Modelo de Procesos de Software

- **Lineal Secuencial.**
- **Prototipo.**
- **Desarrollo de Aplicaciones Rápida. (D.R.A).**

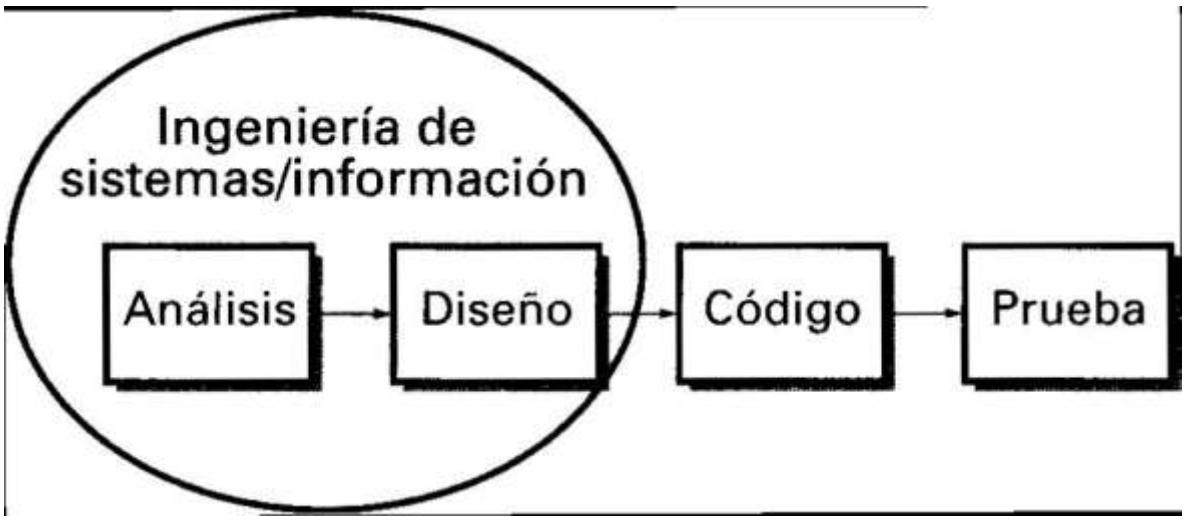
• Lineal Secuencial

- **Análisis de los requisitos del software.** El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza del (los) programa(s) a construirse, el ingeniero del software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento e interconexión.
- **Diseño. El diseño del software es realmente un proceso** de muchos pasos que se centra en cuatro atributos distintos de programa: estructura de datos, arquitectura de software, representaciones de interfaz y detalle procedimental (algoritmo). El proceso del diseño traduce requisitos en una representación del software donde se pueda evaluar su calidad antes de que comience la Codificación.
- **Generación de código. El diseño se debe traducir** en una forma legible por la máquina. El paso de generación de código lleva a cabo esta tarea. Si se lleva a cabo el diseño de una forma detallada, la generación de código se realiza mecánicamente. *Aunque el modelo lineal es a menudo denominado «modelo tradicional», resulto un enfoque razonable cuando los requisitos se han entendido correctamente.*
- **Pruebas. Una vez que se ha generado el código,** comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos.
- **Mantenimiento. El software indudablemente sufrirá** cambios después de ser entregado al cliente (una excepción posible es el software empotrado). Se producirán cambios porque se han encontrado errores, porque el software debe adaptarse para acoplarse a los cambios de su entorno externo (por ejemplo: se requiere un cambio debido a un sistema operativo o dispositivo periférico nuevo), o porque el cliente requiere mejoras funcionales o de rendimiento. El soporte y mantenimiento del software vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo. El modelo lineal secuencial es el paradigma más antiguo y más extensamente utilizado en la ingeniería del software. Sin embargo, la crítica del paradigma ha puesto en duda su

eficacia Entre los problemas que se encuentran algunas veces en el modelo lineal secuencial se incluyen:

- **¿Por qué falla algunas veces el modelo lineal?** Los proyectos reales raras veces siguen el modelo secuencial que propone el modelo. Aunque el modelo lineal puede acoplar interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el equipo del proyecto comienza.
- **A menudo es difícil que el cliente exponga explícitamente** todos los requisitos. El modelo lineal secuencial lo requiere y tiene dificultades a la hora de acomodar la incertidumbre natural al comienzo de muchos proyectos.
- **El cliente debe tener paciencia.** Una versión de trabajo del (los) programa(s) no estará disponible hasta que el proyecto esté muy avanzado. Un grave error puede ser desastroso si no se detecta hasta que se revisa el programa.

