

# Software Project Management

Bob Hughes and Mike Cotterell

Fifth Edition

# Software Effort Estimation

## Chapter Five

# What Makes a Successful Project?

Un proyecto tendrá éxito si:

funcionalidad,  
alcance,  
precio,  
calidad...  
tiempo...

Delivering:

- Agreed functionality *funcionalidad acordada*
- On time *A tiempo.*
- At the agreed cost *Acordado con los costes.*
- With the required quality *Con la calidad requerida.*

Stages:

1. Set targets
2. Attempt to achieve targets

Problemas de la estimación de proyectos.

# Some Problems with Estimating

Cuanto esfuerzo tengo que hacer para...

- Subjective nature of much of estimating

Naturaleza subjetiva de la estimación Por ejemplo, algunas investigaciones muestran que las personas tienden a subestimar la dificultad de las tareas pequeñas y sobrestimar la de las grandes.

- It may be difficult to produce evidence to support your precise target

Cada uno tiene su propio interés.

- Political pressures

Lo tienes que hacer en X días / mes.

Implicaciones políticas Los diferentes grupos dentro de una organización tienen diferentes objetivos. Los gerentes de desarrollo de sistemas de información de la pueden, por ejemplo, querer generar trabajo y presionarán a los estimadores para que reduzcan las estimaciones de costos para alentar a la alta gerencia a aprobar proyectos.

No todos están de acuerdo con esto, ya que los desarrolladores estarán más comprometidos con los objetivos que ellos mismos se han fijado. Que los de comercial por ejemplo.

- Managers may wish to reduce estimated costs in order to win support for acceptance of a project proposal

- Changing technologies

Curva de aprendizaje para ser productivo en X tecnología.

framework / entornos nuevos... más rápido que aprenden.

- These bring uncertainties, especially in the early days when there is a 'learning curve'

a Tecnología cambiante Cuando las tecnologías cambian rápidamente, es difícil utilizar la experiencia de proyectos anteriores en proyectos nuevos.

- Projects differ

Aunque hacer hecho cosas parecidas No es igual.

- Experience on one project may not be applicable to another

# Exercise 5.1

Productividad.  
(Para comparar  
proyectos).  
líneas/mes

Calculate the productivity SLOC/m of each project

Project	Design	Coding	Testing	Total w/m (SLOC)
a	3.9	5.3	7.4	16.7 (6050)
b	2.7	13.4	6.5	22.6 (8363)
c	3.5	26.8	1.9	32.2 (13334)
d	0.8	2.4	0.7	3.9 (5942)

Source line of code

hombres/mes

## Exercise 5.1 (ii)

Productivity rates in SLOC/m of each project

Project	Work-Month <i>lines per mes.</i>	SLOC	Productivity (SLOC/month)
a	16.7	6050	362
b	22.6	8363	370
c	32.2	13334	414
d	3.9	5942	1524

