Métricas, medidas y mediciones del software

Medida

Proporciona un indicio cuantitativo de la extensión, cantidad, capacidad o tamaño de algún atributo de un producto o proceso.

Ej: cantidad de errores en un componente de sw

Medición

Es el acto de determinar una medida.

La medición ocurre como resultado de la recolección de uno o más puntos de datos.

Ej: recolectar medidas del número de errores de un componente y una prueba de unidad.

Métrica

Es una medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado.

Ej: el número promedio de errores que se encuentran por revisión o el número promedio de errores que se encuentran por unidad de prueba.

- Un ingeniero de software recolecta medidas y desarrolla métricas de modo que se obtengan indicadores.
- Un ***indicador*** es una métrica o combinación de métricas que proporcionan comprensión acerca del proceso de software, el proyecto de software o el producto en sí.
- Permite ajustar el proceso, el proyecto o el producto para hacer mejor las cosas.

¿Por qué es importante medir?

- Siempre habrá un elemento cualitativo en la creación del software. El problema es que la valoración cualitativa tal vez no sea suficiente.
- Se necesitan criterios objetivos que ayuden a guiar el diseño de datos, arquitectura, interfaces y componentes.
- Cuando se prueban, es necesaria la guía cuantitativa que ayuda en la selección de los casos de prueba y de sus objetivos. **Las métricas de producto proporcionan una base desde donde el análisis, el diseño, la codificación y las pruebas pueden realizarse de manera más objetiva y valorarse de modo más cuantitativo**.

¿Cuáles son los pasos de un proceso de medición efectivo?

1. **Formulación**. La derivación de medidas y métricas de software apropiadas para la representación del software que se está construyendo.
2. **Recolección**. Mecanismo que se usa para acumular datos requeridos para derivar las métricas formuladas.
3. **Análisis**. El cálculo de métricas y la aplicación de herramientas matemáticas.
4. **Interpretación**. Evaluación de las métricas resultantes para comprender la calidad de la representación.
5. **Retroalimentación**. Recomendaciones derivadas de la interpretación de las métricas del producto, transmitidas al equipo de software.

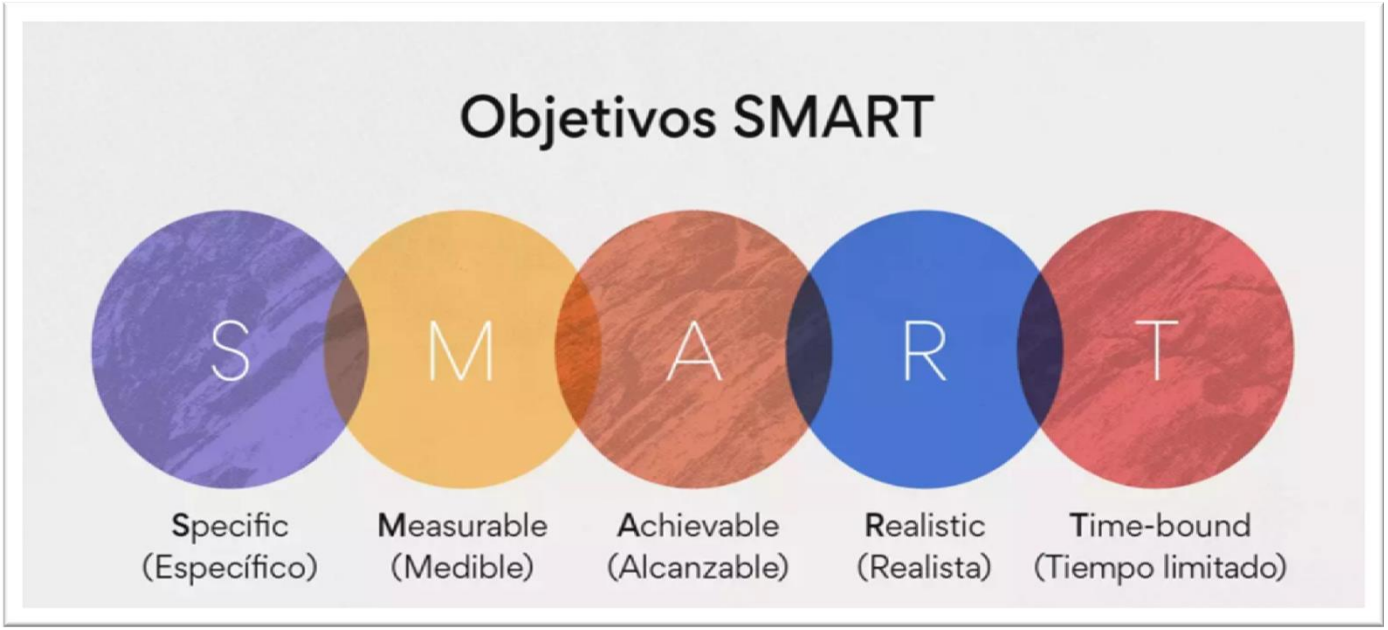
Paradigma Meta/Pregunta/Métrica (MPM)

- Es una técnica para identificar métricas significativas para cualquier parte del proceso de software.

