




# Análisis Estático de Modelos



## Introducción

**El Análisis Estático** tradicional es una disciplina conocida y común en el mundo de implementación de los sistemas. A diferencia de análisis dinámico, que ocurre durante la ejecución del sistema (pruebas de funcionamiento o testing), mediante el análisis estático se analizan las características de código fuente, relaciones entre los módulos, arquitectura, etc. En el mercado existen muchas herramientas<sup>1</sup> para el análisis estático de prácticamente todos los lenguajes de programación, bases de datos, etc.

El portador de la complejidad de sistema es justamente su estructura estática, por lo tanto el análisis estático es una herramienta valiosa para controlar esta complejidad y construir sistemas más robustos, más fáciles de extender y entender, etc.

 Enterprise Analyst introduce una herramienta poderosa para el análisis estático en nivel de modelo conceptual del sistema! Diseñado de una forma equivalente a los métodos tradicionales, los constructores de los modelos pueden empezar a controlar la complejidad de su sistema desde el nivel de modelo.

Este documento contiene la base teórica debajo de análisis estático realizado por Enterprise Analyst. Está dividido en tres secciones:

- Dependencias en los modelos
- Matriz de dependencias, y
- Métricas de Modelo

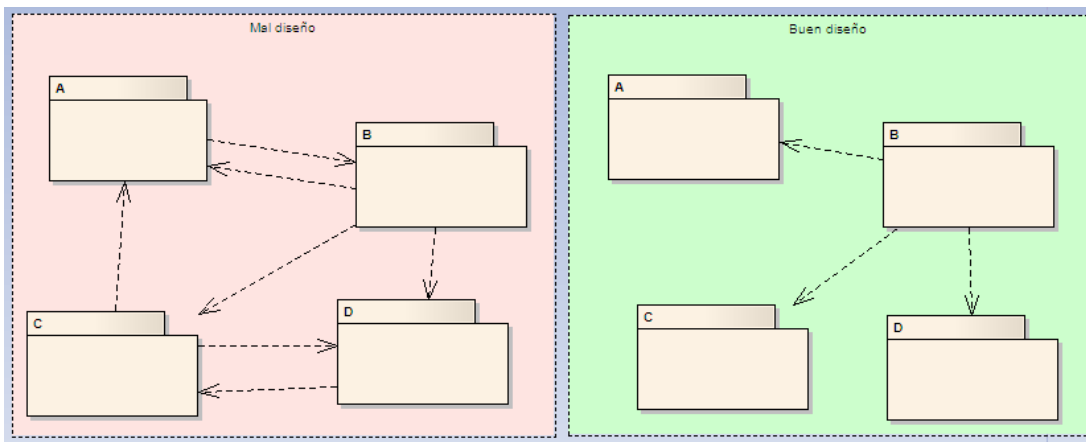
---

<sup>1</sup> Una de las herramientas más conocidas de análisis estático de código es Lattix LDM:  
[www.craftware.net/es/productos/herramienta\\_ldm.htm](http://www.craftware.net/es/productos/herramienta_ldm.htm)

## Introducción

Las dependencias entre sus distintos elementos son el factor fundamental de la complejidad de un sistema (o modelo) grande. Si los elementos son muy dependientes entre ellos, cualquier modificación de uno puede provocar quiebres de funcionamiento o, en el mejor de los casos, la necesidad de realizar modificaciones en otros elementos.

Probablemente no es necesario realizar un análisis formal para tener una idea de la calidad de las dos estructuras siguientes compuestas por los mismos 4 módulos:



*Mal diseño versus buen diseño*

Solución "Mal Diseño": El diseño en la izquierda es muy enredado, con alta cantidad de dependencias entre los módulos, incluyendo las más peligrosas - dependencias cíclicas (llamadas **enredos**). Una estructura como esta es muy poco resistente a las modificaciones de cualquier tipo, porque ellas se propagan fácilmente a través de la red de las dependencias.

