Unidad 1-Introducción a la IS

Objetivo: proporcionar una visión general de los productos, procesos y modelos de desarrollo contemplados en la ingeniería de software.

¿Qué es la Ingeniería de Software?

- Establecimiento y uso de principios con caracteres de ingeniería; métodos, herramientas y procedimientos apropiados y orientados a obtener eficiente y económicamente software fiable, que opere eficaz y eficientemente en máquinas reales.
- El concepto surgió en 1968 en la conferencia de la OTAN en Alemania con la intención de que mediante el uso de filosofías y paradigmas de disciplinas ingenieriles establecidas se resolviera la denominada crisis del software.
- Los ingenieros de software adoptan un enfoque sistemático para llevar a cabo su trabajo y utilizan las herramientas y técnicas necesarias para resolver el problema planteado de acuerdo a las restricciones del desarrollo y recursos disponibles.

Diferencia entre Ingenieria de Software y Ciencias de la Computación

Las ciencias de la computación conciernen a la teoría y fundamentos de cualquier sistema de cómputo, sea de hardware o software.

La ingeniería de software concierne sólo al desarrollo de sistemas o productos de software.

Diferencia entre ingeniería de Software e Ingeniería de Sistemas

La Ingeniería de Sistemas tiene en cuenta a todos los elementos del desarrollo de sistemas basados en cómputo que incluyen hardware, software, base de datos, recursos humanos y afines. La Ingeniería de Software forma parte de este proceso.

¿Qué es el software?

Somerville: programas de cómputo y su documentación asociada

Presman: elemento del sistema de información que es lógico, con características considerablemente distintas a las del hardware.

Productos de software

- **Productos genéricos:** producidos por una organización para ser comercializados
- **Productos hechos a medida:** sistemas que son desarrollados bajo pedido a un desarrollador específico.
    - La mayor parte del gasto del software se realiza en los genéricos pero existe un mayor esfuerzo en los hechos a medida.

Características de los Productos Software

1. **Mantenibles:** el software debe evolucionar y seguir cumpliendo con sus especificaciones.
2. **Confiabilidad:** no debe causar daños físicos o económicos en el caso de fallos.
3. **Eficiencia:** no debe desperdiciar los recursos del sistema**.**
4. **Utilización adecuada:** debe contar con una interfaz de usuario y documentación adecuada.

Importancia de las características de software

La importancia relativa de las características depende del tipo de producto y del ambiente en el que será utilizado. Los costos tienden a crecer exponencialmente si son requeridos altos niveles de alguna característica.

Clasificación de Productos de software

- **Por su Estructura:**
  - Funcionales
  - Orientados a objetos
  - Orientados a listas
  - Orientados a componentes
- **Por su función:**
  - Programas o sistemas de usuario
  - Herramientas de software
  - Librerías
  - Sistemas de uso genérico: compiladores, SO.
  - Base de Datos
  - Sistemas basados en web
- **Por su plataforma de cómputo:**
  - Sistemas embebidos
  - Cómputo distribuido
  - Tiempo real
  - Basados en chips
  - Cómputo ubicuos

Costos del software

A menudo dominan al costo del sistema. Cuesta más mantener el software que desarrollarlo y la ingeniería de software concierne un desarrollo efectivo en cuanto a costos de software.

Proceso de software

Conjunto estructurado de actividades requeridas para desarrollar un sistema de software. Las actividades del proceso son: **especificación** (que debe hacer el software y cuales son sus especificaciones de desarrollo), **desarrollo** (producción del sistema de software), **validación** (verificar que el software hace lo que el cliente pide), **evolución** (cambiar o adaptar el software a las demandas).

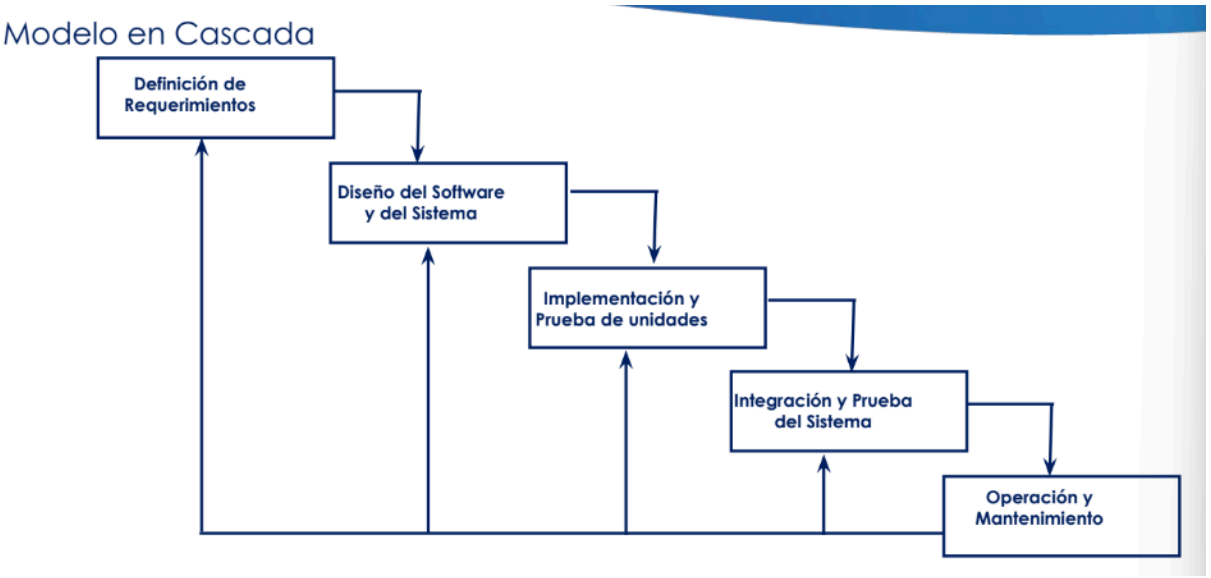
- **Proceso genérico:**
  - **Especificación:** establecer los requerimientos
  - **Diseño:** producir un modelo del sistema
  - **Codificación:** construir el sistema
  - **Prueba:** verificar que el sistema cimpla con las especificaciones
  - **Instalación:** entregar el sistema al usuario y asegurar su operacionalidad.
  - **Mantenimiento:** reparar fallos en el sistema cuando sean descubiertos.

- **Características del proceso:**
  - Entendible
  - Visible
  - Soportable
  - Aceptable
  - Confiable
  - Robusto
  - Mantenible
  - Rapidez

- **Problemas en el proceso**
    - Especificaciones incompletas o anómalas
    - No existe distinción precisa entre la especificación, diseño y la codificación
    - El sistema solo puede ser probado hasta que se haya producido
    - No se puede reemplazar el software durante el mantenimiento.