  Cada uno de estos errores es real. Sin embargo, el paradigma del ciclo de vida clásico tiene un lugar definido e importante en el trabajo de la ingeniería del software.
- **Proporciona una plantilla** en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento. El ciclo de vida clásico sigue siendo el modelo de proceso más extensamente utilizado por la ingeniería del software. Pese a tener debilidades, es significativamente mejor que un enfoque hecho al *azar para* el desarrollo del software.
-



- **PROTOTIPOS**

- **Un cliente, a menudo, define un conjunto de objetivos generales para el software,** pero no identifica los requisitos detallados de entrada, proceso o salida. En otros casos, el responsable del desarrollo del soft-

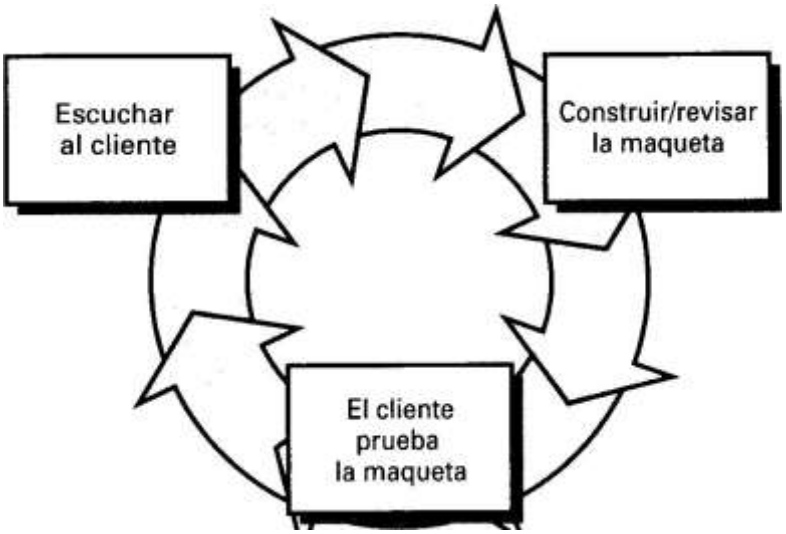
ware puede no estar seguro de la eficacia de un algoritmo, de la capacidad de adaptación de un sistema operativo, o de la forma en que debería tomarse la interacción hombre máquina.

En estas y en otras muchas situaciones, un paradigma de construcción de prototipos puede ofrecer el mejor enfoque.

- **El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición.** Entonces aparece un «diseño rápido». El diseño rápido se centra en una representación de esos aspectos del software que serán visibles para el usuario/cliente (por ejemplo: enfoques de entrada y formatos de salida). El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer.

  Cuando el cliente tiene uno necesidad legítima, pero está desorientado sobre los detalles, el primer paso es desarrollar un prototipo.

  Lo ideal sería que el prototipo sirviera como un mecanismo para identificar los requisitos del software. Si se construye un prototipo de trabajo, el desarrollador intenta hacer uso de los fragmentos del programa ya existentes o aplica herramientas (por ejemplo: generadores de informes, gestores de ventanas, etc.) que permiten generar rápidamente programas de trabajo.



- **El prototipo puede servir como «primer sistema».**

  El cliente ve lo que parece ser una versión de trabajo del software.

El desarrollador, a menudo, hace compromisos de implementación para hacer que el prototipo funcione rápidamente.

Se puede utilizar un sistema operativo o lenguaje de programación inadecuado simplemente porque está disponible y porque es conocido; un algoritmo eficiente se puede implementar simplemente para demostrar la capacidad. Después de algún tiempo, el desarrollador debe familiarizarse con estas selecciones, y olvidarse de las razones por las que son inadecuadas.