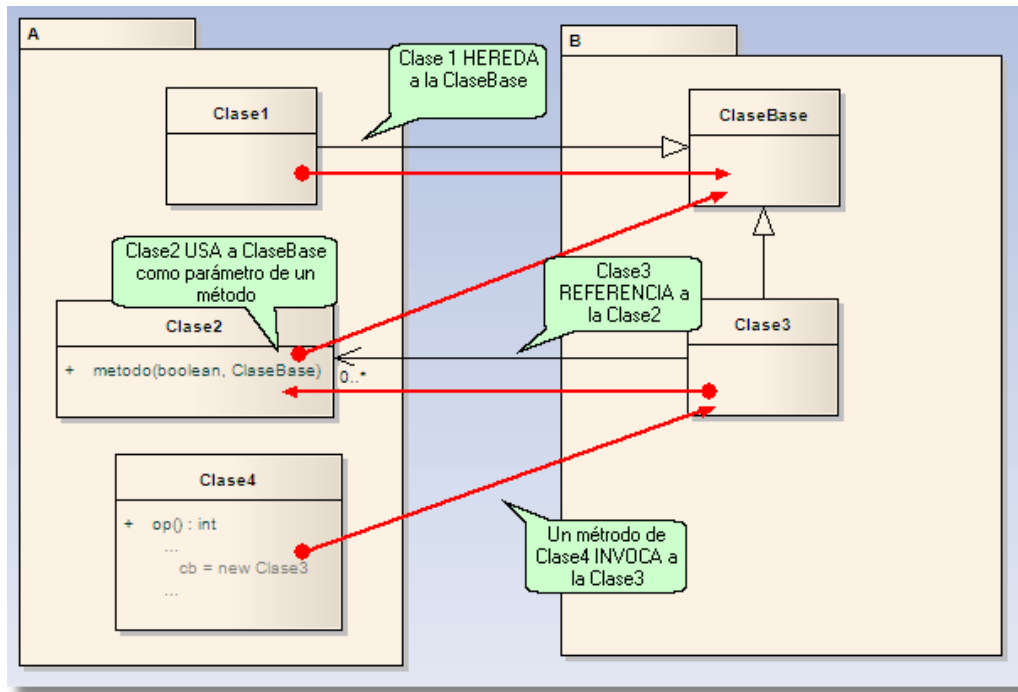
Por ejemplo, una modificación en el módulo D se puede propagar directamente a B y C, e indirectamente al A. Adicionalmente, las modificaciones se "devuelven" al mismo D mediante las dependencias cíclicas. En realidad, en esta solución cualquier modificación se propaga descontroladamente a todos los otros módulos.

Solución "Buen Diseño": La solución alternativa es indudablemente mucho mejor. Con una cantidad razonablemente baja de las dependencias correctamente distribuidas y sin los enredos.

Una modificación en el módulo D se puede propagar hasta el B. Lo mismo pasa con cualquier modificación en los módulos A y C. El módulo B está completamente libre de dependencias entrantes, por lo cual se puede modificar sin ningún impacto al resto del sistema (modelo)!


**Fuentes de las dependencias**

Las dependencias entre los módulos (o paquetes) efectivamente son dependencias "derivadas", y se deben a las dependencias entre los elementos contenidos en los paquetes correspondientes. El siguiente diagrama explica las distintas formas de generar dependencias entre los elementos de un modelo. Las flechas rojas indican el sentido de cada dependencia (desde el elemento dependiente hacia el de que se depende).



*Orígenes de dependencias en modelo*

Como consecuencia de estas dependencias resulta que el paquete A depende de B (clase1 y clase2 dependen de ClaseBase y Clase4 depende de Clase3) y también el paquete B depende de A (clase3 depende de Clase2).

 **Enterprise Analyst** detecta y analiza las 4 posibles fuentes de dependencias: herencia, referencia, uso e invocación.

## Peso de Dependencias

Como consecuencia de las dependencias en la figura anterior se produce que el paquete A depende de B y también el paquete B depende de A.

Tiene sentido pensar en la fuerza de la dependencia entre dos elementos o paquetes. En el ejemplo anterior se observan 3 dependencias distintas desde el paquete A hacia el paquete B y sólo una en el sentido opuesto. También se observa que el paquete A depende de dos elementos del paquete B, mientras el paquete B depende de un solo elemento del paquete A. De una cierta forma, la dependencia de paquete A con respecto a B es "**más pesada**" que la opuesta. Esta dependencia se manejaría con **más dificultades**.

Este fenómeno es conocido como **peso de dependencia**.

Hay 4 distintas formas de calcular el peso de dependencia:

- *Estrategia binaria* - expresa solo si hay o no hay dependencia entre dos paquetes
- *Estrategia de conocimiento de subsistema* - expresa la cantidad de clases del paquete del cual se depende, conocidas por el paquete dependiente
- *Estrategia de conocimiento de clases* - expresa la cantidad de clases del paquete dependiente, que acceden a las distintas clases del otro paquete
- *Estrategia de uso* - expresa la cantidad exacta de las dependencias entre dos paquetes

La versión actual de **Enterprise Analyst** calcula los pesos de dependencias según la estrategia de uso.



