Universidad de las Ciencias Informáticas

Facultades 3 y 7

PROPUESTA DE UN SISTEMA DE MÉTRICAS PARA LA EVALUACIÓN DE LOS PROYECTOS DE GESTIÓN DE LA UCI.

Trabajo de Diploma para optar por el Título de Ingeniero Informático

Autores: Lisset Torres Ilina

Reinel Pérez González

Tutor: Ing. Alejandro Notario Laborí

Ciudad Habana, Mayo 2007

"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Por este medio declaramos que somos los únicos autores de este trabajo y autorizamos a las facultades 3 y 7 de la Universidad de las Ciencias Informáticas para que hagan el uso que estimen pertinente con el mismo.

Para que así conste firmamos la presente a los ___ días del mes de mayo de 2007.

_____ _____ _____

Lisset Torres Ilina Reinel Pérez González Ing. Alejandro Notario Laborí

DEDICATORIA

A todas las personas que puedan usar este conocimiento para el bien de todas las especies que habitan el planeta.

AGRADECIMIENTOS

De Lisset:

Agradezco, primero que todo, a mi mamita, que si no la pongo de primera monta el berro; pero más que todo porque fue la que me trajo a la vida, y esa ha sido la condición necesaria para recibirme de ingeniera. Lo demás ha sido circunstancial.

A papi, porque me hizo razonar cuando quise abandonar la UCI y por sus buenos consejos, espero que no se te acaben.

A mi sister linda. Lo que te espera es mucho, esto es solo el comienzo. Ah! Y por darme ese sobrinón tan lindo, espero que haya muchos más.

A María, por su fe en las cosas buenas que animan este mundo.

A Alekos, ¡el mejor tutor del mundo!, espero que este sea el comienzo de una larga amistad.

A Maire y Yailín por haberme aguantado y ser mis amigas estos cinco años. No se van a librar de mí tan fácilmente.

A Pedrín, por darle sentido a mi vida y curso a mis pensamientos, por ser mi mejor amigo y quererme de esa forma tan especial. Por enseñarme acerca del intento, de la energía y la magia que están atrapados en cada uno de nosotros.

Y a Pirata, ¿por qué no?

De Reinel:

Agradezco a mis padres por haberme apoyado en todo momento, y por darme fuerzas para seguir adelante.

A mi tutor Alejandro, quien nos apoyó muchísimo en todo momento, doy mil gracias por su ayuda.

RESUMEN

No te obsesiones por la métrica "perfecta" porque no existe.

A nivel mundial, toda empresa informática que aplique correctamente la ingeniería al software y que tiene entre sus metas alcanzar la mayor calidad de su producto sabe que medir y evaluar sus productos y procesos a través de métricas e indicadores es un requisito importante a cumplir para alcanzar sus objetivos.

Cientos de trabajos científicos de personalidades del área y el trabajo empírico de muchas empresas de prestigio confirman que las métricas proporcionan al desarrollador una vista significativa de la calidad, tamaño y evaluación de un proyecto.

A pesar de ser conocida y argumentada dicha importancia no existe un estándar general que aúne todas las métricas que se pueden aplicar a determinado tipo de proyecto, cada cual aplica las métricas que miden la parte en particular que le interesa o crea otras nuevas que respondan a sus necesidades. A pesar de esta aparente dicotomía presente en la literatura, estas propuestas no se contradicen, más bien se presentan como opciones a elegir, ya que ambas han dado buenos resultados.

En el presente trabajo se propone un Sistema de Métricas para la Evaluación de los Proyectos de Gestión en la Universidad de las Ciencias Informáticas (UCI) con la intención de que constituya un punto de partida para el uso de las mismas y ayude a fomentar una cultura en torno al tema.

Dicho sistema facilita la interpretación de la información de métricas e indicadores de forma uniforme para todos los proyectos de software de gestión a los que se aplique, y permite comparar los resultados consistentemente.

PALABRAS CLAVES: métricas, software de gestión.

ÍNDICE

ÍINDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Nuestro país se encuentra inmerso en un proceso de transformación en todos los sectores de la sociedad, haciendo un uso adecuado de las nuevas tecnologías de la informática y las comunicaciones, queriendo lograr un buen funcionamiento de las mismas y que beneficien al pueblo.

Como un medio y una alternativa importante para lograr todo lo antes mencionado, surge la UCI, en la cual se desarrollan proyectos, cuyo propósito es el desarrollo de software que respondan a las necesidades sociales del sistema, incidiendo directamente en variadas ramas de la economía, fundamentalmente la educación, la salud y el turismo, y además, para insertarse en el mercado internacional del software.

Gran parte del software que se desarrolla en la UCI es del tipo de gestión, que se define como aplicaciones informáticas o programas informáticos diseñados para facilitar al usuario la realización de un determinado tipo de trabajo. Suele resultar una solución informática para la automatización de ciertas tareas complicadas como puede ser la contabilidad o la gestión de un almacén. Ahora bien, para poder triunfar en este sector y lograr un servicio óptimo resulta imprescindible que estos software cuenten con la calidad requerida. Existen diversas formas de gestionar esta calidad, una de ellas, quizá la más objetiva, es haciendo uso de las métricas del software.

Las métricas no son más que medidas o colecciones de datos de las actividades de los proyectos y recursos [1]. Estas deben ser simples, objetivas, fáciles de coleccionar, fáciles de interpretar y difíciles de malinterpretar. En la actualidad el uso de las métricas se extiende rápidamente entre la comunidad de desarrolladores de software debido a que las mismas producen indicadores a partir de los cuales se pueden tomar decisiones importantes. A pesar de esto, en la UCI se da la siguiente **situación problemática**:

- Se hace un uso incipiente de las métricas de forma dispersa y no centralizada.

- No constituye un requerimiento obligatorio a pesar de su importancia y el valor que le proporciona al proyecto.
- No es un parámetro a partir del cual se tomen acciones concretas durante el proceso de desarrollo de software.
- No determinan un camino a seguir en un punto crítico o crucial del proceso, pudiendo ver a tiempo que algo va mal y tomar las medidas necesarias para corregir el error y un posible fallo del sistema.

A pesar de existir varios sistemas de métricas, probadas y aceptadas en cientos de instituciones y empresas prestigiosas del mundo con resultados importantes, en la UCI no se cuenta con un sistema de métricas adaptado a sus características que permita evaluar desde el punto de vista cuantitativo los proyectos de software de gestión.

De lo anteriormente expuesto, el **problema científico** de la presente investigación queda formulado como: ¿Qué sistema de métricas se adecuará al entorno de producción de la UCI para ser aplicado en los proyectos productivos de gestión, permitiendo su evaluación de una forma centralizada y estándar?

En correspondencia con el problema, el **objeto de estudio** lo constituye: las métricas de software para los proyectos productivos de gestión de la UCI.

El **objetivo general** es: Proponer un sistema de métricas que permita evaluar de una forma centralizada y estándar el avance de los proyectos productivos de gestión en la UCI.

Tareas de la Investigación:

1. Estudiar las métricas de software más utilizadas en la actualidad.
2. Estudiar el entorno de producción de la UCI para determinar qué métricas se adecuan a este.

3. Seleccionar las métricas que mejor se adaptan al entorno de producción de la UCI

4. Proponer el sistema de métricas para evaluar los proyectos productivos de gestión de la UCI.

5. Describir las métricas seleccionadas que conformarán el sistema a proponer.

Estructuración del Contenido:

Capítulo 1. Fundamentación Teórica.

Se estudian los fundamentos teóricos relacionados con las métricas del software y se realiza un estudio del entorno de producción de la UCI para conocer cómo funcionan sus proyectos de software de gestión.

Capítulo 2. Estudio de las Métricas del Software.

Se hace un estudio de las métricas más usadas a nivel mundial, incluyendo una breve descripción de cada una, especificando sus características fundamentales, así como sus ventajas y desventajas.

Capítulo 3. Elección de un Sistema de Métricas para la UCI.

Se hace un análisis crítico de las métricas estudiadas para conformar el sistema de métricas y queda conformada la Propuesta del Sistema de Métricas para la Evaluación de los Proyectos de Software de Gestión de la UCI.

Capítulo 4. Descripción del Sistema de Métricas elegido.

Se describen detalladamente cada una de las métricas que conforman la Propuesta del Sistema de Métricas para Evaluar lo Proyectos de Software de Gestión de la UCI establecida en el capítulo 3.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

En este capítulo se estudian los fundamentos teóricos relacionados con las métricas del software y se realiza un estudio del entorno de producción de la UCI para conocer cómo funcionan sus proyectos de software de gestión dadas sus características distintivas de ser una universidad que combina la enseñanza académica con la producción y comercialización de software.

Este primer acercamiento se hace con el objetivo final de conocer hasta qué punto los proyectos de software de gestión de la universidad usan métricas para evaluar la calidad y productividad de los mismos y así detectar la situación problemática para fundamentar el objetivo de la presente investigación.

1.2 Las métricas del software.

El Instituto de Ingeniería Eléctrica y Electrónica (IEEE por sus siglas en inglés) define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado [2].

Las métricas de procesos de software son algo más que una simple medida de algún "atributo" del proceso de software; una métrica es información que sirve para planificar, predecir y evaluar el estado de un proyecto [3].

Otros autores definen métricas de software como: "La continua aplicación de técnicas basadas en la medición al proceso de desarrollo de software y a sus productos para proveer información administrativa significativa y oportuna, junto con el uso de esas técnicas para mejorar el proceso y sus productos" [4]. La Fig 1.1 ilustra una expansión de esta definición para enfatizar que las métricas de software proveen la información necesitada por los ingenieros para decisiones técnicas.

Fig 1. 1 Métricas de Software.

Las mediciones de software son usadas para medir atributos específicos de un producto de software o del proceso de desarrollo de software[5].

Se usan fundamentalmente para [6]:

- Obtener las bases para la estimación.
- Seguir el progreso de los proyectos.
- Determinar la complejidad (relativa).
- Ayudar a comprender cuándo se ha alcanzado un estado deseado de calidad.
- Analizar los defectos.
- Validar experimentalmente las mejores prácticas.

En resumen, ayudan a tomar mejores decisiones.

Los principios fundamentales que deben seguir las métricas son [7]:

- Las métricas deben ser simples, objetivas, fáciles de coleccionar, fáciles de interpretar y difíciles de malinterpretar.
- La colección de las métricas debe ser automática y no intrusiva, o sea, no interferir en las actividades de los desarrolladores.

- Las métricas deben contribuir a la evaluación de la calidad temprana en el ciclo de vida, cuando los esfuerzos por mejorar la calidad del software son efectivos.

- Los valores absolutos y las tendencias de las métricas deben ser usados activamente por el personal administrativo y el personal ingenieril, para comunicar progreso y calidad en un formato coherente.

- La selección de un mínimo o más extensivo conjunto de métricas, dependerá de las características y contexto del proyecto: Si es muy grande o si tiene restricciones de seguridad o de confiabilidad de los requerimientos; y si el equipo de desarrollo y de valoración (evaluación) es conocedor de las métricas, lo cual hará muy útil coleccionar y analizar las métricas técnicas.

Las métricas y mediciones se enmarcan dentro del Proceso de Gestión, pero se aplican durante todo el proceso de desarrollo del software de forma organizada a través de un sistema o set de métricas que el proyecto u organización determine dadas sus características.

Se usan para ayudar a que el producto gane en eficacia en cuanto al proceso del software y de los proyectos que se guían por los mismos. Los datos de calidad y productividad son recogidos, analizados, comparados con otros resultados para determinar si han mejorado la calidad y la productividad. Las métricas son igualmente usadas para detectar problemas puntuales cuyas soluciones pueden ser desarrolladas y los procesos mejorados [8].

En la Fig 1.1 se puede ver cómo transcurre todo el proceso de recopilación, cálculo y evaluación de las métricas. Los datos se recogen del proceso, el proyecto y el producto obteniéndose medidas con las cuales se calculan las métricas y finalmente se obtienen los indicadores después de evaluar las métricas.

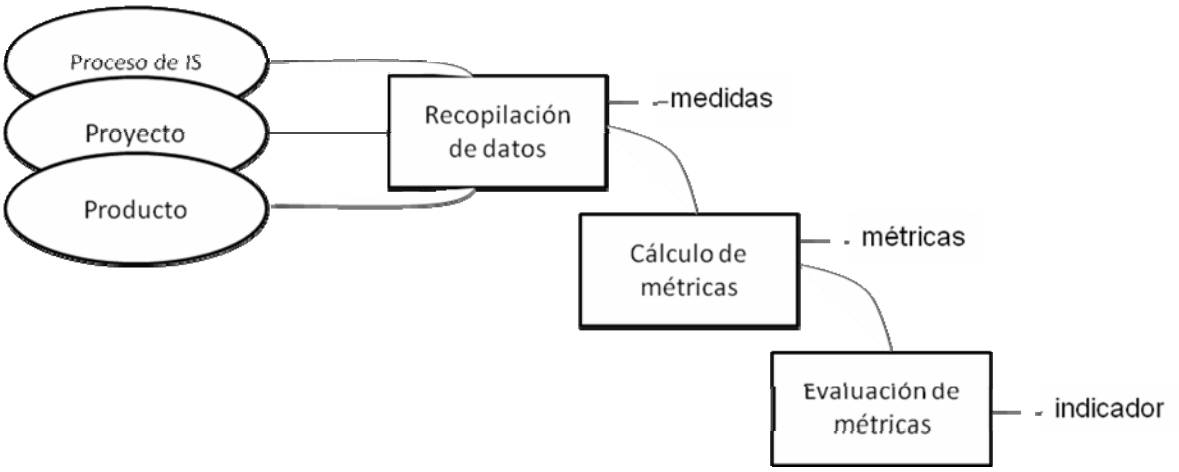


Fig 1. 2 Proceso de recopilación, cálculo y evaluación de métricas.

1.3 Las mediciones en los modelos de procesos del software.

A partir de los años 80 se comenzaron a desarrollar modelos de procesos específicos para el software. En este sentido la International Organization for Standardization 9000 (ISO 9000) desarrolló ISO 9000-3 y la British Standards Institution (BSI) hizo lo propio con el Programa de Certificación de Calidad para Software (TickIT).

Sin embargo las dos líneas que surgieron a principios de los 90 y que continúan como referente en la actualidad son: Capability Maturity Model Integration (CMMI) y las normalizaciones de ISO 15504 [9].

A continuación se describen estos dos últimos modelos, ya que después de un estudio sobre el tema, se llegó a la conclusión de que estos son los más usados internacionalmente en el momento en que esta investigación es redactada, además de ser los que mejor describen los procesos que se efectúan durante el desarrollo de un proyecto de software y lo más importante, contienen en su concepción procesos de medición de software; también se describe el Modelo de Procesos para la Industria del Software (Moprosoft) de factura mexicana, por su similitud con los procesos productivos que se han estado ejecutando actualmente en la UCI.

1.3.1 CMMI.

El Software Engineering Institute (SEI), perteneciente a la Carnegie Mellon University, un auténtico peso pesado en la normalización de los procesos del software, desarrolló su línea de trabajo sobre el concepto de “madurez” de las organizaciones para producir software [9].

En 1991, después de definir un marco de procesos de madurez y un esquema para mejorar dichos procesos, publicó el Modelo de Madurez de Capacidades para el desarrollo de software (CMM por sus siglas en inglés), que tras más de una década de existencia ha demostrado que en muchas organizaciones ha resultado eficaz.

El modelo CMM también determina los estados por los cuales transita un proceso de desarrollo de software. Plantea que actualmente no existe un modelo universalmente aceptado de medidas de procesos o de calidad por lo que insta a las organizaciones a identificar por cada Área Clave del proceso uno o más conjuntos de métricas significativas adecuadas para su entorno, industria y cultura.

En 2005 el SEI definió el CMMI que incluye los procesos de las organizaciones, así como también los procesos de los equipos de desarrollo y los procesos personales, facilitando la implantación de forma conjunta y simultánea de estos modelos.

CMMI establece 6 niveles para determinar la capacidad de un proceso:

- Nivel 0. Incompleto: El proceso no se realiza.
- Nivel 1. Realizado: Se lleva a cabo el proceso, consiguiendo transformar elementos de entrada identificados, en productos de salida.
- Nivel 2. Gestionado: El proceso se ejecuta siempre de la misma manera, de una forma gestionada. Dispone de un conjunto representativo de métricas a nivel de gestión del proyecto (estimaciones).

- Nivel 3. Definido: El proceso está definido en la organización y se ejecuta siempre. Se dispone de un conjunto de métricas a nivel organizacional que facilita realizar valoraciones sobre los proyectos en su conjunto. También se definen métricas relacionadas con la calidad y funcionalidad de los productos.

- Nivel 4. Cuantificadamente gestionado: La ejecución del proceso tiene institucionalizado en la organización un sistema de medición objetivo y cuantificable de su capacidad. La medición se basa en la planificación y gestión de la calidad de los procesos y productos de una forma estadística.

- Nivel 5. Optimizado: El proceso, que se ejecuta siempre, está definido en la organización, se mide y está integrado en un plan, también institucionalizado, de mejora continua basada en las mediciones de los procesos. La medición se basa en la planificación y gestión de la calidad de los procesos y productos de una forma estadística.

De este reconocido modelo, es importante destacar la importancia que se le atribuye a las métricas del software y cómo a partir del segundo nivel de madurez se establece el uso de estas mediciones a los diferentes atributos del software.

1.3.2 SPICE – ISO/IEC STD 15504.

Mientras SEI publicaba y comenzaba a definir su modelo CMM, ISO emprendía los otros dos proyectos que hoy forman los principales puntos de referencia en el ámbito de la calidad para la industria del software: ISO 12207 e ISO 15504. A continuación se explica esta última [9].

El primer borrador de Software Process Improvement and Capability dEtermination (SPICE) fue creado en junio de 1995. En 1998, el trabajo pasó a la fase de informe técnico con la denominación ISO/IEC TR 15504.

La instrucción técnica consta de 9 apartados, que se han ido publicando como redacción definitiva del estándar internacional ISO/IEC 15504 durante el período 2003-2005.

Aunque ISO comenzó con el proyecto SPICE algo más tarde que SEI con el modelo CMM, durante su evolución han ido convergiendo, de forma que la conformidad con uno de ellos implica la también conformidad con el otro. ISO/IEC 15504, al igual que CMMI es un modelo para evaluar los procesos de la organización y determinar si resultan efectivos para conseguir los objetivos.

El proceso de medición del mismo supone la definición de métricas, la gestión de los datos (incluidos los datos históricos) y el uso de las métricas en la organización y tiene como objetivo implementar métricas de proceso y de producto como soporte a la gestión efectiva y a la posibilidad de demostrar objetivamente la calidad de los productos.