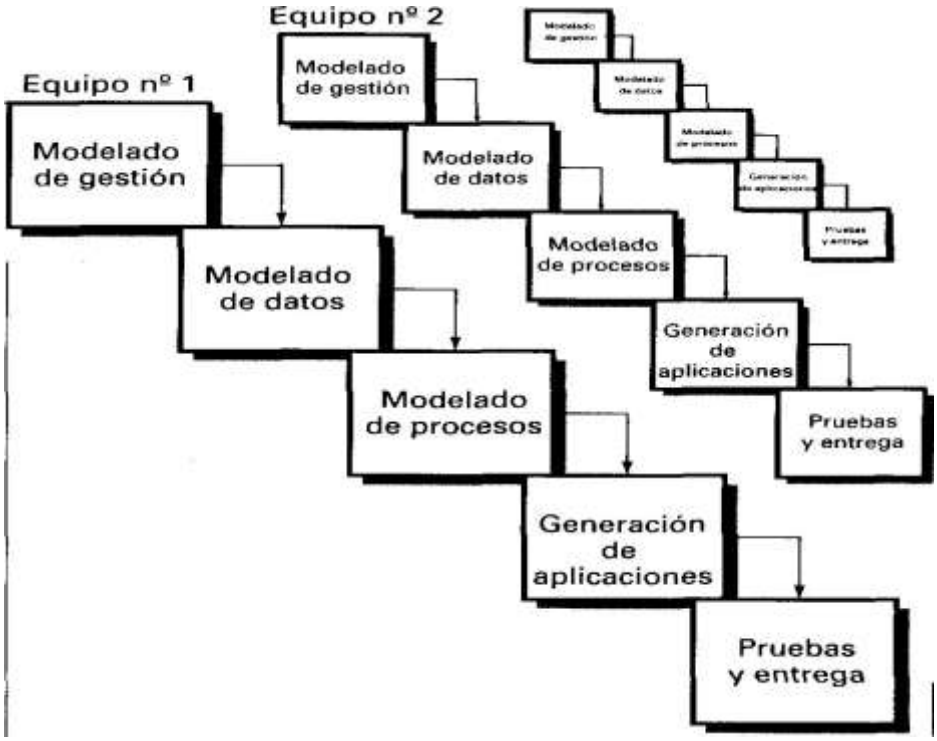
- ## Modelo Desarrollo de Aplicaciones Rápidas (D.R.A)

  - Es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto.

    El modelo DRA es **una adaptación a «alta velocidad» del modelo lineal secuencial en el que se logra el desarrollo rápido utilizando una construcción basada en componentes.**

    Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un «sistema completamente funcional»

  - **Modelado de datos**. El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.
  - **Modelado del proceso**. Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos.
  - **Generación de aplicaciones.** El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas para facilitar la construcción del software.