Las versiones futuras de la herramienta agregarán el soporte para el resto de las estrategias.

## Matriz de Dependencias

Un modelo puede tener docenas o cientos de paquetes y miles de clases. La cantidad de dependencias en un modelo mediano o grande por lo tanto es muy alta (miles o más). Para simplificar la representación e interpretación de las dependencias en un modelo, se define la **Matriz de Dependencias (MD)**.

	A	B	C	D
A		1		2
B	1		1	1
C				
D				

*Ejemplo 1: Matriz de dependencias entre los subsistemas*

Las filas y las columnas de la MD corresponden a los paquetes y/o las clases del modelo. El número en las celdas representan los pesos de dependencias entre los elementos-columnas y elementos-filas. En la matriz de Ejemplo 1, el sistema está compuesto por 4 paquetes (o subsistemas): A, B, C y D. El paquete D depende de los paquetes A (peso 2) y B (peso 1). El paquete C depende sólo del paquete B.

La MD tiene ciertas características algebraicas que permiten un diagnóstico arquitectural relativamente rápido, antes de un análisis profundo usando las métricas.

- la diagonal principal siempre contiene todas las celdas vacías, ya que representa el cruce de cada elemento consigo mismo
- en un modelo sin dependencias cíclicas (enredos) es posible colocar todos los pesos de un solo lado de la diagonal principal
- las dependencias cíclicas (los enredos) obligan tener pesos en ambos lados de la diagonal principal
- mirando las celdas ocupadas de una fila se detectan los elementos que dependen del elemento asociado a la fila
- mirando las celdas ocupadas de una columna se detectan los elementos de los cuales depende el elemento asociado a la columna
- muchas celdas ocupadas muestran que los subsistemas son muy acoplados entre sí


Viendo la MD Ejemplo 1, sin mayores esfuerzos se puede identificar la dependencia cíclica entre los paquetes A y B. Mirando la fila B se ve que todos los demás paquetes dependen del paquete B. La columna D descubre las dependencias del paquete D, etc.

El Ejemplo 2 demuestra una vista más detallada del mismo sistema. Esta vez se muestran las dependencias entre las clases particulares de todos los subsistemas:

	A1	A2	A3	B1	B2	C1	C2	D1	D2	D3	D4
A1					1			1		1	
A2			1								
A3											
B1	1										
B2				1			1		1		
C1											
C2											
D1											
D2											1
D3											1
D4									1		

*Ejemplo 2: Matriz de dependencias entre las clases*

Por ejemplo, la dependencia entre los subsistemas C y B (detectada en la matriz Ejemplo 1) se ve descubierta en la matriz Ejemplo 2. Se debe a la dependencia entre las clases C2 y B2.

 **Enterprise Analyst** soporta completamente la representación visual de las dependencias usando la MD.

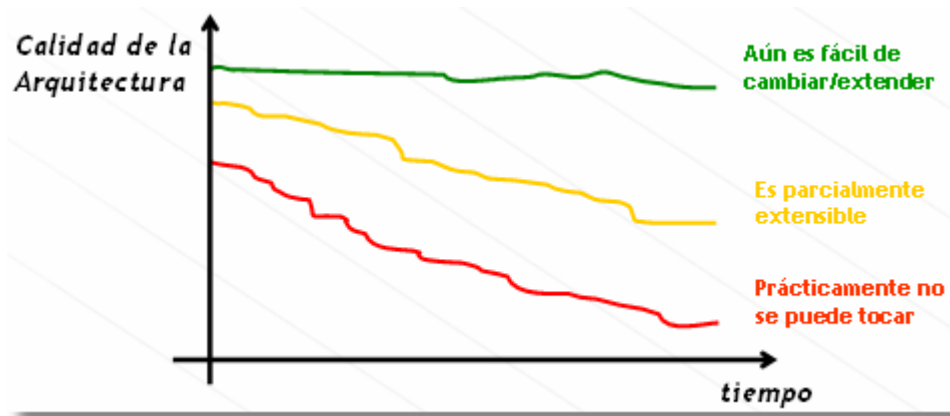
Usando la MD es relativamente fácil analizar las dependencias, detectar los enredos e iniciar un diagnóstico de la estructura de un modelo.

Para un análisis más profundo típicamente se usan las **métricas de modelo**.

## Métricas de Modelo

Las métricas de arquitectura definen un conjunto de atributos numéricos del modelo, expresando objetivamente sus distintas e importantes propiedades. Los modelos, tal como los sistemas mismos, con tiempo tienden a degradarse en el sentido de calidad de su diseño y arquitectura. Cada modificación o extensión realizada a un sistema afecta sus propiedades, típicamente de forma desfavorable. Mientras más baja es la calidad arquitectural - más difícil es realizar nuevas modificaciones.