1.3.3 Moprosoft.

MoProSoft surge por iniciativa de la Secretaría de Economía Mexicana y gracias al trabajo de académicos y empresarios de este país, encabezados por la Dra. Hanna Oktaba, profesora de la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) [10]. MoProSoft es un modelo de procesos para la industria de software mexicano, que fomenta la estandarización de su operación a través de la incorporación de las mejores prácticas en gestión e ingeniería de software. La adopción del modelo permite elevar la capacidad de las organizaciones que desarrollan o mantienen software para ofrecer servicios con calidad y alcanzar niveles internacionales de competitividad. Es también aplicable en áreas internas de desarrollo de software de las empresas de diversos giros.

Este modelo es específico para el desarrollo y mantenimiento de software. Entre sus características se cuentan:

- Sencillo de entender y adoptar.
- Facilita el cumplimiento de los requisitos de otros modelos como ISO 9000:2000, CMM y CMMI.
- Se enfoca a procesos.
- Se le considera práctico en su aplicación, principalmente en organizaciones pequeñas, con bajos niveles de madurez.
- Comprende un documento de menos de 200 páginas que, al compararlo con otros modelos y estándares, lo hace bastante práctico.
- Está orientado a mejorar los procesos, para contribuir a los objetivos de negocio, y no simplemente ser un marco de referencia o certificación.
- Tiene un bajo costo, tanto para su adopción como para su evaluación.

MoProSoft es un modelo integrado donde las salidas de un proceso están claramente dirigidas como entradas a otros; las prácticas de planeación, seguimiento y evaluación se incluyeron en todos los procesos de gestión y administración; por su parte los objetivos, los indicadores, las mediciones y las metas cuantitativas fueron incorporados de manera congruente y práctica en todos los procesos; las verificaciones, validaciones y pruebas están incluidas de manera explícita dentro de las actividades de los procesos; y existe una base de conocimientos que resguarda todos los documentos y productos generados.

La implantación de MoProSoft no demanda la incorporación de personal especializado en las empresas, únicamente requiere de una adecuada capacitación del personal existente.

1.4 Descripción del entorno de producción de la UCI.

La UCI fue creada por la dirección de la Revolución para convertir la informática en la rama más productiva de la economía, aportadora de recursos para la nación, buscando fuentes de riqueza que apenas usen materia prima y para ayudar a la formación de una sociedad intelectual, ya que producir conocimiento deviene uno de los recursos más importantes en la era actual.

Por esta razón las dos Misiones que se traza la UCI como organización son:

- Formar profesionales altamente calificados en la rama de la informática que estén comprometidos con su Patria.
- Producir software y servicios informáticos a partir de la vinculación estudio-producción como modelo de formación.

Cuando surge la UCI durante el curso 2002/2003 no existían proyectos, ni claridad en cómo se iba a enfrentar el tema de la producción. Se estudiaban los Parques Tecnológicos.

En el 2003/2004 se empiezan a buscar proyectos en todos los organismos del país. Se crean un gran número de proyectos pequeños y, aunque no están claras cuáles son sus prioridades, se dan los primeros pasos para una organización metodológica estructurando la producción con un Director de Infraestructura Productiva (IP) y Vicedecanos de Producción. Durante este curso se comienzan a tener las primeras experiencias de impacto en Venezuela.

Ya para el curso 2004/2005 se crea la Vicerrectoría Primera para unir la atención a los procesos de formación y producción. Los proyectos más importantes se organizan y dirigen desde la IP.

La actual estructura de dirección de la IP de la UCI, que se puede ver en la Fig 1.3, está compuesta por una Dirección General que se encuentra subordinada directamente a la

Vicerrectoría Primera y se compone de cinco Direcciones de Control de la Producción que atienden cada una dos facultades, las supervisan, atienden sus solicitudes y problemas y exigen el cumplimiento de las tareas asignadas a los Polos Productivos de cada facultad. Además cuenta con cinco Direcciones de Servicios de la Producción. Estas direcciones son: Calidad, Dirección de Tecnologías, Dirección de Comunicación Visual, Servicios Legales y Dirección de Informatización. Cada una de las facultades puede solicitar los servicios que brinda la Dirección de Servicios para el mejor desenvolvimiento de sus proyectos, que son, finalmente, la célula principal que da vida a la producción de software en la universidad.

Los proyectos productivos de la UCI son todos reales y el fondo de tiempo programado para los mismos crece progresivamente [11]. Además de que la organización del proceso docente se adecua a las necesidades de la producción, la cual se orienta hacia las demandas del mercado como: las soluciones integrales, los productos de ciclo completo, los servicios profesionales, las factorías de software, soporte y localización del mismo y una paulatina inserción al software libre.

La organización de los proyectos es ad hoc ya que cada negocio es un nuevo modelo de proyecto. Un proyecto productivo en la UCI está compuesto por personal de la UCI y una empresa productora de software. El equipo de proyecto se especializa por roles y es tutorado por expertos funcionales. Todos los esfuerzos son dirigidos y controlados por el líder del proyecto. Existen más de 200 proyectos a los cuales estaban incorporados más del 90% de los estudiantes de la universidad al cierre del curso 2005/2006.

Los programas de informatización se centran en los sectores fundamentales del país como la salud, la educación, el deporte y la cultura, el turismo, la prensa y el software libre, además de que se coordinan y desarrollan proyectos de exportación a través de la Empresa de Comercio Exterior, Sociedad Mercantil Cubana para la UCI (ALBET) y se resuelven necesidades de la ciudad universitaria.

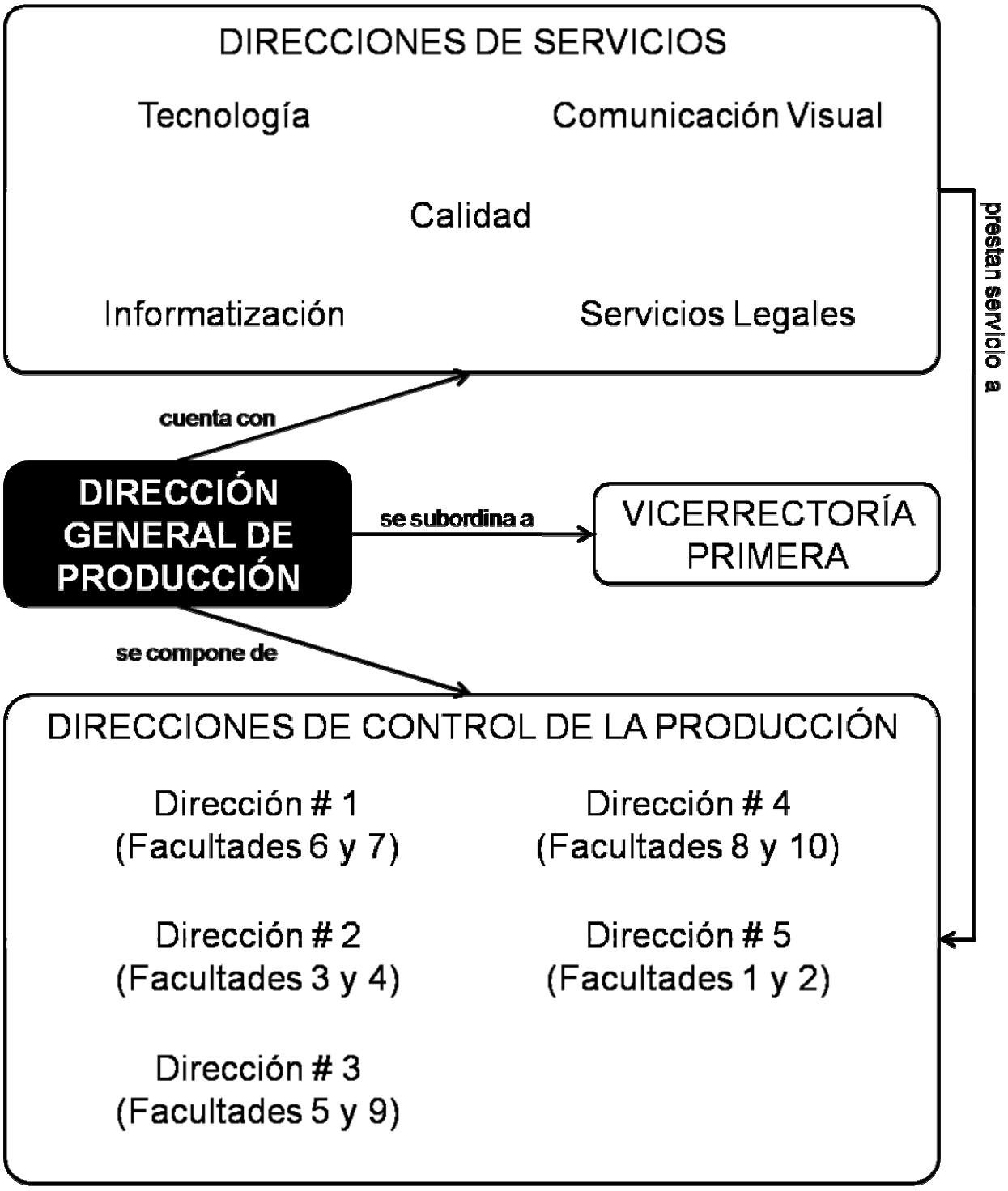


Fig 1. 3 Estructura de Dirección de la Infraestructura de Producción de la UCI.

Los Vicedecanatos de Producción de las facultades son los que dirigen, organizan y garantizan la producción en los proyectos que se le asignan de acuerdo a sus perfiles productivos.

En la actualidad los proyectos de software de gestión no se guían para su desarrollo por un proceso dictado a nivel central. Los resultados que se obtienen son fruto del esfuerzo de los Vicedecanatos de Producción de las facultades de forma individual, aunque para nada se niega que las actividades se realizan cumpliendo con los objetivos que persigue la UCI como institución y no se encuentran alejados de la Propuesta de Modelo de Procesos para la Producción de Software en la UCI que extiende la Dirección General de Producción en su versión 1.0 de noviembre de 2006 [11].

Este modelo de procesos propuesto para la producción de software en la UCI está conformado por tres categorías que a continuación se señalan.

Procesos de la Dirección UCI: Contiene el proceso de la Planeación Estratégica de la producción de software en la UCI, proporciona los lineamientos y políticas a seguir por las restantes categorías y se retroalimenta con la información generada por ellos, este proceso lo debe llevar a cabo un grupo directivo seleccionado para este fin en la UCI.

Procesos de la Dirección General de Producción: Contiene los procesos de Gestión de Proyectos, Gestión de Recursos y Gestión de Procesos, los ejecuta y establece según los lineamientos y políticas establecidas por el proceso de Planeación Estratégica. Además proporciona los elementos para el funcionamiento de los Polos Productivos, recibe y evalúa la información generada por estos y comunica los resultados a la Dirección UCI, los procesos de esta categoría deben ser ejecutados por la estructura de dirección de la producción en la UCI.

Procesos de los Polos Productivos: Contiene los procesos de Administración de los Proyectos Específicos y Desarrollo, Mantenimiento y Soporte de Software, realiza sus actividades según los elementos de funcionamiento entregados por la Dirección General de la Producción, entrega información a esta y los productos generados.

El marco general de esta propuesta de modelo está basado en el modelo mexicano Moprosoft, que ya fue presentado anteriormente en este capítulo, por su similitud con los procesos productivos que se han estado ejecutando actualmente en la UCI.

1.5 Los Polos Productivos.

La categoría de Polos Productivos de la UCI describe mejor el funcionamiento que deben seguir los proyectos de software de gestión de la universidad.

Su propósito es establecer y llevar a cabo sistemáticamente las actividades que permiten cumplir con los objetivos de un proyecto en tiempo y costo esperados, además la realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados, cumpliendo con los requerimientos especificados.

En la mencionada Propuesta de Modelo de Procesos para la Producción de Software en la UCI se describen los procesos por los cuales se deberán regir los proyectos de producción de la UCI; entre estos procesos se encuentran: el Proceso de Administración de Proyectos Específicos y el Proceso de Desarrollo, Mantenimiento y Soporte de Software. En ninguno de los dos procesos se hace mención a la utilización de algún tipo de métrica como parte importante del ciclo de vida de un proyecto de software de gestión. Aunque en pesquisas realizadas se encontró que los proyectos de nueva creación comienzan a poner sus esfuerzos en esta área de la gestión, aún es pobre el trabajo que se viene realizando con las métricas.

1.6 Análisis de los resultados de las encuestas aplicadas a los líderes y especialistas de calidad de algunos de los proyectos más significativos de software de gestión de la UCI.

Al ser los Polos Productivos la célula primaria en la IP de la UCI, se realizó una encuesta a los líderes y especialistas de calidad de los proyectos de software de gestión para conocer principalmente las características del funcionamiento,

metodologías, procesos, entre otras cuestiones de dichos proyectos y el estado del uso de mediciones al software. A continuación se discuten los resultados obtenidos.

Se escogieron 11 proyectos de desarrollo de software y calidad que se encuentran entre los más representativos de la UCI, que llevan entre 36 y 10 meses desarrollando software, con equipos de desarrollo que oscilan entre las 140 y 12 personas. De ellos, 9 son proyectos de software de gestión y se decidió incluir los otros 2 para verificar el uso de las métricas de software.

Estos proyectos utilizan en su totalidad la metodología de desarrollo de software Rational Unified Process (RUP) y algunos incluyen el uso de otras metodologías como son eXtreme Programming (XP), Microsoft Solutions Framework (MSF) y Dynamic Systems Development Method (DSDM). La filosofía de programación más usada es la Orientada a Objetos (OO) aunque se observa la inclusión en un menor grado de otras como las Orientadas a Servicios, a Aspectos y a Eventos. Todos los proyectos encuestados contestaron que se tiene en cuenta la calidad del software aunque sólo la mitad usa las métricas de software y entiende la importancia de estas como una forma de medir calidad.

Es importante recalcar que el uso de métricas de software está dado en su generalidad por iniciativa del propio proyecto, aunque todos acuerdan en la importancia de que también sean orientadas por un nivel superior. No obstante, las métricas que están implantando estos proyectos en la actualidad son:

- Métricas de calidad.
- Métricas para la captura de requerimientos.
- Métricas de productividad.
- Métricas para la revisión del diseño gráfico.
- Métricas para las pruebas.

- Métricas para determinar la complejidad y prioridad de los casos de uso.
- Métricas para medir el tiempo, el esfuerzo y la cantidad de personas.

Sobre la utilidad de estas métricas, contestaron que tienen una gran importancia porque miden la calidad del producto, el avance y la organización del proyecto, los errores cometidos, permiten predecir cuestiones como el tamaño y la duración y en general proveen de herramientas para la toma de decisiones acertadas.

Las encuestas arrojaron que los proyectos sienten la necesidad de hacer mediciones sobre:

- Cohesión y Acoplamiento.
- Estructura, Reusabilidad y Mantenibilidad.
- Parametrización.
- Captura de requerimientos.
- Tiempo de duración del proyecto y cumplimiento del plan (tiempo real vs. tiempo planificado).
- Calidad.

Otras cuestiones de interés señaladas son que aunque se ha avanzado en el tema de la gestión de la calidad, es necesario trabajar aún más. Sobre las métricas del software, los encuestados sugieren que se deben utilizar la mayor cantidad de métricas disponibles que sean aplicables y se deben conocer las principales dificultades de los proyectos para aplicar métricas objetivas y obtener resultados de utilidad aunque debe aplicarse el principio de no utilizar los resultados obtenidos en las mediciones contra las personas.

1.7 Situación problemática y objetivos propuestos.

Como se ha podido observar en el desarrollo de los epígrafes anteriores, existen dificultades en el uso de métricas de software, lo que trae consigo la siguiente situación problemática:

- Se hace un uso incipiente de las métricas de forma dispersa y no centralizada.
- No constituye un requerimiento obligatorio a pesar de su importancia y el valor que le proporciona al proyecto.
- No es un parámetro a partir del cual se tomen acciones concretas durante el proceso de desarrollo de software.
- No determinan un camino a seguir en un punto crítico o crucial del proceso, pudiendo ver a tiempo que algo va mal y tomar las medidas necesarias para corregir el error y un posible fallo del sistema.

A pesar de existir varios sistemas de métricas, probadas y aceptadas en cientos de instituciones y empresas prestigiosas del mundo con resultados importantes, en la UCI no se cuenta con un sistema de métricas adaptado a sus características que permita evaluar los proyectos de software de gestión.

Como se ha podido observar, las investigaciones realizadas alrededor de los proyectos de software de gestión de la UCI, así como del uso de las métricas en su desarrollo, reflejan la necesidad que tiene la universidad de que se implante de forma centralizada y estándar el uso de un sistema de métricas para evaluar los mismos, ya que no se aplican las mismas métricas y en muchos casos no se utilizan. La elección propia de métricas ayuda a los proyectos de forma individual a conocerse a sí mismos y a controlar la calidad, y es un principio aceptado, como ya se ha dicho, que se deben utilizar la mayor cantidad de métricas disponibles que sean aplicables, pero estas métricas no brindan a la estructura de dirección de la producción información que pueda

ser usada en el futuro por otros proyectos con semejantes características que se vayan a crear.

Por todo lo anteriormente expuesto, la presente tesis tiene como objetivo: Proponer un sistema de métricas que permita evaluar de una forma centralizada y estándar el avance de los proyectos productivos de gestión en la UCI.

Se propondrá un sistema adecuado a las características de los proyectos de software de gestión en la UCI a partir de un estudio de las métricas más actuales a nivel mundial para que se puedan aplicar de forma homogénea a todos los proyectos, cosa que facilitará la captura de los datos medidos. Aunque esto no quita que cada proyecto aplique métricas creadas por iniciativa propia y que se adapten mejor a las características internas del proyecto.

1.8 Conclusiones.

Se han estudiado los fundamentos teóricos relacionados con las métricas del software y a partir de un estudio del entorno de producción de la UCI, se detectaron los principales problemas que motivan el desarrollo de la presente investigación justificando el objetivo trazado.

CAPÍTULO 2. ESTUDIO DE LAS MÉTRICAS DEL SOFTWARE.

2.1 Introducción.

En este capítulo se hace un estudio de las métricas más usadas a nivel mundial, con resultados empíricos importantes en empresas y proyectos de gran prestigio nacional e internacional. Se hace una breve descripción de cada una y se especifican sus características fundamentales, así como sus ventajas y desventajas.

Para mejor comprensión de las mismas se organizaron según las P de Pressman, los flujos de trabajo de RUP: Requerimientos, Análisis y Diseño, Implementación y Pruebas y un epígrafe dedicado a las métricas de los sistemas OO. Finalmente, se hace referencia a tres modelos para medir Calidad del Software.

La decisión de organizar las métricas de esta forma se basó en los resultados que arrojaron las encuestas aplicadas a once proyectos de la UCI los cuales se analizaron en el capítulo anterior. Estos resultados mostraron fundamentalmente que los proyectos encuestados usan RUP como proceso de desarrollo de software y filosofía de programación OO. Además de que tienen en cuenta la Calidad de formas muy diversas. Fue preciso estructurar algunas métricas según las P de Pressman por ser estas entidades medibles dentro del desarrollo de software.