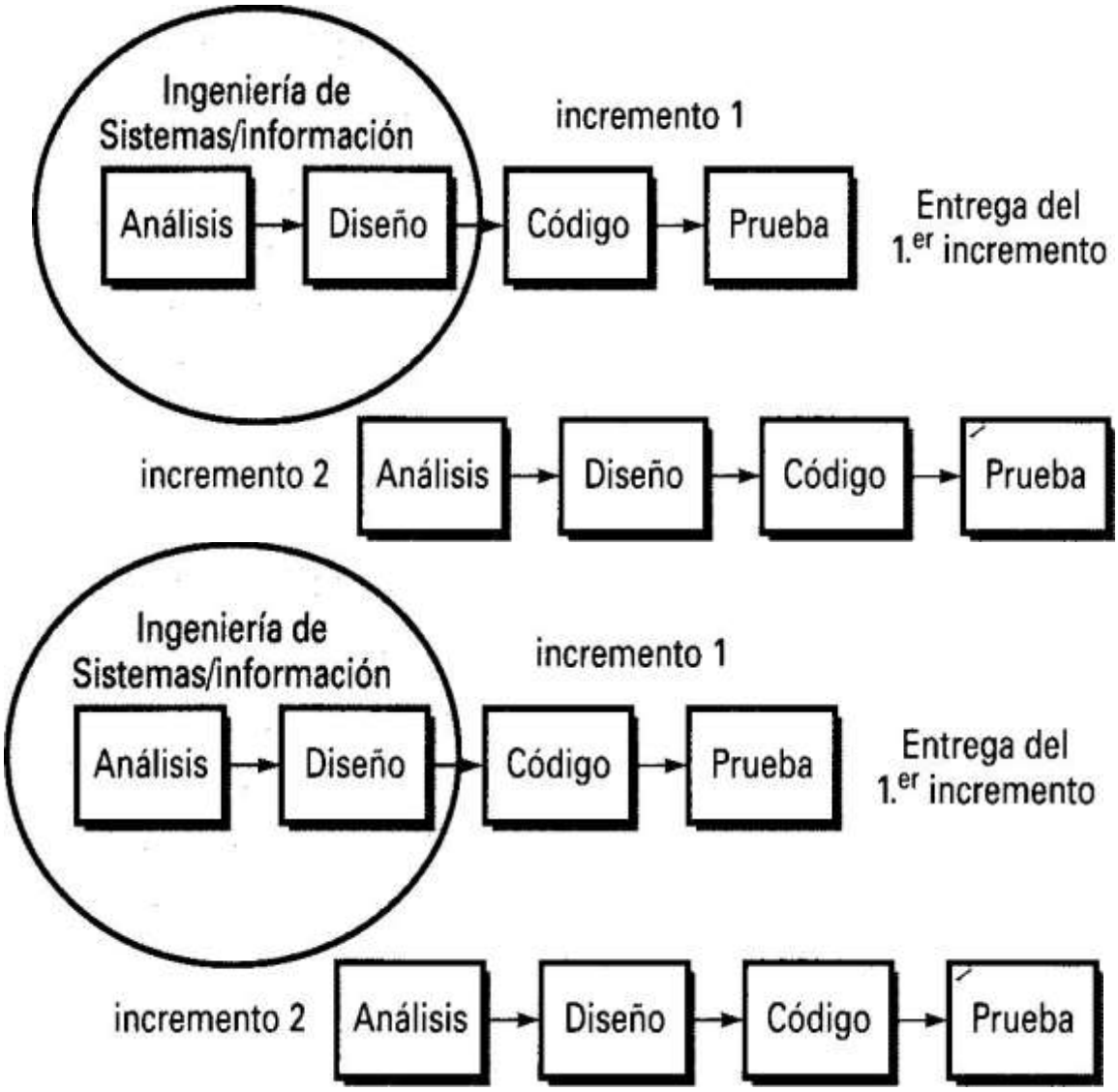
- **Pruebas y entrega.** Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.
- **Problemas:**
  - Para proyectos grandes aunque por escalas, el DRA requiere recursos humanos suficientes como para crear el número correcto de equipos DRA.
  - DRA requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias para completar un sistema en un marco de tiempo abreviado. Si no hay compromiso por ninguna de las partes constituyentes, los proyectos DRA fracasarán.
  - No todos los tipos de aplicaciones son apropiados para DRA. Si un sistema no se puede modularizar adecuadamente, la construcción de los componentes necesarios para DRA será problemático. Si está en juego el alto rendimiento, y se va a conseguir el rendimiento convirtiendo interfaces en componentes de sistemas, el enfoque DRA puede que no funcione.
  - DRA no es adecuado cuando los riesgos técnicos son altos. Esto ocurre cuando una nueva aplicación hace uso de tecnologías nuevas, o cuando el software nuevo requiere un alto grado de interoperabilidad con programas de computadora ya existentes.

