

Tecnologías de Programación



Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas



Paradigma de Programación

- ¿Un estilo de Programación?
- ¿Una manera de visualizar la ejecución del programa?
- ¿Una forma de resolver problemas de Programación?
- ¿Un enfoque, perspectiva o filosofía sobre cómo programar?
- ¿Una clasificación de los lenguajes?



Definición de Paradigma

Un paradigma de programación es un marco conceptual, un conjunto de ideas que describe una forma de entender la construcción de programa, como tal define:

- Las herramientas conceptuales que se pueden utilizar para construir un programa (objetos, relaciones, funciones, instrucciones)
- Las formas válidas de combinarlas.

Los distintos lenguajes de programación proveen implementaciones para las herramientas conceptuales descritas por los paradigmas. Existen lenguajes que se concentran en las ideas de un único paradigma así como hay otros que permiten la combinación de ideas provenientes de distintos paradigmas.



Definición de Paradigma

Dado que un paradigma es un conjunto de ideas, su influencia se ve principalmente en el momento de modelar una solución a un problema dado. No es suficiente saber en qué lenguaje está construido un programa para saber qué marco conceptual se utilizó en el momento de construirlo. El paradigma tiene más relación con el proceso mental que se realiza para construir un programa que con el programa resultante.



Paradigma de Programación

- Representan un enfoque particular o filosofía para la construcción del software.
- No es mejor uno que otro sino que cada uno tiene ventajas y desventajas.
- Hay situaciones donde un paradigma resulta más apropiado que otro.



Primera clasificación

- IMPERATIVA
- DECLARATIVA



Imperativo

- Describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador **cómo** realizar una tarea. En muchos sentidos es la programación natural para las CPUs que se basan en este paradigma al nivel más básico (assembler – código de máquina).

Ejemplos: Basic, Java, C, C++, C#, PHP



Declarativo

- Se construye especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que **describen el problema** y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando)



Diferencia

En la programación **imperativa** se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

En la programación **declarativa** las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.



Segunda clasificación

- Secuencial
- Estructurado
- Orientado a Objetos
- Lógico
- Funcional



Secuencial

- **Secuencial o no estructurado:** La programación no estructurada es un paradigma de programación donde todo el código se contiene en un solo bloque continuo. Los lenguajes de programación “no estructurada” tienen que confiar en declaraciones del flujo de la ejecución tales como ***goto***, utilizado en muchos lenguajes para saltar a una sección especificada del código.
 - Ejemplo: Basic

Secuencial

```
10 INPUT "Cuál es su nombre:"; NN$
20 PRINT "Bienvenido al 'asterisquero' "; NN$
25 PRINT
30 INPUT "con cuántos asteriscos inicia [Cero sale]:"; N
40 IF N<=0 THEN GOTO 200
50 AS$=""
60 FOR I=1 TO N
70 AS$=AS$+"*"
80 NEXT I
90 PRINT "AQUI ESTAN:"; AS$
100 INPUT "Desea más asteriscos:"; SN$
110 IF SN$="" THEN GOTO 100
120 IF SN$<>"S" AND SN$<>"s" THEN GOTO 200
130 INPUT "CUANTAS VECES DESEA REPETIRLOS [Cero sale]:"; VECES
140 IF VECES<=0 THEN GOTO 200
150 FOR I=1 TO VECES
160 PRINT AS$;
170 NEXT I
180 PRINT
185 REM A repetir todo el ciclo (comentario)
190 GOTO 25
200 END
```



Estructurado

- **Estructurado:** es una forma de escribir programación de computadora de forma clara, para ello utiliza únicamente tres estructuras: secuencial, selectiva e iterativa; siendo innecesario y no permitiéndose el uso de la instrucción o instrucciones de transferencia incondicional (GOTO).
 - Ejemplo: Pascal

Estructurado

```
PROGRAM MultiplosDe3 (INPUT, OUTPUT);

VAR
    Numero, Cnt: INTEGER;

BEGIN
    Cnt := 0;
    Write ('Entra el primer número de la serie: '); ReadLn (Numero);
    WHILE Numero <> 0 DO
        BEGIN
            IF (Numero MOD 3) = 0 THEN
                INC (Cnt);
            Write ('Dame otro numero (0 para terminar): '); ReadLn (Numero);
        END;
        WriteLn ('La cantidad de múltiplos de 3 ingresados es ', Cnt);
        Write ('Pulse [Intro] para finalizar...')
    END.
```



Orientado a Objetos

- **Orientado a Objetos:** es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.
 - Ejemplo: Smalltalk, Ruby, Python, Java

