



Ingeniería de Software II (CI-4712)

Métricas de Software

13 de sep de 2006

Referencias básicas

Ingeniería de software. Un enfoque práctico.
Pressman, R. Quinta edición. Mc. Graw Hill
2002

Ingeniería de software. Sommerville, I.
Séptima edición. Addison Wesley 2005

¿Por qué es importante la Medición?

“Cuando puedas medir acerca de lo que hablas y lo expreses en números, sabrás algo acerca de eso, pero cuando no lo puedes medir, cuando no lo puedes expresar en números, tu conocimiento es pobre e insatisfactorio: puede ser el comienzo del conocimiento, pero tienes escasos conocimientos avanzados acerca del estado de la ciencia”. Lord Kevin.

¿Por qué es importante la Medición?

- **Medir:** asignar **números** o **símbolos** a los **atributos de entidades** del mundo de acuerdo a un conjunto de **reglas** definidas claramente.
- La **medida de software** permite cuantificar calendarios de trabajo, esfuerzo de desarrollo, el tamaño del producto, el estado del proyecto y el desempeño de la calidad.

¿Por qué el software debe ser medido?

- Los **proyectos de software** no cumplen con los **tiempos previstos** y **exceden los presupuestos**
- Desarrolladores deben **medir** constantemente el **desempeño** para mejorar las estimaciones futuras.
- Las **métricas** ayudan a **controlar los proyectos** de software de una mejor manera y **aprender** más acerca de **cómo funcionan las organizaciones y procesos**.



Tipos de Medidas

- Medidas para **Producto**: Cuantificar medidas del producto.
- Medidas para **Proceso**: Cuantificar productividad, costo, requerimientos de recursos, etc. del proceso de desarrollo.



Métrica de Software

- Término que abarca un **rango de medidas** de software.
- Las **medidas** se basan en **atributos** del producto.
- Los **atributos** del producto pueden clasificarse en internos y externos.
- Los **atributos internos** describen al producto basado en el producto mismo.
- Se miden de manera estática basado en tamaño, funcionalidad, complejidad y reusabilidad



Métrica de Software

- Los **atributos externos** describen al producto de la manera como funciona en el ambiente.
- El cliente define usualmente estos atributos.
- Algunos atributos externos son adaptabilidad del producto para el problema definido, conformidad con las especificaciones, grado de excelencia, puntualidad.

¿Qué medir?

- Escoger un conjunto de métricas pequeño y balanceado.
- **Goal-Question-Metric (GQM)** es una técnica para selección de métricas.



Goal-Question-Metric

- **Metas:**

- Reducir el costo de mantenimiento en un 40% para el 3er trimestre de este año.
- Reducir el tiempo para resolver cualquier defecto en un 75% para el 2do trimestre del año.
- Reducir el tiempo inutilizado de programadores en un 45% dentro de 3 meses.
- Reducir el tiempo de pruebas de integración en 25 horas en los próximos dos meses.
- Incrementar la reutilización de los componentes de software de un 5% en cada uno de los trimestres del año.
- Aumentar la precisión en la estimación de la programación de los tiempos del proyecto para que se mantenga dentro del 10% de los valores actuales.

Goal-Question-Metric

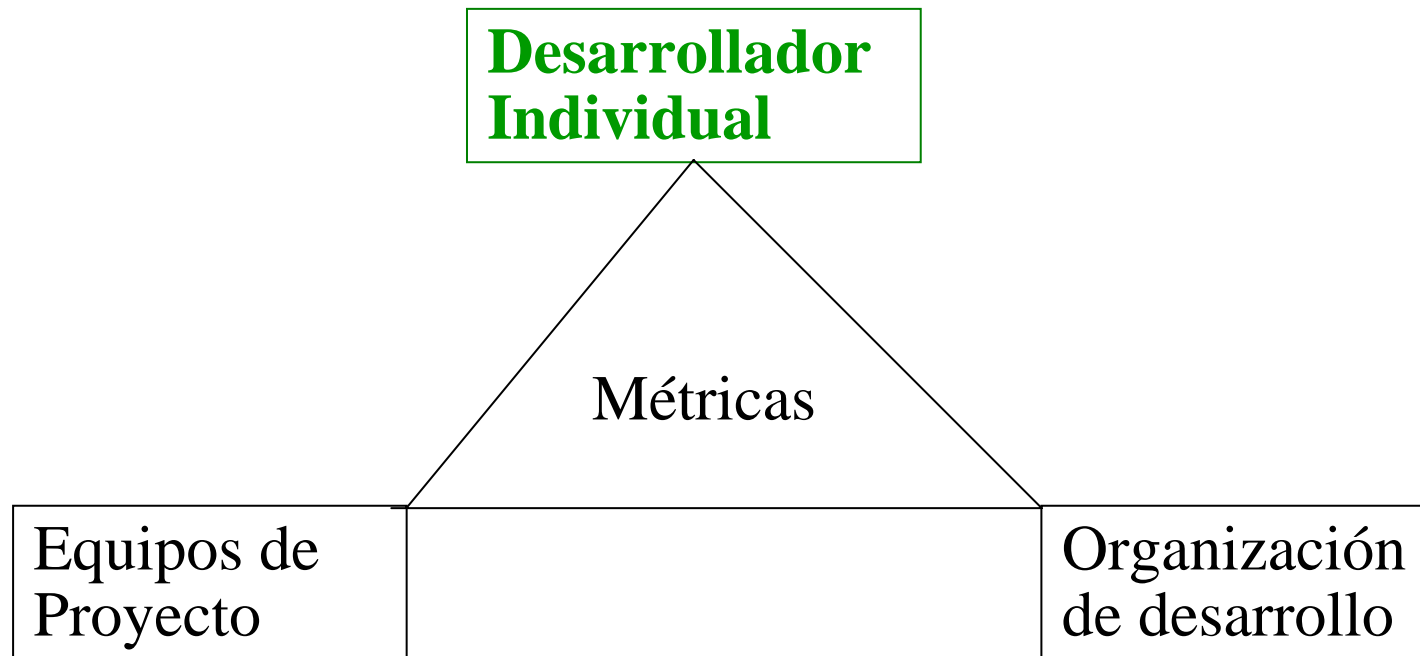
- **Pregunta:** Reducir el costo de mantenimiento en un 40% para el 3er trimestre de este año.
 - ¿cuál es el monto que se gasta en mantenimiento cada mes?
 - ¿qué fracción de los costos de mantenimiento está siendo invertida en cada aplicación que se le brinda soporte?
 - ¿cuál es el monto gastado en mantenimiento adaptativo, perfectivo y correctivo?