2.2 Métricas según las P de Pressman.

Las 4 P de Pressman son de vital importancia ya que estas ayudan a comprender mejor el proceso de desarrollo de software, las mismas se definen a continuación [8]:

Proceso: Un proceso de ingeniería de software es una definición del conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto. Un proceso es una plantilla para crear proyectos.

Proyecto: Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto. Es una instancia de Proceso.

Producto: Artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables y documentación.

Personas: Los principales autores de un proyecto software son los arquitectos, desarrolladores, ingenieros de prueba, y el personal de gestión que les da soporte, además de los usuarios, clientes y otros interesados. Las personas son realmente seres humanos, a diferencia del término abstracto trabajadores.

2.2.1 Métricas del Proceso y del Proyecto.

La recolección de métricas del proceso es esencial para la mejora del mismo, incluso en proyectos a pequeña escala. Se utilizan para evaluar la eficiencia de un proceso o si éste ha mejorado con los cambios realizados. Los indicadores del proceso permiten al gestor, evaluar lo que funciona y lo que no; y a la organización, tener una visión profunda de la eficacia de un proceso ya existente.

Por otro lado las métricas del proyecto son tácticas. Estas permiten que un gestor de proyectos adapte el enfoque a flujos de trabajo de proyectos técnicos en tiempo real. Se utilizan para minimizar la planificación, evitar retrasos y riesgos potenciales, y para evaluar la calidad de los productos y modificar el enfoque técnico que mejore la calidad.

Técnicamente no existe gran diferencia entre las métricas del proyecto y del proceso. Se pueden concebir las métricas del proceso como recopilaciones de métricas del proyecto. Entre ellas se tienen a las métricas de productividad, calidad y las técnicas. En este apartado solo se verá la primera, pues las métricas de calidad y las técnicas son explicadas en detalle en otros epígrafes.

2.2.1.1 Métricas orientadas al tamaño.

Las métricas orientadas al tamaño se usan para determinar en qué tiempo se va a finalizar el software y cuántas personas se van a necesitar para desarrollar el mismo. Son medidas directas al software y al proceso por el cual se desarrolla, si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño de la que se pueden extraer relaciones y razones muy interesantes y de utilidad. Estas métricas se relacionan con el tamaño de alguna salida de una actividad. Una de las medidas es las LDC entregadas. También se utiliza la medida del Punto de Función (PF).

Durante el desarrollo de esta tesis se podrá constatar que muchas de las métricas que se investigaron y que se proponen finalmente son métricas que miden el tamaño del software contando otros elementos del sistema. Se ha querido dar una explicación y ejemplificar las mismas a través de este epígrafe, así que no piense el lector que aquellas son distintas de estas. Solo se separan por pertenecer a sistemas de métricas independientes que proponen otros autores.

2.2.1.1.1 Líneas de Código.

A través del conteo de las LDC se puede medir tamaño, complejidad y productividad, entre otros atributos del software. Estas se aplican a proyectos convencionales donde se usa filosofía de programación estructurada. Aunque parece sencilla no lo es tanto, ya que existen diferentes alternativas: contar sólo las líneas de código ejecutable, añadirle las declaraciones de datos, también podemos contar las líneas de comentarios, añadir las sentencias de control, etc. De hecho suele ser frecuente, que al usar diferentes programas de métricas, estas generen diferentes valores de LDC para un mismo fichero.

El uso principal que tiene contar LDC, es poder determinar proporciones entre LDC y fallos, es decir, generar medidas sobre el número de LDC y el número de fallos de una aplicación.

Una alternativa a las LDC son los PF, que también intentan medir el tamaño del software. Este es bastante independiente del lenguaje de programación utilizado. Sacado de la Wikipedia: la técnica de medición del tamaño en punto-función consiste en asignar una cantidad de "puntos" a una aplicación informática según la complejidad de los datos que maneja y de los procesos que realiza sobre ellos. Como podemos ver, bastante compleja de calcular. Aquí también existen diferentes procedimientos para obtener esta métrica [12].

2.2.1.1.2 Puntos de Función.

La métrica Punto de Función, definida por Allan Albrecht, de IBM, en 1979, es un método para medir el tamaño del software. Pretende medir la funcionalidad entregada al usuario independientemente de la tecnología utilizada para la construcción y explotación del software, y también ser útil en cualquiera de las fases de vida del software, desde el diseño inicial hasta la explotación y mantenimiento [13].

La métrica de PF se puede usar como medio para predecir el tamaño de un sistema que se va a obtener de un modelo de análisis. Para visualizar esta métrica se utiliza un diagrama de flujo de datos, del cual se determinarán las siguientes medidas claves necesarias para el cálculo de la misma [14]: Número de entradas del usuario, Número de salidas del usuario, Número de peticiones de usuario, Número de archivos y Número de interfaces externas.

Una vez que se han recopilado los datos anteriores, a la cuenta se asocia un valor de complejidad. Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada en particular es simple, media o compleja. No obstante la determinación de las complejidades es algo subjetiva.

2.2.1.2 COCOMO II.

El Modelo Constructivo de Coste (COCOMO por sus siglas en inglés), propuesto y desarrollado por Barry Boehm en 1981, es uno de los modelos de estimación de costos

mejor documentados y utilizados [15]. El modelo permite determinar el esfuerzo y tiempo que se requiere en un proyecto de software a partir de una medida del tamaño del mismo expresada en el número de LDC que se estimen generar para la creación del producto software.

Debido a la complejidad de los proyectos de software, el modelo original fue modificado en 1990 por el SEI, denominándose, al modelo actual COCOMO II. El nuevo modelo permite determinar el esfuerzo y tiempo de un proyecto de software a partir de las medidas de tamaño PF sin ajustar y Puntos por Objetivos; lo cual supone una gran ventaja, dado que en la mayoría de los casos es difícil determinar el número de LDC de que constará un nuevo desarrollo, en especial cuando se tiene poca o ninguna experiencia previa en proyectos de software. Aunque su nombre indica que usa técnicas de estimación, es necesario destacar que el mismo puede ser aplicado antes, durante y después del desarrollo del software.

2.2.1.3 Métricas de Productividad.

Las métricas de productividad se centran en el rendimiento del proceso de la ingeniería del software. Los enfoques de LDC y PF se usan para hacer predicciones relativamente exactas, aunque para ello se requiere una línea base de información histórica. De forma análoga, el resultado de las métricas Número de Clases Claves y Número de Métodos de Instancia, entre otras que proponen Lorenz y Kidd, pueden también utilizarse para calcular productividad, como puede ser el número de clases claves promedio por desarrollador o el promedio de métodos por persona/mes.

De forma general, la productividad siempre será calculable si se toma como base cualquier medida de tamaño que se encuentre.

2.2.1.4 Eficacia en la Eliminación de Defectos.

Eficacia en la Eliminación de Defectos (EED) es una métrica de la calidad que proporciona beneficios tanto a nivel del proyecto como del proceso. En esencia es una

medida de la habilidad de filtrar las actividades de la garantía de la calidad y de control al aplicarse a todas las actividades del marco de trabajo del proceso, esto anima a que el equipo del proyecto de software instituya técnicas para encontrar todos los errores posibles antes de su entrega. EED también se puede utilizar dentro del proyecto para evaluar la habilidad de un equipo en encontrar errores antes de que pasen a la siguiente actividad estructural o tarea de ingeniería del software. Por ejemplo, la tarea del análisis de los requisitos produce un modelo de análisis que se puede revisar para encontrar y corregir errores. Esos errores que no se encuentren durante la revisión del modelo de análisis se pasan a la tarea de diseño (en donde se pueden encontrar o no) [14].

2.2.2 Métricas del Producto.

Los productos están compuestos por artefactos, los cuales pueden ser documentos, modelos, módulos, o componentes, por tanto, las métricas del producto deben hacerse sobre la base de medir cada uno de los artefactos. Las métricas del producto describen características como el tamaño, la complejidad, los rasgos del diseño, el rendimiento y el nivel de calidad [16].

Hay varias razones para medir un producto:

1. Para indicar la calidad del producto.

2. Para evaluar la productividad de los que desarrollan el producto.

3. Para evaluar los beneficios en términos de productividad y de calidad, derivados del uso de nuevos métodos y herramientas de la ingeniería de software.

4. Para establecer una línea de base para la estimación.

5. Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

Existen dos tipos de métricas del producto: las dinámicas y las estáticas. Las primeras son aquellas que son recogidas por las mediciones hechas en un programa en

ejecución en relación directa con los atributos de calidad del software, los cuales son explicados más adelante. Se pueden mencionar la medición de tiempo de ejecución como medida de la eficiencia del sistema y el registro del número y tipo de caídas del sistema y su relación con la fiabilidad del software.

Las estáticas son aquellas que son recogidas por las mediciones hechas en las representaciones del sistema como el diseño, el código y la documentación. Tienen relación directa con los atributos de calidad del software. Se pueden mencionar la longitud del programa como predictor de la mantenibilidad o la complejidad y el índice de Fog como predictor de la facilidad de comprensión de un documento de texto [17].

En general las características a medir de los artefactos del producto son las que se presentan en los siguientes epígrafes.

2.2.2.1 Métricas de Complejidad.

Se puede medir la complejidad de una estructura o un algoritmo. Mientras mayor sea la complejidad y más difícil sea de comprender y modificar la estructura del sistema, mayor probabilidad habrá de que falle. En algunas ocasiones, son utilizadas métricas de tamaño para evaluar la complejidad, pero es recomendable hacer uso de otro tipo de métricas [18].

2.2.2.1.1 Complejidad ciclomática o método de McCabe.

La complejidad ciclomática es un método reconocido para medir la complejidad de métodos o algoritmos desarrollado por Tomas McCabe. Este método mide el número de decisiones lógicas en un segmento de código [19].

2.2.2.1.2 fan-in/fan-out.

fan-in se refiere a la medida del número de funciones que llaman a otra función X y fan-out al número de funciones que son llamadas por otra función X [17].

Un valor alto de fan-in indica que X está fuertemente acoplada al resto del diseño y que los cambios en X pueden tener efectos extensivos al resto del sistema. Un valor alto de fan-out indica que la complejidad de X podría ser excesiva, debido a la complejidad de la lógica de control necesaria para coordinar a los componentes llamados.

2.2.2.2 Índice de Fog.

El índice de Fog es un método para ver la dificultad de lectura de un texto determinado. El número resultante indica el número de años que una persona debe educarse para que entienda fácilmente un texto a la primera lectura. Esto significa que si el índice de Fog es doce, el texto puede ser leído por estudiantes de secundaria. El test fue desarrollado por Robert Gunning, un hombre de negocios americano, en 1952.

2.2.3 Métricas de Personas. Procesos de Software Personal y en Equipo.

El Proceso de Software Personal (PSP por sus siglas en inglés) fue diseñado para ayudar a los ingenieros del software a hacer bien su trabajo. Muestra cómo aplicar métodos avanzados de ingeniería a sus tareas diarias. Proporcionan métodos detallados de planificación y estimación, muestra a los ingenieros cómo controlar su rendimiento frente a estos planes y explica cómo los procesos definidos guían su trabajo [20].

El Proceso de Software en Equipo (TSP por sus siglas en inglés) junto al PSP ayuda a elevar el rendimiento de los ingenieros para asegurar la calidad de los productos de software, crear productos de software seguros y mejorar los procesos de desarrollo en la organización [21]. Los grupos de ingenieros usan TSP para aplicar conceptos integrados de equipo para el desarrollo de sistemas. El proceso de lanzamiento en cuatro días lleva al equipo y sus líderes, en primer lugar, al establecimiento de objetivos del proyecto, luego se definen los roles del equipo y se valoran los riesgos, por último se crea un plan de producción del equipo. TSP puede ayudar a la organización a establecer una práctica madura y disciplinada que produzca un software seguro. También es usado como base para un nuevo marco de trabajo de medidas tanto para

desarrolladores como clientes de un software. Estas métricas constituyen el actual proyecto: Métricas para la Adquisición de Proyectos Integrados (ISAM por sus siglas en inglés) cuyos estudios pilotos de desarrollo se llevan a cabo por un equipo de la Carnegie Mellon University liderado por Anita Carlenton [22].

2.3 Métricas según los flujos de trabajo de RUP.

Según Ivar Jacobson, Grady Booch y James Rumbaugh, RUP es un proceso de desarrollo de software que se define como el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software, sin embargo el proceso unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos.

Este proceso está constituido por varias fases por las cuales el software debe atravesar durante su desarrollo: Inicio, Elaboración, Construcción y Transición. En cada una de estas fases se llevan a cabo una serie de flujos de trabajo como son Modelación del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba, Instalación, Administración del Proyecto, Administración de Configuración y Cambios y Ambiente.

Durante el estudio que a continuación se presenta, se hizo una búsqueda de las métricas de cada uno de los Flujos de Trabajo de RUP encontrándose resultados de peso en los flujos: Requerimientos, Análisis y Diseño (separado por modelos para mejor comprensión), Implementación y Pruebas.

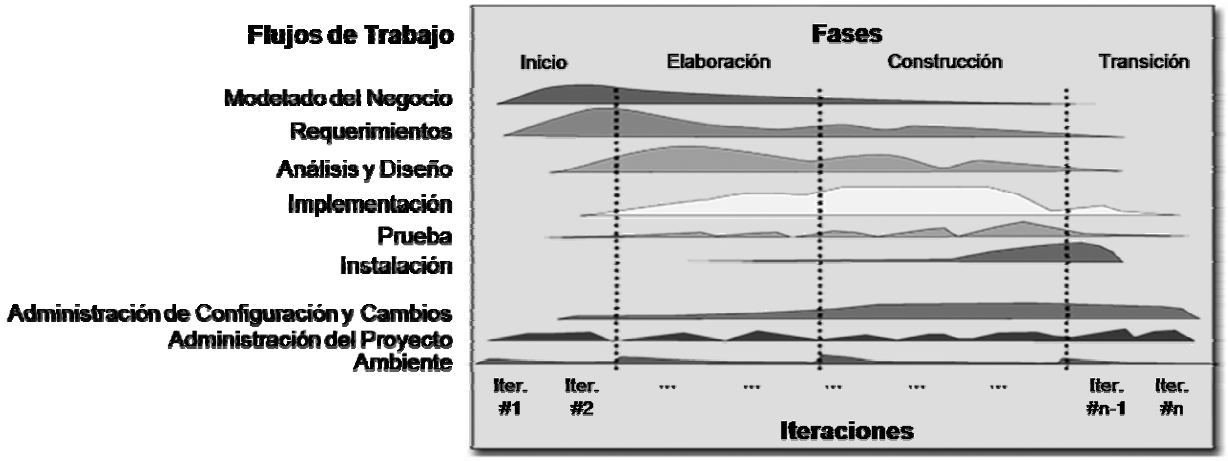


Fig 2. 1 Fases y Flujos de Trabajo que propone RUP.

2.3.1 Métricas de Requerimientos.

Davis y sus colegas proponen una lista de características que pueden emplearse para valorar la calidad del Modelo de Análisis y la correspondiente Especificación de Requisitos: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación, exactitud y capacidad de reutilización. Aunque muchas de las características anteriores parecen ser de naturaleza cualitativa, Davis sugiere que todas pueden representarse usando una o más métricas [14]. Pressman solo describe tres de estas métricas:

- Especificidad de los requisitos.
- Compleción de los requisitos funcionales.
- Grado de validación de los requisitos.

2.3.2 Métricas del Modelo de Análisis.

En esta fase es deseable que las métricas proporcionen una visión interna de la calidad del Modelo de Análisis. Estas métricas examinan el Modelo de Análisis con la intención

de predecir el “tamaño” del sistema resultante ya que el tamaño y la complejidad del diseño están directamente relacionados.

2.3.2.1 Métricas Bang.

Mide el tamaño del software a implementar como consecuencia del modelo de análisis. El propósito es evaluar el conjunto de primitivas, es decir, los elementos más bajos del análisis que no se pueden subdividir más.

2.3.2.2 Métricas de Genero.

Genero y otros autores propusieron en el año 2000 un conjunto de métricas para la medida de la complejidad estructural de los modelos de clase debido al uso de relaciones del Lenguaje Unificado de Modelado (UML por sus siglas en inglés).

Se clasifican en métricas a nivel de Modelos de Clases y Métricas a Nivel de Clases. Estas métricas se enfocan hacia la complejidad estructural debido al uso de las relaciones y es objetiva. Se han validado teórica y empíricamente.

2.3.3 Métricas del Modelo de Diseño.

Las métricas del diseño proporcionan al diseñador una mejor visión interna y ayudan a que el diseño evolucione a un nivel superior de calidad. La gran ironía de las métricas de diseño para el software es que las mismas están disponibles, pero la gran mayoría de los desarrolladores de software continúan sin saber que existen. No son perfectas y continúa el debate sobre su eficacia y como deberían aplicarse. Algunos expertos señalan que es necesaria mayor experimentación hasta que se puedan emplear adecuadamente. Sin embargo el diseño sin medición es una alternativa inaceptable [23]. Así que hay que seguir tratando hasta que se encuentren las métricas adecuadas.

2.3.3.1 Métricas de diseño de alto nivel o arquitectónicas.

Se concentran en las características de la arquitectura del programa, con énfasis en la estructura arquitectónica y en la eficiencia de los módulos.

Estas métricas son de caja negra en el sentido que no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema. Ayudan a localizar los módulos más complejos y, por lo tanto, aquellos en los que debemos poner especial atención. También son utilizadas para saber el número de módulos asignados a cada trabajador.

2.3.3.2 Métricas del diseño de interfaz.

Aunque existe una significativa cantidad de literatura sobre el diseño de interfaces hombre-máquina, se ha publicado relativamente poca información sobre métricas que proporcionen una visión interna de la calidad y facilidad de empleo de la interfaz.

La métrica que se sugiere es la Conveniencia de la Representación (CR) como una valiosa métrica de diseño para interfaces hombre-máquina.

2.3.3.3 Métricas de Fenton.

Sugiere varias métricas de morfología simples que permiten comparar diferentes arquitecturas mediante un conjunto de dimensiones directas. Estas métricas son:

- Tamaño.
- Profundidad.
- Amplitud.
- Relación arco a nodo.

2.3.4 Métricas para el Flujo de Trabajo de Implementación.

Entre las Métricas para el Flujo de Trabajo de Implementación se puede encontrar las Métricas del Código Fuente creadas por Halstead quien utiliza un conjunto de medidas primitivas que pueden obtenerse una vez que se han generado o estimado el código después de completar el diseño. Halstead usa las medidas primitivas para desarrollar

expresiones para la longitud global del programa; volumen mínimo potencial para un algoritmo; el volumen real (número de bits requeridos para especificar un programa); el nivel del programa (una medida de la complejidad del software); nivel del lenguaje (una constante para un lenguaje dado); y otras características tales como esfuerzo de desarrollo, tiempo de desarrollo e incluso el número esperado de fallos en el software.

2.3.5 Métricas para el Flujo de Trabajo de Prueba.

La mayoría de las métricas propuestas para las pruebas no se centran en el proceso de prueba ni en las características técnicas de las mismas [14]. En general, los responsables de las pruebas deben fiarse de las métricas de análisis, diseño y código para que les guíen en el diseño y ejecución de los casos de prueba.

