Administración de Proyectos de Software

Métricas en la Ingeniería de Software

E. Estévez - P. Fillottrani

Depto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Segundo Cuatrimestre 2017

Métricas en la Ingeniería de Software

Conceptos Inciales

- las medidas y las métricas se pueden definir como:

  herramientas que ayudan en la planificación y estimación de proyectos, y a su vez roporcionan datos cuantitativos sobre la calidad y productividad del proceso y del producto

Medidas, Métricas e Indicadores

- medidas: manejan valores independientes
- métricas: manejan relaciones de medida e indican una medida de calidad o productividad
- ejemplo: *LOC*/*persona*
- indicadores: evalúan una o más métricas para sacar conclusiones respecto a algún aspecto del software o del proceso

¿Por qué Medir?

- los gerentes de proyecto miden atributos del proceso y del producto poder responder:
  - cuándo el software estará disponible
  - si el presupuesto será cumplido o no
- los clientes expertos miden los aspectos del producto final para determinar si sus requerimientos fueron satisfechos
- los encargados de mantenimiento miden el producto actual para saber qué debe mejorarse o actualizarse

Medidas cotidianas

- medidas en Economía: determinan variaciones de precios y salarios
- medidas en Sistemas de Radares: permiten volar a los aviones
- medidas en Medicina: permiten a los doctores diagnosticar enfermedades
- medidas en Sistemas Atmosféricos: predicen el tiempo
- sin medidas no puede funcionar la tecnología

¿Qué es una Medida?

- medición: es el proceso por el cual se asignan números o símbolos a atributos de entidades del mundo real para que lo describan de acuerdo a reglas definidas claramente
- entidad: es un objeto o un evento del mundo real
- atributo: es una propiedad de una entidad
- se miden atributos de las entidades
- los números y símbolos utilizados son abstracciones que:
  - usamos para reflejar nuestra percepción del mundo real
  - preservan las relaciones que observamos entre las entidades

Medir

- la idea de medición hace los conceptos más visibles y por lo tanto más comprensibles y controlables
- el debate de cómo medir ya permite mayor comprensión
- existen dos clases de cuantificaciones:
  - medición: es una cuantificación directa
  - cálculo: es una cuantificación indirecta. Se toman las medidas y se combinan en un ítem cuantificado que refleja algún atributo
- ejemplo de medición: medir una habitación; ejemplo de cálculo: la valuación de una casa

Características de las Métricas

- las métricas deben ser:
  - exactas
  - precisas no se debe perder información en los redondeos ya que la información se desvirtúa
  - consistentes distintas mediciones del mismo atributo deben dar el mismo valor

Proceso de Adopción de Métricas

- fase de aprendizaje
  - no se tienen métricas, y es necesario analizar muchas medidas ya que no se sabe cuáles contribuiran a una métrica útil
  - esto implica mucho esfuerzo y poco beneficio
- fase de uso ya se tienen las métricas, poco esfuerzo y aumenta el beneficio

Medir en el Software

- "No se puede controlar lo que no se puede medir" De Marco,1982
- Gerentes:
  - ¿cuánto cuesta este proceso?
  - ¿qué productividad tiene el personal?
  - ¿cuánto de bueno es el producto desarrollado?
  - ¿estará el usuario satisfecho?
  - ¿cómo se puede mejorar?
- Ingenieros:
  - ¿podemos verificar los requerimientos?
  - ¿hemos encontrado todos los errores?
  - ¿hemos cumplido los objetivos?
  - ¿qué pasará en el futuro?

Objetivos de Medir en el Software

- las medidas nos ayudan a:
  - comprender lo que sucede durante el desarrollo y mantenimiento
  - controlar proyectos
  - mejorar productos y procesos

Métricas en el Software

- estimación de Costo y Esfuerzo - Modelo COCOMO, de Putnam, de Albrecht
- medidas y modelos de productividad
- modelos y medidas de calidad
- modelos de confiabilidad

Teoría Representacional

- en cualquier actividad de medición, hay reglas a seguir
- las reglas nos ayudan a ser consistentes con nuestras mediciones, y nos proveen una base para la interpretación de los datos
- la teoría de medición nos dice las reglas
- el enfoque basado en reglas es común en muchas ciencias
- dependiendo del conjunto de reglas elegidas, existen distintas teorías
- se presentará la teoría representacional de mediciones
- la teoría representacional de las mediciones trata de formalizar nuestra intuición sobre la forma en que funcionan las cosas

Relaciones

- las medidas deberían representar atributos de las entidades que observamos
- la manipulación de estos datos deberían preservar las relaciones que observamos
- nuestra intuición es el punto de partida para todas las mediciones

Ejemplo

- por ejemplo, para aprender sobre altura, podemos decir que A es más alto que B sin medirlos
- en general, definimos una relación binaria. Ejemplo: “más alto que” es una relación binaria definida en el conjunto de pares de personas
- dadas cualesquiera dos personas, x e y, podemos observar que: x es más alto que y o y es más alto que x
- “más alto” que es una relación empírica para altura

Relación Empírica

- una relación empírica (binaria) es aquella para la cual hay un consenso razonable acerca de qué pares están en la relación
- las relaciones empíricas no necesariamente son binarias. Pueden ser unarias o de grado > 2
- ejemplo de relación unaria: es alto
- estas relaciones son mapeos del mundo real, empírico, a un mundo matemático formal
- el mapeo matemático debe preservar las relaciones que observamos

Medición

- formalmente se define medición como un mapeo del mundo empírico a un mundo formal, relacional
- una medida es el número o símbolo asignado a una entidad por este mapeo en orden a caracterizar un atributo
- algunas veces, las mediciones no son exactas
- dependen de características subjetivas de la persona que realiza la clasificación
- ejemplo: la tarea de catar vinos - se realiza una aseveración subjetiva, pero el resultado no es necesariamente una medida, en el sentido de la teoría de las mediciones

Medición

- la medición debe especificar el dominio, el rango y las reglas para realizar el mapeo
- el mundo real es el dominio del mapeo, y el mundo matemático es el rango
- regla: el mapeo debe preservar la relación
- esta regla es llamada: representación u homomorfismo

Medición

- el mapeo debe preservar la relación
- cuando se verifica la regla de representación, se puede definir al mapeo como una medida para el atributo
- esta condición asegura que un mapeo de medición M, debe mapear:
  - entidades en números
  - las relaciones empíricas en relaciones numéricas

  de tal manera que las relaciones empíricas preserven y sean preservadas por las relaciones numéricas
- por ejemplo A es más alto que B si y solo si $M(A) > M(B)$

Proceso de medir

1. identificar atributos de algunas entidades del mundo real
2. identificar la relación empírica para el atributo
3. identificar la relación numérica correspondiente a cada relación empírica
4. definir el mapeo de las entidades del mundo real a números
5. controlar que las relaciones numéricas preserven y sean preservadas por las relaciones empíricas

Modelos

- un modelo es una abstracción de la realidad, que permite eliminar detalles y visualizar una entidad o concepto desde una perspectiva particular
- el modelo del mapeo debería suplementarse con un modelo del dominio del mapeo, es decir con un modelo de cómo se relacionan las entidades con sus atributos
- ejemplo: para medir la longitud de un programa necesitamos un modelo del programa
- en el proceso de mediciones existe un peligro: focalizar demasiado en el sistema matemático formal y no lo suficiente en el empírico

Medidas directas e indirectas

- medidas directas en Ingeniería de Software:
  - longitud de código fuente (LOC)
  - duración del proceso de testeo (horas)
  - número de defectos descubiertos durante el testeo (número. de defectos)
- medidas indirectas en Ingeniería de Software:
  - productividad de programadores (LOC/unidad de tiempo) – es conflictiva
  - densidad de defectos (número de defectos/tamaño),
  - eficiencia en detección de errores (número de defectos detectados/número de defectos)

Medir para predecir

- cuando se habla de medición se presume que se calcula sobre una entidad existente
- en algunos casos necesitamos predecir un atributo de una entidad que no existe
- ejemplo: confiabilidad de un producto
- la diferencia entre medición para asegurar y predicción no siempre es clara. Ejemplo: un viaje en auto
- las mediciones para predicción siempre requieren alguna clase de modelo matemático que relacione los atributos a predecir con algún otro atributo de una entidad existente que se pueda medir
- los modelos no necesitan ser complejos para ser útiles

Medir para predecir

- la predicción del esfuerzo es universalmente necesaria para los líderes de proyecto
- ejemplo: predecir la cantidad de páginas de un listado de un programa fuente

$$\text{cantidad páginas} = LOC/\text{líneas por página}$$

- un sistema de predicción consiste de un modelo matemático junto con un conjunto de procedimientos de predicción para determinar parámetros desconocidos y para interpretar resultados
- algunas veces el mismo modelo es usado para evaluar y para predecir

Medir para predecir

- ejemplo: predecir el costo de un viaje en auto de Bahía Blanca a Buenos Aires
- se comienza por obtener algunas medidas:
  - d: la distancia entre Bahía Blanca y Buenos Aires
  - g: el costo del litro de gasoil
  - l: el promedio de distancia que se recorre con un litro de gasoil
- se puede predecir el costo del viaje como:

$$\text{costo} = d * g/l$$

Medir para predecir

- se usa un sistema de predicción que incluye
  - un modelo, la fórmula costo $= d * g/l$
  - un conjunto de procedimientos - para determinar los parámetros del modelo, cómo determinamos los valores d, g y l
  - procedimientos para interpretar los resultados, por ejemplo podemos determinar una función de probabilidades para determinar el margen de error
- aún usando el mismo modelo se pueden lograr distintos resultados si se utilizan procedimientos de predicción diferentes

Escalas de medición

- no todos los mapeos de mediciones son iguales. Las diferencias entre los mapeos pueden restringir la clase de análisis que se puede realizar
- una escala de medición esta dada por el mapeo de medición M, junto con el sistema de relación numérica
- el sistema de relación numérica, dado por el dominio y el rango, son obvios a partir del contexto
- a menudo, nos referimos sólo a M como la escala

Escalas de medición

- ¿cómo se determina que un sistema de relación numérico es mejor que otro? respuesta pragmática: en lo posible usar números reales
- problema de representación: ¿Cómo sabemos, si un sistema de relación empírico particular, tiene una representación en un sistema de relación numérico dado? respuesta: es tema de estudio de la teoría de mediciones
- problema de unicidad: ¿Qué hacer cuando tenemos varias representaciones posibles diferentes (varias escalas) en el mismo sistema de relación numérico? respuesta: la solución depende de nuestra habilidad para determinar qué representación es la más adecuada para medir un atributo

Tipos de Escala

- nominal
- ordinal
- de intervalos
- de razones
- absoluta

Tipos de Escala

- un sistema relacional se dice más rico que otro, si todas las relaciones del segundo están contenidos en el primero
- los tipos de escala mencionados, están enumerados en orden creciente de niveles de riqueza
- cuanto más rico es el sistema de relación empírico, más restrictivo es el conjunto de representaciones, y por lo tanto, más sofisticada la escala de medición