Goal-Question-Metric

- **Métricas:**
 - Total de tiempo
 - Dinero gastado

Posibles Métricas

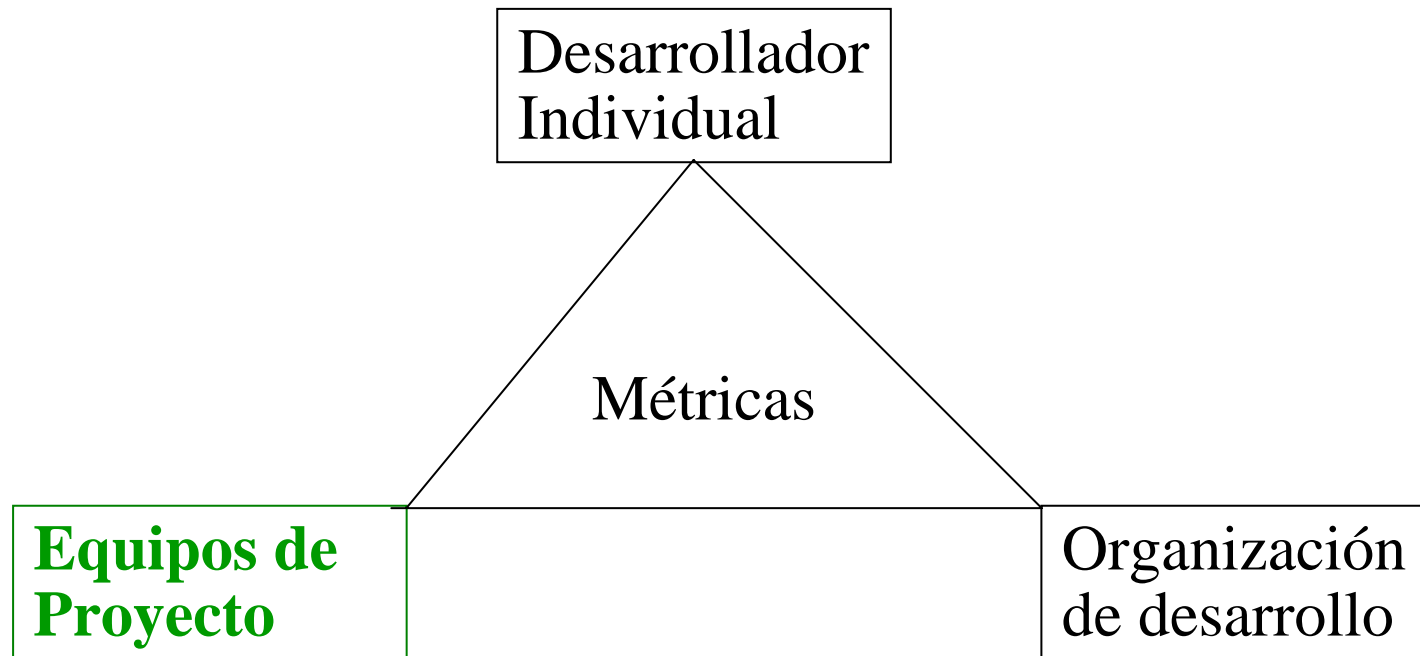




Posibles Métricas

- **Desarrollador Individual:**
 - Nro. defectos encontrados en pruebas unitarias
 - Duración estimada vs. actual de tareas
 - Distribución de esfuerzo de trabajo
 - Extensión de código cubierto por prueba unitaria
 - Complejidad de código y diseño

