

# Tema 2

## Tamaño del Software: Longitud, Complejidad y Funcionalidad

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

Pablo.Bermejo@uclm.es

Tamaño del Software

1

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

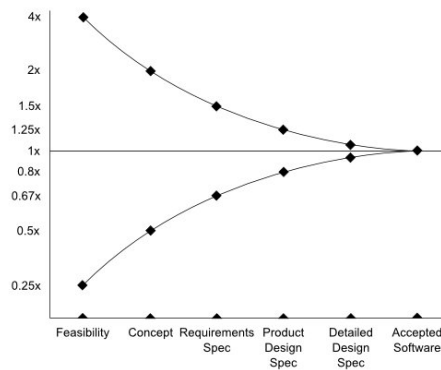
Pablo.Bermejo@uclm.es

Tamaño del Software

2

## 2.1 Introducción

- Uno de los parámetros más importantes **para predecir el coste** de un proyecto software es el **Tamaño del Software**:
  - Debe ser estimado en varias fases del proyecto.
  - A fases más tempranas más incertidumbre sobre el tamaño final y, por lo tanto, también en el coste (dinero o esfuerzo): **cono de incertidumbre**



Tamaño del Software

3

## 2.1 Introducción

- Pero, ¿cómo se mide el tamaño?
- **Según el atributo del software** que se mida, podemos agrupar todos los métodos de medición en 3 categorías:
  - **Longitud**: métodos basados en mediciones del texto en el código fuente.
  - **Complejidad**: caminos lógicos en un bloque de código
  - **Funcionalidad**: requisitos funcionales ofrecidos al usuario. Esta categoría presenta la menor incertidumbre en la planificación del proyecto.

Métodos de medición del Software

Longitud	Complejidad	Funcionalidad
LOC y similares Métricas de Halstead	Ciclomática Profundidad Lógica	Function Points (FP) Full Function Points (FPP) Use Case Points (UCP) Object Points (OP) Feature Points (FeP) Story Points (SP) (esfuerzo)

Tamaño del Software

4

## 2.1 Introducción

- Al estimar el tamaño del software (sea en LOCs, FPs, complejidad,...), podemos estimar el **Esfuerzo** en personal humano.
- Al estimar el esfuerzo en personal humano, tenemos la información clave para predecir el **tiempo** necesario (*Schedule Planning o Planificación del Cronograma*)
- Al estimar el tiempo necesario, podemos realizar la planificación de **costos** (*Cost Planning o Planificación de Costes*)
- Una de las principales funciones del gestor, en tiempo de planificación o incluso de redacción del Acta de Constitución, es predecir:

Tamaño → Esfuerzo → Tiempo → Coste

¡Ojo!, no son parámetros únicos, en cada estimación influyen, como veremos, otros factores aunque con menor peso.

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

## 2.2 Longitud

- La longitud del software hace referencia al tamaño físico del software escrito.
- Las Líneas de Código (LOC) fueron la primera unidad del tamaño del software utilizado:
  - Cada instrucción estaba escrita en una tarjeta
  - $LOC = n^{\circ}$  tarjetas
- A partir de LOC, se pueden derivar otras métricas de longitud:
  - SLOC
  - LLOC
  - STMT
  - Nivel
  - ...

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

## 2.2.1 Longitud - LOC

- **LOSC (Lines of Source Code):** Incluye todas las líneas del código, incluyendo espacios en blanco y comentarios. También referida como SOLC.
- **NCLOC (Non-Commented Lines of Code) o LLOC (Logical Lines of Code):** líneas con código ejecutable y directivas al compilador (ej: `#include`).
- **STMT (Statements):** número de instrucciones ejecutables.
- **CLOC (Commented Lines of Code) o MCOMM (Meaningful Comments) :** líneas comentadas con significado.
- **CLOC% (Densidad de comentarios):**  $(CLOC/NCLOC) * 100$ . Al menos debe llegar al 10%.

```
for(int i=0; i<20; i++){  
/*  
voy a saludar 20 veces */  
System.out.print("Hola Mundo"); System.out.print("\n");  
}
```

SLOC=6  
LLOC=2  
STMT=3  
MCOMM=1  
CLOC% =50%

## 2.2.1 Longitud - LOC

- Por abuso de lenguaje, en diferentes entornos de trabajo y diferentes textos, LOC puede tener varios significados:
  - **LOC=NCLOC=LLOC** es lo más común
  - **LOC=STMT**
  - **LOC=NCLOC+CLOC**
  - **LOC=LOSC=NCLOC+CLOC+espacios en blanco**
- Conviene **especificar de forma explícita el significado** de la unidad de longitud utilizada.
- Cualquier métrica se puede expresar en la escala de almacenamiento digital; por ej.: **KLOC=1.000 LOC**; **MLOC=1.000.000 LOC**,...

