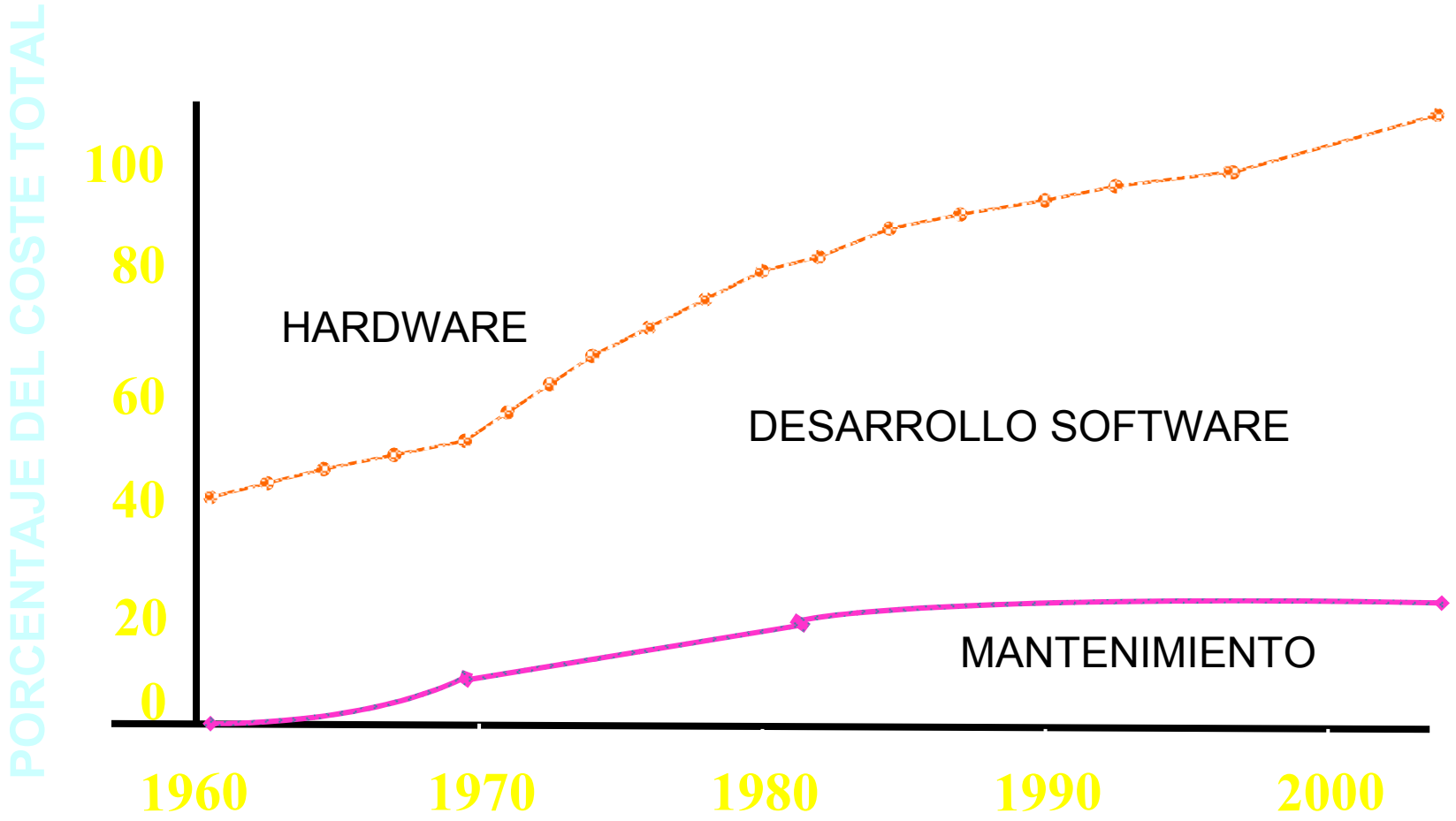
Seminario de Conceptos en Ingeniería Software

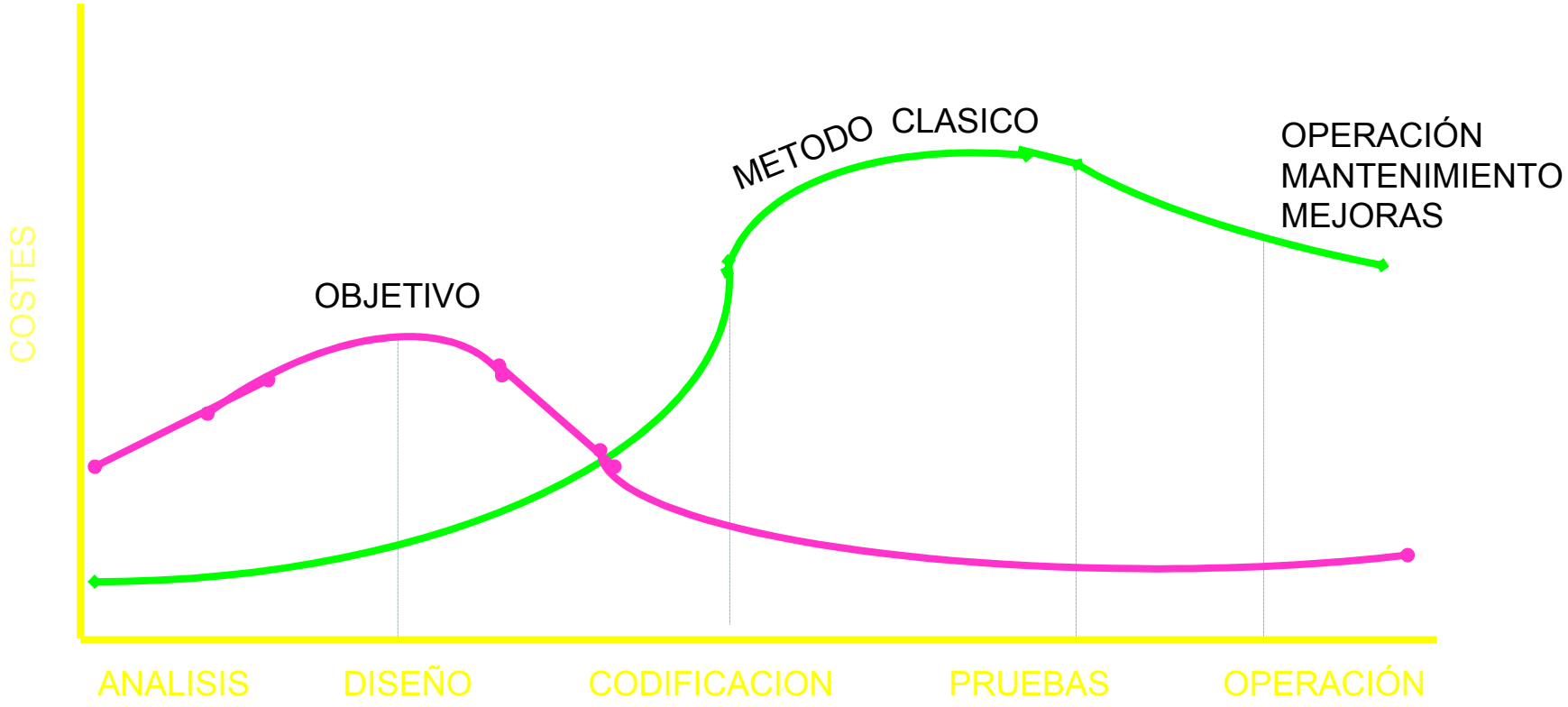
"Muchos conceptos son el mismo perro con distinto collar"