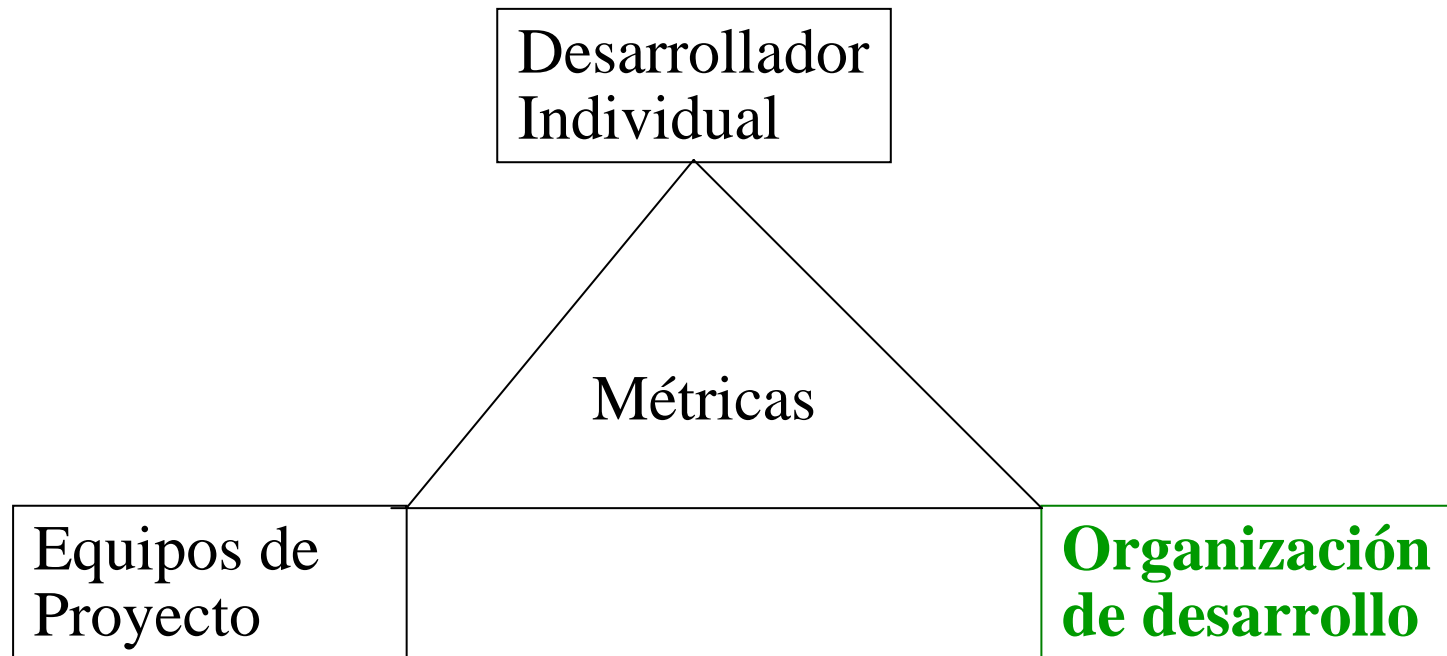
Posibles Métricas



Posibles Métricas

- **Equipos de Proyecto:**
 - Tamaño del producto
 - Distribución de esfuerzo de trabajo
 - Estatus de requerimientos
 - Tareas planeadas vs completadas
 - Proporción de casos de prueba superados
 - Nro. defectos encontrados en cada etapa de prueba
 - Duración actual vs estimada entre metas
 - Estatus del defecto

Posibles Métricas



Posibles Métricas

- **Organización de desarrollo:**
 - Niveles de defecto liberados
 - Ciclo de tiempo de desarrollo de producto
 - Precisión de estimación del cronograma
 - Costo actual vs costo planificado
 - Efectividad de reuso
 - Precisión de estimación de esfuerzo

Medidas sugeridas

Tamaño del producto

Duración y esfuerzo actual vs estimado

Medidas

Defectos

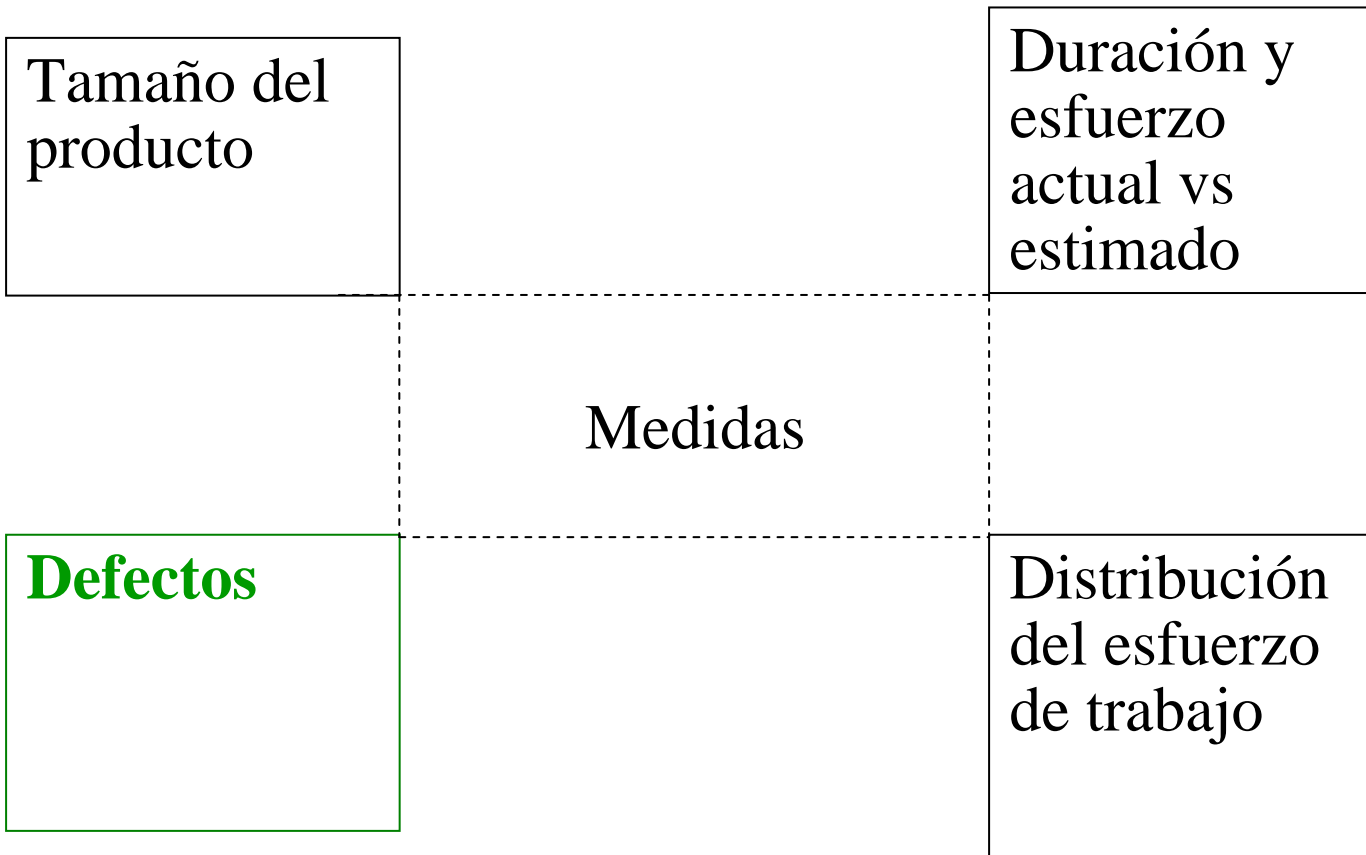
Distribución del esfuerzo de trabajo



Medidas sugeridas

- **Tamaño del producto:**
 - Nro requerimientos
 - Clases de objetos
 - LOC
 - Elementos GUI