Modelos de Desarrollo de Software
Representación formal o simplificada del proceso de software.

- **Modelos genéricos:**
    - **Cascada:** separar en distintas fases de especificación y desarrollo
    - **Desarrollo evolutivo:** especificación y desarrollo intercalados
    - **Prototipado:** un modelo sirve de prototipo para la construcción del sistema final
    - **Transformación formal:** un modelo matemático del sistema se transforma formalmente en la implementación
    - **Reutilización:** el sistema es ensamblado a partir de componentes existentes.



La dificultad del modelo reside en la dificultad de hacer cambios entre etapas

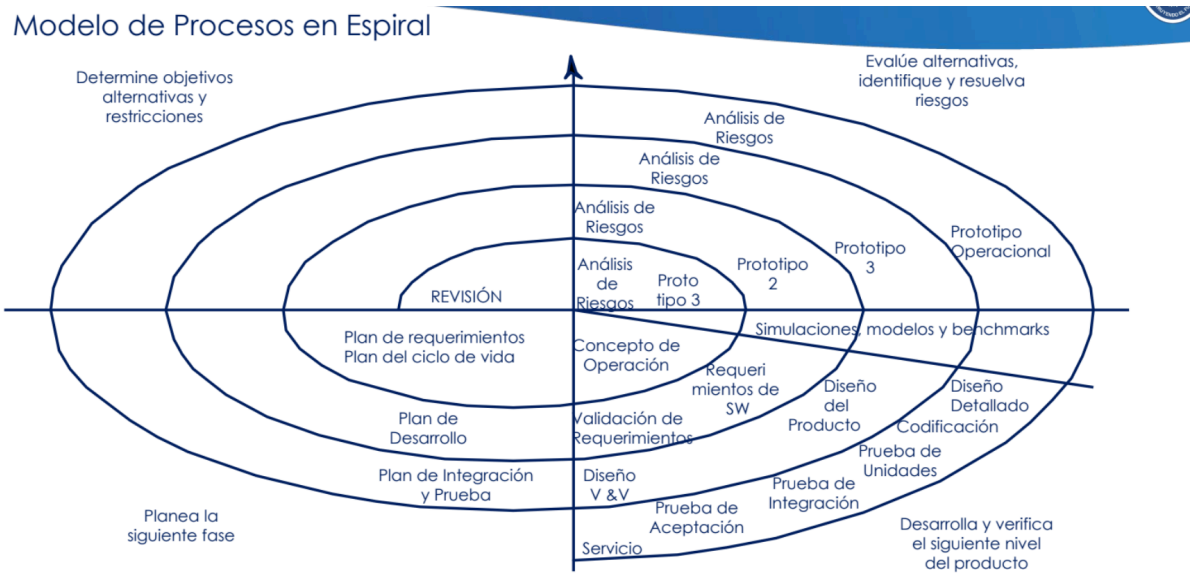
Problemas: poca visibilidad en el proceso, los sistemas están pobremente especificados, se requieren habilidades especiales
Aplicabilidad: para sistemas interactivos pequeños o medianos, para módulos de sistemas grandes y para sistemas de corta vida

Problemas y Riesgos con los modelos

- Cascada:
    - Alto riesgo en sistemas nuevos por problemas en las especificaciones y diseño
    - Bajo riesgo para desarrollos bien comprendidos utilizando tecnología conocida
- Prototipado:
    - Bajo riesgo para nuevas aplicaciones debido a que las especificaciones y el diseño se llevan cabo paso a paso.
    - Alto riesgo debido a la falta de visibilidad.
- Evolutivo:
    - Alto riesgo debido a la necesidad de tecnología avanzada y habilidades del grupo desarrollador

Modelo de procesos Híbridos

- Los sistemas grandes están compuestos usualmente de varios subsistemas
- No es necesario utilizar el mismo modelo de proceso para todos los subsistemas
- El prototipado es recomendado cuando existen especificaciones de alto riesgo
- El modelo de cascada es utilizado en desarrollos bien comprendidos



Fases:

- **Planteamiento de objetivos:** se identifican los objetivos específicos para cada fase del proyecto.
- **Identificación y reducción de riesgos:** los riesgos clave se identifican y analizan y la información sirve para minimizar los riesgos
- **Desarrollo y validación:** se elige un modelo apropiado para la siguiente fase del desarrollo
- **Planeación:** se revisa el proyecto y se trazan planes para la siguiente ronda del espiral.

Ventajas

- Centra su atención en la reutilización de componentes y eliminación de errores en información
- Los objetivos de calidad son el primer objetivo
- Integra desarrollo con mantenimiento
- Provee un marco de desarrollo de hardware/software.

Problemas

- El desarrollo contractual específica el modelo del proceso y los resultados a entregar por adelantado
- Requiere de experiencia en la identificación de riesgos
- Requiere refinamiento para uso generalizado.

¿Qué modelo utilizar?

Cascada: para sistemas bien comprendidos. La fase de análisis de riesgos es relativamente fácil

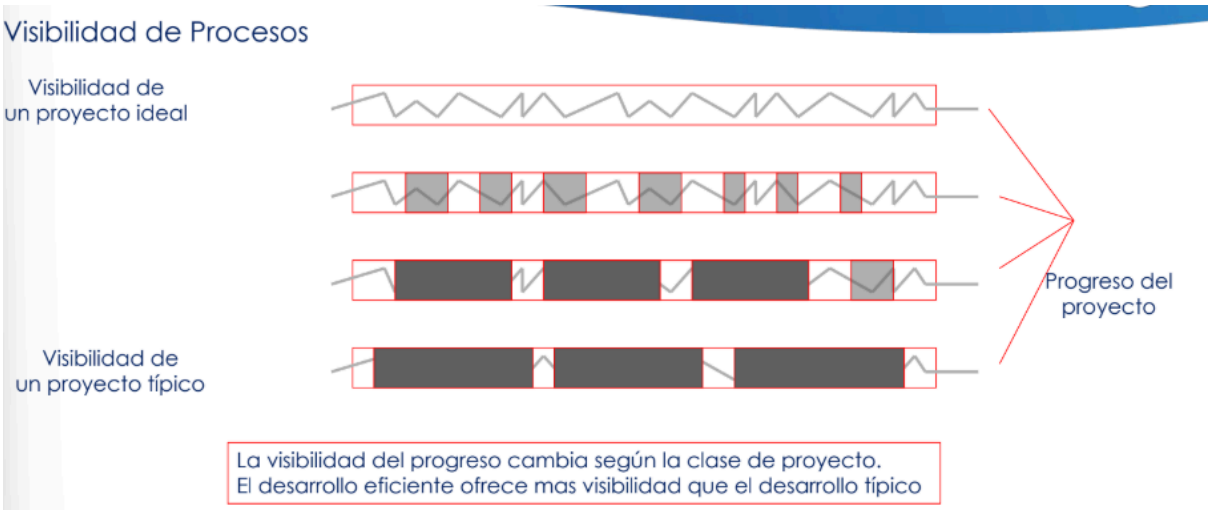
Modelos Formales: con requerimientos estables y sistemas de seguridad críticos

Prototipado: con especificaciones incompletas

Híbridos: para utilizarse en distintos módulos del sistema.