Ignacio López
Cirsa
Abril 2000

Evolución de los costes de hardware/software



Costes del software

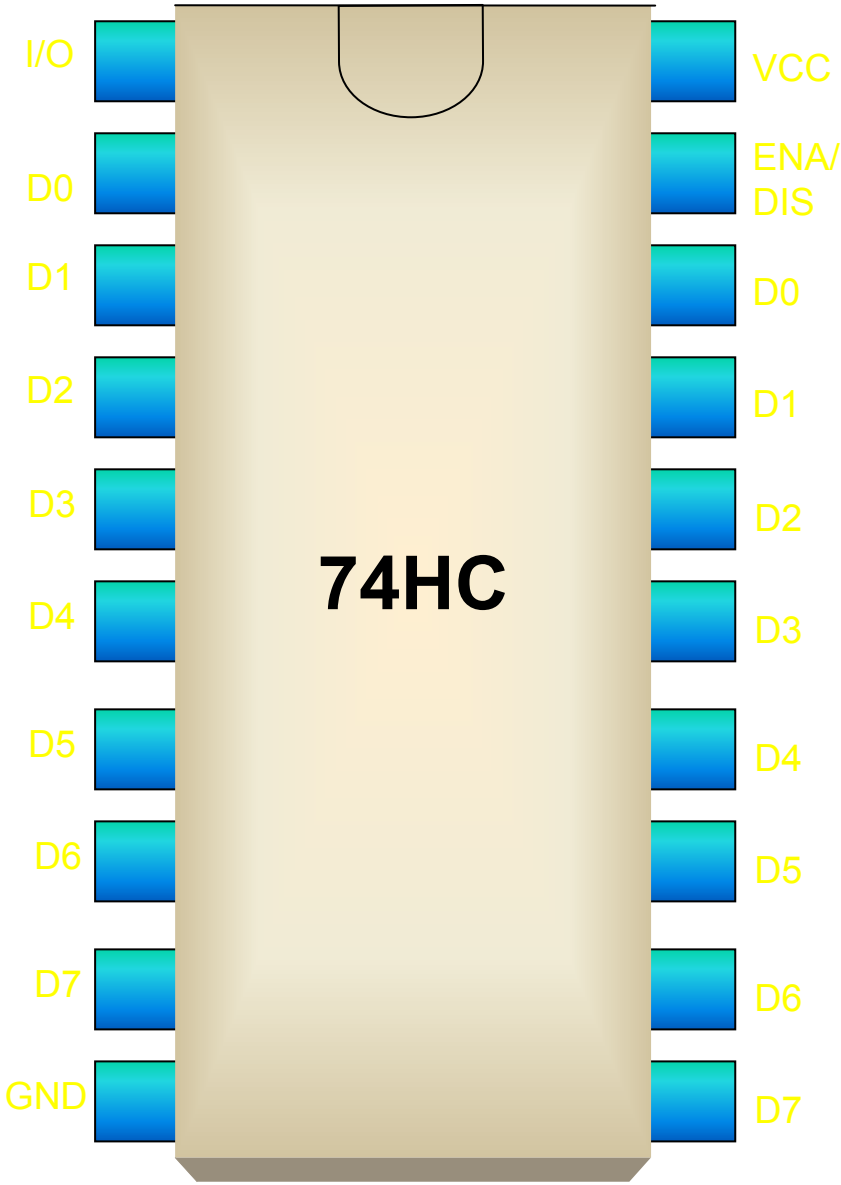


El desarrollo de hardware:
Ya lo ha conseguido

DEFINICIÓN CLARA DE LA OPERACIÓN QUE REALIZA

INTERFACES ESTANDAR

INPUT/OUTPUT PERFECTAMENTE DEFINIDO

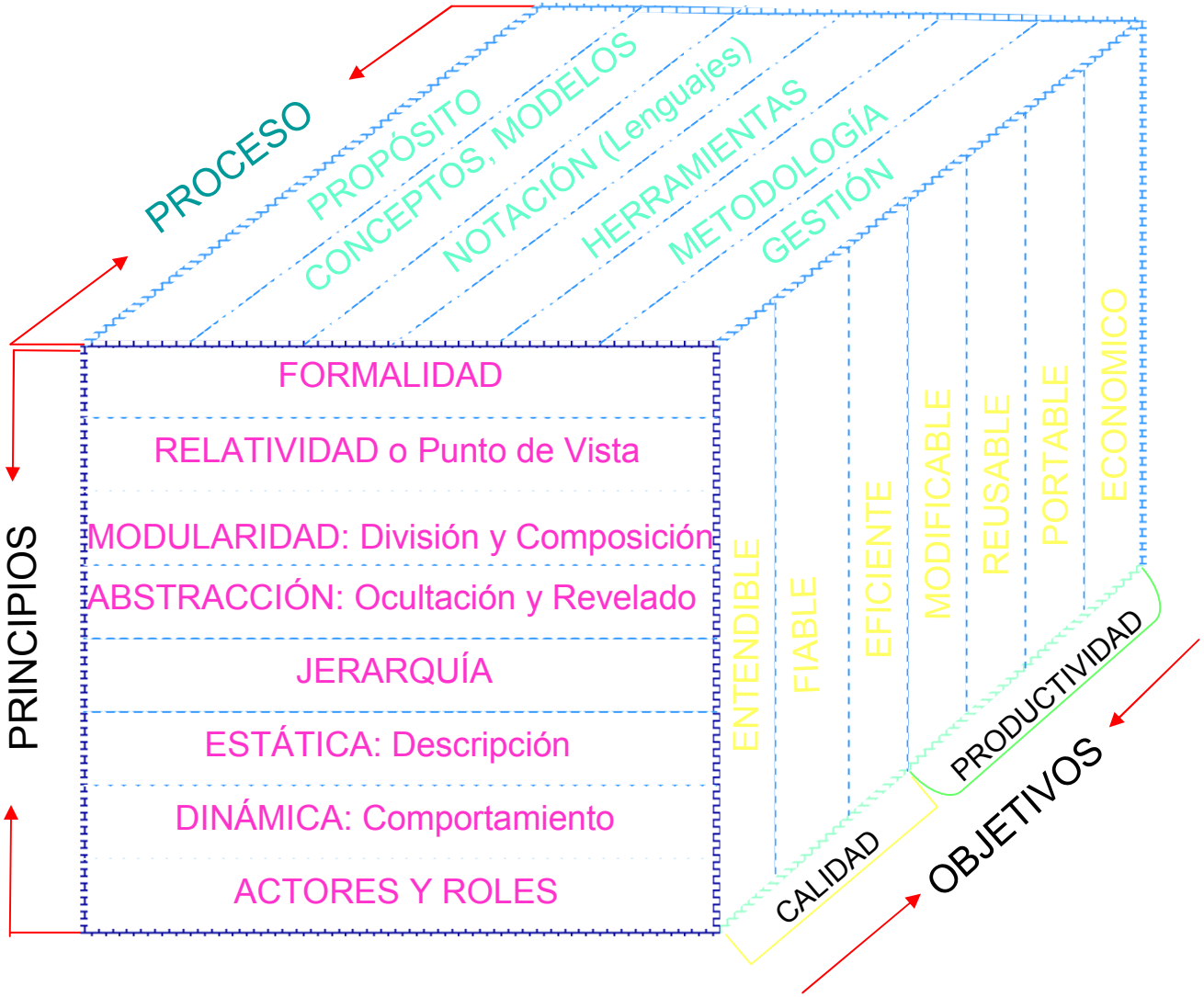


NO SABEMOS NADA DE SU INTERIOR

USADO EN MULTITUD DE APLICACIONES

MUCHOS FABRICANTES

Bloques arquitectónicos en la ingeniería del software



Modelos

- Un modelo es una representación simplificada de la realidad (objeto, acontecimiento, etc.) o de una abstracción, bajo un punto de vista.
- En programación se utilizan distintos modelos o paradigmas para representar un programa (modelo funcional, modelo de objetos, etc.).