- 2.1 Introducción
- 2.2 Longitud
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática
- 2.4 Tamaño Funcional
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.2.2 Longitud - Halstead

### Métricas de Halstead (1977)

- **n1:** número de *operadores* distintos  
 '()' '{}' '[]' for if = ; < > && ? 'comandos' 'nombre de una función' ...
- **n2:** número de *operandos* distintos  
 variables, constantes, tipos definidos
- **N1:** número de ocurrencias de operadores.
- **N2:** número de ocurrencias de operandos.
- **Vocabulario=n=n1+n2 tokens distintos**
- **Longitud del Programa=N= N1 + N2 ocurrencias de tokens**
- **Volumen:** bits necesarios para almacenar el programa.  

$$V = N \times \log_2(n) \text{ bits} = \text{Longitud} \times \log_2(\text{Vocabulario}) \text{ bits}$$

## 2.2.2 Longitud - Halstead

Ej 1. ¿Cuál es el volumen mínimo aproximado de una función con 1 solo comando y sin parámetros?

```
sayHi(){  
  System.out.println("Hi");  
}
```

n1=	n2=	n=
N1=	N2=	N=
V=		

- Para cada método, se recomienda que V no supere los 1000 bits.
- El **Nivel (L)** es representativo de la abstracción que ofrece el lenguaje. Por ejemplo, C tiene mayor nivel que lenguaje ensamblador para el 8085:  
$$L = (2 / n1) * (n2 / N2)$$
- La **Dificultad (D)** en programar: es la inversa del nivel. A menor nivel, mayor dificultad y mayor extensión.  
$$D = 1/L$$
- Las unidades en que se expresan no tienen relación con el significado semántico. Éste es el mayor problema en las métricas de Halstead.

Tamaño del Software

13

## 2.2.2 Longitud - Halstead

- Halstead ofrece una ecuación de cálculo del **Esfuerzo (E)** necesario para desarrollar un programa, el cual depende del Volumen y del Nivel.  
$$E = V/L$$
- La unidad de medida de E es “diferenciaciones mentales elementales”; lo cual dista muy lejos de las medidas estándar de esfuerzo como MM (man-months) o MH(man-hours).
- A partir de E, Halstead ofrece una estimación del **Tiempo (T)** requerido para la implementación del código:

$$T = E/S \text{ segundos}$$

Donde S es el *número de Stroud*: la capacidad de realizar diferenciaciones elementales. Según (Stroud, 1967) S varía entre 5 y 20. Para ingenieros del software se calcula (Hamer, 1982) que S=18. Así, T=E/18 segundos.

Tamaño del Software

14

## 2.2.2 Longitud - Halstead

*Ej 2. Según Halstead, ¿cuánto tiempo necesitamos para programar una función sin parámetros y con solo 1 comando? Usa el resultado de V del ejercicio anterior. ¿Estás de acuerdo?*

- Estimación de la cantidad de **bugs (B)** al entregar el programa:

$$B = E^{2/3} / 3000$$

En el ejercicio anterior, cuál es la probabilidad de cometer 1 error?

Esta ecuación fue ajustada por Halstead según su información histórica, hace 30 años, por lo que no hay que hacerle mucho caso. Solo fijaros en que indica que hay más errores cuanto más esfuerzo es necesario; es decir, cuanto *mayor* es el programa.

## 2.2.2 Longitud - Halstead

### Desventajas de las métricas de Halstead

- **No hay definición** genérica de operador y operando, con lo cual hay cierta subjetividad, y toda comparación entre lenguajes deberá ser realizada dentro de un intervalo de confianza.
- Sus métricas de esfuerzo y tiempo utilizan **unidades** que no se derivan del lado derecho de la ecuación ... matemáticamente poco convincentes.
- A pesar de varios estudios [(Fenton 1997), (Zuse 1998)], no hay consenso en la **escala** de varias de las métricas (Alain Abran, 2010). Por ejemplo, no está claro si el Volumen es ratio u ordinal.