Visibilidad de Procesos

Los sistemas de software son intangibles por lo que los administradores necesitan documentación para identificar el progreso en el desarrollo.



Modelo de Proceso	Visibilidad del Proceso
Modelo de Cascada	Buena visibilidad, cada actividad produce un documento o resultado
Desarrollo Evolutivo	Visibilidad pobre, muy caro al producir documentos en cada iteración.
Modelos Formales	Buena visibilidad, en cada fase deben producirse documentos.
Desarrollo orientado a la reutilización	Visibilidad moderada. Importante contar con documentación de componentes reutilizables.
Modelo de Espiral	Buena visibilidad, cada segmento y cada anillo del espiral debe producir un documento.

Retos de la Ingeniería de Software

- Mantener y tratar con sistemas legados. Tratar con una mayor diversidad de sistemas con mayores demandas de cómputo y menores tiempos de entrega
- Sistemas legados: sistemas antiguos que deben ser mantenidos y mejorados.
- Heterogeneidad: sistemas que incluyen a distintas plataformas de software y hardware
- Entrega: existe una presión incremental por una entrega a tiempo de los productos de software
- Formalidad: existe una gran demanda de que exista formalidad en el proceso de desarrollo de software

Demanda de Ingenieros de Software

Como las aplicaciones de software se han tornado más complejas ha surgido una demanda de ingenieros de software que tengan el conocimiento y la experiencia para desarrollar

sistemas de software de alta calidad. El poder de cómputo se ha incrementado y el costo del hardware y las comunicaciones han caído, haciendo más económica la implementación de software. Los dispositivos mecánicos en automóviles, aviones y plantas se están reemplazando por componentes de software.

Unidad 2-Sistemas Sociotecnicos

Sistema

Es una colección de componentes interrelacionados que trabajan conjuntamente para alcanzar objetivos comunes. Puede incluir software, hardware mecánico, eléctrico y electrónico que puede ser manejado por la gente.

Categorías de Sistema

- **Sistemas técnicos basados en computadoras:** incluyen hardware y software pero en donde los operadores y los procesos operacionales no son normalmente considerados como parte del sistema y este no es consciente de su finalidad.
- **Sistemas socio-técnicos:** sistemas que incluyen sistemas técnicos pero también procesos operacionales y gente que utiliza e interactúa con el sistema técnico. Son gobernados por políticas y reglas organizacionales.

Características de Sistemas Socio-tecnicos

- **Propiedades emergentes:** las propiedades del sistema como un todo, que dependen de los componentes del sistema y sus relaciones. Pueden ser medidas una vez que los componentes han sido integrados dentro de un sistema. Entre estas están: **el volumen, la confiabilidad, la protección, reparabilidad y usabilidad.**
    - T**ipos:**
        - Funcionales: aparecen cuando todas las partes de un sistema trabajan juntas para alcanzar el mismo objetivo.
        - No Funcionales: se relacionan con el comportamiento del sistema en su ambiente operacional.
- **No determinísticos:** no siempre producen la misma salida con la misma entrada porque el comportamiento del sistema es dependiente parcialmente de los operadores humanos.
- **Relaciones complejas con objetivos organizacionales:** el grado en que el sistema apoya los objetivos organizacionales no solo depende del sistema en sí mismo.

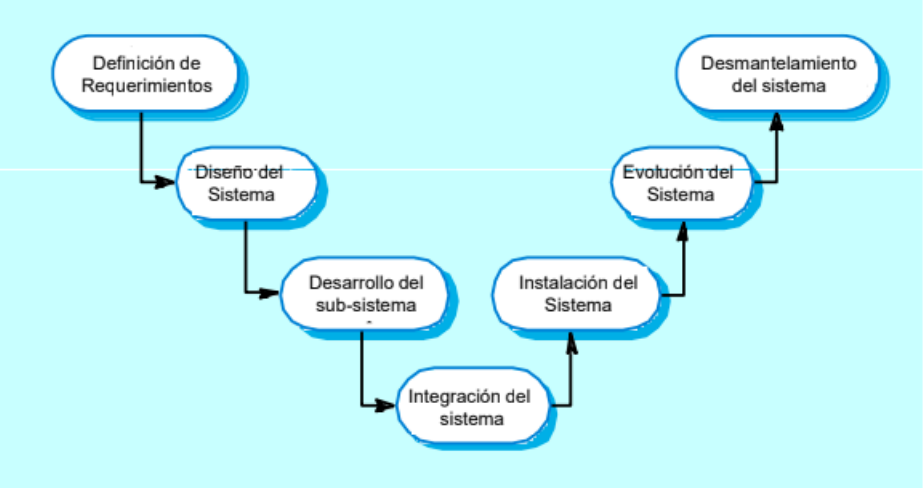
Fiabilidad del sistema: debido a las interdependencias de los componentes, las fallas pueden propagarse por todo el sistema. Las fallas del sistema a menudo suceden debido a interrelaciones entre componentes que no han sido previstas y probablemente sea imposible predecir todas las posibles relaciones entre los componentes.

Ingeniería de Sistemas

Es especificar, diseñar, implementar, validar y mantener sistemas socio-técnicos. Se consideran los servicios que provee el sistema, limitaciones en su construcción y operación y las maneras en las cuales es utilizado.

- **Proceso:** usualmente sigue el modelo cascada, por lo que se necesita desarrollo paralelo de diferentes partes del sistema. Pequeño panorama para la iteración entre las fases debido a cambios muy costosos en el hardware y el software puede tener que compensar por los problemas de hardware e inevitablemente se involucran ingenieros de diferentes disciplinas lo que puede llevar a muchos malentendidos.

El Proceso de Ingeniería de Sistemas



Definición de Requerimientos del sistema

Tipos:

1. Requerimientos funcionales abstractos
2. Propiedades del sistema
3. Características que no debe mostrar el sistema

También se deberían definir los objetivos generales de la organización para el sistema. Se debe definir por qué está siendo diseñado un sistema para un ambiente en particular.

