP
Iniciar Sesión	Pequeño	5 CFP
Buscar Temas	Medio	10 CFP
Calificar Comentarios	Medio	10 CFP
CRUD temas	Grande	15 CFP
CRUD cuentas de informantes	Grande	15 CFP

Tabla 4.2: Clasificación de Requerimientos Funcionales por el método Clasificación por Tamaño Fijo

- Calificar comentarios. Otra parte fundamental es la calificación de los comentarios para indicar el interés que ha generado el marcador.

Para los requisitos funcionales de Iniciar Sesión, Buscar Temas y Calificar comentarios, al tener baja granularidad, se les asignarán las siguientes clases:

Así tendremos aproximadamente:

Tomando en cuenta los requisitos funcionales:

CRUD temas(15 CFP) es el representante de los requisitos CRUD cuentas (15 CFP), CRUD comentarios(15 CFP), CRUD temas(15 CFP), CRUD marcadores(15 CFP).

Iniciar Sesión(5 CFP) es el representante de Listar temas(5 CFP), Buscar temas(5 CFP), Cerrar Sesión(5 CFP).

Buscar Temas(10 CFP) es el representante de Calificar comentarios(10 CFP), Ver comentarios(10 CFP).

Así la cantidad máxima de CFP de la clase Pequeño es 20 CFP, para la clase Medio es 30 CFP y de la clase Grande es 60 CFP. Finalmente la estimación queda en 110 CFP.

Requerimiento Funcional	Categoría
CRUD Cuenta de informantes	Proceso típico
CRUD Marcadores	Proceso típico
Calificar Comentarios	Proceso funcional

Tabla 4.3: Clasificación de Requerimientos Funcionales por el método *Early and Quick*

4.3.2. Aproximación en la Fase de Planeación y Estimación

El desarrollo de la aplicación web fue hecho en 2 Sprints, por lo que se hicieron 2 aproximaciones en esta etapa. Para la elección de la técnica se tomó en cuenta que no se tuvo una lista de procesos funcionales debido a los constantes cambios efectuados en los Sprints, se validó cada requerimiento como un caso de uso que eran aprobados o no por la profesora a cargo del proyecto, por lo que la técnica elegida fue la técnica de **Temprana y Rápida *Early and Quick***.

1. **1er Sprint** Para el primer Sprint se seleccionaron los siguientes requisitos: CRUD cuenta de informates, CRUD comentarios, CRUD temas, CRUD marcadores, Calificar comentarios

   - Se seleccionan los siguientes requisitos como la muestra representativa: CRUD cuentas de informantes, CRUD marcadores, Calificar comentarios. La selección de estos requisitos es debido a que en este punto, se tenía mayor información sobre estos, se contaba con casos de uso revisados y aprobados, además se ser considerados como más densos de programar y se podía reutilizar una parte de código para los otros casos del Sprint.
   - A cada requerimiento, se le asigna la categoría correspondiente:
   - Al no tener datos pasados se tomará la tabla tal cuál se muestra en la Figura 4.4.

Requerimiento Funcional	Clase	CFP
CRUD Cuentas	Medio	32.3 CFP
CRUD Marcadores	Pequeño	20.4 CFP
Calificar comentarios	Medio	5 CFP

Tabla 4.4: Clasificación de Requerimientos Funcionales por el método *Early and Quick*

TIPO	NIVEL	RANGOS/ EQUIVALENTE COSMIC	MIN CFP	LO MAS PROBABLE	MAX CFP
Proceso Funcional	Pequeño	1-5 movimientos de datos	2.0	3.9	5.0
	Medio	5-8 movimientos de datos	5.0	6.9	8.0
	Grande	8-14 movimientos de datos	8.0	10.5	14.0
	Muy Grande	1-5 movimientos de datos	14.0	23.7	30.0
Proceso Típico	Pequeño	CRUD (procesos pequeños/ medianos) CRUD + lista (procesos pequeños)	15.6	20.4	27.6
	Medio	CRUD (procesos medianos/ grandes) CRUD + lista (procesos medianos) CRUD + lista + informe (procesos pequeños)	27.6	32.3	42.0
	Grande	CRUD (procesos grandes) CRUD + lista (procesos meidanos/ grandes) CRUD + lista + informe (procesos medianos)	42.0	48.5	63.0
Proceso General	Pequeño	6 - 10 FP genéricos	20.0	60.0	110.0
	Medio	10 - 15 FP genéricos	40.0	95.0	160.0
	Grande	15 - 20 FP genéricos	60.0	130.0	220.0
Proceso Macro	Pequeño	2 - 4 GP genéricos	120.0	285.0	520.0
	Medio	4 - 6 GP genéricos	240.0	475.0	780.0
	Grande	6 - 10 GP genéricos	360.0	760.0	1300.0