## Exercise 5.1 (iii)

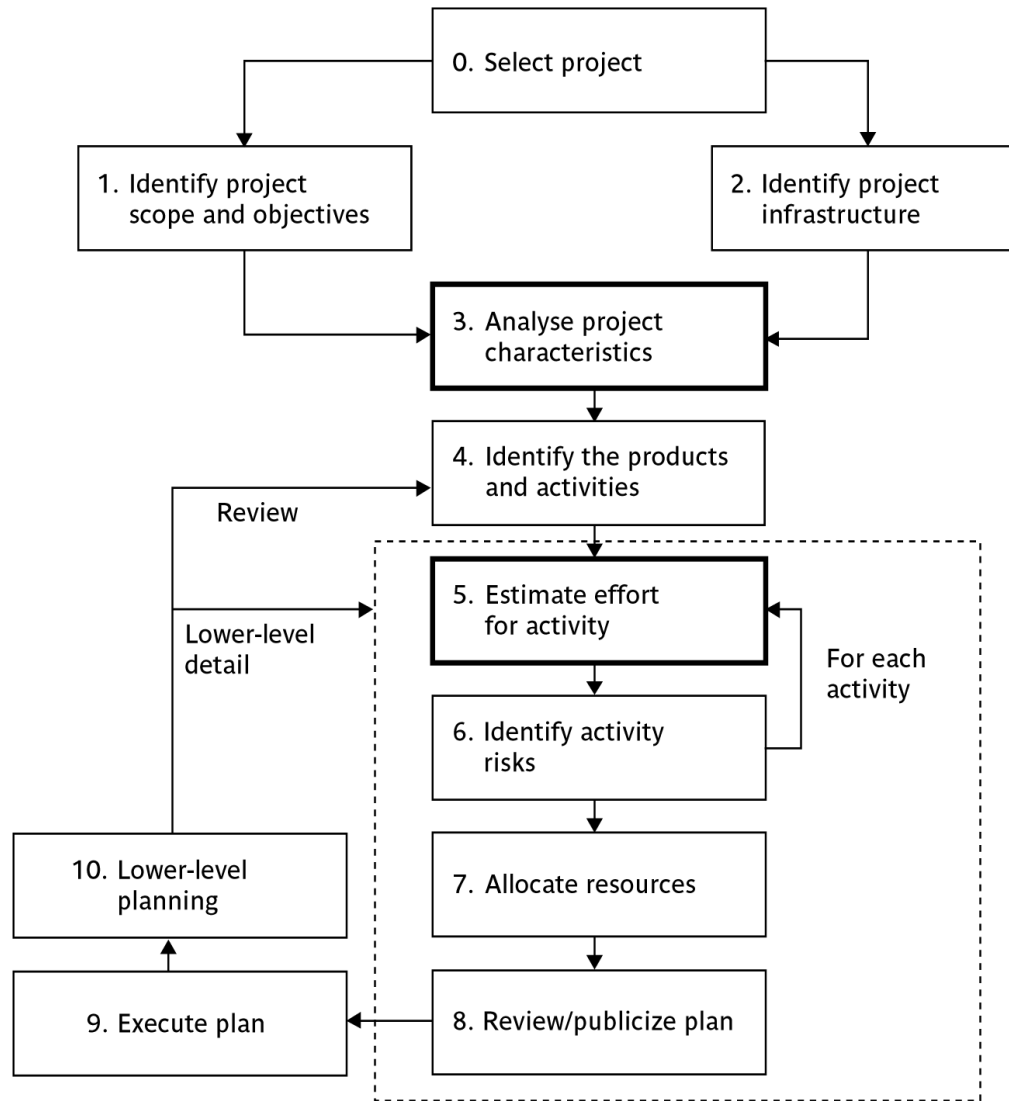
If the average productivity is 617 SLOC/month

Project	Estimated Wok-Month	Actual	Difference
a	$6050 / 617 = 9.8$	16.7	6.9
d	$5942 / 617 = 9.6$	3.9	-5.7

↑  
hombres personas  
mes.

Nos hemos adelantado  
a la media.

Software Project Management - 05.01 Introduction





El líder del proyecto, deberá ser consciente de que una sobreestimación puede hacer que el proyecto tarde más de lo que sería de otra manera. Esto puede explicarse por la aplicación de dos "leyes".

# Over and Under-Estimating

Sobre estimación.

- Lo mejor es pararse? Si lo haces en la mitad de tiempo?

"El trabajo se expande para llenar el tiempo disponible", es decir, dado un objetivo fácil, el personal trabajará menos duro.

- Parkinson's Law

Trabajo que se expande para hacer un trabajo.

Para 3 semanas... como lo distribuyo...

*'Work expands to fill the time available'*

- Brooks' law

Poner más gente a trabajar en un proyecto que va lento.

formación y coordinación del personal.

El sobre coste se convierte en ventaja.

Ley de torres El esfuerzo de implementar un proyecto aumentará desproporcionadamente con la cantidad de personal asignado al proyecto. A medida que el equipo del proyecto crezca, también lo hará el esfuerzo que debe dedicarse a la gestión, la coordinación y la comunicación.

*'Putting more people on a late job makes it later'*

- Weinberg's Zeroth Law of reliability

La calidad debe ser clave... funciona pero... no es de calidad...

*'A software project that does not have to meet a reliability requirement can meet any other requirement'*

# Exercise 5.2

*Como se enfrenta XP a los riesgos por ejemplo... la sobre-carga a la coordinación*

How do agile methods such as XP attempt to address the problems with estimates describe above?