Las métricas de la prueba desembocan en las siguientes categorías:

1. métricas que ayudan a determinar el número de pruebas requeridas en los distintos niveles de la prueba;

2. métricas para cubrir la prueba de un componente dado.

A medida que se van haciendo las pruebas, tres medidas diferentes proporcionan una indicación de la compleción de las pruebas:

1. la amplitud de las pruebas, que proporciona una indicación de cuántos requisitos se han probado;

2. la profundidad de las pruebas, que es una medida del porcentaje de los caminos básicos independientes probados en relación al número total de estos caminos en el programa y

3. los perfiles de fallos: a medida que se va haciendo las pruebas y que se recogen los datos de los errores, se emplean estos perfiles, para dar prioridad y categorizar los errores descubiertos.

2.4 Métricas técnicas para sistemas Orientados a Objetos.

La Programación Orientada a Objetos (POO) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos), comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La POO expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos) [24].

Con la explosión producida por la entrada de la orientación a objetos se produjo una explosión de métricas orientadas a objetos. En aquella época se pensó que estas podían ser una solución definitiva al problema de medir software de estas características, pero solo se logró que afloraran múltiples criterios divergentes, como se ha reflejado en la actualidad; momento en el que aún no se llega a un consenso sobre el tema. Entre las métricas que vieron la luz en los años noventa para sistemas OO, pero que todavía están vigentes se encuentran las que a continuación se exponen.

2.4.1 Métricas propuestas por Lorenz y Kidd, 1994.

Lorenz y Kidd proponen métricas basadas en clases - unidad dominante en los sistemas OO - separadas en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos, aunque también propusieron otras tres simples métricas orientadas al tamaño y la complejidad. En el caso de la métrica de tamaño se incluirá

en el apartado de Métricas Orientadas al Tamaño y se creará otro para agrupar las métricas de complejidad. Esto se hace para contribuir a la organización de las métricas y su comprensión.

Las métricas Tamaño medio de operación, Complejidad de Operación y Número de Parámetros de Media por Operación que propusieron también Lorenz y Kidd examinan las características promedio para los métodos u operaciones pese a que se han propuesto menos métricas para operaciones que las relacionadas con las clases.

Al respecto Churcher y Shepperd declaran que: “Resultados de estudios recientes indican que los métodos tienden a ser pequeños, tanto en términos de número de sentencias como en complejidad lógica, sugiriendo que la estructura de conectividad de un sistema debe ser más importante que el contenido de los módulos individuales.”

De forma general, las métricas que proponen Lorenz y Kidd son:

Dentro de las Métricas orientadas al tamaño para las clases OO que se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo están:

- Número de Métodos de Instancia Públicos (PIM).
- Número de Métodos de Instancia (NIM).
- Número de Variables de Instancia (NIV).
- Tamaño de clase (TC).
- Tamaño medio de operación (TMO).
- Número de escenarios (NE).
- Número de clases clave (NCC).
- Número de subsistemas (NSUB).

Las métricas NE, NCC y NSUB pueden recolectarse sobre proyectos OO pasados, y están relacionadas con el esfuerzo invertido en el proyecto como un todo. Estos datos pueden también utilizarse junto con métricas de diseño, para calcular “métricas de productividad” tales como el número de clases promedio por desarrollador o promedio de métodos por persona/mes. Colectivamente, estas métricas pueden usarse para estimar el esfuerzo, duración, personal y otra información para el proyecto actual.

Entre las métricas basadas en la herencia que se centran en la forma en que las operaciones se reutilizan en la jerarquía de clases están:

- Número de operaciones redefinidas para una subclase (NOR).
- Número de operaciones añadidas por una subclase (NOA).
- Índice de especialización (IES).
- Número de Métodos Heredados (NMI).

Entre las métricas para valores internos de clase que examinan la cohesión y los aspectos orientados al código está [25]:

- Promedio de Parámetros por Método (APPM):

Por último se encuentran las métricas orientadas a valores externos que examinan el acoplamiento y la reutilización.

Por último se creó el siguiente apartado que recoge dos métricas para la complejidad.

- Complejidad de operación (CO).
- Número medio de parámetros por operación (NMP).

2.4.2 Las series de métricas de Chidamber y Kemerer, 1994.

Uno de los conjuntos de métricas OO más ampliamente referenciados ha sido el propuesto por Chidamber y Kemerer (CK), también llamadas MOOSE, los cuales adoptan tres criterios a la hora de definir las métricas: capacidad de satisfacer propiedades analíticas, aspecto intuitivo a los profesionales y gestores y facilidad para su recogida automática [26]. Normalmente son conocidas como la serie de métricas CK**:**

- Métodos ponderados por clase (MPC).
- Profundidad de árbol de herencia (PAH).
- Número de hijos (NDH).
- Acoplamiento entre clases objeto (AEC).
- Respuesta para una clase (RPC).
- Carencia de cohesión en los métodos (CCM).

2.4.3 Li y Henry, 1993.

Las métricas que proponen Li y Henry son una extensión de MOOSE con otras métricas de acoplamiento, tamaño y diseño y un sistema de predicción para la reusabilidad y mantenibilidad, es decir, modificaron y ampliaron las métricas CK además de añadir cinco nuevas métricas.

Esta nueva suite de métricas tiene un objetivo claro, que es la medición y mejora de la mantenibilidad del software OO, y para ello se añaden la abstracción y el tamaño de diseño (aunque una de las dos métricas propuestas para él se basa en las líneas de código). Se da pues un cierto avance conceptual al incluir de manera explícita un atributo de calidad de software como objetivo de las métricas [10].

Las métricas tomadas de CK son:

- Métodos ponderados por clase.
- Árbol de profundidad de herencia (PAH).
- Número de descendientes (NDH).
- Respuesta para una clase.
- Carencia de cohesión en los métodos.

El acoplamiento por herencia lo miden a través de NDH y PAH.

Además definen otras cinco métricas de acoplamiento y tamaño. Estas son:

- Acoplamiento por paso de mensaje (APM).
- Acoplamiento por abstracción de datos (AAD).
- Número de métodos locales (NML).
- Tamaño 1.
- Tamaño 2.

Las métricas de tamaño miden aspectos de la complejidad de una clase.

2.4.4 La colección de métricas MDOO.

Harrison, Counseil y Nithi propusieron un conjunto de Métricas para el Diseño Orientado a Objetos (MDOO), que proporcionan indicadores cuantitativos para el diseño de características OO. Entre las métricas MDOO se encuentran:

- Factor de herencia de métodos.

- Factor de acoplamiento.
- Factor de polimorfismo.

2.4.5 MOOD de Abreu y Melo, 1994.

Abreu y Melo presentaron las métricas Metrics for Object Oriented Design (MOOD) para medir algunos de los principales mecanismos de los modelos OO (encapsulamiento, polimorfismo, herencia y paso de mensajes,...) para poder evaluar la productividad del desarrollo y la calidad del producto. Están encuadradas dentro de lo que se conoce como métricas a nivel de sistema.

Las métricas de Abreu y Melo se enfocan hacia las características de los diagramas de clase, son medidas objetivas y han sido validadas de forma teórica y parcialmente de forma empírica [25]. Se pueden utilizar en la fase de diseño. El proyecto MOOD define seis métricas cuyas principales características son: la fuerte correlación con los conceptos de OO, la evaluación global que hacen del sistema y la independiente que hacen de la implementación, además de que expresa sus resultados en por ciento [27]. Estas métricas son:

- Factor de ocultamiento de los métodos (FOM).
- Factor de ocultamiento de los atributos (FOA).
- Factor de herencia de los métodos (FHM).
- Factor de herencia de los atributos (FHA).
- Factor de polimorfismo (FPO).
- Factor de acoplamiento (FA).

2.5 Modelos de Calidad.

Las métricas de calidad proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente. Es decir, cómo se va a medir para que el sistema se adapte a los requisitos que pide el cliente.

Según la ISO 9000, calidad es el grado en que un conjunto de características inherentes cumplen con los requisitos.

Desde el punto de vista de la producción, calidad es el grado en que un producto cumple las especificaciones de diseño, es conformidad con las especificaciones.

Desde el punto de vista del usuario, calidad se define como la capacidad de satisfacer los deseos de los consumidores, es decir, depende de cómo el producto responda a las preferencias de los clientes y de cómo se adecue al uso [28].

La calidad del producto del software se debe evaluar usando un modelo de calidad definido. El modelo de calidad se usará al fijar los objetivos de calidad para los productos del software y productos de software intermedios [29].

A continuación se plantea el estudio de tres modelos de calidad importantes para evaluar la calidad del software.

2.5.1 Modelo de Calidad ISO IEC 9126.

La ISO/IEC 9126(1991) “Software product evaluation – Quality characteristics and guidelines for their use”, fue elaborada para evaluar la calidad de los productos de software que hoy en día se desarrollan en el mundo. Describe un modelo en dos partes para la calidad de los productos de software: la calidad interna y externa, y la calidad durante el uso.

La primera parte del modelo especifica seis características: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad; que son, a su vez, divididas en sub-características.

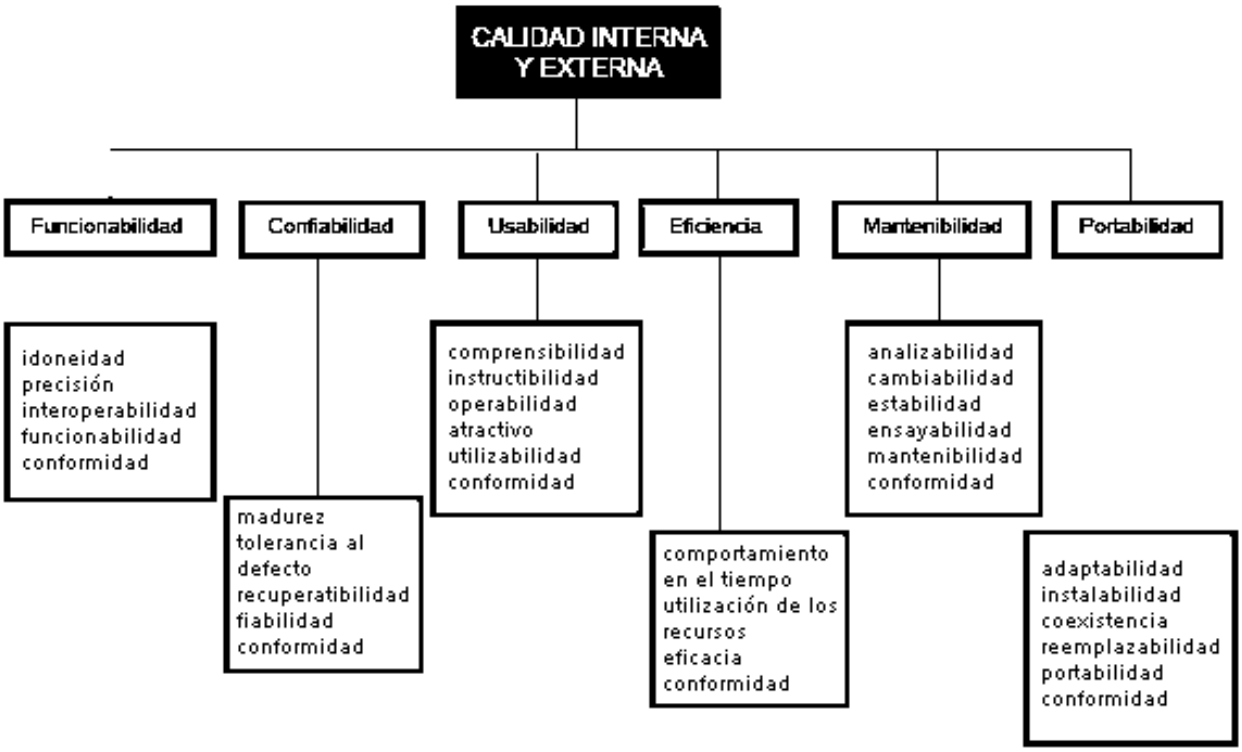


Fig 2. 2 Factores de Calidad Externos e Internos de la Norma ISO IEC 9126.

La segunda parte del modelo especifica cuatro características de calidad durante el uso, que para el usuario no es más que el efecto combinado de las seis características anteriormente mencionadas, estas son: eficacia, productividad, seguridad y satisfacción.

Fig 2. 3 Factores de Calidad Durante el Uso de la Norma ISO IEC 9126.

Este modelo puede ser usado por los programadores, los clientes, el personal de aseguramiento de la calidad y los evaluadores independientes, particularmente los responsables de especificar y evaluar la calidad de los productos de software. Puede usarse para validar la integridad de la definición de los requisitos, identificar los requisitos del software, los objetivos del diseño del software, los criterios de aseguramiento de la calidad y los criterios de aceptación para un producto de software terminado.

2.5.2 Modelo de Calidad de McCall.

Hace 25 años, McCall y Cavano definieron un juego de once factores de calidad como los primeros pasos hacia el desarrollo de métricas de la calidad del software [14]. El modelo de McCall organiza los factores en tres ejes o puntos de vista desde los cuales el usuario puede contemplar la calidad de un producto [30]:

- Operación del producto.
- Revisión del producto.
- Transición del producto.

Cada uno de estos factores se descompone, a su vez, en criterios.

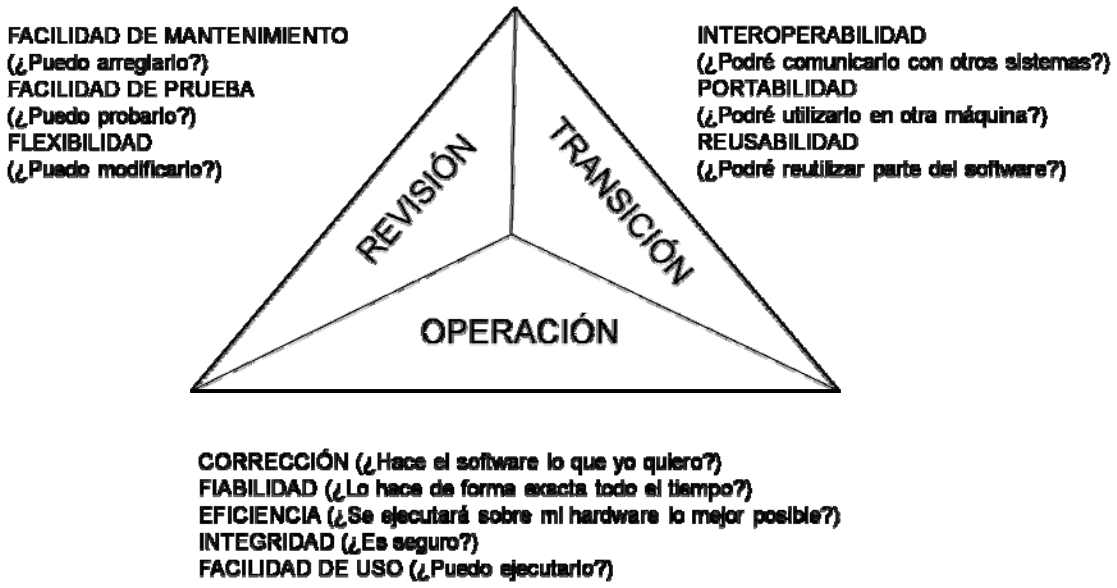


Fig 2. 4 Factores de Calidad propuestos por McCall.

2.5.3 Modelo de Calidad FURPS.

Los factores de calidad descritos por McCall y sus colegas representan solo una de las muchas listas de comprobación sugeridas para la calidad del software. Hewlett – Packard ha desarrollado un conjunto de factores de calidad del software al que se le ha dado el acrónimo de FURPS: funcionalidad, facilidad de uso, fiabilidad, rendimiento y capacidad de soporte [14]. Los factores de calidad FURPS provienen de trabajos anteriores y definen atributos para cada uno de los cinco factores principales. Estos factores y sus atributos pueden usarse para establecer métricas de la calidad para todas las actividades del proceso de software.

2.6 Conclusiones.

En este capítulo se hizo un estudio de las métricas más usadas a nivel mundial, con resultados empíricos importantes en empresas y proyectos de gran prestigio nacional e internacional. Se hizo una breve descripción de cada una y se especificaron sus características fundamentales, así como sus ventajas y desventajas.

CAPÍTULO 3. ELECCIÓN DE UN SISTEMA DE MÉTRICAS PARA LA UCI.

3.1 Introducción.

En este capítulo se hace un análisis crítico de las métricas estudiadas en el capítulo 2 para conformar el sistema de métricas a proponer. Se explica por qué se excluyeron algunas de las métricas encontradas durante el estudio y se analizan los principales sistemas OO. Finalmente se hace la propuesta formal del Sistema de Métricas para la Evaluación de Proyectos de Software de Gestión de la UCI.

3.2 Métricas excluidas.

Los criterios principales que se tuvieron en cuenta para excluir algunas de las métricas que se vieron en el estudio del capítulo 2 fueron, en primer lugar, extraídos de los resultados de las encuestas realizadas a algunos de los proyectos representativos de software de gestión de la UCI, por ejemplo, no tiene sentido hablar de métricas que no respondan a las características de los proyectos que utilizan filosofía OO; y en segundo lugar se tuvieron muy en cuenta las características principales que deben tener las métricas las cuales fueron expuestas con claridad en el epígrafe 1.2. Esto se debió a que no es objetivo de los autores que los gestores que utilizarán en un futuro este sistema como un primer acercamiento al uso de las métricas, no sufran por la complejidad del cálculo y el entendimiento de las mismas ni se tarden en obtener resultados concretos.

3.2.1 Líneas de Código.

La definición de LDC, aunque es básica para muchas métricas del software, es ambigua. El significado de LDC varía de un lenguaje a otro, pero también dentro de un mismo lenguaje de programación.

En el lenguaje de programación C, por ejemplo, una LDC puede ser (1) una instrucción acabada en un salto de línea, (2) una instrucción acabada en un punto y coma o (3) cualquier línea del programa que acabe en un salto de línea (comentarios incluidos).

Por ejemplo:

for (i=0; i<100; ++i) {printf("hola");} /* ¿Cuántas líneas tiene este programa? */ [31]

Frente a las actuales filosofías de programación OO, entre otras, estas métricas quedan obsoletas ya que el conteo de LDC se vuelve engorroso por la longitud de los programas que se implementan. Como una solución alternativa en la actualidad para proyectos OO se usa la métrica basada en PF.

3.2.2 Índice de Fog.

En la descripción de las métricas del producto se mencionó como una de las métricas de tipo estática el índice de Fog como predictor de la facilidad de comprensión de un documento de texto. Se decía que las métricas estáticas miden representaciones del sistema como el diseño, el código y la documentación. Por tanto, el interés de esta métrica para los desarrolladores cae en que la misma ayuda a discernir si la documentación del software que se desarrolla es fácil de comprender.