Problemas:

- Usualmente se diseñan sistemas complejos para resolver problemas confusos
- Se debe anticipar desarrollos de hardware durante el tiempo que el sistema funcione.
- Es difícil definir requerimientos no funcionales sin saber la estructura de los componentes del sistema

Proceso del Diseño del Sistema

1. Dividir requerimientos
2. Identificar subsistemas: identificar un grupo de subsistemas los cuales pueden colectivamente cumplir los requerimientos del sistema
3. Asignar requerimientos a los subsistemas
4. Especificar las funcionalidades del subsistema
5. Definir las interfaces del subsistema

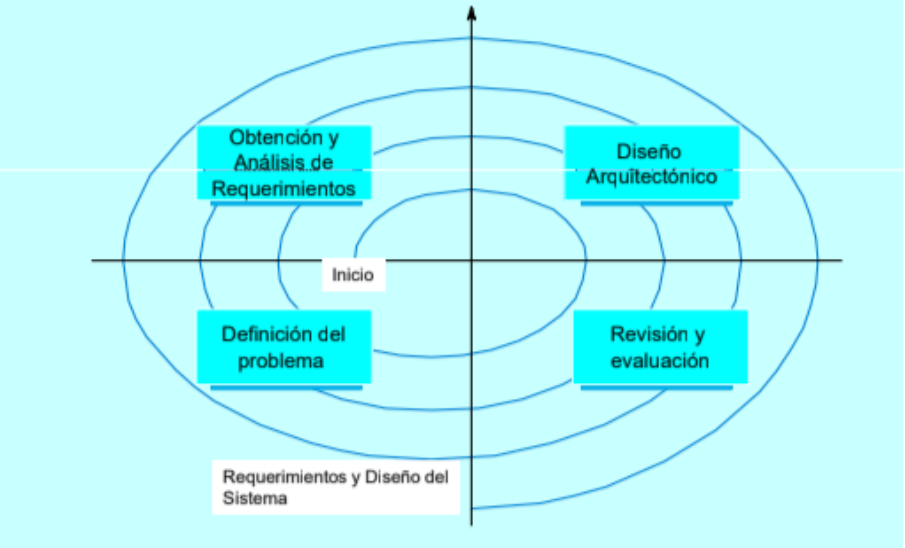
Problemas

- Los requerimientos relativos a los componentes hardware, software y personas pueden involucrar muchas negociaciones.
- Problemas de diseño complicados son considerados listos para ser resuelto con software.

Requerimientos y diseño

- Los requerimientos de ingeniería y diseño de sistemas están relacionados. El diseño inicial puede ser necesario para estructurar los requerimientos.

Modelo en Espiral de Requerimientos/Diseño



Modelado de sistemas

- **Un modelado arquitectónico presenta una vista abstracta de los subsistemas que forman un sistema. Pueden incluir grandes flujos de información entre subsistemas.**

Integración de Sistemas

Es el proceso de poner el hardware, software y las personas juntos para hacer un sistema. El proceso debería ser incremental de modo que los subsistemas sean integrados uno por vez y generalmente se encuentran problemas de interfases entre subsistemas y entregas no coordinadas de componentes.

Instalación de Sistemas

Luego de terminado, el sistema será instalado en el ambiente del cliente.
Problemas: lo supuesto del ambiente puede ser incorrecto, resistencia humana, coexistencia de sistemas, problemas físicos de instalación.

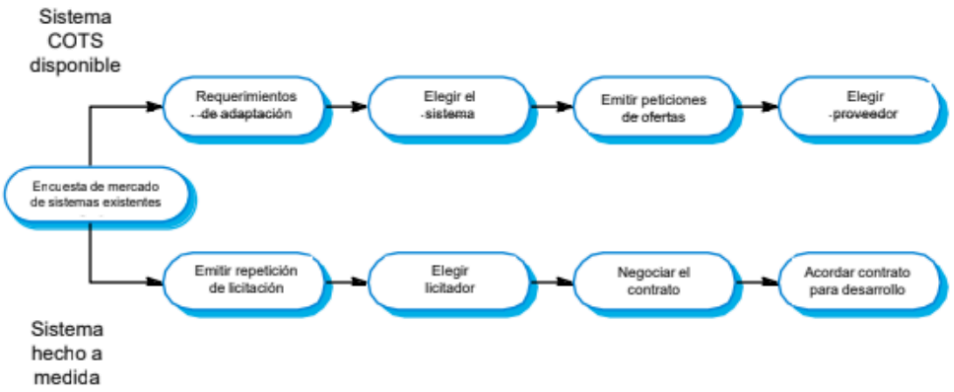
Evolución del Sistema

Los sistemas grandes tienen un tiempo de vida útil extenso y tienen que evolucionar para cumplir requerimientos nuevos pero esta evolución es inherentemente costosa. A veces se deben mantener sistemas existentes llamados sistemas heredados.

Procesos Organizacionales
Los procesos de ingeniería de sistemas superponen e interactúan con los procesos de adquisiciones organizacionales. Los procesos operacionales son aquellos involucrados en utilizar el sistema para el uso que fue definido. Para nuevos sistemas, éstos tienen que ser definidos como parte del diseño del sistema.
Adquisición del sistema

- Usualmente, antes de la adquisición se requiere alguna especificación y un diseño arquitectónico.
    - Se requiere una especificación para incluir en el contrato de desarrollo.
    - La especificación puede permitir comprar un sistema comercial COTS. Casi siempre es más económico que desarrollar un sistema de cero.
- Los sistemas grandes y complejos usualmente consisten en una mezcla de COTS y componentes especialmente diseñados. Los procesos de adquisición para estos diferentes tipos de componentes son a menudo distintos.

Proceso de Adquisición del Sistema



Contratistas y Subcontratistas
La adquisición de grandes sistemas de hardware y software usualmente se basa en un contratista principal y los subcontratos son realizados con otros proveedores para partes del sistema. El cliente trata con el contratista principal y no negocia directamente con los subcontratistas.
Sistemas heredados
Son los sistemas sociotécnicos que han sido desarrollados utilizando tecnología antigua u obsoleta y son cruciales para la operación de un negocio y a menudo es demasiado riesgoso descartar estos sistemas. Limitan nuevos procesos de negocio y consumen una gran porción del presupuesto de la compañía
Componentes

- **Hardware:** obsoleto de mainframe
- **Software:** softwares de soporte de proveedores que ya no están en el negocio
- **Software de aplicación:** escrito en lenguajes de programación obsoletos
- **Datos de aplicación:** incompletos o inconsistentes
- **Procesos de negocio:** limitados por la funcionalidad y la estructura del software
- **Políticas y reglas del negocio:** pueden estar implícitas en el software del sistema.

