TESIS

Umbrales para Métricas Orientadas a Objetos

Tesista: Ing. Pablo Ariel Negro

Director: Dra. Roxana Giandini

Master en Tecnología Informática

Facultad de Tecnología Informática

Universidad Abierta Interamericana

Abril de 2008

Resumen

Una aplicación practica de las métricas orientadas a objetos (OO), es predecir que clases tienen una alta probabilidad de contener defectos. Esto tiende a ser significativo dado que, se cree que las métricas OO son indicadores de complejidad psicológica y, las clases que son más complejas son más probables que contengan defectos. Recientemente se ha propuesto una teoría cognitiva la cual sugiere que existe un efecto de umbral para varias métricas OO. Esto significa que las clases OO son fáciles de entender, mientras su complejidad este por debajo del valor de umbral. Por encima del valor de umbral, su comprensión decrece, llevando a una probabilidad de fallas incremental. Acorde a esta teoría, esto sucede debido a que la memoria humana a corto plazo colapsa. Si esta teoría se confirma, proveería un mecanismo que podría explicar la introducción de fallas en sistemas OO, y proveería también una guía practica de cómo diseñar sistemas OO. En este trabajo se testea empíricamente esta teoría sobre dos sistemas de mercado electrónico. Se testeó el efecto de umbral sobre la suite de métricas de Chidamber & Kemerer (CK). Se utilizo como variable dependiente la incidencia a fallas. Los resultados indican que no existe un efecto de umbral para las métricas estudiadas, donde la propensión a fallas cambie de ser estable a incrementarse rápidamente. Los resultados son consistentes para ambos sistemas. Por lo tanto, no se puede proveer soporte a la teoría cognitiva presentada.

Palabras Clave: Métricas Orientadas a Objetos, calidad de software, modelos de Calidad, Complejidad de software, Umbrales.

TESIS

INDICE TEMATICO

Tema *Página*

CAPITULO 7 – CASO DE ESTUDIO

CAPITULO 8 – CONCLUSIONES

CAPITULO 1 INTRODUCCIÓN

La medición es un elemento clave en cualquier proceso de ingeniería. Las medidas se emplean para comprender mejor los atributos de los modelos que se crean, y evaluar la calidad de los productos de ingeniería o de los sistemas que se construyen. Pero a diferencia de otras disciplinas de ingeniería, la de software, no se basa en las leyes cuantitativas básicas de la física [22]. Las medidas directas, como el voltaje, la masa, la velocidad o la temperatura, no son comunes en el mundo del software. Debido a que las medidas y métricas de software suelen ser indirectas, están siempre expuestas a debate.

La medición es el proceso mediante el cual se asignan números o símbolos a los atributos de entidades reales para definirlas de acuerdo a reglas claramente establecidas [16]. Hoy en día podemos medir atributos que se consideraban imposibles de medir. Por supuesto estas mediciones no tienen el mismo refinamiento que casi todas las de las ciencias físicas, pero existen, y muchas decisiones importantes se toman en base a ellas. La necesidad de *"medir lo inmedible"* para mejorar nuestra comprensión de entidades particulares es tan importante en la ingeniería de software como en cualquier otra disciplina.

La medición permite obtener una visión del proceso y el proyecto dado que proporciona un mecanismo para lograr una evaluación. Cuando se puede medir aquello de lo que se esta hablando y expresarlo en números, se sabe algo acerca de ello; pero cuando no puede medir, cuando no puede expresarse en números, el conocimiento es escaso, deficiente; puede ser el comienzo del conocimiento [22].

La medición se aplica al proceso de software con la finalidad de mejorarlo de manera continua. La medición se utiliza a lo largo de un proyecto de software como apoyo en la estimación, el control de calidad, la valoración de la productividad y el control del proyecto. Finalmente, la medición es aplicada por los ingenieros de software como auxiliar en la evaluación de calidad de los productos de trabajo, y para apoyar la toma de dediciones táctica, conforme el proyecto avanza [16].

1.1 INTRODUCCION A LA HIPOTESIS

Como veremos en los capítulos siguientes, existe una gran cantidad de trabajo y literatura referida al diseño y aplicación de métricas tanto para sistemas convencionales, como para sistemas Orientados a Objetos (OO). Cada esquema de métricas y sus refinamientos, se presenta como una suite determinada para medir diferentes aspectos tanto de un sistema, proyecto o un modulo de software tanto como una de clase en OO.

Una vez que el ingeniero de software realiza las mediciones susceptibles a un proyecto dado, y obtiene los valores que arrojan las métricas utilizadas para tal fin, ¿que decisiones puede tomar frente a estos datos?, ¿Cómo puede determinar si estos datos que arrojo la medición, están dentro de valores aceptables?, ¿Requerirán las estructuras medidas, algún tipo de rediseño o refactoring?, ¿Cómo está afectando la complejidad cognoscitiva de las clases, a las características externas? En definitiva, una vez obtenidos estos datos, deseamos saber ***¿que significan?***, para poder tomar acciones correctivas o no, dependiendo de los valores obtenidos.

Presentado el valor 4, como una media promedio de la cantidad de argumentos para una librería de clases, no se debe necesariamente saber que significa, (bueno, malo, no pertinente). Evaluado contra las medidas publicadas de aceptación, o contra las medidas realizadas en proyectos anteriores, el valor de la medida realizada entregara información más significativa [22].

Particularmente significativos son los puntos periféricos: si el valor medio para una propiedad dada es 5 con una desviación normal de 2, y la medición tomada arroja un valor de 10 para un nuevo desarrollo, probablemente valga la pena realizar una verificación posterior, asumiendo por supuesto que hay alguna teoría para apoyar la asunción que la medida es relevante al proyecto [22].

Una vez que se han seleccionado las métricas apropiadas, y que hemos realizado las mediciones pertinentes, necesitamos saber si el aspecto medido de una clase dada se encuentra dentro de una zona segura, o debe marcársela para un análisis mas detallado o incluso ser rediseñada. La pregunta que surge en este momento es, *¿contra que comparar los valores obtenidos que las mediciones han arrojado?* Aquí es donde cobra relevancia el uso de umbrales. Los umbrales se definen como "*Valores heurísticas usados para fijar rangos de valores deseables y no deseables de métricas, para el software medido*". Por lo tanto, si el modelo nos presenta un valor de umbral, tendremos una herramienta de comparación contra el cual validar nuestras mediciones y así, poder tomar decisiones mas precisas respecto de los atributos de calidad de una clase.

En un estudio realizado en [14], se ha presentado evidencia empírica de que no existe relación (valores de umbral) entre el tamaño[1] de un componente de software (Clase, métodos, etc.), y la propensión a errores. En el presente estudio, se analiza si existen valores de umbral, realizado el análisis con métricas de complejidad[2]; Es decir que se analiza si existe alguna relación entre la complejidad de una clase, y la propensión a fallas.

1.2 OBJETIVOS DE LA TESIS

La investigación busca determinar, mediante el análisis y desarrollo estadístico, valores de umbral que sirvan como puntos de referencia respecto de las mediciones realizadas, con el objetivo de proveer al ingeniero de software una herramienta de valoración, que lo asista a la hora de decidir, y dependiendo del aspecto medido, si un componente de software necesita algún tipo de rediseño, o en la indicación de clases con propensión a fallas, o de aquellas que tengan un grado elevado de complejidad cognoscitiva.

Como se menciono anteriormente, en el presente estudio se analiza si existen valores de umbral, realizando el análisis con métricas de complejidad; Es decir que se analiza si existe alguna relación entre la complejidad en la construcción lógica de una clase, y la propensión a fallas.

Consecuentemente los objetivos del presente trabajo son dos, y se presentan a continuación:

El objetivo principal de la investigación es, determinar si existen umbrales y valores de umbral para la suite de métricas de Chidamber & Kemerer (CK) [6] [7], referidas a la propensión a errores, respecto de la complejidad cognoscitiva de los individuos involucrados en el proceso de desarrollo de software; Es decir, si hay alguna relación entre la propensión a errores y la complejidad de una clase. El método utilizado para llevar a cabo el análisis es la regresión logística (LR).

---

[1] El tamaño general de una clase se puede determinar empleando las medidas siguientes: [16]
El número total de operaciones (tanto operadores heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.
El numero de atributos (tanto heredados como privados de la instancia) que están encapsulados en la clase.
En el estudio realizado en [14], se presentan algunas métricas de tamaño en común, tales como:
Stmts: Numero de declaraciones y sentencias ejecutables en los métodos de una clase. Esto solo puede generalizarse a una cuenta SLOC, simple.
NM (Numero de Métodos): Numero de métodos implementados en la clase
NAI (Numero de Atributos): Numero de atributos en una clase, (excluyendo los heredados).

[2] La complejidad de una operación se calcula empleando cualquiera de las métricas de complejidad propuestas para el software convencional; Estas métricas miden la complejidad en la construcción lógica de un componente de software, sabiendo que, a las operaciones convendría limitarlas a una responsabilidad específica, en donde el diseñador debería esforzarse por mantener el valor de complejidad tan bajo como sea posible.

La regresión logística se utiliza para construir modelos cuando la variable dependiente es binaria, como en nuestro caso.

Se dice que un proceso es binomial cuando sólo tiene dos posibles resultados: *"éxito"* y *"fracaso"*, siendo la probabilidad de cada uno de ellos constante en una serie de repeticiones. El método LR, se explica en detalle en el capitulo 4.

Consecuentemente, seria factible brindar al ingeniero de software una herramienta de comparación y evaluación una vez que haya tomado las medidas pertinentes al dominio del desarrollo que este llevando a cabo. Es decir, toda medida es útil siempre y cuando se la pueda evaluar en comparación con algo. En consecuencia, la determinación de los valores de umbral, estarían aportando el marco de referencia matemático contra el cual podrían compararse las mediciones tomadas, y así, tomar acciones correctivas o no.

Otro de los objetivos de la tesis, es presentar una recopilación y estudio de trabajos realizados en métricas OO existentes. Como veremos en los capítulos 3 y 4, existe una gran cantidad de trabajo y literatura referida al diseño y aplicación de métricas tanto para sistemas convencionales, como para sistemas Orientados a Objetos (OO). Cada esquema de métricas y sus refinamientos, se presentan como una suite determinada para medir diferentes aspectos tanto de un sistema, proyecto o un modulo de software tanto como una clase. Esta recopilación de información relacionada, presenta un amplio panorama del trabajo realizado en las distintas ramas de investigación de métricas y, hasta en algunos casos, los valores de umbral para algunas de ellas, obtenidos de la experiencia previa.

1.3 CONTRIBUCIONES

A continuación se enumeran las implicancias de los resultados presentados.

- Como se presenta en los capítulos siguientes y, dados los datos presentados como evidencia, es claro que no existe un efecto de umbral entre la complejidad cognitiva de una clase y la propensión a errores. Por lo tanto la teoría que declara que la propensión a errores en una clase permanece estable hasta un cierto nivel de complejidad y, una vez que este nivel de complejidad es excedido, la propensión a fallas se incrementa debido a limitaciones en la memoria a corto plazo, queda sin soporte.

- Sin bien no se garantiza que no existe un efecto de umbral, la evidencia de los resultados demuestra lo contrario.

- Aquellos usuarios que derivan o utilizan umbrales de las métricas CK, deberían notar que las clases por debajo del valor de umbral, aun son probables que posean una alta propensión a fallas, aunque quizás no la propensión a fallas más alta.

- Los investigadores que validen las métricas OO, deberían continuar modelando la relación entre las métricas OO, al menos el conjunto de métricas CK, y la propensión a fallas utilizando asunciones de continuidad en lugar de asunciones de umbrales, esto es, conforme la complejidad cognitiva de la clase aumente, también lo hará su propensión a fallas.

- Los resultados presentados, no deberían implicar que la complejidad de una clase, es la única variable que se puede utilizar para predecir propensión a errores para el software OO. Por el contrario, mientras la complejidad de una clase pareciera ser una variable importante, otros factores indudablemente tendrán influencia. Por lo tanto, y con el propósito de construir modelos comprensivos que predigan la propensión a fallas, otras variables deberían ser analizadas.

- Es fundamental reconocer que la formulación de teorías es importante para una disciplina. Las teorías explican los fenómenos que observamos (proveen de un mecanismo). El conocimiento del mecanismo puede potencialmente guiar a avances de campo.

- Finalmente cabe mencionar que ha sido publicado un artículo relacionado al tema de esta tesis, en un congreso nacional con referato. [30]

1.4 ORAGANIZACION DEL TRABAJO

En el capitulo 1, se presentan conceptos básicos de métricas, clasificación y diferentes tipos de enfoques de estas.

En el capitulo 2, se describen los atributos internos y externos, tipos de medidas y el proceso de diseño de una métrica.

En el capitulo 3, se introduce el estado del arte respecto del trabajo realizado en métricas OO.

En el capitulo 4, se presenta en detalle la metodología de investigación de Regresión Logística, utilizada para llevar a cabo el presente estudio.

En el capitulo 5, se introduce el marco teórico de la investigación. Aquí se plantean los diferentes tipos de umbrales y la hipótesis del estudio.

En el capitulo 6, se presenta el caso de estudio, y se presentan los resultados obtenidos como consecuencia de la aplicación del método,

En el capitulo 7, se explica el caso de estudio, y los datos estadísticos arrojados por el mismo.

En el capitulo 8, se presentan las conclusiones y el trabajo fututo.

CAPITULO 2

CONCEPTOS BASICOS DE METRICAS

2. Conceptos básicos de Métricas

Empezaremos por definir los posibles términos que se encuentran encerrados en la palabra métrica, porque es muy común asociarla con las palabras medición y medida, aunque estas tres son distintas. La medición *"es el proceso por el cual los números o símbolos son asignados a atributos o entidades en el mundo real tal como son descritos de acuerdo a reglas claramente definidas"* [9]. Una medida *"proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto"* [16].

El IEEE “Standard Glosary of Software Engering Terms" define el termino métrica como *"una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado"* [16].

Muchos investigadores han intentado desarrollar una sola métrica que proporcione una medida completa de la complejidad del software. Aunque se han propuesto docenas de métricas o medidas, cada una de éstas tiene un punto de vista diferente; y por otro lado, aunque bien se sabe que existe la necesidad de medir y controlar la complejidad del software, es difícil de obtener un solo valor de estas métricas de calidad. Aun así debería ser posible desarrollar medidas de diferentes atributos internos del programa.

Aunque todos estos obstáculos son motivo de preocupación, no son motivo de desprecio hacia las métricas. Por tal motivo se dice que la medición es esencial, si se desea realmente obtener software de calidad. Es por eso que existen distintos tipos de métricas para poder evaluar, mejorar y clasificar al software final, en donde serán manejadas dependiendo del entorno de desarrollo del software al cual pretendan orientarse.

2.1 ¿Qué son las métricas de software?

Michael [16] define las métricas de software como *"La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos, para suministrar información relevante a tiempo, de esta forma el administrador junto con el empleo de estás técnicas, mejorará el proceso y sus productos"*. Las métricas de software proveen la información necesaria para la toma de decisiones técnicas. En la figura 2.1 se ilustra una extensión de esta definición para incluir los servicios relacionados al software como la respuesta a los resultados del cliente.

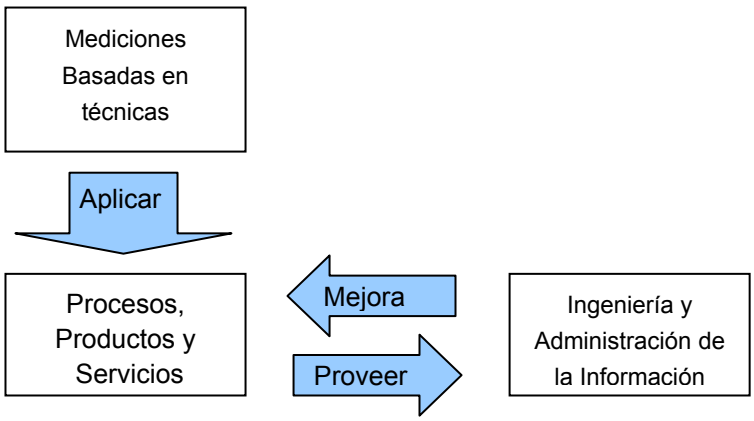

Figura 2.1 Servicios Relacionados al Software [16]

Las métricas son la maduración de una disciplina, que, según Pressman [16] darían soporte en:

- La evaluación de los modelos de análisis y de diseño,
- Proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente.
- Ayudaran en el diseño de pruebas más efectivas.

Por tal motivo propone un proceso de medición, el cual se puede caracterizar por cinco actividades:

1- *Formulación:* La obtención de medidas y métricas de software apropiadas para la representación del software en cuestión.

2- *Recolección:* El mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.

3- *Análisis:* El cálculo de las métricas y la aplicación de herramientas matemáticas.

4- *Interpretación:* La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.

5- *Realimentación:* Recomendaciones obtenidas de la interpretación de métricas técnicas trasmitidas al equipo de software.

Se conoce que no existe un cuerpo de principios en conjunto, que puedan dirigir al desarrollo de métricas de software a que sean independientes del lenguaje, a ambientes y a metodologías de programación. Matemáticamente, estos principios son teorías e implementaciones críticas ya que, una métrica, tiene ciertas propiedades matemáticas y atributos de ingeniería, así como también ciertas realimentaciones de productividad. Por tal motivo se hace necesario responder a tres preguntas fundamentales, respecto de los aspectos deseables en una métrica [9].

- ¿Cuánto mide? ➔ complejidad en la medida
- ¿Qué tan bien mide? ➔ calidad en la medida
- ¿Qué tanto tiempo mide? ➔ predicción

Las métricas de software incluyen otras varias actividades, tales como [16]:

- Estimación de costo y de esfuerzo.
- Medición de la productividad.
- Acumulación de datos.
- Realización de modelos y mediciones de la calidad.
- Elaboración de modelos de seguridad.
- Evaluación y modelos de desempeño.
- Valoración de las capacidades y de la madurez.
- Administración por métricas.
- Evaluación del método y herramientas.

2.2 Clasificación de Métricas

La clasificación de una métrica de software refleja o describe la conducta del software. A continuación se muestra una breve clasificación de métricas de software, descritas en [16]:

Métricas de complejidad:

Son todas las métricas de software que definen de una u otra forma la medición de la complejidad; Tales como volumen, tamaño, anidaciones, costo (estimación), agregación, configuración, y flujo. Estas son los puntos críticos de la concepción, viabilidad, análisis, y diseño de software.

Métricas de calidad:

Son todas las métricas de software que definen de una u otra forma la calidad del software; Tales como exactitud, estructuración o modularidad, pruebas, mantenimiento, reusabilidad, cohesión del módulo, acoplamiento del módulo, etc. Estas son los puntos críticos en el diseño, codificación, pruebas y mantenimiento.

Métricas de competencia:

Son todas las métricas que intentan valorar o medir las actividades de productividad de los programadores o practicantes con respecto a su certeza, rapidez, eficiencia y competencia. No se ha alcanzado mucho en esta área, a pesar de la intensa investigación académica.

Métricas de desempeño:

Corresponden a las métricas que miden la conducta de módulos y sistemas de un software, bajo la supervisión del sistema operativo o hardware. Generalmente tienen que ver con la eficiencia de ejecución, tiempo, almacenamiento, complejidad de algoritmos computacionales, etc.

Métricas estilizadas:

Son las métricas de experimentación y de preferencia; Por ejemplo: estilo de código, identación, las convenciones en la denominación de datos, las limitaciones, etc. Pero estas no se deben confundir con las métricas de calidad o complejidad.

Variedad de métricas:

Tales como portabilidad, facilidad de localización, consistencia. Existen pocas investigaciones dentro del área.

Estas clasificaciones de métricas fortalecen la idea, de que más de una métrica puede ser deseable para valorar la complejidad y la calidad del software.

2.3 Diferentes enfoques de métricas

Se han propuesto cientos de métricas para el software, pero no todas proporcionan suficiente soporte práctico para su desarrollo. Algunas demandan mediciones que son demasiado complejas, otras son tan esotéricas que pocos profesionales tienen la esperanza de entenderlas, y otras violan las nociones básicas intuitivas de lo que realmente es el software de alta calidad. Es por eso que se han definido una serie de atributos que deben acompañar a las métricas efectivas de software, por lo tanto la métrica obtenida y las medidas que conducen a ello deben cumplir con las siguientes características fundamentales [16]:

Simple y fácil de calcular:

Debería ser relativamente fácil de aprender a obtener la métrica y su cálculo no obligara a un esfuerzo o a una cantidad de tiempo inusuales.

Empírica e intuitivamente persuasiva:

La métrica debería satisfacer las nociones intuitivas del ingeniero de software sobre el atributo del producto en cuestión (por ejemplo: una métrica que mide la cohesión de un módulo debería aumentar su valor a medida que crece el nivel de cohesión).

Consistente en el empleo de unidades y tamaños:

El cálculo matemático de la métrica debería utilizar medidas que no lleven a extrañas combinaciones de unidades. Por ejemplo, multiplicando el número de personas de un equipo por las variables del lenguaje de programación en el programa, resulta una mezcla sospechosa de unidades que no son intuitivamente concluyentes.

Independiente del lenguaje de programación:

Las métricas deberían apoyarse en el modelo de análisis, modelo de diseño o en la propia estructura del programa. No deberían depender de los caprichos de la sintaxis o semántica

Un mecanismo eficaz:

Para la realimentación de calidad: la métrica debería suministrar al desarrollador de software información que le lleve a un producto final de superior calidad.

No obstante que la mayoría de las métricas de software compensan las características anteriores, algunas de las métricas usualmente empleadas no cumplen

algunas de las características antes mencionadas. Un ejemplo es el punto funcional (PF), en donde se consigue argumentar que el atributo es *consistente* y *objetivo* pero falla por que un equipo ajeno independiente puede no ser capaz de conseguir el mismo valor de PF que otro equipo que emplee la misma información del software. En tal caso la siguiente pregunta que se manifiesta, ¿Debería entonces rechazarse la medida PF? La respuesta es por supuesto que No. La PF aporta una visión interna útil y por lo tanto provee de un valor claro, incluso si no satisface un atributo perfectamente. Este es un ejemplo en donde el uso de las métricas depende de los factores individuales del desarrollador o desarrolladores del software.

2.4 Las métricas en el proceso y dominio del proyecto

Las métricas de *proceso* de software se emplean para fines estratégicos, y las métricas del *proyecto* de software son tácticas. Estas últimas van a permitir proporcionar al desarrollador de proyectos de software, una evaluación del proyecto que sigue en continuo desarrollo, equivalentemente podrá ver los defectos que logren provocar riesgos a largo plazo (áreas problema); y observar si el área de trabajo (equipo) y las distintas tareas se ajustarán [16].

Existe una preocupación en la unificación de las métricas dentro del proceso del software, dado que los desarrolladores de software no tienen una cultura de medición marcada. Es por eso que el uso de métricas demanda un cambio cultural, tanto en el recopilado de datos (investigación histórica), como en el cálculo de métricas (LDC, PF, métricas de calidad y orientadas a objetos), y asimismo en la evaluación (resultados obtenido), para poder comenzar un programa de bases de métricas y justamente obtener una guía, asimismo una base de datos tanto de los procesos como de los productos, y de este modo adquirir una visión más profunda de lo que se está elaborando.

Las métricas de software nos aportan una manera de estimar la calidad de los atributos internos del producto, permitiendo así al ingeniero de software valorar la calidad antes de construir el producto, consecuentemente el tiempo invertido será identificando, examinando y administrando el riesgo. Este esfuerzo merece la pena por muchas razones, ya que habrá una disminución de inconvenientes durante el proyecto, y asimismo se podrán desarrollar las capacidades de una administracion proactiva pudiendo seguir y controlar el proyecto. De esta manera se pueden planificar alternativas a posibles problemas que puedan surgir. Además, conseguiremos absorber una cantidad significativa del esfuerzo en la planificación.

Del mismo modo existen diferentes tipos de métricas para poder evaluar, mejorar y clasificar al software desde sus inicios hasta el producto final, las cuales se verán en los capítulos siguientes.