Medidas sugeridas



Medidas sugeridas

- **Defectos:**
 - Encontrado por pruebas:
 - Estatus
 - Tipo
 - Severidad
 - Encontrado por cliente:
 - Estatus
 - Tipo
 - Severidad

Medidas sugeridas

Tamaño del
producto

**Duración y
esfuerzo
actual vs
estimado**

Medidas

Defectos

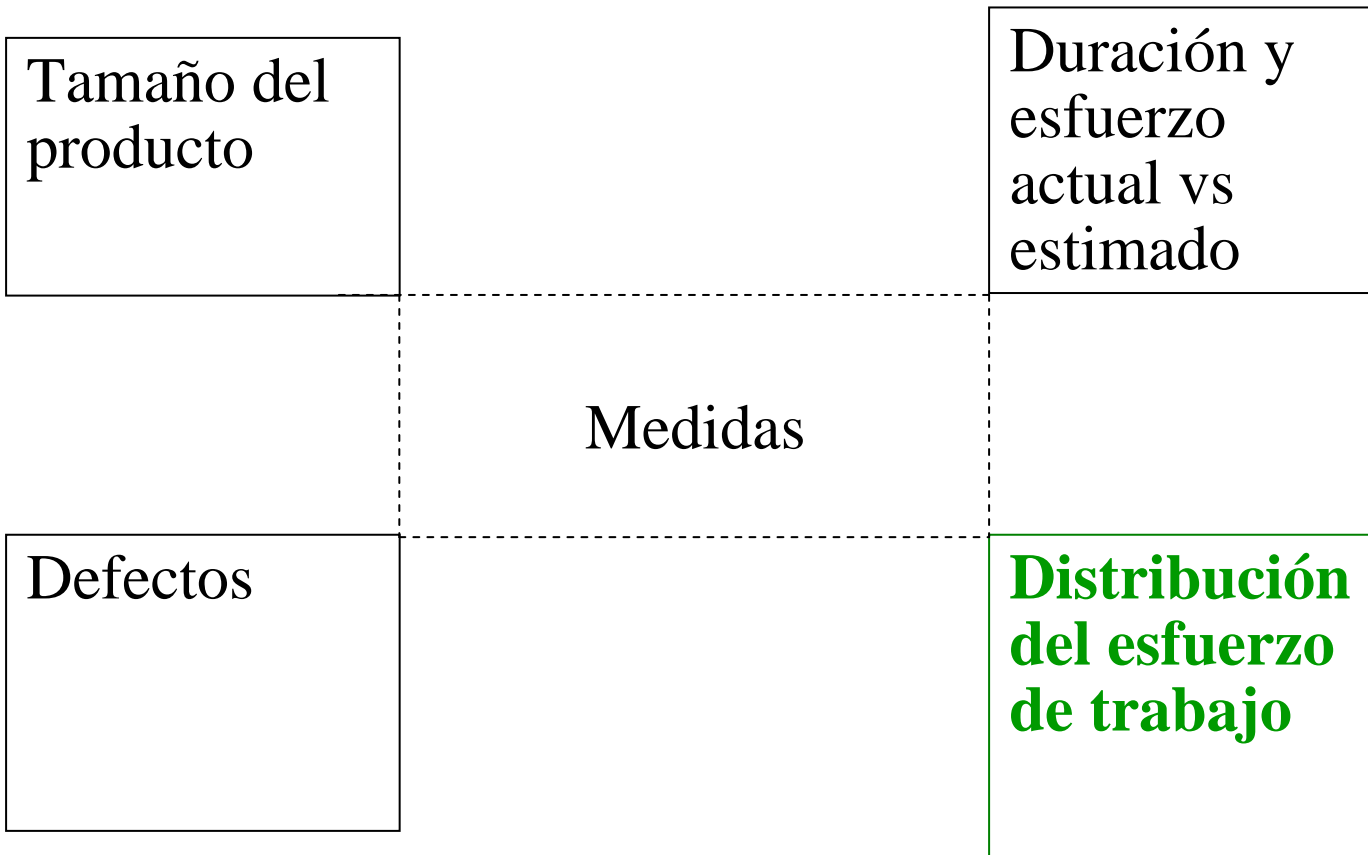
Distribución
del esfuerzo
de trabajo



Medidas sugeridas

- **Duración y esfuerzo actual vs. estimado:**
 - Seguimiento de tareas individuales
 - Seguimiento de metas específicas de proyecto
 - Seguimiento todo el desarrollo de producto

Medidas sugeridas



Medidas sugeridas

- **Distribución de esfuerzo de trabajo:**
 - Tiempo invertido en actividades de desarrollo
 - Administración del proyecto
 - Diseño
 - Codificación
 - Prueba
 - Requerimientos
 - Tiempo invertido en actividades de mantenimiento
 - Adaptativo
 - Perfectivo
 - Correctivo

