La Confiabilidad

J. L. Roca

1. Introducción
2. Calidad y Confiabilidad
3. La Confiabilidad de un Software
4. Errores y Fallas
5. Confiabilidad y Complejidad
6. Modelos de Confiabilidad
7. Obtención de Softwares Confiables
8. Metodologías de Trabajo
9. Herramientas
10. Conclusiones

Introducción

- Esfuerzo de Desarrollo de Software 80%.
- Esfuerzo de Desarrollo de Hardware 20%.
- Hace tres décadas esta relación era inversa.
- Se utilizaban para el Hardware técnicas de Control de Calidad por atributos o por variables como ser:
- AOQL (Average Outgoing Quality Level) a la salida.
- AQL (Acceptable Quality Level) a la entrada.
- LPTD (Low Total Percentage Defects).

Introducción

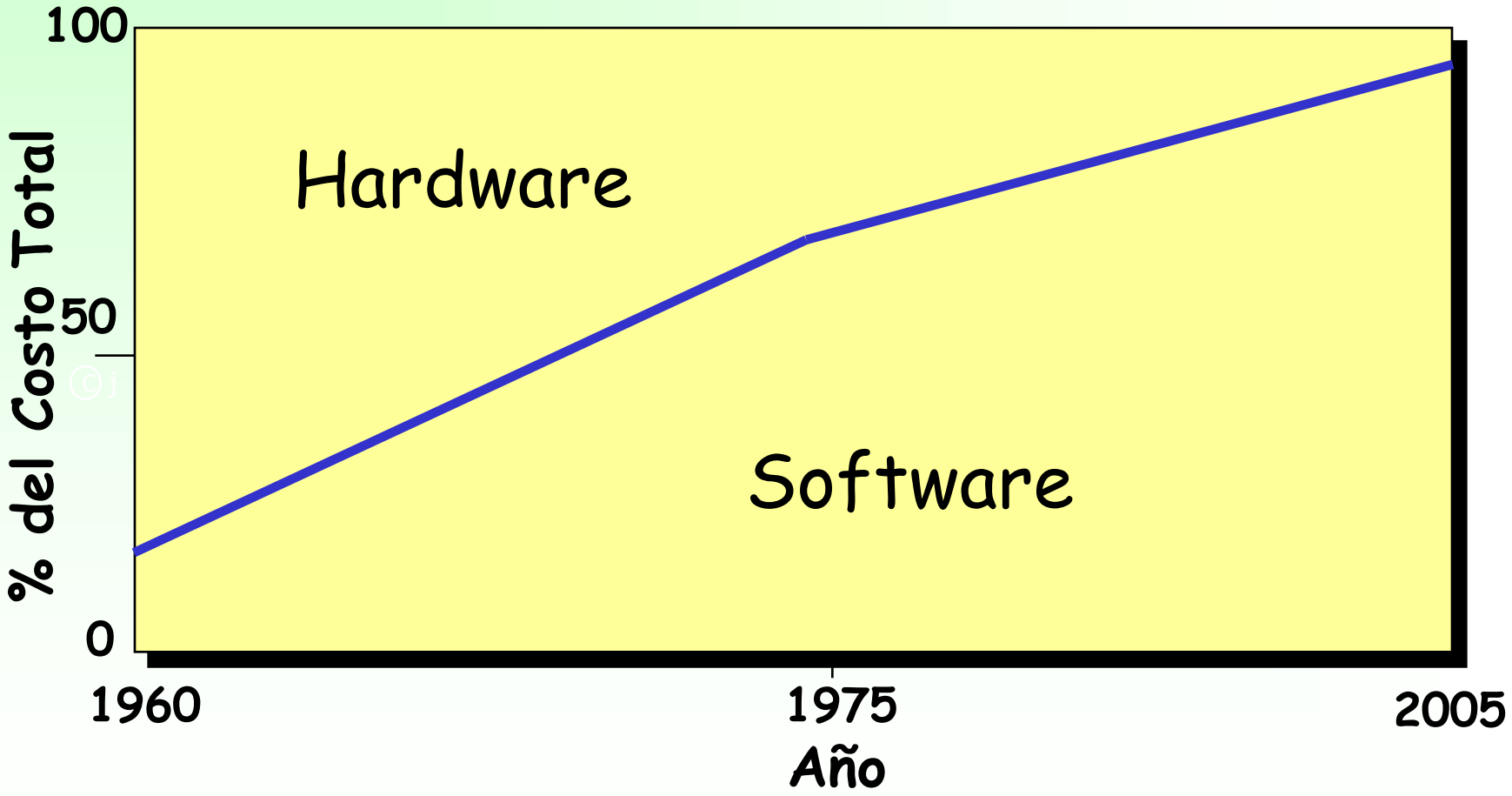
- Actualmente el Control estadístico de procesos:
- SQC (Statistics Quality Control)
- TQM (Total Quality Management)
- Desde hace una década la importancia del software a la hora de diseñar un sistema fue creciendo.
- Pocos utilizan las Herramientas propuestas por la denominada Ingeniería del Software para mejorar la calidad y confiabilidad de un programa.
- Debe hacerse un balance entre Disponibilidad (Availability) y Seguridad (Safety).

Introducción

- Un sistema totalmente Seguro nunca funcionará.
- Un sistema siempre Disponible nunca será totalmente seguro.
- Un "Bug" (error) en un software puede provocar una falla (fault) que termine con una misión espacial.
- O puede implicar vidas humanas cuando la aplicación es electromédica.
- Esta es la importancia que día a día esta teniendo el Software en un sistema.

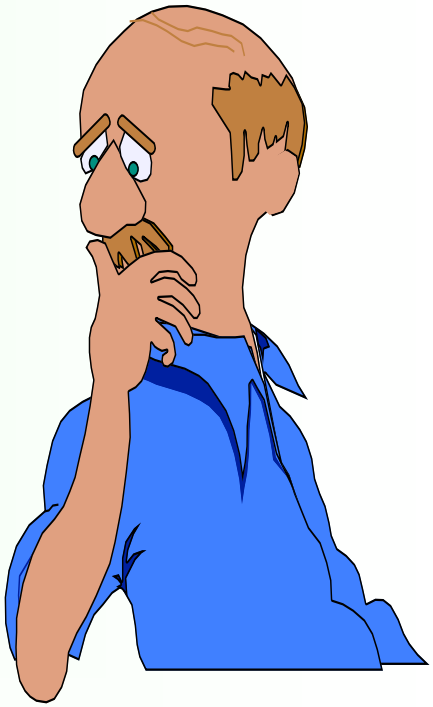
Introducción

Relación Costos Hardware-Software



Calidad y Confiabilidad

- La Calidad es una medida de la performance de un elemento en un punto determinado del tiempo, presumiblemente t=0.
- La Confiabilidad es una medida de performance a lo largo del tiempo.
- Una está relacionada con la variable aleatoria "proporción" la otra con la variable aleatoria "tiempo".
- El Software no admite la misma discriminación.



Calidad y Confiabilidad

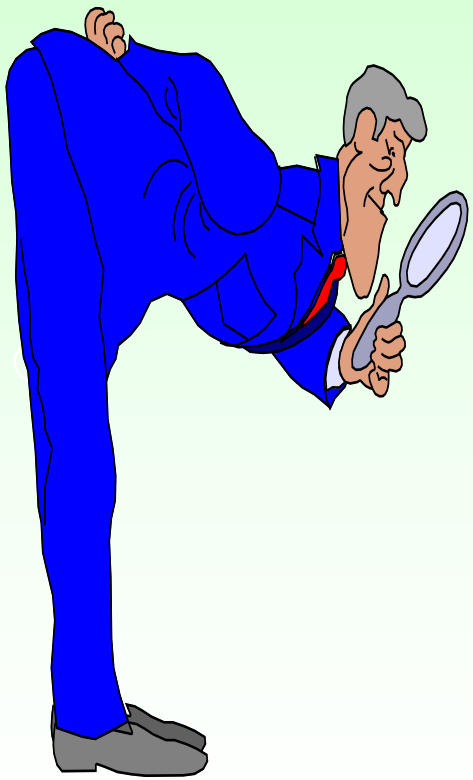
- Es posible hablar de Calidad, expresada ésta como proporción de errores (Bugs) en el programa.
- Mientras la Confiabilidad está expresada en función de la tasa de fallas (Hazard Failure Rate).
- La mayoría de los especialistas hablan de Calidad y Confiabilidad en forma indistinta.
- Lo cierto es que un Software no puede producirse en serie varias veces.
- Una vez desarrollado, probado y liberado, todos los Softwares serán copia de éste, en cualquier sistema que corran.

Calidad y Confiabilidad

- Los errores, no así las fallas, serán siempre los mismos, no importa el entorno en el que corran.

- Por lo tanto, es mucho más correcto hablar de Confiabilidad de un Software que de la Calidad del mismo.