CAPITULO 3 METRICAS TECNICAS

Cualquier cosa que queramos medir o predecir en un software es un *atributo* de cualquier *entidad* de un producto, proceso o recurso asociado a éste. Cada entidad de software tiene varios atributos que pueden ser medidos. Es por eso que se necesita hacer una distinción entre atributos que son internos o externos y medidas directas e indirectas.

3.1 Atributos Internos y Atributos Externos

Los *atributos internos* de un producto, proceso o recurso son aquellos que podemos medir puramente en términos del producto, proceso o recurso del mismo. Pueden ser medidos directamente. Por ejemplo: la longitud de un programa o el tiempo transcurrido en la ejecución de cualquier modulo de software [9].

Los *atributos externos* de un producto, proceso o recurso son aquellos que solamente pueden ser medidos con respecto a cómo el producto, proceso o recurso se relacionan con su entorno. Estos tienden a ser los que el administrador y el usuario del software comúnmente gustan de medir y predecir. Por ejemplo, al administrador de software le gustaría saber el costo de eficacia de algunos procesos o de la productividad de su personal, mientras que a los usuarios les gustaría conocer la usabilidad, fiabilidad, o portabilidad de un sistema que están evaluando para ser adquirido [16]. Desgraciadamente los atributos externos son los más difíciles de medir, porque estos no pueden ser medidos directamente [9]. En la tabla 3.1 se describe la estructura, y se dan ejemplos de varios tipos de atributos.

3.2 Medidas Directas y Medidas Indirectas

La *medida directa* de un atributo es aquella en donde no se depende de cualquier otro atributo [9].

La *medida Indirecta* de un atributo es aquella en la que se involucra la medición de uno o más atributos [9].

3.3 El Reto de las Métricas Técnicas

Muchos investigadores han intentado desarrollar una sola métrica que facilite una medida completa de la complejidad del software. Aunque se han presentado docenas de medidas de complejidad, cada una tiene un punto de vista distinto de lo que es la complejidad en si y de qué atributos de un sistema lo hacen a este complejo. Comparemos con una métrica para evaluar un automóvil. Algunos observadores podrían hacer énfasis en el diseño de la cabina, otros podrían hacer hincapié en las características mecánicas, otros podrían considerar el precio, o el rendimiento, o la economía de consumo o la capacidad de reutilizarlo cuando se vaya a desechar. Como cualquiera de estas características puede competir con las otras, es difícil obtener un solo valor del “atractivo” del automóvil. Lo mismo sucede con el software.

Tabla 3.1 Estructura de Atributos [9]

Entidades	Atributos	
Recursos	**Internos**	**Externos**
Personal	Edad, precio, ..	Productividad, experiencia, inteligencia
Equipo	Tamaño, estructuras ..	Productividad, calidad
Software	Precio, tamaño ..	Usabilidad, seguridad
Hardware	precio, velocidad, tamaño de memoria	Seguridad
Oficinas	tamaño, temperatura, luz, o'	Confort, calidad
….	….	…..
Procesos		
Construcción de especificaciones	Tiempo, esfuerzo, numero de cambios	Calidad, costo, estabilidad
Diseño detallado	Tiempo, esfuerzo,..	Costo, costo efectivo
Prueba	número de errores de código encotrados, tiempo	Estabilidad, costo, costo efectivo
…..	…..	…..
Productos		
Especificaciones	Tamaño, usabilidad, modularidad, funcionalidad, ..	Comprensibilidad, mantenimiento
Diseños	Acoplamiento, modularidad, tamaño,. Usabilidad	Calidad, complejidad, mantenimiento
Código	Funcionalidad, complejidad algoritmica, control de flujo	Seguridad, usabilidad, mantenimiento
Datos de prueba	Tamaño, nivel de proteccion	Calidad

Se ha producido una gran cantidad de literatura sobre métricas de software y es común la crítica de algunas de éstas (incluyendo algunas de las presentadas en este documento). Sin embargo, muchas de las críticas se centralizan en aspectos esotéricos y pierden el objetivo primario de la medición en el mundo real, que es ayudar al ingeniero de software a establecer una manera sistemática y objetiva de conseguir una visión interna de su trabajo y mejorar la calidad del producto como resultado.

Sin embargo sigue existiendo la necesidad de medir y controlar la complejidad del software, y aunque es difícil obtener un solo valor de esta "métrica de calidad", debería ser posible desarrollar medidas de diferentes atributos internos del programa (por ejemplo: modularidad efectiva, independencia funcional y otros atributos). Las métricas y las medidas conseguidas de ellas pueden usarse como guías independientes de la calidad de los modelos de análisis y de diseño. Pero aquí también surgen problemas. Fenton [9] lo advierte cuando dice: *"El peligro de intentar encontrar medidas que caractericen tantos atributos diferentes es que inevitablemente las medidas tienen que satisfacer objetivos incompatibles. Esto es contrario a la teoría de representación de la medición".*

Aunque la declaración de Fenton [9] es correcta, mucha gente cuestiona que la medición técnica llevada a cabo en las primeras fases del proceso del software les proporcione a los desarrolladores de software un mecanismo consistente y objetivo para valorar la calidad. No obstante conviene preguntarse, ¿Qué validez tienen las métricas técnicas? Es decir, ¿Cómo están relacionadas las métricas técnicas con la fiabilidad y la calidad a largo plazo de un sistema basado en computadora?, Fenton [9]

comenta que, a pesar de las conexiones intuitivas entre la estructura interna de los productos de software (métricas técnicas), sus productos externos y los atributos del proceso, ha habido de hecho, muy pocos intentos científicos para establecer relaciones específicas.

3.4 Identificación del Usuario

En el desarrollo de métricas se requiere identificar claramente al usuario. Lo primero que debemos hacer para identificar a un usuario es preguntamos:

- ¿Quién necesita la información?
- ¿Quién va a emplear la métrica?
- Si la métrica no tiene un usuario, ¿no utilizarla?

Dado el conjunto de preguntas anteriores, podemos observar que la selección de una métrica empieza con la identificación de los programas que el usuario quiere medir.

¿Quién necesita la información? Los requerimientos del usuario determinan el programa de métricas. Si el programa de métricas no tiene a un usuario se recomienda no establecer dicho programa.

3.5 Diseño de métricas

A continuación se plantean los pasos para el diseño métricas [16]:

Paso 1: Definiciones claras

El primer paso en el diseño de una métrica es dar una definición estándar para las entidades y los atributos a ser medidos. Cuando empleamos términos como *"errores"*, *"defectos", "problema reportado"*, *"tamaño"* y aún *"proyecto"*, otras personas podrían interpretar estos conceptos, dentro de su propio contexto, con significados que pueden variar de nuestra intención al definirlos. Estas interpretaciones aumentan sus diferencias cuando se emplean términos más ambiguos como: Calidad, mantenibilidad, amigabilidad al usuario, entre otros.

Los ingenieros, administradores u otras personas involucradas en el proyecto pueden emplear diferentes términos para la misma cosa. Por ejemplo, el término defecto reportado, problema reportado, incidente reportado, falla reportada, o reporte de llamada del cliente pueden ser empleados por varias organizaciones para dar significado a la misma cosa. Pero desgraciadamente, pueden también referirse a distintas entidades. Para evitar estos problemas se sugiere:

- Adoptar definiciones estándares y escribirlas
- Aplicarlas con consistencia
- Usar las sugerencias de la industria
- Escoger definiciones que cumplan los objetivos organizacionales

Desafortunadamente, hay muy pocos estándares en la industria para la mayoría de las definiciones de los atributos del software. Cada uno cuenta con una opinión y este debate podría continuar por muchos años. Al abordar este problema se sugieren adoptar definiciones estándar hacia dentro de la organización y aplicarlos consistentemente. Se pueden usar sugerencias de la industria como base para comenzar, seleccionar las definiciones que cumplan con los objetivos de la

organización y emplearlos para la creación de definiciones propias [16].

Paso 2: Definir el modelo

El siguiente paso es definir un modelo para la métrica. En términos sencillos, el modelo define cómo calcular la métrica. Por ejemplo, para la Inspección del código de las métricas primitivas, se puede usar los siguientes modelos [9]:

- Líneas de código inspeccionadas = loc
- Horas empleadas en la preparación = hrs-prep
- Medida de preparación = loc / hrs-prep

La mayoría de los modelos incluyen un elemento de simplificación. Cuando creamos un modelo de medición de software necesitamos ser pragmáticos. Si tratamos de incluir todos los elementos que afectan al atributo o que caracterizan a la entidad, el modelo utilizado llegará a ser demasiado complicado. Ser pragmático significa no tratar de crear un modelo perfecto. Tomar los aspectos más importantes. Recordar que el modelo puede ser siempre modificado para incluir mas niveles de detalle en el futuro. Hay dos métodos para la selección de un modelo.

En muchos casos no es necesario "re-inventar la rueda". Hay muchos modelos de métricas de software que son empleados exitosamente por otras organizaciones. Pero desafortunadamente no se sabe con certeza cual es el grado de "privacidad" que estas requieren.

El segundo método es crear un modelo propio. El mejor consejo es hablar con la gente que actualmente es responsable del producto, de los recursos o con quienes están involucrados en el proceso. Ellos son los expertos y por lo tanto conocen que factores son importantes.

Para ilustrar la selección de un modelo, vamos a considerar una métrica para la duración de caídas no planeadas de los sistemas. Si estamos evaluando un sistema de software instalado en un solo servidor, un modelo sencillo tal como los minutos de caída por mes pueden ser suficientes. Si el objetivo es comparar diferentes versiones del software instalado en varios servidores, se puede seleccionar un modelo como minutos de caída cada 100 meses de operación o, si queremos enfocarnos en el impacto en el cliente, se debe seleccionar minutos de caída al año por "servidor". Se presenta detalladamente, el ejemplo dado de la selección de un modelo [16].

Duración de caídas no planeadas de sistemas

- Minutos de caídas por mes
- Minutos de caídas /100 meses de operación
- Minutos de caída por "servidor'" por año

Paso 3: Establecer un criterio de conteo

El siguiente paso en el diseño de una métrica es dividir al modelo en sus métricas primitivas (cuantificables en forma directa) y la definición del criterio de conteo para medir cada primitiva. En este paso se definirá el sistema de mapeo para la medición de cada métrica primitiva.

Las métricas primitivas y su criterio de conteo definen el primer nivel de los datos que necesitan ser recolectados para implantar la métrica. Para ejemplificar esto se va a usar el modelo de minutos de caída por servidor por año. Una de las métricas primitivas para este modelo es el número de servidores. Al principio, el conteo de esta primitiva parece sencillo. Pero cuando consideramos la dinámica de añadir nuevos servidores o la instalación de software nuevo en servidores existentes, el criterio de conteo se vuelve más complejo. Así que si empleamos el número de servidores del último día del periodo o ¿calcular un número promedio de servidores para el periodo?. De cualquier forma, necesitamos recolectar datos como la fecha en que el sistema fue instalado en el servidor y si queremos comparar diferentes versiones del software, es necesario recolectar la información de las versiones que fueron instaladas en cada servidor, y cuando cada una fue instalada. Se presenta detalladamente, el ejemplo dado de criterio de conteo [16]:

- Número de "servidores" al final de un periodo
- Número de "servidores" al inicio + Número de "servidores" al final / 2.
- Ó ("servidor" en cada día) / número de días

Paso 4: Decidir ¿Qué es bueno?

El cuarto paso en el diseño de una métrica es definir *¿Qué es bueno?* Una vez que se ha decidido que medir y como medirlo, se debe decidir que hacer con el resultado. ¿Es 10 demasiado poco o 100 es mucho? ¿La tendencia debería ser hacia arriba o hacia abajo?

Uno de los primero lugares para empezar a ver *¿Qué es bueno?*, es con los clientes. Muchas veces, los requerimientos del usuario determinan ciertos valores de algunas métricas. Otra fuente de información es la literatura de métricas. Varias investigaciones y estudios han ayudado al establecimiento de normas aceptadas por la industria para mediciones estándares. Si no están disponibles datos históricos, se recomienda esperar hasta recolectar suficiente información para el establecimiento de valores actuales.

Una vez que se ha decidido *¿Qué es bueno?*, se pueden determinar en todo caso que acciones son necesarias. Si se está "excediendo" los valores deseados, o si las acciones correctivas no son necesarias. Para establecer el manejo y monitoreo de actividades de mejoramiento, se determinará en las métricas un ambiente en donde se deberá tener presente cuatro metas:

- La meta debe ser razonable; Es correcto establecer metas que se extienden, pero si ésta es irreal, será ignorada.
- La meta debe estar asociada a un marco de tiempo.
- La meta debe fundamentarse en acciones sustentadas. Por ejemplo podría ser razonable establecer una meta de un incremento del 50% en la solución de errores encontrados, si se crea un equipo especial que tendrá como actividad la solución de errores detectados.
- La meta debe ser dividida en partes. Por ejemplo si se emplea un año para alcanzar el mejoramiento, seria más simple si la meta en cuestión se divide en 12 pequeñas metas que se establezcan por mes, y así las acciones serán más probables de ser realizadas (definición de hitos).

Paso 5: Reporte de la métrica

El siguiente paso es decidir como reportar la métrica. Esto incluye la definición del formato del reporte, la obtención de los datos, mecanismos de reporte de distribución y disponibilidad. El formato del reporte se diseña de acuerdo a lo que se quiera dar a conocer, es por eso que se debe preguntar lo siguiente, ¿Está la métrica incluida en una tabla con otras métricas para un periodo?, ¿Se añade como último valor en una gráfica de tendencia que muestran los valores de la métrica para varios periodos? Esta gráfica de tendencia ¿Puede ser de barra, línea o de área?, ¿Es mejor comparar valores empleando gráficos de barra o diagramas de torta? ¿Es mejor hacer tablas y gráficas solas o se agrega texto detallado de análisis y se incluye en el reporte?, ¿Son metas o valores de control los que se incluyen en el reporte?

En el reporte de métricas, se deberán realizar cuatro pasos: ciclo de obtención de datos, ciclo de reporte de datos, mecanismos de reporte, distribución y disponibilidad que se describen a continuación:

- *El ciclo de obtención de datos* define con que frecuencia la métrica requiere de snap-shot(s) de los datos y, en que momento los elementos de los datos están disponibles para el cálculo de la métrica.

- *El ciclo de reporte* define con que frecuencia el reporte es generado y cuando se prepara para su distribución. Por ejemplo la razón principal de un estudio de métricas puede ser por algún evento, como la terminación de una fase en el proceso de desarrollo del software; Otras métricas como el promedio de defectos reportados pueden obtenerse y reportarse diariamente durante las pruebas del sistema, la obtención mensual de datos y el reporte mensual después de la liberación del software.

- *Los mecanismos de reporte* proyectan los mecanismos de entrega de las métricas.
- Al definir *la distribución*, se determina quienes reciben copias del reporte o tienen acceso a la métrica. También aquí se define cualquier restricción al acceso de la métrica, mecanismos de aprobación para añadir y eliminar accesos y cambios en la distribución normal.

Paso 6: Calificadores Adicionales.

El paso final en el diseño de una métrica es determinar calificadores adicionales.
Una buena métrica es una métrica genérica. Esto significa que la métrica es válida para todos los niveles de calificadores que se puedan adicionar.

Los calificadores añadidos proveen la información estadística necesaria para varios puntos de vista de la métrica. Un ejemplo de calificadores adicionales podría ser la duración de caídas no planeadas del sistema, tales como [9] [16]:

- Liberación del producto / Líneas de productos / Producto final
- Clientes / Segmento de negocios
- Tipo de caída / Causa

En el diseño de las métricas para la duración de caídas no planeadas del sistema pueden emplearse calificadores adicionales para observar toda la información, tanto de las líneas de un producto como las de productos individualmente o, la liberación del producto. También se pueden percibir las caídas desde el cliente o desde el segmento de negocios, o las podríamos observar por el tipo o causa.

La razón principal de los calificadores adicionales necesita ser definido como parte del diseño de las métricas, ya que estos determinan el segundo nivel de los requerimientos de la recolección de datos. Ninguna discusión de selección y diseño de métricas de software sería completa sin tener en cuenta de que manera las mediciones afectan a la gente y cómo la gente afecta a las mediciones [16].

El simple hecho de medir afectara el entorno de los individuos a ser medidos. Cuando alguien comienza a ser evaluado a través de la medición, automáticamente asume que tiene importancia. Las personas quieren ser bien vistas, por lo que van a querer que las mediciones sean buenas. Cuando se crea una métrica siempre se decide que ambientes se desea estimular. Consecuentemente se debe tomar el tiempo necesario para que otros ambientes puedan resultar útiles en el empleo de métricas.

La mejor manera de prevenir problemas con el factor humano al trabajar con métricas, es seguir algunas de las siguientes reglas básicas [16]:

No hacer mediciones del individuo: Las mediciones de productividad individual son los ejemplos clásicos de estos errores. Si se midiera la productividad en líneas de código por hora producida, la gente se concentraría en su propio trabajo y excluiría al equipo de trabajo, o se enfocaría en realizar programación con líneas extras de código. Es por eso que es recomendable enfocarse en el proceso y en el producto, y no en la gente.

No ignorar los datos: Un camino seguro para acabar un programa de métricas es olvidar los datos cuando se toman decisiones, ya que estos *"dan sustento a la gente cuando sus reportes emplean información útil a la organización"*. Si las metas que se establecen y se comunican no son respaldadas con acciones entonces la gente en la organización se desempeñará basándose en *el ambiente y no en las metas*.

Nunca emplear únicamente una sola métrica: El software es complejo y multifacético, es por eso que enfocarse en una sola métrica puede causar que el atributo medido mejore a expensas de otros atributos. Lo que debería realizarse, es

Seleccionar las métricas basándose en objetivos: Para tener métricas que cumplan con nuestra necesidad de información, se deben seleccionar una suite de métricas que proporcionen información relevante, en respuesta a las preguntas asociadas a los objetivos de medición.

Proveer retroalimentación: El proveer retroalimentación con regularidad tiene algunos beneficios:

- Mantener enfocada la necesidad de la recolección de los datos, hará que el equipo vea que los datos actuales son utilizados y, por lo tanto, se volverán consientes de la importancia de la recolección.

- Si los miembros de los equipos son informados respecto a como los datos son usados, serán cada vez menos escépticos de la utilidad de estos.
- Al involucrar a los miembros del equipo en el análisis de los datos y en los esfuerzos de mejoramiento del proceso, el proceso se beneficiara de sus conocimientos y experiencia.
- La retroalimentación en la recolección de datos y en el resultado íntegro ayudará en la responsabilidad de la recolección de los datos, dando como resultado datos más exactos, consistentes y a tiempo.

Obtener buy-In: Para tener compromiso en las metas como en las métricas, los miembros del equipo necesitan tener un sentimiento de propiedad, es por eso que el participar en la definición de las métricas acrecentara este sentimiento de propiedad. La gente que trabaja con un proceso a diario tiene un conocimiento intimo del proceso, esto da una perspectiva valiosa de cómo el proceso se puede medir mejor y como se pueden interpretar los resultados de las mediciones para maximizar su utilización.

3.6 Mediciones de software

Para medir algo es necesario saber que entidades serán medidas y tener una idea de los atributos (propiedades) de la entidad. Primero se debe identificar el atributo a medir y su significado de medición. Para tal propósito podemos comenzar acumulando datos.

Llevar a cabo el análisis de los resultados de estos procesos, normalmente permite la clarificación y la re-valuación de los atributos.

3.6.1 Métricas Orientadas al Tamaño

Las métricas de software orientadas al tamaño provienen de la normalización de las medidas de calidad y/o productividad considerando el "tamaño" del software que se haya producido. Si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño, como la que muestra la tabla 3.2 [16], que lista cada proyecto de desarrollo de software y las medidas correspondientes de cada proyecto.

Refiriéndonos a la entrada de la figura 2.1 del proyecto *alfa:* se desarrollaron 12.100 líneas de código (LDC) con 24 personas-mes y con un costo de $168.000.

Debe tenerse en cuenta que el esfuerzo y el costo registrados en la tabla incluyen todas las actividades de ingeniería del software (análisis, diseño, codificación y prueba) y no sólo la codificación. Otra información sobre el proyecto alfa indica que se desarrollaron 365 páginas de documentación, se registraron 134 errores antes de que el software se entregara al cliente, y se encontraron 29 defectos después de la entrega del sistema al cliente, dentro del primer año de utilización.
También sabemos que trabajaron tres personas en el desarrollo del proyecto alfa.

Tabla 3.2 Tabla de datos orientados al tamaño [Pressman] [16]

Proyecto	LDC	Esfuerzo	$	Pp doc	errores	Defectos	Personas
Alfa	12.100	24	168	365	134	29	3
Beta	27.200	62	440	1.224	321	86	5
Gamma	20.200	43	314	1.050	256	64	6

Para desarrollar métricas que se puedan comparar entre distintos proyectos, se seleccionan las líneas de código como valor de normalización. Con los datos

elementales contenidos en la tabla se puede desarrollar para cada proyecto un conjunto de métricas simples orientadas al tamaño, tales como:

- errores por KLDC (miles de líneas de código, Kilo LDC)
- defectos por KLDC
- $ por LDC
- páginas de documentación por KLDC

Además, se pueden calcular otras métricas interesantes:

- Productividad = KLDC/ persona-mes
- Calidad = errores / KLDC
- Documentación = páginas de documentación / KLDC

Las métricas orientadas al tamaño no están aceptadas universalmente como el mejor modo de medir el proceso de desarrollo del software. La mayor parte de la discusión gira en torno al uso de las líneas de código mostradas en la tabla 3.3 (LDC) como medida clave. Los defensores de la medida LDC afirman que la LDC es un "artificio" que se puede calcular fácilmente para todos los proyectos de desarrollo de software, que muchos modelos de estimación del software existente utilizan LDC o KLDC como clave de entrada. En el lado opuesto los ofensores defienden que las medidas en LDC son dependientes del lenguaje de programación, que perjudican a los programas más cortos, pero bien diseñados, que no pueden acomodar fácilmente lenguajes procedimentales, y que su uso en estimación requiere un nivel de detalle que pude resultar difícil de alcanzar (es decir, el planificador debe estimar las LDC a producirse mucho antes de que se complete el análisis y el diseño)

Tabla 3.3 Estimaciones Informales del número medio de LDC [Pressman] [16]

Lenguaje de Programación	***LDC/PF (media)***
Asembler	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Lenguajes Orientados a Objetos	30
Lenguajes 4 GL	20
Generadores de Código	15
Hojas de calculo	6
Lenguajes gráficos (iconos)	4

3.6.2 Métricas Orientadas a la Función

Estas métricas utilizan una medida de la funcionalidad; ésta no se puede medir directamente, se debe derivar indirectamente por medio de medidas directas. Las primeras fueron propuestas por Albrecht, que sugirió una medida llamada *"Punto de Función"* para un sistema de software, la idea es que al examinar la especificación del sistema, estas se derivan con una relación empírica según las medidas contables (directas) del dominio de información del software y las evaluaciones de complejidad de software [16]. El tamaño de la tarea de diseño y desarrollo de un sistema de cómputo es determinado por el producto de tres factores mostrados en la figura 3.1.

Tamaño intrínseco del producto

(Para estudios de productividad)

Tamaño de información procesada	*Factor técnico de complejidad*	*Factores de entorno (medio)*
. entradas . salidas . etc.	. batch y en-línea . fácil uso . etc.	. Proyectos administrativos. Riesgos . Habilidades de las personas, etc. . Métodos, herramientas, lenguajes.

Tamaño total del proyecto

(Para estimar necesidades)

Figura 3.1 Tamaño de diseño y desarrollo de un sistema de cómputo [16]

- *El tamaño de información procesada:* Es una medida de la información procesada y proporcionada por el sistema.

- *Factor técnico de complejidad:* Toma en cuenta la medida de varias técnicas y otros factores implicados en el desarrollo y en el implemento de la información procesada requerida.

- *Factores de entorno (o medio):* Es el grupo de factores que surge del entorno del proyecto típicamente valorado en proyectos con riesgo de medidas. Incluye habilidades, experiencia y motivaciones del personal involucrado y de los métodos, lenguajes y herramientas usadas por el equipo del proyecto.