Medidas de software - Clasificación

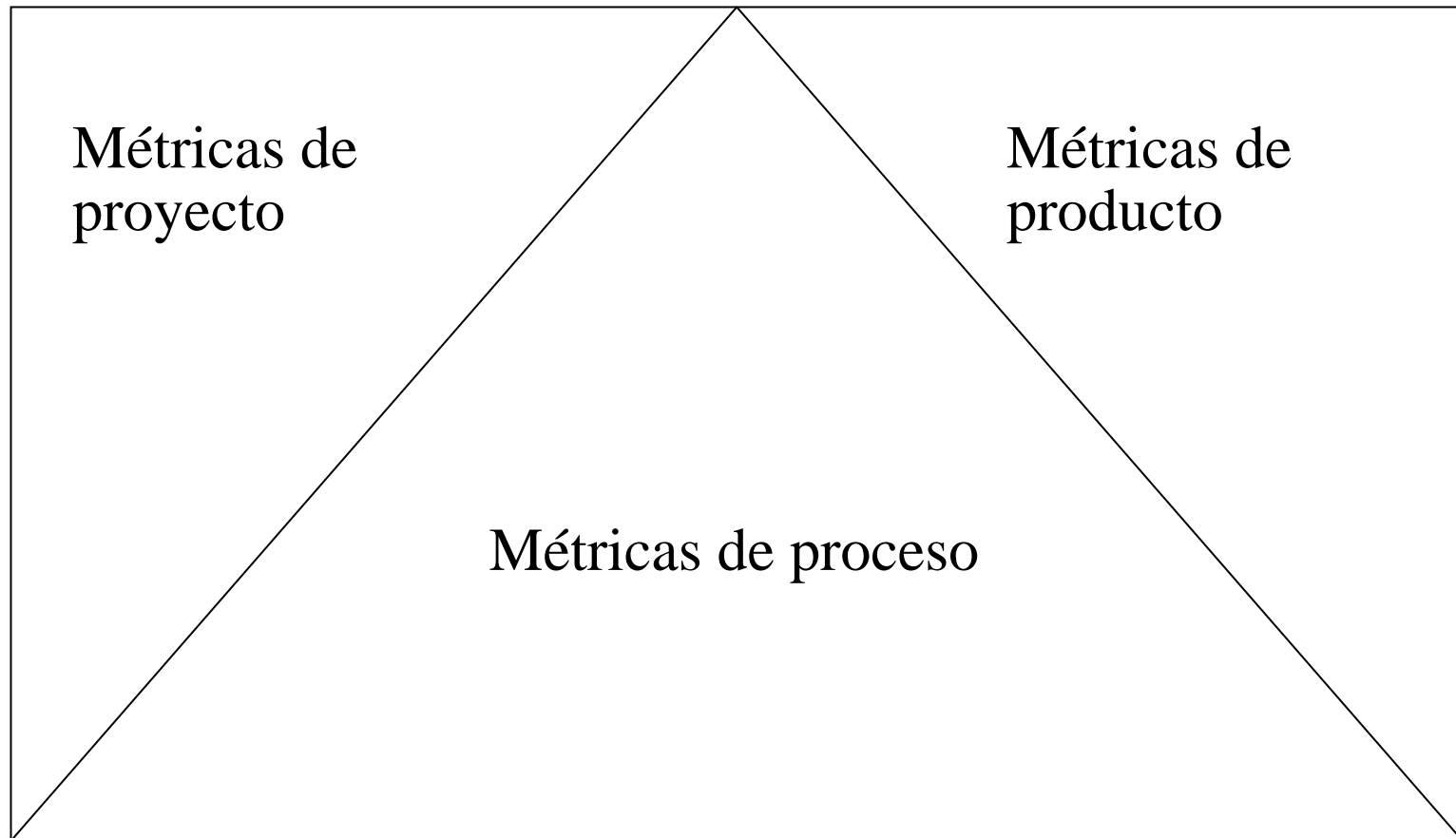
- **Medidas directas:**
 - Incluyen costos y esfuerzo aplicado en el proceso de desarrollo de software:
 - LOC
 - Velocidad de ejecución
 - Tamaño de memoria requerida
 - Nro. defectos reportados sobre cierto período de tiempo



Medidas de software - Clasificación

- **Medidas indirectas:**
 - Calidad
 - Complejidad
 - Eficiencia
 - Confiabilidad
 - Mantenimiento
 - Usabilidad
 - Reutilización
 - Disponibilidad
 - Extensión
 - Mejoramiento

Métricas de software - Areas



Métricas orientadas al tamaño

- Medida directa de software y el proceso por el cual fue desarrollado
- Pueden determinar fácilmente el número de líneas de código y compararlas con el esfuerzo, dinero gastado, KLOC, las páginas de documentación creadas, errores y el número de desarrolladores involucrados en los proyectos.
- Algunas métricas:
 - **Productividad del programador** = KLOC/persona-mes
 - **Calidad del código** = $\text{total de defectos observados/KLOC}$
 - **Costo de desarrollo** = $\text{costo total del proyecto/KLOC}$



Métricas orientadas al tamaño

- Para obtener un programa de calidad del software, se puede usar:
 - La densidad de defecto
 - Agrupaciones de defectos
 - La siembra de defectos
 - Y sus combinaciones

Métricas orientadas al tamaño

- **La densidad de defecto**

- Se define como **número de defectos** por **LOC**
- Se puede usar para determinar si una versión está lista para liberarse
- Ejemplo

Versión	LOC	Defectos (antes)	Densidad	Defectos (después)	Densidad
1.0	650.000	2.500	3,84 KLOC	400 +	4,46 KLOC
2.0	800.000	2.300	2,87 KLOC	225 +	3,15 KLOC
3.0	900.000	2100	2,33 KLOC	?	?