Debido a que esta métrica no es esencial para el buen desenvolvimiento del desarrollo del proyecto, se decidió que no formara parte de la Propuesta del Sistema de Métricas para Medir Proyectos de Software de Gestión de la UCI.

3.2.3 Métricas Bang.

Las métricas Bang, creadas en 1982, calculan el tamaño de un software a partir del Modelo de Análisis. Esta medición puede ser hecha con mayor precisión y confiabilidad con las métricas de PF como bien se explica en el capítulo 2 y resultan de uso más actual. Además, la primitividad de los datos que son necesarios para calcular esta

métrica hace de su recolección una tarea pesada y engorrosa para el desarrollador que pretenda usarla.

Por estas razones es que se decidió no agregarla a la Propuesta del Sistema de Métricas para los Proyectos Productivos de Gestión de la UCI.

3.2.4 Métricas de Genero.

Durante el estudio de este set de métricas se pudo constatar que las mismas tributan a otros sistemas más afianzados como por ejemplo el que propone Chidamber y Kemerer. Esto se debe a que estas métricas se limitan casi exclusivamente a recolectar datos. No se especifica en qué medida estos datos deben ser interpretados para que el desarrollador que las use pueda sacarles provecho.

3.2.5 Diseño de interfaz.

Sobre el diseño de interfaz existe muy poca bibliografía. Se pudo encontrar durante la investigación la métrica: Conveniencia de la Representación para el diseño de interfaces hombre – máquina.

Si bien es cierto que las Interfaces Gráficas de Usuario (IGU) representan un requisito indispensable para que el usuario se sienta satisfecho con el software que se le desarrolla, los autores no ven la necesidad de que esta sea medida por una métrica, en primer lugar porque la fórmula que se propone para ser medida necesita de la recopilación de datos muy específicos y engorrosos de recoger como son el conteo de las cuadrículas en las que se divide la pantalla y la cantidad de entidades de representación que se ubican en dichas cuadrículas y en segundo lugar porque utiliza factoriales en su cálculo. Esto contradice una de las características que propone RUP que debe tener una métrica: las mismas deben ser simples y fáciles de calcular.

Se prefiere tomar la opinión de Nielsen Levy cuando informa que puede haber una posibilidad de éxito si se prefiere la interfaz basándose exclusivamente en la opinión del

usuario ya que el rendimiento medio de tareas de usuario y su satisfacción con la IGU están altamente relacionados. Se propone que se haga un fuerte uso de los prototipos.

3.2.6 Métrica de Profundidad de Fenton.

La métrica Profundidad que propone Fenton de morfología simple, que permite comparar diferentes arquitecturas y la métrica Profundidad de Árbol de Herencia que propone CK calculan el camino más largo desde el nodo raíz a un nodo hoja. Se escogió la métrica propuesta por CK por ser parte de un sistema de métricas probado ampliamente y que en su descripción se llega a un análisis más profundo del resultado del indicador que se obtiene al calcularla.

3.2.7 Halstead.

A pesar de que la teoría de la ciencia del software propuesta por Halstead es probablemente la medida de complejidad mejor conocida y minuciosamente estudiada se consideró que no debe ser incluida en la propuesta porque las medidas primitivas que utiliza para calcular las características son demasiado específicas para un primer encuentro con desarrolladores que utilizarán las métricas en sentido general dentro del ciclo de vida de un proyecto. Se propone que puedan ser usadas a gusto del desarrollador cuando se haya creado en él una cultura acerca de la importancia de las métricas en la determinación de la calidad del software. Además, las diferentes características que mide pueden ser examinadas con los indicadores que proporcionan otras métricas que sí están incluidas en la propuesta de métricas, como por ejemplo la longitud y complejidad del software. Igualmente mide otras características como el número de bits requeridos para especificar un programa, que no resulta ser una métrica imprescindible para el buen diseño de un software, aunque en alguna medida tenga que ver con los recursos en términos de tamaño de memoria que va a usar el mismo.

Además, las fórmulas que propone para dar un indicador de la métrica usan logaritmos, que si bien nos habla de la cientificidad de la misma y la hace más confiable, no es menos cierto que los desarrolladores hacen un rechazo a su uso.

3.3 Análisis de Sistemas de Métricas para proyectos OO.

Ya se ha hablado de la importancia que tiene en la actualidad el uso de la filosofía OO. A continuación se hace un análisis de los Sistemas de Métricas para proyectos OO cuyo propósito fundamental es tomar, de entre los múltiples sets de métricas de este tipo que existen, aquellas que con mayor calidad midan la calidad de los proyectos de software de gestión y además evitar la repetición de aquellas métricas, que a pesar de encontrarse en sistemas diferentes, miden el mismo atributo o característica del software.

3.3.1 Sistema de métricas propuesto por Lorenz y Kidd.

De este sistema se excluye la métrica Complejidad de Operación ya que esta puede ser calculada usando cualquiera de las métricas de complejidad propuestas para el software convencional (véase métricas de diseño de alto nivel).

Las métricas Número de Operaciones Añadidas por una Subclase y el Índice de Especialización miden el mismo atributo: especialización de la subclase, por lo que se dejará esta medida bajo el nombre de Índice de Especialización.

3.3.2 Sistema de métricas propuesto por Li y Henry.

Las métricas Métodos Ponderados por Clase, Árbol de Profundidad de Herencia, Número de Descendientes y Respuesta para una Clase, como son tomadas de las CK no existe ninguna dificultad en dejarlas como parte de ese sistema y no describirlas como parte del sistema propuesto por Li y Henry.

La métrica Carencia en la Cohesión de los Métodos fue modificada por Li y Henry ya que consideraron que la definición original dada por Chidamber y Kemerer es ambigua y permite que dos clases con distintos grados de cohesión real obtengan idénticos valores de CCM [32]. Sin embargo, no será propuesta ya que no se pudo encontrar la métrica definida por Li y Henry en ninguna bibliografía ni en Internet.

El profesor E. Manso de la Universidad de Valladolid, después de demostraciones teóricas y empíricas basadas en las propiedades de Weyuker y los conceptos de Bunge y utilizando como lenguaje de programación ADA, llegó a la conclusión de que la definición de la métrica Acoplamiento por Abstracción de Datos fue encontrada como ambigua ya que esta métrica tiene dos posibles interpretaciones [27]. Una medición halla el número de datos abstractos definidos en una clase (clases cuya definición está incluida en la definición de otra) o lo que es lo mismo el número de atributos de una clase que tienen como tipo otra clase y la otra medición halla el número de atributos de una clase que están en otra clase. Estas mediciones pese a que pueden indicar cuánto depende una clase de la definición de otras y que responden a las propiedades de acoplamiento y abstracción, no se incluirán en el sistema de métricas hasta que no se lleve a cabo un estudio en el que se pueda definir su validez.

Otra de las métricas que definen Li y Henry es Tamaño 1 que no es más que una variación de la tradicional LDC definida específicamente para el lenguaje ADA pues cuenta el número de ";" en la implementación de una clase. Como se mencionó anteriormente sobre la inutilidad de las métricas que dependen del conteo de las LDC dentro de proyectos OO, desechamos el uso de esta métrica para evaluar los proyectos de software de gestión de la UCI.

De forma general el estudio del Profesor E. Manso detectó una fuerte correlación entre todas las métricas de Li y Henry cuando se validaron empíricamente, manteniendo el esfuerzo de mantenimiento durante tres años en dos aplicaciones industriales escritas en el lenguaje Classic – ADA.

Se vio que Tamaño 2 es una importante métrica predictora para las fases de prueba y mantenimiento y que necesita del código fuente, aunque cuando se omitió en el estudio, el modelo era menos confiable a pesar de ser aceptable.

Por las razones antes expuestas solo se describirán en la Propuesta del Sistema de Métricas para la Evaluación de Proyectos de Gestión de la UCI tres de las diez métricas

que proponen Li y Henry. Estas son: Acoplamiento por Paso de Mensaje, Número de Métodos Locales y Tamaño 2.

3.3.3 MDOO y MOOD.

Las tres métricas que proponen Harrison, Counseil y Nithi en su sistema MDOO y tres de las que proponen los autores Abreu y Melo en MOOD miden los mismos atributos. Estas métricas son: Factor de Polimorfismo, Factor de Herencia de los Métodos y el Factor de Acoplamiento. Con el objetivo de no hacer repeticiones innecesarias en la descripción de las métricas en el próximo capítulo, se hará una unión entre ambos sistemas por lo que se comprenderán las métricas del sistema MDOO dentro del MOOD.

Es importante para los autores aclarar algo sobre la métrica Factor de Polimorfismo: Abreu indica que, en algunos casos, sobrecargando métodos se reduce la complejidad y, por tanto, se incrementa la facilidad de mantenimiento y comprensión del sistema. Diversos autores han demostrado que la métrica Factor de Polimorfismo que proponen Abreu y Melo no cumple todas las propiedades definidas para ser válida, ya que en un sistema sin herencia el valor del Factor de Polimorfismo es indefinido, lo que exhibe una discontinuidad [25].

Por esta razón es que se quiere llamar la atención a los gestores del proyecto; cuando se vaya a utilizar esta métrica se ha de tener presente que si no hay herencia en el sistema que se va a medir la métrica es inservible, y por ende se excluye, es decir, no se usa.

Por otro lado, la métrica Factor de Acoplamiento, es quizá, la más compleja de este set, pero no conviene quitarla por la importancia y el peso que tiene el acoplamiento en los sistemas OO. Es preferible que el gestor dedique algunas horas más en su aprendizaje e interpretación, al final lo agradecerá.

3.4 Propuesta del Sistema de Métricas para la Evaluación de los Proyectos de Software de Gestión de la UCI.

A partir del estudio realizado alrededor de las métricas y después de un análisis crítico hecho en el presente capítulo sobre la necesidad de no incluir ciertas métricas y su justificación, se llega a la conclusión de que la Propuesta del Sistema de Métricas para Evaluar los Proyectos de Software de Gestión de la UCI estará conformada por las métricas que a continuación se listan:

Métricas de Proceso y Proyecto.

Nombre del sistema de métricas	**Nombre de la métrica**	**Atributo o factor que mide**
Puntos de Función.	Puntos de Función.	Tamaño de software y funcionalidad.
COCOMO II.	Ver aclaraciones en el epígrafe 4.3	Estimación de costo, tamaño y recursos.
Métricas de productividad.	• Basadas en el número de clases. • Basadas en los PF.	Productividad.
Eficacia en la Eliminación de Defectos.	Eficacia en la Eliminación de Defectos.	Filtra actividades de garantía de la calidad y control.

Métricas de Producto.

Nombre del sistema de métricas	Nombre de la métrica	Atributo o factor que mide
Complejidad.	• Complejidad Ciclomática. • fan in/fan out.	Tamaño de un Módulo, Complejidad y Dificultad.

Métricas de Persona.

Nombre del sistema de métricas	Nombre de la métrica	Atributo o factor que mide
PSP y TSP	Ver aclaraciones en el epígrafe 4.7	

Métricas del Flujo de Trabajo de Requerimientos.

Nombre del sistema de métricas	Nombre de la métrica	Atributo o factor que mide
Métricas de Requerimientos propuestas por Davis.	• Especificidad de los requisitos. • Compleción de los requisitos funcionales.	Consistencia de la interpretación de los requisitos por cada revisor. Porcentaje de funciones.

	• Grado de validación de los requisitos.	Grado de validez de los requisitos.

Métricas del Modelo de Diseño.

Nombre del sistema de métricas	**Nombre de la métrica**	**Atributo o factor que mide**
Métricas de diseño de alto nivel o arquitectónicas.	• Complejidad estructural. • Complejidad de datos. • Complejidad del sistema.	Complejidad desde diferentes puntos de vista.
Métricas de Fenton.	• Tamaño. • Amplitud. • Relación arco a nodo.	Tamaño, Amplitud y Densidad de Conectividad así como Acoplamiento de la Arquitectura.

Métricas de Flujo de Trabajo de Pruebas.

Nombre del sistema de métricas	Nombre de la métrica	Atributo o factor que mide
Métricas para el flujo de trabajo de prueba.	• Métricas para las pruebas basadas en los PF. • Métricas para las pruebas basadas en el Diseño Arquitectónico. • Métricas para las pruebas basadas en las Métricas de McCabe.	Esfuerzo global de las pruebas. Predictor de valores esperados. Proporciona información sobre la facilidad o dificultad de las pruebas de integración.

Métricas Técnicas para Sistemas OO.

Nombre del sistema de	Nombre de la métrica	Atributo o factor que

métricas		**mide**
Lorenz y Kidd.	• Número de métodos de instancias públicas. • Número de métodos de instancia. • Número de variables de instancia. • Tamaño de clase. • Tamaño medio de operación. • Número de escenarios. • Número de Clases Claves. • Número de Subsistemas. • Número de operaciones redefinidas para una subclase. • Índice de especialización.	Número de Métodos Públicos. Tamaño. Redefinición de clases. Grado de especialización. Cohesión. Parametrización.

	• Número de métodos heredados. • Promedio de parámetros por método. • Número de parámetros de media por operación.	
CK.	• Métodos ponderados por clase. • Profundidad de árbol de herencia. • Número de hijos. • Acoplamiento entre clases objeto. • Respuesta para una clase.	Complejidad. Tamaño de la Arquitectura. Características de la herencia. Acoplamiento. Potencial de comunicación entre dos clases.
Li y Henry.	• Acoplamiento por paso de mensaje. • Número de métodos locales.	Acoplamiento y comunicación. Tamaño. Complejidad e

	• Tamaño 2.	importante predictora.
MOOD de Abreu y Melo.	• Factor de ocultamiento de los métodos. • Factor de ocultamiento de los atributos. • Factor de herencia de los métodos. • Factor de herencia de los atributos. • Factor de Polimorfismo. • Factor de Acoplamiento.	Encapsulamiento. Abstracción y Funcionalidad. Flexibilidad. Acoplamiento e Interdependencia.

Métricas para medir la Calidad del Software.

Nombre del sistema de métricas	**Nombre de la métrica**	**Atributo o factor que mide**
Modelo de Calidad ISO 9126	Ver aclaraciones en el epígrafe 4.16	Funcionalidad, Confiabilidad, Usabilidad, Eficiencia, Mantenibilidad,

		Portabilidad, Productividad, Seguridad y Satisfacción.
Modelo de Calidad de McCall	Ver aclaraciones en el epígrafe 4.16	Facilidad de Mantenimiento, Facilidad de Prueba, Flexibilidad, Interoperabilidad, Portabilidad, Reusabilidad, Corrección, Fiabilidad, Eficiencia, Integridad y Facilidad de Uso.
Modelo de Calidad FURPS	Ver aclaraciones en el epígrafe 4.16	Funcionalidad, Facilidad de Uso, Fiabilidad, Rendimiento y Capacidad de Soporte.

Tabla 3. 1 Sistema de Métricas Propuesto para la Evaluación de los Proyectos de Software de Gestión de la UCI.

3.5 Conclusiones.

En este capítulo se hizo un análisis crítico de las métricas estudiadas en el capítulo 2 y se conformó la propuesta del sistema de métricas para la evaluación de proyectos de software de gestión de la UCI.

CAPÍTULO 4. DESCRIPCIÓN DEL SISTEMA DE MÉTRICAS ELEGIDO.

4.1 Introducción.

En este capítulo se describen detalladamente cada una de las métricas elegidas en el capítulo 3 que conforman el sistema propuesto. Las mismas se describen en forma de tabla para ayudar a la mejor comprensión del ingeniero que les dará uso. El formato de la tabla se presenta de la siguiente forma:

Nombre de la métrica.	
Descripción	
Entradas	
Fórmula	
Interpretación	
Bibliografía	

Tabla 4. 1 Formato de la tabla que describe las métricas del sistema.

Explicación del contenido de la tabla:

Nombre de métrica: Aparece el nombre de la métrica que se detalla.

Descripción: Se centra en describir cuál es el atributo del software que mide la métrica.

Entradas: Datos necesarios y que deben ser previamente recogidos para poder obtener el indicador que da como resultado el cálculo de la métrica.

Fórmula: Fórmula matemática que constituye la métrica en sí, con la cual se calcula el indicador a partir de las entradas recolectadas como datos que se describieron anteriormente.

Interpretación: Se hace una interpretación del indicador obtenido. Es decir, cuando se interpreta el resultado de la métrica es preciso tener en cuenta cuáles son los rangos o valores aceptables para la misma, de esta forma se puede saber si el atributo que se mide está bajo control, arrojando resultados sobre el estado de la calidad y el avance del proyecto, y facilitando las decisiones objetivas de los ingenieros de software.

Bibliografía: Se hace referencia a la bibliografía usada por los autores para confeccionar las descripciones. En las mismas se puede encontrar una explicación más profunda de las métricas que se proponen.

4.2 Métricas orientadas al tamaño.

Puntos de Función.	
Descripción	Método para medir el tamaño del software y funcionalidad. Útil en cualquier fase de vida del software.
Entradas	Número de entradas del usuario: Se cuenta cada entrada de usuario que proporciona diferentes datos orientados a la aplicación. Las entradas se deberían diferenciar de las peticiones, las cuales se cuentan de forma separada. Número de salidas del usuario: Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto la salida se refiere a informes, pantallas, mensajes de error, etc. Los elementos de datos particulares dentro de un informe no se cuentan de forma separada. Número de peticiones de usuario: Una petición se define como una

entrada interactiva que produce la generación de alguna respuesta del software inmediata en forma de salida interactiva. Se cuenta cada petición por separado.

Número de archivos: Se cuenta cada archivo maestro lógico (esto es, un grupo lógico de datos que puede ser una parte de una gran base de datos o un archivo independiente).

Número de interfaces externas: Se cuentan todas las interfaces legibles por la máquina (por ejemplo: archivos de datos de cinta o disco) que se utilizan para transmitir información a otro sistema.

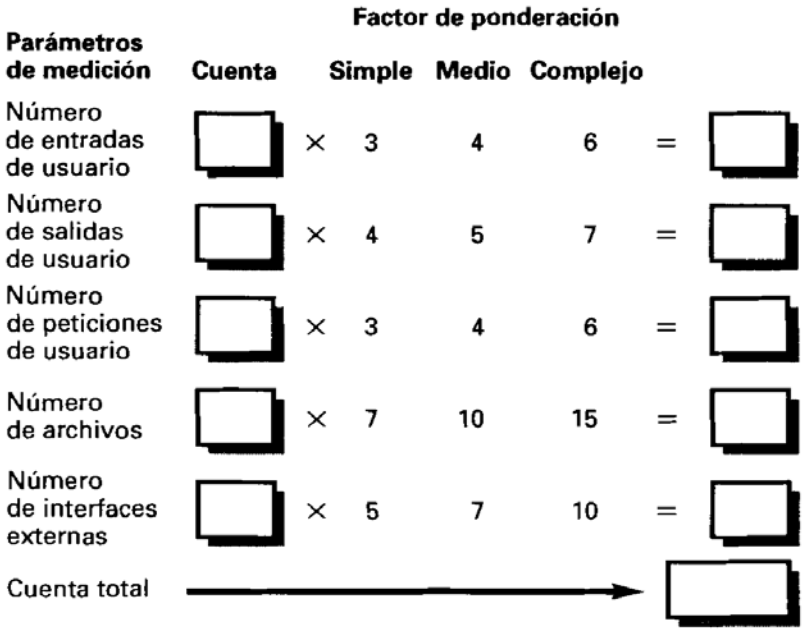


Fig 4. 1 Entradas de los PF y los factores de ponderación para calcular la Cuenta Total que se usará en la fórmula para calcular el PF.

cuenta-total: suma de todas las entradas PF obtenidas de la Fig 4.1.

