# UNIDAD I

# *INTRODUCCION A LA INGENIERIA DE SOFTWARE*

Contenido:


1.1 Definiciones

1.2 Evolucion del Software

1.3 Importancia del Software

1.4 Problemas del Software

1.5 Caracteristicas del Software

1.6 Conceptos de Calidad

1.7 Mitos del Software

1.8 Distribucion del Esfuerzo en un Proyecto de Programación

1.9 Administración de Proyectos de Software

1.10 Paradigmas de la Ingeniería de Software

## 1.1. DEFINICIONES

*Ingeniería*.- Profesión que posee conocimientos cientificos, actividades y criterios ( ingenio ) para crear dispositivos, metodos y sistemas para transformar los recursos y satisfacer mejor las necesidades de una sociedad.

*Software*.- Conjunto de programas que se pueden ejecutar en una computadora, así como toda la información, utilerias y recursos necesarios para su diseño, instalación, operación, mantenimiento y refinamiento.

*Ingenieria de Software*.- Disciplina que establece el uso de principios de ingeniería robustos, orientados a obtener software económico, que sea confiable y funcione de manera eficiente.

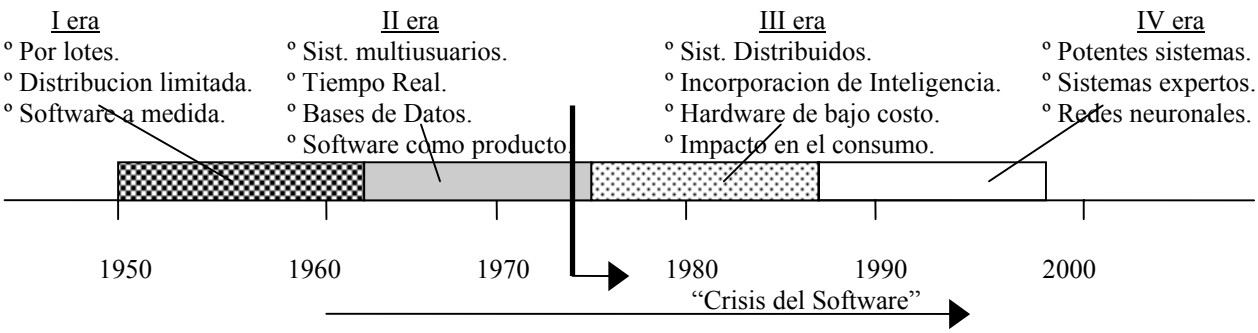
*Perfil del Ingeniero de Software*.- Debe ser capaz de encabezar o ser miembro de grupos multidisciplinarios de desarrollo de todo tipo de software y que en equipo logre producir software de alta calidad.

*Diferencia entre programador e ingeniero de software.-* La ingenieria de software difiere de la programación tradicional en que se utilizan tecnicas de ingeniería para especificar, diseñar, codificar, validar y mantener los productos dentro del tiempo y presupuesto establecidos para el proyecto, ademas esta ingeniería se preocupa por aspectos administrativos que quedan fuera del dominio normal de la programación.

El término “programador” se emplea para denominar a la persona preocupada y abocada a las tareas y detalles de la codificación, empacado y modificación de los algoritmos y estructuras de datos codificados en algun lenguaje de programación particular.

Los ingenieros de software están, ademas, capacitados para hacer frente a aspectos de análisis, diseño, verificación, y prueba de programas, la documentación, el mantenimiento y la administración del proyecto.

## 1.2 EVOLUCION DEL SOFTWARE




- Muchos cambios en hardware.
- Poca evolución del software.
- Desarrollo de software sin planeacion, y sin documentación.

- Tecnicas Interactivas.
- Control en Tiempo Real.
- Mejora de los dispositivos de almacenamiento.
- Primeras casas de software.
- Problemas por el mantenimiento.

- Aparecen las PC's.
- Cias. de Software venden miles de dolares.
- Hardware standar, el software marca la diferencia.

- Software complejo.

La "Crisis del Software" se le conoce a una etapa en la que todos los programas desarrollados se corregían cuando había fallos o modificados a necesidades cambiantes, requerían de altos esfuerzos por mantenerlos, con mayor costo a medida que la complejidad del software crecía.