Punto de vista o sistema de referencia

- La relatividad no es exclusiva de las ciencias físicas. Un cuerpo puede estar en reposo o en movimiento según el sistema de referencia del observador.
- Toda abstracción se hace desde un punto de vista.
- Un programa de ordenador lo ven de forma distinta el usuario, el diseñador, el programador o el vendedor.
- Para evitar confusiones y malas interpretaciones, **indicar explícitamente el sistema de referencia o punto de vista** bajo el cual están observando o describiendo un acontecimiento o un sistema.

Roles

- Un persona, organización o cualquier objeto activo puede desempeñar distintos cometidos o roles.
- Un rol se especifica por un conjunto de reglas de comportamiento a seguir.
- Un programador puede actuar de usuario del programa que está desarrollando, de diseñador, de probador, etc.
- Para evitar confusiones y malas interpretaciones, **indicar explícitamente el rol que un objeto activo desempeña en cada momento**.

MATEMÁTICAS

Conjuntos

- Números naturales: N = (0, 1, 2, ….)
- Colores primarios: C= (rojo, verde, azul)
- **Constante**: elemento de un conjunto. Ej: 3
- **Variable**: “almacén” que puede contener un elemento de un conjunto. Ej.: nX
- **Producto cartesiano**: N x C = ((0,rojo), (0,verde), ….(2,rojo),….)
- Mapeo: N x C -> N

Funciones

- **Función**: Correspondencia de cada elemento de un conjunto A (**dominio**) con un único elemento de un conjunto B (**imagen**). f: A->B ó y = f(x) siendo

  $x \in A,\ y \in B$

- **Grafo de una función**: $f^* = \{(a,b) \mid a \in A, b = f(a)\}$
- El grafo de una función es un subconjunto de AxB y por tanto se puede representar con el **diagrama de coordenadas** de AXB.
- **Función recíproca**: f^{-1}: B-> A
- **Función composición**: Si f:A->B y g:B->C la función compuesta (g.f):A->C es (g.f) ≡ g(f(a))

Ejemplo Producto Cartesiano y Notación de Funciones

Relaciones, Proposiciones y Predicados

- Una relación R entre A y B es R=(A,B, P(x,y))

siendo:

  - A y B conjuntos
  - P(x,y) un enunciado formal o predicado tal que P(a,b) es verdadero o falso para todo par ordenado(a,b) de AxB.

- Ejemplo:
  - A = Escritores, B= obras, P(x,y) = x escribió y
  - P(Cervantes, Quijote) = < Cervantes escribió el Quijote> Proposición verdadera aRb
  - P(Cervantes, Hamlet) = < Cervantes escribió el Hamlet > Proposición falsa a no R b
- Problema: si x, y son números reales F(x,y) = 0, es o no una función y = f(x)

Clases de Relaciones

- Relaciones de equivalencia
- Relaciones de orden (total, parcial)

Relación de Equivalencia

- Una relación R en un conjunto A es una **relación de equivalencia** si:

para todo a $\in$ A, para todo b $\in$ A, para todo c $\in$ A,

  - R es **reflexiva**: (a,a) $\in$ R
  - R es **simétrica**: (a,b) $\in$ R implica (b,a) $\in$ R
  - R es **transitiva**: (a,b) $\in$ R y (b,c) $\in$ R implican (a,c) $\in$ R

- Una relación de equivalencia **particiona** un conjunto en subconjuntos que son las partes del conjunto original. (Ej.: relación : tienen el mismo color)

Relación de Orden

- Una relación R en un conjunto A es una **relación de orden parcial** si: para todo a $\in$ A, para todo b $\in$ A, para todo c $\in$ A,
  - R es **reflexiva**: (a,a) $\in$ R
  - R es **antisimétrica**: (a,b) $\in$ R y (b,a) $\in$ R implica a=b
  - R es **transitiva**: (a,b) $\in$ R y (b,c) $\in$ R implican (a,c) $\in$ R
- Se llama orden parcial porque **algunos elementos de A pueden no ser comparables**
- Ejemplo: Sea R el orden parcial en V = {1,2,3,4,5.6} definido por
- < x divide a y >, los elementos 3 y 5 no son comparables. Ej.: Las instrucciones, funciones en un **programa concurrente -no determinístico-**.
- **Relación de orden total**: Es una relación de orden parcial en la cual todos los elementos son comparables. Ej.: Las instrucciones, funciones en un **programa secuencial -determinístico-**.

Álgebra

Un álgebra se compone de:

- Un **conjunto de elementos**. Ej.: Números naturales, (Booleanos, Conjuntos, Figuras geométricas etc.).
- Un **conjunto de operaciones** sobre los elementos. Ej.: suma, resta, multiplicación, división, exponenciación, radicación, (AND, OR, NOT, Unión, Intersección, Complemento, Mover, Dibujar, etc.).
- Un **conjunto de ecuaciones** (leyes) que definen la estructura matemática Ej.: conmutativa, asociativa, distributiva, elemento neutro, elemento simétrico, etc.

Álgebra de proposiciones

- **Enunciado simple o aserción**: p = < el sol brilla>
  q = < hace calor>
- **Predicado:** < el sol brilla> = verdadero
- **Enunciado compuesto:** enunciados simples unidos por los operadores lógicos (and $\wedge$, or $\vee$, not $\sim$)

  Ejemplo: r = <el sol brilla> **y** <hace calor>; r = p $\wedge$ q
- **Enunciado condicional:** <Si p entonces q> p -> q
- **Enunciado bicondicional:** <p si y solo si q> p <-> q

GRAFOS

“Grafos” en ingeniería

- Diagrama de flujo (flowchart)
- Diagrama/máquina de estados
- Arboles organizativos
- Diagramas Pert
- Redes de Petri
- Flujo de datos
- Redes Semánticas
- Redes Telefónicas (Red conmutada básica, etc)
- Redes informáticas (Red de área local, red de datos, etc.)
- Redes de Información (Internet, Red IP, etc.)

Grafo dirigido etiquetado

- Un **grafo dirigido finito** (digrafo) **etiquetado** se define por la tupla:
- G = <N, A,E,F> siendo
- N: Un conjunto finito no vacío de nodos
- A: Un conjunto de arcos o pares ordenados $A \supseteq NxN$ (aij=(ni,nj)
- E: Conjunto finito no vacío de etiquetas
- F: Una función de (1, 2,…,h) en E.

  - f(i) es la etiqueta de (ni,nj)

Caminos y clase de nodos

- **Camino o ruta**: Secuencia finita y no vacía de arcos
- **Longitud de ruta**: Número de arcos de la ruta
- **Nodo no terminal**: Tiene arcos que salen de él.
- **Nodo terminal**: No tiene arcos que salen de él.
- Nodos operacionales:
  - Nodos OR, AND, NOT
  - Nodos Suma, Resta, etc.