*Porque muchos personas dificultan la comunicación.*

- Programming team does not contain more that ten people in order assist easy team communication
- Quality, project duration, and cost can be controlled by the business management, but scope must be controlled by the development team
- Testing is done as an integral part of the design/code process

*Ajustan mejor los sobre-costes...*

*la calidad etc incorporada en todo el proceso.*

*↳ cada equipo se debe encargar de lo suyo.*

# Basis for Successful Estimating

guardar información sobre proyectos pasados. ↪ utilidad

- Information about **past projects**
  - Need to collect performance details about past project: how big were they? How much effort/time did they need?
  - <https://www.isbsg.org/software-project-estimation/>
- Need to be able to **measure the amount of work** involved
  - Traditional size measurement for software is '*lines of code*' (KLOC) – but this can have problems

# A Taxonomy of Estimating Methods

*Es cuando es totalmente nuevo el proyecto, o no hay datos históricos disponibles.*

*Enfoque de abajo a arriba.*

*La parte ascendente viene al sumar el esfuerzo calculado para cada actividad para obtener una estimación general.*

*Con el enfoque de abajo hacia arriba, el estimador divide el proyecto en sus componentes. Tareas. Con un proyecto grande, el proceso de dividirlo en tareas es iterativo: cada tarea se descompone en sus subtarear componentes y estas, a su vez, podrían analizarse más a fondo. Se sugiere que esto se repita hasta que obtenga tareas que una persona podría hacer en una semana o dos.*

- **Bottom-up - activity based, analytical**

*Planear costes y parámetros en base a históricos (con lo de parametric).*

- **Top-down - overall estimate broken down**

*metes unos parámetros y te da una salida. (de estimaciones)*

- **Parametric or algorithmic models e.g. function points**

*llamas a alguien que sepa.*

- **Expert opinion - just guessing?**

- **Analogy - case-based, comparative**

- **Parkinson and 'price to win'**

*Cuantos somos  
y cuantos meses  
necesitamos*

*y cuanto me vas a  
poder ganar.*

# Bottom-Up versus Top-Down

- **Bottom-up** *Cuanto te va a llevar.  
No lo sabes. No lo has hecho  
nunca.  
dividir el proyecto en actividades  
hasta que consigues estimar el coste del proyecto.*
  - Identify all tasks that have to be done
  - Add up the calculated effort for each activity to get an overall estimate.
  - Use when you have no data about similar past projects
- **Top-down** *Se basa en el resultado previo.  
dices pues esto me va a durar X...  
diseño: X horas...*
  - Based on past project data
  - Produce overall estimate based on project cost drivers
  - Divide overall estimate between jobs to be done

# Bottom-Up Estimating

1. Break project into smaller and smaller components
2. Stop when you get to what one person can do in one/two weeks *(Para de dividir el trabajo en más actividades cuando lo pueda eso ya realizar 1 persona en 1/2 semanas).*
3. Estimate costs for the lowest level activities
4. At each higher level calculate estimate by adding estimates for lower levels

# A Procedural Code-Oriented

## Approach

*Enfoque.*

*En vez de actividades, llegas a módulos y haces la estimación de esfuerzo.*

1. Envisage the number and type of software modules in the final system
2. Estimate the SLOC of each identified module
3. Estimate the work content, taking into account complexity and technical difficulty *no es lo mismo una línea en java que en haskell.*
4. Calculate the work-days effort

Un enfoque orientado al código de procedimiento

El enfoque ascendente descrito anteriormente funciona a nivel de actividades. En el desarrollo de software, una actividad importante es la escritura de código. Aquí describimos cómo se puede utilizar un enfoque ascendente a nivel de componentes de software.

(a) Prevea el número y tipo de módulos de software en el sistema final

La mayoría de los sistemas de información, por ejemplo, se construyen a partir de un pequeño conjunto de operaciones del sistema, p. Ej. Insertar, modificar, actualizar, mostrar, eliminar, imprimir. El mismo principio debería aplicarse igualmente a los sistemas integrados, aunque con un conjunto diferente de funciones primitivas.

(b) Estimar el SLOC de cada módulo identificado

Una forma de juzgar el número de instrucciones que probablemente habrá en un programa

"Módulo de software" es para elaborar un diagrama de estructura del programa y visualizar cuántos aquí implica que se necesitarían instrucciones para implementar cada procedimiento identificado. los

BecaoooCatvestimator puede buscar programas existentes que tengan un conjunto funcional similar y una descripción para ayudar en este proceso.

ejecutado. (c) Estimar el contenido del trabajo, teniendo en cuenta la complejidad y dificultad técnica.

