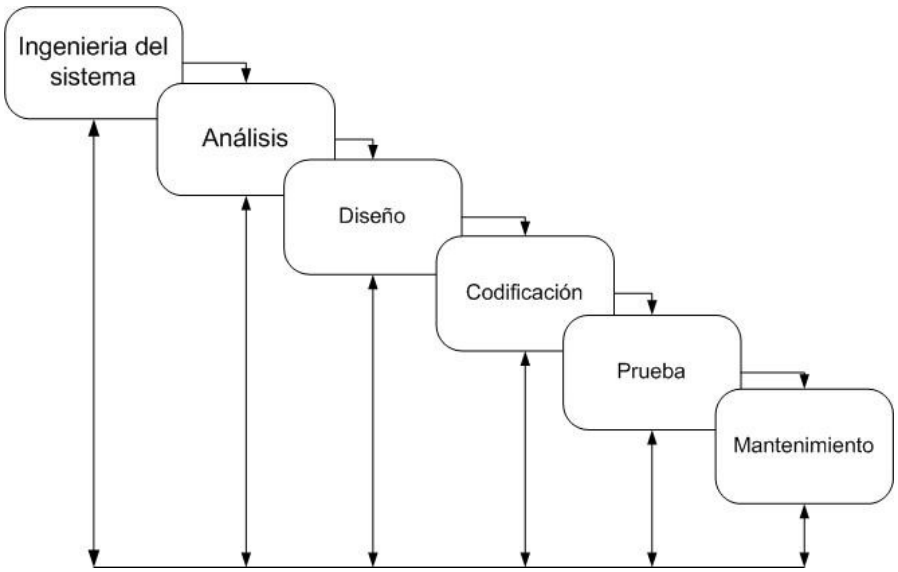
1.- Introducción

Proceso:
El proceso de software es una serie de pasos que involucran actividades, restricciones y recursos que producen un resultado esperado.

Modelo de proceso de software (representación abstracta del proceso):
Para resolver los problemas reales de una industria, los ingenieros del software deben incorporar una estrategia de desarrollo que acompañe al proceso, métodos y herramientas (capas de la ingeniería de software) y las fases genéricas (definición, desarrollo, mantenimiento). Esta estrategia a menudo se llama *modelo de proceso o paradigma de ingeniería del software.* Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse, y los controles y entregas que se requieren.

Modelo lineal secuencial:
Llamado algunas veces *ciclo de vida básico* o *modelo en cascada*, el *modelo lineal secuencial* sugiere un enfoque sistemático, secuencial, para el desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento.



El paradigma del ciclo de vida abarca las siguientes actividades:

a) *Ingeniería del sistema/información:* (Planeamiento). Debido a que el software es siempre parte de un sistema mayor, el trabajo