Figura 4.4: Tabla para el método Quick and Early

- Teniendo en cuenta que se tenían los respectivos Casos de Uso que han sido aprobados para los CRUD, se utilizará la columna de LO MÁS PROBABLE en combinación con la columna de MIN CFP para calificar comentarios. Así obtenemos lo siguiente:

  Por lo que tendríamos para todos los requisitos funcionales del Sprint serían:

- La suma total de los CFP se aproxima a 122.3 CFP.

2. **2ndo Sprint** Para el segundo Sprint, no todos los requisitos del primer Sprint fueron terminados debido a que se agregaron nuevas características por lo que fueron re-asignados a este Sprint. Por lo cual en este Sprint se hicieron los siguientes: Listar temas, Buscar temas, Ver comentarios, Iniciar Sesión, Cerrar Sesión, Crear cuentas de informantes y Actualizar temas.

Requerimiento Funcional	Clase	CFP
CRUD Cuentas	Medio	32.3 CFP
CRUD Marcadores	Pequeño	20.4 CFP
Calificar comentarios	Medio	5 CFP
CRUD Comentarios	Medio	32.3 CFP
CRUD Temas Temas	Medio	32.3 CFP
Total		122.3 CFP

Tabla 4.5: Clasificación de Requerimientos Funcionales por el método *Early and Quick*

Requerimiento Funcional	Categoría
Iniciar Sesión	Proceso funcional
Buscar Temas	Proceso funcional
Crear Cuenta	Proceso funcional
Actualizar Temas	Proceso funcional

Tabla 4.6: Clasificación de Requerimientos Funcionales por el método *Early and Quick*

- Se seleccionan los siguientes requisitos como la muestra representativa: Iniciar Sesión, Buscar temas, Crear cuentas de informantes y Actualizar temas.
- A cada requerimiento, se le asigna la categoría correspondiente:
- Al no tener datos pasados se tomará la tabla tal cuál se muestra en la Figura 4.4.
- Para este momento los requisitos que se encontraban con un nivel de granularidad bajo fueron afinados por lo que se utilizará la columna de LO MÁS PROBABLE. Para los requisitos funcionales que no se completaron del primer Sprint se utilizará la columna del MAX CFP debido a que se realizó una actualización del alcance del requerimiento.

  Por lo que tendríamos para todos los requisitos funcionales del Sprint serían:

- La suma total de los CFP se aproxima a 56.5 CFP.

En lo anterior se aprecia que para la primera estimación en la parte de Inicio se esperaba un número menor de 110 CFP, esto debido a que no se tenían completa la lista de requerimientos y se optó por trabajar con mayor holgura los CFP.

El primer Sprint generó 122.3. Conforme se fue desarrollando el proyecto y los casos de uso se fueron refinando, hubo un decremento en los CFP en el segundo Sprint quedando finalmente 56.5 CFP, los cuales fueron entregados al final del proyecto. Este decremento se debió a dos razones principales:

Requerimiento Funcional	Clase	CFP
Iniciar Sesión	Pequeño	3.9 CFP
Buscar Temas	Medio	6.9 CFP
Crear Cuenta	Grande	14.0 CFP
Actualizar Temas	Grande	14.0 CFP

Tabla 4.7: Clasificación de Requerimientos Funcionales por el método *Early and Quick*

Requerimiento Funcional	Clase	CFP
Iniciar Sesión	Pequeño	3.9 CFP
Buscar Temas	Medio	6.9 CFP
Crear Cuenta	Grande	14.0 CFP
Actualizar Temas	Grande	14.0 CFP
Listar Temas	Pequeño	6.9 CFP
Ver Comentarios	Medio	6.9 CFP
Cerrar Sesión	Pequeño	3.9 CFP
Total		56.5 CFP

Tabla 4.8: Suma de CFP totales en el segunda Sprint

- Al tener el refinamiento de los casos de uso, se pudo trabajar sin tanta holgura en la estimación hecha en la etapa de Planificación y Estimación.
- Por otro lado, el proyecto tuvo varios cambios como lo fueron la no implementación de algunas funciones o el cambio de alcance de unas funciones por otras.