La práctica consiste en multiplicar la estimación del SLOC por un factor de complejidad y dificultad técnica. Este factor dependerá en gran medida del juicio subjetivo del estimador. Por ejemplo, el requisito de cumplir determinados objetivos de rendimiento muy restringidos puede aumentar considerablemente el esfuerzo de programación.

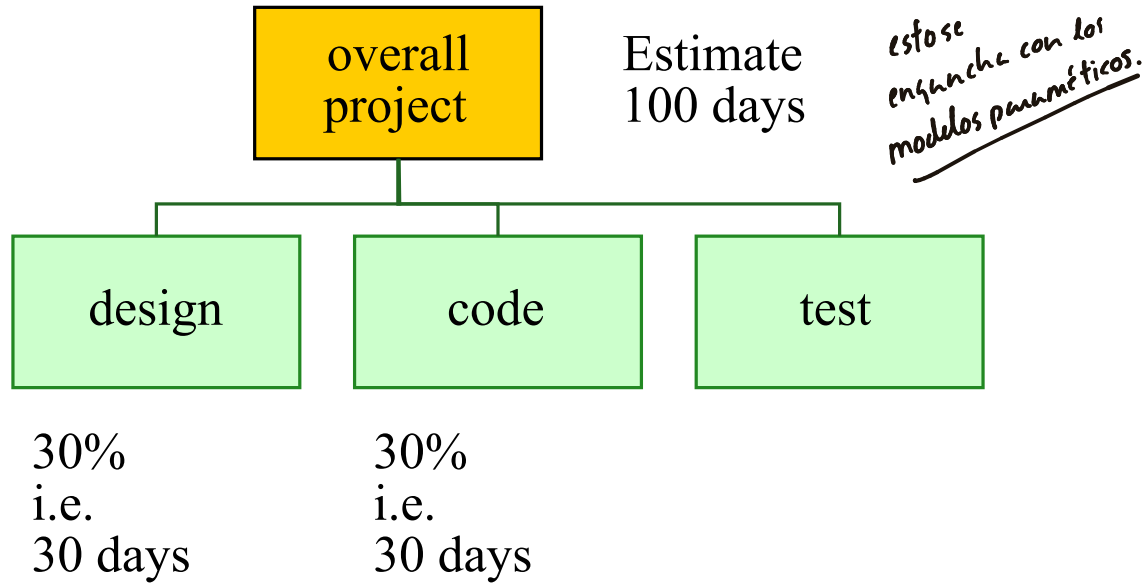
(d) Calcule el esfuerzo de los días laborables

Los datos históricos se pueden utilizar para proporcionar ratios para convertir SLOC ponderado en esfuerzo.



# Top-Down Estimates

*sabes la estimación de horas basandote en experiencias previas.*



- Produce overall estimate using effort driver(s)
- Distribute proportions of overall estimate to components

Tenga en cuenta que los pasos anteriores se pueden utilizar para obtener una estimación de las líneas de código que se pueden utilizar como una entrada a uno de los modelos COCOMO que se describen más adelante:

## Algorithmic/Parametric Models

- Una función que te saca el coste de estimación.  
COCOMO (lines of code) and Function Points examples of these
- Problem with COCOMO etc:
  - Guess → Algorithm → Estimate
- But what is desired is
  - System Characteristics → Algorithm → Estimate

El enfoque de arriba hacia abajo se asocia normalmente con modelos paramétricos (o algorítmicos). Estos pueden explicarse usando la analogía de estimar el costo de reconstruir una casa. Esto es de interés práctico para los propietarios de viviendas que necesitan cobertura de seguro para reconstruir su propiedad si se destruye.

Las compañías de seguros, sin embargo, producen tablas convenientes donde el propietario de la casa puede encontrar estimaciones de los costos de reconstrucción basándose en parámetros tales como el número de pisos y la superficie de una casa. Este es un modelo paramétrico simple.

# Parametric Models - the Need for Historical Data

Para pronosticar el esfuerzo de desarrollo de software tiene dos componentes clave:

El primero (esfuerzo) es un método para evaluar la cantidad de trabajo necesario.