1. Establecer una ***meta*** de medición explicita que sea especifica para la actividad del proceso o para la característica del producto que se quiera valorar,
2. Definir un conjunto de ***preguntas*** que deban responderse con la finalidad de lograr la meta y
3. identificar ***métricas*** bien formuladas que ayuden a responder dichas preguntas.

- **Meta**:

Aumentar el tráfico del sitio web a 25.000 visitas en el segundo trimestre.



Preguntas:

- ¿Cuántas visitas se obtuvo en el ultimo mes?
- ¿Se realizan actualizaciones constantes de contenido en el sitio?
- ¿Se realizan campañas de promoción en diferentes plataformas?

Ejemplo de métricas:

- Páginas vistas únicas del sitio web (número de visitas) en un mes
- Cantidad promedio de actualización de contenidos durante un mes
- Cantidad de campañas realizadas en el primer trimestre
- Rango de edades del publico objetivo realizado en campañas
- Cantidad de regiones que alcanzan las campañas
- Tiempo promedio de duración de las campañas

Atributos de las métricas de software efectivas

1. **Simple y calculable**. Debe ser relativamente fácil aprender cómo derivar la métrica y su cálculo no debe demandar esfuerzo o tiempo excesivo.
2. **Empírica e intuitivamente convincente**. Debe satisfacer las nociones intuitivas del ingeniero acerca del atributo de producto que se elabora (por ejemplo, una métrica que mide la cohesión del módulo debe aumentar en valor conforme aumenta el nivel de cohesión).
3. **Congruente y objetiva**. Siempre debe producir resultados que no tengan ambigüedades. Una tercera parte independiente debe poder derivar el mismo valor de métrica usando la misma información acerca del software.

Atributos de las métricas de software efectivas

4. **Constante en su uso de unidades y dimensiones**. El cálculo matemático de la métrica debe usar medidas que no conduzcan a combinaciones extrañas de unidades. Por ejemplo, multiplicar personas en los equipos de proyecto por variables de lenguaje de programación en el programa da como resultado una mezcla sospechosa de unidades que no son intuitivamente convincentes.
5. **Independiente del lenguaje de programación**. Debe basarse en el modelo de requerimientos, el modelo de diseño o la estructura del programa en sí. No debe depender de la sintaxis o de la semántica del lenguaje de programación.
6. **Un mecanismo efectivo para retroalimentación de alta calidad**. Debe proporcionar información que pueda conducir a un producto final de mayor calidad.

Métricas para el modelo de requerimientos

Examinan el modelo de requerimientos con la intención de predecir el “tamaño” del sistema resultante. En ocasiones, el tamaño es un indicador de la complejidad del diseño y casi siempre es un indicador creciente de codificación, integración y esfuerzo de pruebas.