En las pasadas décadas los ejecutivos y desarrolladores se hacían las siguientes preguntas:

¿ Por qué lleva tanto tiempo terminar los programas ?
¿ Por qué es tan elevado el costo ?
¿ Por qué no podemos detectar los errores antes de entregar el software a los clientes ?
¿ Por qué resulta tan dificil constatar el progreso del desarrollo del software ?

Estas y otras preguntas manifiestan el caracter del software y la forma en que se desarrolla, estos problemas hacen necesaria la adopción de técnicas de Ingeniería de Software.

## 1.3 IMPORTANCIA DEL SOFTWARE.

El software es ahora la clave del exito de muchos de los sistemas basados en computadora. El software marca la diferencia. Lo que diferencia una compañia de otra es la suficiencia, exactitud y oportunidad de la información dada por el software.

*Ejemplo de la importacia del software*: Dos consultorios dentales, ambos cuentan con los ultimos modelos de computadora personal y destinadas a apoyar las tareas y actividades relacionadas con el consultorio. Pero uno de ellos cuenta con un dispositivo especial conectado a la computadora y un SOFTWARE para obtener radiografías de piezas dentales por computadora, en un par de minutos la muestra radiografica esta en pantalla y el medico puede obtener diferentes vistas de la placa usando el software. Ademas puede establecer una conexión a traves de internet o via modem para enviar el archivo de la radiografía a otro colega experto con el fin de consultar y apoyar el diagnostico, todo esto en la misma cita. En la forma tradicional la placa radiografica esta lista en un par de dias.

El desarrollo de software se ha convertido en una industria con crecimiento vertical en los ultimos años, hoy por hoy uno de los hombres mas ricos del mundo es el dueño de una casa de software, Microsoft.

Hace un par de decadas se sostenia la teoria de que los paises que poseian los mejores recursos naturales estaban destinados a ser los mas ricos y poderosos del mundo, en México por ejemplo, se manejo la idea de que el petróleo era la puerta de entrada grande al mundo desarrollado. Indudablemente los recursos naturales tienen un papel importante en la economia de los paises, sin embargo poco a poco se fue acuñando una nueva ideología que se sintetiza en lo siguiente:
"El que posee la información y el conocimiento y hace mejor uso de él, es el que tiene el poder".

## 1.4 PROBLEMAS DEL SOFTWARE.

- La planificación y estimación de costos frecuentemente son imprecisas.
- Falta de "productividad" en la comunidad de software"
- La calidad del software es a veces ni aceptable.

Estos problemas al final crean insatisfaccion y falta de confianza de los clientes. Los problemas anteriores son solo manifestacion de otras dificultades:

- No tenemos tiempo de recoger datos sobre el proceso de desarrollo del software.
- Los proyectos de desarrollo de software se llevan a cabo con solo una vaga indicacion de los requisitos del cliente.
- La calidad del software es normalmente cuestionable.
- El mantenimiento de software es muy costoso y no se le ha considerado un aspecto importante.

Los problemas anteriores son corregibles, la clave es: Dar un enfoque de ingeniería al desarrollo de software.

## 1.5 CARACTERISTICAS DEL SOFTWARE.

El software es un elemento del sistema que es lógico. Por tanto, el software tiene características considerablemente distintas al hardware:

- El software se desarrolla, no se fabrica en un sentido clasico.
- El software no se estropea.
- La mayoria de software se construye a medida, en vez de ensamblar componentes existentes. +

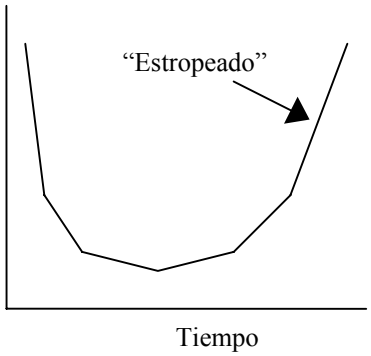



Fig- Curva de fallas del Hardware

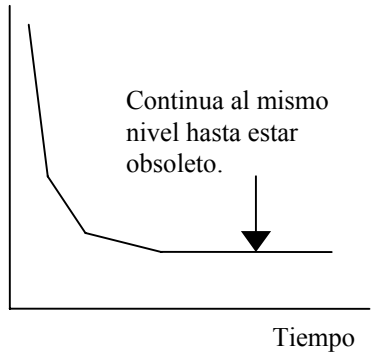



Fig- Curva ideal de fallas del software.