El segundo (productividad) evalúa el ritmo de trabajo al que se puede realizar la tarea.

*necesitamos datos históricos para tener una orientación.*

- Simplistic model for an estimate

Estimated effort = system size / productivity

- System size = lines of code
- Productivity = lines of code per day

Productivity = system size / effort

- Based on past projects

# Parametric Models

- Some models focus on task or system size e.g. **Function Points** *→ Todas las líneas de código.*
- FPs originally used to estimate Lines of Code, rather than effort
  - Number of file types *archivos distintos. operaciones entre ellas.*
  - Number of input and output transaction types
  - → System size

# Parametric Models

- Other models focus on productivity: e.g. **COCOMO**
- Lines of code (or FPs etc) an input
  - System size
  - Productivity factors
  - → Estimated effort

El enfoque de arriba hacia abajo se asocia normalmente con modelos paramétricos (o algorítmicos).

Estos pueden explicarse usando la analogía de estimar el costo de reconstruir una casa. Esto es de interés práctico para los propietarios de viviendas que necesitan cobertura de seguro para reconstruir su propiedad si se destruye. A menos que el propietario de la casa esté en el oficio de la construcción, es poco probable que pueda calcular el número de horas-albañil, horas-carpintero, horas-electricista, etc. requeridas. Las compañías de seguros, sin embargo, producen tablas convenientes donde

el propietario de la casa puede encontrar estimaciones de los costos de reconstrucción basándose en parámetros tales como el número de pisos y la superficie de una casa. Este es un modelo paramétrico simple.

El esfuerzo del proyecto se relaciona principalmente con las variables asociadas con las características del sistema final. ¡Un modo paramétrico! normalmente tendrá una o más fórmulas en la forma:

$$\text{esfuerzo} = (\text{tamaño del sistema}) \times (\text{tasa de productividad})$$

Por ejemplo, el tamaño del sistema podría tener el formato "miles de líneas de código" (KLOC) y tener el valor específico de 3 KLOC, mientras que la tasa de productividad fue de 40 días por KLOC. Estos valores a menudo serán cuestiones de juicio.

Por lo tanto, un modelo para pronosticar el esfuerzo de desarrollo de software tiene dos componentes clave.

El primero es un método para evaluar la cantidad de trabajo necesario. El segundo evalúa el ritmo de trabajo al que se puede realizar la tarea. Por ejemplo, Amanda en IOE puede estimar

que el primer módulo de software que se construirá es 2 KLOC. Entonces ella puede juzgar que si

Kate emprendió el desarrollo del código, con su experiencia podría trabajar a un ritmo de 40 días por KLOC por día y completar el trabajo en  $2 \times 40$  días, es decir, 80 días, mientras que Ken, que tiene menos experiencia, necesitaría 55 días. por KLOC y tarda  $2 \times 55$ , es decir, 110 días para

## 5.7 El enfoque de arriba hacia abajo y los modelos paramétricos 111

112 = Capítulo 5 Estimación del esfuerzo del software

completa la tarea. En este caso, KLOC es un controlador de tamaño que indica la cantidad de trabajo a realizar, mientras que la experiencia del desarrollador es un controlador de productividad que influye en la productividad o en la tasa de trabajo.

Si tiene cifras para el esfuerzo invertido en proyectos anteriores (en días laborables, por ejemplo) y también los tamaños del sistema en KLOC, debería poder calcular una tasa de productividad como

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

Una forma más sofisticada de hacer esto sería mediante el uso de la técnica estadística / regresión de cuadrados este para derivar una ecuación en la forma:

$$\text{esfuerzo} = \text{constante} + (\text{tamaño} \times \text{constante})$$

Algunos modelos paramétricos, como el que implican los puntos de función, se centran en el tamaño del sistema o de la tarea, mientras que otros, como COCOMO, se preocupan más por los factores de productividad. Estos modelos particulares se describen con más detalle más adelante en este capítulo.

Les pide una estimación del esfuerzo de la tarea de alguien que tenga conocimientos sobre ya sea la aplicación o el entorno de desarrollo. Este método se utiliza a menudo al estimar el esfuerzo necesario para cambiar una pieza de software existente. El estimador tendría que examinar el código existente para juzgar la proporción de código afectado y de ahí derivar una estimación. Alguien que ya esté familiarizado con el software estaría en la mejor posición para hacer esto.