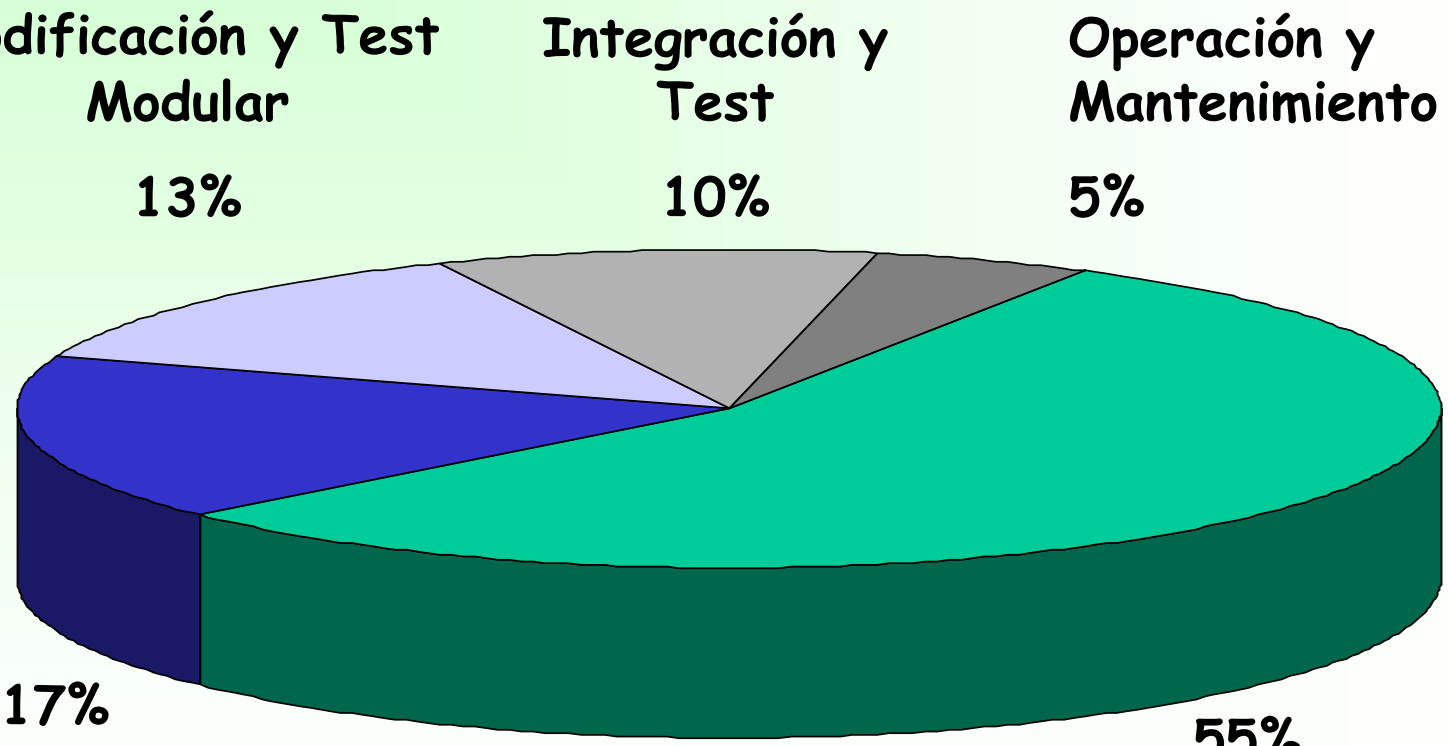
La Confiabilidad de un Software



- Se dice que un Software es confiable si realiza lo que el usuario desea, cuando así lo requiera.
- No es confiable si así no lo hiciera.
- A nuestros fines un Software no es Confiable cuando falla.
- Las fallas se deben a errores en el Software.
- Si corregimos estos errores sin introducir nuevos, mejoramos la Confiabilidad del Software.

La Confiabilidad de un Software

Proporción de Errores de un Software durante su Ciclo de Desarrollo

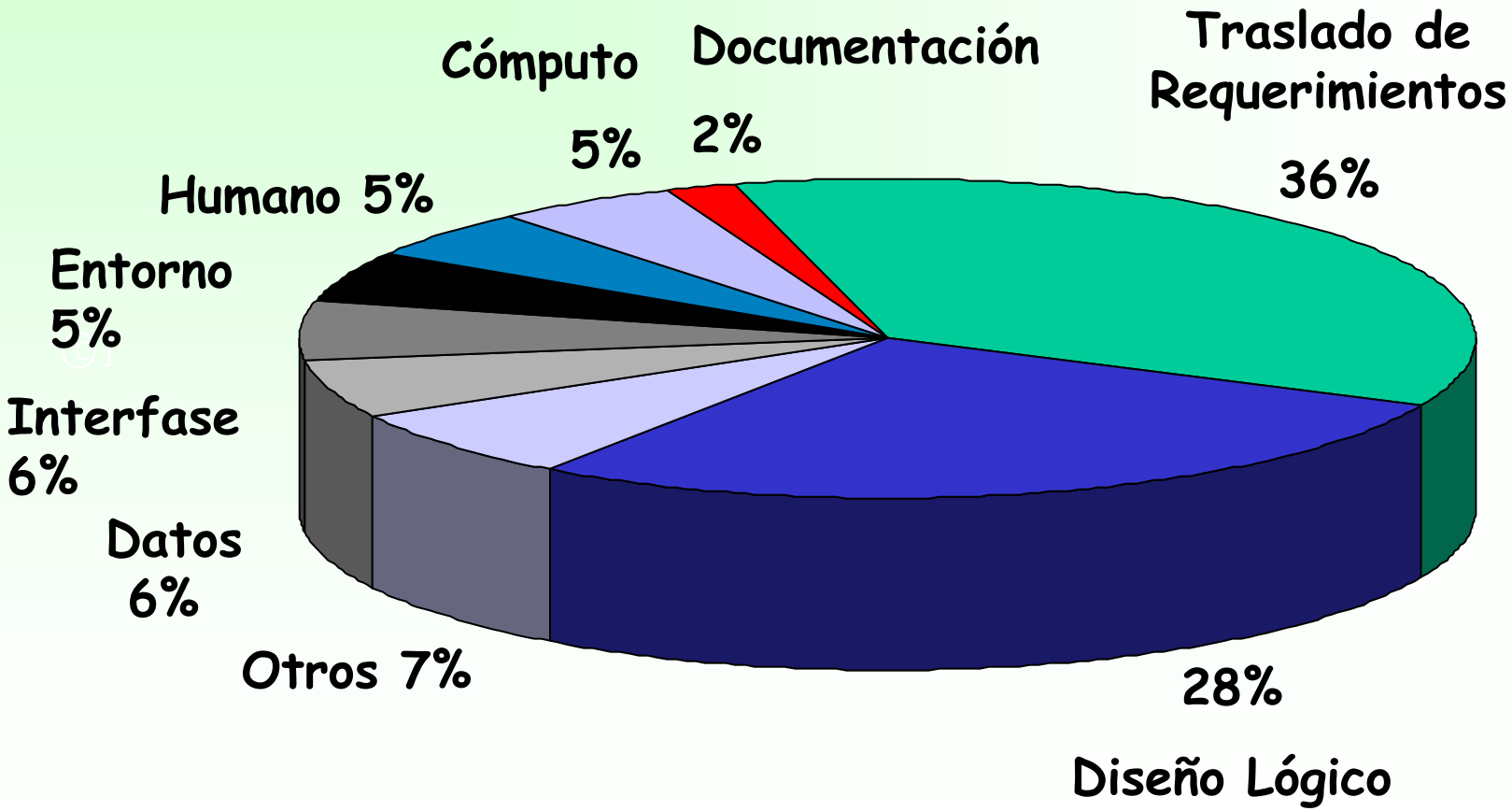


La Confiabilidad de un Software

- El 72% de los errores se originan en: "el traslado de los requerimientos del usuario y en el diseño lógico".
- Podremos aumentar la Confiabilidad de un Software haciendo hincapié en estas dos primeras etapas.
- Fuentes de diversos tipos aseveran que, es en el diseño, en donde debe ponerse énfasis para reducir la proporción de errores.

La Confiabilidad de un Software

Proporción de Errores de un Software por Áreas de Conflicto

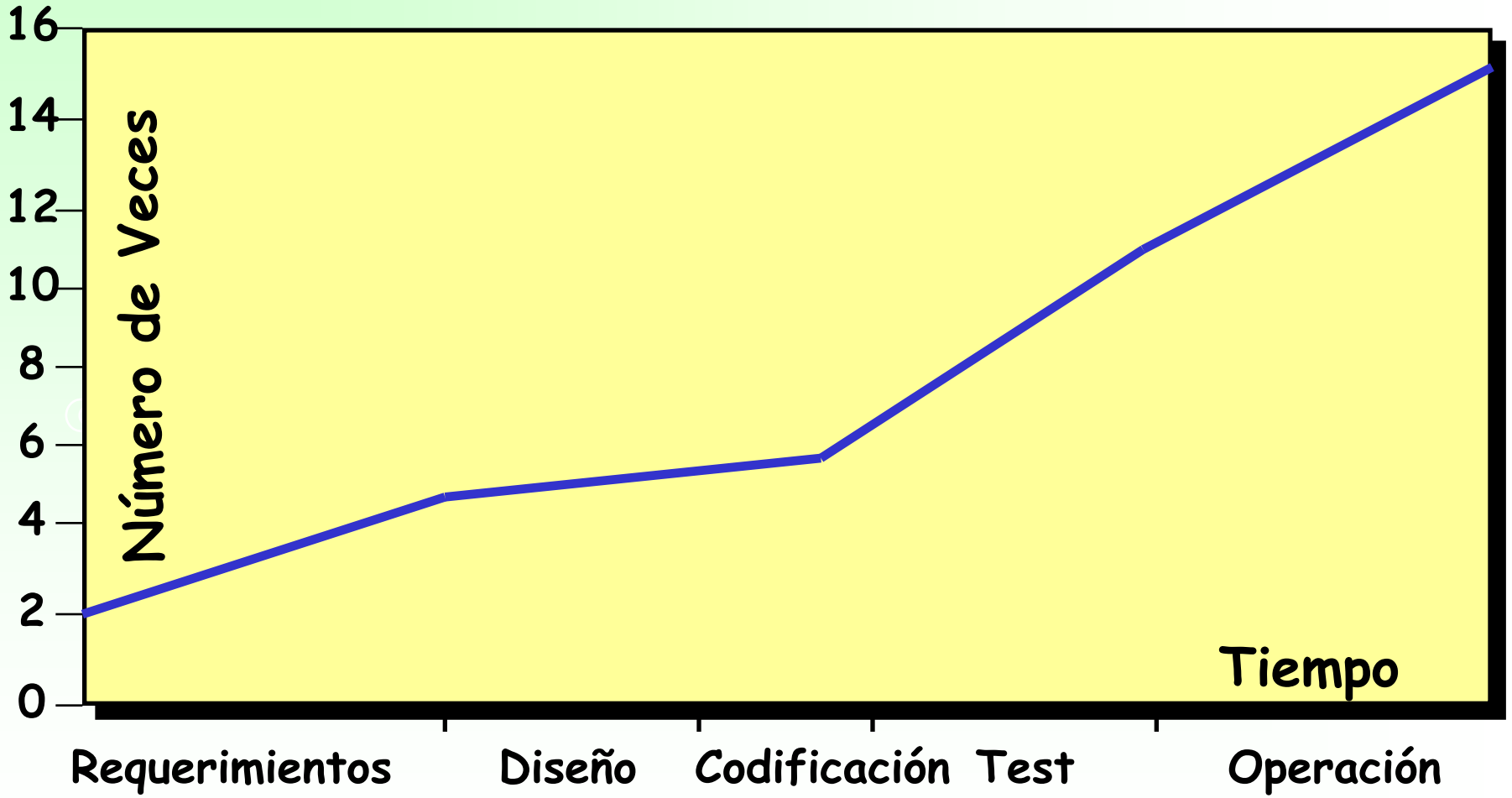


La Confiabilidad de un Software

- Hemos observado nueve categorías en las que se divide la generación de errores.
- La experiencia demuestra que:
- Aproximadamente el 76% de los errores no son descubiertos hasta bien entrada la etapa de pruebas integrales.