Tipos de Escala

- la idea detrás de las definiciones formales de tipos de escala es simple: tenemos una medida satisfactoria para un atributo, con respecto a un sistema de relación empírico
- queremos saber qué otras medidas existen que sean también aceptables
- ejemplo: medimos la longitud de objetos físicos en pulgadas. Existen otras medidas aceptables: cm., pies, millas, etc.
- un mapeo de un sistema de medición aceptable a otro sistema de medición se denomina transformación admisible
- ejemplo: en mediciones de longitud, la clase de transformaciones admisibles es restrictiva, y son de la forma $M' = a * M$

Escala Nominal

- se definen clases o categorías, y luego se ubica a cada entidad en una clase o categoría particular
- la escala nominal tiene 2 características importantes:
  - el sistema de relación empírico consiste sólo de diferentes clases - no existe noción de orden entre las clases
  - cualquier representación de distinción numérica o simbólica de las clases es una medida aceptable, pero no hay noción de magnitud asociada con los números o símbolos
- la medición de escala nominal ubica los elementos en un esquema de clasificación
- las clases no están ordenadas

Escala Nominal - ejemplo

- deseamos detectar el origen de los errores detectados en el software
- la escala de medición tiene a los errores como entidades y la "ubicación" como atributo
- el mapeo para determinar la ubicación será: "especificación", "diseño" o "codificación" dependiendo de dónde fue introducido el error
- un error puede pertenecer sólo a una clase

$$M_1(x) = \begin{cases} 1 & \text{si } x \text{ es de especificación} \\ 2 & \text{si } x \text{ es de diseño} \\ 3 & \text{si } x \text{ es de codificación} \end{cases}$$

Escala Nominal - ejemplo

- la siguiente también es una medida aceptable:

$$M_2(x) = \begin{cases} 800 & \text{si } x \text{ es de especificación} \\ 55{,}6 & \text{si } x \text{ es de diseño} \\ 932 & \text{si } x \text{ es de codificación} \end{cases}$$

Escala Ordinal

- a menudo, es útil aumentar la escala nominal con información de orden de las clases o categorías
- la escala ordinal tiene las siguientes características:
  - el sistema de relación empírica consiste de clases que están ordenadas con respecto al atributo
  - cualquier mapeo que preserve el ordenamiento es aceptable
  - los números sólo representan un ranking
- no tienen sentido las operaciones matemáticas de suma, resta, multiplicación y división

Escala Ordinal - ejemplo

- el conjunto de entidades es un conjunto de módulos de software y el atributo a medir es la “complejidad”
- podemos definir 5 clases de complejidad de módulos: `trivial`, `simple`, `moderado`, `complejo`, `incomprensible`.
- existe un orden implícito que es “menos complejo que”
- en este caso, el mapeo de medición debe preservar el orden
- cualquier mapeo M debería ser una función creciente

Escala Ordinal - ejemplo

- medidas aceptables:

$$M_3(x) = \begin{cases} 1 & \texttt{trivial} \\ 2 & \texttt{simple} \\ 3 & \texttt{moderado} \\ 4 & \texttt{complejo} \\ 5 & \texttt{incomprensible} \end{cases} \qquad M_4(x) = \begin{cases} 5 & \texttt{trivial} \\ 8 & \texttt{simple} \\ 33 & \texttt{moderado} \\ 90 & \texttt{complejo} \\ 784 & \texttt{incomprensible} \end{cases}$$

Escala Ordinal - ejemplo

- medidas no aceptables:

$$M_5(x) = \begin{cases} 1 & \texttt{trivial} \\ 2 & \texttt{simple} \\ 2 & \texttt{moderado} \\ 5 & \texttt{complejo} \\ 6 & \texttt{incomprensible} \end{cases} \qquad M_6(x) = \begin{cases} 2 & \texttt{trivial} \\ 8 & \texttt{simple} \\ 4 & \texttt{moderado} \\ 6 & \texttt{complejo} \\ 10 & \texttt{incomprensible} \end{cases}$$

Escala de Intervalos

- esta escala captura información sobre el tamaño de los intervalos que separan las clases, de tal forma que permita comprender de alguna manera, la magnitud del salto de una clase a la otra
- la escala de intervalos tiene las siguientes características:
  - preserva el orden, al igual que una escala ordinal
  - preserva diferencias, pero no proporciones - conocemos las diferencias entre cualquier par de clases ordenadas en el rango de mapeo, pero no tiene sentido calcular la razón de dos clases
- la suma y la resta son aceptables, pero no las operaciones de multiplicación y división

Escala de Intervalos - ejemplo

- ejemplo de Medición de Temperatura: se miden 20 grados de temperatura en Bahía Blanca y 30 grados en Salta
- el intervalo de un grado es el mismo, si la temperatura aumenta de 20 a 21 grados en Bahía Blanca, el calor aumenta de la misma manera que si subiera de 30 a 31 grados en Salta

Escala de Intervalos - ejemplo

- se desea medir la complejidad, al igual que en el caso anterior
- se asume que la diferencia de complejidad entre trivial y simple, es la misma que entre simple y moderada
- cualquier medida de intervalos debe preservar estas diferencias. Una medida aceptable podría definirse como:

$$M_7(x) = \begin{cases} 0 & \texttt{trivial} \\ 2 & \texttt{simple} \\ 4 & \texttt{moderado} \\ 6 & \texttt{complejo} \\ 10 & \texttt{incomprensible} \end{cases}$$

Escala de Intervalos - transformación

- supongamos que un atributo es medible en esta escala, y M y M' son mapeos que satisfacen la condición de representación
- entonces, siempre podemos encontrar números a y b tal que:

$$M = a * M' + b$$

- este tipo de transformación se denomina transformación afin
- ejemplo: podemos transformar los grados Celsius a Fahrenheit utilizando la ecuación:

$$F = 9/5 * C + 32$$

Escala de Razones

- provee más información que la escala de intervalos
- ejemplo: se necesita saber que un proyecto consume dos veces más recursos que otro, o que un procesador es el doble de eficiente que otro
- en estos casos, se puede utilizar la escala de ratios
- la escala de razones tiene las siguientes características:
  - es un mapeo de medición que preserva el ordenamiento, el tamaño de los intervalos entre entidades, y las proporciones entre entidades
  - existe un elemento cero que representa la falta total del atributo
  - el mapeo de medición debe comenzar en cero y aumentar a intervalos iguales, conocidos como unidades

Escala de Razones

- se pueden aplicar operaciones aritméticas que tengan algún significado entre las clases en el rango del mapeo
- la diferencia fundamental con las escalas anteriores, es que en esta existe una relación empírica que captura ratios
- ejemplo: la longitud de objetos físicos es medible en una escala de ratios, permitiendo hacer enunciados que un objeto es el doble de largo que otro. Podemos medir en cm., metros, etc.
- en general, cualquier transformación posible para esta escala, es un mapeo de la forma $M = a * M'$, donde a es un escalar > 0
- el tipo de transformación se llama transformación escalar

Escala de Razones - ejemplo

- la longitud del código es medible en una escala de razones
- existe la relación empírica: “dos veces más largo”
- existe la noción del elemento cero: un código vacío
- podemos medir la longitud del código de varias maneras: *LOC*, *KLOC*
- supongamos que M es la medida en *LOC*, y M' es la medida en cantidad de caracteres
- podemos transformar M a M', mediante $M = a * M'$, donde a es el promedio de caracteres de una línea de código

Escala Absoluta

- la escala absoluta es la más restrictiva de todas
- para dos medidas M y M' existe una única transformación posible: la transformación identidad
- existe una sola forma en la que la medida se puede tomar
- la escala absoluta tiene las siguientes características:
  - la medida se hace simplemente contando el número de elementos en el conjunto de entidades
  - siempre tiene la forma: número de ocurrencias de x en la entidad
  - existe sólo una medida posible: la cuenta actual
  - todos los análisis aritméticos del resultado son significativos

Escala Absoluta - ejemplos

- número de errores observados durante la etapa de testeo
- número de personas asignadas a un proyecto
- longitud de un programa en *LOC*???
- es un error medir *LOC* con una escala absoluta - Existen muchas formas de medir: *LOC*, *KLOC*, caracteres, etc
- *LOC* es una medida de escala absoluta del atributo “número de líneas de código” de un programa

¿Qué es una métrica?

- desde el momento en que uno asocia un número a una idea, se ha aprendido algo más
- una métrica es una indicación medible de algún aspecto cuantificable de un sistema
- aspecto cuantificable de un sistema: alcance, riesgo, costo, tiempo
- características de una métrica útil:
  - medible
  - independiente
  - controlable
  - precisa

Características: medible

- medible: el indicador debe ser medible para considerarlo una métrica. Si un factor es no medible es no cuántico
- definida una métrica, se debe estimar su valor antes de observarlo
- la diferencia entre la métrica no medida y un no cuántico, es que la métrica es escalable en forma uniforme, es mejorable, y resoluble por observaciones

Características: independiente

- debe ser independiente de la influencia consciente del personal del proyecto
- es independiente en la medida en que no se pueda cambiar salvo por avances del proyecto
- ejemplo: la entrega de documentos, no es independiente
- principio de incertidumbre: el medir cualquier parámetro y asociarlo con evidencia significativa afectará la utilidad del mismo

Características: controlable

- la colección y el análisis de datos de métricas es una actividad sujeta a error
- se deben guardar los datos crudos así como pistas de auditoría o datos de control: fecha de la medición, identidad del observador, autor de la tarea,etc
- se deben analizar los datos observables en tiempo de tal manera de poder realizar correcciones en el proceso

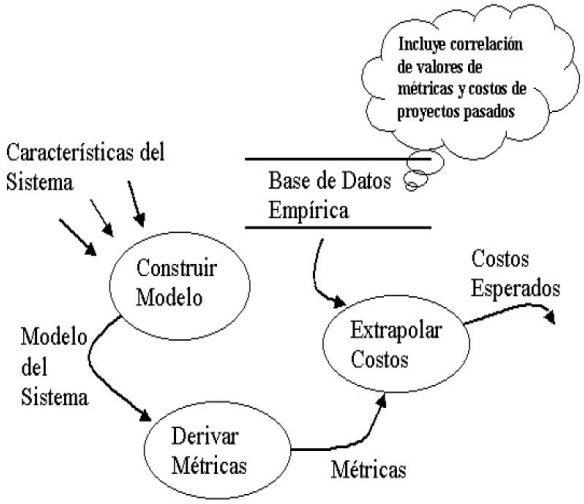
Características: precisa

- la precisión de las métricas, no debe ser maximizada, sino explícitamente señalada y registrada como parte de los datos recolectados
- la precisión puede indicarse como un rango, una tolerancia explícita o especificando el método de recolección de datos
- ejemplo: La longitud promedio de segmentos de código secuencial en el sistema XYZ es de 12.5 instrucciones
- la observación se realizó en una muestra de módulos entre 1000 y 6000 líneas de código
- los módulos incluidos pertenecen al subsistema XY1

Características: resultados y predictores

- las métricas pueden clasificarse en: resultados y predictores
- un resultado es una métrica de costo, alcance o complejidad observada de un sistema terminado
- provienen de una exhaustiva y metódica recolección de datos del proyecto
- ejemplo: costo total, total de esfuerzo (manpower)
- un predictor es un métrica señalada en forma temprana, que tiene una fuerte correlación con algún resultado posterior
- provienen de los modelos de especificación del sistema