Fi (i = 1 a 14): valores de ajuste de la complejidad según las

	respuestas a las siguientes preguntas: 1. ¿Requiere el sistema copias de seguridad y de recuperación fiables? 2. ¿Se requiere comunicación de datos? 3. ¿Existen funciones de procesamiento distribuido? 4. ¿Es crítico el rendimiento? 5. ¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado? 6. ¿Requiere el sistema entrada de datos interactiva? 7. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones? 8. ¿Se actualizan los archivos maestros de forma interactiva? 9. ¿Son complejas las entradas, las salidas, los archivos o las peticiones? 10. ¿Es complejo el procesamiento interno? 11. ¿Se ha diseñado el código para ser reutilizable? 12. ¿Están incluidas en el diseño la conversión y la instalación? 13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones? 14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

	Cada una de las preguntas anteriores es respondida usando una escala de rangos desde 0 (no importante o aplicable) hasta 5 (absolutamente esencial).
Fórmula	PF = cuenta-total x [0,65 + 0,01 x 6(Fi)]
Interpretación	Es importante cuidar que este valor no sobrepase límites aceptables ya que el aumento del tamaño trae consigo el aumento de la complejidad del software. Por otra parte, la funcionalidad debe ser la máxima que se pueda encontrar para un sistema determinado, por esta razón la determinación de estos valores tiene mucho que ver con la experiencia del responsable de aplicar las métricas en cualquier empresa o institución, cuidando al mismo tiempo que la complejidad se mantenga baja, pero cumpliendo con todas las funcionalidades del sistema.
Bibliografía	[14]

Tabla 4. 2 Métrica de Puntos de Función.

4.3 COCOMO II.

La documentación acerca de este tema es amplia y precisa, con una validación de hace más de quince años que reafirma a este modelo como confiable y útil. Además del libro “Ingeniería del Software. Un enfoque práctico” del autor Roger S. Pressman y del sitio oficial de COCOMO II: subset.usc.edu, se quiere señalar una bibliografía en especial que da una explicación muy sencilla y clara de la más reciente tendencia de utilización de COCOMO II, uniendo el uso de tres de las técnicas más utilizadas en la actualidad dentro del entorno OO: el desarrollo iterativo e incremental, los casos de uso y el cálculo de los Puntos de Función (PF). Se está hablando de "Estimación y Planificación de Proyectos Software con Ciclo de Vida Iterativo - Incremental y empleo de Casos de Uso" de los autores José Antonio Pow-Sang Portillo y Ricardo Imbert Paredes [15]. Se

recomienda que para la aplicación de este modelo en la UCI primero deba ser calibrado para su entorno.

A continuación también se describe una métrica para calcular costo basada en PF de una forma más sencilla que cómo se calcula o estima en COCOMO II:

Costo	
Descripción	Cuánto dinero se gasta por cada funcionalidad creada para un sistema en una etapa determinada.
Entradas	Dinero: dinero que se gasta durante la etapa. PF: total de PF creados durante esa etapa.
Fórmula	Costo = Dinero/PF
Interpretación	Este valor debe ser bajo. Mientras menos dinero se gaste por PF, menos costo tendrá el software que se desarrolla.
Bibliografía	[14] [33]

Tabla 4. 3 Métrica de Costo.

4.4 Métricas de productividad.

Productividad basada en el número de clases.	
Descripción	Mide cuántas clases desarrolla una persona en un mes.
Entradas	TCD: total de clase desarrolladas por una persona.
Fórmula	Productividad = TDC/p-mes

Interpretación	Este valor debe ser alto. La persona que más TDC desarrolle en un mes es más productiva.
Bibliografía	[14]

Tabla 4. 4 Métrica de productividad basada en el número de clases.

Productividad basada en PF.	
Descripción	Mide cuántos PF desarrolla una persona en un mes.
Entradas	Cantidad de PF desarrolladas por una persona.
Fórmula	Productividad = PF/p-mes
Interpretación	Este valor debe ser alto. Mientras más PF desarrolle una persona en un mes es más productiva.
Bibliografía	[14]

Tabla 4. 5 Métrica de productividad basada en PF.

4.5 Eficacia en la Eliminación de Defectos.

Eficacia en la Eliminación de Defectos (EED).	
Descripción	Medida de la habilidad de filtrar actividades de la garantía de la calidad y de control durante todas las actividades del marco de trabajo del proceso.
Entradas	E: número de errores encontrados antes de la entrega del software al usuario final.

	D: número de defectos encontrados después de la entrega. E_i: número de errores encontrado durante la actividad de ingeniería de software i. E_{i+1}: número de errores encontrado durante la actividad de ingeniería de software i+1 que se puede seguir para llegar a errores que no se detectaron en la actividad de la ingeniería de software i.
Fórmula	EED = $\frac{E}{E+D}$ $EED_i = \frac{E_i}{E_i + E_{i+1}}$
Interpretación	El valor ideal de EED es 1. Esto significa que no se han encontrado defectos en el software. De forma realista D será mayor que cero, pero el valor de EED todavía se aproxima a 1. Cuando E aumenta (para un valor dado de D), el valor total de EED empieza a aproximarse a 1. De hecho, a medida que E aumenta, es probable que el valor de final de D disminuya (los errores se filtran antes de que se conviertan en defectos). La EED debe usarse como una medida de la eficiencia de las primeras actividades de la Garantía de Calidad del Software (SQA por sus siglas en inglés). Si EED es bajo durante el Análisis y Diseño, debe emplearse algún tiempo mejorando la forma de conducir las revisiones técnicas formales.
Bibliografía	[14]

Tabla 4. 6 Eficacia en la Eliminación de Defectos.

4.6 Métricas de complejidad.

Complejidad Ciclomática o Método de McCabe.	
Descripción	Mide la complejidad de métodos o algoritmos a través del número de decisiones lógicas en un segmento de código. Proporciona una medida cuantitativa para probar la dificultad. Puede emplearse para proporcionar una indicación cuantitativa del tamaño máximo del módulo.
Entradas	n: número de nodos del grafo. A: número de aristas del grafo. R: número de regiones cerradas del grafo. C: número de nodos de condición o nodos que tienen más de una arista.
Fórmula	V (G) = a – n + 2 V (G) = r V (G) = c + 1
Interpretación	Mientras mayor sea la complejidad, más difícil será de comprender y modificar la estructura del sistema y habrá mayor probabilidad de que falle. Estudios experimentales indican una fuerte correlación entre la métrica de McCabe y el número de errores que existen en el código fuente, así como el tiempo requerido para encontrar y corregir dichos errores. Una complejidad ciclomática de 10 parece ser el límite práctico superior para el tamaño de un módulo. Cuando la Complejidad

	Ciclomática de los módulos excede ese valor, resulta extremadamente difícil probar adecuadamente el módulo.
Bibliografía	[19] [23]

Tabla 4. 7 Complejidad Ciclomática o Método de McCabe.

Ejemplo:

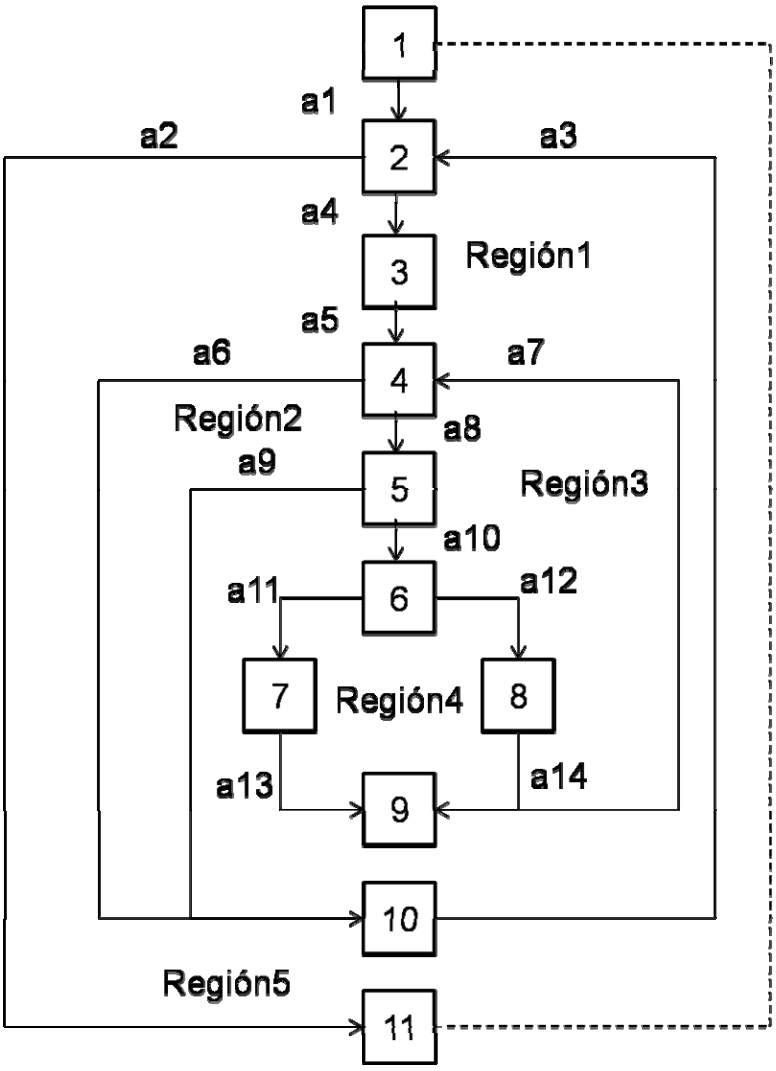


Fig 4. 2 Cálculo de la complejidad ciclomática sobre un grafo.

V (G) = 14 – 11 + 2 = 5

V (G) = 5 regiones cerradas

V (G) = 5 condiciones

Fan-in	
Descripción	Se refiere a la medida del número de funciones que llaman a otra función X.
Entradas	in: número de funciones que llaman a otra función X.
Fórmula	Fan-in = in
Interpretación	Un valor alto de fan-in indica que X está fuertemente acoplada al resto del diseño y que los cambios en X pueden tener efectos extensivos al resto del sistema.
Bibliografía	[17]

Tabla 4. 8 Métrica fan-in.

Fan-out	
Descripción	Se refieren al número de funciones que son llamadas por otra función X.
Entradas	out: número funciones que son llamadas por otra función X.
Fórmula	Fan-out = out
Interpretación	Un valor alto de fan-out indica que la complejidad de X podría ser excesiva, debido a la complejidad de la lógica de control

	necesaria para coordinar a los componentes llamados.
Bibliografía	[17]

Tabla 4. 9 Métrica fan-out.

4.7 PSP y TSP.

Las métricas del PSP y el TSP no son más que un conjunto de tablas que se encuentran definidas y explicadas ampliamente en los libros “Introducción al Proceso de Software Personal” [20] e “Introduction to the Team Software Process” [34] respectivamente, ambos de la autoría de Watts S. Humphrey, por lo cual no se considera necesario hacer una descripción detallada de las mismas en este capítulo.

4.8 Métricas de Requerimientos propuestas por Davis.

Especificidad de los Requisitos.	
Descripción	Mide consistencia de la interpretación de los revisores para cada requerimiento.
Entradas	n_{ui}: número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. n_r: cantidad de requisitos en una especificación, es decir, la suma de los requisitos funcionales y los no funcionales.
Fórmula	$Q_1 = n_{ui} / n_r$
Interpretación	A medida que la razón se acerque a uno mayor será la consistencia de la interpretación de los revisores para cada requerimiento. Esto es: menor será la ambigüedad en la especificación.

Bibliografía	[35] [23]

Tabla 4. 10 Métrica de Davis sobre Consistencia de los Requisitos.

Compleción de los Requisitos Funcionales.	
Descripción	Mide el porcentaje de funciones necesarias que se han especificado para un sistema. No trata los requisitos funcionales.
Entradas	u_n: número de requisitos únicos de función. n_i: número de entradas (estímulos) definidos o implicados por la especificación. n_s: número de estados especificados.
Fórmula	$Q_2 = u_n/(n_i*n_s)$
Interpretación	Da una medida de la funcionalidad del sistema.
Bibliografía	[23]

Tabla 4. 11 Métrica de Davis sobre Compleción de los Requisitos Funcionales.

Grado de Validación de los Requisitos.	
Descripción	Mide el grado de validez que tienen los requisitos de un sistema.
Entradas	n_c: número de requisitos que se han validado como correctos. n_{nv}: número de requisitos que no se han validado todavía.

Fórmula	$Q_3 = n_c/(n_c + n_{nv})$
Interpretación	Este valor debe acercarse a uno tanto como pueda.
Bibliografía	[23]

Tabla 4. 12 Métrica de Davis sobre Grado de Validación de los Requisitos.

4.9 Métricas de Diseño de Alto Nivel o Arquitectónicas.

Complejidad Estructural.	
Descripción	Mide la complejidad a través de la expansión de un módulo.
Entradas	fout (i): expansión del módulo i. La expansión indica el número de módulos que son invocados directamente por el módulo i.
Fórmula	$S(i) = (fout(i))^2$
Interpretación	A mayor cantidad de módulos invocados por el módulo i, mayor será la complejidad de ese módulo, por lo que se hará más difícil de comprender la estructura del sistema y habrá mayor probabilidad de que falle.
Bibliografía	[35] [14]

Tabla 4. 13 Métrica de complejidad estructural.

Complejidad de Datos.	
Descripción	Proporciona una indicación de la complejidad en la interfaz interna de un módulo i.

Entradas	v (i): número de variables de entrada y salida del módulo i.
Fórmula	D (i)=v (i)/ [fout (i) + 1]
Interpretación	Un valor elevado de este indicador alerta sobre la necesidad de poner atención a aquellos módulos cuya interfaz sea muy compleja sin necesidad.
Bibliografía	[35] [14]

Tabla 4. 14 Métrica de complejidad de datos.

Complejidad del Sistema.	
Descripción	Es la suma de las complejidades estructural y de datos y determina la complejidad de todo el sistema.
Entradas	S(i) y D(i)
Fórmula	C(i) = S(i) + D(i)
Interpretación	A medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas.
Bibliografía	[35] [14]

Tabla 4. 15 Métrica de complejidad del sistema.

4.10 Métricas de Fenton.

Tamaño.	
Descripción	Determina el tamaño de una arquitectura determinada.
Entradas	n: número de nodos (módulos) a: número de arcos (líneas de control)
Fórmula	Tamaño = n + a
Interpretación	Compara diferentes arquitecturas mediante el cálculo del número de nodos y de arcos de un grafo que representa la arquitectura de un software.
Bibliografía	[14]

Tabla 4. 16 Métrica de tamaño de Fenton.

Amplitud.	
Descripción	Compara diferentes arquitecturas mediante el cálculo de la cantidad máxima de nodos de cualquier nivel de la arquitectura.
Entradas	a: número máximo de nodos de cualquier nivel.
Fórmula	Amplitud = a
Interpretación	Se puede determinar la amplitud de un nivel cualquiera de la arquitectura. Proporciona una indicación del ámbito de control global.

Bibliografía	[14]

Tabla 4. 17 Métrica de amplitud de Fenton.

Relación arco a nodo.	
Descripción	Mide la densidad de conectividad de la arquitectura y proporciona una indicación sencilla de acoplamiento de la arquitectura.
Entradas	a: número de arcos. n: número de nodos.
Fórmula	r=a/n
Interpretación	El aumento del valor de r indica que existen muchos nodos dentro de la arquitectura que no están conectados con otros nodos.
Bibliografía	[14]

Tabla 4. 18 Relación arco a nodo de Fenton.

Ejemplo:

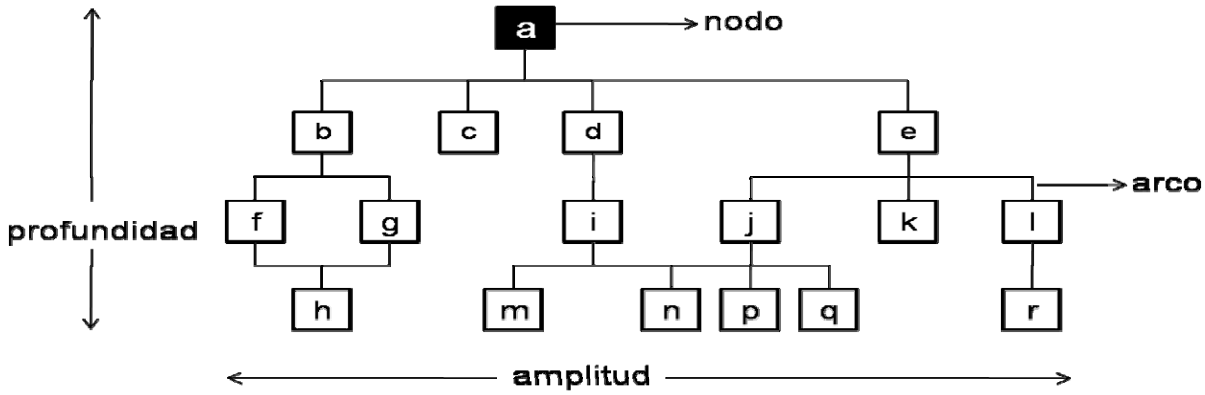


Fig 4. 3 Arquitectura de Software. Grafo para calcular las métricas de Fenton.

Tamaño = 17 + 18 = 35.

Amplitud = 6

Relación arco a nodo = 18/17 = 1

4.11 Métricas para el Flujo de Trabajo de Pruebas.

Métricas de pruebas basadas en la función.	
Descripción	Las métricas de pruebas basadas en la función pueden emplearse para predecir el esfuerzo global de las pruebas. Se pueden juntar y correlacionar varias características a nivel de proyecto de proyectos anteriores con el número de PF producidos por un equipo del proyecto. El equipo puede además predecir los valores "esperados" de estas características del proyecto actual.
Entradas	PF
Fórmula	Las características a nivel de proyecto que se pueden juntar y correlacionar a partir del valor de PF son, entre otras: esfuerzo y tiempo para las pruebas, errores descubiertos, número de casos de prueba que deben ser producidos.
Bibliografía	[14]

Tabla 4. 19 Métricas de pruebas basadas en la función.