Modelos Evolutivos

- **Modelo Incremental.**
- **Modelo Espiral.**

• Modelo Incremental

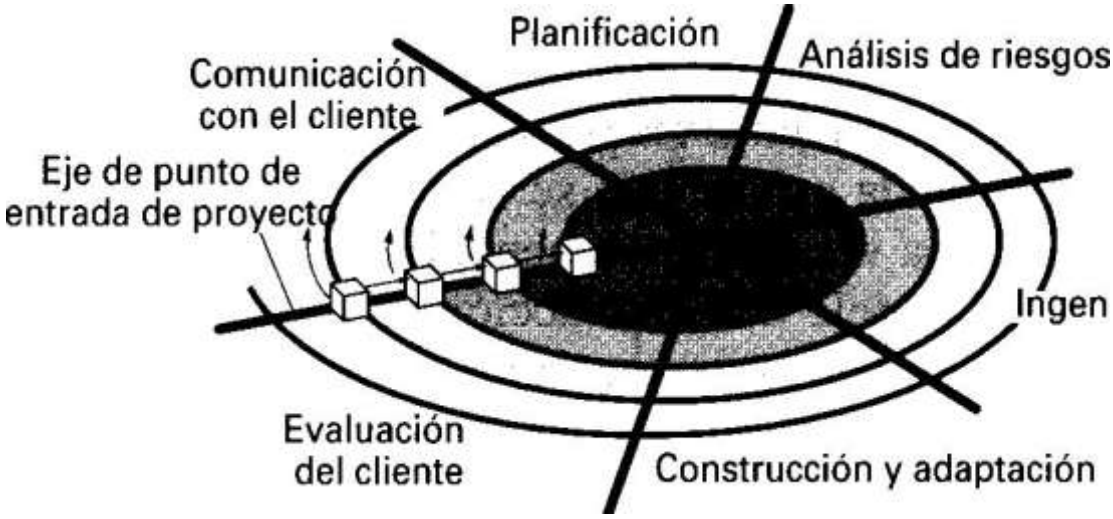
- El modelo incrernental combina elementos del modelo lineal secuencial (aplicados repetidamente) con la filosofía interactiva de construcción de prototipos.
- El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario.
- Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial. Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (algunas conocidas, otras no) quedan sin extraer.
- El cliente utiliza el producto central (o sufre la revisión detallada). Como un resultado de utilización y/o de evaluación, se desarrolla un plan para el incremento siguiente.
- El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales.
- Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo.
- **A partir de la Etapa de Diseño se empieza con análisis y se van realizando cada entrega del incremento** (hasta llegar por ejemplo al 4to incremento).



- **Modelo Espiral**

  - El modelo en espiral es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial.
  - Proporciona el potencial para el desarrollo rápido de versiones incrementales del software diseñado.
  - En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.
  - El modelo Espiral tiene seis regiones:
    - Comunicación con el cliente- las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

- Planificación- las tareas requeridas para definir recursos, el tiempo y otra información relacionadas con el proyecto.
- Análisis de riesgos- las tareas requeridas para evaluar riesgos técnicos y de gestión.
- Ingeniería- las tareas requeridas para construir una o más representaciones de la aplicación.
- Construcción y acción- las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario (por ejemplo: documentación y práctica).
- Evaluación del cliente- las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.



Cada una de las regiones está compuesta por un conjunto de tareas del trabajo, llamado conjunto de tareas, que se adaptan a las características del proyecto que va a emprenderse.

Para proyectos pequeños, el número de tareas de trabajo y su formalidad es bajo.

Para proyectos mayores y más críticos cada región de tareas contiene tareas de trabajo que se definen para lograr un nivel más alto de formalidad.

Cuando empieza este proceso evolutivo, el equipo de ingeniería del software gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral puede producir el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones.

El modelo en espiral es un enfoque realista del desarrollo de sistemas y de software a gran escala. Como el software evoluciona, a medida que progresa el proceso ambos, desarrollador y cliente, comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos.

El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero, lo que es más importante, permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.

Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo iterativo que refleja de forma más realista el mundo real.

El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto, y, si se aplica adecuadamente, debe reducir los riesgos antes de que se conviertan en problemáticos.

- ## Modelo en Espiral WINWIN

El modelo en espiral tratado sugiere una actividad del marco de trabajo que aborda la comunicación con el cliente.

El objetivo de esta actividad es mostrar los requisitos del cliente. En un contexto ideal, el desarrollador simplemente pregunta al cliente lo que se necesita y el cliente proporciona detalles suficientes para continuar. Desgraciadamente, esto raramente ocurre. En realidad el cliente y el desarrollador entran en un proceso de negociación, donde el cliente puede ser preguntado para sopesar la funcionalidad, rendimiento, y otros productos o características del sistema frente al coste y al tiempo de comercialización.