Proceso de Proyección de Costos



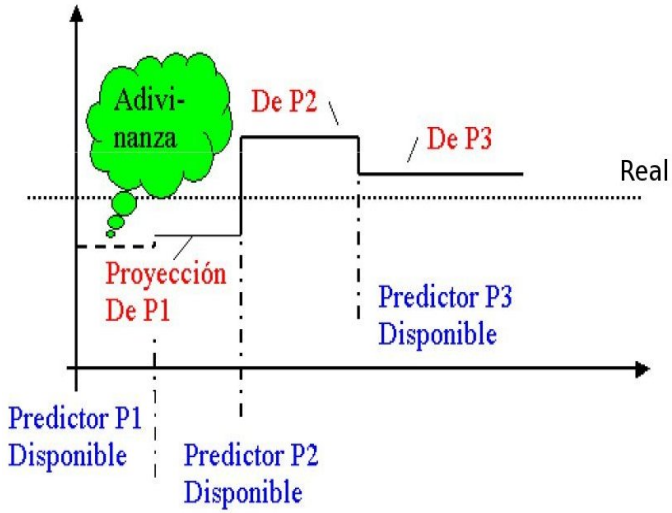
Proceso de Proyección de Costos

- el modelo se puede publicar. Pueden colaborar otras personas
- las características cuantitavivas en un modelo público se pueden extraer de manera consistente
- imposible en un modelo intuitivo
- el modelo empírico puede crecer
- los participantes pueden aportar datos
- los datos se pueden recolectar por personas ajenas al proyecto
- se puede automatizar la parte de cálculo fuera de las proyecciones de costos

Uso de Predictores para FCE

- antes de que un predictor pueda ser observado, puede ser estimado
- el recolectar nuevos predictores, provee una oportunidad para producir predicciones más exactas
- esto se muestra en el gráfico de la transparencia siguiente
- FCE Factor de calidad de la estimación

Uso de Predictores para FCE



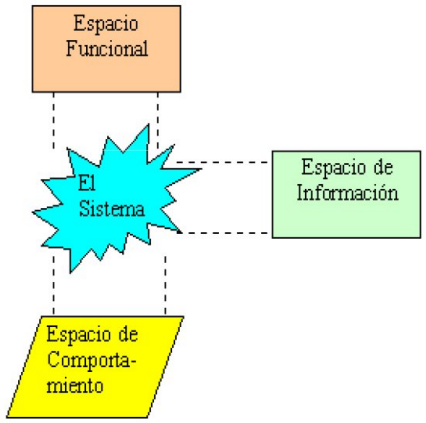
Métricas para el Presupuesto: Bang de DeMarco

- pasos a seguir:
  1. formular un sólo indicador de medida de éxito vs el objetivo, Bang per Buck (BPB)(impacto por peso)
  2. coleccionar datos en una muestra de proyectos para establecer estándares de performance de BPB
  3. buscar y evaluar predictores para aquellas partes de medida del BPB que influyen a futuro
  4. motivar al personal para mejorar el BPB. El personal debe estar informado de cómo se calcula el BPB
  5. publicar el BPB proyectado durante el proyecto, y el real luego de 6 meses de la implementación

Modelización del Problema

- un modelo consiste de una partición, junto con un registro de las interfaces entre las piezas de la partición
- se necesitan tres perspectivas para especificar la mayoría de los sistemas:
  - modelo funcional: visión particionada de lo que hace el sistema
  - modelo de datos retenidos: visión particionada de lo que el sistema recuerda
  - modelo de comportamiento: visión de los diferentes estados de comportamiento que caracterizan al sistema

Modelización del Problema



Métricas de Especificación

- la confección de un modelo formal provee tres beneficios:
  - el modelo de especificación es público. Puede ser corregido y refinado por miembros del proyecto o usuario
  - el modelo de especificación tiene características medibles que pueden ser relacionadas con performance observada
  - el modelo de especificación es terminado en forma temprana durante el proyecto, provee oportunidad para corregir las estimaciones
  - el modelo de especificación describe los requerimientos en sí mismo, no la forma de satisfacerlos
  - un análisis cuantitativo del modelo provee una medida de las funciones

Componentes Primitivas

- una componente primitiva no se descompone en componentes subordinadas. Dependiendo de lo que se particione, se obtienen distintas clases de primitivas:

Elemento	Es usado para particionar...	Primitivas resultantes
DFD	requerimientos	primitivas funcionales
DD	datos del sistema	datos elementales
diagrama de objetos	datos retenidos	objetos
diagrama de objetos	datos retenidos	relaciones
diagrama de estados	características de control	estados
diagrama de estados	características de control	transiciones

Componentes Primitivas

- contar las primitivas provee métricas básicas para medir el Bang:
  - *PF*: número de primitivas funcionales automáticas
  - *PFM*: número de primitivas funcionales manuales modificables
  - *DE*: número de datos elementales dentro del sistema automático
  - *DEI*: número de datos elementales de entrada
  - *DEO*: número de datos elementales de salida
  - *DER*: número de datos elementales retenidos

Componentes Primitivas

- contar las primitivas provee métricas básicas para medir el Bang:
  - *OB*: número de objetos retenidos
  - *RE*: número de relaciones en el modelo de datos retenido
  - *ST*: número de estados en el modelo de comportamiento
  - *TR*: número de transiciones en el modelo de comportamiento
  - TC_i: número de "data tokens" en la primitiva *i*
  - RE_i: número de relaciones que involucran al objeto *i*

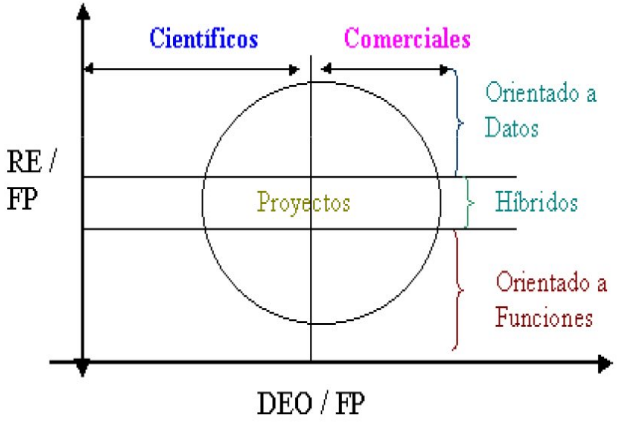
Formulación de una Teoría de Costos

- $$\text{bang tentativo} = \sum_i M_i * (\text{factor peso de } M_i)$$
- para caracterizar el Bang se elige uno de los indicadores como el principal y se usan los otros para modificarlo
- en la mayoría de los sistemas administrativos *PF* es el principal indicador
- hay sistemas altamente orientados a funciones y otros a datos, dependiendo de esto es el indicador que se deberá considerar como principal

Formulación de una Teoría de Costos

- se pueden analizar dos razones en función de *PF* (primitivas funcionales) y *RE* (relaciones entre objetos)
- si $RE/PF < 0{,}7$ es un sistema orientado a funciones
- si $RE/PF > 1{,}5$ es un sistema orientado a datos
- la proporción DEO/PF es una medida de cuánto el sistema está dedicado a cálculos o a administración de datos

Tipos de Sistemas



Particiones Uniformes

- para determinar el criterio de hasta donde se debe particionar se usa:

$$TC_{avg} = \sum TC_i / PF$$

- regla de Partición Uniforme: dejar una componente como primitiva sólo si no es posible una partición o si la nueva partición no reduce el TC_{avg}

Corrección de Indicadores

- el tamaño relativo de una función puede ser aproximado como una función de los TC_i
- el problema de cómo varía el tamaño en función de los TC_i fue estudiado por Halstead
- el tamaño de la primitiva i es proporcional a $TC_i * \log_2(TC_i)$
- existen tablas para corregir el tamaño de las PF en función de los TC_i
- entonces tenemos $PF_{corregido} = \sum_i PFC_i$

Corrección de Indicadores

- las primitivas se pueden clasificar de acuerdo a su función en:
  - separación: dividen los datos de entrada
  - merge: combinan los datos de entrada
  - dirección de datos: dirigen datos de acuerdo a una variable de control
  - actualización simple: actualiza uno o mas datos en almacenamientos
  - administración de almacenamientos: analiza datos almacenados y actúa basado en el estado de los datos

Corrección de Indicadores

- las primitivas se pueden clasificar de acuerdo a su función en:
  - edición: evalúa nuevos datos en frontera hombre-máquina
  - verificación: chequea e informa inconsistencias
  - manipulación de textos: administra textos
  - sincronización: decide cuando actuar o decide por otras
  - generación de output: formatea nuevos flujos de datos (no tabulares)
  - display: construye salidas tabulares (2 dimensiones)
  - aritméticas: realiza cálculos
  - inicialización: setea valores para datos almacenados

Corrección de Indicadores

- las primitivas se pueden clasificar de acuerdo a su función en:
  - computación: cálculos matemáticos complejos
  - administración de dispositivos: controla dispositivos
- los factores dependen del contexto:
  - tipos de sistemas
  - herramientas, lenguajes de programación
- en los sistemas orientados a datos el peso depende de los RE_i de los objetos
- existen factores de corrección en función de los RE_i

Cálculo del Bang

- en sistemas híbridos se aconseja manejar dos Bangs, el funcional y el de datos. No se puede generalizar una fórmula que los relacione
- el Bang es un indicador cuantitativo de las funciones útiles netas desde el punto de vista del usuario. Independiente de la implementacion
- objetivos de calcular el Bang:
  - se usa como un predictor fuerte y anticipado del esfuerzo
  - se usa para calcular eficiencia productiva: BPB
- se deben usar otras métricas para otras actividades, como por ejemplo conversión de la base de datos, etc.