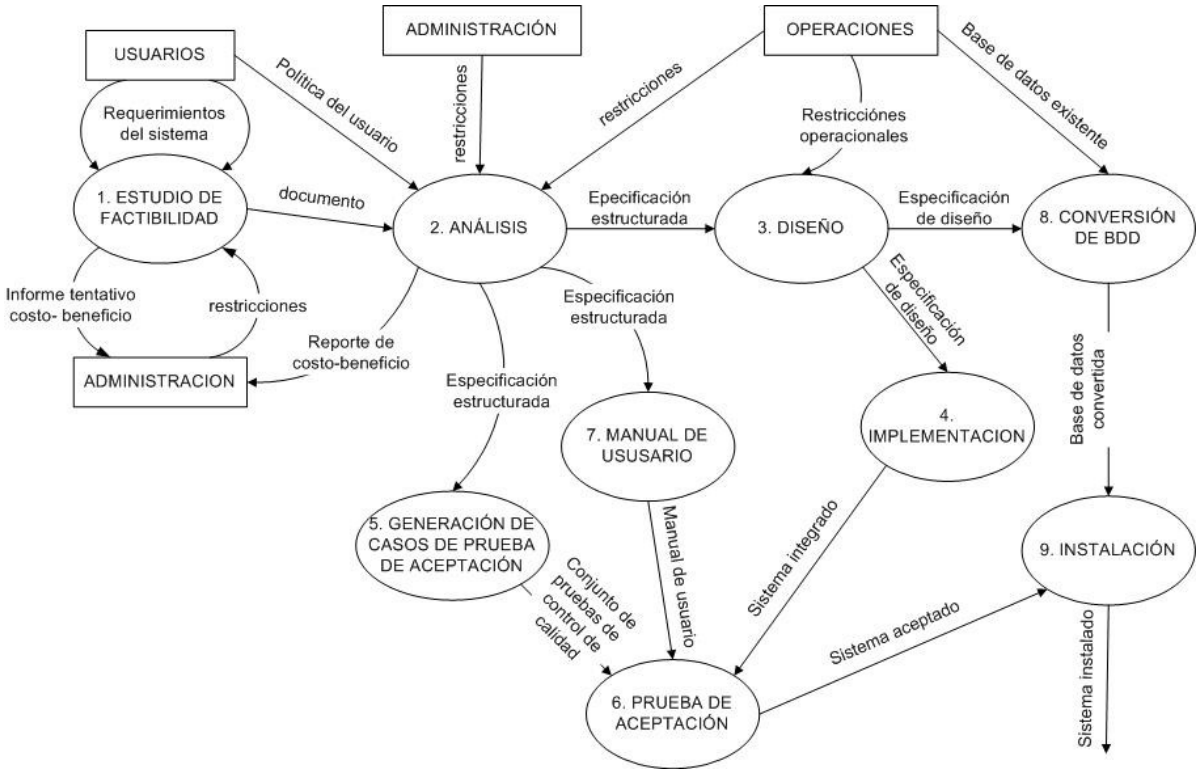
comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software. Este planeamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos tales como hardware, personas y bases de datos. La ingeniería de información abarca los requisitos que se recogen en el nivel de empresa estratégico y en el nivel del área de negocio.

b) *Análisis de requisitos:* El proceso de recopilación de los requisitos se centraliza e intensifica especialmente para el software. Para comprender la naturaleza de los programas a construir debemos comprender el ámbito de la información, así como la función, el rendimiento y las interfaces requeridas. Los requisitos tanto del sistema como del software se documentan y revisan con el cliente.
c) *Diseño:* se enfoca sobre cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de comenzar la codificación. Se documenta y forma parte de la configuración del software.
d) *Codificación:* el diseño debe traducirse en una forma legible en la máquina. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.
e) *Prueba:* la prueba se centra en la lógica interna del software, asegurando que todas las sentencias se han probado, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.
f) *Mantenimiento:* el software sufrirá cambios después de entregado al cliente. Los cambios ocurrirán debido a que se hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo, o debido a que el cliente requiere ampliaciones funcionales o del rendimiento.

Problemas:

1) Los proyectos reales raramente siguen el flujo secuencial. Siempre hay interacciones y se crean problemas en la aplicación del paradigma.
2) Es difícil para el usuario definir todos los requisitos al principio.
3) El cliente debe tener paciencia. Hasta la etapa final, no estará disponible la versión operativa del programa. Un error importante no detectado hasta que el programa este funcionando puede ser desastroso.

Ciclo de vida estructurado



Actividad 1: La encuesta

- ✓ Identificar los usuarios responsables y crear un "campo de actividad" inicial del sistema.
- ✓ Identificar las deficiencias del ambiente actual del usuario.
- ✓ Establecer metas y objetivos para el sistema nuevo.
- ✓ Determinar si es factible la automatización del sistema.
- ✓ Preparar el esquema de guía de la ejecución del proyecto.

Actividad 2: Análisis del sistema

La meta de esta actividad es transformar el esquema del proyecto y los objetivos del usuario en una especificación estructurada, es decir, modelar el ambiente de usuario con diagramas de flujo de datos, diagramas entidad-relación, diagramas de transición de estados y demás herramientas de análisis.

Actividad 3: Diseño

El diseño debe asignar partes de la especificación (módulos, procesos, subsistemas) creando una jerarquía apropiada de módulos y programas soportada por una estructura de hardware.
Se transforma el modelo entidad-relación en un diseño de bases de datos.
Se define el modelo de implementación del usuario.