Orientado a Objetos

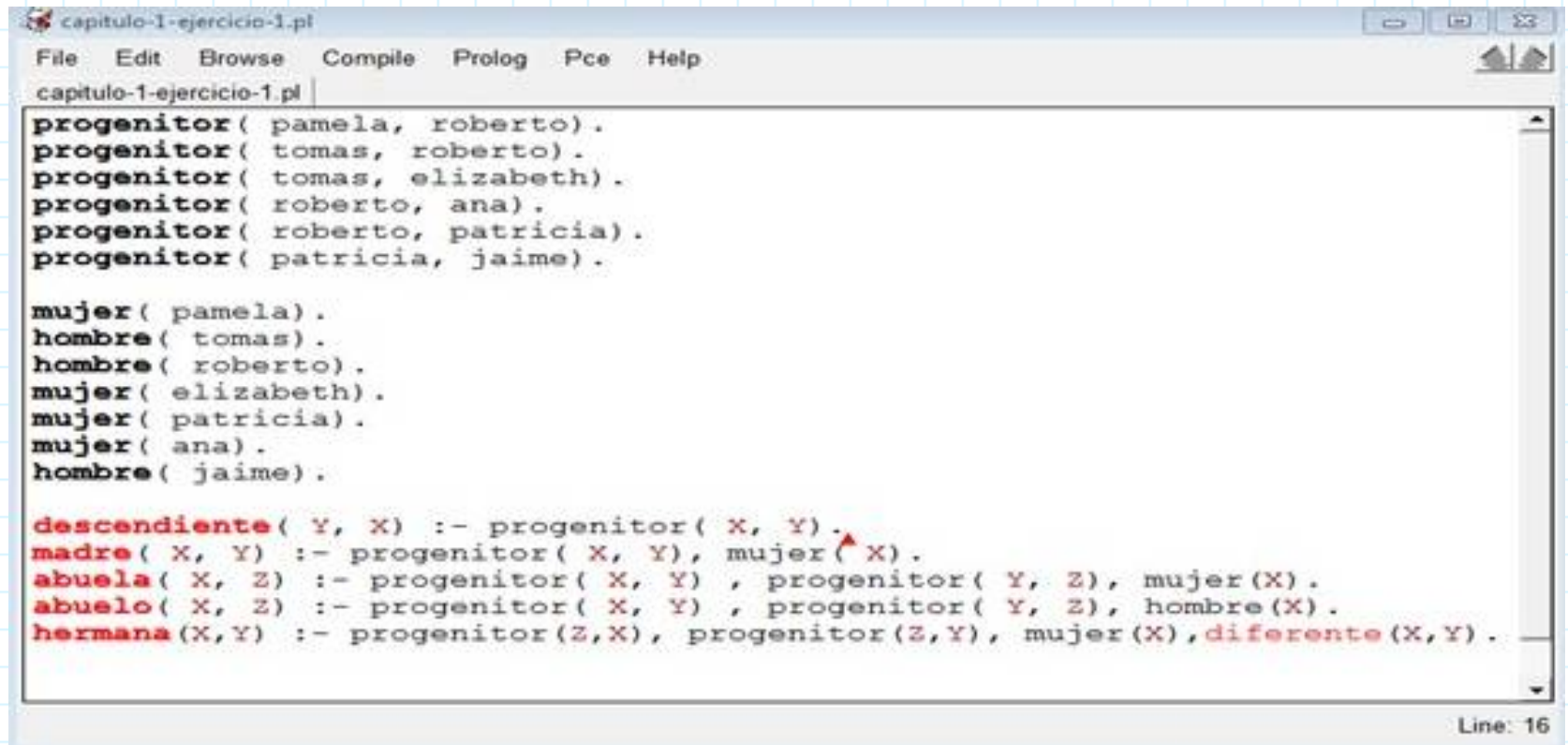
```
1 package ejb3.inscripciones.dto;
2 public class AtributoDTO {
3
4     public static int INPUT_TEXTO = 1;
5     public static int INPUT_NUMERICO = 2;
6     public static int INPUT_SELECT_SIMPLE = 3;
7     public static int INPUT_SELECT_MULTIPLE = 4;
8     public static int INPUT_CHECKBOX = 5;
9     public static int INPUT_RADIO = 6;
10
11     private Integer idAtributo;
12     private String nombre;
13     private String tipoCampoHtml;
14
15     public AtributoDTO(Integer idAtributo, String nombre, String tipoCampoHtml) {
16         this.idAtributo = idAtributo;
17         this.nombre = nombre;
18         this.tipoCampoHtml = tipoCampoHtml;
19     }
20
21     public String getNombre() {
22         return nombre;
23     }
24
25     public String getTipoCampoHtml() {
26         return tipoCampoHtml;
27     }
28
29     public Integer getIdAtributo() {
30         return idAtributo;
31     }
32 }
```