**Ventaja:** después de 30 años, son métricas bien conocidas y extendidas.

## 2.2.2 Longitud - Halstead

Ej 3. Según Halstead, calcula el Esfuerzo y el Tiempo necesario para escribir el siguiente código:

```
int x;
int y;
printf("Dime un número:");
scanf("%d", &x);
printf("Dime otro número:");
scanf("%d", &y);
if(x<y) printf("%d es mayor que %d", y,x);
if(x>y) printf("%d es menor que %d", y,x);
if(x==y) printf("¡Son iguales!");
```

n1=	n2=	n=	tokens diferentes
N1=	N2=	N=	#tokens
V = N x log2(n) =			bits
L = (2 / n1)x(n2 / N2) =			
E=V/L=			dif. Ment. Elemen.
T=E/18=			segundos

n1	#	n2	#

## 2.2 Longitud

### Críticas sobre las métricas de longitud del software

- Todas las métricas basadas en longitud son **sencillas de calcular**, sin embargo presentan desventajas importantes.
- Diferentes **lenguajes** requieren diferentes *longitudes* para la misma funcionalidad.
- La métrica de longitud utilizada puede no tener mucho que ver con el **esfuerzo real**:
  - Herramientas CASE para generación automática de código.
  - Reutilización de código: módulos, clases, herencia,...
  - La programación visual generan menos código aunque presenten la misma funcionalidad.

- 2.1 Introducción
- 2.2 Longitud
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática
- 2.4 Tamaño Funcional
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.2.3 Longitud – Estimación previa

- Si hacemos caso de  
 Tamaño → Esfuerzo → Tiempo → Coste,  
 Si el tamaño en Longitud se calcula cuando el código ya está escrito, ¿cómo se planifica el esfuerzo, tiempo y coste antes de iniciar un proyecto software?
- **Estimación por Analogía:** basándose en información registrada de proyectos anteriores, el gestor de proyecto puede predecir la longitud del software mediante:
  - **Ecuación de regresión:** se usa una función multivariada, ajustando los parámetros (según el repositorio de datos) de las variables utilizadas (lenguaje, número de requisitos, habilidad del programador,...).

Por ejemplo, podemos calcular el ratio de expansión Diseño-a-Código ( $\sigma$ ) de proyectos anteriores similares. Así, dado el diseño realizado del sistema en varios diagramas de clases de tamaño  $C$ , podríamos tener que  $LOC = \sigma \times \sum C_i$

## 2.2.3 Longitud – Estimación previa

- **CBR (Cased-based Reasoning):** utilizando información de proyectos anteriores, se comparan los atributos parecidos, y el grado de las diferencias. En base a unas reglas predefinidas, con cada proyecto pasado se hace una predicción y se mezclan todas las predicciones (cada proyecto puede tener un peso distinto).

El proyecto A tuvo 800.000 LOC, y el B 2.000.000. El proyecto A se parece en un 90% de los atributos al nuevo proyecto, y el B en un 70%.

Peso del proyecto A =  $90/160 = 0,5625$

Peso del proyecto B =  $70/160 = 0,4375$

$LOC_{nuevo\_proyecto} = 0,5625 \times 800.000 + 0,4375 \times 2.000.000 = 1.325.000 \text{ LOC}$

- **Uso de clasificadores de aprendizaje automático:** a partir de la base de datos de proyectos pasados, podemos utilizar técnicas de Aprendizaje Automático para construir modelos que realicen una predicción con los atributos del nuevo proyecto.

Clasificadores probabilísticos, árboles de decisión, coordenadas espaciales,...

## 2.2.3 Longitud – Estimación previa

- **Estimación de Expertos con el método Delphi:** un grupo de gestores o estimadores experimentados, realizan una estimación anónima (normalmente utilizando alguna de las técnicas por analogía). Se presentan los resultados, y se vuelven a realizar estimaciones, hasta llegar a un consenso.