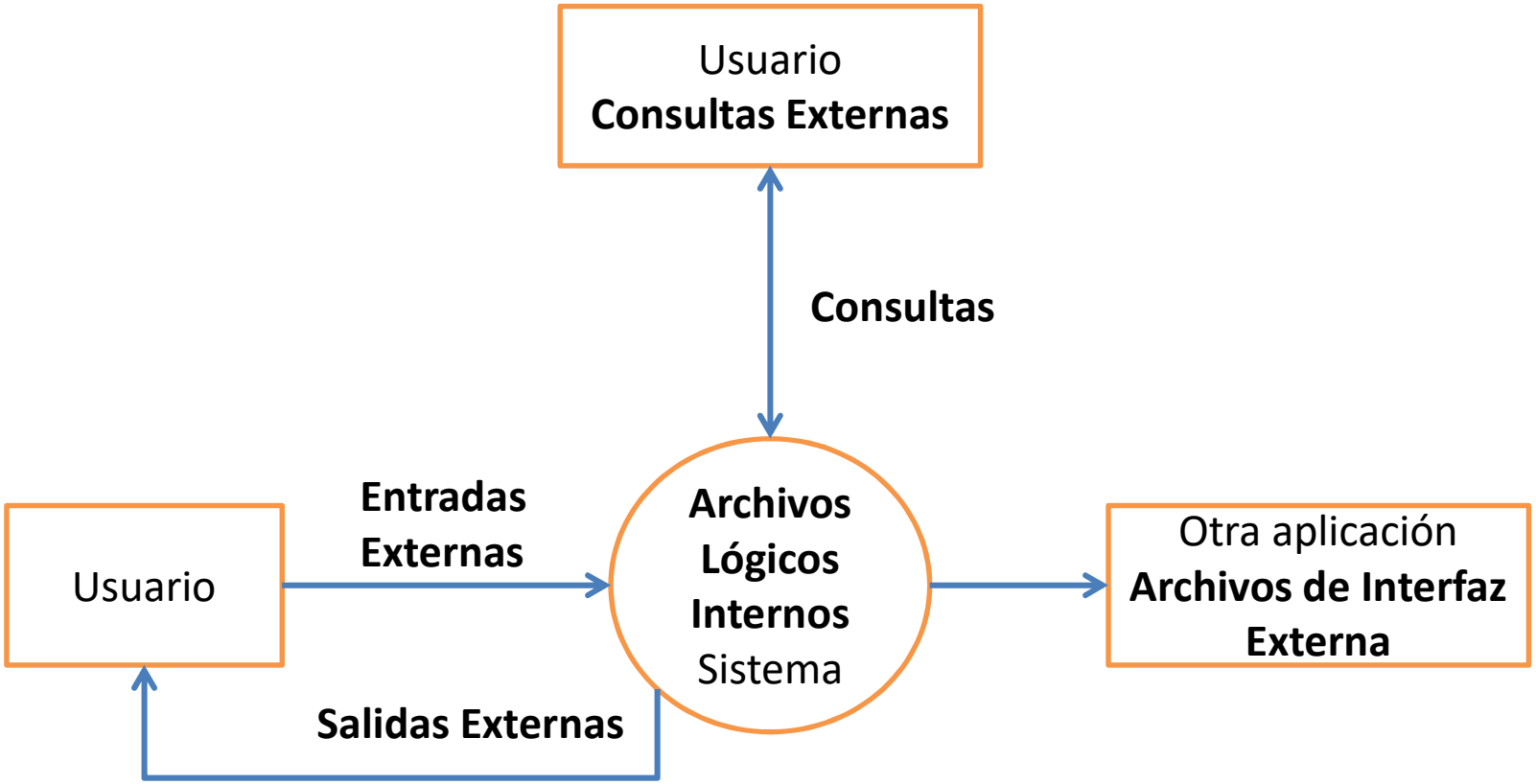
Métrica basada en funciones

- La ***métrica de punto de función*** **(PF)** puede usarse de manera efectiva como medio para medir la funcionalidad que entra a un sistema.
- La métrica PF puede usarse para:
  - Estimar el costo o esfuerzo requerido para diseñar, codificar y probar el software.
  - Predecir el número de errores que se encontrarán durante las pruebas, y
  - Prever el número de componentes y/o de líneas de código proyectadas en el sistema implementado.

- Los puntos de función se derivan usando una relación empírica basada en medidas contables del dominio de información del software y en valoraciones cualitativas de la complejidad del software.

Los valores de dominio de información son:

- **Número de entradas externas (EE).** Cada entrada externa se origina de un usuario o se transmite desde otra aplicación, y proporciona distintos datos orientados a aplicación o información de control. Con frecuencia, las entradas se usan para actualizar archivos lógicos internos (ALI). Las entradas deben distinguirse de las consultas, que se cuentan por separado. Ej: ABM, pantallas en donde el usuario ingresa datos.
- **Número de salidas externas (SE).** Cada salida externa son **datos derivados** dentro de la aplicación que ofrecen información al usuario. En este contexto, salida externa se refiere a reportes, envío de notificación, mensajes de error, etc.
- **Número de consultas externas (CE).** Una consulta externa se define como una entrada en línea que da como resultado la generación de alguna respuesta de software inmediata en la forma de una salida en línea (con frecuencia recuperada de un ALI).
- **Número de archivos lógicos internos (ALI**). Cada archivo lógico interno es un agrupamiento lógico de datos que reside dentro de la frontera de la aplicación y se mantiene mediante entradas externas. Ej: tabla de la base de datos.
- **Número de archivos de interfaz externos (AIE).** Cada archivo de interfaz externo es un agrupamiento lógico de datos que reside fuera de la aplicación, pero que proporciona información que puede usar la aplicación. Ej: Api externas.


Usuario
Consultas Externas
Consultas
Entradas
Externas
Usuario
Archivos
Lógicos
Internos
Sistema
Otra aplicación
Archivos de Interfaz
Externa
Salidas Externas

- Una vez recolectados dichos datos, se completa la siguiente tabla y un valor de complejidad se asocia con cada conteo. Por ejemplo: «Función de interacción con el usuario de un sw».