**Un control permanente de los valores de las métricas permite atenuar el proceso de degradación arquitectónica del modelo.**



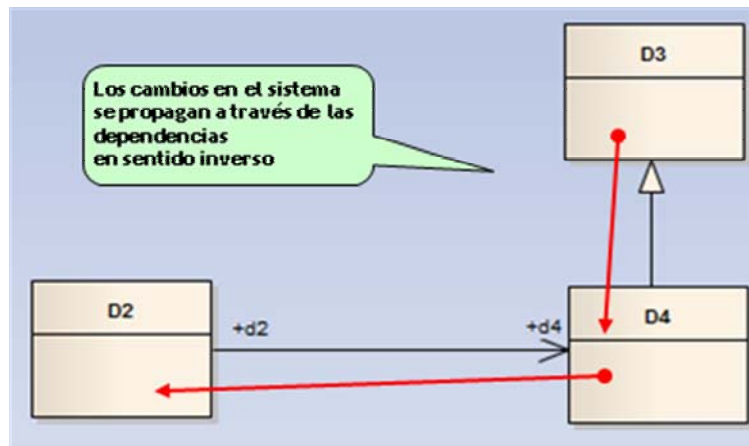
*Degradación de la calidad de modelos (y sistemas) a través de tiempo*

Más que los valores actuales de las métricas de una versión del modelo, hace sentido analizar las tendencias de las métricas entre sus distintas versiones. De esta forma se puede identificar claramente en que dirección evoluciona el modelo y reaccionar según la necesidad.

En esta sección del documento se explican las métricas de arquitectura. Todas se pueden calcular para un subsistema o para el modelo completo.

### Impacto Promedio (IP)

Impacto Promedio es una medida de daño potencial que produce un cambio de una parte del modelo al resto de él. Se calcula como el promedio aritmético de impacto que produce el cambio en cada una de sus clases. El Impacto de una clase se calcula como la cantidad de clases que directa o indirectamente dependen de esta clase. Por ejemplo, el impacto de la clase D3 en el diagrama de ejemplo es 2 (D4 depende de D3 y D2 depende de D4):



*Impacto y Estabilidad Sistémica*

De la misma forma, el Impacto de la clase D4 es 1 y el de D2 es 0.

$$IP = \frac{\sum I}{N}$$

Por lo tanto, el IP de este subsistema es 1.

El valor de IP puede ser cualquier número decimal en rango [0, N], donde N es el número total de clases en el sistema.

### Estabilidad Sistémica (ES)

ES representa el valor normalizado de IP y típicamente se expresa en porcentajes.

$$ES = 1 - \frac{IP}{N}; \text{ está en el rango [0-100\%]}$$

Mientras más cerca al 100% es la ES de un subsistema, menos daño potencial producen los cambios en este subsistema, es decir es **más estable**.

### Dependencias Entrantes y Salientes (DS y DE)

DS y DE se refieren a la cantidad de dependencias que "entran" o "salen" de un subsistema en particular. El valor de esta métrica siempre es un número entero no negativo.

Estos valores se usan para calcular otras métricas.

### Cantidad de Clases (C)

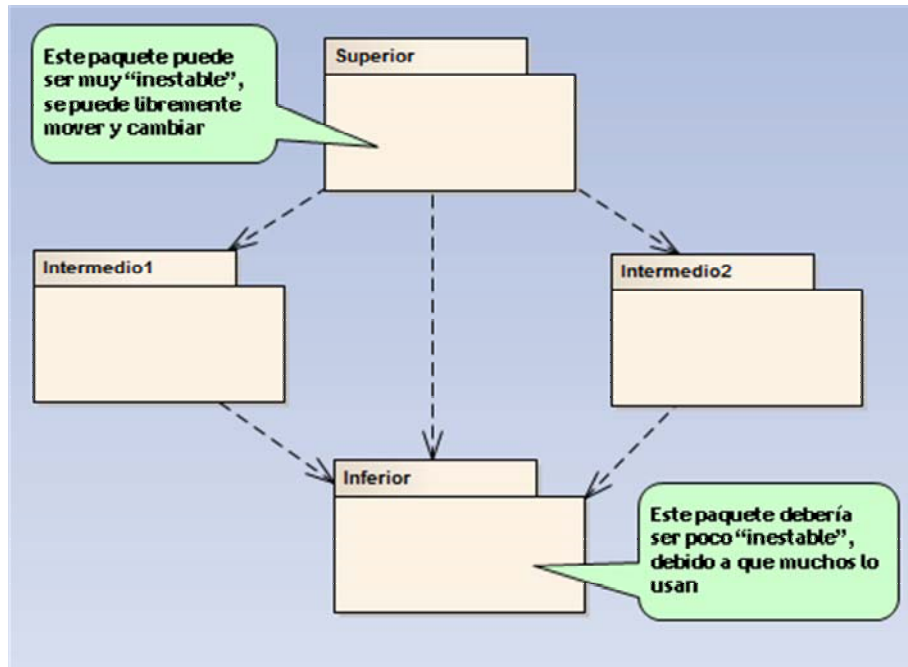
Esta simple métrica calcula el total de elementos (clases) en un subsistema. Se usa para calcular los valores de otras métricas, los promedios, etc.