Unidad 3-Pruebas de Software

Pruebas de Software

Probar es el proceso de ejercitar un sistema con el objetivo específico de encontrar errores antes de entregarlo al usuario final. Surge debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta y debido a que el desarrollo de los sistemas de software implica actividades en las que las posibilidades que aparezca el fallo humano son enormes.

Características

1. Operabilidad: si el sistema opera limpiamente.
2. Observabilidad: los resultados de cada prueba son observados en todo momento.
3. Controlabilidad: cuanto mejor se pueda controlar el software más se puede automatizar y optimizar
4. Capacidad de descomposición: las pruebas pueden ser realizadas de manera modular
5. Simplicidad: reducir arquitecturas complejas y lógicas para simplificar las pruebas

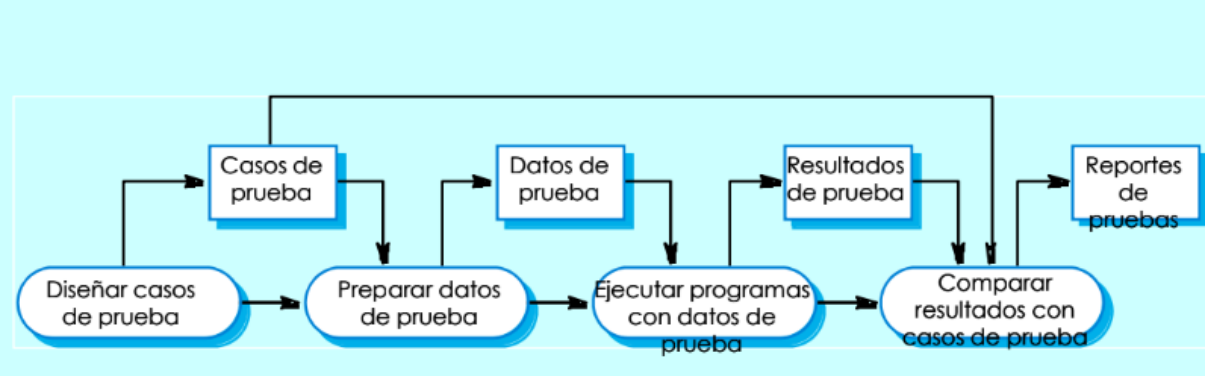
Las pruebas muestran errores, conformidad de requerimientos, rendimiento e indicadores de calidad.

Proceso

- **Pruebas de componentes:** pruebas de programas de componentes individuales, usualmente es la responsabilidad del desarrollador del componente y las pruebas se derivan de su experiencia.
- **Pruebas de sistema:** pruebas de grupos de componentes integrados para crear un sistema o subsistema y la responsabilidad es de un equipo de pruebas independiente y se basan en una especificación del sistema.

Objetivos

- **Pruebas de validación:** demostrar al desarrollador y al cliente de sistema que el software cumpla sus requerimientos. Una prueba exitosa muestra que el sistema opera de la manera en que debe.
- **Pruebas de defectos:** descubrir fallas o defectos en el software donde su comportamiento es incorrecto o no se ajusta a las especificaciones. Una prueba exitosa es aquella que hace que el sistema se comporte incorrectamente y de esa manera expone un defecto existente.



Políticas de pruebas: las políticas de pruebas definen el enfoque a ser utilizado en la selección de los sistemas a probar.

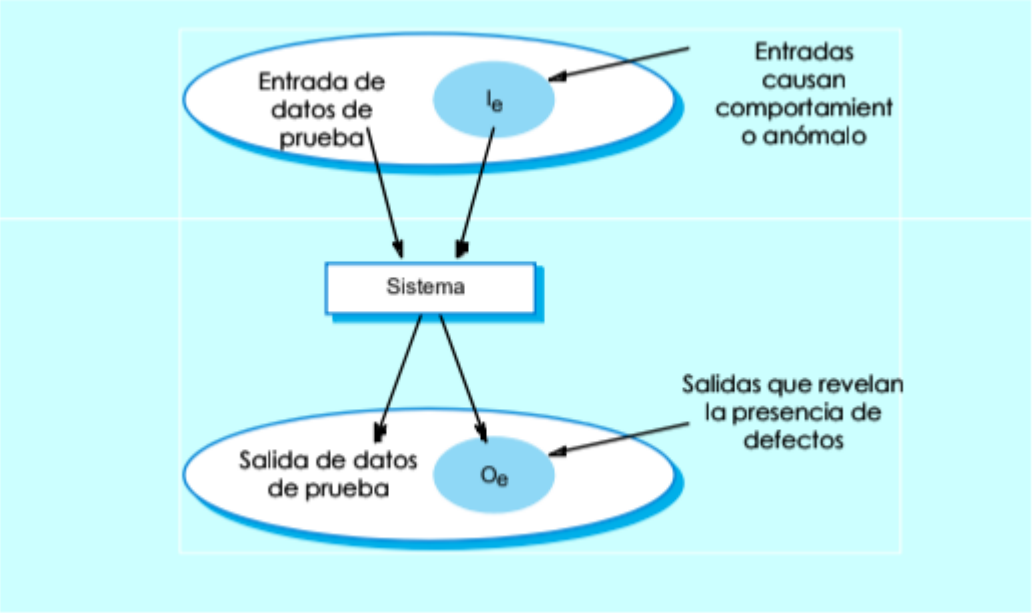
Pruebas de Sistema
Involucra la integración de componentes para crear un sistema o subsistema. Puede significar un incremento a ser entregado al cliente y tiene dos fases:

- **Pruebas de integración (caja blanca):** el equipo de pruebas tiene acceso al código fuente y el sistema se prueba cómo están integrados los componentes.
- **Pruebas de entrega (caja negra):** el equipo de pruebas chequea el sistema completo a ser entregado como una caja negra.

Pruebas de Integración
Involucra construir un sistema desde sus componentes y probarlos para problemas que surgen de la interacción de los componentes.

- Integración descendente: desarrollar el esqueleto del sistema y agregar los componentes.
- Integración ascendente: integrar los componentes de infraestructura y luego agregar los componentes funcionales.

Pruebas de Entrega
Es el proceso de probar una versión de un sistema que será entregada a los clientes. El objetivo primario es aumentar la confianza entre el cliente acerca de que el sistema cumple con los requerimientos. Las pruebas de versión son generalmente cajas negras o pruebas funcionales.



Guia para realizar pruebas: son puntos a considerar para el equipo de pruebas y ayudan a elegir las pruebas que revelan defectos en el sistema.