### Plugins para calcular métricas de Longitud en Eclipse:

-**LOC:** Eclipse Metrics: <http://sourceforge.net/projects/metrics/>

-**Halstead:** Google CodePro AnalytiX: <https://developers.google.com/java-dev-tools/codepro/doc/?hl=es>

- 2.1 Introducción
- 2.2 Longitud
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática
- 2.4 Tamaño Funcional
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.3 Complejidad Ciclomática

- Además de ser una métrica relacionada con la Calidad, las métricas de complejidad dan una idea del tamaño lógico o dificultad que presenta el software.
- La **Complejidad Ciclomática** o **Complejidad de McCabe** [Thomas J. McCabe,1976] es una métrica que mide los caminos independientes en el código fuente de una función, módulo, clase,...
- No confundir con el orden de la complejidad:
  - Orden de complejidad: número de ejecuciones en el peor y mejor caso.
  - Complejidad ciclomática: número de caminos

## 2.3 Complejidad Ciclomática

- El valor de la Complejidad Ciclomática (CC) de una función indica las decisiones lógicas que se deben realizar.
- Pero, lo más importante, es que existe **correlación positiva entre CC y nº de bugs**.
- La CC se mide sobre el **Flujo de Control (G)** de la función a medir:
  - Un nodo representa un bloque de instrucciones ejecutadas de forma secuencial.
  - Un arco representa un punto de decisión (if,else, case, while,...).
- Para un método o función:

$$CC = E - N + 2$$

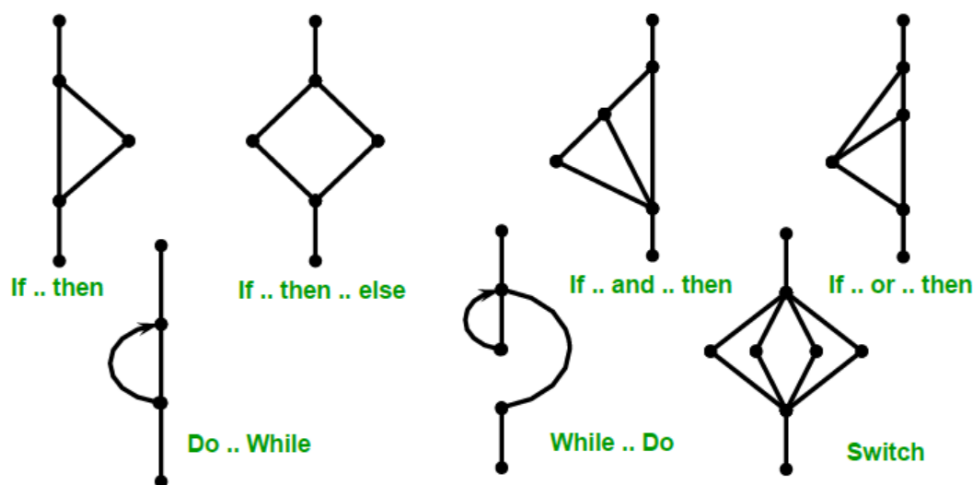
N = #nodos

E = #arcos

CC también se suele escribir como **V(G)**

## 2.3 Complejidad Ciclomática

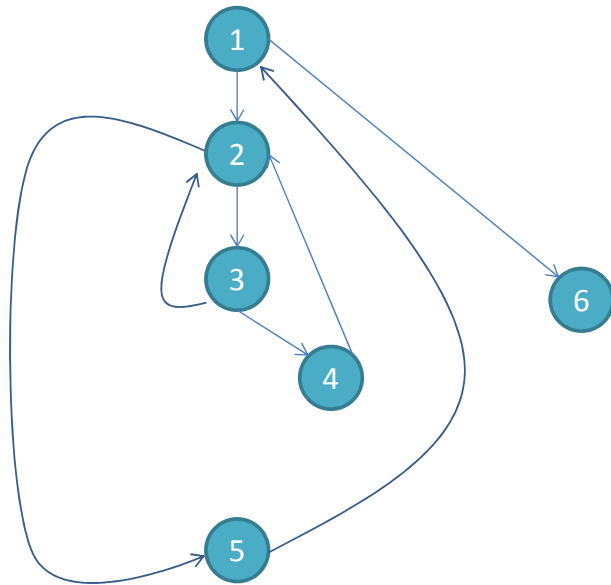
Algunos flujos que podemos encontrar según las decisiones lógicas



[Clark & Brennan, 2008]