Lógico

- Esta programación se basa en un subconjunto del cálculo de predicados, incluyendo instrucciones escritas en formas conocidas como cláusulas de Horn. Este paradigma puede deducir nuevos hechos a partir de otros hechos conocidos. Un sistema de cláusulas de Horn permite un método particularmente mecánico de demostración llamado resolución.
 - Ejemplo: Prolog

Lógico



```
capitulo-1-ejercicio-1.pl
File Edit Browse Compile Prolog Pce Help
capitulo-1-ejercicio-1.pl

progenitor( pamela, roberto).
progenitor( tomas, roberto).
progenitor( tomas, elizabeth).
progenitor( roberto, ana).
progenitor( roberto, patricia).
progenitor( patricia, jaime).

mujer( pamela).
hombre( tomas).
hombre( roberto).
mujer( elizabeth).
mujer( patricia).
mujer( ana).
hombre( jaime).

descendiente( Y, X) :- progenitor( X, Y).
madre( X, Y) :- progenitor( X, Y), mujer( X).
abuela( X, Z) :- progenitor( X, Y), progenitor( Y, Z), mujer(X).
abuelo( X, Z) :- progenitor( X, Y), progenitor( Y, Z), hombre(X).
hermana(X,Y) :- progenitor(Z,X), progenitor(Z,Y), mujer(X), diferente(X,Y).

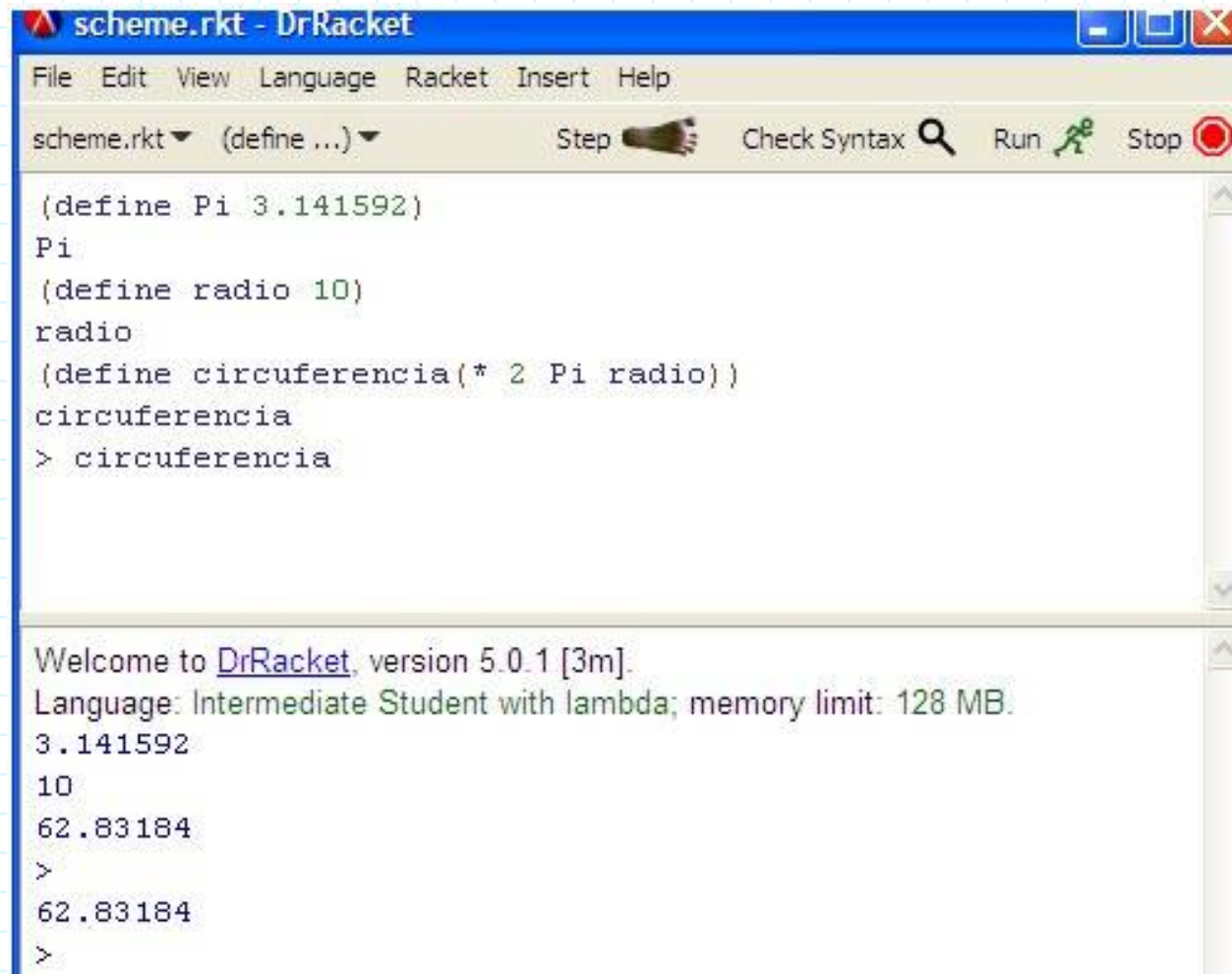

Line: 16
```