Autómatas Finitos

- Un **autómata finito** M sobre un alfabeto A, se define:
- M= (S, A, m, si, F) siendo:
  - S: conjunto no vacío de estados
  - A: Alfabeto finito de entrada
  - m: mapeo de S x A en S (define la transición)
  - si: estado inicial
  - F: conjunto de estados finales

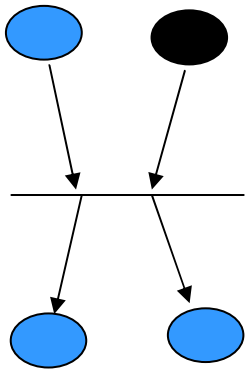
Redes de Petri

- Una **red de Petri Generalizada** es una tupla: R= <P, T, e, s> donde:
- P es un conjunto finito y no vacío de **lugares**
- T es un conjunto finito no vacío de **transiciones**
- $P \cap T = \varnothing$
- e: PxT ->N es la función de incidencia previa (**arcos de entrada** a las transiciones)
- s: TxP ->N es la función de incidencia posterior (**arcos de salida** de las transiciones)
- Una **red de Petri marcada** es <R, M0> siendo
- R una red de petri generalizada y M0 un **marcado** inicial
- Red de Petri: si la etiqueta de los arcos es -1, +1 (implícitas).

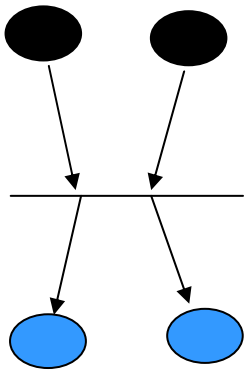
Redes de Petri ordinaria y generalizada

Regla de disparo

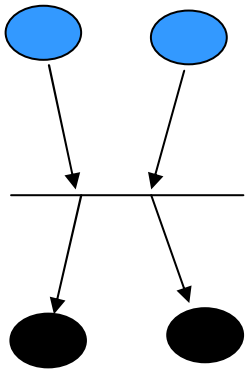
- Una **transición** esta **cargada (enable)** si todos los lugares de entrada tienen al menos tantas marcas como indiquen sus arcos respectivos.
- Si una transición está cargada, se puede **disparar (fire)**, restando tantas marcas a los lugares de entrada, como indiquen sus arcos respectivos, y sumando tantas marcas a sus lugares de salida, como indiquen sus arcos respectivos.



Transición no cargada



Transición cargada antes del disparo



Transición cargada después del disparo

Disparo en Redes de Petri Generalizadas

Subredes, Red Inversa y Red Dual

Conflictos y bloqueos

Concurrencia

- Procesos
- Mecanismos de sincronización
- Propiedades: viveza, limitación, alcanzabilidad, bloqueos, etc.

Algunas formas de sincronización

Propiedades

Red de Petri Interpretada

- Los **lugares** representan **condiciones** (Ej.: estados, eventos, etc.)
- Las **transiciones** representan **acciones** (Ej.: instrucciones, funciones, tarea, etc)
- Las **marcas** representan que **se cumplen las condiciones** de los lugares

Ejemplo Productor-Consumidor

Grafo de marcados

Diagrama de Tiempos

Ecuaciones matriciales

Reducción de la red

LENGUAJES

Lenguajes

- **Natural** (hablado, escrito, mimos, etc.)
- **Requisitos** (natural escrito, gráfico, etc.)
- **Especificación** (natural escrito, gráfico, etc.)
- **Diseño** (organigramas, diagramas de estados, UML -Unify Modelling Languge-, etc.
- **Programación (**Ensamblador, Fortran, Algol, Pascal, Lisp, Prolog, C, C++, Java, etc.).
- **Pruebas** (natural escrito, etc.)

Lenguaje

- **Alfabeto o vocabulario** es cualquier conjunto finito de símbolos.
- **Sentencia** sobre un alfabeto es cualquier cadena de longitud finita compuesta de símbolos de un alfabeto.
- **Lenguaje** es cualquier conjunto de sentencias sobre un alfabeto.
- **Metalenguaje:** Lenguaje usado para definir otro lenguaje

Gramática

- G= (Vn, Vt, P, S)
- Vn: Variables o no terminales
- Vt: Terminales (**Tokens** e Identificadores)
- S: Símbolo de comienzo
- P: Producciones son relaciones entre varias cadenas de variables y terminales

Símbolos, Tokens e Identificadores del lenguaje

- **Símbolos:**

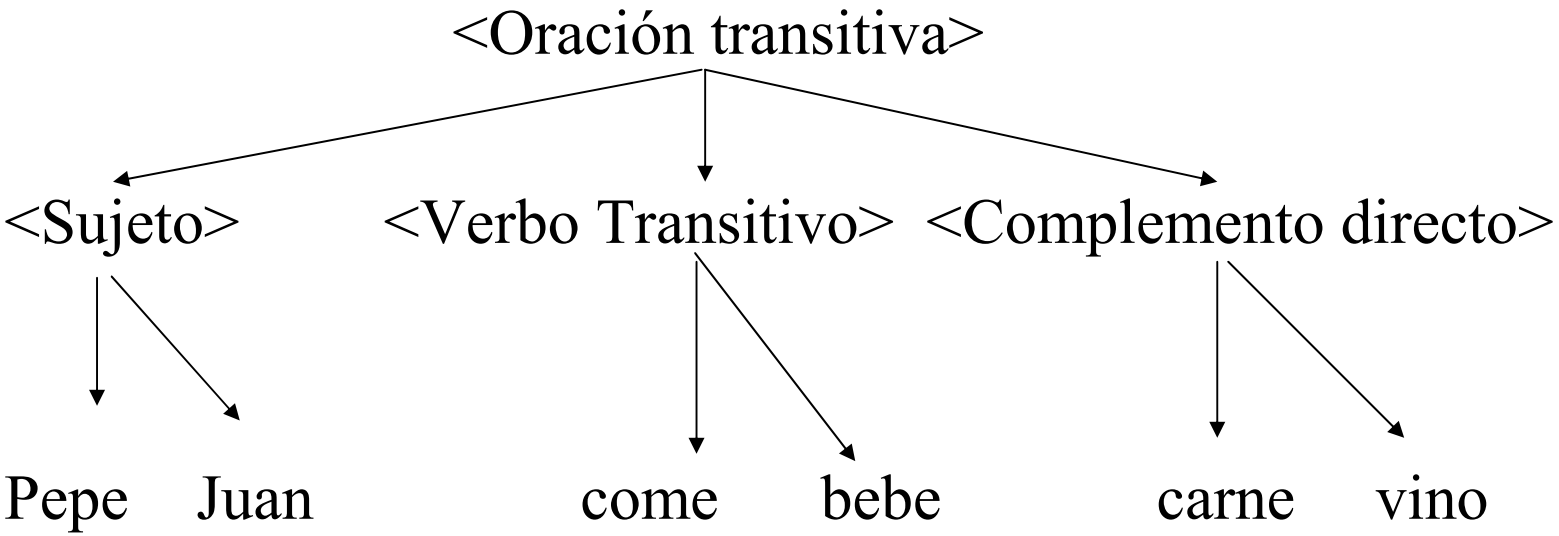
  Ej.: . , ; : = < > ( ), etc.