Cálculo del Bang - sistemas orientados a funciones

1. inicializar $Bang ::= 0$
2. para cada primitiva funcional en el modelo:
   - 2.1 calcular TC_i - suma de los "data token“ de la primitiva
   - 2.2 calcular $PFC_i = Tabla\_de\_correccion(TC_i)$
   - 2.3 clasificar la primitiva en la clase que corresponda
   - 2.4 calcular $PesoCorregido_i$ en base a la tabla de corrección del peso dependiendo del tipo de primitiva
   - 2.5 $Bang ::= Bang + PFC_i * PesoCorregido_i$

Cálculo del Bang - sistemas orientados a datos

1. inicializar $Bang ::= 0$
2. para cada objeto del modelo de objetos:
   - 2.1 calcular RE_i
   - 2.2 $OBC_i ::= Tabla\_de\_correccion(RE_i)$
   - 2.3 $Bang ::= Bang + OBC_i$

Objeciones

- las mediciones de tamaño generalmente se rechazan debido a:
  - esfuerzo: no tienen en cuenta redundancia y complejidad
  - productividad: no consideran funcionalidad y esfuerzo
  - costo: no contabilizan legilibilidad y reuso

Tamaño: propiedades

- el tamaño del software puede ser descripto con tres propiedades:
  - longitud: mide tamaño físico del producto
  - funcionalidad: mide las funciones provistas por el producto
  - complejidad: puede ser interpretada de distintas maneras

Contexto de la medición

- longitud hay consenso en medir longitud de programas, pero no de especificaciones
- funcionalidad existen trabajos para medir funcionalidad de especificaciones
- complejidad se puede medir la siguiente complejidad, si bien hay pocos avances:
  - del Problema – a resolver
  - del Algoritmo – utilizado notación asintótica
  - estructural – mide la estructura del SW implementado
  - cognitiva - esfuerzo requerido para entender el SW

Longitud

- los tres productos mas importantes cuyo tamaño sería importante medir son:
  - especificaciones
  - diseño
  - código
- la medida mas comúnmente usada son las líneas de código: *LOC*
- se debe tener en cuenta: líneas en blanco, líneas de comentarios, declaración de datos y líneas que contienen varias instrucciones

Líneas de Código

- Conte, Dunsmore & Shen: cualquier línea de texto de programa que no es comentario o línea en blanco, independientemente del número de sentencias o fragmentos de sentencias en la línea
- Hewlett-Packard: una sentencia de código fuente no comentada: cualquier sentencia excepto comentarios o líneas en blanco
- $NCLOC - CLOC$: non commented line of code - commented line of code
- $ELOC$: effective line of code
- LOC: longitud total $LOC = NCLOC + CLOC$

Ejemplo LOC

C	COBOL

`#include <stdio.h>`
 	
`int main() {`
`  printf("\nHello\n");`
`}`
	`000100 IDENTIFICATION DIVISION.`
	`000200 PROGRAM-ID. HELLOWORLD.`
	`000300`
	`000400*`
	`000500 ENVIRONMENT DIVISION.`
	`000600 CONFIGURATION SECTION.`
	`000700 SOURCE-COMPUTER. RM-COBOL.`
	`000800 OBJECT-COMPUTER. RM-COBOL.`
	`000900`
	`001000 DATA DIVISION.`
	`001100 FILE SECTION.`
	`001200`
	`100000 PROCEDURE DIVISION.`
	`100100`
	`100200 MAIN-LOGIC SECTION.`
	`100300 BEGIN.`
	`100400  DISPLAY " " LINE 1 POSITION 1 ERASE EOS.`
	`100500  DISPLAY 'Hello world!' LINE 15 POSITION 10.`
	`100600  STOP RUN.`
	`100700 MAIN-LOGIC-EXIT.`
	`100800  EXIT.`

Lines of code: 4 (excluding whitespace)	Lines of code: 17 (excluding whitespace)

Líneas de Código

- sería posible distinguir entre la cantidad de código entregado (DSI: Delivered Source instructions) y la cantidad de código desarrollado
- formula de Halstead: $volumen = longitud * \log_2(vocabulario)$
- otro enfoque es medir longitud de acuerdo a:
  - número de bytes de almacenamiento requerido para el texto del programa
  - número de caracteres (CHAR) en el texto del programa

Líneas de Código

- en programación visual, entornos de ventanas, orientación a objetos lenguajes de cuarta generación cambian las nociones de tamaño
- surgen dos nuevos objetivos de medición:
  - ¿cómo se tienen en cuenta objetos no textuales?
  - ¿cómo medimos componentes construidas externamente?
- Pfleeger indica que contar objetos y métodos conduce a estimaciones más precisas

Longitud de Especificaciones y Diseño

- las especificaciones y diseños consisten de textos y diagramas
- se deben medir diferentes objetos atómicos
- por ejemplo, para diagramas de casos de uso: actores, casos de uso, relaciones de `incluye` y `extiende`, flujos, etc
- por ejemplo, para diagramas de clases: módulos, clases, relaciones de herencia y agregación, etc
- por ejemplo, para especificaciones algebraicas: clases, funciones, operaciones y axiomas
- intuitivamente se predice la longitud para tratar de relacionar la longitud de productos de etapas posteriores con la longitud de productos ya construidos

Longitud de Especificaciones y Diseño

- Proporción de Expansión $\alpha =$ tamaño código$/$tamaño diseño

$$LOC = \alpha * \sum_{i=1}^{n} S_i$$

donde S_i es el tamaño del módulo i

- Walston & Felix: sea D la documentación medida en páginas, L la longitud del programa entonces

$$D = 49L^{1,01}$$

- para estimaciones precisas, se deben recolectar datos para entornos específicos

Reuso

- mejora la productividad y la calidad. Es difícil de definir formalmente
- el grado de reuso publicado por NASA/Goddard's Software Engineering Lab
  - reuso verbatim: reusado sin cambio
  - ligeramente modificado: se reusó modificando menos del 25% *LOC*
  - extensamente modificado: se reusó modificando mas del 25% *LOC*
  - nuevo: ninguna línea proviene de un componente previo
- $$\%\ \text{reuso} = \text{líneas reusadas}/LOC$$

Funcionalidad

- tres enfoques:
  - puntos de función de Albrecht
  - COCOMO ya visto en la materia
  - peso de especificación de DeMarco, ya visto en la materia
- la idea intuitiva es que si un programa P es la implementación de la especificación S, entonces P y S deberían tener la misma funcionalidad

Puntos de Función de Albrecht

- intenta medir la cantidad de funcionalidad de un sistema, descripta en la especificación
- pasos:

1. identificar todos los:
   - entradas externas
   - salidas externas
   - consultas
   - archivos externos
   - archivos internos

Puntos de Función de Albrecht

2. asignar una complejidad subjetiva a cada ítem: `simple`, `media`, `compleja`
3. asignar un peso al item según tabla respectiva

Item	Factor de Peso		
	simple	media	compleja
entradas externas	3	4	6
salidas externas	4	5	7
consultas	3	4	6
archivos externos	7	10	15
archivos internos	5	7	10

Puntos de Función de Albrecht

4. calcular *PFNA*:

$$PFNA = \sum_{i=1}^{15} (\sharp \text{items tipo } i) * \text{Peso}_i$$

5. calcular *FCT* factor de complejidad técnico:

$$FCT = 0{,}65 + 0{,}01 * \sum_{i=1}^{14} F_i$$

donde F_i puede ser: $0, 1, 2, 3, 4, 5$, y resulta $0{,}65 \leq FCT \leq 1{,}35$

6. calcular *PFA*:

$$PFA = PFNA * FCT$$

Puntos de Función de Albrecht

- componente del *FCT*

F1	confiabilidad de backup y recuperación	F8	actualización on-line
F2	comunicación de datos	F9	interface compleja
F3	interfaces distribuidas	F10	procesamiento complejo
F4	performance	F11	reusabilidad
F5	altamente dependiente de la configuración	F12	facilidad de instalación
F6	entrada de datos on-line	F13	múltiples sites
F7	facilidad operacional	F14	facilidad de cambio

Puntos de Función de Albrecht

- los puntos de función forman una base para la estimación del esfuerzo
- Albrecht propone los puntos de función como medida de tamaño independiente de la tecnología
- problemas:
  - subjetividad en el FCT, variación del 35%
  - contar las cosas 2 veces
  - valores no intuitivos: si $F_i = 3$ entonces $FCT = 1$ NO! $FCT = 1{,}07$
  - problemas con exactitud, el FCT no mejora significativamente la estimación de recursos

Administración de Proyectos de Software
Métricas de Atributos Internos del Software
Longitud, Funcionalidad, Complejidad

Puntos de Función de Albrecht

- problemas:
  - no se puede usar anticipadamente: requiere la especificación completa
  - problemas con cambio de requerimientos - variaciones de 400% a 2000% luego de implementación
  - problemas con dominios de aplicación - funciona bien para sistemas de información administrativos, no en sistemas de tiempo real o en aplicaciones científicas
  - problemas de dependencia de tecnología - no es independiente de los métodos de análisis y diseño usados.

Administración de Proyectos de Software
Métricas de Atributos Internos del Software
Longitud, Funcionalidad, Complejidad

Complejidad del Problema y de la Solución

- los puntos de función de Albrecht miden el problema. Un problema puede tener varias soluciones de distinta complejidad
- complejidad de la Solución $\longrightarrow$ complejidad del Problema
- complejidad del problema: cantidad de recursos requeridos para una solución óptima del problema
- complejidad de la solución: cantidad de recursos necesarios para implementar una solución particular
- la complejidad, del problema o de la solución, tiene dos aspectos: tiempo y espacio

Administración de Proyectos de Software
Métricas de Atributos Internos del Software
Longitud, Funcionalidad, Complejidad

Complejidad en Tiempo y Espacio

- complejidad-tiempo: el recurso es tiempo de uso del procesador
- complejidad-espacio: el recurso es la memoria RAM
- para medir la eficiencia de una solución, como la solución está basada en un algoritmo, entonces se mide la eficiencia del algoritmo
- ejemplo:
  - búsqueda secuencial: el peor caso son n comparaciones
  - búsqueda binaria: el peor caso es $\log_2(n)$

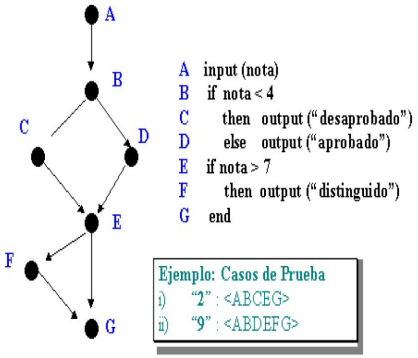
Medidas Jerárquicas para Testing

- la estructura de un módulo está relacionada con la dificultad para testearlo
- sea P un programa, S la especificación de P, e i una entrada para P. Se define caso de test $(i, S(i))$
- el interés es chequear que $S(i) = P(i)$
- las estrategias para testear software pueden ser:
  - pruebas de caja negra: los casos de test se derivan de la especificación sin referencias al código
  - pruebas de caja blanca: los casos de test se seleccionan basados en el conocimiento de la estructura interna del programa

Medidas Jerárquicas para Testing

- en estrategias de caja blanca, los casos de test se seleccionan de tal manera que toda sentencia de programa se ejecute al menos una vez - cobertura de sentencias
- en términos de grafos de programas la cobertura de sentencias se logra encontrando un conjunto de caminos de tal forma que todo nodo esté en al menos un camino

Medidas Jerárquicas para Testing

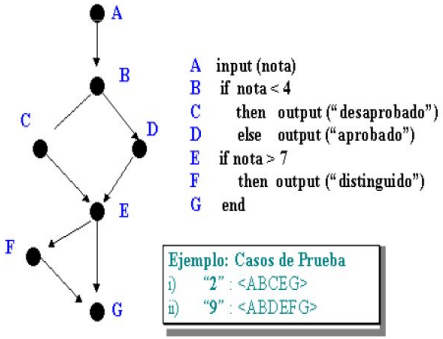


Medidas Jerárquicas para Testing

- una alternativa para tests de caja blanca es seleccionar casos de test de tal manera que cada rama (arco) sea ejecutada al menos una vez: cobertura de arcos
- la estrategia de caja blanca mas exhaustiva es seleccionar casos de test de tal forma que todo camino posible del programa sea ejecutado al menos una vez - cobertura de caminos
- es prácticamente imposible
- los caminos no factibles son un impedimento básico en las estrategias de caja blanca