Valor de dominio de información	Conteo		Factor ponderado Simple	Promedio	Complejo		
Entradas externas (EE)		×	3	4	6	=	
Salidas externas (SE)		×	4	5	7	=	
Consultas externas (CE)		×	3	4	6	=	
Archivos lógicos internos (ALI)		×	7	10	15	=	
Archivos de interfaz externos (AIE)		×	5	7	10	=	
Conteo total →							

- Las organizaciones que usan métodos de punto de función desarrollan criterios para determinar si una entrada particular es simple, promedio o compleja.
- No obstante, la determinación de complejidad es un tanto subjetiva.

Para calcular puntos de función (PF), se usa la siguiente relación:

$$PF = \text{conteo total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

- Los valores constantes en la ecuación y los factores ponderados que se aplican a los conteos de dominio de información se determinan de manera empírica.
- Factor de complejidad técnica es todo lo que se encuentra entre **[]**.
- F_i (i = 1 a 14): son factores de ajuste de valor (FAV) con base en respuestas a las siguientes preguntas, cada una de estas preguntas se responde usando una escala que varía de 0 (no importante o no aplicable) a 5 (absolutamente esencial).

Factor de Ajuste	Puntaje
Comunicación de Datos	4
Procesamiento Distribuido	4
Objetivos de Rendimiento	1
Configuración del equipamiento	1
Tasa de transacciones	3
Entrada de Datos en Línea	5
Interfase con el usuario	2
Actualizaciones en Línea	3
Procesamiento Complejo	1
Reusabilidad del Código	1
Facilidad de Implementación	
Facilidad de Operación	1
Instalaciones Múltiples	2
Facilidad de Cambios	4
Factor de Ajuste	32

1. ¿El sistema requiere respaldo y recuperación confiables?	• Comunicación de datos
2. ¿Se requieren comunicaciones de datos especializadas para transferir información hacia o desde la aplicación?	
3. ¿Existen funciones de procesamiento distribuidas?	• Procesamiento Distribuido
4. ¿El desempeño es crucial?	• Objetivos de rendimiento
5. ¿El sistema correrá en un entorno operativo existente enormemente utilizado?	• Configuración del equipo
6. ¿El sistema requiere entrada de datos en línea?	• Entrada de datos en línea.
7. ¿La entrada de datos en línea requiere que la transacción de entrada se construya sobre múltiples pantallas u operaciones?	• Tasa de transacciones. Interfaz con el usuario.

8. ¿Los ALI se actualizan en línea?	• Actualizaciones en línea.
9. ¿Las entradas, salidas, archivos o consultas son complejos?	• Procesamiento complejo
10. ¿El procesamiento interno es complejo?	
11. ¿El código se diseña para ser reutilizable?	• Reusabilidad del código.
12. ¿La conversión y la instalación se incluyen en el diseño?	• Facilidad de implementación
13. ¿El sistema se diseña para instalaciones múltiples en diferentes organizaciones?	• Instalaciones múltiples
14. ¿La aplicación se diseña para facilitar el cambio y su uso por parte del usuario?	• Facilidad de cambios y de operación.

Valor dominio de información	Conteo		Factor ponderado Simple	Promedio	Complejo		
Entradas externas (EE)	3	×	③	4	6	=	9
Salidas externas (SE)	2	×	④	5	7	=	8
Consultas externas (CE)	2	×	③	4	6	=	6
Archivos lógicos internos (ALI)	1	×	⑦	10	15	=	7
Archivos de interfaz externos (AIE)	4	×	⑤	7	10	=	20
Conteo total							50

Supongamos que Fi es 46 (un producto moderadamente complejo).

$$PF = 50 \times [0.65 + (0.01 \times 46)] = 56$$

El equipo del proyecto puede estimar el tamaño global implementado de la «función de interacción del usuario».

Nota: los puntos de función pueden calcularse a partir de diagrama de flujo de datos, de clases UML y diagramas de secuencia.