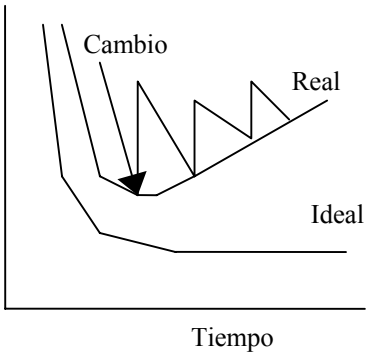



Fig- Curva real de fallas del software.

## 1.6 CONCEPTOS DE CALIDAD

Algunas características de calidad fundamentales en todo producto de programación son : utilidad, claridad, confiabilidad, eficiencia y economía.

*Utilidad*. Que satisfaga las necesidades del usuario, ya que con frecuencia no desempeñana las funciones esperadas debido principalmente a una pobre comunicacion con el cliente.

*Confiabilidad*. Capacidad de un programa para desempeñar una función requerida bajo ciertas condiciones durante un tiempo especifico. El grado de confiabilidad deseado en un producto depende del costo de las fallas.

*Claridad*. Los productos de software deben ser escritos con claridad y ser fáciles de entender tanto internamente como externamente, ya que las pruebas y actividades de mantenimiento consumen gran cantidad del presupuesto del proyecto.

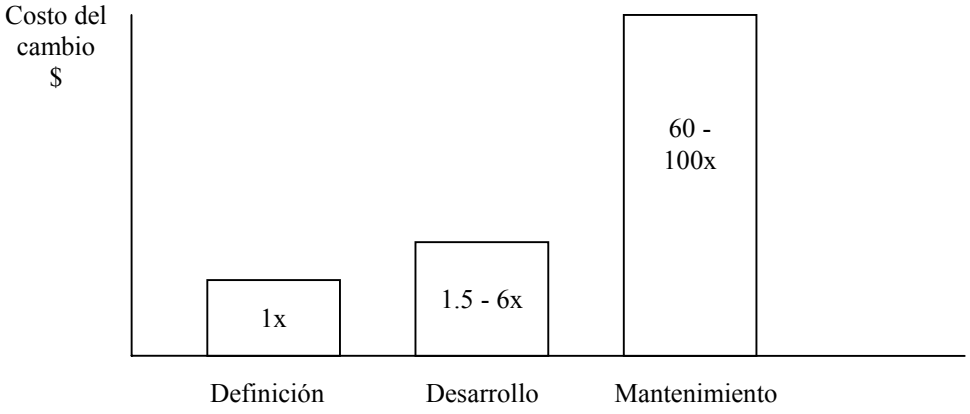
*Económico*. El producto debe ser costeable en su desarrollo, mantenimiento y uso. Un software debe operar normalmente usando menos tiempo o recursos humanos o materiales de los que se requerían antes de tenerlo.

# 1.7 MITOS DEL SOFTWARE

## *Mitos del Cliente*:

1. Mito.- "Una declaración superficial de los objetivos es suficiente para empezar a escribir los programas".
   Realidad.- La mala definición inicial es la principal causa de baja calidad.
   Se requiere un conocimiento formal y detallado de los hechos y procesos y amplia comunicacion con el cliente.

2. Mito.- "Los cambios en el software son faciles y sencillos".
   Realidad.- Es verdad que los requisitos del software cambien, pero el impacto del cambio varía según el momento en que se introduzca.




## *Mitos de los Desarrolladores.*

1. Mito.- "Terminando de escribir el programa y haciendo que funcione, nuestro trabajo habrá terminado".
   Realidad.- Entre el 50 y 70% del esfuerzo total dedicado a un programa se realiza después de entregarlo al cliente por primera vez.