Métricas orientadas al tamaño

- **Agrupaciones de defectos**
 - Suponga dos agrupaciones X y Y
 - Todos los defectos descubiertos los Lunes, Martes y Miércoles en agrupación X
 - Sean D_x y D_y el número de defectos encontrados en las agrupaciones X y Y respectivamente.
 - **Total de defectos únicos** = $D_x + D_y - \text{Defectos encontrados en ambos X y Y}$
 - **Numero total de defectos** = $(D_x * D_y) / \text{Defectos encontrados en ambos X y Y}$

Métricas orientadas al tamaño

- **Agrupaciones de defectos**

- Versión 3.0

- $D_x = 475$, $D_y = 370$, Defectos encontrados en ambos X y Y = 125

- **Total de defectos únicos** = $D_x + D_y - \text{Defectos encontrados en ambos X y Y} = 475 + 370 - 125 = 720$

- **Numero total de defectos** = $(D_x * D_y) / \text{Defectos encontrados en ambos X y Y} = (475 * 370) / 125 = 1406$

- Hay aproximadamente $1406 - 720 = 686$ defectos que faltan por ser detectados $\rightarrow 48.79\%$ de los defectos están por ser detectados.

Métricas orientadas al tamaño

- **La siembra de defectos**
 - Un grupo encargado de las pruebas introduce defectos de software intencionalmente
 - Otro grupo encargado de las pruebas trabaja para detectar defectos
 - Estimar la **proporción** del **número detectado de defectos sembrados** contra **el número total de defectos sembrados intencionalmente**.
 - **Importante**: sembrar los defectos de manera uniforme para cubrir todos los aspectos de la funcionalidad del software y todos los defectos sembrados deben ser eliminados antes de la prueba final del sistema

Métricas orientadas al tamaño

- **La siembra de defectos**

- Total de defectos sembrados = 75
- Nro total de defectos sembrados detectados = $D_s = 35$
- Nro total de defectos sin sembrar encontrados = $D_{uf} = 520$
- Nro total de defectos sin sembrar = (total de defectos sembrados/ D_s)* $D_{uf} = (75/35)*520 = 1.114,28$

Métricas orientadas a la funcionalidad

- 12 meses-hombres de esfuerzo para realizar el análisis de los requerimientos, diseño, etc.
- 4000 LOC en C++ en tres meses
- 800 LOC en Smalltalk en un mes
- Costo total para el desarrollo en C++ = \$65.000
- Costo total para el desarrollo en Smalltalk = \$45.000
- Costo por LOC en C++ = $\$65.000/4.000 = \$16,25$
- Costo por LOC en Smalltalk = $\$45.000/800 = \$56,25$
- LOC por mes con C++ = $4.000/3 = 1.333,33$
- LOC por mes con Smalltalk = $800/1 = 800$

Métricas orientadas a la funcionalidad

- **Idea:** Estimar el tamaño de los proyectos de software de manera acertada sin importar el lenguaje de programación
- Albrecht enumera cinco aspectos externos visibles de cualquier software:
 1. Entradas de la aplicación
 2. Salidas producidas por la aplicación
 3. Nro. de consultas hechas por los usuarios
 4. Nro. de archivos de datos que usará la aplicación
 5. Nro. de interfaces a otras aplicaciones

Métricas orientadas a la funcionalidad

- Albrecht desarrolló una **métrica punto de función** y comprende calcular puntos con pesos.

Parámetro	Cantidad	Peso	Puntos (cantidad por peso)
Entrada	1	4	4
Salida	1	5	5
Consultas	1	4	4
Archivos	1	10	10
Interfaces	1	7	7
Suma-Puntos			30

Métricas orientadas a la funcionalidad

- **Punto de función = Suma-Puntos + $(\sum CAV_i * 0,01 + 0,65)$**
- **CAV_i** son valores de ajuste de complejidad
- **i** varía entre 1 y 14.
- Los valores son capturados en base a las respuestas de 14 preguntas presentadas por L.J. Arthur
- Para cada pregunta se debe dar una respuesta en la escala del 0 al 5

Métricas orientadas a la funcionalidad

1. ¿El sist. requiere un respaldo y una recuperación confiable?
2. ¿Se requiere de comunicaciones de datos?
3. ¿Hay funciones de procesamiento distribuido?
4. ¿El rendimiento es crítico?
5. ¿El sist. se ejecutará en un ambiente operacional altamente utilizado?
6. ¿El sist. requiere ingreso de datos en línea?
7. ¿El ingreso de los datos en línea requiere que la entrada de la transacción se construya a través de múltiples pantallas u operaciones?
8. ¿Se actualizan los archivos maestros en línea?
9. ¿Son complejas las entradas, salidas archivos o consultas?
10. ¿El procesamiento interno es complejo?
11. ¿El código diseñado es reutilizable?



Métricas orientadas a la funcionalidad

12. ¿Están la conversión e instalación incluidas en el diseño?
13. ¿Está el sistema diseñado para múltiples instalaciones en diferentes organizaciones?
14. ¿Está la aplicación diseñada para facilitar el cambio y la facilidad de uso para el usuario?

Métricas orientadas a la Característica

- Software Productivity Research Inc. desarrolló la **métrica punto de característica** en el año 1986.
- Se usan en aplicaciones que tienen a los algoritmos como factor dominante.
- La **métrica punto de característica** usa los mismos parámetros que la **métrica de punto de función**
- Adicionalmente usa los algoritmos como un parámetro con peso por defecto 3.

Métricas orientadas a la Característica

Parámetro	Cantidad	Peso	Puntos (cantidad por peso)
Entrada	1	4	4
Salida	1	5	5
Consultas	1	4	4
Archivos	1	7	7
Interfaces	1	7	7
Algoritmos	1	3	3
Suma-Puntos			30

Métricas orientadas a la Característica

- El cálculo del **punto de característica** = $Nelem_{il} * weight_{il} + Nelem_{im} * weight_{im} + Nelem_{ih} * weight_{ih}$
- $Nelem_{il}$, $Nelem_{im}$ y $Nelem_{ih}$: nro. de ocurrencias de elementos i. Ejemplo: salidas para cada nivel de complejidad bajo (l), medio (m) y alto (h)
- $Weight_{il}$, $weight_{im}$ y $weight_{ih}$: pesos correspondientes a los elementos i.
- El peso que representa los niveles de complejidad de los algoritmos varían de 1 a 10.