Funcional

- Es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.
Modelo matemático de composición funcional donde el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado.
 - Ejemplos: Lisp, Scheme, Haskell

Funcional



```
scheme.rkt - DrRacket
File Edit View Language Racket Insert Help
scheme.rkt (define ...) Step Check Syntax Run Stop

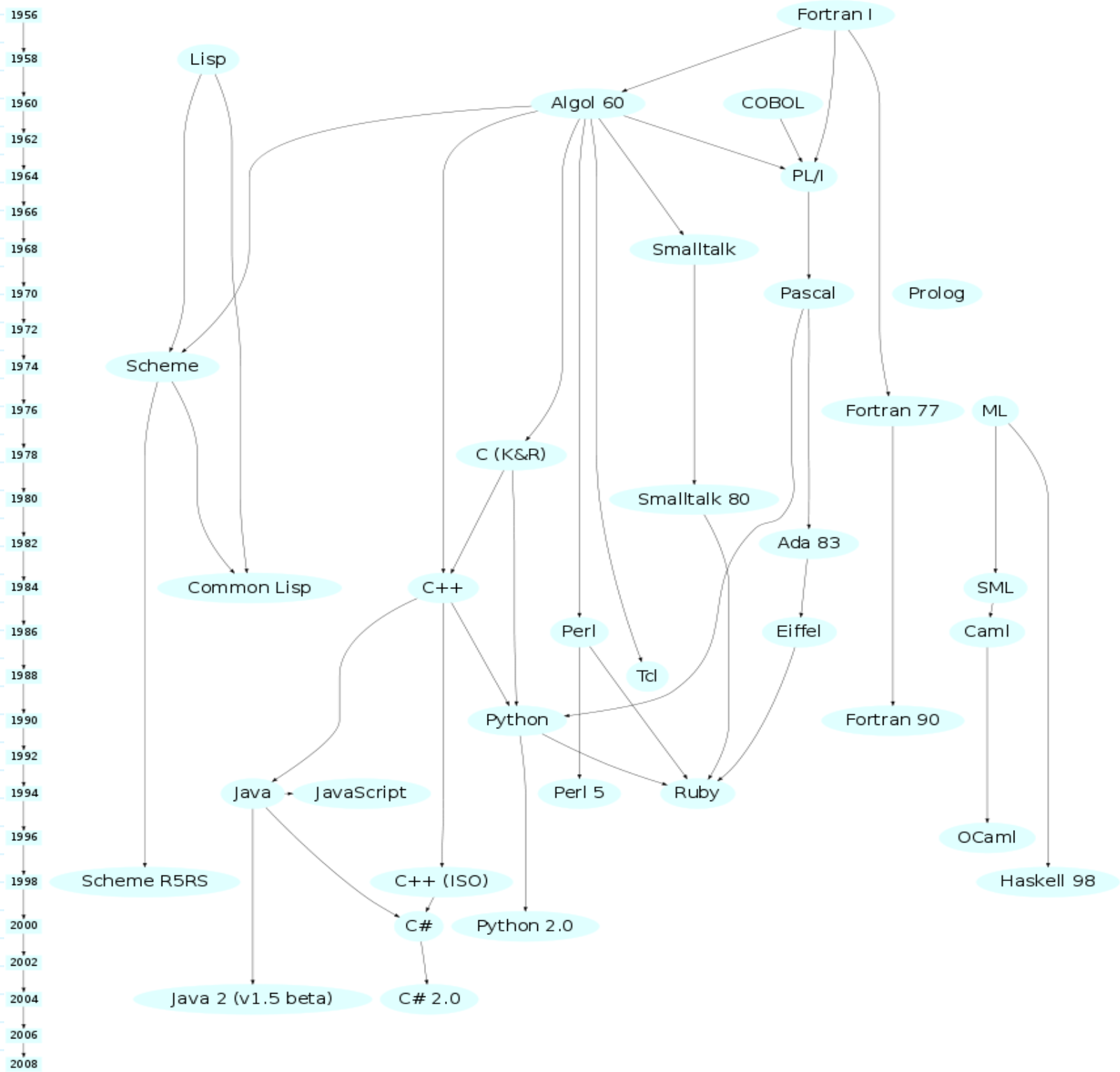
(define Pi 3.141592)
Pi
(define radio 10)
radio
(define circuferencia(* 2 Pi radio))
circuferencia
> circuferencia

Welcome to DrRacket, version 5.0.1 [3m].
Language: Intermediate Student with lambda; memory limit: 128 MB.
3.141592
10
62.83184
>
62.83184
>
```