La Confiabilidad de un Software

Costos de Corrección de Errores de un Software



La Confiabilidad de un Software

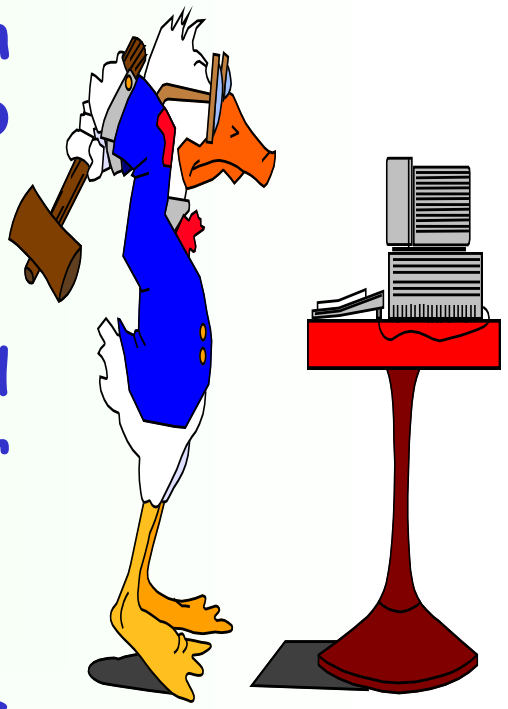
- El costo de detección y corrección de errores durante y después de las etapas de integración y test resultan entre 10 y 15 veces más que en las etapas de desarrollo y codificación.
- Estudios realizados concluyen que el medio ambiente en el que se desarrolla el Software contribuye enormemente al aumento de errores.
- La Confiabilidad del Software pasa a ser un problema de "Management" y no "Técnico"

Errores y Fallas de un Software

- Históricamente, una forma de aumentar la Confiabilidad de un Software era correrlo y probarlo extensivamente antes de liberarlo.

- No es efectivo probar la Confiabilidad en el producto sino hacerla, es decir fabricarla en el mismo.

- La Confiabilidad deberá ser diseñada en el producto.



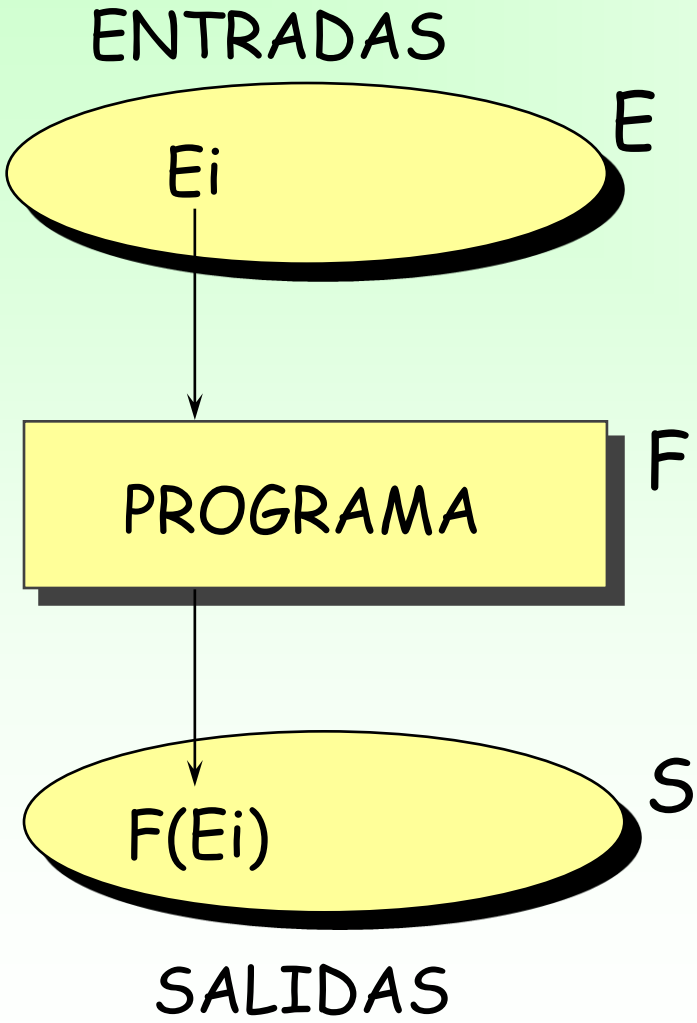
Errores y Fallas de un Software

- Teniendo un 72% de errores generados en el traslado de los requerimientos del usuario, el énfasis debe ser puesto en ese punto.
- Es mucho más efectivo resolver los errores en la misma fase de diseño que en la de prueba.
- Cada vez que se corrige un error se generan nuevos con una cierta probabilidad.

Errores y Fallas de un Software

- Es mucho más costoso encontrar, corregir y documentar errores en los últimos peldaños del ciclo de vida que al comienzo.
- Es necesario utilizar Herramientas, que en base a modelos ayuden a determinar parámetros que sirvan de análisis.
- Las Herramientas son provistas por la asi llamada Ingeniería de Software

¿Porqué Falla un Software?



Esquema de un programa

$E = \{E_i / i = 1,2,....N\}$

N = # de entradas posibles

$F(Ei)$ Real

$\hat{F}(Ei)$ Estimada

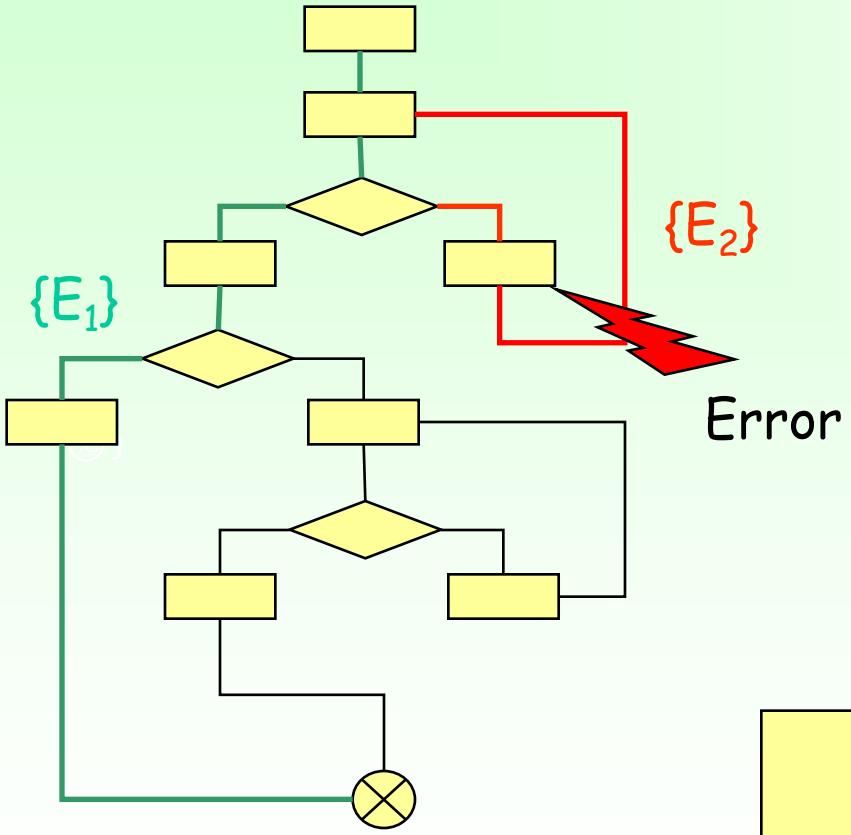
$F(Ei) \neq \hat{F}(Ei)$

FALLA

¿Porqué Falla un Software?

- Un programa establece una correspondencia entre entradas y salidas vía una determinada función estimada o propuesta
- La diferencia entre esta función y la real ejecutada por el programa para un conjunto determinado de datos de entrada, es lo que da origen a la falla.
- Son las entradas las que interactúan con un determinado error en la programación y por lo tanto generan salidas no esperadas (fallas).

¿Porqué Falla un Software?



Esquema de un programa

$E = \{E_i / i = 1,2,....N\}$

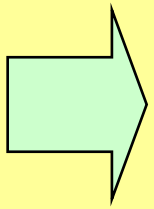
N = # de entradas posibles

La Falla aparece cuando el conjunto de Entradas Interactúa con el Error

La FALLA se Auto expone

¿Porqué Falla un Software?

- Ignorancia de los requerimientos del usuario.
- Ignorancia del entorno en que se utiliza el Software.
- Escaso flujo de información entre usuario y programador
- Escasa documentación del Software.