Tabla IFPUG (International Function Point Users Group)

Lenguaje	HS promedio por PF	Líneas de código por PF
Ensamblador	25	300
2ª y 3ª Generación (COBOL, FORTRAN, PASCAL y MODULA)	15	100
4º Generación (Java, C#, Python, PHP, JS, etc.)	8	20

Ejercicio 1

- Estimar mediante la métrica de punto de función:

1. El esfuerzo del proyecto informático (H/P). Considere 8 horas promedio por punto de función.
2. Duración en meses del proyecto. Considere 8 horas diarias, 20 días al mes.
3. Costo del proyecto informático. Considere:
    1. Sueldo por desarrollador= $400000
    2. Otros costos = $1.000.000

Se consideran que las siguientes funciones son de complejidad media y que el factor de ajuste es de 32:

1. Registrar clientes
2. Registrar ventas
3. Buscar ventas por fecha
4. Actualizar datos del cliente
5. Eliminar cliente
6. Enviar notificación de promociones a clientes
7. 1 reporte de los clientes registrados por rango de fechas
8. 1 reporte de ventas
9. 4 tablas en BD (Cliente, Venta, DetVta y Producto)

Solución

- Horas/hombre = 72,75 * 8 = 582 horas/persona
- Duración= 582/8 hs diarias de trabajo= 72,75 días
  - Si considero 20 días al mes = 72,75/20 = **3,64 meses/persona**
  - Aproximado 3,64 meses por desarrollador.
  - ***Si tengo 2 desarrolladores = 1,82 meses/desarrollador***
- Presupuesto
  - Sueldo por desarrollador= $400000
  - Otros costos = $1.000.000
  - CT= (nº desarrolladores * sueldo * tiempo) + otros costos = (2* $400000 *1,82) + $1.000.000 = $2.455.000

Ejercicio 2

- Un sistema tiene 15 entradas externas, 24 salidas externas, presenta 0 diferentes consultas externas, gestiona 4 archivos lógicos internos y tiene interfaz con 6 diferentes sistemas legados (6 AIE). Todos estos datos son de complejidad promedio (tomar tabla de ponderación) y el sistema global es relativamente simple (Fi = 25). Calcule PF para el sistema.

Métricas para calidad de la especificación

- **Se propone una lista de características que pueden usarse para valorar la calidad del modelo de requerimientos y la correspondiente especificación de requerimientos**:
- Especificidad (falta de ambigüedad), completitud, corrección, comprensibilidad, verificabilidad, consistencia interna y externa, factibilidad, concisión, rastreabilidad, modificable, precisión y reusabilidad.

CARACTERÍSTICAS DESEABLES DE UNA ERS

- **No ambigua**: La ERS es no ambigua si todo requisito posee una sola interpretación.
- **Completa**: Una ERS es completa si todo lo que se supone que el software debe hacer está incluido en la ERS. Por completitud, deberían describirse todas las posibles respuestas a todas las posibles entradas y en todas las situaciones posibles. ***Contemplar todos los escenarios posibles***.
- **Correcta**: Todo requisito de la ERS contribuye a satisfacer una necesidad real.
- **Comprensible**: Todo tipo de lectores (clientes, usuarios, desarrolladores, equipo de pruebas, gestores, etc.) entienden la ERS.
- **Verificable**: Si para cada requisito expresado en la ERS existe un procedimiento de prueba finito y no costoso para demostrar que el futuro sistema lo satisface.
- **Internamente Consistente**: No existen subconjuntos de requisitos contradictorios.
- **Externamente Consistente**: Ninguno de los requisitos está en contradicción con lo expresado en documentos de nivel superior. Por ejemplo, en un sistema (hardware+software), los requisitos del software no pueden contradecir los requisitos del sistema.

CARACTERÍSTICAS DESEABLES DE UNA ERS

- **Factible**: Si, dados los actuales recursos, la ERS se puede implementar.
- **Concisa**: La ERS debe ser lo más breve posible, sin que esto afecte al resto de atributos de calidad.
- **Independiente del diseño:** Existen más de un diseño e implementación que realizan la ERS. Para ello la ERS debería limitarse a describir el comportamiento externo del sistema.
- **Trazable**: Cada requisito se puede referenciar de forma unívoca. Es fundamental para precisar qué requisitos son implementados por qué componente del diseño, lo cual es imprescindible a la hora de realizar las pruebas de dicho componente.
- **Modificable**: Los cambios son fáciles de introducir. Determinar un nivel de cambio a cada requerimiento (bajo, medio, alto).
- **Precisa**: Una ERS es precisa si hace uso de valores numéricos para precisar las características del sistema. La precisión es aplicable, ante todo, a los requisitos no funcionales.
- **Reutilizable**: Si ciertas secciones de la ERS se pueden reutilizar.

- Cada una puede representarse usando una o más métricas. Por ejemplo, se supone que existen *nr* requerimientos en una especificación, tales que *nr = nf + nnf*
- donde *nf* es el número de requerimientos funcionales y *nnf* es el número de requerimientos no funcionales (por ejemplo, rendimiento).

Especificidad (falta de ambigüedad) de los requerimientos:

- ***Q1 = nui/nr***
- donde *nui* es el número de requerimientos para los cuales todos los revisores tienen interpretaciones idénticas.
- Mientras más cercano a 1 esté el valor de Q, menor será la ambigüedad de la especificación.