Finalmente los CFP obtenidos en la segunda fase compuesta por los dos Sprints fueron 178.8 CFP, el cual es un número mayor al reportado en la fase de Inicio.

Capítulo 5

Conclusiones

A lo largo de esta tesis la investigación hecha se centro en responder las preguntas de investigación mostradas en el primer capitulo, encontrando una cantidad considerable de material; sin embargo, en comparación con otras ramas de investigación de la Ingeniería de software como el desarrollo, donde anualmente se publican diversos artículos que muestran nuevas tecnologías o como usarlas.

Las respuestas encontradas a las preguntas de investigación son las siguientes:

1. **¿Cuál es el estado actual de la estimación en el desarrollo de software con metodologías ágiles?**
   - Para el año 2013 el panorama de la estimación de software era sombrío debido a el poco interés de investigadores respecto al tema (56).
   - Hasta el año 2020 se ha visto un incremento es artículos que tratan sobre la estimación de software, lo cual implica un interés sobre esta rama de la Ingeniería de Software(19).
   - A pesar de existir más métodos basados en Inteligencia Artificial que es una rama que se encuentra en auge según la revista FORBES(21), el juicio de experto sigue siendo tomando a consideración en los métodos estimación(19).
2. **¿Cómo puede ayudar a mejorar un método de estimación basado en COSMIC las estimaciones?**
   - Al generar resultados cuantitativos, estos pueden ser comparables.
   - Replicación de resultados para su posterior análisis, permitiendo buscar propuestas novedosas que solucionen los problemas de esfuerzo.
3. **¿Cuáles son las principales ventajas de utilizar el método COSMIC como base para las estimaciones?**
   - La propuesta es fácil de aplicar teniendo las nociones básicas de COSMIC y de SCRUM.
   - Genera resultado cuantitativos.

- Permite la replicación de resultados independientemente de la persona a cargo de efectuar la estimación.
- No existen complicaciones hasta el momento encontradas de la integración de COSMIC con SCRUM hasta el momento, conforme exista un mayor interés en la comunidad por la integración en COSMIC se podrá tener un panorama más amplio.

4. **¿Cómo lograr estimar el esfuerzo basado en el estándar 19761 COSMIC en una metodología ágil, la cual está enfocada en una rápida respuesta al cambio y tiene iteraciones muy cortas?**

    - En esta tesis se propuso una forma de estimar esfuerzo con COSMIC utilizando metodologías ágiles, dando una pauta que pueda servir a futuro a nuevas pautas. COSMIC nos permite medir requerimientos funcionales, los cuales pueden ser determinados en las primeras etapas del ciclo de SCRUM; sin embargo algunos requisitos no funcionales que posteriormente deriven en requisitos funcionales también podrán ser estimados.
    - Generación de estimaciones tanto en la etapa de Inicio y en la fase de Implementación y Estimación de SCRUM, lo cual no es común(19).

5.0.1. Trabajo a Futuro

En esta tesis se trabajo con un ejemplo pequeño. Por falta de tiempo no pudo implementarse la propuesta en un proyecto grande, por lo que se plantean las siguientes opciones como trabajo a futuro:

1. Aplicar la propuesta en un proyecto más robusto que contenga más iteraciones en su fase de implementación, esto mostraría como se van generando nuevos requerimientos funcionales y afinando los requerimientos funcionales previamente estimados.
2. Aplicar la propuesta con diferentes equipos de trabajo, esto para poder comparar los resultados encontrados que como ya mencionó, los resultados serán datos cuantitativos.
3. Finalmente la propuesta generada es una pauta para integrar COSMIC a la metodología ágil SCRUM, pero se puede extender a otras metodologías ágiles como Kanban, Extreme Programming, Desing Sprint, étc.