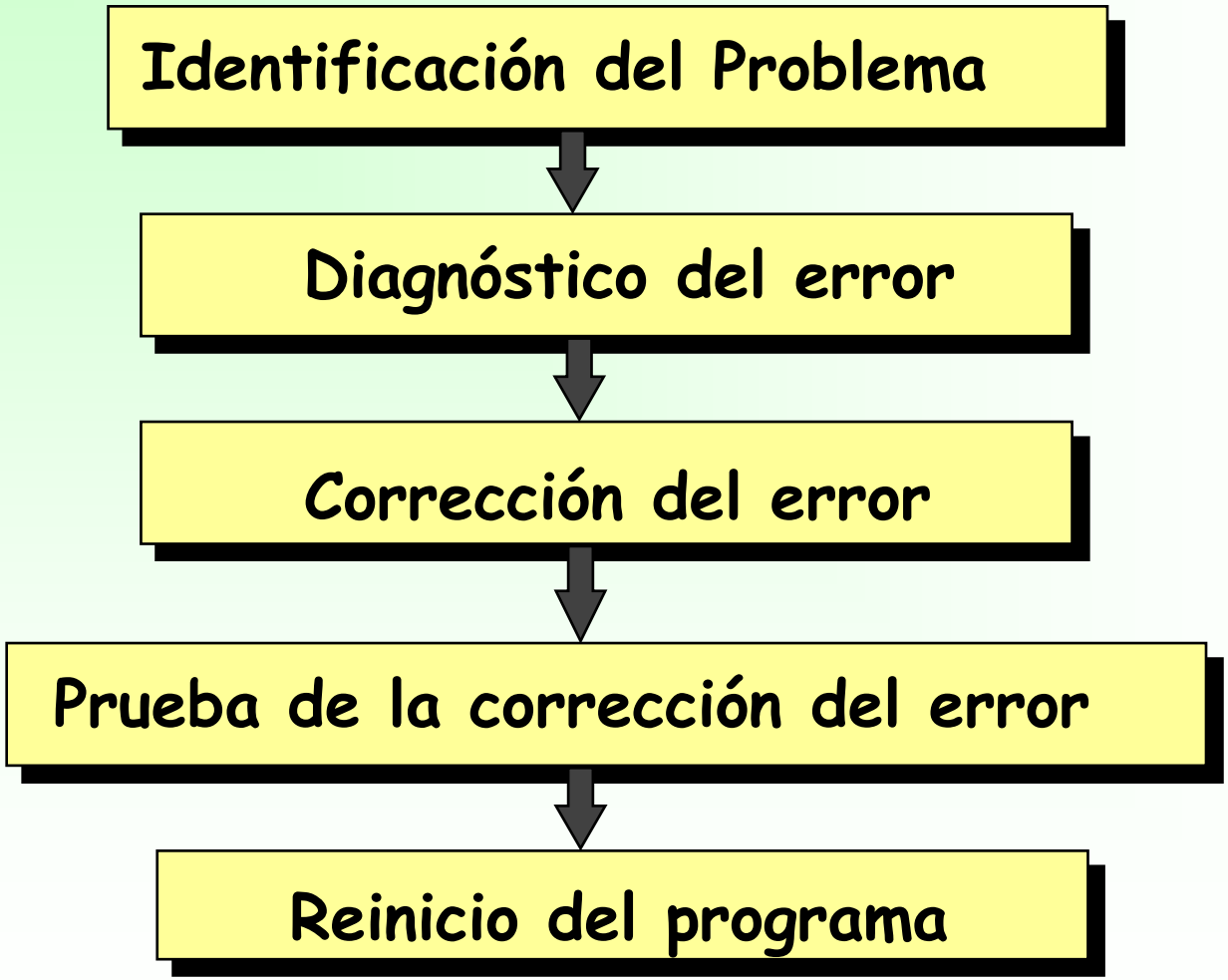
ERRORES (BUGS)  FALLAS (FAULTS)

Especificación de los Requerimientos

Características de Una Especificación de Requerimientos de un software ERS

- No Ambigua
- Completa
- Verificable
- Consistente
- Modificable
- Trazable
- Utilizable

Proceso de Depuración o Debugging



Definición de Errores

Errores Previos Fijos: Persisten en el Software luego de que el programador ha trabajado en él corrigiendo un error o cambiando un código (Debugging).

Errores Generados: No existían en el Software, hasta que son introducidos como consecuencia del Debugging.

Errores Clásicos

- Matrices sobre grabadas
- Errores en los bits de bandera
- Errores en los bits de indexado
- Errores en los bits de desplazamiento
- Errores de complementación aritmética
- Problemas de puntero
- Errores de transferencia de control
- Problemas de direccionamiento indirecto
- Problema de reinicio del programa

Confiabilidad y Complejidad

La complejidad de un programa de computación es una medida de la dificultad para llevar a cabo esa computación y está muy relacionada con su confiabilidad.

Es evidente que cuanto mas complejo sea el algoritmo de cómputo tanto mas probabilidad existe de que se cometan errores en su programación y por lo tanto de fallas del software y deterioro de su confiabilidad.

Confiabilidad y Complejidad

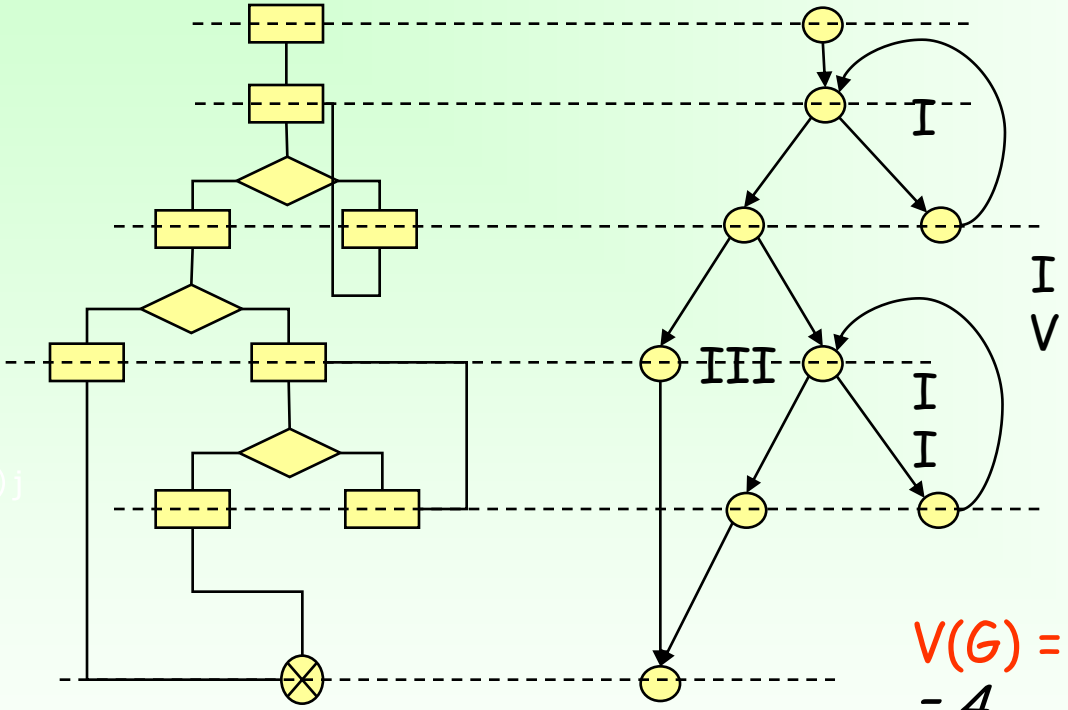
Esta demostrado que bajando la complejidad de un software su confiabilidad mejora.

Existen diversas teorías que permiten evaluar la complejidad del software.

Algunas son mas utilizadas que otras por su sencillez, sin embargo son dos las principales:

La complejidad McCabe y la Complejidad Halstead.

Complejidad McCabe



V(G) = e-n+2.p = 11-9+2.1 = 4

V(G) = # de áreas en que deda dividido el plano

V(G) = # de desiciones + 1

Complejidad Halstead

Longitud del programa : $N=N_1+N_2$

Vocabulario utilizado en el programa : $\eta=\eta_1+\eta_2$

Volumen del programa : $V=(N_1+N_2).\log_2 (\eta_1+\eta_2)=N.\log_2 \eta$

Volumen potencial del programa : $V^*=(2+n).\log_2 (2+n)$

Número de operandos de entrada-salida necesarios para el programa : $n = 2$.