- Elegir entradas que obliguen al sistema a generar todos los mensajes de error
- Diseñar entradas que causan sobrecarga del sistema
- Repetir la misma prueba varias veces
- Obligar a que se generen salidas inválidas.

Casos de uso: pueden ser una base para derivar las pruebas del sistema y ayudan a identificar operaciones a ser probadas y a diseñar los casos de prueba requeridos.

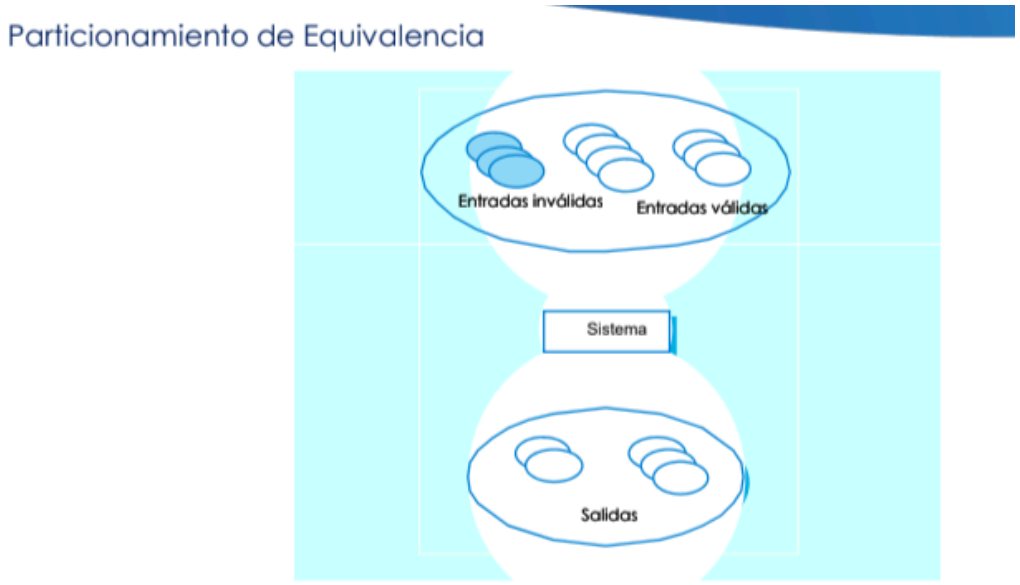
Pruebas de rendimiento
Parte de las pruebas de entrega pueden involucrar las pruebas de propiedades emergentes de un sistema como el rendimiento y la fiabilidad. Las pruebas de rendimiento incluyen la planificación de una serie de pruebas donde la carga se incrementa establemente hasta que el rendimiento del sistema se torna inaceptable.
Pruebas de Estrés
Ejercitan al sistema más allá de su máxima carga diseñada. Sobrecargar al sistema a menudo causa que los defectos salen a la luz. Los sistemas no deberían fallar catastróficamente y son relevantes para sistemas distribuidos que pueden mostrar una degradación cuando la red se satura.
Pruebas de componentes
Es el proceso de probar componentes individuales por separado, con métodos, clases de objeto o componentes realizados con interfaces definidas.
Pruebas de clases de objeto
Una completa cobertura de pruebas de una clase involucra: probar todas las operaciones asociadas con un objeto, asignación y consulta de todos los atributos de un objeto y ejercitar el objeto en todos sus estados posibles.
Pruebas de Estación Climática
Se requiere definir casos de prueba para una clase y sus métodos. Usando un modelo de estado se identifican las secuencias de transición de estados a ser probadas y la secuencia de eventos que causan estas transiciones.
Pruebas de Interfaces
Los objetivos son detectar fallas debido a errores de interfaces o supuestos inválidos acerca de estas.

- TIpos de interfaces:
    - De parámetros: datos pasados de un procedimiento a otro
    - De memoria compartida: el bloqueo de memoria es compartido entre procedimientos o funciones
    - Procedurales: el subsistema encapsula un grupo de procedimientos a ser llamados por otros subsistemas.
    - De traspaso de mensajes: el subsistema solicita servicios de otro subsistema.
- Errores
    - Mal uso
    - Mal entendidas
    - Errores de tiempo

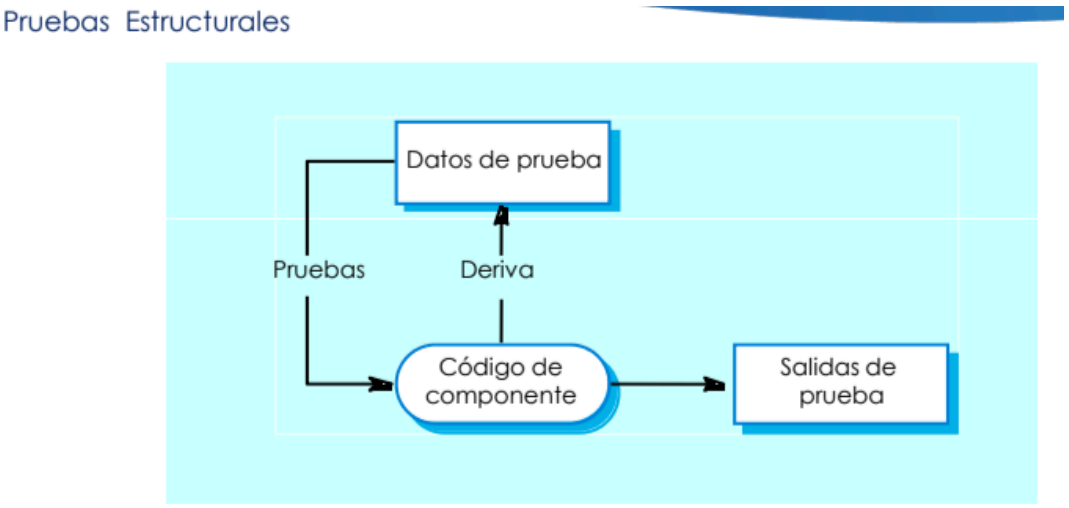
Diseños de casos de Prueba
Involucra diseñar los casos de prueba utilizados para probar el sistema. El objetivo es crear un grupo de pruebas que sean efectivas en validación y pruebas de defectos.

- Enfoques
    - **Pruebas basadas en requerimientos:** Un principio general de la ingeniería de requerimientos es que éstos deben poder ser probados. Las pruebas basadas en requerimientos son una técnica de pruebas de validación donde se considera cada requerimiento y se deriva un grupo de pruebas para ese requerimiento.

- **De partición:** los datos de entrada y salida de resultados se pueden agrupar en diferentes clases donde todos los miembros de una clase están relacionados.