Medidas Jerárquicas para Testing

- un camino no factible es un camino del programa que no puede ser ejecutado por ningún input. Ejemplo: ABCEFG



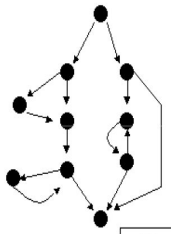
Medidas Jerárquicas para Testing

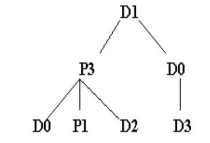
- ninguna estrategia de caja blanca puede asegurar por sí misma un adecuado testeo del software
- ejemplo: La ejecución del camino ABDEFG para "9" fue correcta. ¿Qué pasa con el input 11? Ejecuta el mismo camino y el resultado es erróneo
- el conocer todos los caminos que satisfacen una estrategia no significa conocer cómo definir los casos de test
- asociado con toda estrategia de test, existen dos medidas:
  - el mínimo número de casos de test
  - el ratio de efectividad del test

Medidas Jerárquicas para Testing

- es importante no sólo diseñar la estrategia, sino también el número mínimo de casos de test (NMCT)
- el teorema de descomposición nos permite calcular el NMCT: un caso de test corresponde a un camino a través del grafo F
- para calcular el NMCT, debemos calcular el número mínimo de caminos $m(F)$ requeridos para satisfacer la estrategia
- podemos calcular $m(F)$ a partir del árbol, conociendo $m(F)$ para los primos, la secuencia y el anidamiento

NMCT - ejemplo





F = D1 ((D0;P1;D2) , D0(D3))

m(F) = m(D1((D0;P1;D2), D0(D3))
= m((D0;P1;D2)) + m(D0(D3))
= m(D0) m(P1) m(D2) + m(D3) +1
= 2 * 1 * 2 + 2 + 1
= 7

Proporción de Efectividad del Test

- para un programa y un conjunto de casos de test, deseamos conocer en que grado los casos de prueba satisfacen una estrategia particular de test
- ejemplo: Ejecutamos con 2 casos de prueba: “6” y “9” Los casos de test cubren dos caminos: ABDEG y ABDEFG. Cubren 6 de las 7 sentencias, 7 de los 8 arcos y 2 de los 4 caminos
- decimos que:
  - cubrimiento de sentencias es 86 %
  - cubrimiento de arcos es 87,5 %
  - cubrimiento de caminos es 50 %

Efectividad del Test

- dada una estrategia T que requiere cubrir una clase de objetos (sentencias, caminos,...), para un programa dado y un conjunto de casos de prueba, se define la proporción de efectividad del test:

$$ET_T = \frac{\text{número de objetos de } T \text{ usados al menos una vez}}{\text{número total de objetos de } T}$$

- los gerentes asumen que normalmente ET es del 100 %. La experiencia dice que no supera el 40 %
- ¿se testea correctamente el software?

Medidas de Chidamber y Kemerer

- son métricas propuestas por Shyam Chidamber y Chris Kemerer (1995) para:
  - definición de objetos y relaciones entre objetos
  - análisis de atributos y propiedades de objetos
  - caracterización de la estructura de la comunicación entre objetos

Medidas de Lorenz y Kidd

- son métricas propuestas por M. Lorenz y J. Kidd (1994) , divididas en cuatro categorías:
  - tamaño (recuento de atributos y operaciones)
  - herencia (reutilización del código a lo ancho y alto de la jerarquía de clases)
  - valores internos (cohesión y análisis de código)
  - valores externos (acoplamiento y reuso)

Definición de objetos y relaciones entre objetos

1. métodos ponderados por clase WMC (weighted methods per class)
2. profundidad del árbol de herencia DIN (depth of inheritance)
3. número de descendientes NOC (number of children)

Métodos ponderados por clase (WMC)

- $WMC = \Sigma_i c_i$, donde c_i es una medida de complejidad del método i
- el número de métodos y su complejidad es un predictor de cuanto tiempo y esfuerzo es necesario para desarrollar y mantener la clase
- cuanto más métodos mayor impacto en los hijos (herencia)
- clases con más métodos son mas específicas, limitando el reuso

Profundidad del árbol de herencia (DIN)

- la longitud máxima desde el nodo hasta la raíz del árbol
- cuanto más profunda está una clase en una jerarquía, mayor número de métodos hereda, haciendo más complejo predecir su comportamiento
- una jerarquía de clases profunda lleva también a una mayor complejidad de diseño ya que involucra más clases
- por otro lado, los valores grandes de esta medida implican que se pueden reutilizar muchos métodos

Descendientes de una clase (NOC)

- definida como el número inmedidato de subclases
- a medida que crece el número de descendientes se incrementa la reutilización
- puede darse una mayor posibilidad de una incorrecta abstracción y mayor complejidad de la clase padre
- un gran número de hijos puede requerir mayor testing de los métodos de la clase
- un gran número de hijos también es un indicador de la influencia potencial de una clase en el diseño

Atributos y propiedades de objetos

1. respuesta para una clase RFC (response for a class)
2. falta de cohesión de los métodos LCO (lack of cohesion)

Respuestas para una clase (RFC)

- es el número de métodos que pueden ser invocados en respuesta a un mensaje enviado a un objeto de la clase
- un valor muy alto indica que la clase es compleja y probablemente altamente acoplada
- aumenta el esfuerzo de testeo y mantenimiento
- puede surgir el interrogante de si la clase está modelada correctamente

Falta de cohesión de los métodos (LCO)

- el número de pares de métodos cuya similitud es cero menos el número de pares de métodos cuya similitud es distinta de cero
- si el valor es negativo, se asume cero
- similitud si dos pares de métodos acceden a uno o más de los mismos atributos
- la cohesión de los métodos dentro de una clase es deseable ya que promueve el encapsulamiento
- la falta de cohesión implica que una clase debiera dividirse en dos o más clases

Comunicación entre objetos

1. respuesta para una clase RFC (response for a class)
2. acoplamiento entre clases CBO (coupling between objects)

Acoplamiento entre clases (CBO)

- la cantidad de clases con las cuales una clase está acoplada
- una clase está acoplada con otra si usa métodos o variables de instancia de la otra
- un valor alto disminuye el diseño modular y dificulta el reuso
- el acoplamiento debe mantenerse mínimo para mejorar modularidad y encapsulamiento
- una medida de acoplamiento es útil para determinar cuanto de complejo será el diseño de testing
- cuanto más acoplamiento presenta el diseño más riguroso debe ser el testing

Medidas de tamaño

- número de operaciones (heredadas + propios)
- número de atributos (heredados + propios)
- número de operaciones invalidadas por una subclase (*NOI*) la invalidación se da cuando una subclase substituye una operación heredada por una versión especializada para su propio uso
- número de operaciones agregadas por una subclase (*NOA*)

Índice de especialización

- las subclases se especializan agregando atributos y operaciones privadas
- el índice de especialización (*IE*) se define

$$IE = (NOI * nivel)/M_{total}$$

donde *NOI* es el número de operaciones invalidadas por la subclase, *nivel* es el nivel de la jerarquía de clases donde reside la clase, y M_{total} es el número total de métodos para la clase

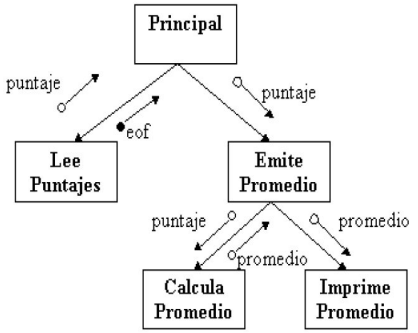
Medidas para métodos

- tamaño medio de operación en una clase (TO_{avg}), puede ser medido en *LOC* o en cantidad de mensajes enviados por la operación
- complejidad de operación (*CO*) se calcula mediante cualquier métrica de complejidad en el código del método
- número medio de parámetros por operación (NP_{avg})

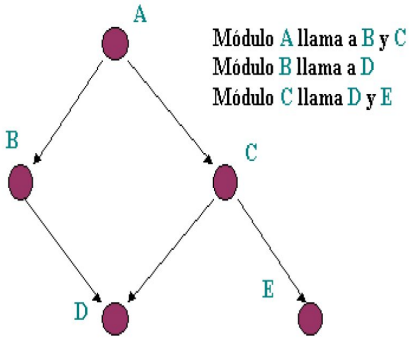
Modularidad y Flujos de Información

- se examinaron atributos de módulos individuales: medidas intramodulares
- ahora estudiaremos relaciones entre módulos: medidas inter-modulares
- módulo: secuencia contigua de sentencias de programa, acotadas por elementos de contorno y que tienen un identificador agregado (Yourdon y Constantine, 1979)
- para describir los atributos inter-modulares, construimos modelos para capturar la información necesaria sobre las relaciones entre módulos
- ejemplo: carta estructurada. grafo de dependencias de módulos

Acoplamiento entre Módulos



Acoplamiento entre Módulos



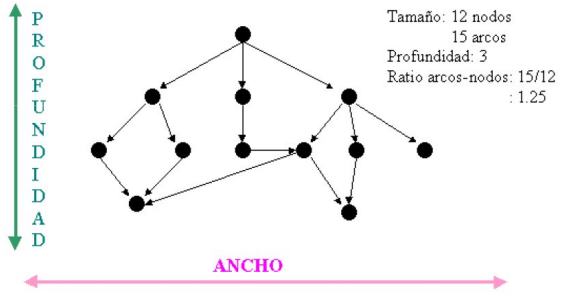
Modularidad

- existen distintos enfoques:
  - longitud promedio de los módulos (Boehm)
  - $M_1 = \frac{\text{módulos}}{\text{procedimientos}}$ (Hausen)
  - $M_2 = \frac{\text{módulos}}{\text{variables}}$
- ambos sugieren focalizar primero en aspectos específicos de modularidad
- morfología es la “forma” general de la estructura del sistema
- las características son profunidad y ancho, y se usan para diferenciar buenos de malos diseños

Características Morfológicas de los Módulos

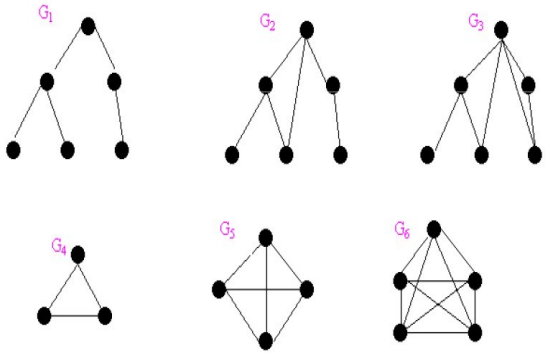
- muchas características morfológicas son medibles directamente, incluyendo:
  - tamaño: medido como el número de nodos, arcos, o combinación de ambos
  - profundidad: medido como la longitud del camino más largo desde el nodo raiz a un nodo hoja
  - ancho: medido como el máximo número de nodos en cualquier nivel
  - proporción arco-nodo: puede considerarse una medida de densidad de conectividad, ya que aumenta a medida que agregamos más conexiones entre nodos

Características Morfológicas de los Módulos



Flujo de Control entre Módulos

Diferentes estructuras de sistemas que se pueden encontrar en diseños típicos.