Nótese que los primeros dos de estos tres factores son intrínsecos al tamaño del sistema, en el sentido que éstos resultan directamente de los requerimientos del sistema que serán entregados al usuario.

El método de Punto de Función ha ganado aceptación en el negocio de sistemas de información, para la evaluación del tamaño del sistema, como un componente de la medida de productividad. Cuando están disponibles datos históricos de productividad, este método puede también utilizarse como ayuda en la estimación de horas/persona. Para estimar propósitos, el tercer grupo de factores del entorno también debe ser tenido en consideración.

El método de Punto de Función (PF) de Allan Albrecht [16], consiste en componentes de un sistema que se clasifican en cinco tipos: *entradas externas* (o lógicas), *salidas, consultas, interfaces externas a otros sistemas,* y los *archivos lógicos internos.* Dependiendo del número de elementos de datos estos se denominan como "simple", "promedio" o "complejo". Cada componente es el número dado de puntos dependiendo en tipo y complejidad (tabla 3.5) y la suma para todos los componentes es expresado en "Puntos funcionales sin ajustar" [16].

Los factores técnicos de complejidad se determinan, estimando el grado de influencia de algunos componentes "características generales de aplicación" (Tabla 3.4). El grado de influencia en la escala va desde de cero (no presente o no influenciada) hasta 5 (influencia fuerte). La suma de las 14 características (mostradas en la tabla 3.4), que es el *Grado Total de Influencia* (DI), se convierte al *Factor Técnico de Complejidad (TCF)* calculándose [16]:

$$TCF = 0.65 + 0.01 * Di \qquad (3.1)$$

El valor de Di, surge de la sumatoria de los valores de ajuste de complejidad *i*

que va de 1 a 14 según las respuestas a las preguntas de tabla 3.4:

Tabla 3.4 Preguntas de Di [16]

Di	*Preguntas*
C1	¿Requiere el sistema copias de seguridad y de recuperación fiables?
C2	¿Se requiere de comunicación de datos?
C3	¿Existen funciones de procesamiento distribuido?
C4	¿Es crítico el rendimiento?
C5	¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
C6	¿Requiere el sistema entrada de datos interactiva?
C7	¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
C8	¿Se actualizan los archivos maestros de forma interactiva?
C9	¿Son complejos las entradas, salidas, archivos o las peticiones?
C10	¿Es complejo el procesamiento interno?
C11	¿Se ha diseñado el código para ser reutilizable?
C12	¿Están incluidas en el diseño la conversión y la instalación?
C13	¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
C14	¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Valores de Di

- No presente o no influencia = 0
- Influencia insignificante o incidental = 1
- Influencia moderada = 2
- Influencia promedio o medio = 3
- Influencia significante = 4
- Influencia esencial o fuerte, a través de = 5

Los valores constantes de la ecuación anterior (3.1) Y los pesos que se aplican a las cuentas de los dominios de información se determinan empíricamente.

El tamaño intrínseco relativo del sistema en Puntos Funcionales FP's se calcula con ayuda de la tabla 3.5, utilizando la siguiente fórmula:

$$FP's = UFP's * TCF \qquad (3.2)$$

Tabla 3.5 Nivel de Información Procesando Funciones [Pressmna'98] [16]

Descripción	*Simple*	*Promedio*	*Complejo*	*Total*
Entradas externas	* 3 =	* 4 =	* 6 =	---
Salidas externas	* 4 =	* 5 =	* 7 =	---
Archivos internos lógicos	* 7 =	* 10 =	* 15 =	---
Archivos de interfaz externa	* 5 =	* 7 =	* 10 =	---
Indagación externas	* 3 =	* 4 =	* 6 =	---

(UFP) Total de Puntos funcionales sin ajustar = ---

Podemos notar que los Puntos Funcionales son por lo tanto números

dimensiónales en una escala arbitraria. Los valores de la información mostrados en la tabla 3.5 se definen a continuación:

- *Entradas Externas:* (o número de entradas de usuario). Se suma cada entrada dada por el usuario, donde nos proporciona distintos datos orientados a la aplicación. Estas se diferencian de las peticiones.

- *Salidas Externas:* (o número de salidas de usuario). Se suma cada salida que le proporcionará al usuario información orientada a la aplicación (informes, pantallas, mensajes de error, etc.). Los elementos de datos particulares de un informe no se cuentan de forma separada.

- *Archivos Internos Lógicos* o *Número de archivos:* Se suma cada archivo maestro lógico (grupo lógico de datos que sean parte de una base de datos o un archivo independiente).

- *Archivos de Interfaz Externa* o *Número de interfaces externas: Se* suman todas las interfaces legibles por la máquina (archivos de datos de cinta o discos, etc.) que se utilizan para transmitir información a otro sistema.

- *Indagaciones externas* o *Número de peticiones de usuario:* La petición es una entrada dada que nos va a producir una respuesta inmediata del software en forma de salida. Las peticiones se cuentan por separado.

- *Total de Puntos funcionales sin ajustar* o *Cuenta-Total:* Es la suma de todas las entradas obtenidas de la tabla 3.5

Cuando se calculan los puntos de función, éstos se utilizan de forma análoga a las LDC (Líneas de Código) para normalizar medidas de productividad, calidad y otros ámbitos de software, como por ejemplo:

- Errores por Puntos de Función.
- Defectos por Puntos de Función.
- Costo (dinero) por Puntos de Función.
- Página de documentación por Puntos de Función.
- Puntos de Función por persona-mes.

Las consideraciones expuestas por Albrecht [16] para proponer los Puntos Funcionales como medidas de tamaño de un sistema son [9] [16]:

- Estas medidas aíslan el tamaño intrínseco del sistema de los factores del medio, facilitando el estudio de factores que influyen en la producción.

- Estas medidas están basadas en las observaciones de los usuarios externos al sistema, y es tecnología independiente.

- Estas medidas pueden determinarse al inicio del ciclo de desarrollo lo que permite utilizar los Puntos Funcionales en la estimación de procesos.

- Los Puntos Funcionales pueden ser entendidos y evaluados por usuarios que no son técnicos.

3.6.3 Medidas de complejidad de Halstead

Las medidas fueron desarrolladas por Maurice Halstead [16] para determinar una medida cuantitativa de la complejidad directa de los operadores y operandos en un módulo de software. Entre las métricas de software más avanzadas, éstos son indicadores fuertes de complejidad de código; se basa ampliamente en la evidencia empírica encontrada en el trabajo del *"Maintainability Index Work"*, pero existe evidencia de que las medidas de Halstead son también útiles durante el desarrollo, para valorar la calidad de código en aplicaciones computablemente densas.

Las medidas de Halstead se basan en cuatro números escalares, que serán derivados directamente del código fuente del programa, tales como:

n1 = el número de operadores distintos
n2 = el número de operandos distintos
N1 = el número total de operandos
N2 = el número total de operadores

De estos números, se derivan cinco medidas (tabla 3.6)

Tabla 3.6 Fórmulas derivadas de los números de escalar de Halstead [16]

Medida	*Símbolo*	*Formula*
Longitud del programa	N	N = N1 + N2
Vocabulario del programa	.n	.n = n1 + n2
Volumen	V	V = N * log2 (n)
Dificultad	D	D = (n 0.5) * (N2 / n2)
Esfuerzo	E	E = D * V

3.7 Paradigma *Meta / Pregunta / Métrica*

El Paradigma Meta / Pregunta / Métrica (Goal-Question-Metric), provee un mecanismo excelente para la definición de un programa de medición basado en metas (objetivos). Funciona de la siguiente manera [16]:

- El primer paso es definir una o más metas mensurables. Éstas pueden ser metas estratégicas de alto nivel, como por ejemplo minimizar el costo o maximizar la satisfacción del usuario. Se pueden especificar metas como la evaluación de la efectividad de procesos nuevos, o determinar si un producto está listo para ser liberado al usuario.

- El segundo paso es definir las preguntas necesarias para ser contestadas, y determinar si la meta es cumplida.

- El paso final es determinar que métricas son necesarias para contestar cada pregunta.

3.8 Ciclo del tiempo

El ciclo de tiempo es una medida que nos proporciona el tiempo empleado para llevar a cabo un proceso. Hay dos medidas de tiempo [16]:

- *Ciclo de tiempo estático*: se utiliza el tiempo promedio actual que se emplea para ejecutar un proceso. Por ejemplo que tiempo emplea un módulo en corregir una falla y ejecutar un caso de prueba.

- *Ciclo de tiempo dinámico*: se calcula dividiendo el número de elementos en progreso (elementos que únicamente han completado el proceso parcialmente) entre los nuevos elementos comenzados y nuevos elementos terminados durante el periodo.

Conociendo el ciclo del tiempo para los subprocesos en el proceso de desarrollo de software, se podrá hacer una mejor estimación del plan y de los recursos requeridos, también nos permite monitorear el impacto de las actividades del proceso en el mejoramiento del ciclo de tiempo para este proceso [16].

El siguiente ejemplo ilustra el cálculo de métricas del ciclo de tiempo estático y del ciclo de tiempo dinámico:

Ciclo del tiempo estático:

Calendario de tiempo para completar el proceso:
Módulo A: 5 días
Módulo B: 10 días
Módulo C: 7 días
Módulo D: 8 días

CTE = (5 + 10 + 7 + 8) / 4 = 7.5 días

Ciclo de tiempo dinámico:

Total de elementos en progreso / (elementos iniciados + elementos completados)/2 (3.3)

52 módulos iniciados en este mes

68 módulos terminados en este mes

12 módulos en progreso en este mes

CTD = (12/ ((52+68) /2)) * 30 días en el mes = 6 días

3.9 Enfoques de Productividad

La relación entre las líneas de código y los puntos de función depende del lenguaje de programación que se utilice para implementar el software y de la calidad del diseño. Las medidas LDC y PF se utilizan a menudo para extraer métricas de productividad. Esta invariabilidad conduce a la discusión sobre el uso de tales datos. ¿Se debe comparar la relación LDC/personas-mes (o PF/PM) de un grupo con los datos similares de otro grupo?, ¿Deben los administradores evaluar el rendimiento de

las personas usando estas medidas?. La respuesta a estas preguntas es un terminante *"No"*. La razón para esta respuesta es que hay muchos factores que influyen en la productividad, haciendo que la comparación de elementos incompatibles sea mal interpretada con facilidad. Basili y Zelkowltz definen cinco factores importantes que inciden en la productividad del software [16]:

Factores humanos
El tamaño y la experiencia de la organización de desarrollo.
Factores del problema:
La complejidad del problema que se debe resolver y el número de cambios en las restricciones o los requisitos del diseño.
Factores del Proceso:
Técnicas de análisis y diseño que se utilizan, lenguajes y herramientas CASE y técnicas de revisión.
Factores del producto:
Fiabilidad y rendimiento del sistema basado en computadora.
Factores de recursos:
Disponibilidad de herramientas CASE, y recursos de hardware y software.

Si uno de los factores de productividad está por encima de la media (altamente favorable) para un proyecto dado, la productividad de desarrollo del software será significativamente más alta que el mismo factor por debajo de la media (desfavorable).

3.10 Recursos, Procesos y Productos

La primera obligación en cualquier actividad de medición de software es identificar las entidades y atributos de interés que deseamos medir. Sabiendo de antemano que una *entidad* es un objeto o un evento, y los *atributos* son las características o propiedades del software a medir. En el software hay tres clases de entidades cuyos atributos son susceptibles de medición; Estos se muestran en la figura 3.3 [16]

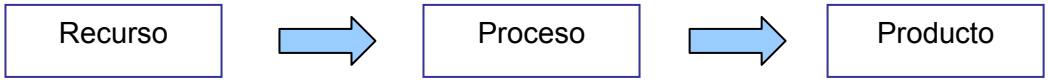

Figura 3.2 Entidades del Software [16]

Recursos: son los artículos que corresponden a las entradas a los procesos.

Procesos: es cualquier software relacionado con las actividades, teniendo éstos normalmente un factor de tiempo.

Productos: cualquier artefacto, liberados o documentos que surjan de los procesos.

3.10.1 Recursos

Los recursos son los diversos puntos de partida a considerar para la producción de software. La estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software se muestra en la figura 3.3 en forma de pirámide. En la base de la pirámide de recursos se encuentra el *entorno de desarrollo* - hardware y software que nos van a proporcionar la infraestructura de soporte al esfuerzo de desarrollo. En un nivel más alto se encuentran los *componentes de*

software reutilizables, que son bloques de software que pueden reducir drásticamente los costos de desarrollo y acelerar la entrega.
En la parte más alta de la pirámide está el *recurso primario:* las personas.

Cada recurso queda especificado mediante cuatro características: descripción del recurso, informe de disponibilidad, fecha cronológica en la que se requiere el recurso, tiempo durante el que será aplicado el recurso. Las dos últimas características pueden verse como una *ventana temporal.* La disponibilidad del recurso para una ventana específica tiene que establecerse lo más pronto posible [16].

Recursos Humanos: El encargado de la planificación comenzará evaluando el ámbito y seleccionando las habilidades técnicas requeridas para llevar a cabo el desarrollo. Dentro de la selección se deberá especificar la posición que ocupará dentro de la organización (ingeniero de software, administrador,...) y la especialidad, (telecomunicaciones, bases de datos, microprocesadores, administrador de proyectos o de grupo). Para proyectos relativamente pequeños (una persona-año o mes o menos) una sola persona puede llevar a cabo todos los pasos de ingeniería del software, consultado con especialistas siempre que requiera. El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo (personas-mes, personas-año) [16].

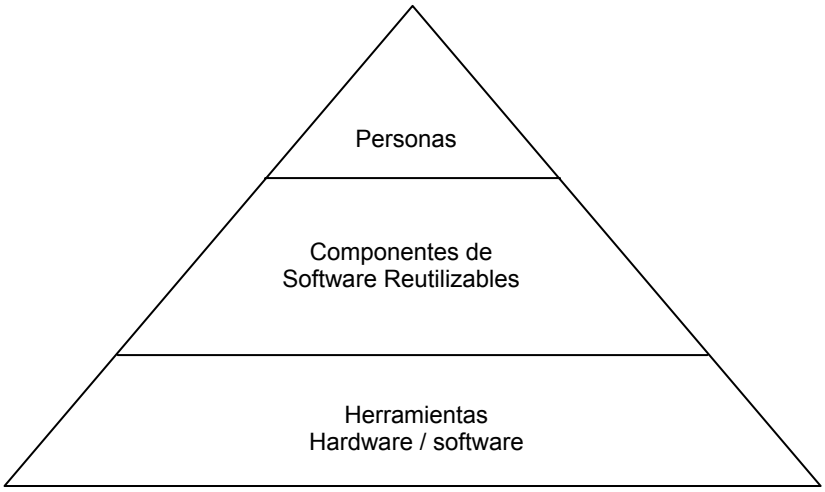

Figura 3.3 Recursos de Desarrollo de Software [Pressmnan] [16].

Recurso de software reutilizable: Se refiere la creación y reutilización de bloques de construcción de software, en donde deberán establecerse catálogos para una consulta más fácil, estandarizarse para una fácil aplicación y validarse para una fácil integración. Pressman [16] sugiere cuatro categorías de recursos de software que se deberán tener en cuenta a medida que se avanza con la planificación.

Componentes ya desarrollados: El software ya existente se puede adquirir de una tercera parte o provenir de un desarrollo interno para un proyecto anterior. Por lo tanto estos componentes están listos para utilizarse en el proyecto actual y se han validado totalmente.

Componentes ya experimentados: Son las especificaciones, diseño, código o datos de prueba existentes y desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual y que los miembros del equipo de software actual ya han tenido la experiencia completa en el área de aplicación representada para estos componentes, y por lo tanto las modificaciones

requeridas tendrán un riesgo relativamente bajo.

Componentes con experiencia parcial: Las especificaciones, los diseños, código o los datos de prueba existentes desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial y los miembros del equipo de software actual han limitado su experiencia sólo a el área de aplicación representada por estos componentes y es por eso que las modificaciones requeridas para componentes de experiencia parcial tendrán un mayor grado de riesgo.

Componentes nuevos. Los componentes de software que el equipo debe construir específicamente para las necesidades del proyecto actual. De forma irónica, a menudo se descuida la utilización de componentes de software reutilizables durante la planificación, llegando a convertirse en la preocupación primordial durante la fase de desarrollo del proceso de software. Por lo tanto, es mejor especificar al principio las necesidades de recursos del software. De esta forma se puede dirigir la evaluación técnica de alterativas y puede tener lugar la adquisición oportuna.

Recursos de entorno. El entorno es en donde se apoya el proyecto de software, llamado a menudo *entorno de ingeniería de software* (EIS), que incorpora hardware y software, en donde el hardware aporta una plataforma con las herramientas (software) requeridas para producir los productos finales. Cuando a un sistema basado en computadora (hardware y software especializado) le es aplicado ingeniería, el equipo de software puede requerir acceso a los elementos en desarrollo por otros equipos de ingeniería y es por eso que cada elemento de hardware debe ser especificado por el planificador del proyecto de software.

Un atributo de gran interés el cuál es relevante para todos, es el *"costo",* que es considerado en muchas situaciones, dependiente del número de atributos en adición a los cuales son mas fáciles de medir, llamado *precio.* En el caso de personas como un recurso en adición al costo, será de interés el atributo de *producción.* Este será externo porque es dependiente en un proceso en particular.
Otros atributos de interés para las personas, de forma individual, son *experiencia, edad, inteligencia* o, para equipos de trabajo *tamaño, estructura,* y *tipos de comunicación"* [9].

3.10.2 Procesos

Las entidades de procesos de software incluyen actividades relacionadas con el software y eventos que usualmente son asociados con un factor de tiempo.

Las métricas del proceso de software se utilizan para propósitos estratégicos. Por ejemplo: actividades definidas, como el desarrollo de un sistema de software desde los requerimientos hasta la liberación al usuario, o la inspección de una parte del código. Las métricas de proceso también se extraen midiendo las características de tareas específicas de la ingeniería de software y obteniendo como resultados medidas de errores detectados antes de la entrega del software, defectos detectados e informados por los usuarios finales, productos de trabajo entregados, el esfuerzo humano y tiempo consumido, ajuste con la planificación y otras medidas [16]. Por ejemplo:

Ejemplo 1: Quizás sea razonable usar una medida indirecta para un proceso

como una prueba formal:

$$\frac{Costo}{Número\ de\ errores\ encontrados} \qquad (3.4)$$

Esta fórmula nos da el promedio entre el costo y el número de errores encontrados durante el proceso, donde el costo serían los atributos de procesos externos los cuales se cree que son importantes tales como: *controlabilidad, observabilidad* y *estabilidad*

Ejemplo 2: Se podría usar la fórmula 3.5 como base para una medida de estabilidad del proceso del diseño durante un período específico de tiempo.
Específicamente:

$$Estabilidad\ de\ diseño = \frac{número\ total\ de\ metodologías\ de\ diseño}{Número\ de\ procesos\ de\ diseño} \qquad (3.5)$$

Los indicadores de procesos permiten a una organización de ingeniería de software tener una visión profunda de la eficacia de un proceso ya existente, por ejemplo: el paradigma, las tareas de ingeniería de software, los productos del trabajo e hitos. También permiten que los administradores evalúen lo que funciona y lo que no. Las métricas de proceso se recopilan de todos los proyectos y durante un largo periodo de tiempo. El objetivo es proporcionar indicadores que lleven a mejoras de los procesos de software a largo plazo.
Según Pressman [16] los indicadores de proyecto permiten al administrador de software:

- Evaluar el estado del proyecto en curso.
- Dar trazabilidad de riesgos potenciales.
- Detectar las áreas de problemas antes de que se conviertan en críticas.
- Ajustar el flujo y las tareas de trabajo.
- Evaluar la habilidad del equipo del proyecto en controlar la calidad de los productos de trabajo de la ingeniería de software.

La única forma racional de mejorar cualquier proceso, es medir atributos del mismo, desarrollando un conjunto de métricas que proporcionen indicadores que conducirán a una estrategia de mejora [9]. En la figura 3.4, se muestra tal *proceso,* en el centro de un triángulo que es conectado por tres factores con una gran influencia en la calidad del software y en el rendimiento como organización donde la complejidad del *producto* puede tener un impacto sustancial sobre la calidad y el rendimiento del equipo, la *tecnología* (p. Ej. métodos de la ingeniería del software) que puebla el proceso también tiene su impacto. Están dentro del círculo de condiciones, entornos que incluyen "Entornos de Desarrollo" (p. Ej. Herramientas CASE), "Condiciones del Negocio" (p. Ej. fechas, tope, reglas de empresa), y "Características del Cliente" (p. Ej.: facilidad de comunicación" [16].

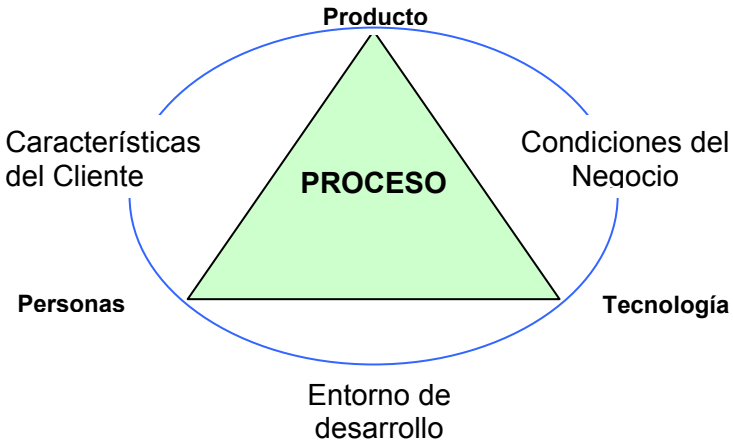

Figura 3.4 Estrategia de Mejora del Proceso [Pressman] [16]

Mayhauser [16] argumenta que existen usos *"privados y públicos"* para diferentes tipos de datos del proceso. Los ingenieros de software podrán sentirse no familiarizados con la utilización de métricas recopiladas de una base particular, estos datos deberían ser *privados* para el individuo y servir sólo como un indicador de ese individuo. Algunos ejemplos podrían ser los índices de defectos, los índices de defectos por módulo, errores encontrados durante el desarrollo, ya que sabemos que los datos *privados* de proceso pueden servir como referencia importante para mejorar el trabajo individual del ingeniero de software.

Mayhauser [16] menciona que algunas métricas de proceso son *privadas* para el equipo del proyecto de software, pero *públicas* para todos los miembros del equipo. Entre los ejemplos se incluyen los defectos informados de funciones importantes del software (que un grupo de profesionales han desarrollado), errores encontrados durante revisiones técnicas formales y líneas de código o puntos de función por módulo y función. El equipo revisa los datos para detectar los indicadores que pueden mejorar el rendimiento del equipo.

Las métricas de proceso del software pueden proporcionar grandes resultados a medida que una empresa trabaja para aumentar su nivel de madurez del proceso. También favorecen a clarificar el estatus de los proyectos y su aprobación con el plan de desarrollo ya que, al crear un buen proceso durante el desarrollo del software, (buenas técnicas, administración, horarios, etc.) se reflejará en la calidad de desarrollo del software. Aunque como en todas las métricas éstas pueden ser mal utilizadas originando así un mayor número de problemas, Robert U. Charette [16] sugiere la aplicación de una receta de métricas de software apropiadas para administradores, en la medida que instituyen un programa de métricas de proceso:

- Utilice el sentido común y una sensibilidad organizativa al interpretar datos de métricas.
- Proporcione una realimentación regular a particulares y equipos que hayan trabajado en la recopilación de medidas y métricas.
- No utilice métricas para evaluar a particulares.
- Trabaje con profesionales y equipos para establecer objetivos claros y métricas que se vayan a utilizar para alcanzarlos.
- No utilice nunca métricas que amenacen a particulares o equipos.
- Los datos de métricas que indican un área de problemas no se deberían considerar "negativos". Estos datos son meramente un indicador de mejora del proceso.

- No se obsesione con una sola métrica y excluya otras métricas importantes.