La completitud de los requerimientos funcionales

- Puede determinarse al calcular la razón:
- ***Q2= nu / (ni x ns)***
- donde *nu* es el número de requerimientos funcionales únicos, *ni* es el número de entradas (estímulos) definidas o implicadas por la especificación y *ns* es el número de estados especificados.
- ***La razón Q2 mide el porcentaje de funciones necesarias que se especificaron para un sistema***.
- Sin embargo, no aborda requerimientos no funcionales.
- Para ello se usa *Q*3 = *nc /(nc* + *nnv)*

donde *nc* es el numero de requerimientos que se validaron como correctos y *nnv* es el numero de requerimientos que no se han validado.

Métricas para especificar requerimientos no funcionales:

Rapidez	Transacciones procesadas por segundo. Tiempo de respuesta al usuario y a eventos Tiempo de actualización de la pantalla
Tamaño	K Bytes Número de chips de RAM
Facilidad de uso	Tiempo de formación Número de cuadros de ayuda
Fiabilidad	Tiempo medio entre fallos Probabilidad de no disponibilidad Tasa de ocurrencia de fallos Disponibilidad
Robustez	Tiempo de reinicio después de fallos Porcentaje de eventos que provocan fallos Probabilidad de corrupción de los datos después de fallos
Portabilidad	Porcentaje de declaraciones dependientes del objetivo Número de sistemas objetivo

MÉTRICAS PARA EL MODELO DE DISEÑO

"La arquitectura de un sistema
es un marco general que describe
su forma y estructura: sus
componentes y la manera en la
que ajustan entre sí".
Jerrold Grochow

MÉTRICAS PARA EL MODELO DE DISEÑO

- Del diseño arquitectónico
- Para el diseño OO
  - orientadas a clase: suite CK
  - orientadas a clase: suite MOOD
  - O.O propuestas por Lorenz y Kidd
- De diseño en el nivel de componente
- Orientadas a operación
- De diseño de IU

Métricas del diseño arquitectónico

- Se enfocan en características de la arquitectura del programa con énfasis en la estructura arquitectónica y en la efectividad de los módulos o componentes dentro de la arquitectura.
- Dichas métricas son “caja negra” en tanto no requieren conocimiento alguno del funcionamiento interior de un componente de software particular.

Se pueden definir 3 medidas de complejidad del diseño de software: complejidad estructural, complejidad de datos y complejidad del sistema.

Por ejemplo para arquitecturas jerárquicas, la ***complejidad estructural*** *de un módulo i* se define de la forma:

- $\mathbf{S(i) = f^2_{out}(i)}$
- donde $f_{out}(i)$ es el fan-out del módulo i.
- *Fan-out: es el número de módulos que invoca directamente el modulo i.*

Por ejemplo el módulo clientes invoca a los módulos ventas y suscripción.

Entonces la complejidad del módulo clientes es:

$S(clientes) = 2^2 = 4$

La *complejidad de datos* ofrece un indicio de la complejidad que hay en la interfaz interna para un modulo *i* y se define como:

- **$D(i) = v(i) / f_{out}(i) + 1$**
- donde *v*(*i*) es el numero de variables de entrada y salida que pasan hacia y desde el modulo *i*.

Por ejemplo, v(i) = 21

D(clientes) = 21 / (2+1) = 21/3 = 7

La **complejidad del sistema** se define como la suma de las complejidades estructural y de datos y se especifica como:

- **C(i) = S(i) + D(i)**

Para nuestro simple ejemplo:

C(clientes)= 4 + 21/3 = 11

Conforme aumenta el valor de cada una de estas complejidades, la complejidad arquitectónica global del sistema también aumenta.

Esto conduce a una mayor probabilidad de que también aumenten el esfuerzo de integración y el de pruebas.

Del diseño orientado a objetos Tres grupos de métricas orientadas a objetos

- ✓ Métricas orientadas a la clase (CK).
- ✓ Métricas orientadas a la clase (MOOD) .
- ✓ Métricas de Lorenz y Kidd.

Métricas orientadas a clase: la suite de métricas CK

- Chidamber y Kemerer (CK) propusieron uno de los conjuntos de métricas de software OO de mayor referencia.
- Los autores proponen seis métricas de diseño basadas en clase para sistemas OO.
- *En ocasiones el simple valor que se obtiene para una métrica o la comparación de una misma métrica para dos sistemas distintos permite hacernos una idea de su nivel de calidad.*

1. **Métodos ponderados por clase (MPC)**
    1. Clase con n métodos, de complejidad $c1$, … , cn . La complejidad se mide con alguna métrica elegida.
    2. $MPC = \sum_{i=1}^{n} Ci$

2. **Profundidad del árbol de herencia (PAH)**
    1. Conforme crece la PAH, es probable que las clases de nivel inferior hereden muchos métodos. Esto conduce a potenciales dificultades cuando se intenta predecir el comportamiento de una clase y conduce a mayor complejidad de diseño.
    2. Grandes valores de PAH implican que muchos métodos pueden reutilizarse.