Impureza de Árbol

- decimos que un grafo es conexo si para cada par de nodos en el grafo, existe un camino entre ambos
- un grafo es completo si siempre dos nodos cualesquiera están conectados por un sólo arco. Tiene entonces $n(n-1)/2$ arcos
- los grafos G_4, G_5 y G_6 son grafos completos
- el grafo $G-1$ es llamado un árbol, ya que es un grafo conexo sin ciclos (un camino que empiece y termine en el mismo nodo)
- Ince y Hekmatpour: “Cuanto mas se desvíe un sistema de ser una estructura pura de árbol para ser una estructura de grafo, peor es el diseño...”

Impureza de Árbol

- propiedades de grafos y árboles:
  - un árbol con n nodos siempre tiene $n-1$ arcos
  - para cada grafo conexo G, podemos encontrar al menos un subgrafo que es un árbol construído exactamente sobre los mismos nodos de G (árbol de cubrimiento)
  - el árbol de cubrimiento G' de un grafo G es construído sobre los mismos nodos que G, pero con un mínimo subconjunto de arcos de tal manera que siempre dos nodos de G' están conectados por un camino
- intuitivamente la impureza de G aumenta a medida que aumenta la diferencia entre G y G'

Impureza de Árbol (I)

- formalmente la medida de impureza de árbol debe satisfacer las siguientes propiedades:
  - $m(G) = 0, \forall G$ es un árbol
  - un grafo que es un árbol no tiene impureza
  - $m(G) > m(G')$ si G difiere de G' sólo en el agregado de un arco (representando una llamada a un procedimiento existente). Ejemplo: $m(G_3) > m(G_2)$

Impureza de Árbol (II)

- formalmente la medida de impureza de árbol debe satisfacer las siguientes propiedades:
  - sea a, a' el número de arcos en G, G', y n, n' el número de nodos en G, G'. Si $N > N'$ y $A - N + 1 = A' - N' + 1$, es decir el árbol de cubrimiento de G tiene más arcos que el de G', pero en ambos casos el número de arcos adicionales al árbol de cubrimiento es el mismo, entonces $m(G) < m(G')$
  - $\forall G, m(G) <= m(K_n) = 1$ donde n es el número de nodos de G y K_n es el grafo completo de n nodos. Esto significa que para todos los grafos con n nodos, el grafo completo tiene máxima impureza (1)

Impureza de Árbol (III)

- definimos medidas de impureza:

$$m(G) = \frac{\text{número de arcos más que el árbol de cubrimiento}}{\text{máximo número de arcos más que el árbol de cubrimiento}}$$

- en un grafo completo, el número de arcos es: $a = n(n-1)/2$
- la cantidad de nodos en el árbol de cubrimiento es siempre $n-1$
- el divisor es:

$$n(n-1)/2 - (n-1) = (n-1)(n/2-1) = (n-1)(n-2)/2$$

Impureza de Árbol (IV)

- el cociente es: arcos totales a - arcos del árbol abarcado $n-1$

$$a-(n-1)=a-n+1$$

- reemplazando en la fórmula:

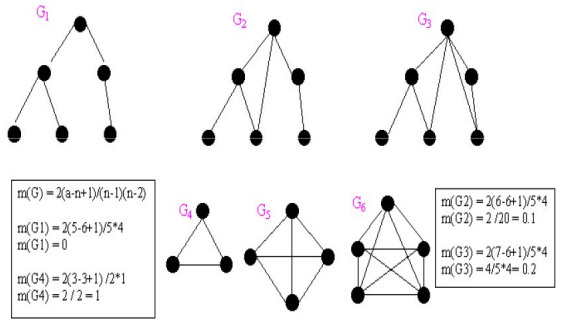
$$a-n+1/((n-1)(n-2)/2)$$

$$2(a-n+1)/(n-1)(n-2)$$

- medida de impureza: $m(G)=2(a-n+1)/(n-1)(n-2)$

Impureza de Árbol - ejemplos

- diferentes estructuras de sistemas que se pueden encontrar en diseños típicos

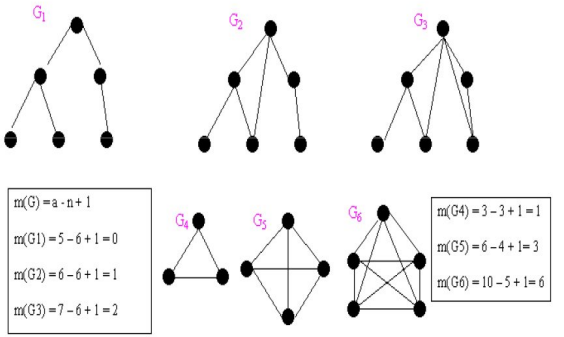


Reuso

- el reuso mide la proporción del sistema construído fuera del proyecto.
- el reuso interno mide el grado en que los módulos son usados dentro del mismo producto
- los grafos de llamada no proveen esa información
- Yin y Winchester propusieron como medida de diseño de sistema: $r(G)=a-n+1$
- no tiene en cuenta distintas llamadas ni tamaño de componentes
- ejemplos: $r(G_1)=0$, $r(G_2)=1$, $r(G_3)=2$, $r(G_4)=1$, $r(G_5)=3$, $r(G_6)=6$

Reuso - ejemplos

- diferentes estructuras de sistemas que se pueden encontrar en diseños típicos



Acoplamiento

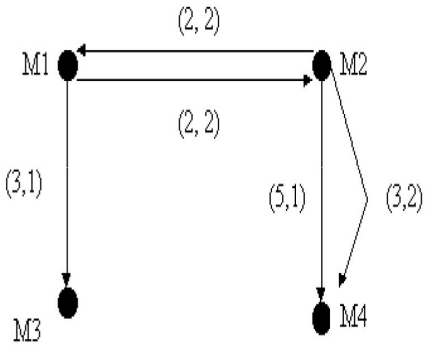
- dados dos módulos x e y, se puede clasificar distintos tipos de acoplamiento
- R_0 no existe relación de acoplamiento: totalmente independientes
- R_1 acoplamiento de datos: se comunican por parámetros (no datos de control). Necesaria para comunicación entre módulos
- R_2 acoplamiento de molde: aceptan el mismo tipo de registro como parámetro
- R_3 acoplamiento de control: x pasa parámetro a y para controlar su comportamiento (dato de control)
- R_4 acoplamiento común: se refieren a los mismos datos globales
- R_5 acoplamiento de contenido: x se refiere al interior de y (x ejecuta, cambia datos o altera sentencias de y)

Acoplamiento

- $R_i > R_j$ si $i > j$
- para medir el acoplamiento se usa un grafo dirigido:
  - los nodos representan los módulos
  - los arcos representan acoplamiento entre módulos. Puede haber más de un arco entre dos nodos.
- cada arco tiene un rótulo $< i, j >$ donde i es la relación de acoplamiento R_i que ocurre entre los extremos, y j es el número de veces que ocurre el acoplamiento

Acoplamiento - ejemplo

- diferentes estructuras de sistemas que se pueden encontrar en diseños típicos



Acoplamiento

- Fenton y Melton proponen medir el acoplamiento como

$$c(x, y) = i + n/(n+1)$$

donde i es la peor relación R_i de acoplamiento entre x e y y n es el número de interconexiones entre x e y

- se puede definir C, la medida de acoplamiento general de un sistema:

$$C(S) = valormedioC(Di, Dj)$$

Cohesión

- la cohesión es grado en el cual las componentes individuales de cada módulo se necesitan para realizar la misma tarea
- se puden definir distintos niveles de cohesión:
  - coincidental: más de una función no relacionadas
  - lógica: más de una función relacionadas lógicamente
  - temporal: más de una función relacionadas por el tiempo
  - procedimiental: más de una función relacionadas por un procedimiento
  - comunicacional: más de una función sobre los mismos datos
  - secuencial: más de una función ejecutadas en orden secuencial
  - funcional: una sola función bien definida

Cohesión

- un módulo puede tener más de un tipo de cohesión. Se lo clasifica por el peor de todos
- Macro y Buxton: extendieron el concepto de cohesión agregando cohesión abstracta
- cohesión abstracta: módulo que encapsula un tipo de dato abstracto. Ejemplo: módulo que administra una pila.
- proporción de cohesión es $\frac{\text{número de módulos funcionales}}{\text{número total de módulos}}$

Flujo de Información

- atributo inter-modular: nivel total de flujo de información a través del sistema, donde los módulos son componentes atómicos
- atributo intra-modular: nivel total de flujo de información entre módulos individuales y el resto del sistema
- enfoque de Henry y Kafura dentro de la segunda línea

Flujo de Información

- flujo directo local: x invoca a y y le pasa información o y devuelve un resultado a x
- flujo indirecto local: y devuelve un dato a x que luego x lo pasa a z
- flujo global: información de un módulo a otro vía estructura de datos global
- fan in del módulo x: número de flujos locales que terminan en x, más el número de estructuras de datos leídas por x
- fan out del módulo x: número de flujos locales que salen de x, más el número de estructuras de datos actualizadas por x
- complejidad flujo informacion M:

$$M = \text{Longitud}(M) * (\text{fanIn}(M) * \text{fanOut}(M))^2$$

Flujo de Información - ejemplo

módulo	fan in	fan out	longitud	$(\text{fanIn} * \text{fanOut})^2$	CFI
A	2	2	10	16	160
B	2	1	10	4	40
C	3	2	10	36	360
D	1	0	10	0	0

Flujo de Información

- Shepperd cuestionó este enfoque debido a:
  - se hace distinción entre flujos locales y globales
  - se deberían ignorar flujos duplicados
  - se debe descartar la longitud del módulo. Es un atributo separado
- $\text{complejidadShepperd}(M) = (\text{fanIn}(M) * \text{fanOut}(M))^2$
- sus estudios demostraron que el nivel de flujo de información está íntimamente relacionado con el tiempo de desarrollo

Estructuras de Datos

- hay pocos enfoques para tratar de medir datos y su estructura
- localmente se desea medir la cantidad de estructura en cada ítem de dato
- Elliott sugiere usar teoría de grafos como se hizo con la estructura de control
- considera tipos de datos simples (enteros, caracteres, booleanos) como primos y luego considera las distintas operaciones que permiten construir estructuras más complejas

Estructuras de Datos

- globalmente se desea medir la cantidad de datos para un sistema
- se puede usar lo propuesto por Halstead:

  $\mu_2 = \text{número de variables} + \text{número de constantes únicas} + \text{número de etiqueta}$

  $N_2 = \text{número total de ocurrencias de operandos}$
- Bohem definió:

$$D/P = \frac{\text{tamaño de la base de datos en bytes}}{\text{tamaño de programa en DSI}}$$

Medición de Atributos Externos

- el principal objetivo de IS es mejorar la calidad de productos de software
- ¿qué significa calidad?
  - adecuación al objetivo
  - concordancia con la especificación
  - grado de excelencia
  - puntualidad
- los atributo externo son medidos solamente con respecto a cómo el producto se relaciona con su entorno. Ejemplo: confiabilidad. usabilidad