2. Mito.- "Lo único que se entrega al terminar el proyecto es el programa funcionando".
   Realidad.- El software funcionando es solo una parte de una CONFIGURACION DE SOFTWARE.
   La documentación es la base de un buen desarrollo y guiás para las tareas de mantenimiento.

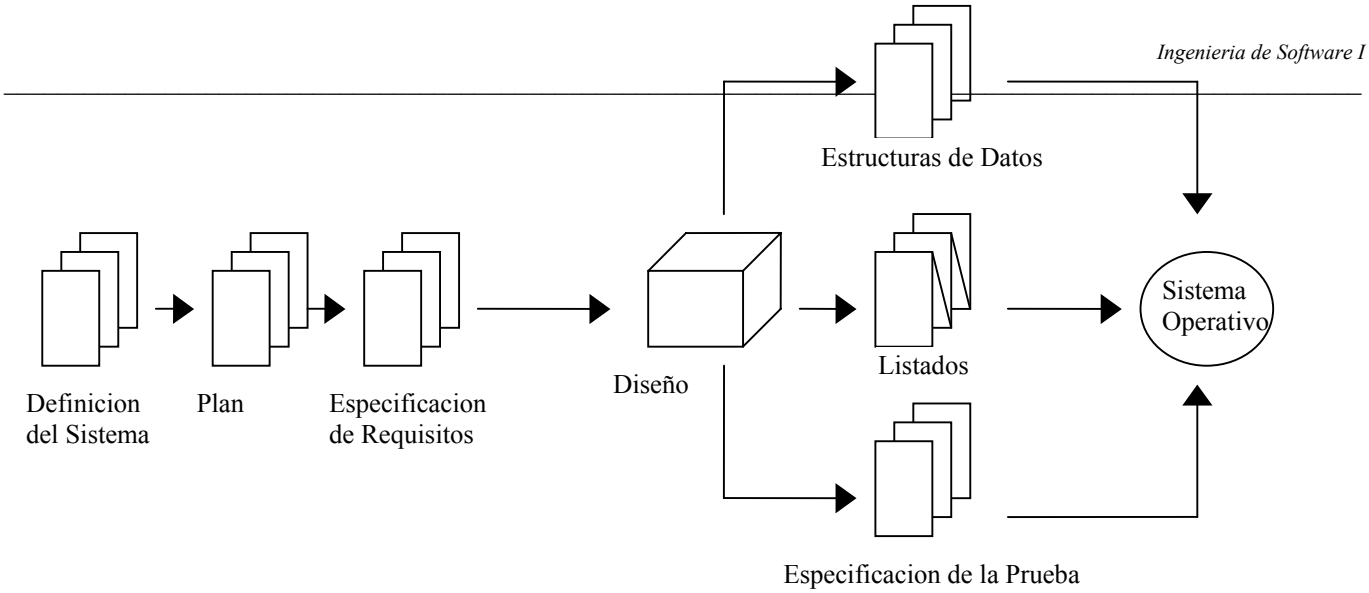



Fig- Configuracion de Software

## 1.8 DISTRIBUCION DEL ESFUERZO EN UN PROYECTO DE PROGRAMACION.

Se entiende por mantenimiento a todas las actividades posteriores a la liberación inicial del producto.

El mantenimiento de los paquetes de software contempla 3 actividades: mejoramiento de las capacidades del producto, adaptacion del producto a nuevos ambientes de computo y la depuración de errores.