A medida que una organización está más a gusto con la recopilación y utilización de métricas de proceso, la derivación de indicadores simples abre el camino hacia un enfoque más riguroso, llamado Mejora *de Estadística del Proceso del Software* **MEPS** (Statistical Software Process Improvement (SSPI). Pressman [16] menciona que MEPS utiliza el análisis de fallas del software para recopilar información de errores y defectos encontrados en el desarrollo y utilizar una aplicación de sistema o producto. El análisis de fallos funciona de la siguiente manera:

- Todos los errores y defectos se categorizan por origen (p. ej.: defectos en la especificación, en la lógica, no acorde con los estándares).

- Se registra tanto el costo de corregir cada error como el del defecto.

- El número de errores y de defectos de cada categoría se cuentan y se ordenan en orden descendente.

- Se calcula el costo global de errores y defectos de cada categoría que producen el costo más alto para la organización.
- Los datos resultantes se analizan para detectar las categorías que producen el costo más alto para la organización.

- Se desarrollan planes para modificar el proceso con el intento de eliminar (o reducir la frecuencia de apariciones) la clase de errores y defectos que sean más costosos.

3.10.2.1 Métricas del proyecto.

Las métricas del proyecto de software son tácticas. Esto es, las métricas de proyectos y los indicadores derivados de ellos son utilizados por un administrador de proyectos y un equipo de software, para adaptar el flujo de trabajo del proyecto y las actividades técnicas.

La primera aplicación de métricas de proyectos en la mayoría de los proyectos de software ocurre durante la estimación. Las métricas recopiladas de proyectos anteriores se utilizan como una base desde la que se realizan las estimaciones del esfuerzo y del tiempo para el actual trabajo de software.

A medida que avanza un proyecto, las medidas del esfuerzo y del tiempo consumido se comparan con las estimaciones originales (y la planificación del proyecto). El administrador de proyectos utiliza estos datos para supervisar y controlar el avance.

A medida que comienza el trabajo técnico, otras medidas de proyectos comienzan a tener significado. Se miden los índices de producción representados mediante páginas de documentación, las horas de revisión, los puntos de función y las líneas de código fuente entregadas. Además, se sigue la pista de los errores detectados durante todas las tareas de ingeniería del software. Conforme el software va evolucionando desde la especificación al diseño, se recopilan métricas técnicas para evaluar la calidad del diseño y para proporcionar indicadores que influirán en el enfoque tomado para la generación de código, módulos y pruebas de integración [9].
La utilización de métricas para el proyecto tiene dos aspectos fundamentales [16]:

En primer lugar, estas métricas se utilizan para minimizar la planificación de

desarrollo guiando los ajustes necesarios que eviten retrasos y atenúen problemas y riesgos potenciales. En segundo lugar, las métricas para el proyecto se utilizan para evaluar la calidad de los productos en el momento actual y cuando sea necesario, modificar el enfoque técnico del mejoramiento de la calidad.

A medida que mejora la calidad, los errores se minimizan y el número de defectos disminuye, consecuentemente la cantidad de trabajo que ha de rehacerse se reduce también. Esto lleva a una reducción del costo global del proyecto.

Otro modelo de métricas del proyecto de software sugiere que todos los proyectos deberían medir [16]:

- *Entradas:* la dimensión de los recursos (p. ej personas, medio ambiente) que se requieren para realizar el trabajo.

- *Salidas:* medidas de las entregas o productos creados durante el proceso de ingeniería de software.

- *Resultado:* medidas que indican la efectividad de las entregas.

En realidad, este modelo se puede aplicar tanto al proceso como al proyecto. En el contexto del proyecto, el modelo se puede aplicar recursivamente a medida que aparece cada actividad estructural. Por consiguiente, las salidas de una actividad se convierten en las entradas de la siguiente. Las métricas de resultados se pueden utilizar para proporcionar una indicación de la utilidad de los productos cuando fluyen de una actividad (o tarea) a la siguiente.

3.10.2.2 Integración de las métricas dentro del proceso de software

La mayoría de los desarrolladores de software todavía no miden, y por desgracia, la mayoría no desean ni comenzar. Como se ha señalado, el problema es cultural. En un intento por recopilar medidas en donde no se haya recopilado nada anteriormente, a menudo se opone resistencia: *¿Por qué es necesario hacer esto?*, *¿Por qué es tan importante medir el proceso de ingeniería del software y el producto (software) que se produce?*. La respuesta es relativamente obvia. Si no se mide, no hay una forma real de determinar si se está mejorando, y si no se está mejorando, se está perdido [16]. La medición es una de las *medicaciones* que pueden ayudar a curar el "*mal del software*". Esta proporciona beneficios al nivel de proyecto, estratégico y técnico.

La administración de alto nivel puede establecer objetivos significativos de mejora del proceso de ingeniería del software solicitando y evaluando las medidas de productividad y de calidad. Se señaló que el software es un aspecto de administración estratégico para muchas compañías. Si el proceso por el que se desarrolla puede mejorarse, entonces puede producirse un impacto directo en lo sustancial. Pero para establecer objetivos de mejora, se debe comprender el estado actual de desarrollo del software.

Los rigores del trabajo diario de un proyecto de software no dejan mucho tiempo para pensar en estrategias. Los administradores del proyecto de software están más preocupados por aspectos mundanos (aunque igualmente importantes), como desarrollo de estimaciones significativas del proyecto, producción de sistemas de alta calidad, y finalización del producto a tiempo. Mediante el uso de la medición para establecer una línea base del proyecto, cada uno de estos asuntos se hace más fácil de manejar. Ya se ha apuntado que la línea base sirve como un lineamiento para la

estimación. Además, la recopilación de métricas de calidad permite a una organización "sintonizar' su proceso de ingeniería del software para eliminar las causas *"poco vitales"* de los defectos, que tienen el mayor impacto en el desarrollo del software [9] [16].

Técnicamente (en las trincheras) las métricas de software, cuando se aplican al producto, suministran beneficios inmediatos. Cuando se ha terminado el diseño de software, la mayoría de los que desarrollan pueden estar ansiosos por obtener respuestas a preguntas como:

- ¿Qué requerimientos del usuario son más susceptibles al cambio?
- ¿Qué módulos del sistema son más propensos a error?
- ¿Cómo se debe planificar la prueba para cada módulo?
- ¿Cuántos errores (de tipos concretos) pueden esperarse cuando comiencen las pruebas?

Se pueden encontrar respuestas a esas preguntas si se han recopilado métricas y se han usado como guía técnica.

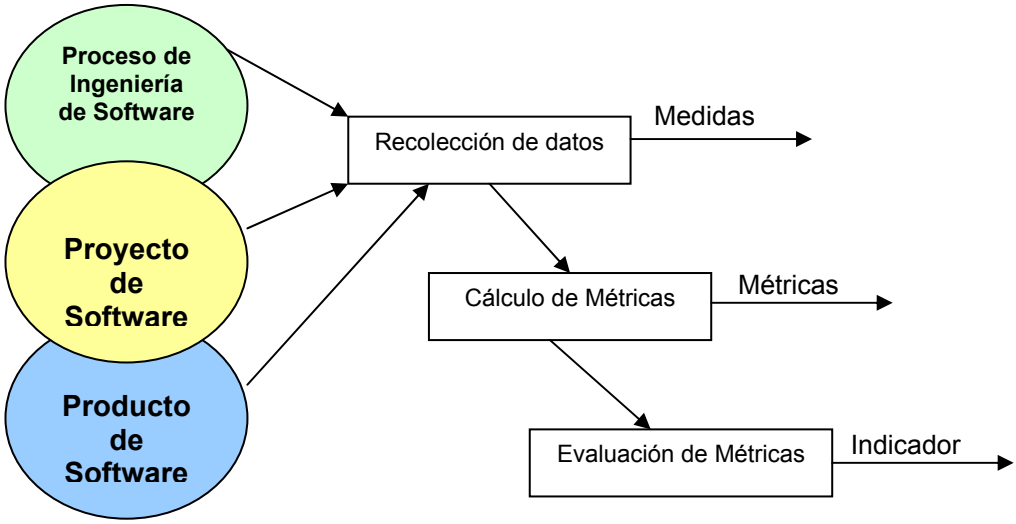

Figura 3.5 Proceso de recopilación de métricas de software [Pressman] [16]

El proceso que se establece en una línea base (datos recopilados de proyectos de desarrollo de software anteriores) se muestra en la figura 3.5.
Idealmente, los datos necesarios para establecer una línea base han sido recopilados a medida que sé ha ido progresando. Por desgracia, este no es siempre cierto. Por consiguiente, la *recopilación de datos* requiere una investigación histórica de los proyectos anteriores para reconstruir los datos requeridos, cada vez que sea posible. Dependiendo de la amplitud de las medidas recopiladas, las métricas pueden abarcar una gran gama, tales como: LDC y PF, así como métricas de calidad y orientadas a objetos. Finalmente, las métricas se deben evaluar y aplicar durante la estimación, el trabajo técnico, el control del proyecto y la mejora de proyectos, la *evaluación de métricas* se centra en las razones de los resultados obtenidos, y produce un grupo de indicadores que guían el proyecto o el proceso.

3.10.3 Productos

Los productos de software son las salidas del proceso de producción del mismo. Éstas incluyen todos los artefactos entregados o documentos que son producidos durante el ciclo de vida del software. Anneliese Von [16] menciona que las métricas de los productos usualmente cuantifican algunos aspectos de calidad relacionados a la tabla 3.7.

Tabla 3.7 Aspectos de calidad de software

Lista de Requerimientos de Calidad	
Las capacidades de los productos	El grado de la funcionalidad
Usabilidad	fiabilidad
desempeño (eficiencia)	seguridad
factores humanos	Portabilidad
La extensión de la solución	Futuro/expectativas del tiempo de vida
Compatibilidad	Mantenibilidad
Adaptabilidad	Correctivo
Recursos	Perfectivo
Limitaciones personales	Instalación
Limitaciones de presupuesto	Horario
Beneficios:	Limitaciones físicas
Tangibles	Documentación
Intangibles	Limitaciones organizacionales

No siempre será posible medir éstos directamente, especialmente cuando las métricas son *predecibles,* por ejemplo, cuando el software no esta en desarrollo y la métrica es usada para predecir su calidad operacional, entonces los subfactores y atributos deliberados son medidas que correlacionan dando como resultado una métrica indirecta [16].

3.10.3.1 Modelo de Planificación Predictiva

Es intuitivamente obvio que un programa largo tiende a ser más complejo y costoso de desarrollar y mantener. Un programa largo también tiende a tener un vocabulario grande y usa más operaciones y operandos. También tiene más líneas de código (LOC). La figura 3.6 muestra una relación entre métricas y el modelo de desarrollo para las variables que planean. El signo de interrogación en la fórmula indica el factor de tamaño, el cual puede estar influyendo en los parámetros de salida del modelo. Este modelo de fórmulas sólo funciona, si el tamaño tiene una influencia significativa en el tiempo, costo y requerimientos de los recursos para un proyecto, y si la relación entre tamaño y complejidad es fuerte. Si este no es el caso, el modelo quizá funcione para algún proyecto pero no para otros.

Cuando el tamaño no es el único factor significativo para la variable planeada, el modelo a predecir entonces será multidimensional y más complejo.

Un *modelo* es una relación de la forma Y = *f(x1, x2,….., xm).* Donde Y es la variable que deseamos predecir. Esta es llamada *variable dependiente.* Las Xi son las *variables independientes* en que la predicción depende de Y. Mayrhauser [16]

menciona que hay dos caminos para establecer esta relación funcional:

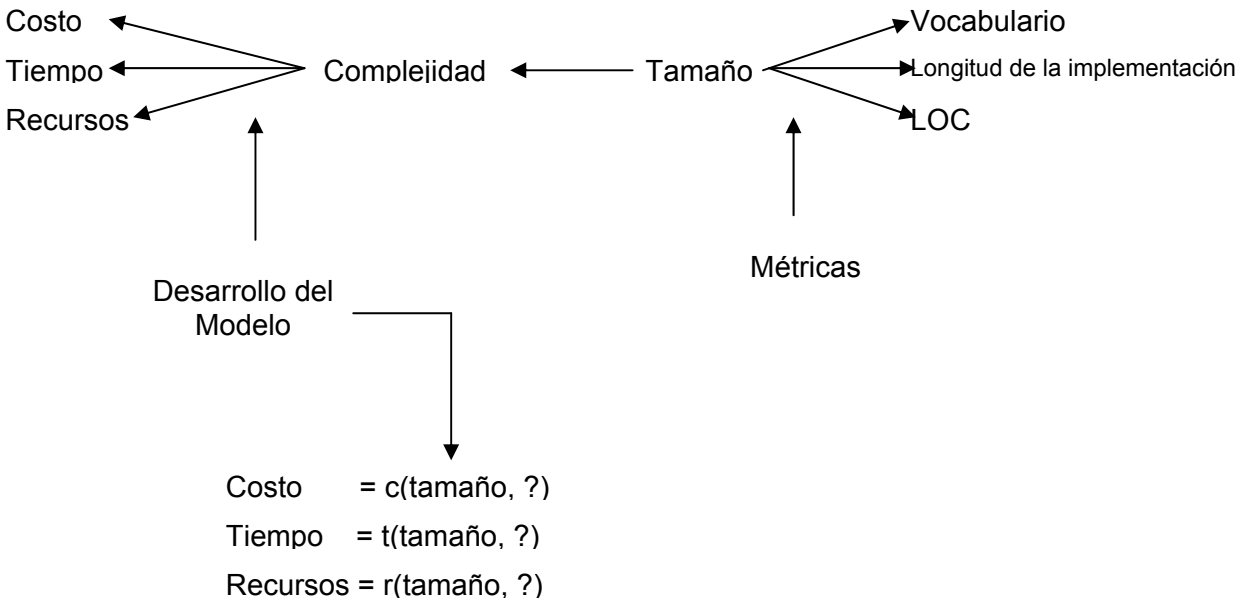

Figura 3.6 Métricas Posibles para un modelo predictivo [9]

El primero es, por análisis estadístico de los datos del caso. Este es un *modelo empírico.* Muchos modelos de *costo* de estimación son de este tipo, considerándose que son tan representativos como los datos sobre los que estos se basan, y la calidad de los métodos estadísticos de inferencia utilizados.

El segundo camino se lleva acabo a través de razonamiento analítico y pruebas que establecen la relación entre variables independientes y dependientes. Algunos modelos analíticos postulan una familia de funciones con situaciones constantes.

Estos tienen que ser medidos separadamente o estimados por métodos estadísticos. El modelo se *calibra* a su medio (o ambiente) en la aplicación.

CAPITULO 4 ESTADO DEL ARTE

4.1 Introducción

La medición es una actividad fundamental a cualquier disciplina de ingeniería, y la ingeniería de software no es la excepción [1]. Típicamente, las métricas son esenciales en la ingeniería de software dado que proveen mecanismos para la medición de la complejidad y calidad, estimando costos y esfuerzo invertido en un proyecto, solo por mencionar algunos aspectos.

En una disciplina de ingeniería no podemos tolerar ambigüedades, probablemente inevitable en las sociologías. Si vamos en busca de rigor, las herramientas de lógica matemática y el razonamiento formal son cruciales, aunque ellas no sean cuantitativas.

Los números entregados por las métricas nos ayudan a entender y controlar el proceso de la ingeniería [22].

Dado que el Software Orientado a Objetos (OO) es fundamentalmente distinto del software que se desarrolla utilizando métodos convencionales, las métricas para sistemas OO deben ajustarse a las características que lo distinguen del software convencional.

Las métricas tradicionales como *"puntos de función"* y *"complejidad ciclomatica"* [16], se han usado eficientemente en el paradigma procedural. Sin embargo, estas no aplican a los aspectos del paradigma orientado a objetos (OO): clases, objetos, acoplamiento, etc.,

Las métricas OO hacen hincapié en conceptos tales como el encapsulamiento, herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas OO se centran en las mediciones que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hacen única a una clase.

Lamentablemente, la utilización de métricas para sistemas orientados a objetos ha progresado con mucha más lentitud que la utilización de los demás métodos OO. Sin embargo a medida que los sistemas OO van siendo más habituales, resulta fundamental que los ingenieros del software dispongan de mecanismos cuantitativos para estimar la calidad de los diseños y la efectividad de los programas OO [16].

Durante aproximadamente una década, los investigadores han estado discutiendo si era necesario un conjunto separado de métricas de software OO, y qué debía incluirse en este conjunto [2]. Las propuestas Iniciales se enfocaron más hacia la extensión de las métricas de software existentes, referidas a la programación procedural [3] [4]. Sin embargo, casi todas las propuestas recientes se han enfocado en el paradigma OO [5] [7].

Desde la propuesta de las seis métricas OO hecha por Chidamber y Kemerer (CK) [6], otros investigadores han hecho el esfuerzo de validarlas teórica y empíricamente.

A lo largo de este documento, se estudia el campo de las métricas de software Orientado a Objetos, destacando el objetivo de las mismas en lo referente a las métricas de diseño OO. Se repasan conceptos fundamentales del paradigma OO, y se plantea un análisis de umbrales con el objetivo de estimar y cuantificar parámetros que indiquen aproximaciones de la calidad del diseño. El trabajo realizado se fundamenta en la literatura publicada en esta área, en los últimos cinco años.

4.2. Objetivo de las métricas Orientadas a Objetos

Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estructurado:

- Evaluar mejor la calidad del producto.
- Estimar la efectividad del proceso.
- Mejorar la calidad del trabajo realizado en el nivel del proyecto.

Cada uno de estos objetivos es importante en sí, pero para el ingeniero de software, la calidad del producto debe ser lo esencial. ¿Cómo Se puede medir la calidad de un sistema OO?, ¿Que características del modelo de diseño se pueden estimar para determinar si el sistema será o no fácil de implementar?, ¿se podrá probar?, ¿será fácil de modificar?, y lo que es más importante, ¿resultará amigable para los usuarios finales? [16]. Estos argumentos se discuten a lo largo de este documento.

4.3. Características del software Orientado a Objetos

El software orientado a objetos es esencialmente distinto del software que se desarrolla utilizando métodos convencionales. La tecnología de objetos es particularmente útil debido a sus propiedades de cohesión que reducen la brecha entre la estructura del problema y la estructura del programa (la propiedad de mapeo directo) [22].

En particular, se puede sostener en un contexto OO, que la noción de puntos de función, una medida ampliamente aceptada de funcionalidad, puede reemplazarse por una medida mucho más objetiva: el número de características exportadas (operaciones) de clases relevantes que no requieren ninguna decisión humana y pueden medirse trivialmente por una herramienta de análisis gramatical simple [22].

Por esta razón, las métricas para sistemas OO deben ser concordantes con las características que distinguen el software OO del software convencional.
Berard [16] define seis características que dan lugar a unas métricas especializadas:

- Localización
- Encapsulamiento
- Ocultamiento de información
- Herencia
- Polimorfismo
- Técnicas de abstracción de objetos.

4.3.1 Localización

La localización es una característica de software que, indica la forma en que se concentra la información dentro de un programa. En el contexto OO, la información se concentra mediante el encapsulamiento tanto de datos como de procesos dentro de los límites de una clase u objeto.

Dado que el software convencional hace hincapié en las funciones como mecanismos de localización, las métricas de software se han centrado en la estructura interna o complejidad de las funciones (p. ej.. longitud del módulo, cohesión o complejidad ciclomática) o bien en la forma en que las funciones se conectan entre sí (p ej: acoplamiento de módulos).

Dado que las clases constituyen la unidad básica de los sistemas OO, la localización esta basada en los objetos. Por tanto, las métricas deberían ser aplicables a la clase (objeto) como si se tratara de una entidad completa. Además la relación entre operaciones (funciones) y clases no es precisamente uno-a-uno (1:1). Por lo

tanto, las métricas que reflejan la forma en que colaboran las clases deben ser capaces de adaptare a las relaciones uno-a-muchos (1:n) y, muchos-a-uno (n:1) [16].

4.3.2 Encapsulamiento

Berard [16] define el encapsulamiento como “el empaquetamiento” (o enlazado) de una colección de elementos.

Para los sistemas OO el encapsulamiento comprende las responsabilidades de una clase, incluyendo sus atributos (y otras clases para objetos agregados) y operaciones, y los estados de la clase según se definen mediante valores específicos de atributos.

El encapsulamiento influye en las métricas cambiando el objetivo de la medida que pasa de ser un único módulo a ser un paquete de datos (atributos) y de módulos de procesamiento (operaciones). Además el encapsulamiento impulsa a la medida hasta un nivel de abstracción más elevado.

4.3.3 Ocultamiento de información

El ocultamiento de información suprime los detalles operativos de un componente de un programa. Tan solo se proporciona la información necesaria para acceder a ese componente o a aquellos otros componentes que deseen acceder a él.

Un sistema OO bien diseñado debería impulsar el ocultamiento de información. Por tanto aquellas métricas que proporcionen una indicación del grado en que se ha logrado el ocultamiento, proporcionarán una indicación de la calidad de diseño OO [16].

4.3.4 Herencia

Le herencia es un mecanismo que hace posible que los compromisos de un objeto se difundan a otros objetos. Le herencia se produce a lo largo de todos los niveles de la jerarquía de clases. La Herencia es una característica importante del paradigma OO.

El uso de herencia apunta a reducir la cantidad de trabajo de mantenimiento del software necesario y aliviar así la carga de la actividad de prueba [7].

La reutilización de software a través de la herencia apunta a producir software mantenible, entendible y fiable [8].

Los resultados de las investigaciones, no se aplican típicamente al software industrial [9] por ejemplo, la investigación experimental llevada a cabo por Harrison [10] indica que un sistema que no usa herencia es mejor para comprenderlo y mantenerlo, que un sistema que la utiliza. Sin embargo, el experimento realizado por Daly [11] indica que un sistema con tres niveles en la jerarquía de herencia es más fácil de modificar que un sistema sin herencia.

Se ha convenido que cuanto más profunda es la jerarquía de herencia, mejor es la reusabilidad de las clases, pero cuanto más alto es el nivel de acoplamiento entre las clases heredadas, más difícil es mantener el sistema.

Los diseñadores pueden tender a mantener las jerarquías de herencia poco profundas, intercambiando reusabilidad, a través de la herencia, por simplicidad en el entendimiento [7]. De este modo, es necesario medir la complejidad de la jerarquía de herencia para conciliar entre la profundidad y poca profundidad de la misma.

4.3.5 Abstracción

La abstracción es un mecanismo que permite al diseñador centrarse en los detalles esenciales de algún componente de un programa (tanto si es un dato, como si es un proceso) sin preocuparse por los detalles de nivel inferior. Cuando los niveles de abstracción van elevándose, se ignoran más y mas detalles, por lo tanto, se proporciona una visión más general de un concepto u objeto. A medida que pasamos

a niveles mas reducidos de abstracción, se muestran más detalles, esto es, se proporciona una visión mas especifica de un concepto u objeto.

Dado que una clase es una abstracción que se puede visualizar con muchos niveles distintos de detalle, y de muchas maneras diferentes, las métricas OO representan la abstracción en términos de medidas de una clase (p ej: numero de instancias por clase por aplicación) [16].

4.3.6 Polimorfismo

El polimorfismo representa un concepto de teoría de tipos, en el que un solo nombre (tal como una declaración de una variable), puede denotar instancias de muchas clases diferentes, en tanto en cuanto estén relacionadas por alguna superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto común de operaciones, de diversas formas.

El polimorfismo, se vuelva más útil cuando existen muchas clases con los mismos protocolos. Con polimorfismo, no son necesarias grandes sentencias case, porque cada objeto conoce implícitamente su propio tipo [21].

4.4. Métricas para el modelo de diseño Orientado a Objetos

Gran parte del diseño orientado a objetos es subjetivo, un diseñador experimentado *sabe* como puede caracterizar un sistema OO para que se implementen de forma efectiva los requerimientos del cliente. Pero a medida que los modelos de diseño OO van creciendo en tamaño y complejidad, puede resultar beneficiosa una visión mas objetiva de las características del diseño, tanto para el diseñador experimentado como para el que posee menos experiencia [16].

Una visión objetiva del diseño debería tener un componente cuantitativo, y esto nos lleva a las métricas OO, en donde se pueden aplicar no solo al modelo de diseño, sino también al modelo de análisis.