## 2.3 Complejidad Ciclomática

```
incrementaX(boolean print){  
  x = global_x; //nodo 1  
  y=global_y; //nodo 1  
  while (x<y) do  
    int j = x; //nodo 2  
    while (j<y) do  
      j++; //nodo 3  
      if(print) then  
        println(j); //nodo 4  
      end do;  
      x=x+1; //nodo 5  
    end do;  
  //salir es nodo 6  
}
```



$$CC = E - N + 2 = 8 - 6 + 2 = 4 \text{ caminos independientes}$$

## 2.3 Complejidad Ciclomática

*Ejercicio 4: dibuja el grafo de flujo del siguiente programa y calcula la complejidad de McCabe*

```
int a,b,c, min;  
scanf("%d",&a);  
scanf("%d",&b);  
scanf("%d",&c);  
  
If(a<b && a<c)  
  min=a  
else  
  if(c<b)  
    min = c;  
  else min=b;  
printf("%d es el menor",min);
```

## 2.3 Complejidad Ciclomática

- Al igual que a mayor LOC mayor nº de bugs, a mayor número de caminos, más casos de prueba de caja blanca hay que hacer para conseguir cobertura máxima en las pruebas → Calidad del Software.
- La escala de la CC es la siguiente:
  - 1-10: código mantenible y no complejo.
  - 10-20: conviene comenzar a dividir los métodos.
  - 20-40: código demasiado complejo; además es demasiado difícil de testear
  - 40-... : te van a echar!

## 2.3 Complejidad Ciclomática

- En realidad no hace falta hacer el grafo: contar condiciones + 1, así es como lo calculan las herramientas.
  - Compruébalo en los ejercicios anteriores.
  - La utilidad del grafo está en preparar los casos de prueba (ajeno al propósito de este tema).
- Plug-ins que implementan la complejidad de McCabe:
  - Java: <http://eclipse-metrics.sourceforge.net/>
  - C#: <http://sourceforge.net/projects/devadvantage/>
- La CC se puede estimar previamente por Analogía o Comité de Expertos como en el caso de LOC.

- 2.1 Introducción**
- 2.2 Longitud**
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática**
- 2.4 Tamaño Funcional**
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.4 Tamaño Funcional

- Si medimos la productividad de un programador en LOC/mes, estaremos diciendo que a más código se genere, es mejor.  
  
→ no se debe sesgar el diseño en términos de tamaño
- Además, ya hemos visto que dependiendo del entorno de programación, la misma funcionalidad puede conseguirse con diferentes tamaños.

## 2.4 Tamaño Funcional

- El **tamaño funcional** representa los requisitos funcionales que se ofrecen al usuario.
- Es posible calcularlo en **fases tempranas** del proyectos.
- No se debe utilizar como único parámetro para estimar el esfuerzo, ya que no tiene en cuenta los esfuerzos para cumplir con requisitos no funcionales (Calidad).
- Existen muchas propuestas para estimar el tamaño funcional:
  - Function Points (FP)
  - Full Function Points (FPP)
  - Use Case Points (UCP)
  - Object Points (OP)
  - Feature Points (FP)
  - Story Points (SP)

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

## 2.4.1 Tamaño Funcional - FP

### Function Points (FP)

- Existen diferentes metodologías para su cálculo: FPA, COSMIC, NESMA...
- La primera y más famosa es FPA (Function Point Analysis), propuesta por Allan J. Albretch en 1978.
  - Actualmente el IFPUG es el grupo encargado de mantener la metodología FPA.
  - Versión actual: 4.3.1 de 2010.
- En general, los FP se calculan en base al número y complejidad de:
  - Entradas Internas
  - Salidas y Peticiones Externas
  - Archivos lógicos

## 2.4.1 Tamaño Funcional - FP

- Los FP calculados deben ser ajustado según las propiedades de la aplicación con un valor de ajuste VAF.
- El VAF puede depender de características como necesidad de cálculos distribuidos, facilidad de uso de la aplicación, complejidad de las operaciones matemáticas, etc...
- Para el cálculo del tamaño funcional en FP es necesaria una **especificación muy detallada** de los requisitos → no es posible su cálculo sin realizar varias entrevistas y depurar los requisitos (al no ser que se estime!).