Métricas de pruebas basadas en el diseño arquitectónico.	
Descripción	Las métricas del diseño arquitectónico proporcionan información sobre la facilidad o dificultad asociada con la

	prueba de integración y la necesidad de software especializado de prueba.
Entradas	Resultado del valor de la métrica Complejidad Ciclomática.
Interpretación	La Complejidad Ciclomática se encuentra en el núcleo de las pruebas de caminos básicos, que es un método de diseño de casos de prueba. Además la Complejidad Ciclomática puede usarse para elegir módulos como candidatos para pruebas más profundas.
Bibliografía	[14]

Tabla 4. 20 Métricas de pruebas basadas en el diseño arquitectónico.

Métricas de pruebas basadas en las métricas de McCabe.	
Descripción	Se eligen tantos casos de prueba como caminos independientes son calculados como V (G).
Entradas	V (G): número de decisiones lógicas en un segmento de código. (descrito en detalle en 4.6)
Fórmula	Casos de prueba = V(G)
Interpretación	V (G) marca el límite mínimo de casos de prueba para un programa. Cuando V (G) es mayor que diez la probabilidad de defectos en el módulo o programa crece mucho por lo que quizá sea interesante dividir el módulo.
Bibliografía	[14]

Tabla 4. 21 Métricas de pruebas basadas en las métricas de McCabe.

4.12 Métricas propuestas por Lorenz y Kidd.

Número de Métodos de Instancia Públicos (PIM).	
Descripción	Calcula el número total de métodos públicos de instancias.
Entradas	mpi: número total de métodos que están disponibles para otras clases.
Fórmula	PIM = mpi
Interpretación	Mide la cantidad de responsabilidad que tiene una clase.
Bibliografía	[25]

Tabla 4. 22 Métrica de Lorenz y Kidd. Número de Métodos de Instancia Públicos.

Número de Métodos de Instancia (NIM).	
Descripción	Se define como la suma de todos los métodos (públicos, protegidos y privados) de una clase.
Entradas	mp: número total de métodos públicos de una clase. mr: número total de métodos protegidos de una clase. mi: número total de métodos privados de una clase.
Fórmula	NIM = mp + mr + mi
Interpretación	Mide la cantidad de colaboración utilizada.
Bibliografía	[25]

Tabla 4. 23 Métrica de Lorenz y Kidd. Número de Métodos de Instancia.

Número de Variables de Instancia (NVI).	
Descripción	Se determina por el número total de variables (privadas y protegidas) a nivel de instancia que tiene una clase.
Entradas	vp: número total de variables privadas a nivel de instancia de una clase. vr: número total de variables protegidas a nivel de instancia de una clase.
Fórmula	NVI = vp + vr
Bibliografía	[25]

Tabla 4. 24 Métrica de Lorenz y Kidd. Número de Variables de Instancia.

Tamaño de Clase (TC).	
Descripción	Calcula el tamaño de la clase basada en sus operaciones y atributos. Durante la revisión del modelo de Análisis Orientado a Objetos (AOO), las tarjetas CRC proporcionan una indicación razonable de los valores esperados por TC.
Entradas	o: total de operaciones tanto heredadas como privadas de la instancia que se encapsulan dentro de la clase. a: número de atributos tanto heredados como privados de la instancia encapsulados por la clase.
Fórmula	TC = o + a
Interpretación	Si encuentra una clase con demasiada responsabilidad durante el AOO, considere el particionarlo. Esto aumentaría la

	reutilización de la clase y simplificaría la implementación y las pruebas. En general, operaciones y atributos heredados o públicos deben ser ponderados con mayor importancia, cuando se determina el tamaño de clase. Operaciones y atributos privados, permiten la especialización y son más propios del diseño. También se pueden calcular los promedios para el número de atributos y operaciones de clase. Cuanto menor sea el valor promedio para el tamaño, sería más posible que las clases dentro del sistema puedan reutilizarse.
Bibliografía	[25]

Tabla 4. 25 Métrica de Lorenz y Kidd. Tamaño de Clases.

Tamaño medio de operación (TMO).	
Descripción	Proporciona una alternativa para calcular el tamaño de operación.
Entradas	nm: número de mensajes enviados por operación.
Fórmula	TMO = nm
Interpretación	A medida que el número de mensajes enviados por una sola operación se incrementa, es más probable que las responsabilidades no hayan sido correctamente asignadas dentro de la clase.
Bibliografía	[25]

Tabla 4. 26 Métrica de Lorenz y Kidd. Tamaño Medio de Operación.

Número de Escenarios (NE).	
Descripción	Proporciona un indicador importante del tamaño de un programa.
Entradas	E: número de pasos que describen la interacción entre el usuario y la aplicación.
Fórmula	NE = E
Bibliografía	[25]

Tabla 4. 27 Métrica de Lorenz y Kidd. Número de Escenarios.

Número de subsistemas (NSUB).	
Descripción	Proporciona el número de subsistemas.
Entradas	NSUB: número de subsistemas.
Fórmula	NSUB = NSUB
Interpretación	Proporciona una visión sobre la asignación de recursos, la planificación y el esfuerzo de integración global.
Bibliografía	[25]

Tabla 4. 28 Métrica de Lorenz y Kidd. Número de Subsistema.

Número de Clases Claves (NCC).	
Descripción	Una clase clave se centra directamente en el dominio del negocio para el problema, y tendrá una menor probabilidad de ser implementada por medio de la reutilización.
Entradas	NCC: número de clases claves. Clase clave: clase central al dominio del problema.
Fórmula	NCC = NCC
Interpretación	Valores altos para NCC indican gran trabajo de desarrollo substancial. Lorenz y Kidd sugieren que entre el 20 y el 40 % de todas las clases en un sistema OO típico corresponde a las clases clave. El resto es infraestructura de soporte (GUI, comunicaciones, bases de datos, etc.).
Bibliografía	[25]

Tabla 4. 29 Métrica de Lorenz y Kidd. Número de Clases Claves.

Número de Operaciones Redefinidas para una Subclase (NOR).	
Descripción	Existen casos en que una subclase reemplaza una operación heredada de su superclase por una versión especializada para su propio uso. A esto se le llama redefinición.
Entradas	nor: número de operaciones redefinidas para una subclase.
Fórmula	NOR = nor
Interpretación	Los valores grandes para el NOR, generalmente indican un

	problema de diseño. Tal como indican Lorenz y Kidd: "Dado que una subclase debe ser la especialización de sus superclases, deben, sobre todo, extender los servicios (operaciones) de las superclases." Si el NOR es grande, el diseñador ha violado la abstracción representada por la superclase. Esto provoca una débil jerarquía de clases y un software OO que puede ser difícil de probar y modificar.
Bibliografía	[25]

Tabla 4. 30 Métrica de Lorenz y Kidd. Número de Operaciones Redefinidas para una Subclase.

Índice de especialización (IES).	
Descripción	El índice de especialización proporciona una indicación aproximada del grado de especialización, para cada una de las subclases en un sistema OO. La especialización se puede alcanzar añadiendo o eliminando operaciones, pero también redefiniendo.
Entradas	NOR: número de operaciones redefinidas para una subclase. nivel: nivel en la jerarquía de clases en que reside la clase. M_{total}: número total de métodos de la clase.
Fórmula	IE = $\frac{NOR * nivel}{M_{total}}$
Interpretación	Cuanto más elevado sea el valor de IE**,** más probable será que

	la jerarquía de clases tenga clases que no se ajusten a la abstracción de la superclase.
Bibliografía	[25]

Tabla 4. 31 Métrica de Lorenz y Kidd. Índice de Especialización.

Número de Métodos Heredados (NMH).	
Descripción	Se define como el número de métodos que hereda una clase.
Entradas	nmh: número de métodos que hereda una clase.
Fórmula	NMI = nmh
Interpretación	Mide la calidad del uso de la herencia.
Bibliografía	[25]

Tabla 4. 32 Métrica de Lorenz y Kidd. Número de Métodos Heredados.

Promedio de Parámetros por Método (PPM).	
Descripción	Examina la cohesión y aspectos relacionados con el código. Estas métricas no se han validado teóricamente. Se han validado parcialmente de forma empírica [25].
Entradas	p: número total de parámetros por método. m: número total de métodos.
Fórmula	APPM = p/m

Interpretación	Este valor alto presupone una cohesión elevada.
Bibliografía	[25]

Tabla 4. 33 Métrica de Lorenz y Kidd. Promedio de Parámetros por Métodos.

Número Medio de Parámetros por Operación (NPM).	
Descripción	Calcula el número de parámetros por operación.
Entradas	npo: número de parámetros por operación.
Fórmula	NPM = npo
Interpretación	Tan largo como sea el número de parámetros de operación, más compleja será la colaboración entre objetos. En general, NPM, debe mantenerse tan baja como sea posible.
Bibliografía	[25]

Tabla 4. 34 Métrica de Lorenz y Kidd. Número de Parámetros de Media por Operación.

4.13 La serie de métricas CK.

Métodos Ponderados por Clase (MPC).	
Descripción	Asumen que n métodos de complejidad c_1, c_2,... c_n se definen para la clase C, por lo que mide complejidad de la clase.
Entradas	c_1, c_2,... c_n: complejidad de los métodos de la clase C que puede ser calculado por cualquiera de los métodos para medir complejidad que existen.

Fórmula	$MPC = \sum_{i=1}^{n} c_i$
Interpretación	Un resultados menor o igual que 100 es aceptable para esta métrica. El número de métodos y su complejidad son indicadores razonables de la cantidad de esfuerzo requerido para implementar y verificar una clase. En suma, cuanto mayor sea el número de métodos, más complejo es el árbol de herencia (todas las subclases heredan métodos de sus padres). Finalmente, a medida que crece el número de métodos para una clase dada, más probable es que se vuelvan más y más específicos de la aplicación, así que se limita el potencial de reutilización. Por todas estas razones, el MPC debe mantenerse tan bajo como sea razonable. El número de métodos y su complejidad están directamente relacionados con el esfuerzo requerido para comprobar una clase. Es importante conocer que la Administración Nacional de Aeronáutica y del Espacio (NASA por sus siglas en inglés) recomienda que la cantidad de métodos por clases deben estar preferiblemente por debajo de 20, aunque es aceptable la cifra de 40. Aún así, teniendo en cuenta los constructores y destructores de los métodos de conteo, estas cifras están infladas, por lo que recomienda que esa cifra esté por debajo de 10.
Bibliografía	[14] [26] [27]

Tabla 4. 35 Métrica de Chidamber y Kemerer. Métodos Ponderados por Clases.

Profundidad de Árbol de Herencia (PAH).	
Descripción	Se define como la máxima longitud del nodo a la raíz del árbol.
Entradas	n: número de niveles del árbol.
Fórmula	PAH = n
Interpretación	A medida que la PAH crece, es posible que clases de más bajos niveles hereden muchos métodos. Esto conlleva dificultades potenciales, cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (el PAH es largo) también conduce a una complejidad de diseño mayor. Por el lado positivo, los valores PAH grandes implican un gran número de métodos que se reutilizarán. Una clase con un alto número de hijos, se considerará mucho más en el diseño debido a su influencia y requiere un mayor número de test en la fase de pruebas. Un número pequeño de hijos puede indicar en ciertos casos una falta de comunicación entre los desarrolladores que desaprovechan oportunidades de reutilización y sugiere también que no se ha adoptado completamente el uso de la herencia en el diseño. La experiencia que ha tenido la NASA con esta métrica nos muestra que su resultado debe estar entre 2 y 3, esto indica que hay elevadas posibilidades de reusabilidad. Un valor por debajo de 2 representa un diseño pobre y por encima de 5 podría ser un suicidio, ya que se aprovecha mucho la heredad, pero se paga un precio muy alto por la complejidad que adquiere el diseño.

Bibliografía	[14] [26] [36] [27]

Tabla 4. 36 Métrica de Chidamber y Kemerer. Profundidad de Árbol de Herencia.

Número de Hijos (NDH).	
Descripción	Las subclases inmediatamente subordinadas a una clase en la jerarquía de clases se denominan sus descendientes. Luego, NDH se define como el número de descendientes que tiene una jerarquía de clases.
Entradas	nd: número de descendientes de una clase.
Fórmula	NDH = nd
Interpretación	A medida que el número de descendientes crece, la reutilización se incrementa y la abstracción representada por la clase predecesora puede diluirse. Esto significa que existe una posibilidad de que algunos descendientes no sean miembros realmente apropiados de la clase predecesora. A medida que el NDH crece, la cantidad de pruebas (requeridas para ejercitar cada descendiente en su contexto operativo) se incrementará también. La herencia es una habilidad extremadamente poderosa que puede crear problemas si no se usa con cuidado. El PAH se usa para dar una lectura de la complejidad de la jerarquía de clases.
Bibliografía	[14] [26] [27]

Tabla 4. 37 Métrica de Chidamber y Kemerer. Número de Hijos.

Acoplamiento entre Clases Objetos (AEC).	
Descripción	El modelo clases-responsabilidades-colaboraciones (CRC) debe utilizarse para determinar el valor de AEC. En esencia, AEC es el número de colaboraciones listadas para una clase en la tarjeta índice CRC.
Entradas	c: número de colaboraciones listadas para una clase en la tarjeta CRC.
Fórmula	AEC = c
Interpretación	Este valor debe ser menor o igual que 5. A medida que AEC se incrementa, es más probable que el grado de reutilización de una clase decrezca. Valores altos de AEC además complican las modificaciones y las pruebas, que se producen cuando se realizan modificaciones. En general, los valores de AEC para cada clase deben mantenerse tan bajos como sea razonable. Mantenga el acoplamiento de clases baja y la cohesión de operación alta.
Bibliografía	[14] [26] [27]

Tabla 4. 38 Métrica de Chidamber y Kemerer. Acoplamiento entre Clases Objetos.

Respuesta de una Clase (RPC).	
Descripción	El conjunto de respuesta de una clase es “una serie de métodos que pueden potencialmente ser ejecutados, en respuesta a un mensaje recibido por un objeto, en la clase”. Se puede considerar esta métrica como el potencial de

	comunicación entre dos clases [36].
Entradas	m: número de métodos en el conjunto de respuesta.
Fórmula	RPC = m
Interpretación	Una cifra por la que calibrar qué valor debe tener este indicador es 100 o menos. A medida que la RPC aumenta, el esfuerzo requerido para la comprobación también se incrementa. Así mismo, se dice que, así como la RPC aumenta, la complejidad del diseño global de la clase se incrementa. Además si el valor de esta métrica es alto, indica que el número posible de métodos a invocar será alto, por tanto, se necesitará montar un sistema de pruebas y depuración que exigirá un nivel elevado de comprensión del sistema. A mayor respuestas que deba dar una clase a una petición, mayor complejidad de esta. Un número pequeño de clases generará usualmente un alto número de métodos.
Bibliografía	[14] [36] [26] [27]

Tabla 4. 39 Métrica de Chidamber y Kemerer. Respuesta de una Clase.

Ejemplo:

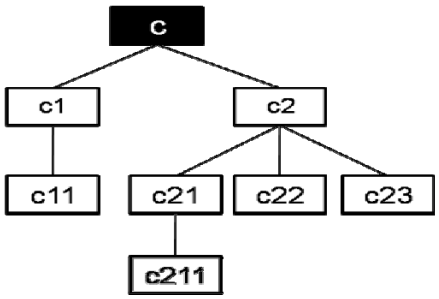
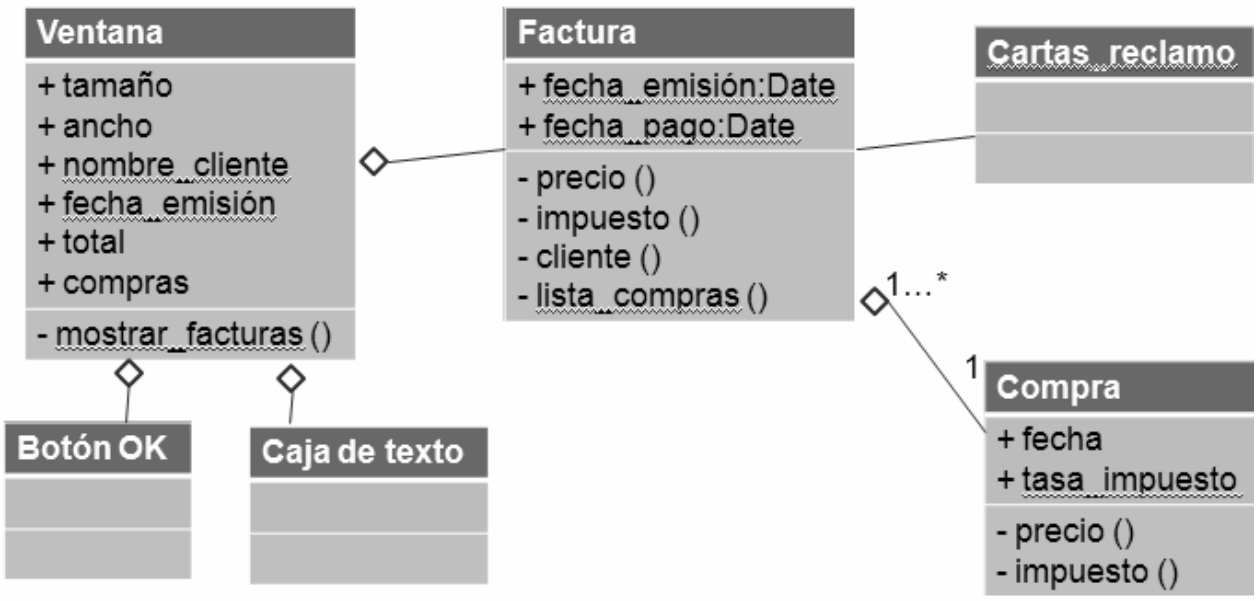


Fig 4. 4 Una jerarquía de clases.

PAH = 4

NDH (C_2) = 3

La clase C_2 tiene tres descendientes – subclases C_{21}, C_{22} y C_{23}



Métrica	factura	compra	cartas_ reclamo	botón OK	ventana	caja de texto
MPC	4	2	0	0	1	0
NDC	0	0	0	0	0	0
PAH	0	0	1	0	1	0
RPC	-	-	-	-	-	-
AEC	3	4	2	1	3	1

Fig 4. 5 Ejemplo de aplicación de métricas CK.

4.14 Li y Henry.

Acoplamiento por Paso de Mensajes (APM).	
Descripción	Número de mensajes enviados por una clase a otras del sistema. Mide las propiedades de acoplamiento y comunicación.
Entradas	me: número de mensajes enviados por una clase a otras del sistema.
Fórmula	APM = me
Interpretación	Indica cuánto depende la implementación de los métodos de las otras clases.
Bibliografía	[14]

Tabla 4. 40 Métrica de Li y Henry. Acoplamiento por Paso de Mensajes.

Número de Métodos Locales (NML).	
Descripción	Se refiere al número de métodos localmente definidos en la clase, incluyendo los métodos heredados.
Entradas	mld: número de métodos localmente definidos en la clase, incluyendo los métodos heredados.
Fórmula	NML = mld
Interpretación	Indica el incremento de la interfaz aportada por esta clase. Relacionado con propiedades operacionales de una clase y

	complejidad de una clase.
Bibliografía	[10] [14]

Tabla 4. 41 Métrica de Li y Henry. Número de Métodos Locales.