*Otra metodología de estimación de proyectos.*

## Expert Judgement *(llamar a alguien que sepa).*

*(alguien que verdaderamente es familiarizado/conozca bien el proyecto).*

- Asking someone who is familiar with and knowledgeable about the application area and the technologies to provide an estimate
- Particularly appropriate where existing code is to be modified
- Research shows that experts judgement in practice tends to be based on analogy

Esto también se llama razonamiento basado en casos. El estimador identifica proyectos completados (casos de origen) con características similares al nuevo proyecto (el caso de destino). El esfuerzo recabado para el caso de origen coincidente se utiliza luego como una estimación base para el objetivo. los

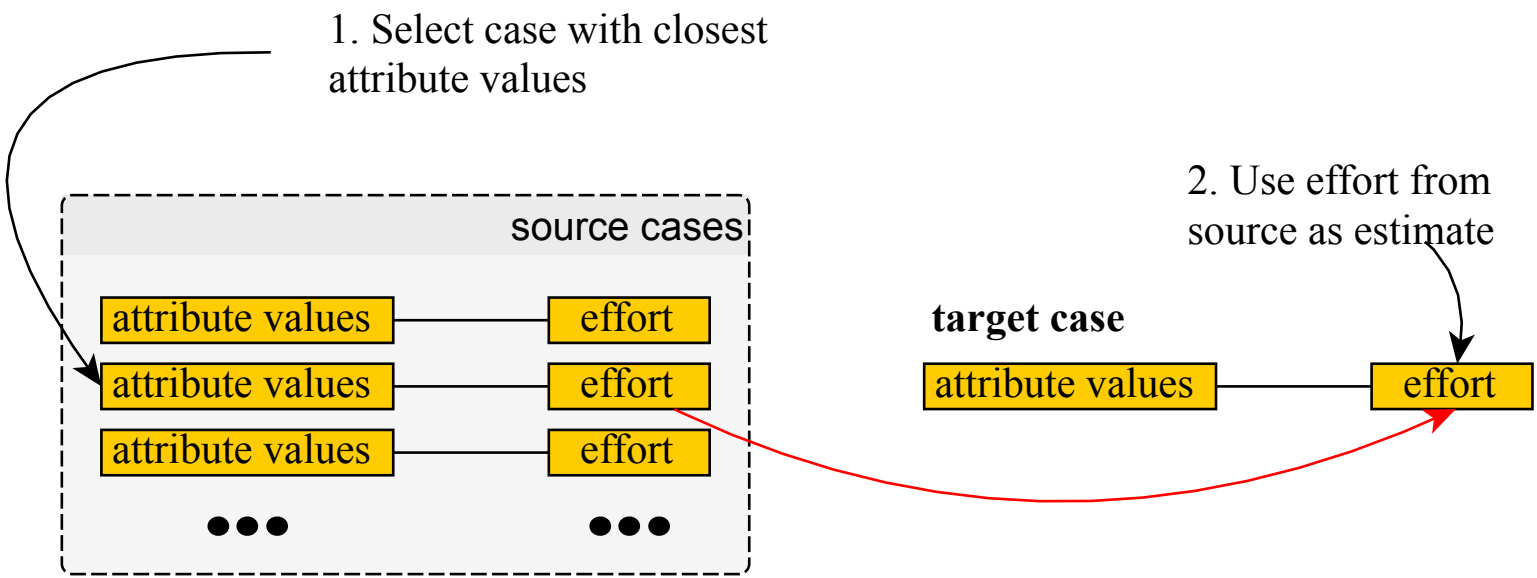
El estimador luego identifica las diferencias entre el objetivo y la fuente y ajusta la estimación base para producir una estimación para el nuevo proyecto. Este puede ser un buen enfoque en el que tiene información sobre algunos proyectos anteriores, pero no lo suficiente para sacar conclusiones generalizadas sobre lo que podrían ser impulsores útiles o tasas de productividad típicas.