- **Tokens:** Palabras clave o reservadas del lenguaje.

  Ej.: if, then, else, array, etc.

- **Identificadores:** Nombres de las constantes, variables, funciones, etc.

Lenguaje en notación Grafo



Sentencia:

Pepe come carne

Diagrama Sintáctico

Metalenguaje BNF (Backus Normal/Naur Form)

Metasímbolo	Interpretación
• < P >	producción
• ::	**se define como**
• <P> <Q>	producción P **y (seguida de)** producción Q
• <P>| <Q>	producción P **o** producción Q
• { }	conjunto de (puede ser vacío)

Sintaxis del Lenguaje en notación BNF

<Oración transitiva> :: <Sujeto> <Verbo transitivo> <Complemento directo>

<Sujeto> :: <Pepe> | <Juan>

<Verbo transitivo> :: <come> | <bebe>

<Complemento directo> :: <carne> | <patatas> |<Carne y patatas>

Sentencias sintácticamente válidas:

Pepe come carne Semánticamente válida

Pepe bebe carne Semánticamente inválida

Sentencias sintácticamente inválida:

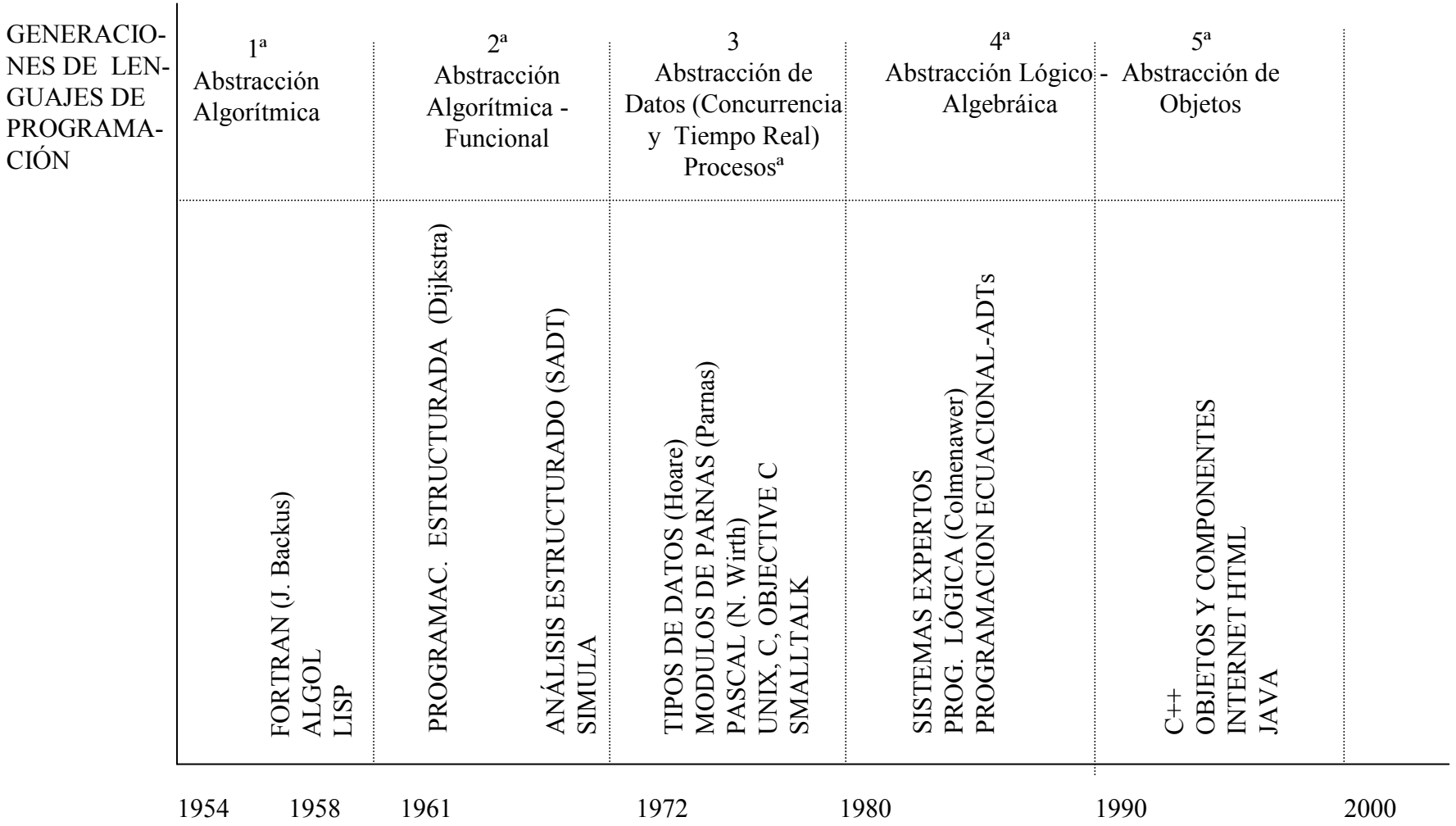
Pepe es alto

Scanner y Parser

- **Scanner**: programa o módulo que **analizador de léxico** y cada vez que se invoca devuelve a partir del estado en curso, el siguiente símbolo, token o identificador del programa fuente que está analizando y actualiza el estado a la siguiente posición.
- **Parser:** programa o módulo que comprueba la **validez sintáctica** del símbolo, token o identificador, según las reglas definidas generalmente en notación BNF.
- Desde el Parser se suele invocar las funciones para construir la **tabla de símbolos**, el **tratamiento de errores**, las **comprobaciones semánticas** y la **generación de código**.
- UNIX dispone de las herramientas **Lex** (analizador de léxico) y **Yacc** (generador de Parsers) muy útiles para la generación semiautomática de parsers.

PRINCIPIOS DE INGENIERÍA SOFTWARE

Historia de la Programación



Notas: 1) las tecnologías y lenguajes han seguido una curva de Gauss. 2) Por la razón anterior se han solapado en el tiempo.
3) El diagrama no expone toda la historia de la programación sino una idea de los acontecimientos más importantes.

Ciclo de Vida y sus problemas a través del tiempo

Arquitecturas de ordenadores

- - Maquinas von Newman
- - Data Flow (pipelined)

Sistemas distribuidos/Redes

- Arquitecturas:
  - Cliente-Servidor
  - Agente-Manager (TMN)
- Entornos
  - Java

Sistemas operativos

- Comerciales
  - MS DOS
  - MS Windows 3.x
  - MS Windows 95/98
  - MS Windows NT
  - UNIX
  - UNIX bajo X11, Motiv, Solaris
- Amper

Paradigmas de programación

• Orientada a Procedimientos	Algoritmos
• Orientada a Datos	Tipos
• Orientada a Funciones	Funciones
• Orientada a Procesos	Procesos y comunicación
• Orientada a Limitaciones	Relaciones Invariantes
• Orientada a la Lógica	Objetivos
• Orientada al Álgebra	Ecuaciones
• Orientada a Reglas	Reglas If then
• Orientada a Objetos	Clases, Objetos y Operaciones

Topología de la primera y comienzos de la segunda generación de los lenguajes de programación

Programación estructurada

Programa = Control + Datos

“ Cada bloque con una entrada y una salida”

“La sentencia goto es innecesaria”

Dijkstra

- Programación estructurada:

Teoría que persigue para los flowcharts (programas) :

- Entender y modificar los programas propios y ajenos
- Notación formal para ordenar el pensamiento del programador
- Probar la corrección
- Refinamiento por pasos (Abstracción y descomposición)
- Convertir en formas estándar
- Representar los programas por iteración y anidamiento de un número pequeño de estructuras de control estándar y básicas (componentes de control).