Las mejores negociaciones se esfuerzan en obtener «victoria-victoria». Esto es, el cliente gana obteniendo el producto o sistema que satisface la mayor parte de sus necesidades y el desarrollador gana trabajando para conseguir presupuestos y lograr una fecha de entrega realista.

El modelo en espiral WINWIN de Boehm define un conjunto de actividades de negociación al principio de cada paso alrededor de la espiral. Más que una simple actividad de comunicación con el cliente, se definen las siguientes actividades:

1. Identificación del sistema o subsistemas clave de los «directivos»
2. Determinación de las «condiciones de victoria» de los directivos.
3. Negociación de las condiciones de «victoria» de los directivos para reunirlas en un conjunto de condiciones «victoria-victoria» para todos los afectados (incluyendo el equipo del proyecto de software).
4. Valida las definiciones del producto y del proceso.

5. Definir el siguiente nivel del proceso y producto incluyendo peticiones
6. Valida las definiciones del producto y el proceso
7. Revisión, compromiso

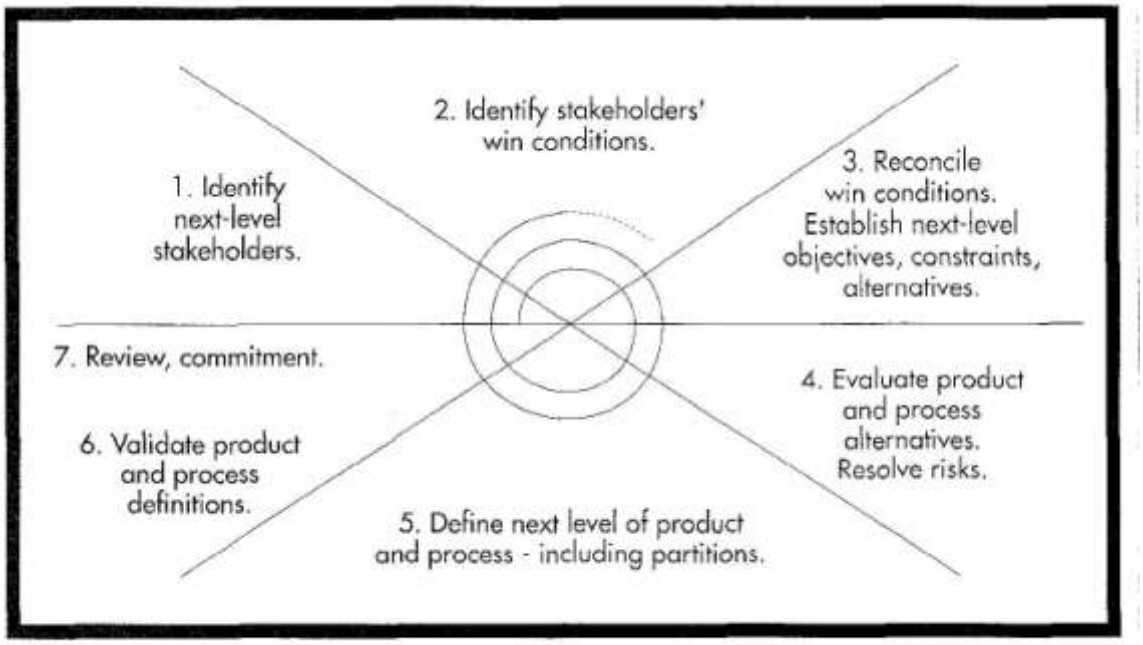


Figure 1. The Win-Win spiral model.

El primer punto de fijación, llamado objetivos del ciclo de vida (OCV), define un conjunto de objetivos para cada actividad principal de ingeniería del software. Como ejemplo, de una parte de OCV, un conjunto de objetivos asociados a la definición de los requisitos del producto/sistema del nivel más alto.

El segundo punto de fijación, llamado arquitectura del ciclo de vida (ACV), establece los objetivos que se deben conocer mientras que se define la arquitectura del software y el sistema. Como ejemplo, de una parte de la ACV, el equipo del proyecto de software debe demostrar que ha evaluado la funcionalidad de los componentes del software reutilizables y que ha considerado su impacto en las decisiones de arquitectura.