- **Estructurales:** pruebas de caja blanca, derivan de los casos de prueba de acuerdo a la estructura del programa y el conocimiento del mismo es utilizado para identificar casos de prueba adicionales. El objetivo es ejercitar todos los comandos del programa.



Pruebas de camino
El objetivo es asegurar que el grupo de casos de prueba es tal que cada camino en el programa se ejecuta por lo menos una vez. El punto de partida es un gráfico de flujo de programa que muestra nodos que representan decisiones del programa y arcos que representan el flujo de control

Unidad 4-Estimación de Proyectos Software

Estimación de software

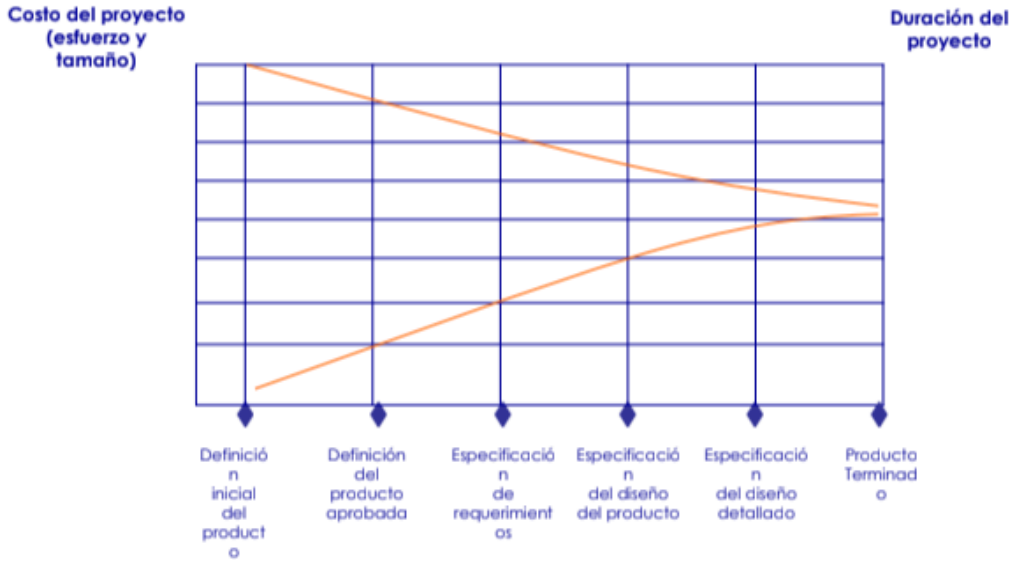
Es una predicción de cuánto tiempo durará o costará su desarrollo y mantenimiento. Si la estimación es de tiempo, el esfuerzo puede expresarse en horas-hombre, si se trata de estimación de costo, se puede expresar de manera monetaria. El reto de elaborar estimaciones es realizar predicciones realistas, basándose en información incompleta e incierta.

¿Para qué se utilizan las estimaciones?

- Desarrollar planes de proyectos
- Elaborar planificaciones de iteración de desarrollo de software
- Elaborar presupuestos
- Realizar analisis de inversion
- Fijación de precios de un software para un cliente empresarial
- Análisis para determinar el precio en software dirigido al consumidor
- Para planificar una estrategia para participar en subastas de contratos

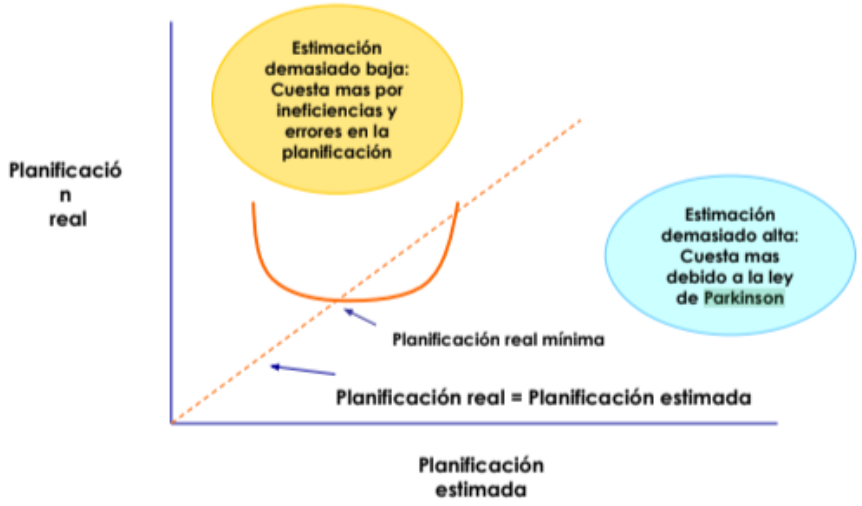
Según Jones, 1994: algunas estimaciones son realizadas cuidadosamente y otras a ojo. La mayoría de los proyectos rebasan los límites de sus planificaciones estimadas entre 25 y 100 % pero muy pocas organizaciones han logrado realizar una predicción de la planificación con una precisión de un 10% e incluso se ha llegado a un caso de 5%.

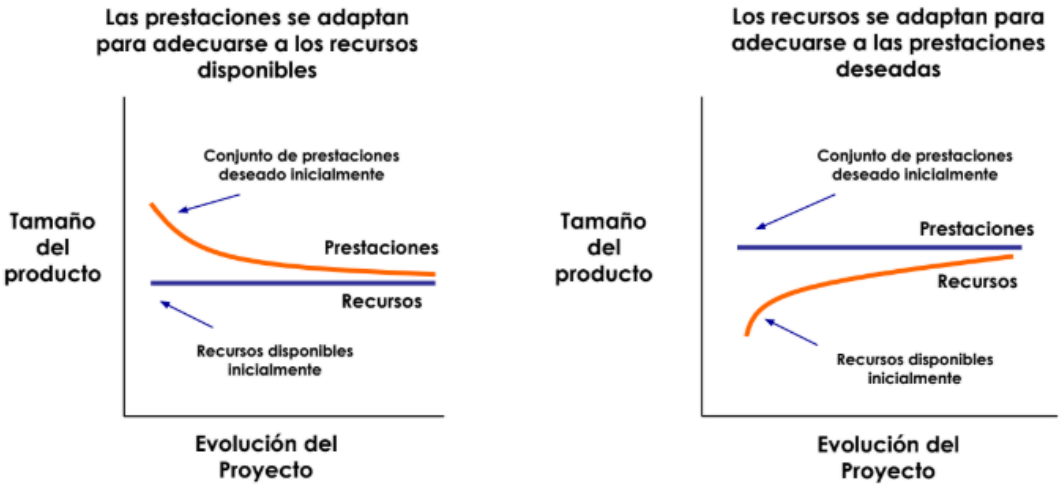
Introducción al proceso de estimación



Los investigadores han descubierto que las estimaciones de proyectos entran dentro de precisiones previsibles en varios estados del proyecto. La mayoría de los clientes de software desean inicialmente más de lo que se pueden permitir y tienen que adaptar sus ideas sobre el producto a los recursos con los que están dispuestos o comprometerse.