Un problema es identificar las similitudes y diferencias entre aplicaciones en las que tiene una gran cantidad de proyectos anteriores para analizar. Un intento de automatizar este proceso de selección es la herramienta de software ANGEL. Esto identifica el caso de origen más cercano al objetivo midiendo la distancia euclidiana entre los casos. La distancia euclidiana se calcula como:

distancia = raíz cuadrada de ((target\_parameter, - source\_parameter,) \* +. (target\_parameter, - source\_parameter,) ')

# Estimating by Analogy

*Te fijas en como te ha ido en proyectos anteriores.*      *Te basas en... Cuantas entra das tenía este programa por ejemplo.*

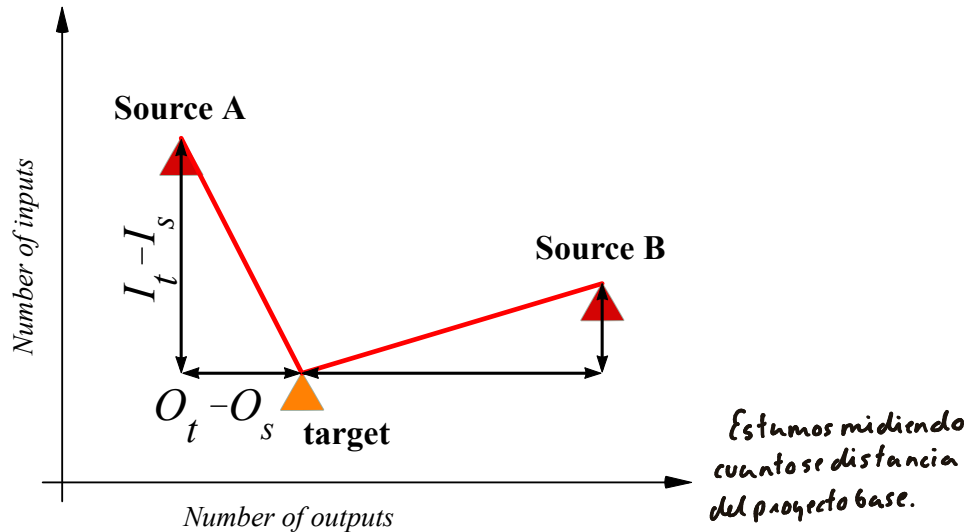




# Stages: Identify

- Significant features of the current project
- Previous project(s) with similar features
- Differences between the current and previous projects
- Possible reasons for error (risk)
- Measures to reduce uncertainty

# Machine Assistance for Source Selection



$$\text{Euclidean distance} = \sqrt{(I_t - I_s)^2 + (O_t - O_s)^2}$$

New project: 7 inputs, 15 outputs

Project A: 8 inputs, 17 outputs, distance 2.24

Project B: 5 inputs, 10 outputs, distance 5.39

*Se trata mas de cerca los modelos  
paramétricos.*

# Parametric Models

- Albrecht estaba investigando la productividad de la programación y necesitaba cuantificar el tamaño funcional de los programas independientemente de sus lenguajes de programación. Desarrolló la idea de puntos de función (FP).
- We are now looking more closely at four parametric models:
    - Albrecht/IFPUG function points
    - Symons/Mark II function points
    - COSMIC function points
    - COCOMO81 and COCOMO II

# Albrecht/IFPUG Function Points

que caracterizaba a los proyectos para que los esfuerzos sean distintos. → Puntos de función.

- Albrecht worked at IBM and needed a way of measuring the relative productivity of different programming languages
- Needed some way of measuring the size of an application without counting lines of code
- Identified **five types of component or functionality** in an information system
- Counted occurrences of each type of functionality in order to get an indication of the size of an information system

La base del análisis de puntos de función es que los sistemas de información comprenden cinco componentes principales,

*que se cuantifican el tamaño de los programas  
independientemente de los lenguajes de programación.  
Entonces las características comunes a todos es:  
entradas, salidas...*

## Albrecht/IFPUG Function Points (ii)

Five function types

1. **Logical Interface File (LIF)** types – equates roughly to a datastore in systems analysis terms. Created and accessed by the target system
2. **External Interface File (EIF)** types – where data is retrieved from a datastore (also as output) which is actually maintained by a different application

# Albrecht/IFPUG Function Points - (ii)

- 3. **External Input (EI)** types – input transactions which update internal computer files
- 4. **External Output (EO)** types – transactions which extract and display data from internal computer files. Generally involves creating reports
- 5. **External Inquiry (EQ)** types – user initiated transactions which provide information but do not update computer files. Normally the user inputs some data that guides the system to the information the user needs