Actividad 4: Implementación

Esta actividad consiste en codificar e integrar los módulos definidos en el diseño. Esto incluye Programación Estructurada e Implementación Descendente.

Actividad 5: Generación de pruebas

La especificación estructurada debe tener toda la información necesaria para definir un sistema que sea aceptable para el usuario. Por esto, una vez generada la especificación, puede comenzarse la preparación de un conjunto de casos de prueba de aceptación del sistema.
Esta actividad normalmente se superpone con las de diseño e implementación.

Actividad 6: Garantía de calidad

Esta actividad, también denominada prueba final del sistema, significa verificar mediante una prueba exhaustiva que el sistema cumple con las especificaciones convenidas. En realidad en el marco del ciclo de vida del software, cada una de las etapas debe concluir con una prueba final de verificación de calidad (Desde la documentación inicial hasta el código final).

Actividad 7: Procedimiento

El manual del usuario es el resultado final como documentación a emplear por los operadores del sistema. Este manual puede tener diferentes implementaciones y presentaciones para el usuario, tendiéndose cada vez más a que el usuario disponga de una interacción dinámica (en tiempo de ejecución) con el sistema de modo de consultar todos los elementos de la documentación auxiliar del sistema sobre la misma máquina.

Actividad 8: Conversión de base de datos

En muchas oportunidades de migración de un sistema a nuevas tecnologías, nos vemos en la obligación de transformar la información ya existente. Es muy importante el trabajo de análisis y diseño para poder realizar la transformación de datos existentes a las nuevas estructuras.

Actividad 9: Instalación

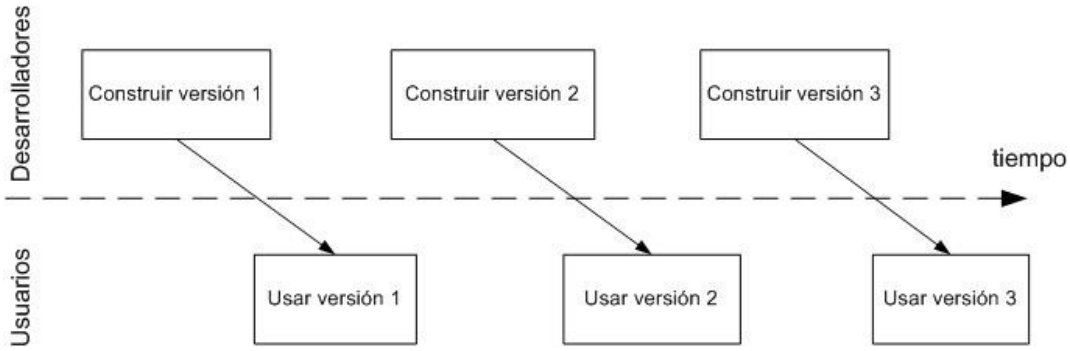
El paso de incorporación final del sistema automatizado puede ser gradual o sin transición. En ocasiones es conveniente mantener actividades manuales paralelas.
Es muy importante tener definidos los conjuntos de tests que permitirán abandonar la transición y declarar el sistema como "instalado".

El desarrollo de estas 9 actividades tiene dos enfoques: radical y conservador. En el enfoque radical las 9 actividades se realizan paralelamente, mientras que en el enfoque conservador deben realizarse secuencialmente. La elección entre estos dos enfoques depende de:

- ✓ Que tan cambiante es el usuario.
- ✓ Cuál es la presión para producir resultados tangibles e inmediatos.
- ✓ Cuál es la presión para producir un presupuesto y estimaciones de recursos.
- ✓ Cuales son los peligros de cometer un error técnico importante.

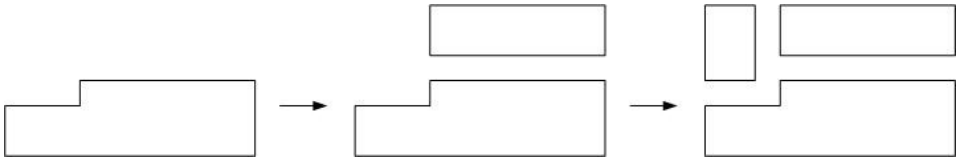
El método radical requiere gran coordinación, puede ser usado para proyectos de investigación y permite tener algo funcionando rápidamente.
El método conservador es recomendado para proyectos muy grandes y costosos.