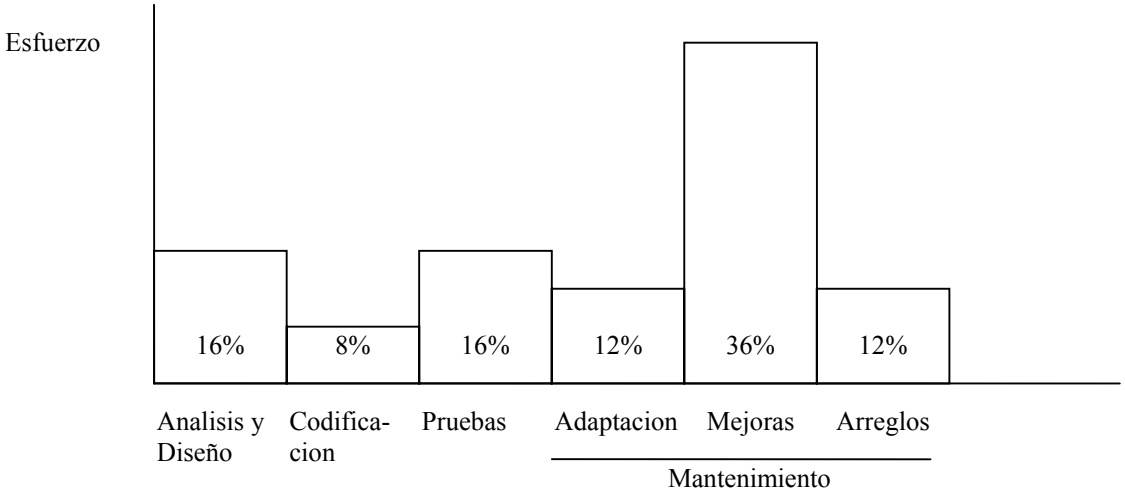



Fig- Distribucion del Esfuerzo durante el ciclo de vida de un proyecto de programacion.

- El mantenimiento gasta mas recursos que las actividades de desarrollo.

- Gran porcentaje del esfuerzo total se dedica a mejorar el producto.

- Asignar poco tiempo a las pruebas piloto y de aceptacion es una de las razones de sobrepasar el costo y tiempo de entrega de un producto.

## 1.9 ADMINISTRACION DE PROYECTOS DE SOFTWARE

Las actividades técnicas y gerenciales son igualmente importantes para el éxito de un proyecto de programación. Las actividades de la administración de un proyecto comprenden los métodos para organizar y seguir el curso del proyecto; estimación de costos, políticas de asignación de recursos, control de presupuesto, determinación de avances, ajustes al calendario de trabajo, procedimientos de control de calidad, comunicación con el cliente, etc.

Algunos problemas importantes identificados en la administración de software son:

1. Planeación de proyectos de software pobres.
2. Procedimientos de selección de gerentes de proyecto pobres.
3. La medición de proyectos es pobre.
4. Falta de procedimientos para vigilar el avance del proyecto.
5. Falta de standares para medir la calidad del desempeño y cantidad de producción esperada.

Algunos métodos sugeridos para solucionar estos problemas son:

1. Entrenar y educar a la dirección, jefes de proyecto y constructores.
2. Obligar al uso de standares, procedimientos y documentación.
3. Definir objetivos de la calidad deseada.
4. Desarrollar estimaciones de calendario y costos de forma exacta y verdadera.
5. Seleccionar jefes de proyecto basados en su capacidad para administrar proyectos mas que en su habilidad tecnica.

## 1.10 PARADIGMAS DE LA INGENIERIA DE SOFTWARE

La ingeniería de software surge de la ingeniería de sistemas y de hardware. Abarca un conjunto de tres elementos que facilitan el control sobre el proceso de desarrollo de software y suministran las bases para construir software de calidad de una forma productiva:

- Métodos
- Herramientas
- Procedimientos

*Métodos* que indican cómo construir el software técnicamente e incluyen un amplio espectro de métodos para la planificación, la estimación, el análisis, el diseño, codificación, prueba y mantenimiento.

*Herramientas* automáticas y semiautomáticas que apoyan a la aplicación de los métodos. Cuando se integran las herramientas de forma que la información creada por una herramienta puede ser usada por otra, se establece un sistema para el soporte del desarrollo de software, llamado Ingeniería de Software Asistida por Computadora ( CASE ).

*Procedimientos* que definen la secuencia en la que se aplican los métodos, las entregas, los controles de calidad y guías para evaluación del progreso.

La Ingeniería de Software está compuesta por una serie de pasos que abarcan los métodos, herramientas y procedimientos mencionados, a los que se denominan *Paradigmas de la Ingeniería de Software*.

### Ciclo de vida clásico

Este paradigma exige un enfoque secuencial del desarrollo de software. Abarca las siguientes actividades:

*Ingeniería y Análisis del Sistema*.- El Software es siempre parte de un sistema mayor, por tanto se comienza estableciendo las entidades, roles, funciones, etc de los que intevienen en el sistema, se identifican los requisitos del sistema y luego se asigna un sub conjunto de estos requisitos al software.