1º y 2º clasificación

IMPERATIVA	DECLARATIVA
ASSEMBLER	FUNCIONAL
SECUENCIAL	LOGICO
ESTRUCTURADO	
OBJETOS	



A que nos enfrentamos ...



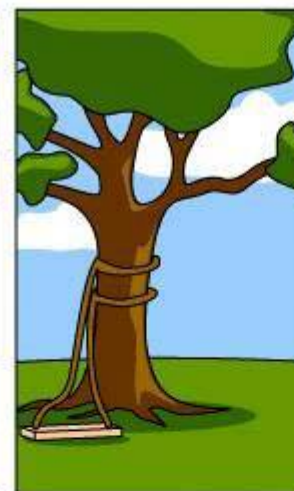
Cómo el cliente lo explica



Cómo el líder del proyecto lo entiende



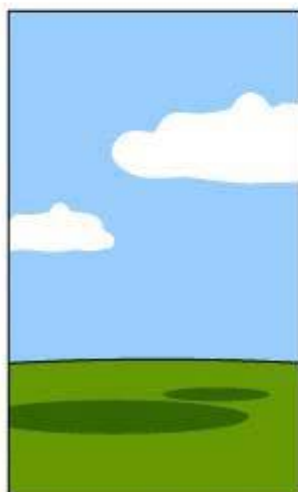
Cómo el analista lo diseña



Cómo el programador lo escribe



Cómo el asesor lo describe



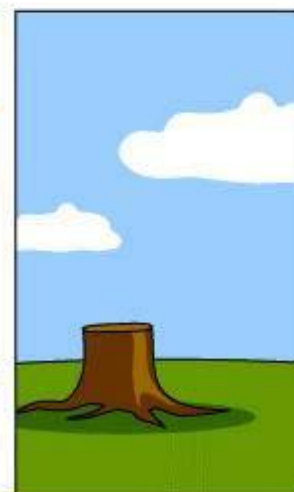
Cómo se documenta el proyecto



Qué aplicaciones se instalan



Cómo se factura al cliente




Así se le dará soporte



Lo que el cliente realmente necesitaba

La Concepción del Sistema, el Funcional, el Técnico, el Desarrollo, las Pruebas... es decir, todo el coste incurrido hasta la puesta en marcha de la Aplicación es la zona verde de la curva; una vez puesta en marcha comienza su fase de mantenimiento (la zona naranja), que supondrá seguramente, a lo largo de toda la Vida de la Aplicación, entre un 75% y un 90% del coste total dedicado a dicha Aplicación a lo largo de su vida útil.





De seguir así, en unos pocos años el 90% de la plantilla de Desarrollo, o más, estaría dedicada exclusivamente al Mantenimiento... y entonces, ¿quién escribiría las nuevas Aplicaciones? Nuevo personal, claro está. Que habría que fichar o subcontratar, obviamente. Y que, una vez terminada su Aplicación, quedaría, en un 90%, dedicado a su mantenimiento. Y habría que fichar entonces más personal nuevo para escribir las nuevas aplicaciones... No parecía que el círculo vicioso originado por esta forma de trabajar resultara muy eficiente a la larga.

Así que las connotaciones son evidentes:

No hay que escatimar esfuerzos durante la etapa inicial de Diseño y Construcción de la Aplicación para reducir todo lo posible el coste del Mantenimiento posterior.