## 2.4 Tamaño Funcional - FP

*Tabla obtenida de [Strengths and Weaknesses of Software Metrics, 2006]*

**Table 4: Ratios of Logical Source Code Statements to Function Points for Selected Programming Languages Using Version 4.1 of the IFPUG Rules**

Language	Nominal Level	Source Statements Per Function Point		
		Low	Mean	High
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
FORTRAN	3.00	75	107	160
COBOL	3.00	65	107	150
PASCAL	3.50	50	91	125
PL/I	4.00	65	80	95
ADA 83	4.50	60	71	80
C++	6.00	30	53	125
Ada 95	6.50	28	49	110
Visual Basic	10.00	20	32	37
SMALLTALK	15.00	15	21	40
SQL	27.00	7	12	15

Tamaño del Software

37

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

Pablo.Bermejo@uclm.es

Tamaño del Software

38

## 2.4.2 Tamaño Funcional - FFP

### Full Function Points (FFP)

- Es una extensión a los FP de FPA iniciada en 1997.
- Añade formas de contar el tamaño de sub-procesos de control, y grupos de datos que cambian en tiempo real.
- Resulta más apropiado que FP cuando se mide un software de tiempo-real o empotrado, en el que las transacciones de datos no son tan importantes como los eventos.
- FFP es un super-conjunto de los FP.
- La metodología COSMIC-FFP mejora los FFP, también con el objetivo de poder calcular el tamaño funcional en aplicaciones de tiempo real y software empotrado: Sistemas operativos, sistemas de aviación,... Su versión actual es la 3.0.1 de 2009.

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

## 2.4.3 Tamaño Funcional - UCP

### Use Case Points (UCP)

- Propuestos por Gustav Karner en 1993.
- Si somos capaces de utilizar los Casos de Uso para estimar el tamaño funcional del software, posiblemente podamos predecir el esfuerzo de desarrollo en las **etapas más tempranas** de captura de requisitos (depende de si ya tenemos formado el equipo y conocemos sus aptitudes).
- Los elementos a medir son:
  - Casos de uso
  - Actores
  - Restricciones y factores ambientales (dificultad, idiomas, requisitos técnicos,...)

## 2.4.3 Tamaño Funcional - UCP

- Al igual que con los FPs, los UCP no solo se calculan a partir del número de elementos sino también según la complejidad de estos.
  - Complejidad de los Casos de Uso se basa en transacciones (cualquier entrada o salida; o cualquier acción en el diagrama de actividad)

#Transacciones	Complejidad	Peso
<4	Simple	5
[4-7]	Media	10
>7	Alta	15

- Complejidad de los Actores

Tipo de interacción con el caso de uso	Complejidad	Peso
API	Simple	1
Protocolo (ftp, http,...), BBDD o interfaz de texto	Media	2
GUI	Alta	3

## 2.4.3 Tamaño Funcional - UCP

- Los UCP sin ajustar serán la suma de los elementos multiplicados por sus respectivos pesos.
- Este tamaño hay que ajustarlo según requisitos técnicos y factores ambientales.
- En la literatura se proponen diferentes maneras de convertir los **UCP en tiempo**. Casi todos se basan en:
  - Multiplicar los casos de uso por un número fijo de horas.
  - El anterior utilizando pesos según la complejidad.
  - El anterior añadiendo los factores ambientales.
  - Por analogía con proyectos pasados.

*Tendréis un ejemplo completo en la presentación de uno de vosotros como propuesta del Trabajo 1, y también aquí:*

<http://www.laboratorioti.com/2013/02/14/metodo-de-estimacion-puntos-casos-de-uso-use-case-points/>

### 2.1 Introducción

### 2.2 Longitud

#### 2.2.1 LOC

#### 2.2.2 Halstead

#### 2.2.3 Estimación Previa

### 2.3 Complejidad Ciclomática

### 2.4 Tamaño Funcional

#### 2.4.1 Function Points

#### 2.4.2 Full Function Points

#### 2.4.3 Use Case Points

#### 2.4.4 Object Points

#### 2.4.5 Feature Points

#### 2.4.6 Story Points

## 2.4.4 Tamaño Funcional - OP

### Object Points (OP)

- Propuestos por Fenton&Pfleeger en 1997.
- No tienen nada que ver con los objetos de programación OO. De hecho, en COCOMO II le cambian el nombre a **Application Points**.
- Se pueden utilizar cuando ya se dispone del diseño de las **pantallas e informes** que el sistema debe generar, y los componentes de lenguajes de Tercera generación (Java , C,...). SQL o Visual Basic se consideraría Cuarta generación, así que los OP estándar no los contarían como módulo 3GL.
- Tiene en cuenta la **reutilización de código**, de forma que este no se cuenta en su totalidad para el tamaño funcional a generar.