Métricas relacionados a la Complejidad

- Dos métricas principales:
 - La ciencia de software de Halsted
 - El número de complejidad ciclomática de McCabe

Ciencia de software de Halstead

- Métrica desarrollada por Halstead en 1977.
- Constituye un conjunto de métricas diseñadas para estimar el esfuerzo de programación
- Los operadores aritméticos, operadores relacionales, operadores booleanos, índices o un separador de sentencias son considerados **OPERADORES**.
- Los **OPERANDOS** incluyen literales, constantes y expresiones usadas en un programa

Ciencia de software de Halstead

- OperadoresUnicos = Nro. de un operador único o distinto
- OperandosUnicos = Nro. de operandos únicos o distintos
- TotalOperadores = Uso total de todos los operadores
- TotalOperandos = Uso total de todos los operandos
- Vocabulario = OperadoresUnicos + OperandosUnicos
- LongitudImplementacion = TotalOperadores + TotalOperandos

Ciencia de software de Halstead

- Se define la longitud de un programa en función del vocabulario como sigue:

$$\text{Longitud-calculada} = (\text{TotalOperadores} * \log_2 \text{TotalOperadores}) + (\text{TotalOperandos} * \log_2 \text{TotalOperandos})$$

- Para cuantificar el volumen de un programa:

$$\text{VolumenPrograma} = \text{Longitud-calculada} * \log_i(\text{Vocabulario})$$

- VolumenPotencial describe el mejor algoritmo a través de:

$$\text{VolumenPotencial} = (\text{OperadoresUnicos} + \text{OperandosUnicos}) * \log (\text{OperadoresUnicos} + \text{OperandosUnicos})$$

Ciencia de software de Halstead

- **NivelPrograma =**
VolumenPotencial/VolumenPrograma
- Para cuantificar el “contenido inteligente” de un programa:

$$\text{ContenidoInteligente} = \text{NivelPrograma} * \text{VolumenPrograma}$$

- El esfuerzo de programación se calcula como:

$$\text{EsfuerzoProgramacion} = \text{VolumenPrograma}/\text{NivelPrograma}$$

Número de complejidad ciclomática de McCabe

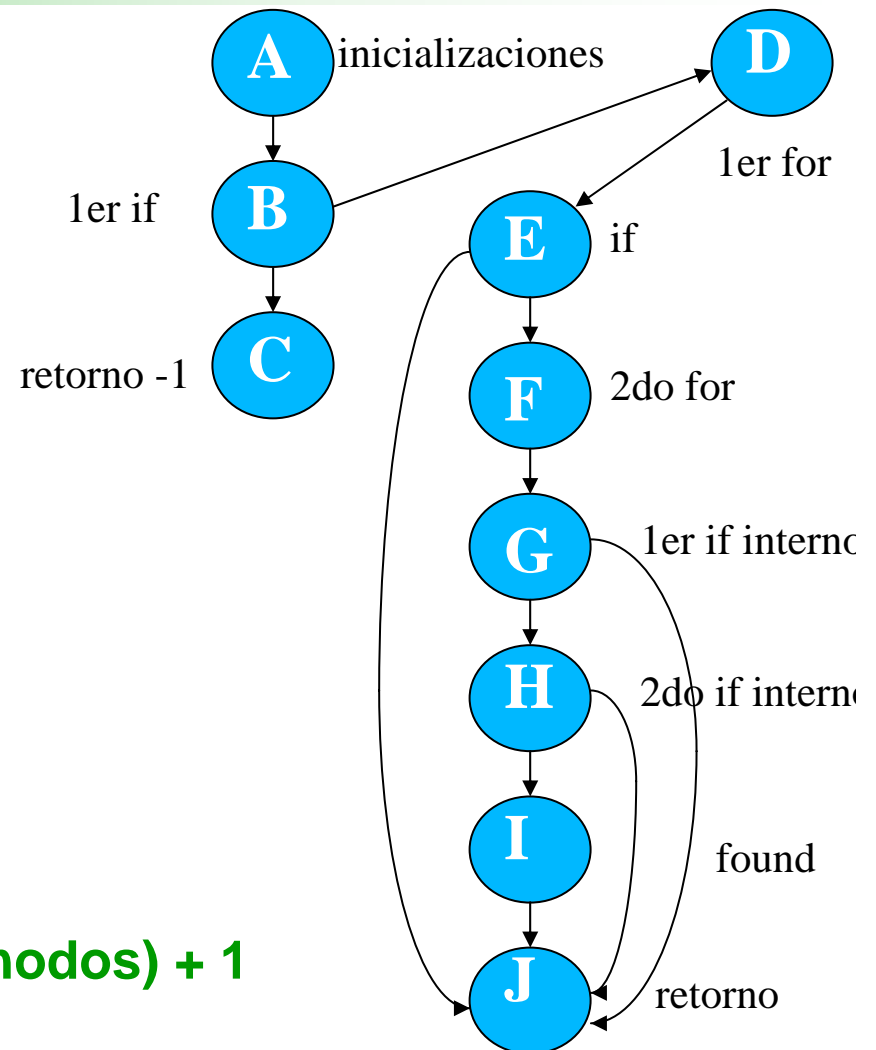
- Métrica desarrollada en 1971.
- Calcula el número de caminos independientes en un programa
- El programa es representado como un grafo
- Sea G el grafo de un programa, la complejidad ciclomática del grafo G se calcula como:

$$V(G) = \text{Numero(aristas)} + \text{Numero(nodos)} + 1$$

Número de complejidad ciclomática de McCabe

```
int edgeExists(Graph_type graph, int
  start, int end) {
  int i, j, found = 0;
  If (start < 0 || start >= graph.n || end < 0 ||
    end > graph.n)
    return -1;
  For (i=0; i<graph.n && !found;i++)
    if (i == start - 1)
      for (j=0;j<graph.n&&!found;j++)
        if (j == end - 1)
          if (graph.adjTable[i][j]==1)
            found = 1;
  return found;
}
```

$$V(G) = \text{Numero(aristas)} + \text{Numero(nodos)} + 1$$
$$= 12 - 10 + 1 = 3$$





Número de complejidad ciclomática de McCabe

- Estudios empíricos indican que cuando se programa en C o Pascal, la complejidad ciclomática se calcula entre 5 y 9, lo cual conlleva a que los programas tengan pocos defectos.
- Otros estudios muestran que el software cuya complejidad ciclomática es controlada entre 10 y 15 se minimiza el número de cambios hechos a los módulos.