Desarrollo por fases:
Se desarrolla el sistema de tal manera que puede ser entregado en piezas. Esto implica que existen dos sistemas funcionando en paralelo: el sistema operacional y el sistema en desarrollo.



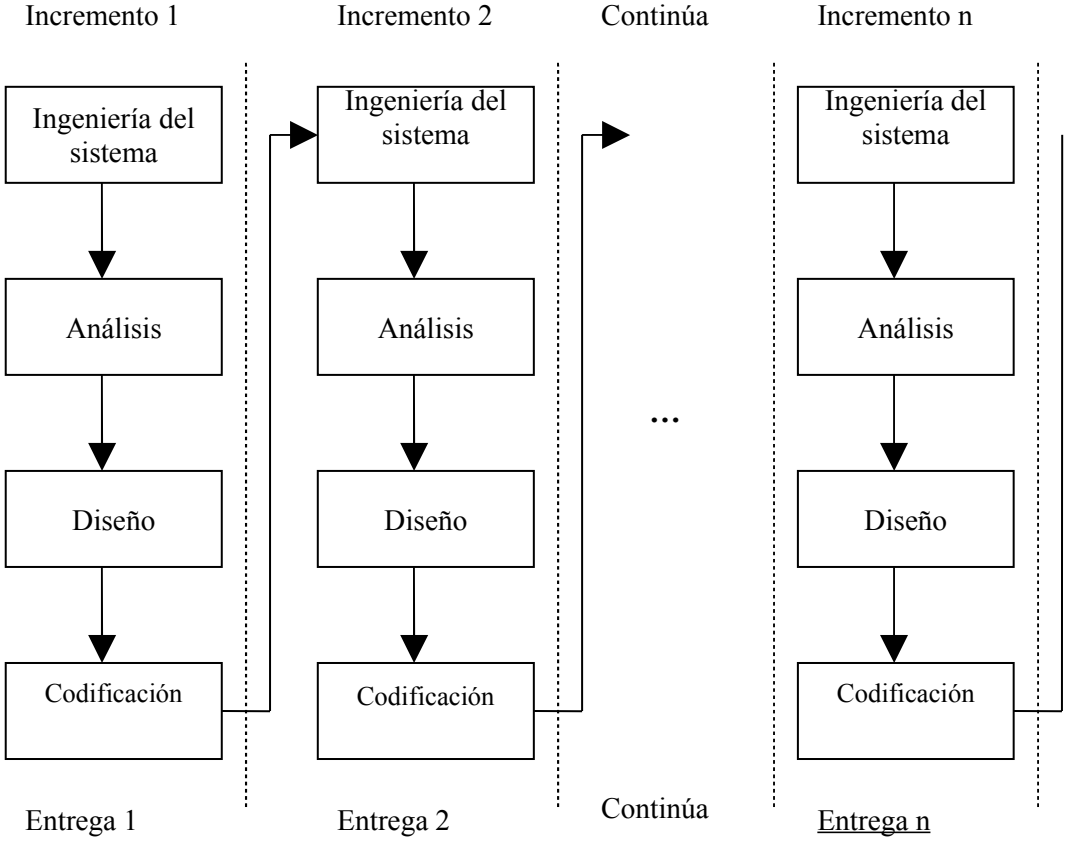
Dos maneras distintas del desarrollo por fases: desarrollo Incremental y desarrollo iterativo.

Desarrollo incremental:

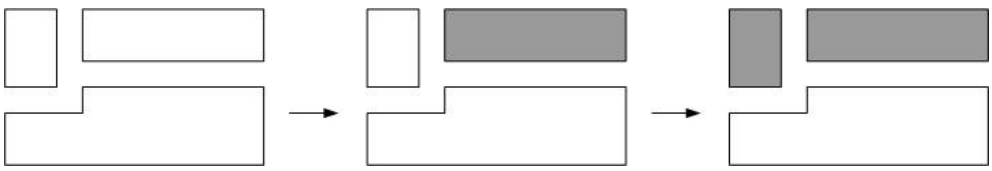


El modelo incremental corrige la necesidad de una secuencia no lineal de pasos de desarrollo, en este modelo se va creando el sistema software añadiendo componentes funcionales al sistema, llamados incrementos. En cada paso sucesivo, se actualiza el sistema con nuevas funcionalidades o requisitos, es decir, cada versión o refinamiento parte de una versión previa y le añade nuevas funciones.