3. **Número de hijos (NDH)**
    1. Conforme crece el numero de hijos, el reuso aumenta y la cantidad de pruebas (requeridas para ejercitar cada hijo en su contexto operativo) también aumentara.

4. **Acoplamiento entre clases de objetos (ACO)**
    1. Valores altos complican las modificaciones y las pruebas . En general, los valores de ACO para cada clase deben mantenerse tan bajos como sea razonable.

5. Respuesta para una clase (RPC): es un conjunto de métodos que potencialmente pueden ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. Conforme aumenta la RPC, también los esfuerzos de las pruebas y la complejidad del diseño global de la clase.

6. Falta de cohesión en métodos (FCOM): es el numero de métodos que acceden a uno o mas de los mismos atributos. Si hay 6 métodos y 4 de ellos acceden a atributos comunes, FCOM = 4. Si ninguno accede FCOM = 0. *Es deseable mantener baja la FCOM.*

Métricas orientadas a clase: La suite de métricas MOOD

- **Factor de herencia de método (FHM)**
  - El grado en el que la arquitectura de clase de un sistema OO utiliza la herencia tanto para métodos como para atributos**.**
  - Ofrece un indicio del impacto de la herencia sobre el software OO.
- **Factor de acoplamiento (FA)**
  - Conforme el valor de FA aumenta, la complejidad del software OO tambien aumentara

Métricas OO propuestas por Lorenz y Kidd

- Dividen las métricas basadas en clase en cuatro amplias categorías; cada una tiene una relación en el diseño en el nivel de componentes:
  - Tamaño, herencia, internos y externos

Métricas OO propuestas por Lorenz y Kidd

Tamaño de una clase	Herencia	Métricas internas de clase	Métricas externas de clase
Se enfocan en conteos de atributos y operaciones para una clase individual y en valores promedio para el sistema OO como un todo.	Se enfocan en la forma en la que las operaciones se reutilizan a lo largo de la jerarquía de clases.	se enfocan en la cohesión y en los conflictos orientados a código.	Examinan el acoplamiento y el reuso.

Métricas de diseño en el nivel de componente

- Se enfocan en las características internas de un componente de software e incluyen medidas de cohesión de módulo, acoplamiento y complejidad

Métricas orientadas a operación (métodos)

- **Tamaño promedio de operación (TOprom)**. El tamaño puede determinarse al contar el número de líneas de código o el de mensajes enviados por el método. *Conforme aumenta el número de mensajes enviados por un solo método, es probable que las responsabilidades no se hayan asignado bien dentro de una clase*.
- **Complejidad de la operación (CO).** Los métodos deben limitarse a una responsabilidad específica, el diseñador debe luchar por mantener la CO tan baja como sea posible.
- **Número promedio de parámetros por operación (NPprom)**. Mientras más grande sea el número de parámetros del método, más compleja es la colaboración entre objetos. En general, el NPprom debe mantenerse tan bajo como sea posible.

Métricas de diseño de interfaz de usuario

- “La correcta distribución de objetos visuales”
  - Posición de las entidades (gráficos, íconos, textos, menús, ventanas, etc.) en la distribución de la pantalla.
  - Frecuencia con la que se usa
  - La dificultad para moverse de una entidad a otra

Métricas de usabilidad y UX

Los usuarios experimentan e interactúan con el software de diferentes formas. Así como es difícil clasificar las emociones de las personas, también es un desafío evaluar su reacción al software. Si bien ninguna métrica de software puede comunicar la totalidad de la experiencia de usuario, hay algunas que son útiles.

- **Métricas de UX**. Las mediciones de UX suelen ser cualitativas y pueden incluir las respuestas emocionales o corporales de los usuarios, como cuánto confían en el software y cómo se mueven sus ojos a través de una interfaz de usuario.
- **Métricas de usabilidad**. La usabilidad mide qué tan bien el software permite a los clientes alcanzar sus objetivos. La usabilidad se puede dividir en componentes más pequeños, como los siguientes:

Facilidad de descubrimiento - eficiencia – memorabilidad - facilidad de aprendizaje – satisfacción - accesibilidad, particularmente accesibilidad digital

- **Net Promoter Score (NPS).** Refleja la voluntad de los clientes de recomendar una aplicación a otros. Net Promoter Score - **Puntuación neta del promotor** - se presenta como un rango de números de 0 a 10. Los clientes con una puntuación de 0 a 6 son Detractores; las puntuaciones 7 y 8 son Pasivos; y 9 y 10 son Promotores.