# Albrecht Complexity Multipliers

External Users	Low Complexity	Medium Complexity	High Complexity
EI	3	4	6
EO	4	5	7
EQ	3	4	6
LIF	7	10	15
EIF	5	7	10

<https://www.ifpug.org>

# Example

Payroll application has:

- Transaction to input, amend and delete employee details – An EI that is rated of medium complexity
- A transaction that calculates pay details from timesheet data that is input – An EI of high complexity
- A transaction that prints out pay-to-date details for each employee – An EO of medium complexity



# Example

- A file of payroll details for each employee – Assessed as of medium complexity LIF
- A personnel file maintained by another system is accessed for name and address details – A simple EIF

What would be the FP counts for these?

# Example

FP Counts	
Medium EI	4 FPs
High complexity EI	6 FPs
Medium complexity EO	5 FPs
Medium complexity LIF	10 FPs
Simple EIF	5 FPs
Total	30 FPs

If previous projects delivered 5 FPs a day, implementing the above should take  $30/5 = 6$  days

# Function Points Mark II

- Developed by Charles R. Symons
- 'Software sizing and estimating - Mk II FPA', Wiley & Sons, 1991.
- Builds on work by Albrecht
- Work originally for CCTA:
  - Should be compatible with SSADM; mainly used in UK
- Has developed in parallel to IFPUG FPs
- A simpler method

## Function Points Mk II (ii)

- For each transaction, count
  - Data items input ( $N_i$ )
  - Data items output ( $N_o$ )
  - Entity types accessed ( $N_e$ )

$$FPcount = W_i \times N_i + W_e \times N_e + W_o \times N_o$$

In practice:

$$FPcount = 0.58 \times N_i + 1.66 \times N_e + 0.26 \times N_o$$

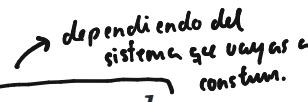
# Exercise 5.9

Calculate the number of unadjusted Mk II function points for the following transaction:

- The operator will input: Customer account number, Customer name, Address, Postcode, Customer type, and Renewal date
- All this information will be set up in a CUSTOMER record on the system's database
- If a CUSTOMER account already exists for the account number that has been input, an error message will be displayed

# COCOMO (COnstructive COnst MOdel)

puedes hacerlo con  
este enfoque COCOMO.

- Based on industry productivity standards - database is constantly updated
- Allows an organization to benchmark its software development productivity
- Basic model:  $effort = c \times size^k$   

- Parameter  $c$  and  $k$  depend on the type of system: **organic**, **semi-detached**, **embedded**
- *Size* is measured in thousands of lines of code, **KLOC**
- *Effort* is measured in person-month (152h)

*Como modelar la  
complejidad de un  
proyecto de software.*

# The COCOMO Constants

System Type	c	k
Organic (broadly, information systems)	2.40	1.05
Semi-detached	3.00	1.12
Embedded (broadly, real-time)	3.60	1.20

- k exponentiation – ‘to the power of...’ adds disproportionately more effort to the larger projects takes account of bigger management overheads

# Development Effort Multipliers (DEM)

- According to COCOMO, the major productivity drivers include:
- **Product attributes:** required reliability, database size, product complexity
- **Computer attributes:** execution time constraints, storage constraints, virtual machine (VM) volatility
- **Personnel attributes:** analyst capability, application experience, VM experience, programming language experience
- **Project attributes:** modern programming practices, software tools, schedule constraints



# Using COCOMO Development Effort Multipliers (DEM)

An example: for analyst capability:

- Assess capability as very low, low, nominal, *high* or very high
- Extract multiplier:
  - Very low: 1.46
  - Low: 1.19
  - Nominal: 1.00
  - High: 0.80
  - Very high: 0.71
- Adjust nominal estimate e.g.  $32.6 \times 0.80 = 26.8$  staff months

# Some Conclusions: How to Review Estimates

- Ask the following questions about an estimate
  - What are the task size drivers?
  - What productivity rates have been used?
  - Is there an example of a previous project of about the same size?
  - Are there examples of where the productivity rates used have actually been found?