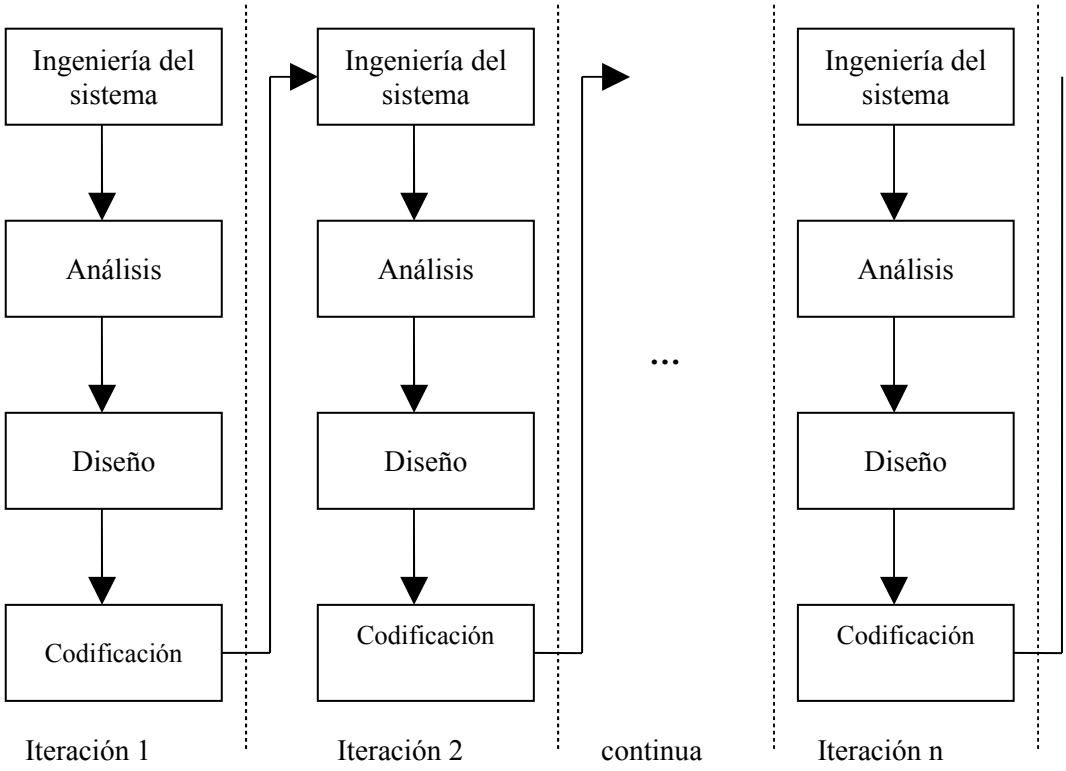
El modelo incremental se ajusta a entornos de alta incertidumbre, por no tener la necesidad de tener un conjunto exhaustivo de requisitos, especificaciones, diseño, etc., al comenzar el sistema, ya que cada refinamiento amplía los requisitos y las especificaciones derivadas de la fase anterior.
El modelo incremental constituyó un avance sobre el modelo en cascada, pero también presenta problemas. Aunque permite el cambio continuo de requisitos, aún existe el problema de determinar si los requisitos son válidos. Los errores en los requisitos se detectan tarde y su corrección resulta tan costosa como en el modelo en cascada.



Desarrollo iterativo:



Entrega un sistema completo desde el principio y luego aumenta la funcionalidad de cada subsistema con las nuevas versiones.



Modelo en espiral

Ha sido desarrollado para cubrir las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: el análisis de riesgo, que falta en esos paradigmas.