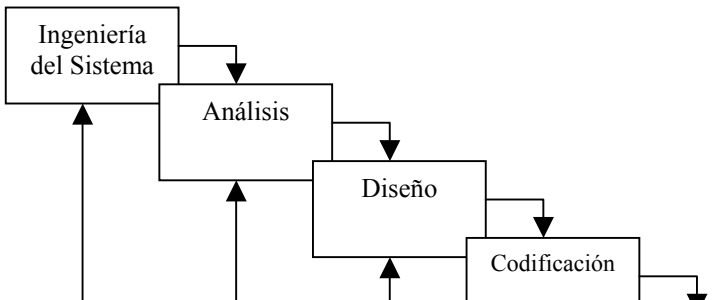
*Análisis de Requisitos del Software*.- Proceso de recopilación de los requisitos especificamente del software. El analista debe comprender el ámbito de la información, la función, el rendimiento y las interfaces del software.

*Diseño*.- Traduce los requisitos en una representación de software que pueda ser codificada.

*Codificación*.- Traducción del diseño en código fuente escrito en un lenguaje de programación.

*Prueba*.- Verificación de que las funciones del software producen los resultados que realmente se requieren.

*Mantenimiento*.- El mantenimiento aplica cada uno de los pasos precedentes para implementar los cambios que con el tiempo indudablemente sufrirá el software.



Este paradigma es el más antiguo y más ampliamente usado, sin embargo se ha cuestionado su aplicabilidad por lo siguiente:

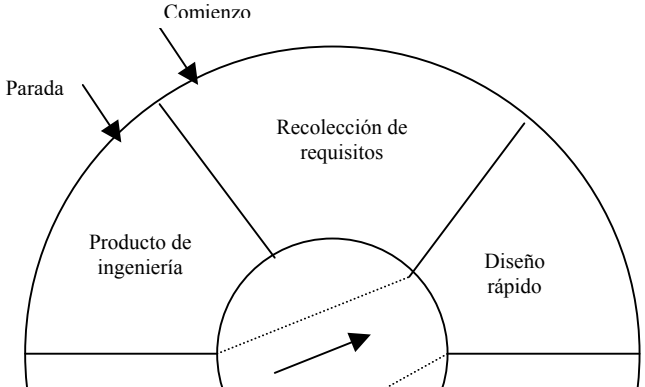
- Los proyectos reales rara vez siguen el flujo secuencial propuesto.
- Normalmente es difícil para el cliente establecer claramente todos los requisitos y el paradigma lo exige.
- El cliente debe ser paciente ya que solo hasta las etapas finales de desarrollo estará disponible una versión operativa del software.

### Construcción de Prototipos

Normalmente el cliente define los objetivos generales del software pero no identifica detallamente todos los requisitos. En otros casos el desarrollador puede no estar seguro de entender al cliente, de cómo podrá ser el software que requiere, de la eficiencia que espera, etc. En estas situaciones puede ser mejor el método de construcción de un prototipo.

El prototipo puede ser elaborado en papel o programado para que implemente algunas funciones requeridas de manera rudimentaria, sin todos los detalles y acabados del programa final.

Se empieza con la recolección de requisitos, se produce un diseño "rápido" que se enfoca sobre los aspectos visibles al usuario ( pantallas, informes, etc ) Se construye el prototipo y se evalúa por parte del cliente y sus observaciones se usan para refinar los requisitos del software a desarrollar. Se produce un proceso iterativo en el que el prototipo se "afina" para que satisfaga las necesidades del cliente y al mismo tiempo facilita al desarrollador una mejor comprensión de lo que hay que hacer.



Idealmente el prototipo solo sirve como mecanismo para identificar los requisitos del software y también puede servir como "primer sistema". Sin embargo este paradigma enfrenta algunos problemas:

- El cliente a veces ignora que el prototipo está hecho de "plastilina y alambres" y que no tiene la calidad de un software terminado, cuando se le informa esto el cliente presiona para que cuanto antes el prototipo sea el producto final.