Tamaño 2.	
Descripción	Deriva indirectamente de las LDC. Mide aspectos de la complejidad de una clase a través del conteo de sus atributos y métodos locales. Es importante predictora [27].
Entradas	pc = número de punto y comas en la implementación de una clase.
Fórmula	Tamaño 1 = pc
Bibliografía	[10] [14]

Tabla 4. 42 Métrica de Li y Henry. Tamaño 2.

4.15 MOOD de Abreu y Melo.

Factor de Ocultamiento de los Métodos (FOM).	
Descripción	Mide encapsulamiento.
Entradas	C_i: clase dentro de la arquitectura. M_h (C_i): número de métodos ocultos en C_i. M_d (C_i): número de métodos definidos en C_i. T_c: número de clases del sistema.

	Para el cálculo de esta métrica no se consideran los métodos heredados.
Fórmula	$FOM = \frac{\sum_{i=1}^{T_c} M_h(C_i)}{\sum_{i=1}^{T_c} M_d(C_i)}$
Interpretación	Abreu y Melo demostraron empíricamente que cuando se incrementa FOM, la densidad de defectos y el esfuerzo necesario para corregirlos deberían disminuir. Se plantea que el resultado del indicador debe oscilar entre el 10 y el 30 por ciento. Es decir que lo mejor es que los métodos se traten de declarar siempre públicos.
Bibliografía	[25] [27]

Tabla 4. 43 Métrica de Abreu y Melo. Factor de Ocultamiento de los Métodos.

Factor de Ocultamiento de los Atributos (FOA).	
Descripción	Mide encapsulamiento. Se define como el cociente entre el número de invisibilidades de todos los atributos definidos en todas las clases y el número total de atributos definidos en el sistema.
Entradas	C_i: clase dentro de la arquitectura. A_h (C_i): número de atributos ocultos en C_i. A_d (C_i): número de atributos definidos en C_i.

	T_c: número de clases del sistema.
Fórmula	$FOA = \frac{\sum_{i=1}^{T_c} A_h(C_i)}{\sum_{i=1}^{T_c} A_d(C_i)}$
Interpretación	Idealmente el valor de esta métrica debería ser siempre 1, siendo necesario para ello ocultar todos los atributos. Las pautas de diseño OO sugieren que no hay que emplear atributos públicos, ya que se considera que esto viola los principios de encapsulamiento al exponer la implementación de las clases. Para mejorar el rendimiento, a veces se evita el uso de métodos que acceden o modifican atributos accediendo a ellos directamente. Un rango recomendado en porciento para este indicador oscila entre el 70 y el 100%.
Bibliografía	[25] [27]

Tabla 4. 44 Métrica de Abreu y Melo. Factor de Ocultamiento de los Atributos.

Factor de Herencia de los Métodos (FHM).	
Descripción	Grado en que la arquitectura de clases de un sistema OO hace uso de la herencia a través de los métodos u operaciones. Mide abstracción y funcionalidad.
Entradas	C_i: clase dentro de la arquitectura. M_h (C_i): número de métodos heredados (y no redefinidos) en Ci.

	$M_d(C_i)$: número total de métodos disponibles en C_i, es decir, los heredados más los definidos y redefinidos por la propia clase. T_c: número de clases del sistema.
Fórmula	$FHM = \frac{\sum_{i=1}^{T_c} M_h(C_i)}{\sum_{i=1}^{T_c} M_d(C_i)}$
Interpretación	El valor de FHM proporciona una referencia sobre el impacto de la herencia en el software OO. Se propone como ayuda para evaluar la cantidad de recursos necesarios a la hora de probar. El empleo de la herencia se ve como un compromiso entre la facilidad de reutilización que proporciona, y la facilidad de comprensión y mantenimiento del sistema. Un rango recomendado en el cual debe estar el valor del indicador es entre el 65 y el 80%.
Bibliografía	[25] [27]

Tabla 4. 45 Métrica de Abreu y Melo. Factor de Herencia de los Métodos.

Factor de Herencia de los Atributos (FHA).	
Descripción	Está definido como el cociente entre el número de atributos heredados en todas las clases del sistema y el número total de atributos existentes (heredados y definidos localmente) en todas las clases. Mide abstracción.

Entradas	A_h (C_i): número de atributos heredados en C_i. A_d (C_i): número total de atributos disponibles en C_i, es decir, los heredados más los definidos en la clase. T_c: número de clases en el sistema.
Fórmula	$$FHA = \frac{\sum_{i=1}^{T_c} A_h(C_i)}{\sum_{i=1}^{T_c} A_d(C_i)}$$
Interpretación	Demasiada reutilización de código a través de la herencia hace que el sistema sea más difícil de entender y mantener. Se recomienda un rango para el indicador entre un 50 y un 60%, esto da la medida de que los atributos deben heredarse con menor frecuencia que los métodos.
Bibliografía	[25] [27]

Tabla 4. 46 Métrica de Abreu y Melo. Factor de Herencia de los Atributos.

Factor de Polimorfismo (FPO).	
Descripción	Mide flexibilidad. Es una medida indirecta de la asociación dinámica del sistema. Se debe a la herencia.
Entradas	M_r (C_i): número de métodos redefinidos en C_i. M_n (C_i): número de métodos nuevos en C_i. T_c: número de clases del sistema.

	DC (C_i): número de clases derivadas de C_i.
Fórmula	$FPO = \frac{\sum_{i=1}^{T_c} M_r(C_i)}{\sum_{i=1}^{T_c} (M_n(C_i) * DC(C_i))}$
Interpretación	Abreu indica que, en algunos casos, sobrecargando métodos se reduce la complejidad y, por tanto, se incrementa la facilidad de mantenimiento y comprensión del sistema. Diversos autores han demostrado que la métrica Factor de Polimorfismo que proponen Abreu y Melo no cumple todas las propiedades definidas para ser válida, ya que en un sistema sin herencia el valor del Factor de Polimorfismo es indefinido, lo que exhibe una discontinuidad por lo que no se recomienda que su uso en caso de que el sistema a medir no presente herencia. Un rango recomendado en el que debe fluctuar el resultado de esta métrica es del 3.5 al 10%.
Bibliografía	[25] [27]

Tabla 4. 47 Métrica de Abreu y Melo. Factor de Polimorfismo.

Factor de Acoplamiento (FA).	
Descripción	Un objeto está acoplado a otro si instancias de una clase utiliza métodos o variables de instancias de otra clase. Esta situación no es nada deseable en el diagrama orientado a objetos. El FA es un indicador de las conexiones entre los elementos del diseño OO, es decir, mide interdependencia.

Entradas	C_i: clase dentro de la arquitectura. Tc: número de clases del sistema. DC (C_i): número de clases derivadas de la C_i. Denominador: número máximo de acoplamientos menos máximo número de acoplamientos debidos a la herencia. $es\_cliente(C_i, C_j)\begin{cases} 1.si.existe.una.relación.entre.la.clase.cliente.C_i \\ .y.la.clase.servidora.C_j.y.C_i \neq C_j. \\ 0.en.cualquier.otro.caso. \end{cases}$
Fórmula	$FA = \dfrac{\sum_{i=1}^{T_c}\sum_{j=1}^{T_c} es\_cliente(C_i, C_j)}{T_c^{\,2} - T_c - 2 * \sum_{i=1}^{T_c} DC(C_i)}$
Interpretación	Conforme el valor del FA crece, la complejidad del software OO también crece, reduciendo el encapsulamiento y el posible reuso; limita, por tanto, la facilidad de comprensión y de mantenimiento del sistema. Al incrementar el acoplamiento entre clases, se incrementa la densidad de defectos y más rigurosas deberán ser las pruebas requeridas sobre las clases involucradas. Un rango para el indicador se propone que esté entre un 4 y un 20%.
Bibliografía	[25] [36] [27]

Tabla 4. 48 Métrica de Abreu y Melo. Factor de Acoplamiento.

4.16 Modelos de Calidad.

Todos los modelos de calidad estudiados y finalmente propuestos se estructuran básicamente de la misma forma, es decir, utilizan factores de calidad para medir indirectamente la calidad del software. No se ha querido forzar a la utilización de uno en específico debido a que todos se encuentran muy bien definidos y son aplicables en la actualidad, aunque se debe señalar que el modelo ISO/IEC 9126 es el más usado y conocido por los proyectos de la UCI. Estos modelos se encuentran definidos y explicados ampliamente en el libro “Ingeniería del Software. Un enfoque práctico” [14] del autor Roger S. Pressman, por lo cual no se considera necesario hacer una descripción detallada de las mismas en este capítulo. No obstante, para profundizar y refinar el uso del Modelo ISO/IEC 9126 puede hacerse uso de la Norma.

A continuación también se describe una métrica de calidad basada en PF.

Calidad.	
Descripción	Cuántos errores se comenten por PF.
Entradas	Errores cometidos durante una etapa a la que se le calcula el PF.
Fórmula	Calidad = Errores/PF
Interpretación	Este valor debe ser bajo. A menor cantidad de errores por PF, más será la calidad.
Bibliografía	[14]

Tabla 4. 49 Métrica de Calidad.

4.17 Conclusiones.

En este capítulo se describió detalladamente cada una de las métricas elegidas, quedando conformada finalmente la Propuesta del Sistema de Métricas para Evaluar los Proyectos de Software de Gestión de la UCI.

CONCLUSIONES

Durante el desarrollo de este trabajo:

- Se expuso la necesidad de proponer un sistema de métricas acorde a las características de los proyectos de software de gestión de la UCI.
- Se realizó un estudio amplio de las métricas actuales.
- Se elaboró la Propuesta de un Sistema de Métricas para la Evaluación de Proyectos de Software de Gestión de la UCI.

Por todo lo anteriormente dicho se considera cumplido el objetivo planteado en esta investigación.

RECOMENDACIONES

- Continuar profundizando en el estudio y enriquecimiento del sistema propuesto.
- Validar esta propuesta empíricamente.
- Crear una cultura del uso de las métricas dentro de la UCI.
- Proponer una plantilla de recolección de datos para ser usados en los cálculos de las métricas.
- Utilizar herramientas automatizadas que permitan llevar el control de las métricas.
- Crear un repositorio con la información de los resultados de las métricas aplicadas a los diferentes proyectos de la universidad

BIBLIOGRAFÍA

1. Inc., S.P.C. *Metrics*. [cited diciembre 2006]; Available from: http://www.spc.ca/resources/metrics/index.htm.

2. IEEE, *"Standard Glossary of Software Engineering."* 1993.

3. Vargas., J.M.T., *"Temas selectos de programación 2. CIMAT Métricas del proceso de software."*

4. Westfall, L.L., *"Software metrics that meet your information needs"*. 1995.

5. Nusenoff., D.C.B.y.R.E., *"A Guide Book and Spedsheet Tool for a Corporate Metrics Program"*. 1993.

6. Zuse., H. *"History of Software Measurement."* 1995 [cited 8 marzo 2007]; Available from: http://irb.cs.tu-berlin.de/~zuse/metrics/History_00.html.

7. Corporation., R.U.P.R.S., *"Rational Unified Process." Version 2003.06.00.65 Copyright 1987 - 2003.* 2003.

8. R.S. Pressman & Associates, I. *"Software Process Metrics"*. 2001-2005 [cited 9 marzo 2007].

9. Palacio, J. (2005) *"Gestión y procesos en empresas de software"*. **Volume**, 19

10. Ma. Teresa Ventura Miranda, M.P.B. (2006) *"MoProSoft: modelo de procesos de software hecho en México."* **Volume**,

11. Trabajo., R.P.G.y.C.d. *"Propuesta de Modelo de Procesos para la Producción de Software en la UCI v1.0."* 2006 [cited 2007 16 febrero].

12. Daniel, M. (2006) *"Métrica y Complejidad del Software"*. junio **Volume**,

13. Wikipedia. *"Puntos de Función."* 2007 [cited 18 abril 2007]; Available from: http://www.wikipedia.org/Puntos_de_Función.

14. Pressman, R.S., *"Ingeniería del Software. Un enfoque práctico."* Quinta Edición. ed. Vol. 2. 2005. 601.

15. José Antonio Pow-Sang Portillo, R.I.P. *"Estimación y Planificación de Proyectos Software con Ciclo de Vida Iterativo - Incremental y empleo de Casos de Uso."* **Volume**, 6

16. Estévez, I.P., *"Métricas para el control de proyectos de software. "* 2002, Instituto superior politécnico "José Antonio Echeverría": Ciudad de la Habana. p. 143.

17. Alonso., E.B., *"Medición y métricas del software."* in *Administración de proyectos.*, Universidad de Vigo. Departamento de informática. p. 5.

18. I. Jacobson, G.B., J. Rumbaugh, *"El proceso unificado de desarrollo de software."* Segunda Edición ed. 2000, Madrid: Addison-Wesley. 109.

19. Maribel Ariza Rojas, J.C.M.G. (2004) *"Introducción y principios básicos del desarrollo de software basado en componentes."* **Volume**, 12

20. Humphrey, W.S., *"Introducción al Proceso de Software Personal."* Segunda Edición ed. 2001, Madrid: Addison Wesley. 301.

21. University, C.M. *"Team Software Process (TSP)."* 2007 [cited 16 abril 2007]; Available from: http://www.sei.cmu.edu/tsp/tsp.html.

22. University, C.M. *"TSP and the Integrated Software Acquisition Metrics (ISAM) Project."* 2007 [cited 16 abril 2007]; Available from: http://www.sei.cmu.edu/tsp/isam.html.

23. Pressman, R.S., *"Ingeniería del Software. Un enfoque práctico."* Quinta Edición. ed. Vol. 2. 2002. 601.

24. Wikipedia. *"Programación Orientada a Objetos."* 2007 [cited 20 abril 2007]; Available from: http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos.

25. Escorial, J.S. *"Métricas para modelos conceptuales"*. [cited; Available from: http://www.infor.uva.es/~jsalama1/calsoft/Tema5.pdf.

26. Fuente, A.A.J. *"Necesidad de sistemas formales de métricas para proyectos de software OO."* **Volume**, 12

27. Manso, E., *"Medición en Ingeniería del Software."* p. 34.

28. Wikipedia., L.E.L. *"Calidad"*. 2007 [cited 18 marzo 2007]; Available from: http://www.wikipedia.org/calidad.

29. ISO/IEC. *"Technical Report 9126 - 3. Software engineering - Product quality - Part 3: Internal metrics"*. 2003 [cited 2007 1º julio 2003]; Primera Edición:[

30. Navarro, A., *"Proceso de software y métricas de proyectos."*

31. Wikipedia. (2007) *"Línea de código"*. **Volume**,

32. Arregui., J.J.O., *"Revisión sistemática de métricas de diseño orientado a objetos."* 2005, Universidad politécnica de Madrid.: Madrid. p. 12.

33. Lic. Daina Ivette Reynoso, I.H.E.B., Ing. Igor Martínez. *"Métricas en ingeniería de software."* [cited 18 marzo 2007]; Available from: http://www.it.aut.uah.es/juanra/docencia/GestiondeProyectos/traspas/Metricas.pdf.

34. Humphrey, W.S., *"Introduction to the Team Software Process (sm)."* 2000: Addison - Wesley.

35. Interamericana., U.A. *"Trabajo de campo 1. Segundo Parcial."* 2002 [cited 18 abril 2007]; Available from:

http://www.altillo.com/EXAMENES/uai/ingsistemas/trabajocampo/trabajocampo2005p2.asp.

36. Soldado, R.M. *"Métricas aplicables al Diseño Orientado a Objetos"*. 2000 [cited 2007 jueves, 18 enero 2007].

GLOSARIO DE TÉRMINOS

Acoplamiento: dependencia entre clases, paquetes, etc… Según Bunge, el acoplamiento entre clases u objetos se produce cuando uno actúa sobre otro o interfiere en su historia, es decir, un método usa a otro o a otro atributo.

ADA: Es un preprocesador que adiciona los conceptos de clases y herencia. No es considerado como un lenguaje OO.

ALBET: Empresa de Comercio Exterior. Sociedad Mercantil Cubana, para la UCI. Creada en 2005. Es la representación comercial de todos los proyectos de exportación coordinados a través de la Infraestructura Productiva. Integra esfuerzos de múltiples organizaciones y empresas para garantizar las soluciones tecnológicas integrales.

Atributo: Una propiedad física o abstracta mensurable de una entidad. Pueden ser internos o externos.

BSI: British Standards Institution.

CMM: Capability Maturity Model.

CMMI: Capability Maturity Model Integration.

COCOMO: COnstructive COst MOdel.

Cohesión: Ocurre cuando los elementos de una clase están fuertemente relacionados. Por ejemplo, la cohesión entre dos métodos de una misma clase lo constituye el conjunto de atributos comunes a ambos.

Complejidad: Número de elementos y relaciones entre ellos.

CRC: El modelado de clases-responsabilidades-colaboraciones (CRC) aporta un medio sencillo de identificar y organizar las clases que resulten relevantes al sistema o

requisitos del producto. Un modelo CRC es realmente una colección de tarjetas índice estándar que representan clases. Las tarjetas están divididas en tres secciones. A lo largo de la cabecera de la tarjeta usted escribe el nombre de la clase. En el cuerpo se listan las responsabilidades de la clase a la izquierda y a la derecha los colaboradores. El caso es desarrollar una representación organizada de las clases. Las responsabilidades son los atributos y operaciones relevantes para la clase. Puesto de forma simple, una responsabilidad es cualquier cosa que conoce o hace la clase. Los colaboradores son aquellas clases necesarias para proveer a una clase con la información necesaria para completar una responsabilidad. En general, una colaboración implica una solicitud de información o una solicitud de alguna acción.

DSDM: Dynamic Systems Development Method.

EED: Eficacia en la Eliminación de Defectos.

IEC: Internacional Electrotechnical Commission.

IEEE: The Institute of Electrical and Electronics Engineers, el Instituto de Ingeniería Eléctrica y Electrónica, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

IGU: Interfaz Gráfica de Usuario. También conocida como Graphic User Interface (GUI por sus siglas en inglés).

Indicador: soporte de información (habitualmente expresión numérica) que representa una magnitud, que a través del análisis del mismo se permita la toma de decisiones sobre los parámetros de actuación (variables de control) asociados.

ISO: International Organization for Standardization.

MOOD: Metrics for Object Oriented Design

Moprosoft: Modelo de Procesos para la Industria del Software.

MSF: Microsoft Solutions Framework. Modelo para diseñar aplicaciones para Microsoft.

NASA: National Aeronautics and Space Administration.

PSP: Personal Software Process.

RUP: Rational Unified Process.

SEI: Software Engineering Institute.

SPICE: Software Process Improvement and Capability dEtermination.

SQA: Software Quality Assurance. Garantía de la Calidad del Software.

TickIT: Programa de certificación de calidad para software creado por BSI.

TSP: Team Software Process.

UML: Unified Modeling Language.

UNAM: Universidad Nacional Autónoma de México.

XP: eXtreme Programming.