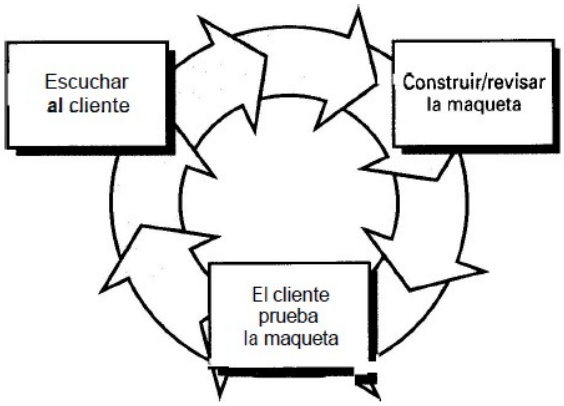
El modelo en espiral define cuatro actividades principales:

1. *Planificación*: determinación de los objetivos, alternativas y restricciones.
2. *Análisis de riesgo*: análisis de las alternativas e identificación/resolución de riesgos.
3. *Ingeniería*: desarrollo del producto del siguiente nivel.
4. *Evaluación del cliente*: valoración de los resultados de la ingeniería.

Con cada vuelta alrededor de la espiral se construye una nueva versión del software, cada vez más completa. Durante la primera vuelta se definen los objetivos, alternativas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay incertidumbre en los requisitos, en la etapa del análisis de riesgo, se puede utilizar la creación de prototipos en la parte de la ingeniería, dando así, asistencia al encargado del desarrollo como al cliente.
Luego, el cliente evalúa el trabajo de la ingeniería y sugiere modificaciones. De acuerdo a los comentarios del cliente, se produce la siguiente fase de planificación y de análisis de riesgo. Cada vuelta alrededor de la espiral, la culminación del análisis de riesgo resulta en la decisión de seguir, o no seguir. Si los riesgos son demasiado grandes, se puede dar por terminado el proyecto. En la mayoría de los casos se sigue avanzando alrededor de la espiral, y el camino lleva a los desarrolladores hacia un modelo más completo del sistema, y al final al propio sistema operacional.
Cada vuelta requiere ingeniería, que puede llevarse a cabo mediante el enfoque del ciclo de vida clásico o de la creación de prototipos.
Este paradigma es el enfoque más realista del desarrollo de software y de sistemas a gran escala. Utiliza un enfoque evolutivo, permitiendo al desarrollador y al cliente entender y reaccionar a los riesgos en cada nivel evolutivo, antes de que se conviertan en problemáticos. Puede ser difícil de convencer a grandes clientes de que el enfoque evolutivo es controlable. Requiere una considerable habilidad para la valoración del riesgo y cuenta con esta habilidad para el éxito.

Prototipos:

Un prototipo es un producto parcialmente desarrollado que permite que clientes y desarrolladores examinen algunos aspectos del sistema propuesto, y decidan si éste es adecuado o correcto para el producto terminado.
Esta es una alternativa de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales.



Hay dos formas distintas de implementar el desarrollo por prototipos: Desechable y evolutivo

***Prototipo desechable*:**
El objetivo es obtener el conjunto de requerimientos y luego se combina con otro paradigma para obtener el producto final.
Este enfoque de prototipos supone que el modelo simulará alguna o todas las funciones que el usuario desea y que al concluirse el modelado los programas se descartarán y reemplazarán con programas reales.
Es peligroso tentarse a adoptarlo como producto final
Si el prototipo se descarta y se reemplaza con el sistema real puede que no quede documentación de los requerimientos del usuario

Prototipo evolutivo:
El objetivo es obtener el sistema a entregar.
Permite que todo el sistema o alguna de sus partes se construyan rápidamente para comprender o aclarar aspectos y asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución.

Proceso Unificado:
Es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental.