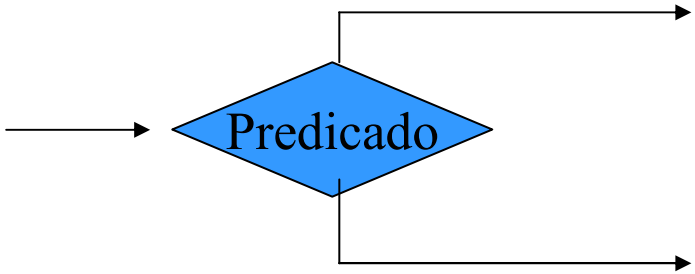
En otras palabras...

- Identificación de las Estructuras de Control mínimas
- Transformación de programas
  - Equivalencia y reducibilidad de estructuras
  - Refinamiento
- Estudio de las Propiedades de un programa (completitud, corrección, terminación).

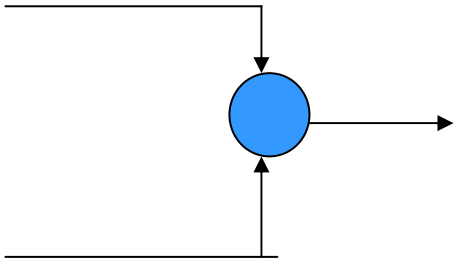
Nodos Fundamentales

- Secuencia

- Selección

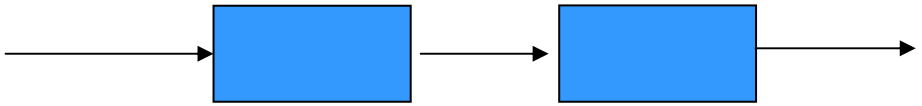


- Iteración

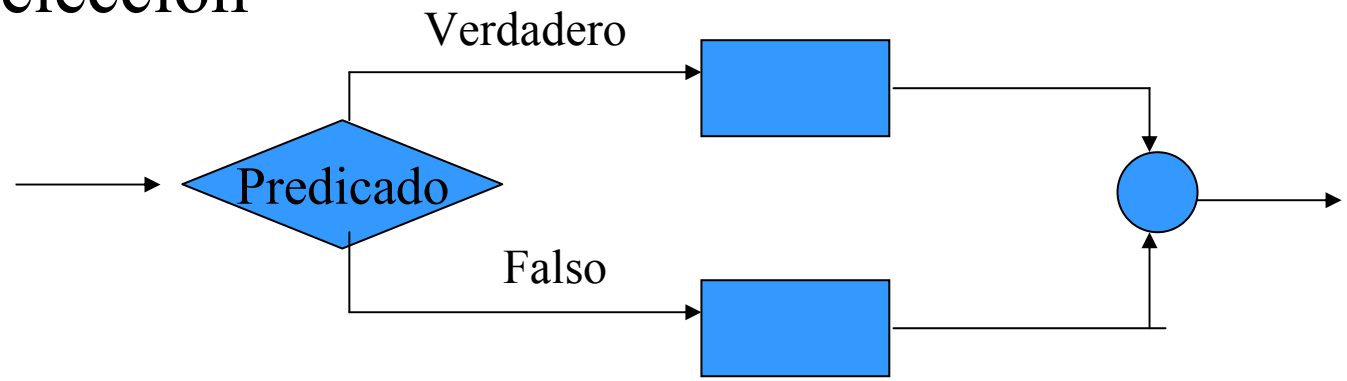


Estructuras de control
Básicas o atómicas (Böhm-Jacopini)

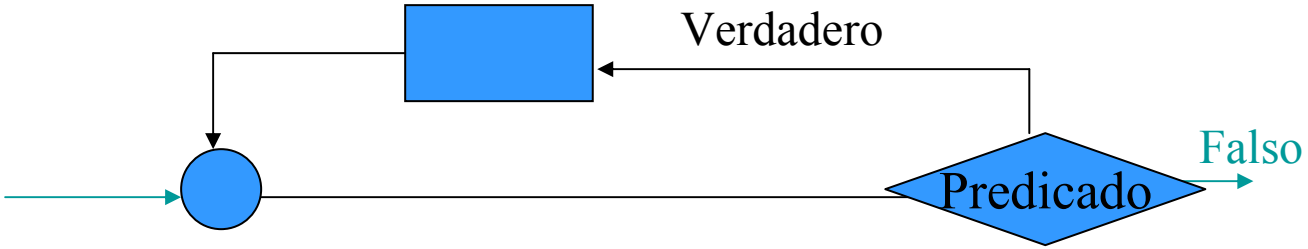
- Secuencia



- Selección



- Iteración



Estructuras de control Prácticas

- Secuencia
- Decisión
  - if expresión lógica then acción else acción end if
  - if expresión lógica then acción end if
  - Select variable case valor1 acción, case valor2 acción ... else acción end Select
- Iteración
  - For variable= valor inicial, valor final, incremento, acción, end for
  - While expresión acción end while
  - Do acción Until expresión
- Otras
  - Exit
  - Goto

Topología de finales de la segunda y comienzos de la tercera generación de los lenguajes de programación

TIPOS DE DATOS

A. Hoare

Datos e Información

- **Datos** son las representaciones físicas de nuestras abstracciones.
- **Información** es el significado que asignamos a los datos.
- Un dato se define por el par (nombre-identificador, valor)
- Si el valor es único el dato es una **constante**.
- Si el dato puede tomar un conjunto de valores, el dato es una **variable**.
- En ordenadores el termino constante y variable se refiere al nombre o identificador del dato, o al almacén asociado con el dato.

Estructuras de Datos, Tipos de Datos y Tipos Abstractos de Datos

- Estructura de datos: Como los datos están organizados
- Tipo de Datos: Los valores de los datos
- Tipos Abstractos de Datos (ADTs): Operaciones a realizar con los datos

Tipo de Datos

- Es el **conjunto "finito" de valores que el dato puede tener**.
- El tipo de dato puede ser referido por un identificador
- Ej.: int = (mín,…, -2,-1,0,1,2,…,máx)

Clasificación de los Tipos de Datos

- Tipos de datos **estándar** (integer. Array,…)
- Tipos de datos **definidos por el usuario**
- Tipos **primitivos**: Se definen por enumeración de sus valores o por la definición de su rango.
- Tipos **estructurados**: Se definen en términos de los tipos primitivos.

- Tipos **no estructurados**
  - Tipos **estándar** (Boolean, Integer, Real, Character)
  - Tipos **definidos por el usuario** o enumerados(Escalares, Subrango)
- Tipos **estructurados**
  - estándar (String ,Array, etc.)
  - **definidos por el usuario** Record, Secuencia (Streams, Listas, Pilas, etc)

Formas de representar un Dato

- Los enteros por 16 ó 32 bits
- Los booleanos como el subconjunto (0, -1) de los enteros
- Las cadenas como arrays de caracteres

Operaciones

- Son las reglas usadas para la manipulación de los datos.
- Una operación relaciona un conjunto finito de datos de entrada con un conjunto finito de datos de salida.