5.0.2. Aprendizaje

El desarrollo de esta tesis ha sido un proceso largo pero gratificante, que ha comprendido el hecho de buscar material de apoyo con sus respectivas referencias, aprender sobre nuevos temas que implícitamente eran necesarios para el entendimiento completo de sub-temas que iban apareciendo, hasta buscar formatos adecuados que sean visualmente agradables a la vista del lector.

Por otro lado respecto al tema escogido debo mencionar que no es un tema que sea mencionado en la mayoría de clases o cursos que haya tomado, salvo en dos clases en maestría, por lo que poder adentrarme más en el para comprender las dificultades y áreas de oportunidad que existen en la Ingeniería de Software. Aunque no sea un tema del completo agrado de la

comunidad que desarrolla software, considero que leer e informarse sobre nuevas propuestas es de vital importancia para seguir creciendo y desarrollándonos.

Esta tesis me ayudó en dos aspectos fundamentales: el aspecto académico y laboral. En el aspecto académico el hecho de crear un texto de carácter científico que implica cierto rigor, además de una estructura y constantes revisiones permite explorar y tener un acercamiento al área académica. Por otro lado me permitió identificar áreas fuertes y débiles de mi escritura y redacción, cualidades que son mayormente estudiadas en primaria y secundaria pero que nos acompañaran el resto de nuestra vida. Por otra parte en el aspecto laboral, el poder consultar artículos que hablan directamente de como se encuentra la industria y su evolución me proporcionó un panorama más amplio y una posible opción sobre dedicarme al área de Project Managment, el cual no era considerado por mi como una opción de trabajo.

Bibliografía

[1] Abran, A. (2015). *Software Project Estimation The Fundamentals for Providing High Quality Information to Decision Makers*. Wiley-IEEE Computer Society PR. 13, 14, 15, 16

[2] Alain Abran, P. F. and Lestherhuis, A. (2020). Manual de medición de cosmic para iso 19761. *COSMIC standar*, pages 1–200. 22

[3] Alexandrov Vladımir V, Avila Pozos Roberto, C.-M. C. C. C. R. E. R. J. A. G. P. A. L. L. F. J. L. and Gomez, L. C. (2015). *Modelización matemática. Principios y aplicaciones*. Editorial de la Benemérita Universidad Autónoma de Puebla. 15

[4] Andreas Schmietendorf, M. K. and Dumke, R. (2008). Effort estimation for agile software development projects. *Proceedings of 5th Software Measurement European Forum*, pages 113 – 126. 19

[5] Angara, J., Prasad, S., and Gutta, S. (2018). Towards benchmarking user stories estimation with cosmic function points-a case example of participant observatio. *International Journal of Electrical and Computer Engineering (IJECE)*, 8:3076. 4, 41

[6] Association, N. S. M. U. (2009). Function point analysis for software enhancement. professional guide of the netherlands software metrics users association. guide 2.2, Netherlands Software Metrics Users Association. 21

[7] Ayyıldız, T. E. and Terzi, H. C. (2017). Case study on software effort estimation. *International Journal of Information and Electronics Engineering*, 7(3):103. 17

[8] Bhatt, K., Tarey, V., and Patel, P. (2012). Analysis of source lines of code(sloc) metric. *IJETAE*, 2. 19, 20

[9] Briand, L.C., M. S. y. B. V. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–85. 12

[10] Coelho, E. and Basu, A. (2012). Effort estimation in agile software development using story points. *International Journal of Applied Information Systems*, pages 7 – 10. 19

[11] Commeyne, C., Abran, A., and Djouab, R. (2016). Effort estimation with story points and cosmic function points - an industry case study. volume 21, pages 25 – 36. 43

[12] Conte, S.D., D. D. and Shen, V. (1986). *Software Engineering Metrics and Models*. The Benjamin/Cummings Publishing Company, Inc. 17, 42

[13] (COSMIC), T. C. S. M. I. C. (2020). Early software sizing with cosmic: Experts guide. 28, 29, 30, 32, 33, 34, 35, 49, 50