Relación de volúmenes : $L=V^*/V$

Esfuerzo de programación : $E=V/L=V^2/V^*$

Número de errores : $B=V/S^*=V/3000$

Ley de Zipf

Teoría del Lenguaje Natural → Chomsky

Operadores → Verbos

Operandos → Nombres

Símbolos → Palabras

Expresión → Frase

Programa → Libro

Módulo → Párrafo

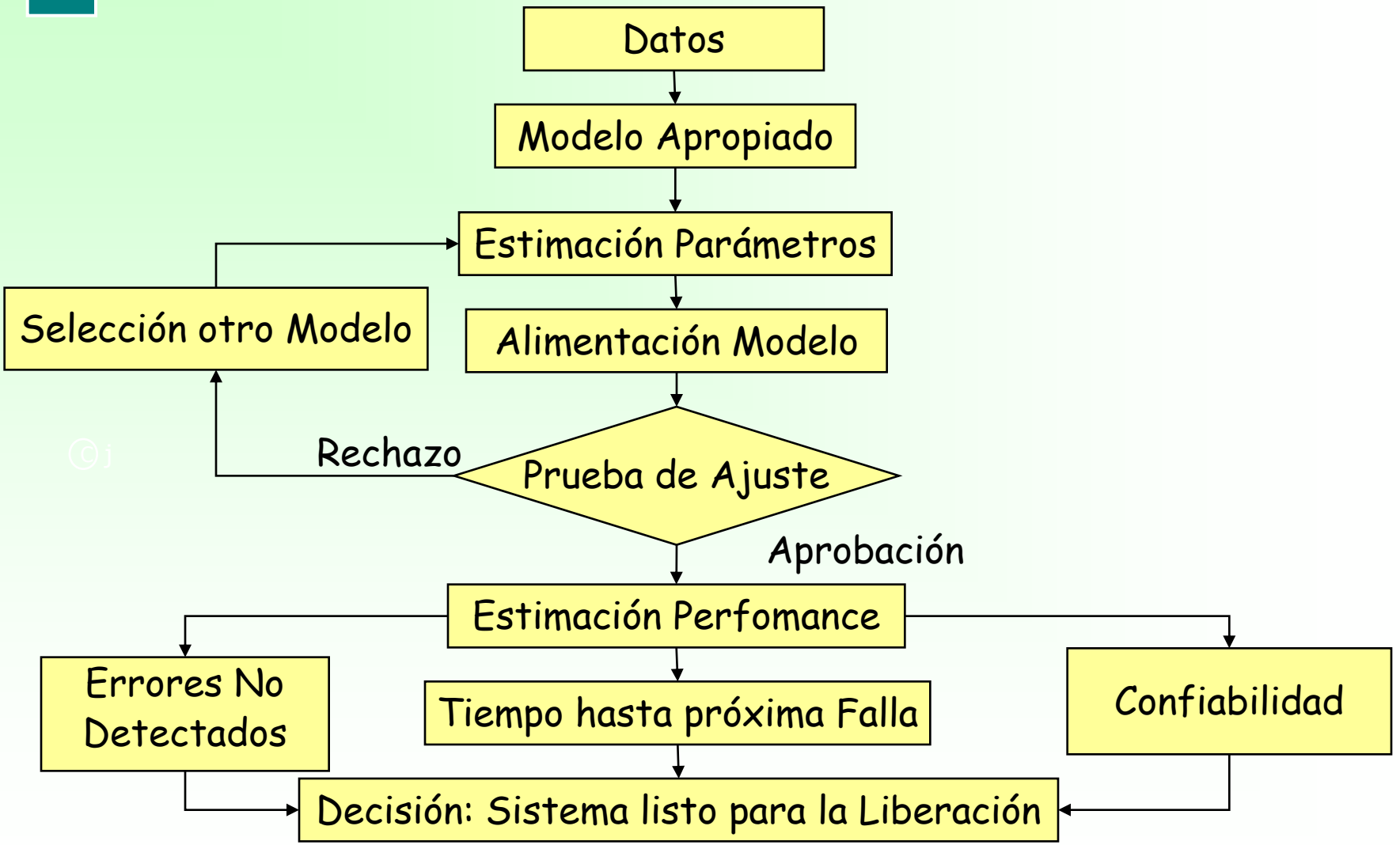
Tipos de Instrucciones		Calculado	Real
Tipos Distintos	t	n=t.(0,5772+ln t)	n
Operadores	12	36,7	31
Operandos	9	25	24
Operadores + Operandos	21	76	55

Modelos de Confiabilidad de un Software

- Existen tres clasificaciones importantes de los modelos utilizados en el análisis de Confiabilidad de un Software:

- Modelos de acuerdo al Ciclo de Vida.
- Modelos de acuerdo a la Naturaleza del Proceso de Falla.
- Modelos de acuerdo a Consideraciones Estructurales.

Diagrama Operativo Utilizando Modelos



Modelos de Acuerdo al Ciclo de Vida

FASE DESARROLLO

El software se prueba y se corrige - La confiabilidad crece

(Crecimiento de la confiabilidad con el tiempo)

JELINSKI-MORANDA DE-EUTROPHICATION (Determinístico)

MUSA (Determinístico)

SHOOMAN (Determinístico)

POISSON (Determinístico)

SHICK-WOLVERTON (Determinístico)

TRIVEDI-SHOOMAN (Markoviano)

INPUT DOMAIN BASED (Estocástico)

LITTLEWOOD-VERRAL (Bayesiano)

Modelos de Acuerdo al Ciclo de Vida

FASE VALIDACIÓN

El software no se corrige-Se aprueba o rechaza

(Softwares para aplicaciones críticas)

NELSON
SHOOMAN PATH RELIABILITY
INPUT DOMAIN BASED MODEL

FASE OPERACIONAL

Validación continua-Entradas al software dependientes

(Softwares para control de Procesos)

INPUT DOMAIN BASED MODEL
MARKOV PROCESS -LITTLEWOOD - CHENG

Modelos de Acuerdo al Ciclo de Vida

FASE MANTENIMIENTO

Adición de nuevas posibilidades-Mejora de algoritmos

(Existen pocos modelos)

INPUT DOMAIN BASED MODEL

Modelos de Acuerdo al Ciclo de Vida

MEDIDA DE EXACTITUD (CORRECTNESS)

Medida de la confianza en el test

(Softwares para aplicaciones críticas)

SIEMBRA DE ERRORES: BASIN,MILL,DE MILLO-LIPTON

FENOMENOLOGICOS: HALSTEAD

ESTADISTICOS: NELSON,EHRENBERGER,BROWN-LIPOW

INPUT DOMAIN BASED MODEL

Modelos de Acuerdo al Proceso de Falla

TIEMPO ENTRE FALLAS

Se estudia el tiempo entre fallas

JELINSKI-MORANDA DE-EUTROPHICATION

GOEL-OKUMOTO-IMPERFECTO DEBUGGING

SCHICK-WOLVERTON

LITTLEWOOD-VERRAL-BAYESIANO

SIEMBRA DE ERRORES

Se estudia la reacción ante la introducción forzada de errores

MILLS

Modelos de Acuerdo al Proceso de Falla

CONTEO DE FALLAS

Se estudia el número de fallas detectadas

GOEL-OKUMOTO-POISSON NO HOMOGENEO

GOEL-POISSON NO HOMOGENEO GENERALIZADO

MUSA-TIEMPO DE EJECUCIÓN

SHOOMAN-EXPONENCIAL

POISSON GENERALIZADO

IBM-BINOMIAL-POISSON

MUSA-OKUMOTO-POISSON LOGARITMICO

Modelos de Acuerdo al Proceso de Falla

DOMINIO DE LAS ENTRADAS

Se estudia la reacción ante la variabilidad de las entradas

NELSON

RAMAMOORTHY-BASTANI

Modelos de Acuerdo al Proceso de Falla

MEDIDA DE EXACTITUD (CORRECTNESS)

Medida de la confianza en el test

(Softwares para aplicaciones críticas)

SIEMBRA DE ERRORES: BASIN,MILL,DEMILLO-LIPTON

FENOMENOLOGICOS: HALSTEAD

ESTADISTICOS: NELSON,EHRENBERGER, BROWN-LIPOW

INPUT DOMAIN BASED MODEL

Modelos de Acuerdo a Estructura

MACROMODELOS

Se estudia el software como una caja negra

JELINSKI-MORANDA

GOEL-OKUMOTO

SCHICK-WOLVERTON

LITTLEWOOD-VERRAL

SHOOMAN

MUSA

NELSON

MILLS

Modelos de Acuerdo a Estructura

MICROMODELOS

Se estudia la estructura interna del software

SHOOMAN

MODELOS DE DISPONIBILIDAD

Se estudia el proceso del mantenimiento del software

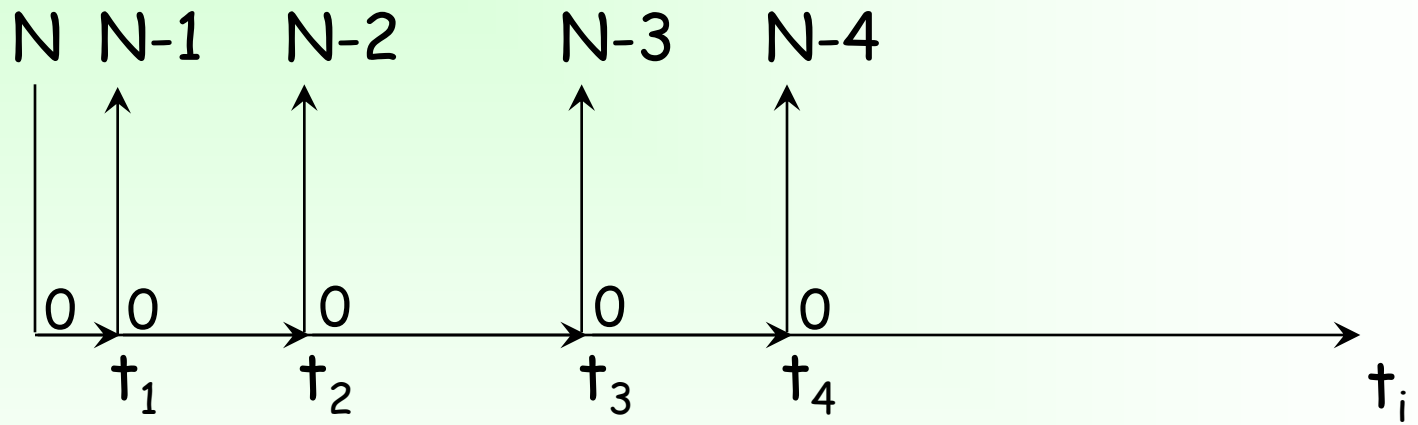
SHOOMAN-TRIVEDI

Macromodelo JELINSKI-MORANDA (1)

$$z(t_i) = \Phi.(N-i+1)$$

$z(t_i)$ = Tasa de fallas del periodo de tiempo t_i de debugging.
t_i = Periodo de tiempo de debugging (no incluye tiempos de reparación del software).
Φ = Constante de proporcionalidad.
N = Número total de errores del software.
i = Número de errores encontrados y removidos después del periodo de tiempo t_i de debugging.