Métricas Orientadas a Objetos

- **Métricas propuestas por Morris en su tesis de grado de Maestría en el año 1.989:**
- **Método por clase**
Promedio del nro. de métodos por clases = $\frac{\text{total de nro. de métodos}}{\text{total de métodos por objetos de clases}}$
- **Dependencia de Herencia**
Profundidad en el árbol de herencia = max (longitud del árbol)
- **Efectividad de colecciones de objetos**
Promedio = $\frac{\text{nro. total de objetos reutilizados}}{\text{nro. total de colecciones de objetos}}$

Métricas Orientadas a Objetos

- **Métricas propuestas por Morris en su tesis de grado de Maestría en el año 1.989:**
- **Promedio de la complejidad del método**
Promedio de la complejidad del método = suma de la complejidad ciclométrica de todos los métodos / número total de métodos de la aplicación
- **Granularidad de la aplicación**
Granularidad de la aplicación = nro. total de objetos / total de funciones de punto

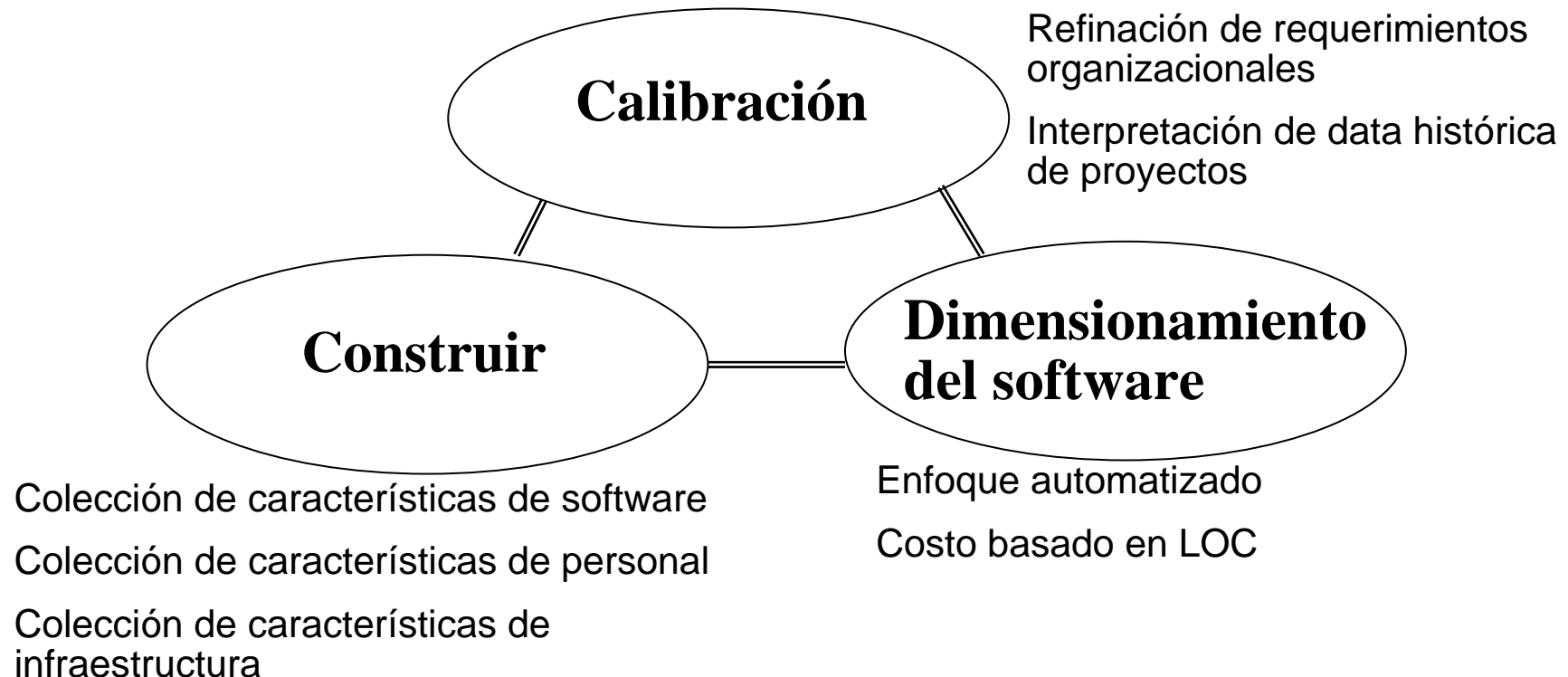


Modelos de Estimación de Costos

- Proveen estimados de costos de desarrollo de software.
- Usan datos históricos
- Los diferentes componentes de costos son analizados usando métodos matemáticos para proveer una fórmula que enlace los costos de entrada con el estimado de salida.
- Modelos de costos conocidos:
 - SLIM o Modelo de Administración de Ciclo de Vida de Software
 - COCOMO o Modelo de Costos Constructivos

SLIM

- Conocido también como Putnam
- Desarrollado por Putnam en 1978
- Útil en proyectos grandes



COCOMO

- Desarrollado por Barry Boehm en 1981
- Conocido como COCOMO'81
- Usado en la industria del software