4.4.1 Métricas Orientadas a Clases

La clase es la unidad principal de todo sistema OO. Por consiguiente, las medidas y métricas para una clase individual, las jerarquías de clases, y las colaboraciones de clases resultaran sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño. Se ha visto que la clase encapsula a las operaciones (procesamiento), y a los atributos (datos). La clase suele ser el “Predecesor” de las subclases (también denominadas *descendientes*) las cuales heredan sus atributos y operaciones. La clase suele colaborar con otras clases. Todas estas características se pueden utilizar como bases de las métricas abordadas a continuación.

4.4.1.1 El Conjunto de Métricas CK

Uno de los conjuntos de métricas de software OO a los que se hace referencia mas ampliamente, es el propuesto por Chidamber y Kemmerer [6]. Estas métricas propuestas se refieren al diseño de las clases, a las cuales suele aludirse con el nombre de conjunto de métricas CK para sistemas OO.

A continuación se presenta y explica cada una de estas métricas, junto los refinamientos sugeridos para cada una de ellas.

4.4.1.1.1 Métodos Ponderados por Clase (**WMC**),

Son aquellos en donde se definen n métodos de complejidad C1, C2, ….Cn, para una clase C. La métrica de complejidad específica que se selecciona, (por ej.:

complejidad ciclomática [16]) debe normalizarse de modo tal que la complejidad nominal para un método tome el valor 1

$$MPC = \sum Ci \quad \Leftrightarrow \quad \text{para } i = 1 \text{ hasta } n$$

El numero de métodos y su complejidad es un indicador razonable de la cantidad de esfuerzo necesario para implementar y comprobar una clase. Además, cuanto mayor sea el número de métodos, mas complejo será el árbol de herencia, (ya que todas las subclases heredan los métodos de sus predecesoras).

Finalmente, a medida que el número de métodos crece para una clase dada, es más probable que se vuelva cada vez más específico de la aplicación, limitando por tanto su potencial de reutilización.

Por todas estas razones, WMC debería mantenerse en un valor tan bajo como sea posible.

Aun cuando podría parecer relativamente sencillo desarrollar un contador del número de métodos de una clase, el problema es en realidad más complejo de lo que parece. Churcher y Shepperd [16] examinan esta inconveniente cuando dicen:

> *Para contar métodos, es preciso responder a una pregunta fundamental: ¿Pertenece un método solamente a la clase que lo define e implementa, o bien pertenece también a todas aquellas clases que lo heredan directa o indirectamente?. Las preguntas como estas pueden parecer superficiales, por cuanto el sistema de ejecución acabara por resolverlas. Sin embargo, las implicaciones para las métricas pueden ser significativas.*
>
> *Una posibilidad es restringir el recuento a la clase en curso, ignorando los miembros heredados. La motivación para esto seria que los miembros heredados ya habrán sido contados en las clases que fueran definidos, así que el incremento de clase es la mejor medida de su funcionalidad. Con objeto de entender lo que hace una clase, la fuente de información más importante son sus propias operaciones. Si una clase dada no es capaz de responder un mensaje (es decir que la clase carece del correspondiente método propio) entonces esta pasara el mensaje a su predecesor o predecesores.*
>
> *Por otra parte, el recuento podría envolver a todos aquellos métodos definidos en la clase en curso, junto con todos lo métodos heredados. Este enfoque hace hincapié en la importancia del espacio de estados, en lugar de hacer hincapié en el incremento de la clase, para comprender la clase.*

Al igual que la mayoría de las convenciones de recuento en las métricas de software, cualquiera de los enfoques mencionados anteriormente es aceptable, siempre y cuando este enfoque de recuento se aplique de forma consistente cada vez que se recoja una métrica.

4.4.1.1.2 Profundidad del Árbol de Herencia (DIT)

Esta métrica se define como la longitud máxima desde el nodo analizado, hasta la raíz del árbol. El valor resultante de DIT para la jerarquía de clases mostrada en la figura 4.1 es 4 [6] [16].

A medida que DIT crece, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de DIT) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de DIT implican que se pueden reutilizar muchos métodos.

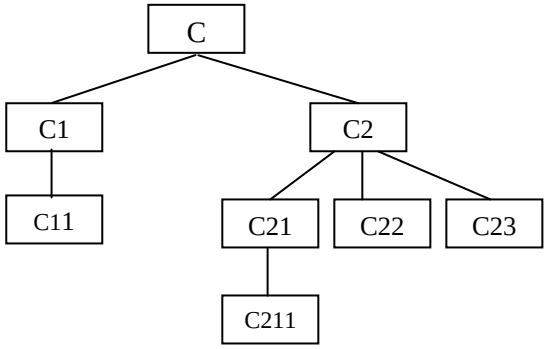

Figura 4.1 *Jerarquía de Clases [16]*

Li [18] ha apuntado que la métrica DIT de CK tiene ambigüedades. Primero, la definición de DIT es ambigua ante la presencia de herencia múltiple, donde una clase tiene raíces múltiples. Considerando el árbol de herencia de clases con raíces múltiples en la Figura 4.3, la longitud máxima de la clase E es incierto.

Hay dos raíces en este diseño, la longitud máxima de la clase E hasta la raíz B es 1 (DIT(E) = 1) y la longitud máxima de la clase E hasta la raíz A es dos (DIT(E) = 2).

El segundo factor que causa ambigüedad radica en las metas contradictorias declaradas en la definición y la base teórica para la métrica DIT.

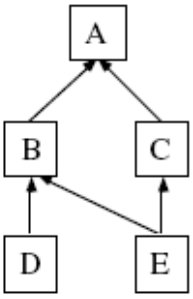

Figura 4.2*. Árbol de herencia* [19]

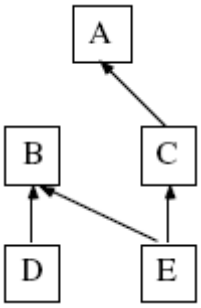

Figure 4.3*. Árbol de herencia entre clases con raíces múltales.* [19]

Esta ambigüedad sólo es visible cuando existe herencia múltiple, dónde la distancia entre una clase y la raíz en el árbol de herencia no devuelve el mismo número de clases ancestros para la clase analizada. Este conflicto se muestra en la Figura 4.2 donde, según la definición, las clases D y E tienen la misma longitud máxima respectivamente desde la raíz del árbol a los nodos; así DIT(D) = DIT(E) = 2. Sin embargo, la clase E hereda de más clases que D, según la base teórica, las clases D y E deben tener valores de DIT diferentes.

Li [18] propuso una nueva métrica, denominada Número de Clases Ancestros (NAC), como una alternativa a la métrica DIT. La cual cuenta la cantidad total de ancestros para una clase dada.

4.4.1.1.3 Número de Descendientes (NOC)

Las subclases que son inmediatamente subordinadas a una clase se denominan descendientes. En la figura 4.1 la clase C2 tiene tres descendientes, las subclases C21. C22 y C23.

A medida que crece el número de descendientes, se incrementa la reutilización, pero también es cierto que, a medida que crece NOC, la abstracción representada por la clase predecesora, puede verse diluida. Es decir, existe la posibilidad de que algunos de los descendientes no sean claramente miembros

propios de la clase predecesora. A medida que NOC va creciendo, la cantidad de pruebas crecerá también [16].

Chaumun [12], extiende el alcance de las métricas propuestas por CK, como predictores de mantenibilidad, para la propiedad de capacidad al cambio. Debido a la especificidad del impacto al cambio, el autor propone la siguiente extensión de la métrica NOC:

> *NOC * (Número de Hijos en el árbol subalterno):* cuando algún componente de una clase cambia, puede no sólo afectar a sus hijos, sino al árbol-subalterno entero del cual la clase que cambia, es la raíz.

Li [18] también sugiere que NOC, métrica de la suite CK, tiene algunas ambigüedades. La base teórica declarada y los puntos de vista indican que NOC mide el alcance de influencia de una clase en sus subclases, debido a la herencia. No queda claro porque solo se cuentan las subclases inmediatas a la clase en cuestión, dada que la clase tiene influencia sobre todas sus subclases, inmediatas y no inmediatas. Para remediar esta insuficiencia, Li [18] propuso una nueva métrica, llamada Número de Clases Descendientes (NDC), como una alternativa a la métrica NOC.

La definición de la métrica NDC seria entonces, el número de clases descendientes de una clase dada. Considerando la jerarquía de clases de la figura 4.2, NDC(A) = 4, NDC(B) = 2 y NDC(C) = 1. En la Figura 4.3, NDC(A) = NDC(B) = 2

Sheldon [19] cambia la notación del árbol de herencia de clases. El término árbol de herencia de clases no es válido, dado que si existe herencia múltiple, nos encontramos en presencia de un grafo en lugar de un árbol. El modelo matemático más conveniente para describir la taxonomía del objeto con herencia múltiple es un Grafo Acíclico Dirigido (DAG)

La notación del árbol de herencia de clases DAG, se presenta en la figura 4.4

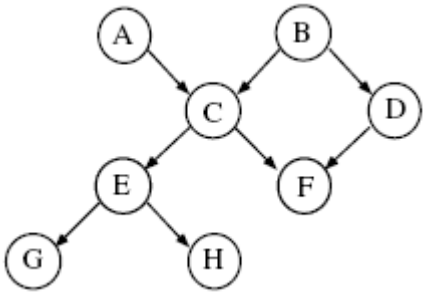

***Figura 4.4**, Herencia de clases DAG* [19]

El vértice i es predecesor de vértice j si existe un camino del vértice i al vértice j y, el vértice j, es un sucesor de vértice i.

- PRED(i): Número total de predecesores del nodo i,
- SUCC(i): Número total de sucesores de nodo i.

En la Figura 4.4, *PRED*(E) = 3, *PRED*(A) = *PRED*(B) = 0, *SUCC*(C) = 4, *SUCC*(G) = *SUCC*(H) = 0. Sheldon [19] declara que hay dos actividades fundamentales en el mantenimiento, comprender la estructura del sistema y modificar el sistema. La capacidad de *Comprensión* (*Understandability*) se define como la facilidad de entender la estructura del programa o estructura de herencia de clases, y la *Modificabilidad (Modifiability)*, se define como la facilidad con que un cambio puede hacerse a la estructura del programa, o estructura de herencia de clases.

En tal sentido los autores [19], proponen el siguiente conjunto de métricas:

Understandability: Donde definen el grado de comprensión (U) de una clase como sigue:

U de la Clase Ci = PRED(CI) + 1

Donde Ci es la i-esima clase. El grado total de comprensión (TU) de una herencia de clase DAG se define como:

$$\text{TU de la herencia de clases DAG} = \sum_{i=1}^{t} (\text{PRED (Ci)} + 1)$$

Donde *t* es el número total de clases en la estructura de herencia DAG

Los autores [19] introducen el concepto de promedio de profundidad de la jerarquía de herencia, el cual indica el nivel general de modelado o abstracciones usadas en la jerarquía.
Definen el promedio del grado de entendimiento (AU) de una herencia de clases DAG como:

$$\text{AU de la herencia de clases DAG} = (\sum_{i=1}^{t} (\text{PRED (Ci)} + 1) / t$$

Un punto importante a enfatizar es que un valor de AU pequeño, resalta una mejor compresión.

Modificabilidad: Define la facilidad de mantenimiento de una clase dada.
Se sigue que la mitad de las subclases sucesoras deben modificarse en promedio. Consecuentemente definen el grado de Modificabilidad (M) de una clase como:

M de una clase Ci = U(Ci) + SUCC(Ci) / 2

Donde Ci es la i-esima clase. El grado total de modificabilidad (TM) de una herencia de clases DAG es:

$$\text{TM de la herencia de clases DAG} = \text{TU} + \sum_{i=1}^{t} (\text{SUCC (Ci) / 2)}$$

Donde *t* es el número total de clases en la estructura de herencia DAG

El promedio del grado de modificabilidad (AM) de una herencia de clases DAG es:

$$\text{AM de la herencia de clases DAG} = \text{AU} + (\sum_{i=1}^{t} (\text{SUCC (Ci) / 2) / } t$$

Tegarden [3], propone la métrica CLD (Class-to-Leaf-Depth), la cual define el numero máximo de niveles en la jerarquía, que están por debajo de una clase dada, y la métrica NOA (Number of Ancestor), la cual define el numero de clases de las cuales una clase dada hereda directa o indirectamente.

4.4.1.1.4 Respuesta Para una Clase (RFC)

El conjunto de respuestas de una clase, es un conjunto de métodos que pueden ser ejecutados potencialmente en respuesta a un mensaje recibido por un objeto de esa clase [6] [16]. RFC se define como el número de métodos existentes en el conjunto de respuestas. A medida que RFC crece, el esfuerzo necesario para la comprobación crece también, debido a que la sucesión de comprobación va creciendo y también la complejidad global de diseño de la clase crece.

4.4.1.1.5 Acoplamiento entre clases (CBO)

CBO mide el Acoplamiento entre Clases de Objetos [6]. Una clase esta acopla a otra si los métodos de una clase usan métodos o atributos de la otra, y viceversa. Dada esta definición, los usos pueden significar, el tipo de miembro, el tipo de parámetro, variable de método local o el llamado a métodos de una clase. CBO es el número de clases con la que una clase dada esta acoplada. Incluye el acoplamiento basado en herencia (es decir, acoplamiento entre clases relacionadas vía herencia).

La métrica CBO fue evaluada empíricamente en [8]. Los autores encontraron que esta métrica esta relacionada con la propensión a fallas.

En esencia CBO, es el número de colaboraciones enumeradas para una clase en su tarjeta CRC [16]. A medida que los valores de CBO crecen, es probable que la reusabilidad de la clase vaya descendiendo. Valores altos de CBO complican también las modificaciones y la comprobación que se produce cuando se efectúan modificaciones.

Chaumun [12], propone una extensión de 3 métricas a la métrica CBO. Donde CBO, es aproximadamente igual al acoplamiento con otras clases, y donde la llamada a un método o variable de instancia de otra clase, constituye el acoplamiento.

CBO_NA (CBO No Ancestros): igual que CBO, pero el acoplamiento entre la clase designada y su ancestro, no es tomada en cuenta: El acoplamiento entre la clase designada y sus ancestros, tenidos en cuenta por CBO, es irrelevante para el impacto al cambio, dado que los ancestros de la clase designada, nunca serán impactados por cambios en la misma. Para eliminar tal "ruido", los ancestros son excluidos en CBO_NA.

CBO_IUB(CBO Es Usada Por (Is Used By)): Consiste en las clases que usan a la clase designada.

CBO_U (CBO Usando (Using)); Es la cuenta de las clases usadas por la clase designada. Esta métrica es introducida para cubrir la parte no considerada por CBO_IUB.

En [20], los autores proponen una técnica para medir la fuerza de las interrelaciones entre clases, la cual toma en consideración el número de sentencias, participantes en la conexión, y la complejidad de tales sentencias. El modelo propuesto es denominado *Complejidad de Interconexión de Clases*, como medida de la fuerza de acoplamiento entre clases.

La formula calcula la complejidad de cada sentencia incorporando los pesos de complejidad individual de cada operando embebido en cada sentencia. Considera también, la manera en que las estructuras de control afectan, o son afectados por el envío de mensajes.

Los autores en [20], proponen también que la *complejidad ciclomática* [16] del modulo debe ser considerada, dado que los programas con estructuras de control profundas, son mas complejos que los programas secuenciales simples.

En tal sentido, la métrica de interconexión de clases, esta diseñada para determinar cuan ocupado esta el código de interconexión subyacente.

Balasubramanian [1], propone una métrica denominada *Peso del Método Enviado* (WM), que cuenta el numero de mensajes enviados entre distintas clases, y el peso del mensaje, acorde al numero de parámetros enviados en cada mensaje. Esta es una mejora respecto de la técnica simple de conteo, pero aun no tiene en consideración la complejidad de la preparación del parámetro a ser enviado, o el uso en la clase receptora [20].

4.4.1.1.6 Carencia de Cohesión en los Métodos (LCOM).

Todo método situado dentro de una clase C, accede a uno o más atributos (denominados también variables de instancia). LCOM es el número de métodos que acceden a uno o más de los mismos atributos. Es decir que, los métodos tienen acceso a atributos en común. Si ningún método accede a los mismos atributos, entonces LCOM será 0 [6].

Como un ejemplo en el que LCOM es distinto de 0, supongamos una clase con 6 métodos, en donde cuatro de estos tienen en común uno o más atributos (es decir que acceden a atributos comunes), por consiguiente, LCOM = 4. Sí LCOM es elevado, los métodos pueden estar acoplados entre sí a través de dichos atributos. Esto incrementará la complejidad del diseño de clases. En general unos valores elevados para LCOM implican que la clase podría diseñarse mejor descomponiéndola en dos o más clases distintas. Aun cuando existen casos en que es justificable un valor elevado de LCOM, es deseable mantener un elevado grado de cohesión, en otras palabras mantener un valor bajo para LCOM [16].

4.4.1.2 Métricas propuestas por Lorenz y Kidd

Lorenz y Kídd [16], dividen las métricas basadas en clases, en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaño para una clase OO se centran en el cálculo de atributos y de operaciones para una clase individual, y promedian los valores para el sistema OO en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.
A continuación se tratan en forma más exhaustiva cada una de estas.

4.4.1.2.1 Tamaño de Clase (TC)

El tamaño general de una clase se puede determinar empleando las medidas siguientes [16]:

- El número total de operaciones (tanto operadores heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.

- El numero de atributos (tanto heredados como privados de la instancia) que están encapsulados en la clase.

Si existen valores grandes de TC, éstos mostrarán que una clase puede tener demasiadas responsabilidades, lo cual reducirá la reusablidad de la clase, y complicará la implementación y la comprobación. Por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

En el estudio realizado en [14], se presentan algunas métricas de tamaño en común, tales como:

Stmts: Numero de declaraciones y sentencias ejecutables en los metodos de una clase. Esto solo puede generalizarse a una cuenta SLOC, simple.
NM (Numero de Métodos): Numero de métodos implementados en la clase
NAI (Numero de Atributos): Numero de atributos en una clase, (excluyendo los heredados). Incluye atributos que son tipos básicos tales como enteros o cadenas.

4.4.1.2.2 Número de Operaciones Añadidas por una subclase (NOA)

Las subclases se especializan mediante la adición de operaciones y atributos privados. A medida que crece el valor de NOA, la subclase deriva con respecto a la abstracción implicada por la superclase. En general, a medida que crece la profundidad de la jerarquía de clases (DIT se vuelve grande), el valor de NOA en los niveles inferiores de la jerarquía debería disminuir.

4.4.1.2.3 Número de Operaciones Invalidadas por una subclase (NOI)

Existen casos en que una subclase sustituye una operación heredada de su superclase, por una versión especializada para su propio uso, a esto se le denomina invalidación. Los valores altos de NOI suelen indicar un problema de diseño ya que si NOI es elevado, entonces el diseñador ha violado la abstracción implicada por la superclase. Esto da lugar a una jerarquía de clases débil y a un software OO, que pueda resultar difícil de comprobar y modificar [16].

4.4.1.2.4 Índice de Especialización (EI)

El índice de especialización proporciona una indicación aproximada del grado de especialización de cada una de las subclases existentes en un sistema orientado a objetos.
La especialización se puede alcanzar añadiendo o borrando operaciones, o bien por invalidación [16].

EI = [NOI x nivel] M(total)

en donde *nivel,* es el nivel de la jerarquía de clases en que reside la clase, y *M(total)* es el numero total de métodos para la clase. Cuanto mas elevado sea el valor de IE, es más probable que la jerarquía de clases tenga clases que no se ajustan a la abstracción de la superclase.

4.4.2 Métricas orientadas a operaciones

Dado que la clase es la unidad dominante en los sistemas OO, se han planteado menos métricas para las operaciones de clases. Churcher y Shepperd [16] describen esto cuando afirman:

> *Los resultados de los últimos estudios indican que los métodos tienden a ser pequeños tanto en términos del número de sentencias, como en términos de su complejidad lógica, lo cual sugiere que la estructura de conectividad de un sistema pueda resultar más importante que el contenido de los módulos individuales.*

Sin embargo, existen algunas ideas que pueden llegar a estimarse. Se indican a continuación tres métricas sencillas propuestas por Lorenz y Kidd [6] [7].

4.4.2.1 Tamaño medio de operación (TOavg)

Aunque se lograría utilizar las líneas de código como indicador para el tamaño de una operación, la medida LDC [16] padece de considerables problemas. Por esta razón el número de mensajes enviados por la operación proporciona una alternativa para el tamaño de la operación. A medida que asciende el número de mensajes enviados por una única operación, es posible que las responsabilidades no hayan sido bien estipuladas dentro de la clase.

4.4.2.2 Complejidad de operación (CO)

La complejidad de una operación se consigue calcular empleando cualquiera de las métricas de complejidad propuestas para el software convencional sabiendo que, a las operaciones convendría limitarlas a una responsabilidad específica, en donde el diseñador debería esforzarse por mantener el valor de CO tan bajo como sea posible [7].

4.4.2.3 Número Medio de Parámetros por Operación (NPavg).

Esta métrica define el promedio de parámetros por operación dentro del conjunto total de clases del sistema.

Cuanto sea más grande el número de parámetros de la operación, será más compleja la colaboración entre objetos. En general, NPavg debería de mantenerse tan bajo como sea posible [16].

4.5. Umbrales

Hatton [15] ha propuesto una explicación cognoscitiva acerca de por qué un efecto de umbral existiría entre el tamaño de un pieza de información y la propensión a fallas. La teoría propuesta esta basada en el modelo de memoria humana que consiste en *memoria a corto plazo* y *memoria a largo plazo*. Hatton [15] sostiene que los humanos pueden albergar, alrededor de siete (+/- dos) piezas de información en un momento dado en la memoria a corto plazo, independientemente del volumen de información. El autor se refiere a que los volúmenes de memoria a largo plazo están codificados y los códigos de recuperación pueden corromperse bajo determinadas condiciones.

La memoria a corto plazo incorpora un buffer que se refresca continuamente. El estudio sugiere que algo que puede caber en la memoria a corto plazo es más fácil de entender y menos probable que falle.

La teoría de Hatton [15] declara un componente de tamaño S, que cabe completamente en la memoria a corto plazo. Si el tamaño del componente excede S, entonces la memoria a corto plazo desborda. Esto hace al componente menos comprensible porque el código de recuperación que conecta la comprensión de la información con la memoria a largo plazo colapsa. Sugiere entonces un *modelo cuadrático que relaciona el tamaño del componente mayor que S con la incidencia de fallas*. Se suponen que hay un límite en la capacidad mental de los seres humanos, y que una persona no puede manipular la información eficazmente cuando la cantidad es mayor que ese límite.

Se cree que la herencia dificulta la comprensión del software orientado a objetos. Un estudio entre usuarios del paradigma OO, mostró que el 55% está de acuerdo en que la profundidad del árbol de herencia es un factor crítico al intentar entender el objeto de software [13]. Sin embargo, se ha argumentado que existe un incremento de inconsistencia mientras se desciende por la jerarquía de herencia (es decir, los niveles más profundos en la jerarquía se caracterizan por poseer extensiones incoherentes y/o especializaciones de las superclases) [13], por consiguiente las jerarquías de herencia no pueden diseñarse propiamente en la práctica. Según la teoría de Hatton [15], los objetos que son manipulados en la memoria a corto plazo, que poseen propiedades heredadas de objetos codificados en la memoria a largo plazo, requieren referenciar dicha memoria. Sin embargo, el acceso a la memoria a largo plazo rompe el hilo de pensamiento y es inherentemente menos exacto. Por consiguiente, acorde a esta teoría, es probable que las clases sean más propensas a fallas, si se usa herencia, y esta propensión a fallas aumenta, conforme aumenta la magnitud de la profundidad del árbol de herencia [14].

Las estrategias orientadas a objetos que limitan las responsabilidades de una clase reutilizándola en contextos múltiples, devino en una profusión de clases

pequeñas en los sistemas orientadas a objetos [13]. Dado que la mayoría de los sistemas OO se han diseñado con las características de reusabilidad, flexibilidad y mantenibilidad en mente, tienden a estar compuestos por un número alto de objetos pequeños y altamente cohesivos [20].