Formas de representar una operación

En los lenguajes de programación, las operaciones se representan por:

- Operadores (+, -, *. /, …)
- Sentencias o instrucciones (add, if, …)
- Funciones (sqrt, sin, create…)

Formas de definir una operación

Una operación se puede definir por:

- Enumeración de los valores de salida para todas las combinaciones de los valores de entrada
- Reglas computacionales que representen una secuencia finita de operaciones simples
- Aserciones sobre el tipo de sus variables y las relaciones entre los valores antes y después de la ejecución de la operación.

Dualidad Estructuras de Control y Estructuras de Datos

Estructura de control	Estructura de Datos
• Sentencia compuesta	Producto cartesiano
• Sentencia condicional	Tipo Union
• Sentencia iterativa limitada	Array, set
• Sentencia iterativa ilimitada	Sequence
• Abstracción de Procedimiento	Abstracción de datos
• Procedimiento recursivo	Tipo de dato recursivo
• Sentencia goto	Pointer

Topología de finales de la tercera generación de los lenguajes de programación

Tipo Abstracto de Datos

- Es el tipo de dato (conjunto de valores) y las operaciones que se pueden definir sobre el tipo
- En muchos casos, el tipo de dato y el tipo de dato abstracto se consideran sinónimos.

Bases de datos:

- Relacionales
  - Tablas y Relaciones
  - Lenguaje SQL
- Orientadas a Objetos

Programación por eventos

- Paradigma para la ejecución multitarea.
- Las tareas son desencadenadas por el sistema operativo como respuesta a eventos o mensajes del sistema o del usuario.
- Las tareas pueden estar programadas usando la programación secuencial (procedural u orientada a objetos).
- Aplicación: Interfaz de usuario

Eventos y Mensajes

- **Un evento es una ocurrencia en el tiempo**. Ej.: Mover el ratón, pulsar tecla, etc.
- **Un evento se concreta en un mensaje con: nombre, parámetros, etc**.
- Los mensajes se almacenan en Colas de mensajes gestionadas por el en el Sistema Operativo.
- El Sistema Operativo despacha tareas como respuesta a ciertos mensajes.
- Otros mensajes se envian al programa del usuario a los Manejadores de eventos para que se ejecute la acción correspondiente.

INTELIGENCÍA ARTIFICIAL Y PROGRAMACIÓN MATEMÁTICA

Inteligencia artificial:

- Programación Lisp (Programación por listas)
- Sistemas Expertos (Programación por reglas)

Distintas formas de representar expresiones lógicas y aritméticas

Programación matemática

- - Programación lógica (Prolog)
- - Ecuacional o algebraica (Abstract Data Types -ADTs-)
- - Programación Funcional (John Backus)

Especificación Textual

Especificación Lógica

Especificación Algebraica (I)

Especificación Algebraica (II)

Topología de aplicaciones pequeñas y medianas usando lenguajes de programación orientados a objetos y basados en objetos

Topología de aplicaciones grandes usando lenguajes de programación orientados a objetos y basados en objetos

OBJETOS

¿Qué modelo de Objetos?: C++, Java,
ISO, UIT, UML, etc.

Orientación a Objetos

Principio de ocultación de la información:

"El desarrollo del software se realiza más rápidamente, cuando los programadores conocen lo menos posible de lo que cada uno de ellos hace"

D. Parnas 1971

Encapsulado

- Los objetos ocultan sus datos y se opera o accede a ellos solo a través de sus propios métodos.

Componentes de un objeto

- **Atributos** - Características/propiedades
- **Relaciones** - Dependencias
- Operaciones/Funciones/**Métodos** - Interfaz/Protocolo
- **Mensajes Salientes**- Llamadas
- **Eventos** - Ocurrencias
- **Excepciones** - Condiciones de error que pueden ocurrir durante una operación o al recibir un mensaje.

Tipos de Comunicación

- Por dato en memoria compartida
- Por llamada de función
- Por mensajes
- Por eventos

Mensaje

- Un mensaje es una comunicación o interacción entre objetos.
- Un objeto **cliente** (el jefe) envía un mensaje a un objeto **servidor** (el currante) que realiza la operación que porta el mensaje.

Excepciones

- Condiciones de error que pueden ocurrir durante una operación o al recibir un mensaje.
- Cuando se detecta una excepción un manejador de excepciones debe responder con una acción apropiada a la excepción.
- El conjunto de excepciones debe ser completo para cada operación.
- Normalmente las excepciones están originadas por operaciones con valores límites de los atributos (división por cero, fichero vacío, no hay memoria, corte de paralelas,etc). (condiciones de contorno)

Clase u Objeto Abstracto

- Definición (plantilla) para objetos
- Ejemplos: persona, coche, mesa
- Lenguaje: Nombres comunes