Macromodelo JELINSKI-MORANDA (2)



$$z(t_i) = \Phi.(N-i+1) \qquad C(t_i) = \exp.[\, -\Phi.(N-i+1).t_i \,]$$

Macromodelo JELINSKI-MORANDA (3)

$$C(t_i) = \exp.[\, -\Phi.(N-i+1).t_i \,]$$

$$L(N, \Phi) = \prod_{i=1}^{n} f(t_i) = \prod_{i=1}^{n} [\Phi.(N-i+1)].\exp.[\, -\Phi.(N-i+1).t_i \,]$$

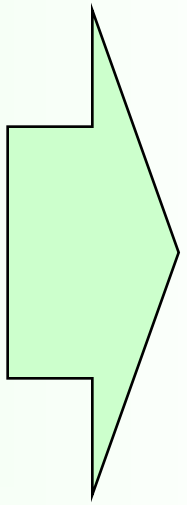
$$\mathcal{L}(N, \Phi) = \ln.[L(N, \Phi)] = \sum_{i=1}^{n} \{\ln.(N-i+1) + \ln.\Phi - (N-i+1).\Phi.t_i \}$$

$$\partial \mathcal{L}(N, \Phi)/\partial N = 0$$
$$\partial \mathcal{L}(N, \Phi)/\partial \Phi = 0$$

Macromodelo JELINSKI-MORANDA (4)

$$\sum_{i=1}^{n} [1/(N-i+1)] = \Phi . \sum_{i=1}^{n} t_i$$

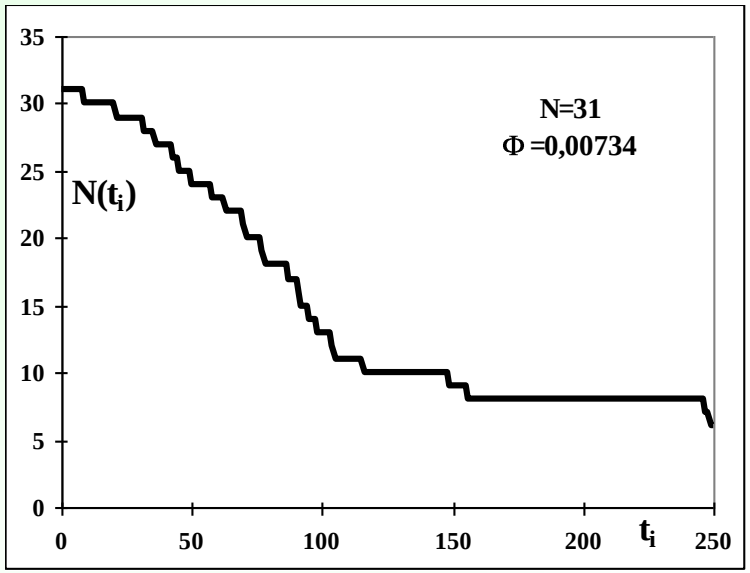
$$\Phi = n \,/\, [N . \sum_{i=1}^{n} t_i - \sum_{i=1}^{n} (i-1) . t_i]$$



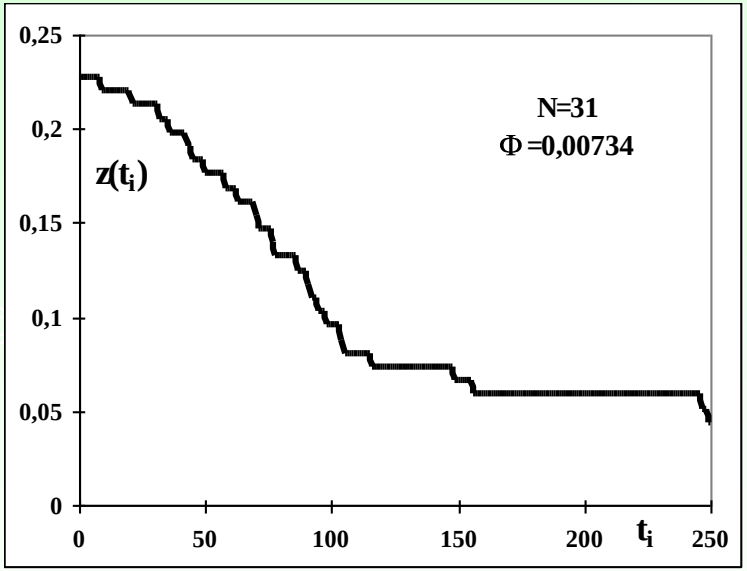
$$\hat{N}, \hat{\Phi}$$

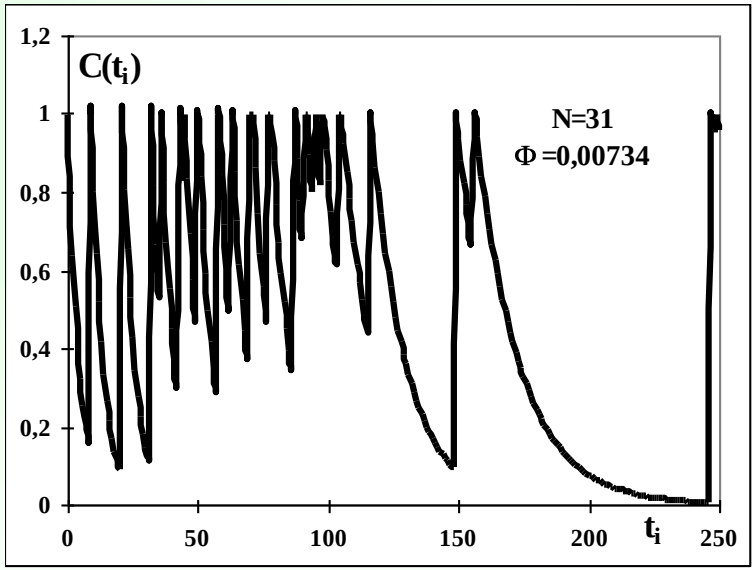
Macromodelo JELINSKI-MORANDA (5)

i	t_i	i	t_i
1	9	14	9
2	12	15	4
3	11	16	1
4	4	17	3
5	7	18	3
6	2	19	6
7	5	20	1
8	8	21	11
9	5	22	33
10	7	23	7
11	1	24	91
12	6	25	2
13	1	26	1



Macromodelo JELINSKI-MORANDA (6)





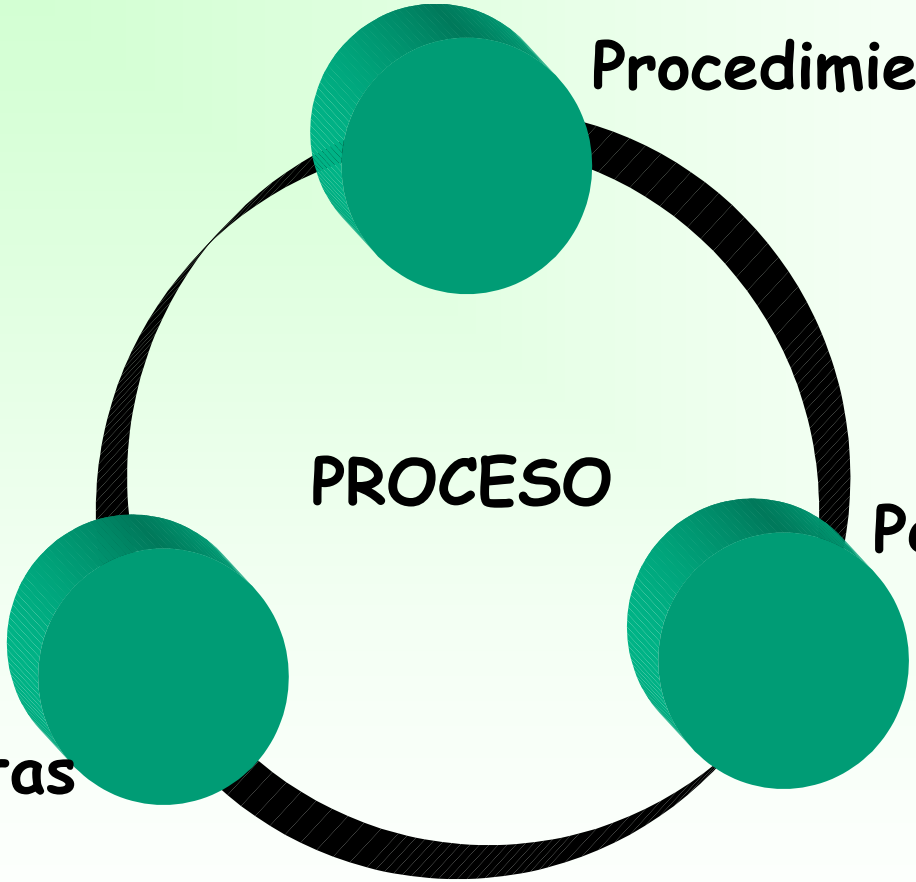
Obtención de Softwares Confiables

- Existen técnicas de programación simples:
- ESTRUCTURACIÓN
- MODULARIZACIÓN
- KISS
- Técnicas de programación complejas:
- N-VERSIONES DE UN PROGRAMA
- BLOQUES RECUPERABLES