### Cantidad de Clases Abstractas (CA)

Esta métrica cuenta el total de elementos abstractos (clases abstractas) en un subsistema.

### Inestabilidad (I)

Inestabilidad se define en función de las dependencias entrantes y salientes de un subsistema. El razonamiento es el siguiente - si un subsistema tiene muchos otros subsistemas dependiendo de él, será más difícil y peligroso cambiarlo.



*Inestabilidad*

Inestabilidad se define como una relación entre la cantidad de las dependencias salientes de un subsistema y la suma de las salientes y entrantes.

$$I = \frac{DS}{DS+DE} ; \text{ está en el rango } [0..1]$$

### Abstracción (A)

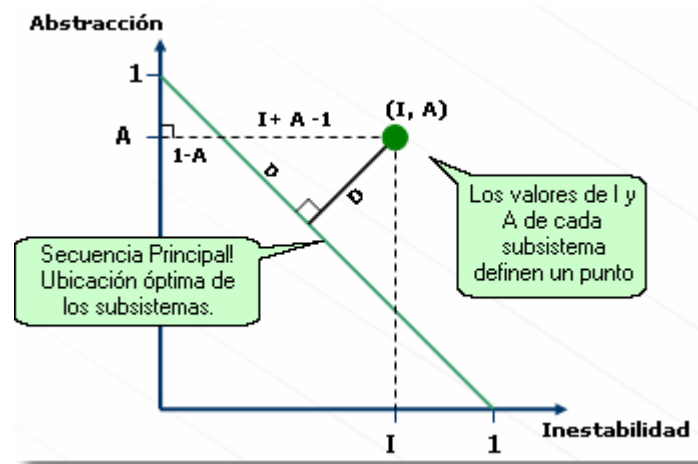
Abstracción de un subsistema es la relación entre la cantidad de los elementos abstractos (clases abstractas e interfaces) y total de clases.

$$A = \frac{CA}{C} ; \text{ está en el rango } [0..1]$$

## Distancia (D)

Las métricas Abstracción e Inestabilidad están relacionadas. Un subsistema muy inestable generalmente debería ser muy poco abstracto y viceversa. Esto tiene una explicación lógica: Un subsistema pensado para que no cambie mucho (estable) no debe ser muy concreto, es decir debería ser abstracto. Los subsistemas de este tipo típicamente se usan como base para otros subsistemas, teniendo muchas dependencias entrantes (baja Inestabilidad). Por su lado, un subsistema muy concreto (poco abstracto) puede libremente depender de muchos otros sistemas, más que otros dependen de él.

Esta situación se representa en el siguiente gráfico:



*Distancia (Abstracción versus Inestabilidad)*

Los buenos subsistemas tienden a estar ubicados cerca a la Secuencia Principal (la línea recta que une dos puntos extremos del gráfico, que corresponden a los paquetes abstractos-estables y concretos-inestables).

El valor normalizado de la métrica es:

$$D = |I + A - 1|; \text{ está en el rango } [0..1]$$

El valor ideal para un subsistema es 0.

### Cohesión Relacional (CR)

Esta métrica es una medida de la fuerza de las relaciones entre las clases de un subsistema. Su valor es:

$$CR = \frac{D}{C}$$

donde D es la cantidad de dependencias en un subsistema y C la cantidad de clases de este subsistema.

Un buen sistema debería tener alta cohesión y bajo acoplamiento (relaciones entre los paquetes).



**Enterprise Analyst** soporta las 10 métricas más conocidas de arquitectura de software: Estabilidad Sistémica, Impacto Promedio, Inestabilidad, Abstracción, Distancia, Cohesión, Dependencias Entrantes y Salientes, Cantidad de Clases y Cantidad de Clases Abstractas.