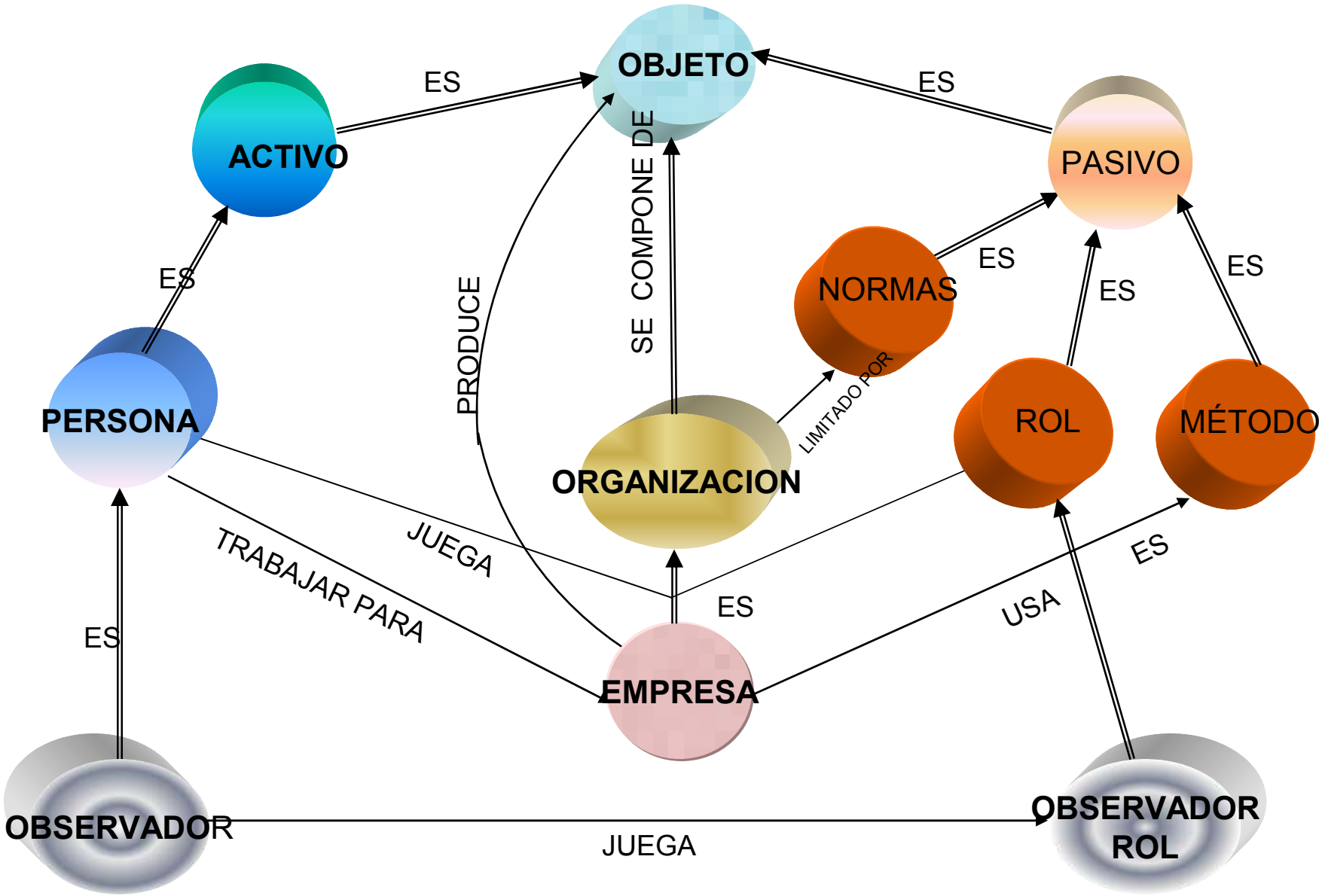
Objeto

- Instancia (variable) de una clase
- Ejemplos: Pepe, Opel Vectra, de Pepe
- Lenguaje: Nombres propios

Descripción abreviada de una clase

• Clase	Coche
• Descripción de la clase	Suministra transporte
• Superclase	Vehículo
• Subclase	Ranchera
• Atributos	Potencia, color, estado
• Operaciones	Arrarcar, acelerar, parar
• Mensajes salientes	Pitido
• Excepciones	Sin gasolina
• Comentarios	Vehículos de motor

OBJETOS Y RELACIONES



Clase de Relaciones

- **Colección** (conjunto de objetos)
- **Instancia** (objeto particular de una clase)
- **Herencia** (simple, múltiple, Todo, nada, selectiva - de atributos y métodos)
- **Polimorfismo** (la misma operación es realizada con distintos métodos. Ej.: calcular el salario de un (objeto) directivo o un (objeto) trabajador)
- **Binding** dinámico (reasignación de un objeto a una clase en tiempo de ejecución.Ej. Dibujar un objeto (triángulo o circunferencia)
- **Comunicación** o interacción
- Otras

Ventajas de la Orientación por Objetos

- Mejor correspondencia entre el mundo real y su modelo Software.
- Mejor entendimiento y mantenibilidad debido a que los objetos y sus operaciones están localizados.
- Fuerte interconexión o acoplamiento (cohesión) dentro de un objeto y débil entre objetos.
- Mejora la reusabilidad de Componentes.

Inconvenientes

- Está en desarrollo (conceptos y terminología en evolución)
- Hay que cambiar la forma de pensar
- Puede tener impacto favorable o desfavorable en espacio y tiempo
  - herencia y binding dinámico
- Herramientas CASE nuevas
- Relativamente poca experiencia
- Estándares de clientes/militares orientados a descomposición funcional

Componentes SW

- Librerías de funciones
  - Estáticas (clásicas: matemáticas,etc.,...)
  - Dinámicas (DLLs, NLMs,...)
- Librerías APIs (Application Programmer Inteface)
  - Interfaz de usuario (VBXs, OCX,...)
  - Librería de comunicaciones
- Librerías de clases
  - OLE, C++, Java

JAVA

Java

- Desarrollo de SUN Microsystems
- **Lenguaje Java:** lenguaje de programación para sistemas distribuidos
- **Java Development Kit** (JDK): Es un conjunto de herramientas de programación
- **JavaOS**: Plataforma para correr Java en una variedad de ordenadores. Máquina Java es una máquina virtual que corriendo en un ordenador, ejecuta por interpretación, el código del programa Java.
- **Java Beans**: Componentes arquitectónicos de Java. Widget o controles GUI, funciones para servicios, applets, aplicaciones Java.
- Externos: Internet Browser (Netscape, MS Internet Explorer,…) para ejecución de applets

Objetivos de diseño de Java

- **Simple**: Simplificación de C y C++
- **Orientado a Objetos**: Casi todo en Java es una clase, un método o un objeto. Solo las operaciones primitivas y tipos de datos (for, while, int, etc.) están a nivel de subobjeto.
- **Portable**: Los programas Java son compilados a un código intermedio que puede ser interpretado en muchas plataformas (Windows 95/98, NT, Solaris, AIX and OS/2).
- **Seguro**: Del ataque de virus, borrado o modificación de archivos, etc.
- **Altas prestaciones**: Compilado en tiempo real y el ejecutable tan eficiente como C++.
- **Multi-Threaded** (hilo): Un programa Java puede tener muchas cosas diferentes procesandose independientemente y continuamente.

¿Que se ha quitado de C?

- **pre-procesador** (#include, #if, #define)
- **goto** (tiene break y continue)
- **typedefs, structs, unions ó enumeración** (se pueden suplir con ventaja por clases)
- **pointers** (por seguridad y portabilidad)
- **funciones** (ni propias ni de librería, se sustituyen por métodos de clases)
- **coerción implícita y casting limitado** (menos propenso a error asegurando typado fuerte, no se pueden hacer cosas ilegales).

Clases de programas Java

- Programas Java propiamente dichos o aplicaciones
- Java Applets
- Java Scripts

Programa Java

- Código que se ejecuta en un entorno Java

Java Applet

- Es un programa Java (clase) que se invoca por un Web Browser (Netscape Navegator, Internet Explorer, etc.) cuando se procesa una página HTML con el tag Applet, el Browser envía un mensaje al Servidor para que envíe el Applet y cuando llega este se ejecuta.

Java Script

- Programa en lenguaje Java Script que está contenido y se ejecuta por interpretación en el entorno de una página HTML.

Algunos Librerías de Clases Java

- **java.lang**: para operaciones básicas y programar concurrencia
  - Clases básicas: numéricas, strings, objects, compiler, runtime, security y **threads**
- **java.awt**: para desarrollo de interfaz de usuario (GUI). Eventos
  - Clases: Window, Frame, Dialog, Button, Checkbox, Graphics, etc
- **java.applet**: permite la construcción de applet
- **java.io**: manejo de ficheros
- **java.net**: para programar aplicaciones (cliente, servidor) en red
  - Clases URL, Socket y ServerSocket
- **java.util**: utilidades misceláneas (fecha, hora, random, sistema,…)

Eventos

- Son objetos equivalentes a los mensajes de MS Windows.
- Constan de:
  - arg (ej. Título del botón)
  - Id (identificador del evento. Ej.: MOUSE_MOVE)
  - targer (objetivo del evento. Ej.: Botón)
  - x, y (coordenadas del ratón)

Threads (hilos)

- Un thread es un **proceso** liviano que opera concurrentemente dentro de un proceso simple, de tal forma que la comunicación y la compartición de datos entre threads en la aplicación, es rápida y eficiente.
- Métodos del thread: New, start, run, stop, sleep, set priority, etc.
- La **sincronización** se basa en el concepto de **monitor** de Tony Hoare.

Socket

- Soporta protocolos: TCP, UDP, IP, ICMP con DNS y NIS (para nombres del host y password)
- Métodos:
  - new ServerSocket: crea un ServerSocket para un puerto
  - accept: acepta una conexión
  - getInetAddress: obtiene la dirección del cliente
  - getOutSteam: realiza una salida de datos al cliente
  - close: cierra el Socket

Continuará …