- El desarrollador, con el fin de hacer un prototipo rápido, utiliza un lenguaje o un sistema operativo inapropiados.

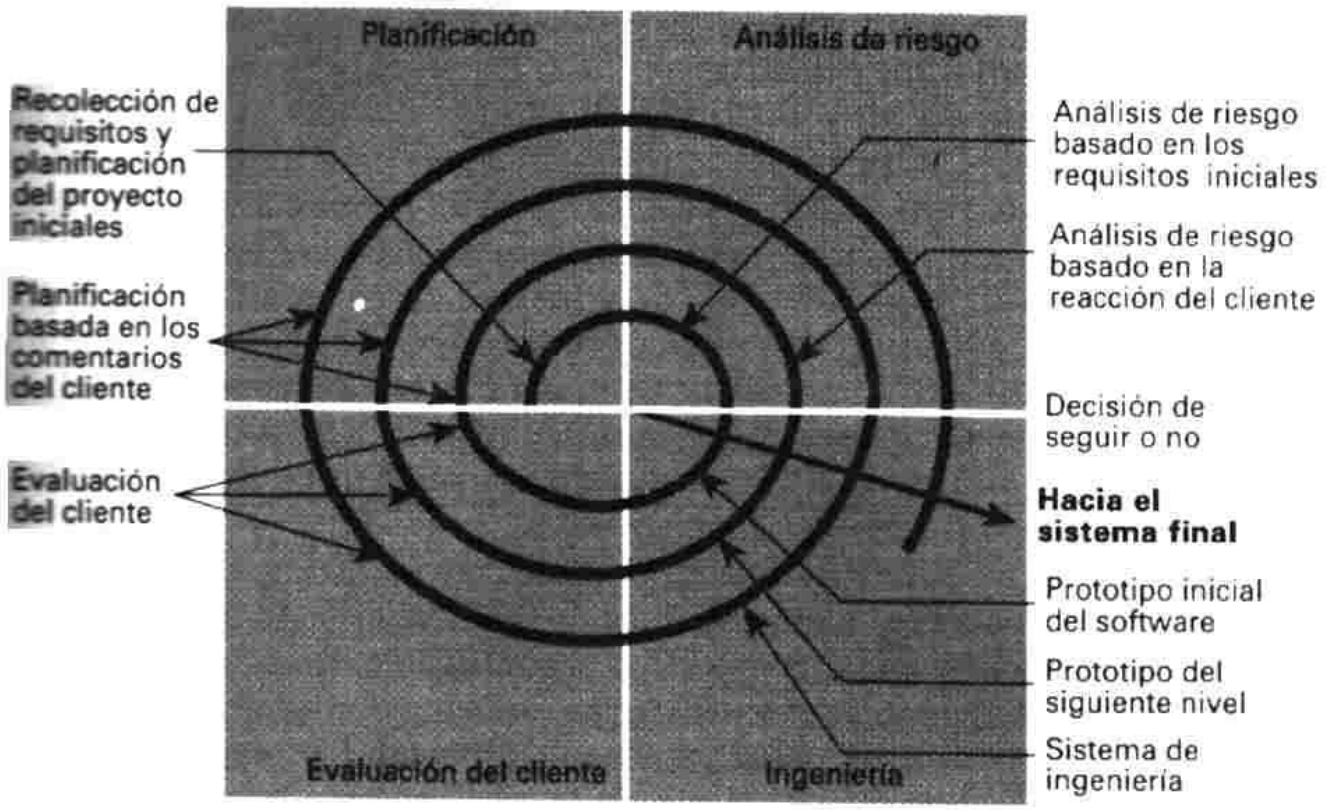
### El modelo en espiral

Combina las características de los dos paradigmas anteriores y añade al mismo tiempo nuevos elementos.
El modelo define cuatro actividades representadas en cuatro cuadrantes.

1. *Planificación*.- determinación de objetivos, alternativas y restricciones.
2. *Análisis de riesgo*.- identificación y resolución de riesgos que pueden hacer que el proyecto fracase.
3. *Ingeniería*.- desarrollo del producto de "siguiente nivel"
4. *Evaluación del cliente*.- valoración de los resultados de ingeniería.