Medición de Atributos Externos

- muchos miden y analizan atributos internos por que son predictores de atributos externos
- ventajas de medir atributos internos:
  - están disponibles con anterioridad. Los externos están disponibles cuando el producto está completo
  - son más fáciles de medir que los externos

Modelo de Calidad de McCall

- focalizan en el producto final (generalmente código ejecutable)
- identifican atributos claves de calidad desde el punto de vista del usuario. Se los conoce como factores de calidad
- generalmente son atributos externos de alto nivel: confiabilidad, usabilidad, mantenibilidad, ...
- pero también incluyen atributos internos: testeabilidad, eficiencia, ...
- los factores de calidad se descomponen en atributos de menor nivel: criterios de calidad
- a los criterios de calidad se les asocia un conjunto de atributos de bajo nivel medibles directamente: métricas de calidad

Modelo de Calidad de McCall

- Uso $\longrightarrow$ Factores $\longrightarrow$ Criterios $\longrightarrow$ Métricas y Medidas
- Operación del producto: 5 factores, 16 criterios, muchas métricas
- Transición del producto: 3 factores, 12 criterios, muchas métricas
- Revisión del producto: 3 factores, 9 criterios, muchas métricas

Modelo de Calidad de McCall

Uso	Factor	Criterio
Operación	Correctitud	trazabilidad
		completitud
		consistencia
	Confiabilidad	tolerancia a errores
		concisidad
		simplicidad
		precisión
	Eficiencia	en tiempo
		en espacio

Modelo de Calidad de McCall

Uso	Factor	Criterio
Operación	Integridad	control de acceso
		auditoría de acceso
		integr. de datos
	Usabilidad	operabilidad
		entrenamiento
		comunicación
		volumen y de E/S
		interop. en datos

Modelo de Calidad de McCall

Uso	Factor	Criterio
Transición	Portabilidad	auto-descripción
		modularidad
		independencia de la máquina
		independencia del sistema operativo
	Interoperabilidad	modularidad
		interop. en comunicación
		interop. en datos

Modelo de Calidad de McCall

Uso	Factor	Criterio
Transición	Reusabilidad	generalidad
		modularidad
		auto-descripción
		indep. de la máquina
		indep. del sistema operativo
Revisión	Testeabilidad	simplicidad
		instrumentación

Modelo de Calidad de McCall

Uso	Factor	Criterio
Revisión	Mantenibilidad	consistencia
		simplicidad
		concisidad
		auto-descripción
		modularidad
	Flexibilidad	expandibilidad
		generalidad
		auto-descripción
		modularidad

Monitoreo de Calidad de Software

- enfoque de Modelo Fijo: asume que todos los factores de calidad necesarios para monitorear un proyecto son un subconjunto de los publicados en el modelo
- se asume estrictamente lo publicado en el modelo
- enfoque de Modelo Particular: asume la filosofía general que la calidad está compuesta por varios atributos, pero no se asume ningún modelo
- se logra consenso con el usuario para determinar los atributos de calidad importantes para el producto
- se decide una descomposición (criterios) y relaciones entre ellos. Se miden atributos de calidad objetivamente para ver si se alcanzan los valores deseados

Monitoreo de Calidad de Software

- ejemplo: modelo fijo de McCall
- incluye 41 métricas para medir 23 criterios de calidad generados a partir de factores de calidad
- medir cualquier factor requiere considerar una lista de condiciones que pueden aplicarse a requerimientos (R), diseño (D) o implementación (I)
- la condición se responde con sí o no dependiendo si se satisface o no
- ejemplo: medir el criterio completitud para el factor correctitud. La lista de condiciones es la siguiente

Monitoreo de Calidad de Software - ejemplo completitud

1. referencias no ambiguas (input, funciones y output) [R,D,I]
2. todas las referencias de datos definidas, calculadas y obtenidas de fuente externa [R,D,I]
3. todas las funciones definidas usadas [R,D,I]
4. todas las funciones referenciadas definidas [R,D,I]
5. todas las condiciones y procesamiento definidos para cada punto de decisión [R, D,I]
6. todos los parámetros de secuencia de llamados referenciados y definidos [D,I]
7. todos los informes de problemas resueltos [R,D,I]
8. el diseño coincide con los requerimientos [D]
9. el código coincide con el diseño [I]

Monitoreo de Calidad de Software - ejemplo completitud

- existen 6 condiciones aplicables a requerimientos, 8 a diseño y 8 a implementación
- asignar 1 a cada respuesta sí y 0 a cada respuesta no
- métrica de completitud = $1/3 * ((\sharp \text{sí de R})/6 + (\sharp \text{sí de D})/8 + (\sharp \text{sí de I})/8)$
- la correctitud se divide en completitud, trazabilidad y consistencia
- métrica de correctitud = $1/3 * (x + y + z)$ en este caso se pesan todos por igual. Esto es a elección

Definición de Modelos Propios

- método de Gilb: diseñar por objetivos medibles. Complementa su filosofía de desarrollos evolutivos
- el Ingeniero de Software entrega el producto de manera incremental al usuario basado en la importancia de clases de funcionalidad provista
- para asignar prioridades el usuario identifica atributos críticos
- los atributos críticos se describen en términos medibles

Definición de Modelos Propios

- objetivos de calidad
  - disponibilidad: % de tiempo de funcionamiento del sistema. Mejor Caso: 99 % Peor caso: 95 %
  - facilidad de uso: días de los empleados para aprender el uso. Mejor Caso: 5 Peor caso: 10

Modelo ISO 9126

- evaluación de un Producto de Software: Características de Calidad y Guía para su Uso, ISO 9126
- la calidad de software se define como la totalidad de rasgos y características de un producto de software que tiene la habilidad de satisfacer las necesidades enunciadas
- la calidad es descompuesta en 6 factores: funcionalidad, confiabilidad, eficiencia, usabilidad, mantenibilidad, portabilidad
- por ejemplo la confiabilidad son los atributos que permiten al sofware mantener su nivel de performance bajo condiciones enunciadas por un período de tiempo

Modeo ISO 9126

- la portabilidad son los atributos que permiten que el software sea transferido de un entorno a otro
- muchos ingenieros se basan en medidas definidas para propósitos específicos distintos de los modelos de calidad formal
- ejemplo:

$$\text{portabilidad} = 1 - \frac{\text{recursos para mover el sistema}}{\text{recursos para hacer el sistema}}$$

- estos enfoques son subjetivos. Los métodos formales también requieren respuestas subjetivas

Medidas de Calidad basadas en Defectos

- la correcta implementación de atributos de calidad requiere recursos extra. No siempre están disponibles
- muchos piensan que calidad es considerar un producto libre de defectos
- defecto: error conocido, falla, falta
- una medida de calidad de software es la densidad de defectos

Densidad de Defectos

- podemos clasificar defectos en:
  - defectos conocidos: descubiertos por testing, inspecciones u otras técnicas
  - defectos latentes: presentes en el sistema, aún no conocidos
  - densidad de defectos: es la proporción entre el número de defectos conocidos y el tamaño del producto
- el tamaño del producto generalmente se mide en LOC, también podría tomarse por puntos de función

Densidad de Defectos

- para implementarla se debe recordar que no hay consenso en que se considera un defecto:
  - fracasos pre-release
  - fallas residuales (descubiertas luego del release)
  - todas las fallas conocidas
  - todas las fallas descubiertas luego de algún punto del ciclo de vida (testing unitario)
- separar la proporción de defectos (implica tiempo) de densidad de defectos

Densidad de Defectos

- usar siempre la misma unidad para medir tamaño (LOC, LOCNC, DSI...)
- la métrica debe medir la calidad del software y no el procedimiento para detectar y reportar errores
- si se usa para predecir el comportamiento del sistema (estimando defectos residuales), se debe prestar atención a:
  - es difícil determinar la seriedad de una falla
  - no todos los usuarios usan el sistema de la misma manera, ni de la manera deseada
- existen productos con gran número de defectos y que fallan muy eventualmente. Estos productos son de alta calidad pero tienen alta densidad de defectos

Evidencias reportadas (spoilage)

- las empresas no las publican. Son publicadas por terceras partes y de una manera que no es posible validarlas
- EEUU y Europa: 5 a 10 defectos por KLOC (nro.de defectos post entrega y en los primeros 12 meses)
- empresas americanas 4,44 d/KLOC, japonesas 1,96 d/KLOC
- medida Japonesa:

$$\text{spoilage} = \frac{\text{tiempo para corregir defectos post-release}}{\text{tiempo total de desarrollo}}$$

Medidas de Usabilidad - Boehm

- mide cómo el usuario va a interactuar con el sistema
- juega un papel importante en la satisfacción del cliente, funcionalidad adicional y costos del ciclo de vida
- según Boehm la usabilidad de un producto de software es el grado en el cual el producto es conveniente y práctico de usar
- muchas veces se lo identifica con el atributo de amigable al usuario
- es difícil de medir. Se buscan características internas: manuales bien estructurados, buen uso de menúes y gráficos, mensajes de error informativos, funciones de ayuda, interfaces consistentes, etc

Visión Externa de la Usabilidad - Gilb

- según Gilb, hay que descomponerla en atributos mas detallados:
  - nivel de entrada: medir el atributo en términos de experiencia con clases de aplicaciones similares (procesador de texto, planilla de cálculo,...) o edad (programas educativos)
  - aprendizaje: medido en términos de rapidez de aprendizaje. Cantidad de horas de entrenamiento necesarias para un uso independiente
  - habilidad de manejo: velocidad de trabajo luego del entrenamiento. Errores cometidos trabajando a velocidad normal
- la visión de usabilidad es el esfuerzo requerido para aprender y operar el sistema

Atributos Internos que afectan Usabilidad

- pantallas de ayuda, opciones de menú
- legibilidad, comprensibilidad del texto
- nuevamente se miden atributos internos como atributos externos

Medidas de Mantenibilidad

- mantenible: fácil de entender, modificar, corregir
- según Ghezzi se puede clasificar en:
  - mantenimiento correctivo: corrige una falla. Encontrar y corregir fallas
  - mantenimiento adaptativo: el producto se adapta para preservar funcionalidad y performance
  - mantenimiento preventivo: se descubren fallas antes que el usuario las vea
  - mantenimiento perfectivo: cambia, reescribe para mejorar calidad. También para agregar funcionalidad

Medidas de Mantenibilidad

- no es solamente restringido al código: especificaciones, documentos de diseño, etc
- existen dos visiones para medir mantenibilidad:
  - reflejando atributos externos: depende del producto y también de la persona para medir si el proceso es efectivo
  - reflejando atributos internos: indentificando atributos internos del producto relevantes

Visión Externa de la Mantenibilidad

- a efectos de métrica se ve la necesidad de hacer un cambio sin importar la intención. Una vez que el cambio es identificado, se mide la velocidad de implementar el cambio: medida de mantenibilidad
- MTTR (Mean Time To Repair) es el tiempo medio para implementar un cambio y restablecer el sistema
- para medirlo hay que recolectar:
  - tiempo de reconocimiento del problema
  - tiempo de demora administrativa
  - tiempo de colección de herramientas de mantenimiento
  - tiempo de análisis del problema
  - tiempo de cambio de especificación
  - tiempo de cambio (testeo, revisión)