En detrimento del modelo, muchas clases pequeñas sugieren que habrá un alto grado de interacciones entre estas, incrementando así la complejidad del programa.

No todos los estudios que revisaron la relación entre el tamaño y fallas, encontraron un efecto de umbral. Por ejemplo, los estudios realizados por Basili [6] notaron que los componentes para los que tenían datos tendían a ser pequeños y, por consiguiente, no podrían observar el efecto en componentes de gran tamaño. Se pueden considerar las curvas en U *(Fig 4.5)* identificadas en [13][14], para encontrar un efecto de umbral. Sin embargo, trazar la densidad de fallas versus el tamaño, no tiene demasiado sentido.

Hatton [15] extiende este modelo al desarrollo orientado a objetos, donde postula el concepto de encapsulamiento, central al desarrollo orientado a objetos, el cual nos permite pensar sobre un objeto en aislamiento. Si el tamaño de este objeto es lo bastante pequeño para entrar en la memoria a corto plazo, entonces será más fácil de entender y razonar sobre el. Los objetos que son demasiado grandes y desbordan la memoria a corto plazo, tienden a ser más propensos a fallas.

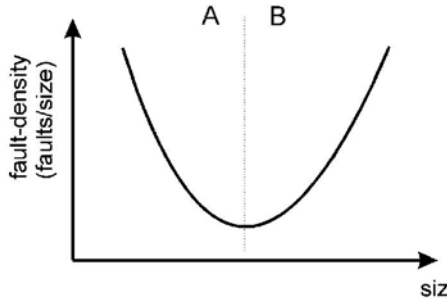

Figura 4.5 *Curva en U relacionando la densidad de fallas con el tamaño de la clase. Ejemplifica el tamaño óptimo teórico. La curva puede dividirse en dos partes, A y B [15].*

Ha habido interés en el software industrial que diseña la comunidad, en establecer umbrales para las medidas orientadas a objetos. Tal interés no fue dirigido por preocupaciones teóricas, o el deseo de probar una teoría, sino por necesidades pragmáticas. Los umbrales mantienen una manera simple, identificando las clases problemáticas [13]. Si una clase tiene un valor de medición mayor que el umbral, entonces se la identifica para su investigación.

Lorenz y Kidd presentan un catálogo de umbrales [16] basado en experiencias con sistemas escritos en C++, y proyectos en Smalltalk. En [13] se presentan una serie de umbrales para varias métricas orientadas a objetos, por ejemplo, se propone un umbral del tamaño de 40 métodos por clase.

En ninguno de los trabajos mencionados se realizó una comprobación sistemática para demostrar que, las clases que excedieron el tamaño del umbral, eran de hecho más problemáticas que aquéllos que no lo hacían, por ejemplo, que eran más probables de contener una falla.

4.6 Conclusiones

El Software Orientado a Objetos (OO) es fundamentalmente distinto del software que se desarrolla utilizando métodos convencionales. Por lo tanto, las métricas para sistemas OO se centran en mediciones que se pueden aplicar a las características de diseño de las clases (localización, encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos), que hacen única a una clase.

El conjunto de métricas CK define seis métricas de software orientado a clases, que hacen especial hincapié en estas, y en la jerarquía del árbol de herencia de clases. Este conjunto de métricas desarrolla también métricas para estimar las comprobaciones entre clases y la cohesión de los métodos que residen dentro de una clase. En un nivel orientado a clases, se puede complementar con las métricas propuestas por Lorenz y Kidd. Entre estas se encuentran las medidas de *tamaño* de una clase, y las métricas que ofrecen ideas generales acerca del grado de especialización de las mismas.

Las métricas orientadas a operaciones hacen hincapié en el tamaño y complejidad de las operaciones individuales, sin perder de vista que el objetivo principal de las métricas de diseño, se encuentra en el nivel de clase.

Los umbrales aplicados a las mediciones tomadas, tienen como rol principal determinar puntos de inflexión, respecto de los resultados obtenidos, pudiendo así determinar si un diseño dado, es correcto o necesita ser sometido a un nuevo análisis.

Las características mensurables del modelo de análisis y diseño, pueden ayudar al gestor del proyecto de un sistema OO, a la hora de planificar y continuar las actividades.

En resumen, la reseña se enfoca principalmente en las investigaciones que han establecido el desarrollo y refinamiento de métricas para sistemas OO, y en consecuencia el desarrollo de umbrales teóricos para la evaluación y análisis estadístico de los resultados obtenidos.

La tendencia del trabajo a futuro, debería seguir en el sentido de la definición de umbrales que determinen estándares de medición, con el objetivo de que el ingeniero de software tenga una base comparable respecto a los resultados obtenidos en sus mediciones. Por otra parte cobra gran importancia, el desarrollo de herramientas que proporcionen estos valores de umbral, y que permitan realizar mediciones de los diseños.

4.7 Glosario de Términos

Termino	Descripción
WM	Weighted Method Send-Out
LR	Logistic Regression
OO	Object-Oriented
WMC	Weighted Method Class
DIT	Deep in Inheritance Tree
NOC	Number of Children
NOC*	Number of Children in Sub Tree
DAG	Directed Acyclic Graph
NDC	Number of Descendants Classes
RFC	Response For a Class
CBO	Coupling Between Objects
CBO_NA	Coupling Between Objects, No Ancestors
CBO_IUB	Coupling Between Objects, Is Used By
CBO_U	Coupling Between Objects, Using
LCOM	Lack of Coupling between Objects Methods
EI	Specialized Index.
TU	Total Understandability
AU	Average Understandability
TM	Total Modifiability
AM	Average Modifiability
CRC	Class Responsibility Collaboration
NM	Number of Methods
NAI	Number of Attributes
CRC	Clase Responsabilidad Colaborador

CAPITULO 5 REGRESION LOGISTICA

5.1. Metodología de la investigación

El método utilizado para llevar a cabo el análisis es la regresión logística (LR). La regresión logística se utiliza para construir modelos cuando la variable dependiente es binaria.

Se dice que un proceso es binomial cuando sólo tiene dos posibles resultados: "éxito" y "fracaso", siendo la probabilidad de cada uno de ellos constante en una serie de repeticiones. A la variable número de éxitos en n repeticiones se le denomina variable binomial. A la variable resultado de un sólo ensayo y, por tanto, con sólo dos valores: 0 para fracaso y 1 para éxito, se le denomina binomial puntual.

Un proceso binomial está caracterizado por la probabilidad de éxito, representada por p (es el único parámetro de su función de probabilidad), la probabilidad de fracaso se representa por q y, evidentemente, ambas probabilidades están relacionadas por p+q=1. En ocasiones, se usa el cociente p/q, denominado "odds", y que indica cuánto más probable es el éxito que el fracaso, como parámetro característico de la distribución binomial aunque, evidentemente, ambas representaciones son totalmente equivalentes.

Los modelos de regresión logística son modelos de regresión que permiten estudiar si una variable binomial depende, o no, de otra u otras variables (no necesariamente binomiales): Si una variable binomial de parámetro *p* es independiente de otra variable X, se cumple p= p|X, por consiguiente, un modelo de regresión es una función de *p* en X que a través del coeficiente de X permite investigar la relación anterior.

La regresión logística es útil cuando se trata de predecir el valor de una variable respuesta dicotómica Y, esto es, una respuesta binaria del tipo 0/1, ausente/presente, sano/enfermo, etc., que presumiblemente depende de otras m variables explicativas (Xj, j= 1, ..., m) a través del modelo de probabilidad [25]

$$\Pr\{Y_i = 1\} = \frac{1}{1 + \exp(-\beta_0 - \beta_1 x_1 - \ldots - \beta_m x_m)}$$

Los n vectores muestrales se agrupan en la matriz

$$\begin{pmatrix} (y_1, x_{11}, x_{12}, \ldots, x_{1m}), \\ (y_2, x_{21}, x_{22}, \ldots, x_{2m}), \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ (y_n, x_{n1}, x_{n2}, \ldots, x_{nm}) \end{pmatrix}$$

donde los yi sólo pueden tomar los valores 0 ó 1.

El estimador de máxima verosimilitud del vector paramétrico,

$$\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_m)^T,$$

se calcula mediante un procedimiento iterativo del tipo Newton-Raphson.

Es posible que algunas de las supuestas variables explicativas no sean tales y no tengan ningún efecto sobre la variable respuesta; para poder identificarlas y eliminarlas del modelo, se recurre a la prueba *de Wald*, la cual se limita a contrastar la hipótesis de nulidad del coeficiente β_j asociado a la variable Xj:

H0j: "Xj no influye sobre Y: β_j =0"

frente a la alternativa:

H1j: "Xj influye sobre Y: $\beta_j \neq 0$"

El estadístico de contraste para la j-ésima variable explicativa es

$$W_j = \frac{\hat{\beta}_j^2}{s_j^2}$$

que se distribuye como una x^2 con 1 grado de libertad cuando la muestra es grande, siendo s2j la varianza del estimador de β_j . x^2

Lo que se pretende determinar mediante la utilización del modelo de Regresión Logística (LR), es si mediante la relación entre las variables como la propensión a fallas o la complejidad cognoscitiva, existen umbrales (en este caso la (LR)), determinaría la presencia o ausencia del umbral), y en caso de existir, cual seria los valores de umbral para cada tipo de relación.

Por ejemplo un caso a determinar seria si, el tamaño de una clase tiene influencia en la propensión a errores, o en la complejidad cognoscitiva de su comportamiento [25] [26].

5.2. DISEÑO DETALLADO Y JUSTIFICACION DEL METODO

5.2.1 Regresión logística binaria

Introducción

El objetivo es comprender el uso adecuado de la regresión cuando la variable dependiente es categórica dicotómica. La regresión logística está diseñada para utilizar una mezcla de predictores categóricos y continuos para predecir una respuesta categórica [26].

5.2.2 ¿Por qué no utilizar regresión lineal?

Existen dos líneas argumentales de porqué la regresión lineal no es apropiada para cuando la variable dependiente es dicotómica. El primero es conceptual: Si tratamos de predecir una respuesta codificada como 0 o 1, podemos considerar que las predicciones son probabilidades, esto es la probabilidad de obtener un 1. Pero al ajustar una línea a valores que solamente son 0 o 1 podemos obtener valores predichos menores que 0 o mayores que 1. Si bien es posible reajustar posteriormente los valores extremos para que se ajusten a los límites entre 0 y 1, no hay ninguna razón teórica que impida que la regresión lineal los produzca en primera instancia, obligándonos a introducir modificaciones posteriores. El segundo es técnico: La regresión lineal asume homogeneidad de varianza de los residuos a lo largo de la

recta. Si estamos trabajando con probabilidades o proporciones, se puede demostrar que la distribución de una proporción

$$\sqrt{((p)(1-p))}_p$$

con media p tiene un desvío típico de (()(1 −p)). Dado que hay una relación funcional entre el desvío típico y la media, no es posible asumir homogeneidad. Anteriormente esta situación se solucionaba mediante una variante técnica llamada regresión por mínimos cuadrados ponderados, pero el problema de valores menores a cero o superiores a uno se mantenía en tales regresiones. La regresión logística fue desarrollada en 1960 como una solución a estos problemas. Cuando se predice el valor de una variable que varía en una escala entre 0 y 1, tiene sentido ajustarla a una curva como se muestra en la Figura 5.1. Imaginemos que estamos tratando de predecir la probabilidad de que alguien compre una casa utilizando ingreso como predictor. Cuando el ingreso crece, más y más gente es capaz de alcanzar la compra de su casa, y entonces la tasa de incremento en la probabilidad de compra por unidad de ingresos también crece. En los altos niveles de ingresos, la mayoría de la gente puede comprar su casa, (pero recordemos que la probabilidad debe ser menor o igual a uno), por lo que la curva se aplana, de manera que la tasa de incremento en la probabilidad de compra por unidad de ingresos decrece. El resultado es una curva con forma de S o curva logística [25].

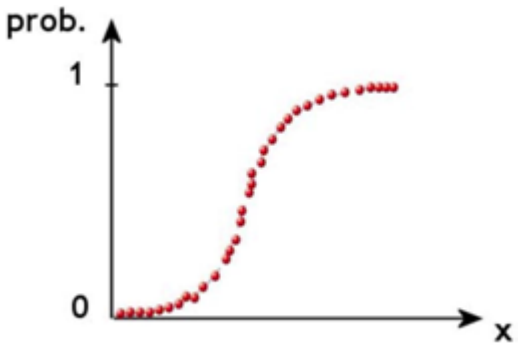

***Figura 5.1**: Curva logística.*

La regresión logística está limitada a cero y uno, por lo que no pueden producirse predicciones erróneas. Realmente existe toda una familia de funciones con forma de S. Nosotros nos ocuparemos solamente de la regresión logística. Antes de comenzar con la regresión logística propiamente dicha revisaremos los denominados procesos binomiales que sirven como base y sustento de la misma [26].

5.2.3 Asociación entre variables binomiales

Se dice que un proceso es binomial cuando sólo tiene dos posibles resultados: "éxito" y "fracaso", siendo la probabilidad de cada uno de ellos constante en una serie de repeticiones. A la variable número de éxitos en n repeticiones se le denomina variable binomial. A la variable resultado de un sólo ensayo y, por tanto, con sólo dos valores: 0 para fracaso y 1 para éxito, se le denomina binomial puntual. Un proceso binomial está caracterizado por la probabilidad de éxito, representada por p (es el único parámetro de su función de probabilidad), la probabilidad de fracaso se representa por q y, evidentemente, ambas probabilidades están relacionadas por p+q=1. En ocasiones, se usa el cociente p/q, denominado "odds", y que indica cuánto más probable es el éxito que el fracaso, como parámetro característico de la distribución binomial aunque, evidentemente, ambas representaciones son totalmente

equivalentes. Los modelos de regresión logística son modelos de regresión que permiten estudiar si una variable binomial depende, o no, de otra u otras variables (no necesariamente binomiales): Si una variable binomial de parámetro p es independiente de otra variable X, se cumple p=p|X, por consiguiente, un modelo de regresión es una función de p en X que a través del coeficiente de X permite investigar la relación anterior. Supongamos una prueba clínica donde se define a la variable tratamiento como X=1 para el tratamiento A y X=0 para el B, en esta prueba se registra el número de pacientes curados (este es el éxito) y las de los no curados (fracaso) [25].

Entonces la probabilidad de curación para el tratamiento B es: p|(X=0) y para el tratamiento A: p|(X =1) (ec.1)

Si ambas probabilidades resultaran distintas podría *"parecer"* que la probabilidad de curación depende del tratamiento. Aquí se nos plantean varias preguntas:

¿Esta dependencia es generalizable ("estadísticamente significativa")? ¿Cuánto depende ("clínicamente relevante")? La primera pregunta la podemos resolver mediante la prueba x^2, la segunda mediante las denominadas "medidas de asociación", o "de fuerza de la asociación", o "de efecto": diferencia de riesgo (DR), riesgo relativo (RR) y "odds ratio" (OR).

Para nuestro ejemplo:

$$DR = p_A - p_B = p|(X=1) - p|(X=0) \quad \textit{(ec.2)}$$

$$RR = p_A / p_B = \frac{p|(X=1)}{p|(X=0)} \quad \textit{(ec.3)}$$

$$OR = \frac{p_A / q_A}{p_B / q_B} = \frac{\dfrac{p|(X=1)}{1-(p|(X=1))}}{\dfrac{p|(X=0)}{1-(p|(X=0))}} \quad \textit{(ec.4)}$$

DR es 0 cuando no hay diferencias y RR y OR son ambos 1. Recordemos que el OR, aunque es la medida menos intuitiva, es la más extendida por diversas razones y que es conveniente que a estas estimaciones puntuales las acompañemos de su intervalo de confianza que nos indica la precisión de la estimación.

5.2.4 La ecuación logística

La regresión logística es la aplicación de una regresión a una variable dependiente dicotómica. La relación lineal no ocurre en la escala de los datos crudos o de las probabilidades de que ocurra un evento sino en el logaritmo de los odds del evento de interés. La ecuación general para la regresión logística es:

$$\ln(Odds) = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X|_n \quad \textit{(ec.5)}$$

Hay varias razones para plantear el modelo con el logaritmo del odds, en lugar de plantearlo simplemente con la probabilidad de éxito o con el odds. En primer lugar, el campo de variación de ln(Odds) es todo el campo real (de -∞ a ∞), mientras que, para p el campo es sólo de 0 a 1 y para p/q de 0 a ∞ . Por lo tanto, con el modelo

logístico no hay que poner restricciones a los coeficientes que complicarían su estimación. Por otro lado, y más importante, en el modelo logístico los coeficientes son, como veremos enseguida, fácilmente interpretables en términos de independencia o asociación entre las variables. El término *ln(Odds)* es también conocido como logit. Recordemos que la relación entre el odds y la probabilidad es:

$$Odds = \frac{p}{1-p} \quad (ec.6)$$

Nótese que hay relación lineal entre las variables independientes en la regresión logística, pero es con el logaritmo de los odds y no con las probabilidades. Dado que estamos interesados en la probabilidad de un evento (el valor superior de una variable dicotómica), combinaremos las dos ecuaciones anteriores para obtener la probabilidad.

$$p = \frac{e^{\alpha+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_n X_n}}{1+e^{\alpha+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_n X_n}}, \text{ o lo que es lo mismo: } p = \frac{1}{1+e^{-(\alpha+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_n X_n)}} \quad (ec.7)$$

Estas dos últimas expresiones, si son conocidos los coeficientes, permiten calcular directamente la probabilidad del proceso binomial para los distintos valores de la variable X.

A la función:

$$f_{(z)} = \frac{1}{1+e^{-(z)}} \quad (ec.8)$$

que aparece en otros muchos campos de la matemática aplicada se le denomina función logística. El modelo de regresión logística, por tanto, modeliza la probabilidad de un proceso binomial como la función logística de una combinación lineal de la(s) variable(s) dependiente(s) [25] [26].

Es evidente que la ecuación 7 no se puede derivar por el proceso de mínimos cuadrados de la regresión lineal. En su lugar se ha implementado un proceso iterativo de búsqueda de los parámetros de la ecuación mediante el algoritmo de máxima verosimilitud.

5.2.5 Elementos de la regresión logística

Hay dos objetivos generales cuando se realiza una regresión logística:

1) Determinar el efecto de un conjunto de variables sobre la probabilidad y además el efecto de las variables individuales.

2) Obtener la mayor precisión predictiva posible para un conjunto dado de predictores.

Estos dos objetivos no son mutuamente incompatibles, pero uno u otro tienden a ser el foco del análisis. Aquellos interesados en la teoría y en efectos causales están típicamente interesados por el primer objetivo, mientras que aquellos interesados en

predecir si un evento futuro caerá en una u otra categoría de la variable dependiente se focalizarán sobre el segundo. Muchos de los pasos al realizar un modelo de regresión logística son similares a una regresión lineal. Se debe seleccionar un conjunto de predictores razonables y se debe examinar previamente y con cuidado los datos para detectar patrones inusuales, valores extremos, problemas de valores perdidos, etc. Luego de estimar la ecuación y examinar los efectos de las variables individuales, se debería realizar unas pocas pruebas para comprobar si los datos cumplen los supuestos del modelo logístico y si existen casos que ejerzan excesiva influencia en los resultados. Es muy importante validar el modelo si se está interesado en predecir la pertenencia a una categoría para datos futuros. Existe más de una manera de evaluar el ajuste del modelo y más de una manera de evaluar la cantidad de varianza explicada. Dado que la media y la varianza están relacionadas fue difícil inicialmente encontrar un equivalente razonable del R2 de la regresión lineal. Actualmente existen pseudo R2 que tratan de cumplir esta misma función. La bondad de ajuste o significación de un modelo no es necesariamente igual a la mayor precisión predictiva. Cuando se incrementa el tamaño de la muestra un conjunto de variables independientes puede ser estadísticamente significativa, pero aún así no alcanzar un alto porcentaje de predicciones correctas. La clasificación de los casos es simple de realizar en regresión logística. Un caso se asigna a la categoría menor de la variable dependiente si su probabilidad predicha es menor a 0,5; en caso contrario se asigna a la categoría superior [25].

5.2.6 Supuestos de la regresión logística

La regresión logística requiere menos supuestos que la regresión lineal:

1) Las variables independientes pueden ser intervalares, de tasa o dicotómicas.
2) Todos los predictores relevantes son incluidos, no se incluye ningún predictor irrelevante y la forma de la relación es lineal.
3) El valor esperado del término del error es cero.
4) No hay autocorrelación.
5) No hay correlación entre el error y las variables independientes.
6) No hay multicolinealidad entre las variables independientes.

Los dos supuestos siguientes valen para la regresión lineal pero no para la regresión logística:

1) *Normalidad de los errores:* se supone que los errores se ajustan a una distribución binomial, quienes sólo se aproximan a una normal en muestras muy grandes.

2) *Homogeneidad de varianza*: como se discutió anteriormente, esta condición no se puede alcanzar por definición.

La utilización de grandes cantidades de variables dicotómicas como predictores no viola ningún supuesto de la regresión logística. Volveremos al tema de los supuestos luego de ejecutar una regresión logística.

Se debe mencionar otro punto más. Manteniendo todo igual, la regresión logística requiere mayores tamaños muestrales que la regresión lineal para una inferencia correcta. Aunque los expertos no han llegado a un acuerdo al respecto, una regla razonable es tener al menos 30 veces tantos casos como parámetros a ser estimados en el modelo [25] [26].

CAPITULO 6 UMBRALES PARA METRICAS OO

6.1 MARCO TEORICO Y DISEÑO DE LA INVESTIGACION

Una aplicación practica de las métricas Orientadas a Objetos (OO), es predecir que clases son más probables que contengan fallas. Esto tiende a ser significante dado que se piensa que las métricas OO, son indicadores de complejidad psicológica y, las clases que son más complejas son más probables que contengan fallas.

Recientemente se ha propuesto una teoría cognitiva basada en estudios de Alzheimer [28], la cual favorece la existencia de un sistema de razonamiento dual o de dos niveles, el cuál sugiere además que existe un efecto de umbral para varias métricas OO. Un esquema simplificado del razonamiento humano, se presenta en la figura 6.1.

Esto significa que, las clases son fáciles de comprender mientras sus medidas de complejidad permanezcan por debajo de un valor de umbral. Por encima del valor de umbral entonces, el entendimiento decrece rápidamente, llevando a una probabilidad de fallas creciente. Esto ocurre, acorde a esta teoría, debido a que la memoria humana de corto plazo colapsa. Si esta teoría se confirma, proveerá un mecanismo que explicaría la introducción de errores en los sistemas OO, y proveería también una guía practica de cómo diseñar programas OO.

Se ha realizado una gran cantidad de trabajo, con el propósito de investigar la relación entre las métricas OO y la propensión a fallas en las clases. Una clase propensa a fallas, se define como *aquella que tiene una alta probabilidad de tener una falla, que cause un error una vez que el sistema ha sido puesto en producción*.

Una vez validadas, estas métricas pueden servir como indicadores que llevan a clases propensas a errores. Tales clases pueden identificarse para someterlas a acciones específicas de administracion de calidad, tales como inspecciones más intensas y testeo o, más aun, pueden rediseñarse.

Un enfoque operacional que apela a la administracion de la calidad utilizando métricas OO, es el desarrollo de umbrales. Los umbrales se definen como "*Valores heurísticas usados para fijar rangos de valores deseables y no deseables de métricas, para el software medido*". Estos valores de umbral se utilizan para identificar anomalías. Por ejemplo podríamos decir que una métrica dada que mide acoplamiento, tiene un valor de umbral de 6; Si el valor de la medición es mayor a 6, entonces podríamos identificar a esa clase como de alto riesgo.