En el modelo, con cada iteración de la espiral se construye una versión sucesiva del software, cada vez más completas.

Se empieza con la recolección de requisitos, luego se identifican y analizan los riesgos; si este análisis indica incertidumbre en los requisitos se puede crear un prototipo en el cuadrante de ingeniería. El cliente evalúa el trabajo de ingeniería y sugiere modificaciones. En base esos comentarios se produce la siguiente fase de planificación y análisis de riesgo. Si los riesgos son demasiado grandes se puede dar por terminado el proyecto. Cada vuelta de la espiral requiere de trabajo de ingeniería que puede ser mediante el enfoque de ciclo de vida clásico o con la creación de prototipos.




El paradigma es más realista pero se enfrenta muchas veces a que se requiera gran habilidad para valorar los riesgos. Si no se descubren los riesgos importantes indudablemente surgirán problemas.

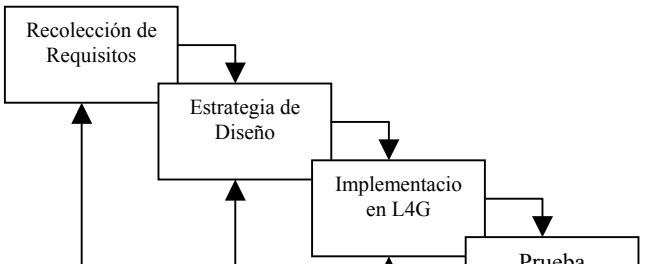
### Técnicas de Cuarta Generación ( T4G )

Abarca un amplio espectro de herramientas de software que facilitan el desarrollo del proyecto. Estas herramientas se orientan hacia la posibilidad de especificar el software a un nivel próximo al lenguaje natural. Las herramientas incluyen lenguajes no procedimentales, generadores de pantallas, informes, código automático, etc.

Se empieza con la recolección de requisitos. Para aplicaciones pequeñas se puede ir directamente al paso de implementación usando un lenguaje de cuarta generación no procedimental ( L4G ). Sin embargo es necesario invertir algo de esfuerzo en plantear una estrategia de diseño, ya que sin este tendríamos los mismos problemas de calidad y mantenimiento.

La implementación se hace con un lenguaje de cuarta generación L4G que permite especificar los resultados deseados y obtener el código fuente que produce tales resultados.

Se debe dirigir una prueba completa similar a la requerida en los paradigmas anteriores.



Las controversias sobre este paradigma:

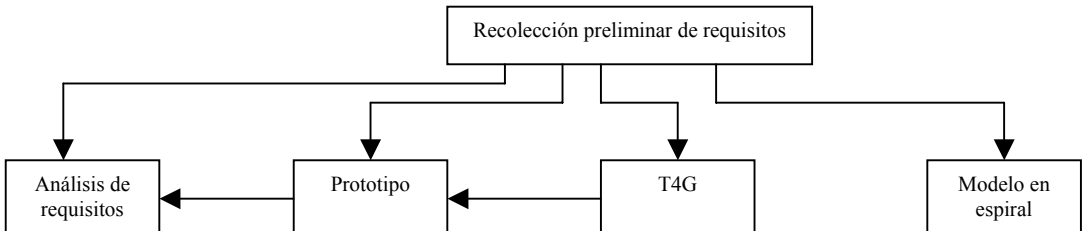
- Los defensores aducen una reducción significativa en el tiempo de desarrollo.
- Los detractores aducen que las herramientas producen código ineficiente.

Ambos tienen una parte de razón. Sin embargo las herramientas de cuarta generación han evolucionado a herramientas CASE sofisticadas que automatizan gran parte de las actividades de ingeniería y que se han vuelto indispensables para los desarrolladores de software.

### Combinación de Paradigmas

Ningún paradigma es infalible, en muchos casos los paradigmas deben combinarse para aprovechar las ventajas de cada uno en un único proyecto.

A continuación se muestra cómo pueden combinarse los paradigmas mencionados en un proyecto de software.



*** Considere que la naturaleza del proyecto dicta cuál paradigma elegir. ***

-oOo-