Visión Externa de la Mantenibilidad

- tambien es importante medir:
  - la proporción entre el total de tiempo de implementación de cambios y el número total de cambios implementados
  - número de problemas no resueltos
  - porcentaje de cambio que introducen nuevas fallas
  - número de módulos modificados por cambio

Atributos Internos que afectan la Mantenibilidad

- se relacionan con la complejidad relativa a niveles de esfuerzo de mantenimiento
- se trabaja relacionando el número de complejidad ciclomática y el esfuerzo. No trabajar con módulos con dicho número mayor que 10
- en algunos productos la legibilidad es indicador de mantenibilidad. Los atributos internos que determinan la estructura de los documentos son considerados importantes para la legibilidad
- medida de Gunning Indice Fog:

$$F = 0{,}4 * \sharp\text{palabras}/\sharp\text{oraciones} + \%\text{palabras de 3 o más sílabas}$$

Medición de recursos

- los desarrolladores de software a menudo son grupos heterogéneos
- es desable tratar de comprender cómo mejorar nuestro aporte personal para mejorar la calidad del software
- entonces hay que definir cómo medir productividad y cómo la productividad es afectada por el grupo y las herramientas
- se piensa el proceso de producción de software como otro proceso productivo: si se agregan líneas de ensamblado o personal se termina en tiempo

Significado de productividad

- productividad: producción de un conjunto de componentes en un período de tiempo, se desea maximizar las componentes construidas para una duración dada
- productividad: la proporción de salida por entrada, usado especialmente en mediciones de aumento de capital y en ensamblar el uso efectivo de tareas, materiales y equipamiento
- intuitivamente la idea de producción involucra el contraste entre la entrada a un proceso y la salida
- aumentando la entrada o mejorando el proceso para la misma entrada, se debería aumentar la salida

Significado de productividad

- en Ingeniería de Software hay que definir:
  - ¿en qué constituye la entrada?
  - ¿cómo el proceso afecta la relación entrad-salida?
- se relaciona el tamaño del producto con el esfuerzo requerido

$$\text{productividad} = \text{tamaño}/\text{esfuerzo}$$

$$\text{productividad} = LOC/\text{personas-mes}$$

- se presentan dificultades para medir esfuerzo:
  - un día de trabajo: 8, 12 o 16hs?
  - algunos días más productivos que otros
  - ¿es igual dos personas medio día que una persona todo el día?

Significado de Productividad

- considerando el tamaño de la salida no se tiene en cuenta su valor. Se debería medir en función del beneficio entregado
- construir software no es como producir autos
- se debe distinguir entre productividad del proceso y productividad de los recursos
- se debe ser cuidadoso para monitorear y medir. Las personas pueden entregar resultados de baja calidad. Se debe seguir un enfoque guiado por objetivos y claramente mostrar beneficios

¿Productividad de qué?

- existen otros recursos que se podrían medir por productividad:

atributo	compilador 1	compilador 2
tiempo de compilación	3.15	22.41
tiempo de compilación y linkeo	6.55	22.49
tiempo de ejecución	6.59	10.11
tamaño de código objeto	239	249
tamaño de ejecución	5748	7136
precio	$ 100	$ 450

¿Productividad de qué?

- aún en este ejemplo, se pueden confundir los atributos del producto o del recurso con los del proceso:
  - se puede evaluar la productividad del compilador (un producto o recurso)
  - ¿qué debe tener en cuenta el proceso de compilación
- pero es incorrecto hablar de la productividad del proceso de compilación
- la ecuación de productividad no debería definirse y usarse como única medidad de productividad de personal

Problemas para Medir Productividad

- ejemplo de procesos y productos que podrían considerarse al medir ciertos recursos típicos:

recurso	proceso	producto	herramientas
programador	codificación	programa	compilador
grupo programación	codificación	programas	compilador y LAN
tester	testing	programa	debugger
diseñador	desarrollo	diseño detallado	herramientas CASE
programador	mantenimiento	programas y documentación	herramienta de ing. reversa

Administración de Proyectos de Software
Métricas de Atributos Externos del Software
Productividad

Problemas para Medir Productividad

- ejemplo: un programador tarda 10 días para producir un programa P, de 5000 líneas de código. Productividad = 500 LOC por día
- el programador crea P': una copia de P que nunca se usa. P'tiene 10.000 líneas de código con una funcionalidad equivalente a P. Productividad = 1000 LOC por día
- ¿realmente duplicó la productividad? ¿Qué pasa si elimina código?

Administración de Proyectos de Software
Métricas de Atributos Externos del Software
Productividad

Problemas para Medir Productividad

- se debe resolver el problema de variantes en la definición de líneas de código
- variaciones entre una línea de código de un lenguaje comparado con otro lenguaje
- una posible solución es considerar puntos de función de Albrecht:

$$\text{productividad} = \sharp\text{puntos de función implementados}/\text{personas-mes}$$

Administración de Proyectos de Software
Métricas de Atributos Externos del Software
Productividad

Problemas para Medir Productividad

- ventajas de usar Puntos de Función:
  - refleja mejor el valor de las salidas
  - se pueden medir distintas etapas, no sólo la construcción
  - se puede medir progreso, puntos de función terminados vs no terminados

Problemas para Medir Productividad

- desventajas:
  - algunos gerentes desconfían de los puntos de función. No es medida directa
  - Albrecht presenta tabla de conversión entre puntos de función y LOC para distintos lenguajes
  - dificultad par calcularlo
- una alternativa es usar el Bang de De Marco que se puede extraer directamente de algunas herramientas CASE

Mediciones de Productividad

- tanto las medidas de longitud como las de funcionalidad no capturan la calidad y utilidad del software
- se podría medir: errores introducidos por el programador, mantenibilidad
- ejemplo: productividad de diseñadores

diseñador	productividad	% reuso	acopl.	cohesión	impureza
A	25 pág., 350 mód., 200 PF	20 %	3.1	0,2	0.8
B	15 pág., 250 mód., 150 PF	10 %	1.4	0.9	0.3

Mediciones de Productividad

- para programadores, diseñadores y analistas la naturaleza de la salida es clara
- ¿qué pasa con los gerentes, personal de calidad, personal del grupo de métricas?
- supongamos dos testeadores, ¿se mide la cantidad de LOC que testean al mes?
- no dice nada acerca de la calidad del test
- ¿se mide cantidad de defectos encontrados? No siempre implican mejor confiabilidad del producto

Equipo de Trabajo

- la estructura del equipo es un factor fundamental en la productividad del mismo: equipos con estructuras complicadas presentan poca productividad y producen productos de baja calidad
- hay pocas evidencias publicadas entre el proceso actual de estructura de equipos y productividad o calidad
- un factor importante es la comunicación entre los miembros
- complejidad de comunicación: es la complejidad causada por el número de individuos involucrados o los métodos de comunicación requeridos entre los miembros del equipo

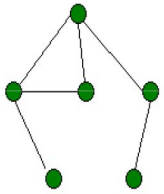
Estructuras de Software y Equipo de Trabajo

- Rook hace una analogía entre las estructuras de software y las estructuras de equipos de trabajo
- pequeño y fácil de entender ⟶ pequeños y fáciles de controlar
- bajo acoplamiento ⟶ asignar tareas de tal manera de minimizar comunicación innecesaria entre equipos
- alta cohesión ⟶ asignar tareas altamente cohesivas
- el alcance efecto es un subconjunto del alcance de control ⟶ agrupados bajo un líder y que las decisiones queden encapsuladas
- estructuras jerárquicas y niveles de decisión / conexiones patológicas ⟶ idem

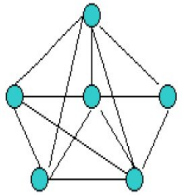
Estructura de Equipos

- se epresenta la estructura de comunicación del equipo como un grafo
- tamaño: número de nodos
- densidad de comunicación: ratio arco-nodos
- nivel de comunicación: impureza de árbol
- nivel de comunicación individual: fan-in + fan-out

Estructura de Equipos - ejemplos



tamaño	6
densidad com.	1
nivel com.	0.1
prom. nivel com. ind.	2.00



tamaño	6
densidad com.	2.17
nivel com.	0.8
prom. nivel com. ind.	4.33

Experiencia de Personal

- la experiencia puede considerarse un elemento clave para la productividad
- se deben distinguir varias categorías: experiencia del personal, experiencia del grupo, con el tipo de proyecto, herramientas, entorno, métodos, lenguages, etc
- posibles valores:
  - sin experiencia previa
  - familiaridad (teórica sin práctica)
  - experiencia práctica hasta 20 horas
  - experiencia práctica entre 21 y 100 horas
  - experiencia avanzada

Experiencia de Personal

- se calcula experiencia individual asignando peso y luego la media del equipo
- COCOMO evalúa la experincia del personal en la aplicación (baja: 4 a 12 meses, alta: aprox. 6 años) y en el lenguaje de programación
- se deben buscar elementos de motivación de personal para aumentar el entusiasmo y la productividad
- los sociólogos estudian los atributos del personal, tanto individuales como de equipo, y sus efectos en la productividad y productos tales como edad, nivel y tipo de educación, inteligencia, estado civil, tipo de remuneración, género, etc

Métodos y Herramientas

- los métodos y herramientas pueden aumentar considerablemente la productividad (¿argumento de venta?)
- generalmente se cuantifican en una escala binaria: se usan o no
- COCOMO intenta medirlo de una manera un poco más sofisticada: “uso de herramientas de software” y “uso de prácticas modernas de programación”

Uso de Herramientas - COCOCMO herramientas de SW

categoría	valor	significado
muy baja	1.24	uso de herramientas básicas de microcomputadoras
baja	1.08	uso de herramientas básicas de minicomputadoras
nominal	1	uso de herramientas básicas de maxicomputadoras
alta	0.91	uso de herramientas fuertes de programación y testeo
muy alta	0.83	uso de herramientas fuertes de análisis, diseño y documentación

Uso de Herramientas - COCOMO prácticas modernas de programación

categoría	valor	significado
muy baja	1.24	no usa
baja	1.08	comienza a usar
nominal	1	algún uso
alta	0.91	uso general
muy alta	0.83	uso rutinario

Uso de Herramientas - COCOMO uso de herramientas CASE

categoría	significado
muy baja	editar, codficar, debug
baja	CASE front-end, back-end, poca integración
nominal	herramientas básicas de ciclo de vida integradas moderadamente
alta	herramientas maduras de ciclo de vida integradas moderadamente
muy alta	herramientas maduras de ciclo de vida, proactivas, bien integradas con procesos, métodos y reuso

Administración de Proyectos de Software
└Métricas de Atributos Externos del Software
└Productividad

Métodos y Herramientas - COCOMO uso de herramientas CASE

- se evalua el uso de herramientas CASE y para cada diseñador:
  1. no se usaron herramientas
  2. se usaron herramientas como ayuda para menos del 20 % de la documentación
  3. se usaron herramientas para generar al menos el 50 % del diseño de alto nivel
  4. se usaron herramientas para generar al menos el 50 % del diseño de alto nivel y del diseño detallado
  5. se usaron herramientas para diseño y generación automática de código en al menos el 50 % del sistema
  6. se usaron herramientas para diseño y generación automática de código en al menos el 90 % del sistema