## 2.4.4 Tamaño Funcional - OP

- La complejidad de las pantallas e informes dependen del número de vistas (datos) y fuentes de datos (clientes y servidores).

Complejidad de las pantallas

# vistas	#fuentes de datos		
	<4	[4-7]	>7
<3	Simple	Simple	Media
[3-7]	Simple	Media	Difícil
>7	Media	Difícil	Difícil

Complejidad de los informes

# vistas	#fuentes de datos		
	<4	[4-7]	>7
[0-1]	Simple	Simple	Media
[2-3]	Simple	Media	Difícil
>3	Media	Difícil	Difícil

Valor de la complejidad según elemento

# vistas	Complejidad		
	Simple	Media	Difícil
Pantalla	1	2	3
Informe	2	5	8
Componente 3GL	Cada uso de un 3GL suma 10		

## 2.4.4 Tamaño Funcional - OP

- Los OP son la suma de la complejidad de todas las pantallas, informes y componentes de 3GL.
- Si el  $R\%$  de los objetos se reutilizan:

$$\text{Nuevos OP} = \text{OP} \times (100-R)/100$$

- Dentro del marco de OP, se recomienda una de **Esfuerzo** en personas/mes según la productividad (estimada de forma subjetiva) del personal:

$$\text{PM} = \text{Nuevos OP} / \text{Productividad}$$

	Muy baja	Baja	Nominal	Alta	Muy alta
Productividad	4	7	13	25	50

## 2.4.4 Tamaño Funcional - OP

*Supón que disponemos de los primeros prototipos para una aplicación de Trabajo de Fin de Grado que sirve para gestionar un servicio de compra online. Aunque no sabemos con seguridad cuántas fuentes de datos habrá, sabemos que serán menos que 8; y la productividad del equipo (el estudiante) es Muy baja.. El resumen de nuestros prototipos es:*

Elemento	Vistas
Pantalla 1: mostrar los artículos	artículo y precio
Pantalla 2: introducir datos del cliente	9 campos
Pantalla 3: administrar inventario	Artículo, precio, precio original, clientes que lo han comprado en el último año.
Informe 1: Factura	Datos comprador (8), artículos, precio, precio final, fecha compra.
Informe 2: Error compra	3 campos

### Paso 1: medir los OP

Elemento	Complejidad	OP
Pantalla 1	Simple	1
Pantalla 2	Difícil	3
Pantalla 3	Media	2
Informe 1	Difícil	8
Informe 2	Media	5

### Paso 2: estimar el esfuerzo

$$\text{Esfuerzo} = 19 \text{ OPs} / 4 = 4.75 \text{ PM}$$

Hacen falta 4.75 meses programando 1 persona. Como Solo programará el estudiante, ese es el tiempo estimado.

- 2.1 Introducción
- 2.2 Longitud
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática
- 2.4 Tamaño Funcional
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.4.5 Tamaño Funcional - FeP

### Feature Points (FeP)

- Son una extensión de los FP.
- Cuentan los mismos elementos que en FP con metodología FPA (funciones transaccionales y funciones de datos), pero además añaden **Algoritmos**, refiriéndose a operaciones complejas: trabajos con eventos, cálculos matriciales,...
- La cuenta y el cálculo del VAF se realiza de forma similar a FP.
- Al igual que COSMIC y FFP, se puede utilizar para sistemas en tiempo real y empotrados.