[14] (COSMIC), T. C. S. M. I. C. (2021). Cosmic measurement manual. 23, 25, 27, 28

[15] Deloitte (11 de Agosto de 2021). `https://www2.deloitte.com/es/es/blog/todo-tecnologia/2021/los-interesados-en-scrum.html`. 6

[16] Dumke, R. and Abran, A. (2014). *Software Measurement*. Springer, New York. 11, 12

[17] Fenton, N. (1994). Software measurement: a necessary scientific basis. *IEEE Transactions on Software Engineering*, 20(3):199–206. 12

[18] Fenton, N. and Bieman, J. (2015). *Software Metrics. A rigorous ans practical approach*. CRC Press. 18

[19] Fernández-Diego, M., Méndez, E. R., González-Ladrón-De-Guevara, F., Abrahão, S., and Insfran, E. (2020). An update on effort estimation in agile software development: A systematic literature review. *IEEE Access*, 8:166768–166800. 1, 4, 7, 37, 40, 44, 58, 59

[20] Finkelstein, L. and Leaning, M. (1984). A review of the fundamental concepts of measurement. *Measurement*, 2(1):25–34. 11

[21] FORBES (09 de Agosto de 2022). `https://www.forbes.com.mx/machine-learning-y-auge-de-inteligencia-artificial/`. 58

[22] from Software, S. R. and Vocabulary, S. E. (11 de Agosto de 2017). `https://pascal.computer.org/sev`$_d$*isplay*/*index.action*. 5

[23] Gallego, M. T. (2017). Metodología scrum. *Gestión de proyectos informáticos*, pages 1–56. 7

[24] Gencel, C. and Demirors, O. (2008). Functional size measurement revisited. *ACM Trans. Softw. Eng. Methodol.*, 17:2 – 36. 20, 21

[25] Gibbons, J. D. and Pratt, J. W. (1975). P-values: Interpretation and methodology. *The American Statistician*, 29(1):20–25. 17

[26] GUIDES, S. (14 de Septiembre de 2021). `https://scrumguides.org/scrum-guide.html`. 8, 9

[27] Hacaloglu, T. and Demirörs, O. (2019). Measureability of functional size in agile software projects: Multiple case studies with cosmic fsm. *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 204–211. 4, 41

[28] Hacaloglu, T. and Demirors, O. (2019). Measureability of functional size in agile software projects: Multiple case studies with cosmic fsm. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 204–211. 4, 41

[29] IEEE (1980). Ieee standard dictionary of electrical and electronics terms. *IEEE Transactions on Power Apparatus and Systems*, PAS-99(6):37a–37a. 19

[30] Institute, P. M. (2017). Agile practice guide. 5

[31] ISO 14143:2012(en) (2012). Information technology — Software measurement — Functional size measurement — Part 6: Guide for use of ISO/IEC 14143 series and related International Standards. Standard, International Organization for Standardization, Geneva, CH. 22

[32] ISO 19761:2011(E) (2011). Software engineering — COSMIC: a functional size measurement method. Standard, International Organization for Standardization, Geneva, CH. 22

[33] ISO 20968:2002(E) (2002). Software engineering — Mk II Function Point Analysis — Counting Practices Manual. Standard, International Organization for Standardization, Geneva, CH. 21

[34] ISO 24570:2018(E) (2018). Software engineering — NESMA functional size measurement method . Standard, International Organization for Standardization, Geneva, CH. 21

[35] J. Becker, T. Bernhold, D. B. N. K. R. K. V. L. H. P. R. (2012). Construction of productivity models. *Enterprise Modelling and Information Systems Architectures*, 7(1):28–42. 15

[36] Jones, C. et al. (1988). Feature points (function point logic for real time and system software). In *IFPUG Fall Conference*. 21

[37] Keaveney, S. and Conboy, K. (2006). Cost estimation in agile development projects. *Fourteenth European Conference on Information Systems*, pages 183 – 197. 19

[38] Kent Beck, e. (14 de Septiembre de 2021). `https://agilemanifesto.org/iso/es/manifesto.html`. 5, 6, 7

[39] Khuat, T. and Thi My Hanh, L. (2017). A novel hybrid abc-pso algorithm for effort estimation of software projects using agile methodologies. *Journal of Intelligent Systems*, 27. 17