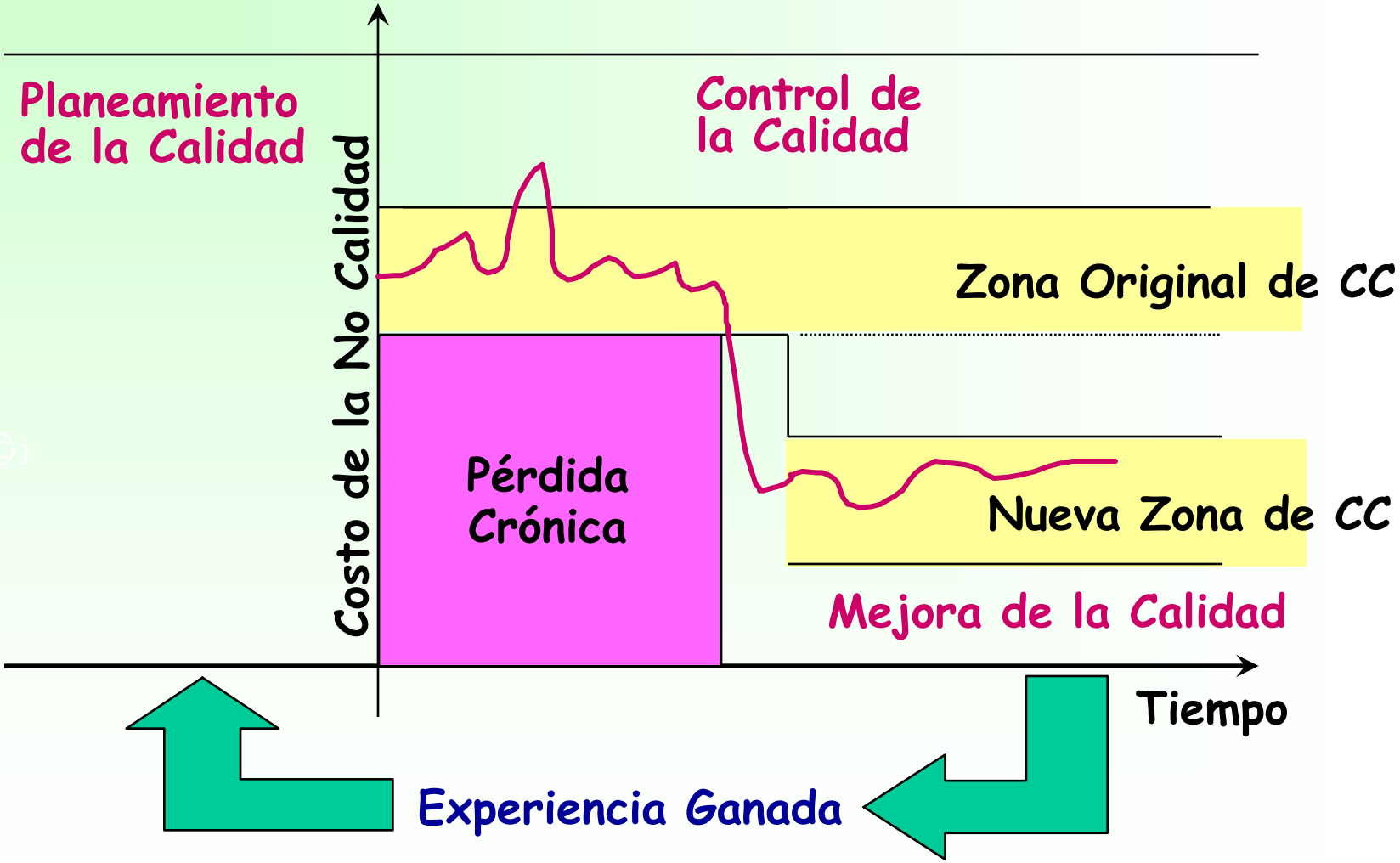
Metodologías de Trabajo

1. Modelo de Cascada (Radice - 1985)
2. Modelo de Prototipo (Gomaa & Scott - 1981)
3. Modelo de Espiral (Bohem - 1988)
4. Modelo Iterativo (Basili & Turner - 1975)
5. Modelo Orientado a Objetos (Branson & Herness - 1992)
6. Modelo de Sala Limpia (Linger & Hausler - 1992)
7. Modelo de Prevención de Defectos (Jones - 1990)
8. Modelo de Capacidad de Maduración (Humphrey -1989)
9. Modelo de Productividad (Jones - 1986)
10. Modelo Malcom Baldrige (DOC - 1988)
11. Modelo ISO 9000-3 (ISO - 1995)