La capacidad operativa inicial (COI) es el tercer punto de fijación y representa un conjunto de objetivos asociados a la preparación del software para la instalación/distribución, preparación del lugar previamente a la instalación, y la asistencia precisada de todas las partes que utilizará o mantendrá el software.

- **Modelo de Desarrollo Concurrente**

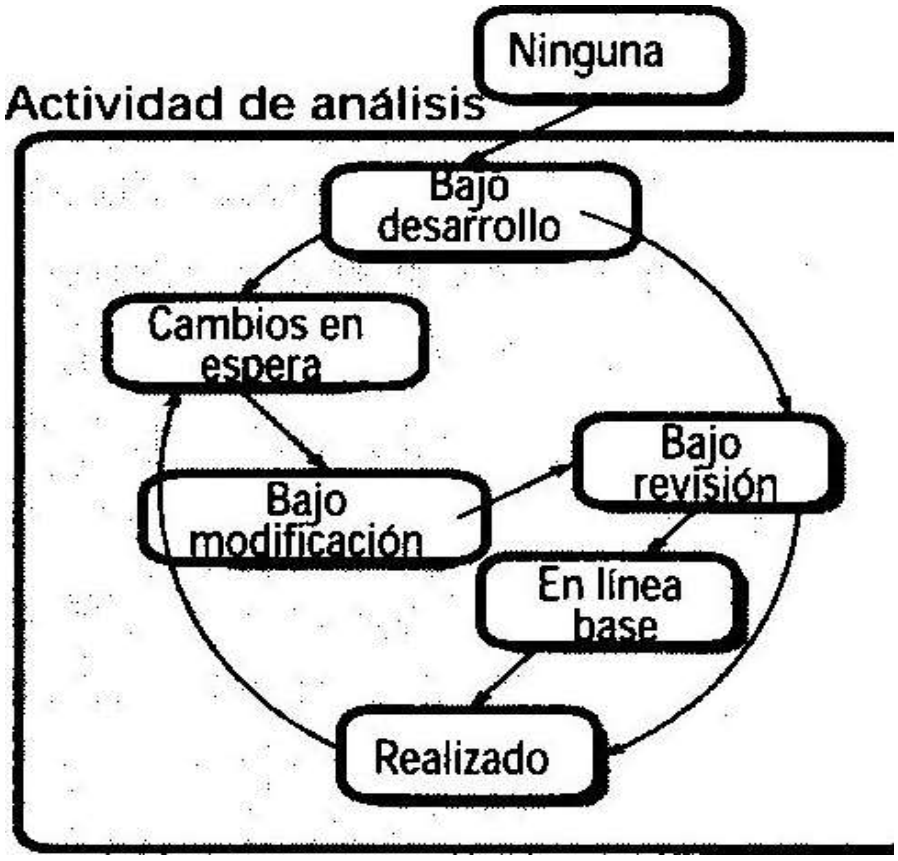
El modelo de proceso concurrente se puede representar en forma de esquema como una serie de actividades, técnicas importantes, tareas y estados asociados a ellas.

La actividad -análisis- se puede encontrar en uno de los estados"' destacados anteriormente en cualquier momento dado. De forma similar, otras actividades (por ejemplo: diseño o comunicación con el cliente) se puede representar de una forma análoga.

Todas las actividades existen concurrentemente, pero residen en estados diferentes. Por ejemplo, al principio del proyecto la actividad de comunicación con el cliente (no mostrada en la figura) ha finalizado su primera iteración y está en el estado de cambios, en espera.

La actividad de análisis (que estaba en el estado ninguno mientras que se iniciaba la comunicación inicial con el cliente) ahora hace una transición al estado bajo desarrollo.

Sin embargo, si el cliente indica que se deben hacer cambios en requisitos, la actividad análisis cambia del estado bajo desarrollo al estado cambios en Espera.



- **Desarrollo Basado en Componentes**

  El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo en espiral.

  Es evolutivo por naturaleza [NIE92], y exige un enfoque iterativo para la creación del software.