Características:
Iterativo e Incrementa:
El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio sólo consta de varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un *incremento* del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

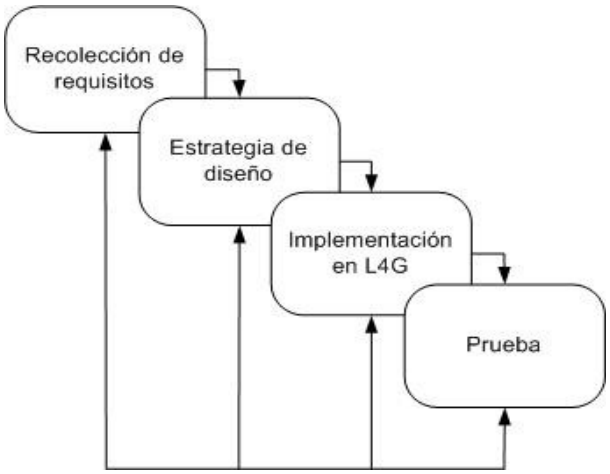
Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

Dirigido por los casos de uso:
En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.

Centrado en la arquitectura:
El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

Enfocado en los riesgos:
El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero

Técnicas de cuarta generación.
El término "técnicas de cuarta generación" (T4G) abarca un amplio espectro de herramientas de software que tienen algo en común: todas facilitan, al que desarrolla el software, la especificación de algunas características del software a alto nivel. Luego, la herramienta genera automáticamente el código fuente basándose en la especificación.
Un entorno para el desarrollo de software que soporte el paradigma T4G puede incluir todas o algunas de las siguientes herramientas: lenguajes no procedimentales para consulta a bases de datos, generación de informes, manipulación de datos, interacción y definición de pantallas, generación de código, facilidades gráficas de alto nivel y facilidades de hoja de cálculo. Todas estas herramientas están disponibles, pero solo para ámbitos de aplicación muy específicos
Pasos:



Al igual que otros paradigmas, T4G comienza con la recolección de requisitos, el cliente describe los requisitos, que son a continuación traducidos directamente a un prototipo operativo. Sin embargo en la práctica no se puede hacer eso. El cliente puede no estar seguro de lo que necesita, puede ser incapaz o no desear especificar la información en la forma en que una herramienta de cuarta generación puede aceptarla.
Para aplicaciones pequeñas se puede ir directamente desde el paso de recolección de requisitos al paso de implementación, usando un lenguaje de cuarta generación no procedimental (L4G). Sin embargo, es necesario mayor esfuerzo para desarrollar una estrategia de diseño para el sistema. El uso de tecnología de cuarta generación sin diseño, para grandes proyectos, causará las mismas dificultades (poca calidad, mantenimiento pobre, mala aceptación por el cliente) que se encuentran cuando se desarrolla software mediante los enfoques convencionales.
La implementación en lenguajes de cuarta generación permite, al que desarrolla el software, centrarse en la representación de los resultados deseados, que es lo que se traduce automáticamente en un código fuente que produce dichos resultados. Obviamente, debe existir una estructura de datos con información relevante y a la que el L4G pueda acceder rápidamente.
Para transformar una implementación con tecnología de cuarta generación en un producto, el que lo desarrolla debe dirigir una prueba completa, desarrollar una documentación con sentido y ejecutar el resto de actividades de "transición" requeridas en los otros paradigmas.
Ventajas:
Los defensores aducen reducciones dramáticas en el tiempo de desarrollo del software y una mejora en la productividad de la gente que construye software.
Desventajas:
Los detractores aducen que las herramientas actuales de cuarta generación no son fáciles de usar, que el código fuente es ineficiente y el mantenimiento está abierto a discusión.
Ambas posturas tienen su parte de razón. Aunque son difíciles los hechos de las suposiciones pues hay pocos estudios controlados.

1. En la actualidad su ámbito está limitado a sistemas de información de gestión, concretamente al análisis de información y a la obtención de informes relativos a grandes bases de datos.
2. En aplicaciones pequeñas y de tamaño medio, los estudios indican que se reduce el tiempo requerido para producir software, y que la cantidad de análisis y diseño para las aplicaciones pequeñas, también se reduce.
3. Para grandes trabajos exige el mismo o más tiempo de análisis, diseño y prueba.

Combinación de Paradigmas

En muchos casos los paradigmas pueden y deben considerarse de forma que puedan utilizarse las ventajas de cada uno en un único proyecto.
En todos los casos se comienza con la determinación de objetivos, alternativas y restricciones, pasos que se llaman "recolección preliminar de requisitos". A partir de ese punto se puede tomar cualquier camino