TQM : Total Quality Management

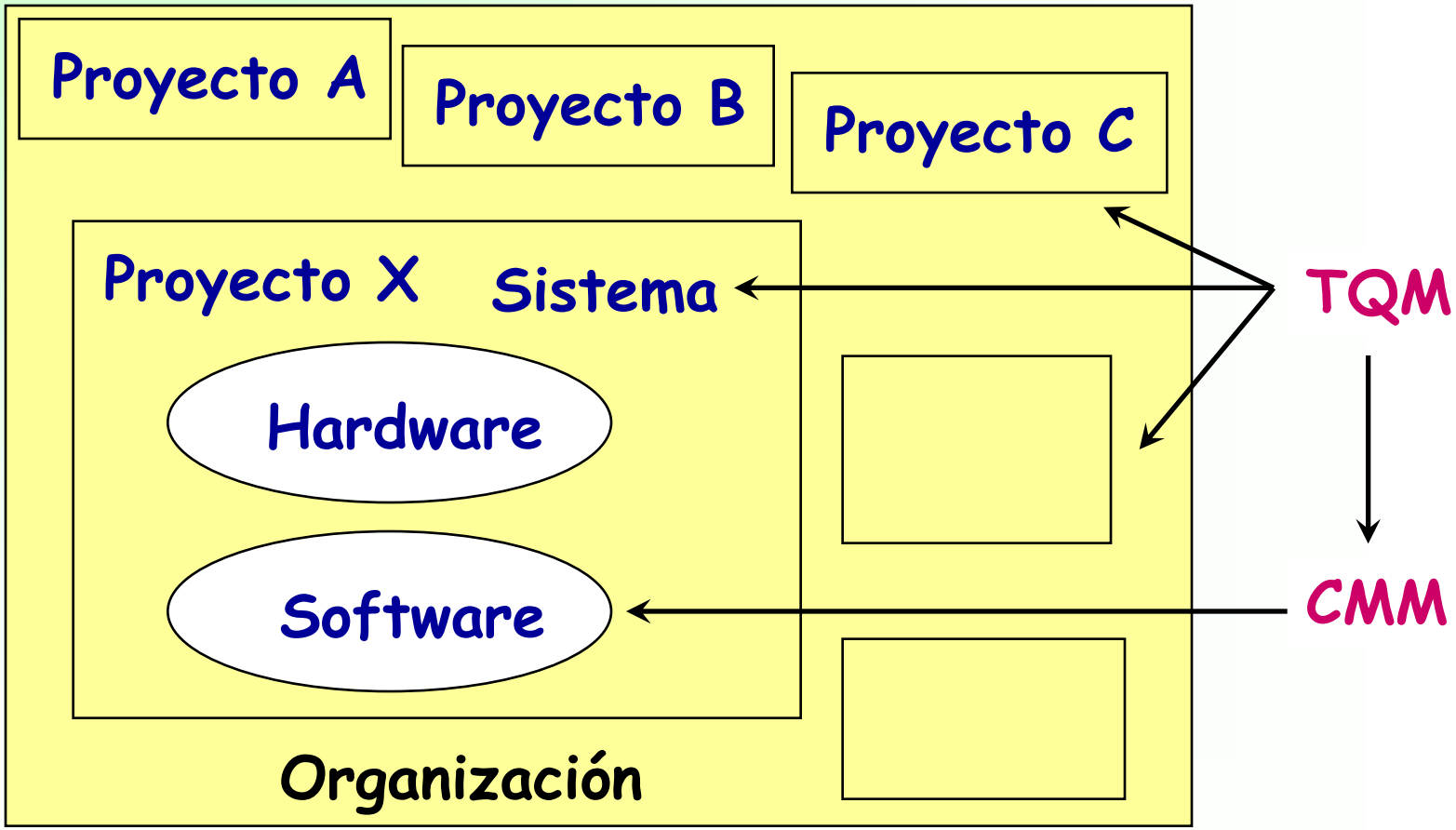


Trilogía de JURAN

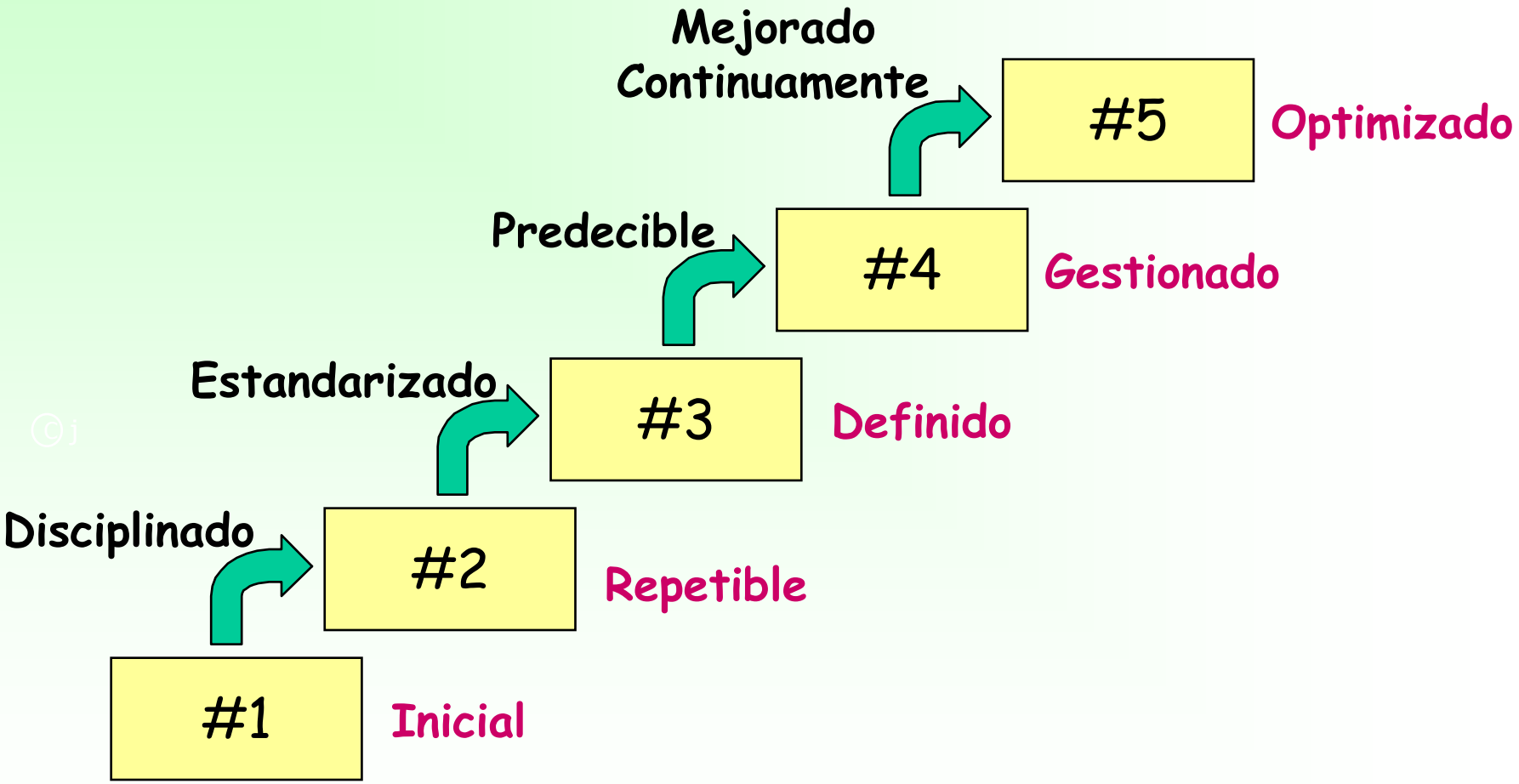


CMM : Capability Maturity Model

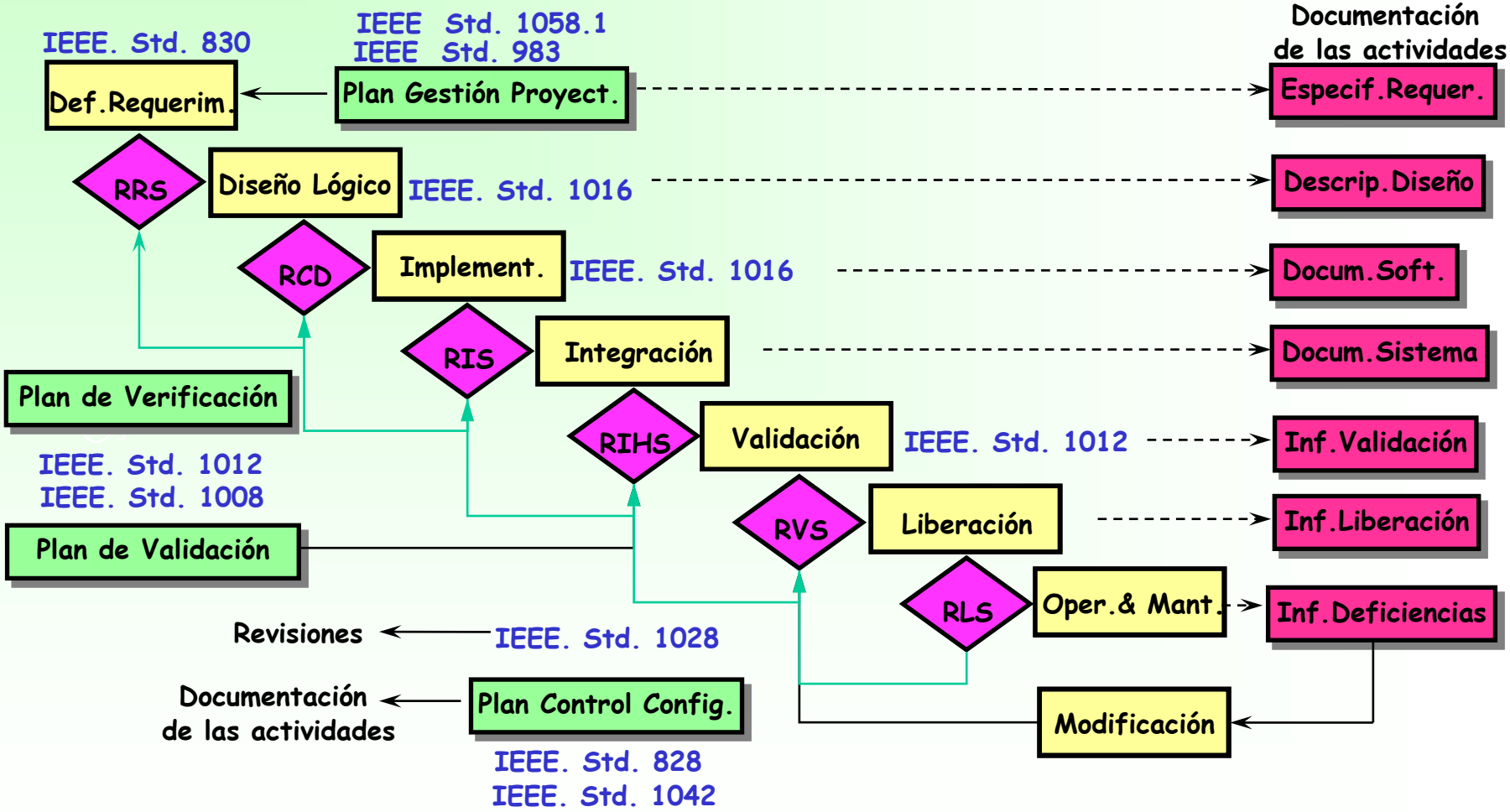
TQM aplicado al Software



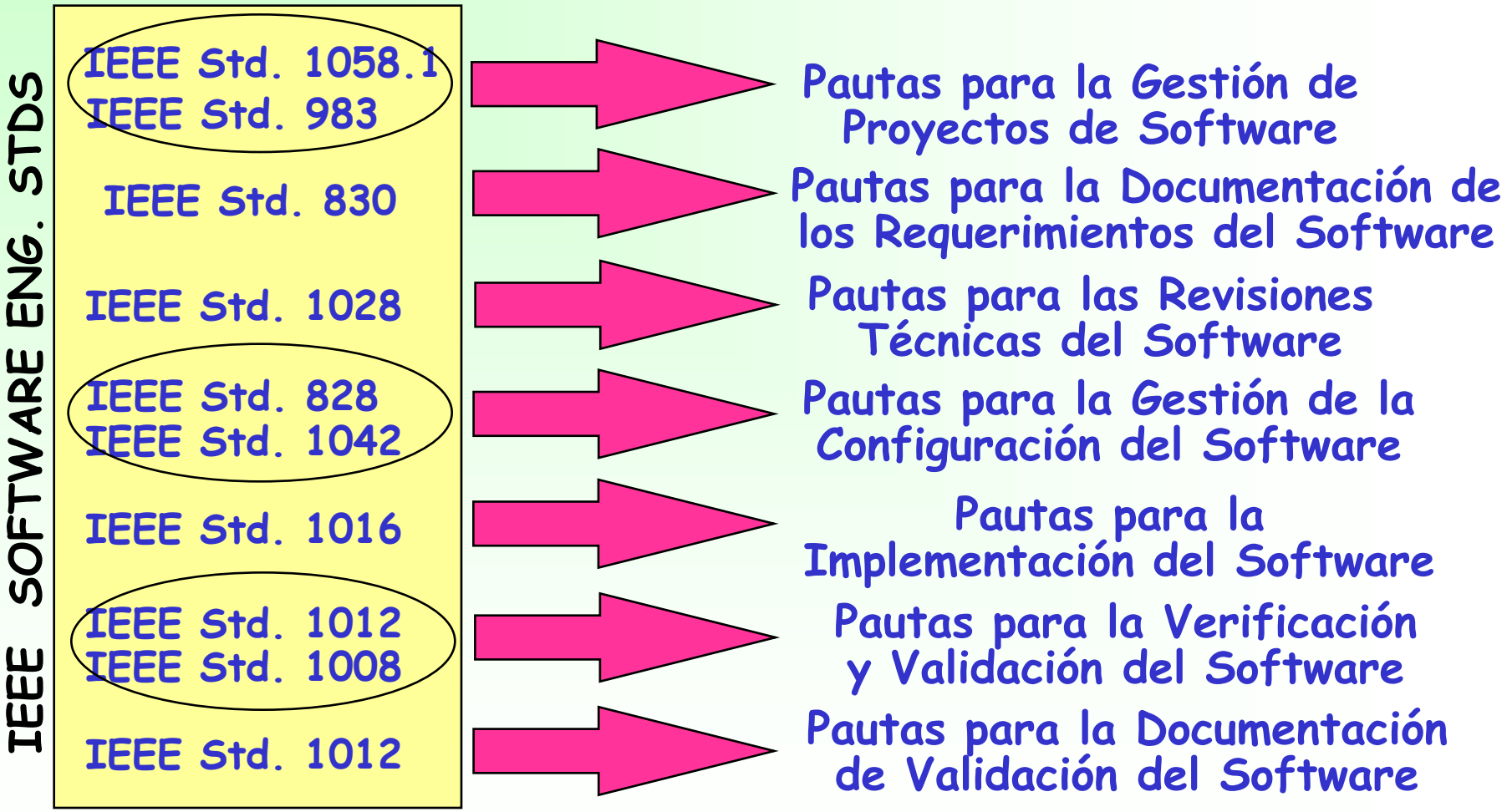
CMM - 5 Niveles del Proceso de Maduración



Modelo de Cascada - IEEE Std.



Modelo de Cascada - IEEE Std.



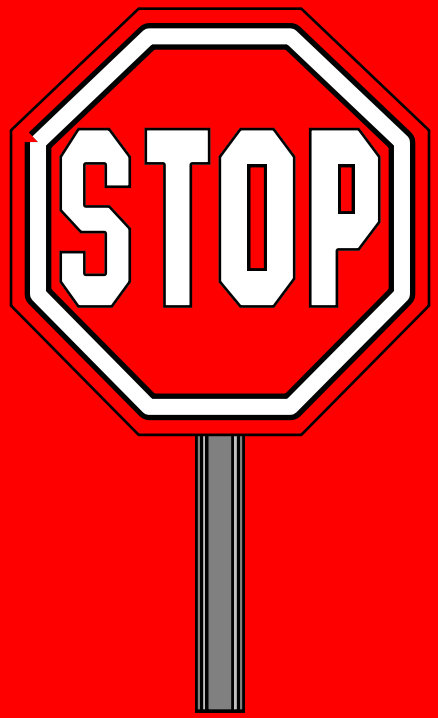
Conclusiones

- Obtener softwares cada vez más confiables y seguros es necesario tanto desde el punto de vista práctico como ético.
- Los objetivos solo pueden alcanzarse mediante la aplicación sistemática de herramientas a veces poco conocidas. La divulgación de este paquete de conocimiento debe comenzar desde los primeros pasos de la enseñanza universitaria y propagarse a cada emprendimiento que en materia de software se comience.
- Mejores softwares, menos complejos y más portables podrán obtener mejores resultados aplicativos.

Actitud Creativa

No nos atrevemos a muchas cosas porque son difíciles, pero son difíciles porque no nos atrevemos a hacerlas.

Séneca



Las HERRAMIENTAS están sólo hay que UTILIZARLAS