  Sin embargo, el modelo de desarrollo basado en componentes configura aplicaciones desde componentes preparados de software llamados «clases»

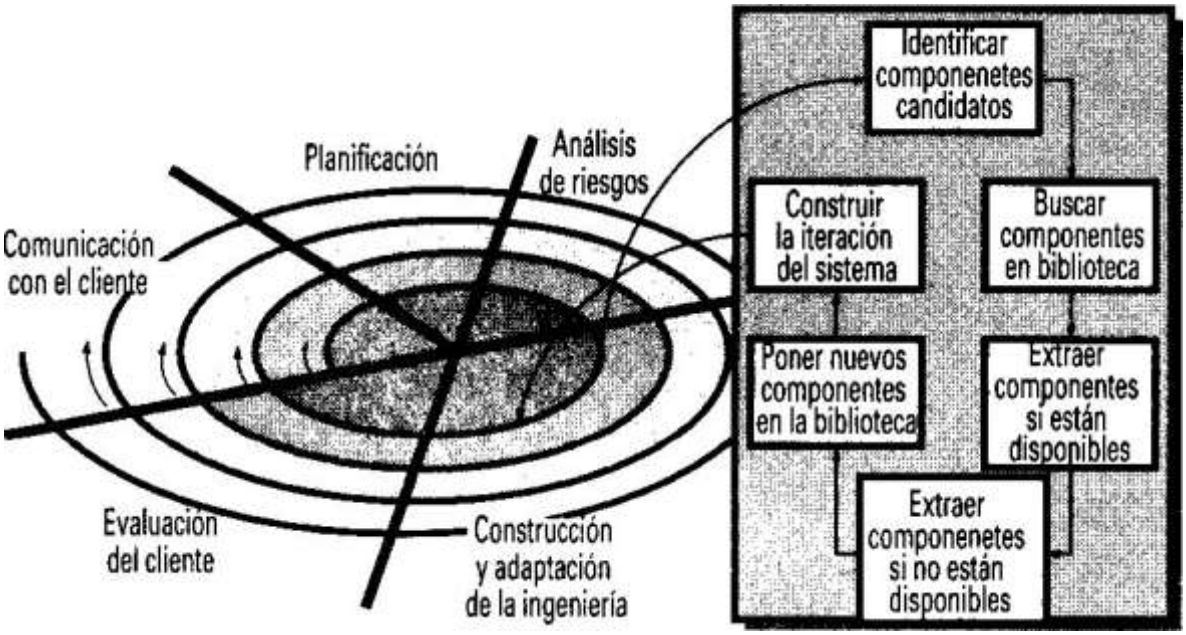
  La actividad de la ingeniería comienza con la identificación de clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la

  aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento.

  Los datos y los algoritmos correspondientes se empaquetan en una clase.

  Las clases creadas en los proyectos de ingeniería del software anteriores, se almacenan en una biblioteca de clases o diccionario de datos (repository) (Capítulo 3 1).

  Una vez identificadas las clases candidatas, la biblioteca de clases se examina para determinar si estas clases ya existen. En caso de que así fuera, se extraen de la biblioteca y se vuelven a utilizar. Si una clase candidata no reside en la biblioteca, se aplican los métodos orientados a objetos



- **Modelo de métodos formales**

El modelo de métodos formales comprende un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten que un ingeniero de software especifique, desarrolle y verifique un sistema basado en computadora aplicando una notación rigurosa y matemática.

Cuando se utilizan métodos formales durante el desarrollo, proporcionan un mecanismo para eliminar muchos de los problemas que son difíciles de superar con paradigmas de la ingeniería del software. La ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen más fácilmente –no mediante un revisión a propósito para el caso, sino mediante la aplicación del análisis matemático-. Cuando se utilizan métodos formales durante el diseño, sirven como base para la verificación de programas y por descubra y corrija errores que no se pudieron detectar de otra manera.

Aunque todavía no hay un enfoque establecido, los modelos de métodos formales ofrecen la promesa de un software libre de defectos. Sin embargo, se ha hablado de una gran preocupación sobre su aplicabilidad en un entorno de gestión:

1. El desarrollo de modelos formales actualmente es bastante caro y lleva mucho tiempo.
2. Se requiere un estudio detallado porque pocos responsables del desarrollo de software tienen los antecedentes necesarios para aplicar métodos formales.
3. Es difícil utilizar los modelos como un mecanismo de comunicación con clientes que no tienen muchos conocimientos técnicos.