- 2.1 Introducción
- 2.2 Longitud
  - 2.2.1 LOC
  - 2.2.2 Halstead
  - 2.2.3 Estimación Previa
- 2.3 Complejidad Ciclomática
- 2.4 Tamaño Funcional
  - 2.4.1 Function Points
  - 2.4.2 Full Function Points
  - 2.4.3 Use Case Points
  - 2.4.4 Object Points
  - 2.4.5 Feature Points
  - 2.4.6 Story Points

Pablo.Bermejo@uclm.es

## 2.4.6 Tamaño Funcional - SP

### Story Points (SP)

- El uso de los Puntos de Historia es diferente al resto de estimaciones, siendo el motivo principal que se utiliza en metodologías ágiles de desarrollo software:
  - El equipo selecciona un conjunto de requisitos (Historia de Usuario) cuya complejidad  $X$ , sin número, servirá de referencia.
  - En cada iteración (1 semana más o menos), el equipo selecciona las Historias de Usuario (requisitos) a implementar.
  - Estima el tamaño SP comparando cada Historia con  $X$ .
    - Ej: 3 veces  $X$
    - El factor multiplicador puede ser una serie modificada de fibonacci: 3,5,8,13,40,100, o una etiqueta: *Muy pequeña, Pequeña, X, Grande, Muy grande*, o tallas de ropa: XS, S, M,...
- Es necesario que los requisitos ya estén bien definidos.
- La estimación no es el tamaño funcional del software sino de un **subconjunto de requisitos** a implementar en breve.

## 2.4.6 Tamaño Funcional - SP

- Así, no se estima el esfuerzo en PM ni en tiempo, sino en:
  - “Esta semana tenemos que implementar la Historia de complejidad 3X y esta otra que es Baja”
  - Mejor dejamos la de 3X y cogemos alguna de complejidad X
- El equipo debe ser consciente de sus propias capacidades y seleccionar el conjunto de historias que sea capaz de implementar en 1 iteración.
- SP **no sirve** para estimar el tamaño del desarrollo y por lo tanto tampoco para **planificar el cronograma ni estimar el esfuerzo**, pero sí para que el equipo se gestione de forma autónoma en cada iteración: las metodologías ágiles dan independencia al equipo de desarrollo y lo antepone a los procesos.
- El tamaño SP de una Historia de Usuario no es comparable con los SP de otra Historia de otro equipo → **productividad no comparable** en SP.

## 2.4 Tamaño Funcional – en general

- El tamaño funcional calculado puede convertirse a una estimación de LOC a partir de la información histórica de la empresa:
  - CF: factor de conversión; es decir, LOC/punto funcional
  - **LOC=tamaño funcional x CF**
- Según [Ian Sommerville, 2011], normalmente el factor de conversión es de 200 a 300 LOC/FP
- Según [Jones, 1996], dependiendo del lenguaje:
  - 4GL: 20 LOC/FP
  - 3GL: 80 LOC/FP
  - 2GL: 107 LOC/FP
  - 1GL: 320 LOC/FP

## 2.4 Tamaño Funcional 2.4 Tamaño Funcional – en general

- Y en tiempo, según un estudio reciente por <http://www.isbsg.org/http://www.drdobbs.com/jvm/the-comparative-productivity-of-programm/240005881>

Language	Hours Per Function Point
ASP*	06.1
Visual Basic	08.5
Java	10.6
SQL	10.8
C++	12.4
C	13.0
C#	15.5
PL/1	14.2
COBOL	16.8
ABAP	19.9

Tamaño del Software

55

## 2.4 Tamaño Funcional 2.4 Tamaño Funcional – en general

<http://www.laboratorioti.com/2013/04/02/resultados-encuesta-de-marzo-que-metodo-de-medicion-utilizas-habitualmente-i/>

¿Qué método de estimación utilizas habitualmente?

Otro Método de Medición (38%, 200 Votos)

Puntos Casos de Uso (26%, 137 Votos)

Puntos de Historia (17%, 88 Votos)

Puntos Función IFPUG (16%, 86 Votos)

Objetos físicos (10%, 55 Votos)

Líneas de Código (10%, 53 Votos)

Otro Método de Puntos Función: MK-II, FISMA, etc (4%, 23 Votos)

Puntos Función COSMIC (4%, 22 Votos)

Puntos Función NESMA (4%, 21 Votos)

Votantes Totales: 529

Tamaño del Software

56

## ¿Te acuerdas de...?

- 1) ¿Cuál crees que es el mayor problema de las métricas de Halstead? ¿A qué crees que se debe que aún así se utilicen?
- 2) ¿Qué ventajas tiene el tamaño Funcional frente a la Longitud?
- 3) ¿Cómo calcular nuestra herramienta de programación la complejidad ciclomática?
- 4) ¿Para qué fases del ciclo de vida del proyecto de desarrollo son más útiles lo OP?