[40] Kitchenham, B. (1997). The problem with function points. *Software, IEEE*, 14:29 – 31. 43

[41] Lee, T. K., Wei, K. T., and Ghani, A. A. A. (2016). Systematic literature review on effort estimation for open sources (oss) web application development. *2016 Future Technologies Conference (FTC)*, pages 1158–1167. 1

[42] Marcela Maya, Alain Abran, S. O. D. S.-P. J.-M. D. (1998). Measuring the functional size of real-time software. *European Software Control and Metrics Conference*, pages 191–199. 21

[43] Matson, J., Barrett, B., and Mellichamp, J. (1994). Software development cost estimation using function points. *IEEE Transactions on Software Engineering*, 20(4):275–287. 21

[44] McConnell, S. (2006). *Software estimation: demystifying the black art*. Microsoft press. 12, 17

[45] Mora, B., García, F., Ruiz, F., Piattini, M., Boronat, A., Gómez, A., Carsí, J. Á., and Ramos, I. (2007). Marco de trabajo basado en mda para la medición genérica del software. In *JISBD*, pages 211–220. 12

[46] Morris, W. (1982). *The American Heritage Dictionary*. Person. 12

[47] Ochodek, M. (2016). Approximation of cosmic functional size of scenario-based requirements in agile based on syntactic linguistic features—a replication study. pages 201–211. 41

[48] Okoli, C. and Schabram, K. (2010). A guide to conducting a systematic literature review of information systems research. *SSRN Electronic Journal*, 10. 37

[49] Oligny, S., Desharnais, J.-M., and Abran, A. (1999). A method for measuring the functional size of embedded software. pages 7–9. 20, 21

[50] Pekka Abrahamsson, K. C. and Wang, X. (2009). Lots done, more to do: The current state of agile systems development researc. *European Journal of Information Systems*, page 281. 5

[51] Pressman, R. S. (2010). *Ingeniería de Software. Un enfoque práctico*. McGraw Hill. 18

[52] Rask, R., Laamanen, P., and Lyyttinen, K. (1993). Simulation and comparison of albrecht's function point and demarco's function bang metrics in a case environment. *IEEE Transactions on Software Engineering*, 19(7):661–671. 20

[53] Rubey, R. J. and Hartwick, R. D. (1968). Quantitative measurement of program quality. page 671–677, New York, NY, USA. Association for Computing Machinery. 11

[54] Sudarmaningtyas, P. and Mohamed, R. B. (2021). A review article on software effort estimation in agile methodology. *pertanika journal of science and technology*, 29. 1, 4, 37, 44

[55] Teologlou, G. (1999). Measuring object oriented software with predictive object points. *Conference on European Software Confrol and Metrics*, 10:2–14. 22

[56] Usman, M., Mendes, E., Neiva, F., and Britto, R. (2014). Effort estimation in agile software development: A systematic literature review. 1, 2, 4, 7, 37, 38, 40, 41, 47, 58

[57] Valdés-Souto, F. and Naranjo-Albarrán, L. (2021a). Improving the software estimation models based on functional size through validation of the assumptions behind the linear regression and the use of the confidence intervals when the reference database presents a wedge-shape form. *Program. Comput. Softw.*, 47(8):673–693. 16, 35

[58] Valdés-Souto, F. and Naranjo-Albarrán, L. (2021b). Improving the software estimation models based on functional size through validation of the assumptions behind the linear regression and the use of the confidence intervals when the reference database presents a wedge-shape form. *Program. Comput. Softw.*, 47(8):673–693. 35

[59] Valdés-Souto, F. and Naranjo-Albarrán, L. (2021). Improving the software estimation models based on functional size through validation of the assumptions behind the linear regression and the use of the confidence intervals when the reference database presents a wedge-shape form. *Programming and Computer Software*, 47:673–693. 36, 46

[60] Wen, J., Li, S., Lin, Z., Hu, Y., and Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Inf. Softw. Technol.*, 54:41–59. 17

[61] Wiener, N. (1964). *The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction*, pages 129–148. 17

[62] Zuse, H. (2013). *3 History of Software Measurement*, pages 57–80. De Gruyter. 20