Los umbrales tienen un significado práctico, teórico y metodológico. Es mucho más simple para el equipo de SQA, utilizar umbrales para identificar clases con un alto riesgo potencial. Más aun, Hatton [23] presento el caso de umbrales basados en la teoría cognitiva. Específicamente, utiliza el modelo de memoria humana presentado en [28], para sugerir que, clases mas complejas, colapsaran la memoria de corto-plazo, llevando a mas fallas. Como disciplina, es importante testear empíricamente la verosimilitud de esta teoría dado que, si se verifica, puede mejorar ampliamente nuestro entendimiento del diseño OO. Si la teoría de umbrales se confirma, proveerá un mecanismo para explicar la introducción de errores en un sistema OO. Finalmente, si ciertamente existen umbrales, entonces los modelos empíricos utilizados para validar métricas OO, abordaran los datos mucho mejor; Esto resultaría en una mejora en la predictibilidad de clases con alto riesgo.

Una forma práctica de ver los umbrales puede ser la de utilizarlos como una alarma que se dispara cada vez que el valor de una métrica interna excede el valor de umbral.

Al día de hoy, ninguno de los estudios realizados para determinar valores de umbral, han sido testeados empíricamente. En este trabajo, se presenta una técnica estadística para estimar y evaluar umbrales, y aplicarla en un subconjunto de métricas de la suite de Chidamber & Kemerer (CK). Debe notarse que por razones históricas, las métricas CK son las más referenciadas.

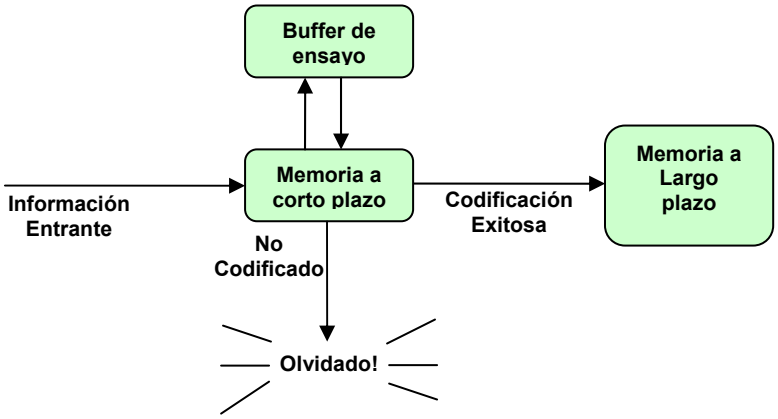

Figura 6.1 *– Esquema simple de las propiedades psicológicas del sistema de memoria humana [23]*

6.2 Background

6.2.1 Teoría y evidencia del efecto de los umbrales.

En la figura 6.2, se resumen las bases teóricas para el desarrollo de modelos cuantitativos, que relacionan métricas de productos y métricas de calidad externas. Aquí se plantea la hipótesis de que las propiedades estructurales de un componente de software, tales como su acoplamiento, tiene impacto en su complejidad cognitiva. La complejidad cognitiva se define como *la carga mental de los individuos que tienen que tratar con el componente*, por ejemplo, desarrolladores, testers, inspectores, y mantenedores del sistema [23]. Complejidad cognitiva alta, lleva a que un componente exhiba atributos externos de calidad indeseables, tales como aumento de la propensión a fallas y reducción en la mantenibilidad. Acorde a esta teoría, las métricas OO que afecten la complejidad cognitiva, serán por lo tanto, relacionadas con la propensión a fallas.

Típicamente, propiedades estructurales tales como el acoplamiento y cohesión, se considera que ejercen una influencia significativa sobre la complejidad cognitiva. Por ejemplo, los sistemas compuestos de clases altamente acopladas tienden a ser propensas a error, difíciles de entender y difíciles de mantener. Por otro lado, los sistemas altamente cohesivos y de bajo acoplamiento, tienden a ser menos propensos a error, fáciles de corregir y adaptar a nuevas características.

La teoría expuesta, no plantea ninguna hipótesis respecto de un efecto de umbral específico. Sin embargo, Hatton [23], propone una explicación cognitiva de por que existe un efecto de umbral entre las métricas de complejidad y los errores.

La explicación cognitiva propuesta se basa en el modelo de memoria humana, el cual consiste en una memoria de corto plazo y una memoria de largo plazo. Ver figura 6.1. Hatton [23] argumenta que las personas pueden mantener al rededor de 7 +/- 2 piezas de información en un momento dado en la memoria a corto plazo, independientemente del contenido de la información. Luego nota que el contenido en la memoria a largo plazo se encuentra almacenado en forma codificada y que los códigos de recuperación de la información pueden ser confusos bajo determinadas circunstancias. La memoria a corto plazo incorpora además un buffer de ensaño el cual se refresca a si mismo continuamente. Sugiere que cualquier cosa que entre dentro de la memoria de corto plazo, es mas fácil de entender y menos propenso a error. Las piezas de información que son demasiado grandes o demasiado complejas

colapsan la memoria a corto plazo, involucrando el uso de mecanismos de recuperación codificados más propensos a error, utilizados en la memoria a largo plazo.

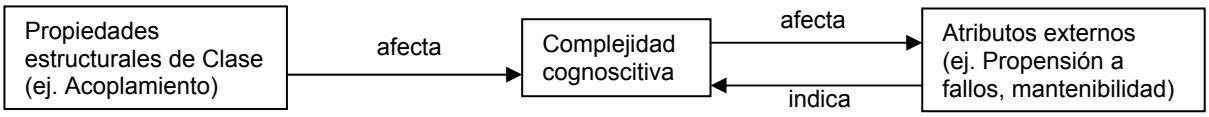

Figura 6.2 *- Bases teóricas para el desarrollo de métricas para artefactos OO [24].*

6.2.1.1 Umbrales de Tamaño

Hatton [23] argumenta que el concepto de encapsulamiento, central en el paradigma de objetos, nos permite pensar a cerca de un objeto en aislamiento. Si el tamaño de este objeto es lo suficientemente pequeño para que quepa en la memoria a corto plazo, entonces será fácil entender y razonar respecto de este objeto. Los objetos que son muy grandes y colapsan la memoria a corto plazo, tienden a ser más propensos a error. Sin embargo, un estudio reciente ha demostrado que no existe un umbral de tamaño para las clases OO [24]. Por lo tanto, no se consideraran este tipo de umbrales.

6.2.1.2 Umbrales de Herencia

Se piensa que la herencia dificulta el entendimiento del software OO. Es notorio que la herencia da lugar a descripciones de clases distribuidas. Es decir que, la descripción completa de una clase A, solo puede ensamblarse examinando A tanto como las superclases de A. Dado que diferentes clases se describen en diferentes lugares dentro del código fuente de un programa, no hay un lugar único donde un programador pueda acudir para obtener una descripción completa de una clase. Mientras que este argumento se base en términos de código fuente, no es difícil generalizar a documentos de diseño. Para entender el comportamiento de un método, se debe rastrear las dependencias de la herencia, lo cual es considerablemente complejo debido al binding dinámico. En un estudio propuesto en [24], se propone que si la jerarquía de herencia esta diseñada de manera apropiada, entonces el efecto de distribuir funcionalidad sobre la jerarquía de herencia no va en detrimento del entendimiento. Sin embargo, se ha argumentado que existe una creciente inconsistencia conceptual conforme se viaja por la jerarquía (los niveles inferiores en la jerarquía se caracterizan por ser extensiones inconsistentes o especializaciones de las superclases), por lo tanto las jerarquías de herencia no pueden ser diseñadas adecuadamente en la practica.

Acorde a la teoría de Hatton [23], los objetos que son manipulados en la memoria a corto plazo que poseen propiedades heredadas de objetos codificados en la memoria de largo plazo, requieren referenciar a la memoria de largo plazo. Sin embargo, acceder a la memoria a largo plazo rompe el tren de pensamiento y es inherentemente menos exacto [23]. Por lo tanto, acorde a esta teoría, es probable que las clases sean mas propensas a error si utilizan herencia, y la propensión a fallos aumenta conforme la magnitud de la jerarquía de herencia aumenta.

6.2.1.3 Umbrales de Acoplamiento

Las estrategias OO de limitar las responsabilidades de una clase y reutilizarla en múltiples contextos, resulta en una profusión de pequeñas clases en los sistemas OO. Por ejemplo Chidamber & Kemerer [6] encontraron en dos sistemas estudiados que la mayoría de las clases tienden a tener un pequeño numero de métodos (0-10), lo cual sugiere que la mayoría de las clases son relativamente simples en su

construcción, proporcionando abstracción y funcionalidad especifica. Otro estudio llevado a cabo en Bellcore [3], encontró que la mitad de los métodos tiene menos de 4 líneas SmallTalk o 2 sentencias C++, sugiriendo que las clases están implementadas con métodos pequeños. Muchas clases pequeñas, significa que habrá muchas interacciones entre estas clases.

Se cree que esta situación incrementa la complejidad del programa, dado que los programadores tienen que entender el contexto de uso de un método, rastreando a través de la cadena de llamados que lo alcanzan, y rastreando la cadena de métodos que este utiliza. Al haber muchas interacciones, se exacerba el problema del entendimiento.

La teoría de Hatton declara que cuando hay una difusión de la funcionalidad, entonces un objeto en la memoria a corto plazo puede estar haciendo referencia a varios objetos en la memoria a largo plazo. Por lo tanto esto conlleva a dificultades en la comprensión y, acorde a la figura 6.2, mayor propensión a fallas. Por lo tanto se puede argumentar que, cuando los objetos que interactúan colapsan la memoria a corto plazo, se incrementa la propensión a errores.

6.2.2 Métricas estudiadas

A continuación se presenta un breve sumario de las métricas CK que se han analizado. Se ha excluido explícitamente la métrica de cohesión LCOM, dado que no existe una razón a priori, basada en la teoría presentada, para creer que podría exhibir un valor de umbral.

6.2.2.1 WMC (Weighted Methods per Class)

Esta métrica puede clasificarse como una métrica de complejidad tradicional. Básicamente es la cuenta de los métodos en una clase. Los desarrolladores de esta métrica, dejan el esquema de peso, como una decisión de implementación. En el presente estudio, se ha utilizado complejidad ciclomatica. Sin embargo, otros autores no adoptan un esquema de peso. Los métodos en clases ancestras, no se tienen en cuenta. Basado en sus experiencias con proyectos OO en NASA GSFC, Rosenberg [27] define un umbral para WMC de 100.

6.2.2.2 DIT (Depth in Inheritance Tree)

Esta métrica se define como la longitud del camino mas largo desde la clase raíz en la jerarquía de herencia. Se ha declarado que cuanto mas abajo se va en la jerarquía de herencia, las clases se vuelven mas complejas y, por lo tanto, mas propensas a error.

6.2.2.3 NOC (Number Of Children)

Esta métrica cuenta el número de clases que heredan de una clase dada, es decir, el número de clases en el árbol de herencia hijos de una clase determinada. No se ha determinado un umbral específico para esta métrica.

6.2.2.4 CBO (Coupling Between Object Classes)

Se dice que una clase esta acoplada con otra, si los métodos de una clase, utilizan métodos o atributos de otra y viceversa. En esta definición, los usos pueden significar tipos miembro, parámetros o variables de métodos locales. CBO es el nro. de clases a las cuales una clase dada esta acoplada. Incluye el acoplamiento basado en

[3] El estudio consistió en analizar sistemas en C++ y SmallTalk y entrevistar a los desarrolladores de ambos sistemas. Para el sistema en C++, la métrica de tamaño fue el número de sentencias ejecutables, mientras que para el sistema en SmallTalk el tamaño fue medido en líneas de código.

herencia (acoplamiento entre clases relacionadas vía herencia). Rosenberg [27] define un umbral para CBO de 5.

6.2.2.5 RFC (Response for a Class)

El conjunto de respuestas de una clase, consiste del conjunto de N métodos de la clase, y el conjunto de métodos invocados directamente por los métodos en N (el conjunto de métodos que pueden ser potencialmente ejecutados en respuesta a un mensaje recibido por la clase.). RFC es el número de métodos en el conjunto de respuesta de la clase. Rosenberg [27] derivo un umbral de 100 para RFC.

6.3. Planteamiento del problema

La medición es una actividad fundamental a cualquier disciplina de ingeniería, y la ingeniería de software no es la excepción. Típicamente, las métricas son esenciales en la ingeniería de software dado que proveen mecanismos para la medición de la complejidad y calidad, estimando costos y esfuerzo invertido en un proyecto, solo por mencionar algunos aspectos [16].

En una disciplina de ingeniería no podemos tolerar ambigüedades. Si vamos en busca de rigor, las herramientas de lógica matemática y el razonamiento formal son cruciales, aunque estas no sean cuantitativas.

El concepto de ***calidad,*** dentro de la Ingeniería de Software, se ha convertido en uno de los objetivos principales durante el ciclo de construcción de un sistema. Pero dado que este puede tener implicancias diferentes según las necesidades del observador, no necesariamente los aspectos de calidad que se quieren cubrir, son los mismos [24]. Aquí es donde se vuelve fundamental el rol de las métricas durante el ciclo de desarrollo, dado que las mismas entregan mediciones sobre diferentes aspectos, tanto del proceso de desarrollo de software, como de la construcción de las clases (entidades) y relaciones, que conforman el sistema resultante [22].

Los números entregados por las métricas nos ayudan a entender y controlar el proceso de ingeniería.

Dado que el Software Orientado a Objetos (OO) es fundamentalmente distinto del software que se desarrolla utilizando métodos convencionales, las métricas para sistemas OO deben ajustarse a las características que lo distinguen del software convencional [16].

Las métricas OO hacen hincapié en conceptos tales como el encapsulamiento, herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas OO se centran en las mediciones que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hacen única a una clase [16].

Hemos visto varios modelos propuestos de métricas OO y técnicas de medición, tanto orientadas al tamaño de las clases como a la complejidad de las mismas, y al nivel de acoplamiento entre clases de un sistema. Pero una vez determinado lo que se va a medir, como medirlo y que técnica utilizar, debemos contar con algún criterio de evaluación para poder determinar si el aspecto medido se halla dentro de los valores aceptados, o si es necesario el rediseño del modelo.

6.3.1 Análisis de los diferentes aspectos del problema (dimensiones)

Las propiedades estructurales de un componente de software (como su acoplamiento) tienen un impacto asociado a su complejidad cognoscitiva. La complejidad cognoscitiva es definida como la carga mental de los individuos quienes tienen que tratar con dichos componentes, por ejemplo, diseñadores, verificadores, inspectores, y mantenedores. La alta complejidad cognoscitiva lleva a que un componente exhiba cualidades externas indeseables, como el incremento en la propensión a fallas, y el aumento del costo de mantenimiento.

6.3.2. Formulación del problema concreto

Una vez que el ingeniero de software realiza las mediciones susceptibles a un proyecto dado, y obtiene los valores que arrojan las métricas utilizadas para tal fin, ¿que decisiones puede tomar frente a estos datos?, ¿Cómo puede determinar si estos datos que arrojo la medición, están dentro de valores aceptables?, ¿Requerirán las estructuras medidas, algún tipo de rediseño o refactoring?, ¿Cómo está afectando la complejidad cognoscitiva de las clases, a las características externas? En definitiva, una vez obtenidos estos datos, deseamos saber ***¿que significan?***, para poder tomar acciones correctivas o no, dependiendo de los valores obtenidos [22].

Presentado el valor 4, como una media promedio de la cantidad de argumentos para una librería de clases, no se debe necesariamente saber que significa, (bueno, malo, no pertinente). Evaluado contra las medidas publicadas de aceptación, o contra las medidas realizadas en proyectos anteriores, el valor de la medida realizada entregara información más significativa [22].

Particularmente significativos son los puntos periféricos, si el valor medio para una propiedad dada es 5 con una desviación normal de 2, y la medición tomada arroja un valor de 10 para un nuevo desarrollo, probablemente valga la pena realizar una verificación posterior, asumiendo por supuesto que hay alguna teoría para apoyar la asunción que la medida es relevante al proyecto [22].

6.3.3. Definición de las dimensiones incluidas en el estudio

En el presente caso de estudio la determinación de las variables y dimensiones se vuelve un tanto difusa, debido a la relación que existe entre las variables a medir. Pero en primer lugar, expongamos que entendemos por variables y dimensiones.

> *Por dimensión entendemos **un componente significativo de una variable que posee una relativa autonomía.** Nos referimos a componentes porque estamos considerando a la variable como un agregado complejo de elementos que nos dan un producto único, de carácter sintético [29].*

6.3.3.1 Variables, tipo y definición

Las variables que se pretende encuadrar en el marco de referencia del presente trabajo son:

- El Tamaño de una clase
- La Profundidad y el ancho de una jerarquía de herencia de clases.
- El Grado de Acoplamiento entre clases.

Una vez expuestas las variables, veamos que queremos significar respecto de cada uno de estos aspectos a tratar.

Tamaño de una clase:

Por tamaño de una clase, se entiende la cantidad de métodos que la clase implementa, ya sean públicos y/o privados, y por cada uno de estos métodos, la cantidad de líneas de código en cada método.

Las métricas asociadas a esta variable podrían ser, SLOC (Source Lines Of Code), RFC (Response For a Class), y WMC (Weighted Method Class).

Profundidad y Ancho de una jerarquía de herencia de clases:

La profundidad del árbol de herencia en una estructura de clases, se obtiene mediante la cuenta desde el nodo raíz, de la estructura, hasta el último nodo hoja. Este

tipo de cuenta se realiza por niveles, es decir, cuantos niveles jerárquicos hay desde el nodo raíz, hasta el nodo hoja .

El ancho del árbol de herencia en una estructura de clases, se obtiene mediante la cuenta desde el nodo en el extremo derecho, hasta el nodo en el extremo izquierdo, contando los nodos intermedios, como se muestra en la figura 6.3.

Las métricas asociadas a esta variable podrían ser, DIT (Deep in Inheritance Tree) y NOC (Number of Children).

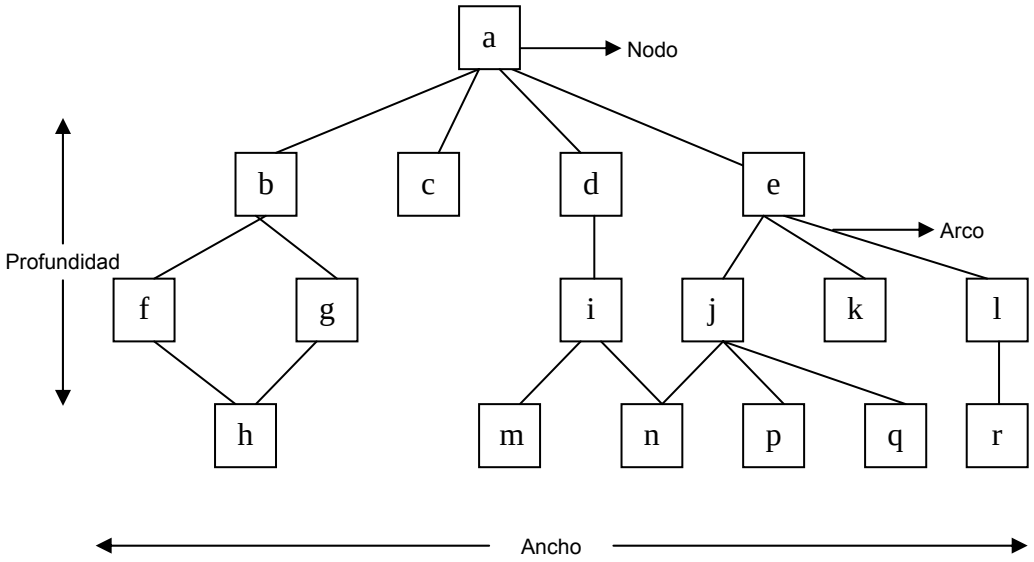

Figura 6.3 *Profundidad y Ancho en una jerarquía de Clases.*

El Grado de Acoplamiento de entre clases:

El acoplamiento de módulo proporciona una indicación de la *"conectividad"* de un módulo con otros módulos, datos globales y entorno exterior. Es la cantidad de relaciones que se establecen entre los módulos de un programa.

Las métricas asociadas a esta variable podrían ser, CBO (Coupling Between Objects) y todas sus redefiniciones, y LCOM (Lack of Coupling between Objects Methods).

Presentado el conjunto de variables que conformaran el estudio, podemos concluir dado el tipo y naturaleza de las mismas que, tales variables son más bien atómicas, en el sentido que no son susceptibles de ser descompuestas en dimensiones. Al tratarse de conceptos independientes entre sí, los consideramos como elementos autónomos.

6.3.4. Objetivos de la investigación

La investigación busca determinar, mediante el análisis y desarrollo estadístico, valores de umbral que sirvan como puntos de referencia respecto de las mediciones realizadas, con el objetivo de proveer al ingeniero de software una herramienta de valoración, que lo asista a la hora de decidir, y dependiendo del aspecto medido, si un componente de software necesita algún tipo de rediseño, o en la indicación de clases con propensión a fallas, o de aquellas que tengan un grado elevado de complejidad cognoscitiva.

En un estudio realizado en [14], se ha presentado evidencia empírica de que no existe relación (valores de umbral) entre el tamaño de un componente de software (Clase, métodos, etc.), y la propensión a errores. En el presente estudio, se analiza si existen valores de umbral, realizado el análisis con métricas de complejidad; Es decir

que se analiza si existe alguna relación ente la complejidad de una clase, y la propensión a fallas.

Los objetivos de la investigación son dos, y se relacionan en función de los resultados obtenidos.

El objetivo principal de la investigación es, determinar si existen umbrales y valores de umbral para la suite de métricas de Chidamber & Kemerer (CK), referidas a la propensión a errores, respecto de la complejidad cognoscitiva de los individuos involucrados en el proceso de desarrollo de software; Es decir, si hay alguna relación entre la propensión a errores y la complejidad de una clase. El método utilizado para llevar a cabo el análisis es la regresión logística (LR). La regresión logística se utiliza para construir modelos cuando la variable dependiente es binaria, como en nuestro caso.

Se dice que un proceso es binomial cuando sólo tiene dos posibles resultados: *"éxito"* y *"fracaso"*, siendo la probabilidad de cada uno de ellos constante en una serie de repeticiones. Un proceso binomial está caracterizado por la probabilidad de éxito, representada por p (es el único parámetro de su función de probabilidad), la probabilidad de fracaso se representa por q y, evidentemente, ambas probabilidades están relacionadas por p+q=1. En ocasiones, se usa el cociente p/q, denominado "odds", y que indica cuánto más probable es el éxito que el fracaso, como parámetro característico de la distribución binomial aunque, evidentemente, ambas representaciones son totalmente equivalentes [25].

Los modelos de regresión logística son modelos de regresión que permiten estudiar si una variable binomial depende, o no, de otra u otras variables (no necesariamente binomiales). Si una variable binomial de parámetro p es independiente de otra variable X, se cumple p= p|X, por consiguiente, un modelo de regresión es una función de p en X que a través del coeficiente de X permite investigar la relación anterior. La regresión logística es útil cuando se trata de predecir el valor de una variable respuesta dicotómica Y, esto es, una respuesta binaria del tipo 0/1, ausente/presente, sano/enfermo, etc., que presumiblemente depende de otras m variables explicativas (Xj, j= 1, ..., m) a través del modelo de probabilidad [25]. El método LR, se explica en detalle en el capitulo 4.

Consecuentemente, seria factible brindar al ingeniero de software una herramienta de comparación y evaluación una vez que haya tomado las medidas pertinentes al dominio del desarrollo que este llevando a cabo. Es decir, toda medida es útil siempre y cuando se la pueda evaluar en comparación con algo. En consecuencia, la determinación de los valores de umbral, estarían aportando el marco de referencia matemático, contra el cual podrían compararse las mediciones tomadas y así, tomar acciones correctivas o no.

6.4 Sumario

La exposición anterior ha presentado las bases teóricas existentes para los efectos de umbral en las métricas OO. También se presentan las métricas CK que se evalúan en el presente estudio, como así también el umbral que se ha derivado en la literatura para cada una de ellas.

CAPITULO 7 CASO DE ESTUDIO

7.1 Método de Investigación

7.1.1 Medidas

Las métricas OO estudiadas descriptas anteriormente, fueron recolectadas utilizando un analizador de código comercial. Con el objetivo de probar el efecto de umbral, se necesita controlar el efecto potencial del tamaño [24]. Para tal fin la métrica utilizada fue SLOC.