Métricas para el modelo de implementación

MÉTRICAS PARA CÓDIGO FUENTE

Halstead asignó leyes cuantitativas al desarrollo de software usando un conjunto de *medidas primitivas* que pueden derivarse después de generar el código o de que el diseño este completo. Las medidas son:

- *n*1 = numero de operadores distintos en un programa.
- *n*2 = numero de operandos distintos en un programa
- *N*1 = numero total de ocurrencias de operador
- *N*2 = numero total de ocurrencias de operando

Usos

- Longitud de programa global, $N = n_1 \log_2 (n_1) + n_2 \log_2 (n_2)$
- volumen mínimo potencial para un algoritmo, $V = N \log_2 (n_1 + n_2)$
- volumen real (numero de bits requeridos para especificar un programa),
- nivel del programa (una medida de complejidad del software),
- nivel del lenguaje (una constante para un lenguaje determinado), esfuerzo de desarrollo, tiempo de desarrollo e incluso el numero proyectado de fallas en el software.

Se utiliza en herramientas como **IBM® Application Discovery and Delivery Intelligence (ADDI)** es una plataforma analítica para la modernización de las aplicaciones. Utiliza tecnologías cognitivas para analizar aplicaciones de mainframe y descubrir y comprender rápidamente las interdependencias de los cambios.

SonarQube para diversos lenguajes de programación.

Ejemplo para calcular las medidas de Halstead.
Calcular las medidas de Halstead de longitud y esfuerzo para el siguiente algoritmo:

```
{
for (i=2;i<=n;i++)
for (j=1;j<=i;j++)
if (x[i]<x[j])
{
aux = x[i];
x[i] = x[j];
x[j] = aux;
}
}
```

El número total de operadores (n1) son 10 y la cantidad de operadores (N1) que hay son 23:

2 {..} + **2 for(;;)** + **5 =** + **1 if** + **3 ;** + **1 (..)** + **1 <** + **2 <=** + **2 ++** **4 []**

El número total de operandos (n2) son 5 y la cantidad total (N2) son 22.
7 i + **1 n** + **6 j** + **6 x** + **2 aux**

La longitud es N = N1 + N2 = 23 + 22 = 45
N es una simple medida del tamaño de un programa. Cuanto más grande sea el tamaño de N, mayor será la dificultad para comprender el programa y mayor el esfuerzo para mantenerlo.

El volumen es V = N x log2(n)

= 45 x log2 (10+5)

= 175.8

(nro. mínimo de bits necesarios para codificar el programa)

- El esfuerzo es una medida del trabajo requerido para desarrollar un programa. Desde el punto de vista del mantenimiento, el esfuerzo se puede interpretar como una medida del trabajo requerido para comprender un software ya desarrollado. Cuanto menor es su valor, más fácil será modificar el programa:
- **E**= V * 1/L = V*(n1/2)*(N2/n2)=**175,8***(5)*(22/5) = **3867,6**
- **Tiempo de entendimiento o implementación:**

  T=E/18= 214,87 segundos.

MÉTRICAS PARA PRUEBAS

Métricas de Halstead aplicadas para probar

El esfuerzo de prueba puede estimarse usando métricas derivadas de las medidas de Halstead.

El esfuerzo de un módulo/ la suma de esfuerzo de todos los módulos del sistema.

$$\text{Porcentaje de esfuerzo de prueba } (k) = \frac{e(k)}{\Sigma e(i)}$$

Métricas para pruebas orientadas a objetos

Las métricas consideran aspectos de encapsulación y herencia.

MÉTRICAS PARA MANTENIMIENTO

- Todas las métricas de software presentadas anteriormente pueden usarse para el desarrollo de nuevo software y para el mantenimiento del software existente.
- Sin embargo, se han propuesto métricas diseñadas explícitamente para actividades de mantenimiento.

Índice de madurez de software (IMS)

- Proporciona un indicio de la estabilidad de un producto de software (con base en cambios que ocurran para cada versión del producto).
- Conforme el IMS tiende a 1, el producto comienza a estabilizarse

- M_T = nro de módulos en la versión actual
- F_c = nro de módulos con cambios en la versión actual
- F_a = nro de módulos que se agregaron
- F_d = nro de módulos de la versión anterior borrados en la actual
- $IMS = \frac{M_T - (F_c + F_a + F_d)}{M_T}$