7.1.1 Medidas de Fallas

Dentro del contexto de construir modelos cuantitativos referidos a fallas de software, se ha argumentado que es más importante considerar a aquellas fallas que suceden cuando un sistema entra en producción, que aquellas fallas encontradas durante el estadio de prueba [24]. De hecho, se ha argumentado que el objetivo principal del modelado de calidad es identificar la propensión a fallos post-release. En al menos un estudio se encontró que la propensión a fallas pre-release no es una buena medida substituta para la propensión a fallas post-release. La razón que fundamenta esta afirmación es que la propensión a fallas pre-release es una función del esfuerzo de testeo [24].

Como consecuencia, las fallas contadas para los sistemas bajo estudio fueron debido a fallas ocurridas durante su uso actual en producción. A cada clase se las a clasificado como *“con falla”* o *“sin falla”*. Una clase con errores tiene al menos una falla detectada durante su uso en producción. Las distintas fallas que son originadas por la misma falla, son contadas como una.

7.1.2 Fuentes de Datos

El estudio se llevo a cabo sobre dos aplicaciones java, los cuales se describen a continuación.

7.1.2.1 Aplicación java Nº 1

Este es un sistema de mercado electrónico, desarrollado en java. El mismo se encuentra operacional desde hace aproximadamente 10 años. Este sistema se ha puesto en producción en distintos países de Latinoamérica y en distintos entornos de negociación electrónica. En total han trabajado seis desarrolladores en la construcción y mantenimiento del mismo. Esta compuesto de 85 clases que han sido analizado.

Lógicamente y dado que el sistema ha evolucionado en funcionalidad durante los años, este trabajo se enfoca en una versión del mismo, donde se pudieron obtener datos confiables referidos a las fallas.

Los datos sobre fallas fueron recolectados del sistema de administracion de configuración. En este sistema se documenta la razón para cada cambio realizado al código fuente y, por lo tanto, facilita la tarea de identificar que cambios fueron realizados debido a fallas. Se ha puesto el foco en las fallas reportadas por errores en producción.

En total, se encontró que 31 clases tenían una o mas fallas que se atribuían a errores de campo.

7.1.2.2 Aplicación java Nº 2

Este conjunto de datos viene de un framework de mercado electrónico desarrollado para realizar la firma digital de contratos, desarrollado en java. El sistema implementa varios patrones de diseño en los diferentes niveles de la arquitectura. Entre las funcionalidades de este sistema de firma digital se pueden mencionar, generación del par de claves, firma digital de contratos, liquidación de contratos, delegación de contratos, firma de contratos por cta. y orden de terceros, etc.

Este sistema esta siendo utilizado por el mercado líder en negociación de cereales y oleaginosas de Latinoamérica. Un total de 174 clases componen este framework bajo análisis. Un total de 4 programadores han sido involucrados en el proceso de desarrollo y mantenimiento de este conjunto de clases.

Para este producto, se han obtenido los datos sobre fallas del framework en consecuencia al uso actual. Cada error se debió a un único campo de fallas, el cual representa un defecto en el programa que causo la falla. Los errores fueron reportados por los usuarios del sistema. Los desarrolladores del framework documentaron las razones de cada una, en un sistema de control de versiones y, de fue de aquí de donde se extrajo información sobre que clases contenían fallas. Se detectaron un total de 192 fallas. Estas fallas se encontraron en 70 de las clases.

7.1.3 Método de análisis

El método utilizado para llevar a cabo el análisis es la regresión logística (LR) o análisis logit. Este método es utilizado para construir modelos cuando la variable dependiente es binaria, como en este caso de estudio. El enfoque general utilizado es construir un modelo de LR sin umbral, y un modelo LR con umbral, los cuales son expuestos en los apartados 7.1.3.1 y 7.1.3.2 respectivamente, y luego comparar ambos modelos. Para una explicación detallada del método LR ver capitulo 4.

7.1.3.1 Regresión Logística – Modelo sin umbral

La forma general de un modelo LR es la siguiente

$$\prod = \frac{1}{1 + e^{-(\beta_0 + \beta_1 size + \beta_2 M)}} \qquad \textbf{eq. N° 1}$$

Donde π es la probabilidad de que una clase contenga fallas. La variable de tamaño es SLOC, y M es la métrica especifica que se esta evaluando. Los parámetros β se estiman a través de la maximización del logaritmo de probabilidad (incondicional).[4]

7.1.3.2 Modelo con umbral

Un modelo LR con umbral puede definirse como

$$\prod = \frac{1}{1 + e^{-(\beta_0 + \beta_1 size + \beta_2 (M - \tau) I + (M - \tau))}} \qquad \textbf{eq. N° 2}$$

Donde

[4] La regresión logística condicional se utiliza cuando existen concordancias en el diseño del estudio y a cada conjunto concordante se lo trata como un estrato en el análisis.

$$I + (z) \begin{cases} 0 \text{ si } z <= 0 \\ 1 \text{ si } z > 0 \end{cases}$$

Y t es el valor de umbral para la métrica. En este modelo se mantiene al tamaño como variable continua, dado que un estudio realizado en [24], indicó que no hay valor de umbral para el tamaño de las clases.

7.1.3.3 Comparación de modelos

La diferencia entre los modelos con y sin umbral, se ilustran en la figura 7.1

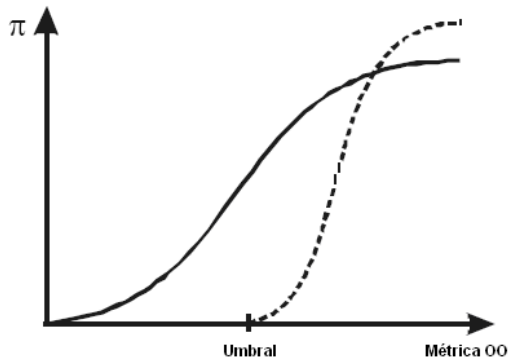

Figura 7.1. Relación entre la métrica OO M y la probabilidad de fallas para los modelos con umbral y sin umbral. En esta relación vi variable se asume que el tamaño es constante.

Para el modelo con umbral, la probabilidad de fallas solo comienza a incrementarse cuando la métrica OO es mayor al valor de umbral t**.** Para estimar el valor de umbral t se debe maximizar el log de probabilidad para el modelo en la ecuación 2. una vez que el umbral se ha estimado, se lo debería evaluar. Esto se lleva a cabo comparando el modelo con umbral y sin umbral. Esta comparación se realiza utilizando un ratio de probabilidad estadística, donde la hipótesis nula que se testea es:

$$H_0: t <= M^* \qquad \textbf{eq N° 3}$$

Donde M* es el valor mas pequeño para la métrica M dentro del conjunto de datos. Si la hipótesis nula no se rechaza, significa que el umbral es igual o esta por debajo del valor mínimo. En este último caso, el modelo con umbral coincide con el modelo sin umbral. En el caso anterior, donde t = M*, el modelo con umbral será muy similar al modelo sin umbral, dado que solo una pequeña porción de las observaciones tendrán el valor mínimo. Por lo tanto, se podría concluir que el umbral no existe. El ratio de probabilidad estadística se calcula de la siguiente forma: 2(ll(H1) – ll(H0)), donde ll(.) es el logaritmo de probabilidad para el modelo dado.

Debería notarse que si el valor estimado de t es igual o cercano a M(n) (el valor más grande dentro del conjunto de datos), significaría que la mayoría de las observaciones en el conjunto de datos tendrían un valor de cero, haciendo a los parámetros estimados para el modelo inestables. En tal caso se concluye que no se encuentra un efecto de umbral para este conjunto de datos. Idealmente, si existe un umbral entonces no estaría demasiado cerca del valor máximo ni del mínimo de la métrica M dentro del conjunto de datos.

7.2 Resultados

7.2.1 Estadísticas descriptivas

Las estadísticas descriptivas para las métricas de complejidad OO y la métrica SLOC se presentan en las tablas 7.1 y 7.2. Estas incluyen un resumen tradicional de de medidas descriptivas de datos tales como, media y desviación estándar. La última columna presenta el número de observaciones que no tienen valor cero. Una de las cosas más notables en los datos presentados en las tablas, es que la herencia tiende a ser baja en ambos sistemas. La métrica NOC para el sistema 2, solo presenta cinco observaciones que no son cero. Por lo tanto esta variable no se considera en lo consecuente, dado que no es posible construir modelos cuando la variación es tan pequeña.

Tabla 7.1. Estadísticas descriptivas para las métricas extraídas del sistema 1.

	Mean	Median	Std. Dev.	IQR	N>0
WMC	16,27	12	17,44	7	85
DIT	0,81	1	0,85	1	49
NOC	0,56	0	1,33	0	17
CBO	13,2	9	9,19	12	85
RFC	35,2	25	35,18	22	85
SLOC	436	280	492	244	85

Tabla 7.2. Estadísticas descriptivas para las métricas extraídas del sistema 2.

	Mean	Median	Std. Dev.	IQR	N>0
WMC	15,27	8	19,75	17	159
DIT	0,45	0	0,52	1	76
NOC	0,034	0	0,212	0	5
CBO	0,667	0	1,169	1	64
RFC	12,87	8	15,91	11	159
SLOC	64,34	41,5	61,33	56,75	174

7.2.2 Evaluación del efecto de umbral

Los resultados para los modelos de umbral de ambos sistemas, y los resultados de la comparación de los modelos con y sin umbral, se muestran en las tablas 7.3 y 7.4.

Todos los modelos con umbral, tienen un numero condicional bajo (por debajo del umbral tradicional de 30), por lo tanto no se considera a la colinearidad como una amenaza. Como se esperaba los valores R^2 son bajos. Contrario de lo que se podría esperar, algunos de los coeficientes de regresión son negativos (DIT para el sistema 1, y DIT, WMC, RFC para el sistema 2). Para el sistema 1, los resultados dejan claro que el modelo con umbral para la métrica CBO, tiene un parámetro estadísticamente significativo. Sin embargo, el modelo con umbral no es diferente del modelo sin umbral (la ultima columna en las tablas muestran el valor-p, para la comparación de los modelos).

De hecho, ninguna de estas métricas presenta diferencias para los modelos con y sin umbral. Las mismas conclusiones pueden obtenerse del sistema 2.

Para la métrica DIT, se identificaron dos umbrales diferentes para ambos sistemas, aunque cuando se testeo la hipótesis nula, esta no pudo rechazarse. El valor mínimo de DIT es cero, es decir, sin herencia. Por lo tanto, este resultado es de alguna manera consistente con la literatura mencionada anteriormente en que el umbral esta en un DIT igual a cero, en lugar de un DIT mayor a cero.

Basado en estos resultados, se concluye que la incorporación de modelos con umbral, no aporta ninguna información nueva, con lo cual los modelos sin umbral, los cuales son más simples, son preferidos ante los modelos con umbral.

Tabla 7.3. Resultados para el modelo con umbral del sistema 1, y la comparación de los modelos

Métrica	G (valor-p)	R^2	η	β_2 (valor-p)	Umbral	Comparación Valor-p
WMC	9,17 (0,0102)	0,085	4,946	0,0946 (0,173)	11	0,72
DIT	12,21 (0,0022)	0,109	2,87	-7,498 (0,0918)	2	0,22
NOC	12,39 (0,002)	0,111	3,08	0,61 (0,0821)	2	0,602
CBO	33,64 (<0,0001)	0,301	4,71	0,1848 (<0,0001)	1	--
RFC	11,98 (0,0025)	0,107	4,5	0,0249 (0,105)	27	0,835

Tabla 7.4. Resultados para el modelo con umbral del sistema 2, y la comparación de los modelos

Métrica	G (valor-p)	R^2	η	β_2 (valor-p)	Umbral	Comparación Valor-p
WMC	15,19 (0,0005)	0,0647	4,521	-0,03114 (0,2042)	31	0,244
DIT	15,28 (0,0005)	0,065	2,87	-6,636 (0,1919)	1	0,27
CBO	14,32 (0,0008)	0,061	2,87	5,5069 (0,3886)	8	0,393
RFC	16,15 (0,0003)	0,0697	4,43	-0,0532 (0,167)	25	0,2288

7.2.3 Discusión

Los resultados presentados anteriormente, indican que no existe un efecto de umbrales para el subconjunto de métricas orientadas a objetos CK. Esto significa que, si hay alguna relación entre estas métricas y la propensión a errores, entonces es una relación continua. Por lo tanto, conforme crece la complejidad cognitiva de las clases, la propensión a error se desplaza en forma continua, sin presentar discontinuidades en la función que puedan hacer que la curva presente algún tipo de salto a partir del valor de umbral. Esta verificación también puede llevarse a cabo, calculando los límites acercándose por derecha y por izquierda al punto del valor de umbral. Esto también significa que la teoría cognitiva que se ha postulado en [15] no recibe ningún tipo de soporte del presente estudio, al menos para el software OO. No hay ningún pasaje en las métricas CK estudiadas donde la probabilidad a fallas cambie de ir en aumento progresivo continuo, a incrementarse fuertemente, como se muestra en la fig.7.1.

Con esto, no se desea significar que los umbrales OO existentes, derivados del conocimiento experimental, no son de utilidad práctica a la luz de estos resultados. Aun si existiera una relación continúa (sin umbral), entre estas métricas y la propensión a fallas como se propone, si se trazara una línea en el valor más alto de la métrica y se tomara a este valor como umbral, las clases que estén por debajo de este valor de umbral, continuaran siendo propensas a error. Esta situación se ilustra en el panel izquierdo de la figura 7.2.

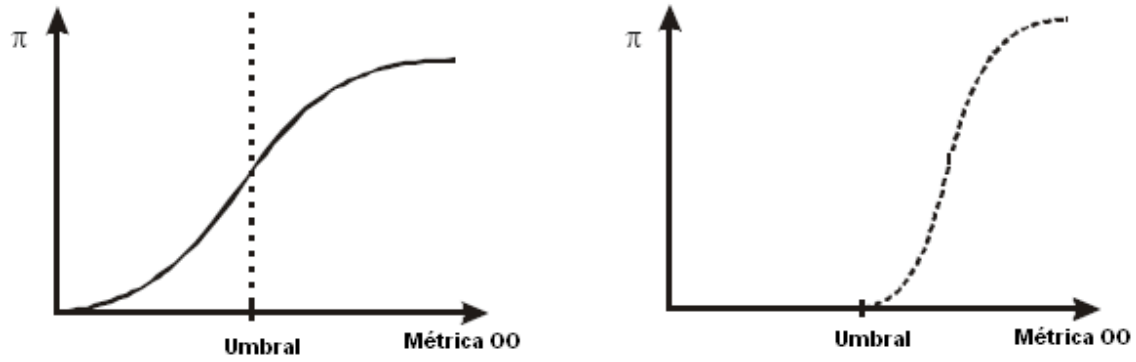

Figura 7.2. Diferentes tipos de umbral

Por lo tanto, con el propósito de identificar las clases más propensas a error, estos umbrales probablemente funcionen. Pero debería notarse que las clases por debajo del umbral, aun pueden contener una alta propensión a fallas, simplemente no el más alto.

Habiendo identificado un umbral, las clases con valores por debajo del mismo representan una región segura donde los diseñadores deliberadamente pueden restringir sus clases. Dentro de esta región se puede tener alguna certeza de que las clases, manteniéndose todo lo demás igual, poseen una propensión a fallas mínima. Esta situación se ilustra en el panel derecho de la figura 7.2.

CAPITULO 8 CONCLUSIONES

8.1 Conclusiones

En [23], Hatton postula una teórica cognitiva en la cual sugiere que existe un efecto de umbral para varias métricas de software. Esta teoría también se ha extendido al software OO. Independientemente de esta teoría, otros investigadores han derivado sus propios umbrales de métricas OO, basados en sus experiencias.

El propósito principal del estudio presentado en este documento es testear esta teoría empíricamente. El estudio se llevo a cabo en dos sistemas de mercado electrónico, desarrollados en java, utilizando un subconjunto de métricas CK [6] [16].

La variable dependiente en el análisis fue la incidencia a fallos, la cual lleva a campos de fallas (propensión a fallos). Los resultados presentados anteriormente, indican que no existe el efecto de umbrales para el subconjunto de métricas orientadas a objetos CK. Esto significa que, si hay alguna relación entre estas métricas y la propensión a errores, entonces es una relación continua. Esto también significa que la teoría cognitiva que se ha postulado no recibe ningún tipo de soporte del presente estudio, al menos para el software OO. No hay ningún pasaje en las métricas CK estudiadas donde la probabilidad a fallas cambie de ir en aumento progresivo continuo, a incrementarse fuertemente. Estos resultados son consistentes en ambos sistemas y para todas las métricas utilizadas.

Sin bien no se garantiza que no existe un efecto de umbral, la evidencia de los resultados demuestra lo contrario.

Las implicancias de estos resultados son:

- Aquellos usuarios que derivan o utilizan umbrales de las métricas CK, deberían notar que las clases por debajo del valor de umbral, aun son probables que posean una alta propensión a fallas, aunque quizás no la propensión a fallas mas alta.

- Los investigadores que validen las métricas OO, deberían continuar modelando la relación entre las métricas OO, al menos el conjunto de métricas CK, y la propensión a fallas utilizando asunciones de continuidad en lugar de asunciones de umbrales.

- Con esto, no se desea significar que los umbrales OO existentes, derivados del conocimiento experimental, no son de utilidad práctica a la luz de estos resultados. Aun si existiera una relación continúa (sin umbral), entre estas métricas y la propensión a fallas como se propone, si se trazara una línea en el valor mas alto de la métrica y se tomara a este valor como umbral, las clases que estén por debajo de este valor de umbral, continuaran siendo propensas a error.

- Para terminar, es fundamental reconocer que la formulación de teorías es importante para una disciplina. Las teorías explican los fenómenos que observamos (proveen de un mecanismo). El conocimiento del mecanismo puede potencialmente guiar a avances de campo. Por lo tanto necesitamos constantemente proponer teorías, explicaciones y validarlas empíricamente.

8.2 Contribución del estudio

Es claro dada la evidencia presentada que no existe un efecto de umbral entre la complejidad de una clase y la propensión a errores. Por lo tanto la teoría que declara que la propensión a errores en una clase permanece estable hasta un cierto nivel de complejidad y una vez que este nivel de complejidad es excedido la propensión a fallas se incrementa, debido a limitaciones en la memoria a corto plazo, queda sin soporte. La única evidencia que podemos presentar es que existe una relación continua entre la complejidad cognitiva de una clase y la propensión a errores. Esto quiere decir que, conforme la complejidad cognitiva de la clase aumente, también lo hará su propensión a fallas.

8.3 Trabajo Futuro

Los resultados presentados, no deberían implicar que la complejidad de una clase, sea la única variable que se puede utilizar para predecir propensión a errores para el software OO. Por el contrario, mientras la complejidad de una clase pareciera ser una variable importante, otros factores indudablemente tendrán su influencia. Por lo tanto, y con el propósito de construir modelos comprensivos que predigan la propensión a fallas, otras variables deberían ser analizadas.

REFERENCIAS

1. Balasubramanian NV. Object oriented metrics. *Proceedings 3rd Asia–Pacific Software Engineering Conference(APSEC'96)*. IEEE Computer Society Press: Los Alamitos CA, 1996; 30–34.

2. Tahvildari L, Singh A. Categorization of object-oriented software metrics. *Proceedings IEEE Canadian Conference on Electrical and Computer Engineering*. IEEE Computer Society Press: Los Alamitos CA, 2000; 235–239.

3. Tegarden DP, Sheetz SD, Monarchi DE. A software complexity model of object-oriented systems. *Decision Support Systems* 1995; **13**(3–4):241–262.

4. McCabe TJ, Dreyer LA, Dunn AJ, Watson AH. *Testing an Object-Oriented Application*. Quality Insurance Institute: Orlando FL, 1994; 21–27.

5. Harrison R, Counsell SJ, Nithi RV. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering* 1998; **24**(6):491–496.

6. Chidamber SR, Kemerer CF. Towards a metric suite for object-oriented design. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*. ACM Press: New York NY, 1991; 197–211.

7. Chidamber SR, Kemerer CF. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering* 1994;**20**(6):476–493.

8. Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. *Technical Report*, University of Maryland, Department of Computer Science, College Park MD, 1995; 1–24.11. Tang MH, Kao MH, Chen MH. An empirical study on object-oriented metrics. *Proceedings 23rd Annual International Computer Software and Application Conference*. IEEE Computer Society.

9. Fenton NE, Neil M. Software metrics: Successes, failures and new directions. *The Journal of Systems and Software* 1999; 47(2–3):149–157.

10. Harrison R, Counsell SJ, Nithi RV. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering* 1998; 24(6):491–496.

11. Daly J, Brooks A, Miller J, Roper M, Wood M. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1996; **1**(2):109–132.

12. Chaumun, M Ajmal; Kabaili, Hind; Keller, Rudolf K; Lustman, Francois; Design properties and object-oriented software changeability. Département IRO Université de Montreal C.P. 6128, succursale Centre-ville Montréal, Québec H3C 3J7, Canada - 2000

13. Benlarbi, Saida; El Emam, Khaled; Goel, Nishith; Rai, Shesh; Thresholds for object-oriented measures; Alcatel CID, Can; PROC INT SYMP SOFTWARE RELIAB ENG ISSRE. pp. 24-37. 2000.

14. El Emam, Khaled; Benlarbi, Saida; Goel, Nishith; Melo, Walcelio; Lounis, Hakim; Rai, Shesh N;
The optimal class size for object-oriented software; National Research Council of Canada Institute for Information Technology Building M-50, Ottawa, Ont., K1A OR6, Canada; IEEE Transactions on Software Engineering. Vol. 28, no. 5, pp. 494-509. May 2002

15. L. Hatton: "Re-examining the Fault Density - Component Size Connection". In *IEEE Software*,
pages 89-97, 1997.

16. Pressman R, "Ingeniería del Software, un enfoque práctico"; McGraw-Hill; Cuarta Edición.1998

17. Berard E., “Metrics for Object-Oriented Software Engineering” comp-software-eng; 1995.

18. Li W. Another metric suite for object-oriented programming. The Journal of Systems and Software 1998; 44(2):155–162.

19. Sheldon, F T; Jerath, K; Chung, H, Metrics for maintainability of class inheritance hierarchies; 73 Distribution, Maintenance, and Enhancement; JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE J. Softw. Maint. Evol.: Res. Pract. 2002; 14:147–160 (DOI: 10.1002/smr.249)

20. Ma, C-S; Chang, C K; Cleland-Huang, J; Measuring the intensity of object coupling in C++ programs; PROC IEEE COMPUT SOC INT COMPUT SOFTWARE APPL; CONF. pp. 538-543. 2001.

21. Grady Booch, Análisis y Diseño Orientado a Objetos. 2° Edición Addison Wesley Longman – 1996. ISBN: 968-444-352-8

22. The role of object-oriented metrics - Meyer - (November 1998).
http://archive.eiffel.com/doc/manuals/page.html

23. L. Hatton: "Does OO Sync with How We Think ?" In IEEE Software, pages 46-54, May/June 1998.

24. K. El Emam, S. Benlarbi, N. Goel, and S. Rai: "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics". To appear in IEEE Transactions on Software Engineering, 2000.

25. SPSS - Estadistica avanzada – AMA 2003

26. Hair, Anderson, Tathan, Black, “Analisis Multivarante” 5ta. Edicion - Prentice Hall, 1999

27. L. Rosenberg, R. Stapko, and A. Gallo: "Object-Oriented Metrics for Reliability". Presentation at *IEEE International Symposium on Software Metrics*, 1999.

28. F. I. M. Craik, R. S. Lockhart “Levels of Processing: A Framework for Memory Research”Journal of Verbal Learning and Verbal Behavior, 11, 671-684. (1972).

29. C. Sabino – El proceso de investigación – Cap. 5 - http://paginas.ufm.edu/sabino/PIcap-5.htm

30. Umbrales para Métricas Orientadas A Objetos - Pablo Negro, Roxana Giandini, Universidad Abierta Interamericana, LIFIA, UNLP, Argentina – ASSE (36 JAIIO07) Agosto 2007. Mar del Plata. Argentina