- Si los clientes desean la planificación más corta posible, no deberían presionar para reducir la estimación o proporcionar unas estimaciones precisas equivocadas.
- La planificación real más corta resulta de la programación planificada más precisa. Si la estimación es demasiado baja, la planificación ineficiente conducirá al costo real del proyecto. Si la estimación es demasiado alta, según la ley de Parkinson conducirá al costo real del proyecto.
    - **Ley de Parkinson:** el trabajo se reparte para cubrir el tiempo disponible. Es decir, nos dice cómo ajustar nuestro ritmo de trabajo a la cantidad de tiempo que tenemos disponible.





Proceso para una buena planificación:

1. **Estimar el tamaño del producto (número de líneas de código o puntos de función):** una estimación efectiva necesita estimar primero el tamaño del software para poder construirlo. Es el más difícil intelectualmente y se suele saltarlo.
2. **Estimar el esfuerzo (persona-mes):** es fácil realizar la estimación del esfuerzo si se tiene una estimación exacta del tamaño y los datos previos de la organización.
3. **Estimar la planificación (en meses):** una vez estimados tamaño y esfuerzo, estimar la planificación resulta casi trivial pero obtener la aceptación de una estimación de la planificación realista puede ser la más difícil del proyecto.

Estimar el tamaño de un proyecto

- **Utilizar un enfoque algorítmico:** como los puntos de función o casos de uso para estimar el tamaño del programa a partir de las prestaciones
- **Utilizar un software de estimación:** para estimar el tamaño a partir de la descripción de las prestaciones del programa.
- **Si se ha trabajado con proyectos similares:** se conoce el tamaño por lo que se puede estimar cada una de las partes principales del nuevo sistema como un porcentaje del tamaño de una parte similar del sistema antiguo.

Puntos de Función

Es una medida sintética del tamaño del programa que se suele utilizar en los primeros estados del proyecto. Son más fáciles de determinar a partir de la especificación de los requerimientos que las líneas de código y proporcionan una medida más exacta sobre el tamaño del programa.

Se basa en el número y la complejidad de cada uno de los siguientes elementos:

1. **Entradas:** pantallas, formularios, cuadros de diálogo, controles o mensajes, a través de los cuales el usuario final o cualquier otro programa puede añadir, borrar o cambiar datos del programa
2. **Salidas:** pantallas, informes, gráficos o mensajes que el programa genera para el usuario final o cualquier otro programa.
3. **Consultas:** combinaciones de entrada/salida, en las que cada entrada genera una salida simple e inmediata. La diferencia entre salida y consulta es que las consultas recuperan datos directamente de una base de datos y muestran solo el formato elemental mientras que las salidas pueden procesar, combinar o resumir datos complejos y presentar muchos formatos.
4. **Archivos lógicos internos:** son los principales grupos lógicos de datos de usuarios finales o información de control que están completamente controlados por el programa.
5. **Archivos de interfaz externos:** archivos controlados por otros programas con los que el programa va a interactuar.

Estimación del primer Orden de Jones

Si se tiene la suma total de todos los puntos de función, se puede realizar a partir de ellos un cálculo aproximado de la planificación, utilizando un método de Capers Jones. Para utilizarlo hay que tomar el total de puntos de función y elevarlo a la potencia apropiada.

Exponentes para calcular planificaciones a partir de puntos de función

Clase de Software	Mejor caso	Media	Peor caso
Sistemas	0,43	0,45	0,48
Gestión	0,41	0,43	0,46
"Pret-a-porter"	0,39	0,42	0,45

Cocomo II

Es un modelo de estimación de costo utilizable en la evaluación del esfuerzo estimado para un proyecto.

- **Etapas:**
  - Etapa 1: soporta la estimación de los esfuerzos de prototipado o desarrollo de aplicaciones
  - Etapa 2: soporta la estimación a comienzos de la etapa de diseño de un proyecto, cuando se conoce poco acerca del costo del proyecto
  - Etapa 3: soporta la estimación en la etapa post arquitectural del proyecto.
- **Entradas**
  - La entrada principal es el tamaño del programa, puntos de función o puntos de casos de uso. Se deben evaluar 17 atributos adicionales incluidos en las siguientes 4 categorías:
    - **Atributos del Producto:** describen el ambiente en el cual opera el programa, como requerimientos de confiabilidad, tamaño de la BD, concordancia entre la documentación y las necesidades del ciclo de vida, reutilización y complejidad del programa.
    - **Atributos de la Plataforma:** se refieren a las limitaciones que afectan el esfuerzo del desarrollo debido al hardware y al SO que se están utilizando en la ejecución del proyecto como almacenamiento principal, tiempo de ejecución y volatilidad.
    - **Atributos del Personal:** describen la habilidad del personal asignado al proyecto. Capacidad del analista, experiencia en las aplicaciones y en los lenguajes de programación, plataforma.
    - **Atributos del Proyecto:** se refieren a las restricciones y condiciones bajo las cuales opera el desarrollo del proyecto.
  - Estos factores (o multiplicadores de esfuerzo) se multiplican y de esta forma están incorporados en las fórmulas de estimación del tiempo y esfuerzo. El valor numérico del i-ésimo factor de ajuste es llamado EMi y su producto es llamado factor de ajuste o EAF. El esfuerzo total PMtotal es el producto del esfuerzo nominal y el EAF.

- **Procesamiento Cocomo II**
    - Una evaluación nominal de los meses-hombre basada solo en el tamaño se realiza al programa considerado. Y luego se multiplica la evaluación de todos los atributos para obtener el esfuerzo en meses-hombre requerido por el proyecto. Los desafíos principales de Cocomo II son determinar el tamaño del proyecto y asignar valores apropiados a los atributos.
- **Salidas de Cocomo II**
    - El nivel de esfuerzo en meses-hombre para el proyecto siendo estimado
    - El tiempo del proyecto en meses. El esfuerzo puede ser convertido a valor monetario si se conoce el costo por mes-hombre
    - Distribución de fases: muestra un desglose del esfuerzo del software y el tiempo requerido a las fases del ciclo de desarrollo.
- **Calibración**
    - Es esencial para el uso correcto de modelos de costo de software. El usuario puede calibrar los EAF y las ecuaciones de esfuerzo/tiempo del proyecto actual
- **Aplicabilidad de la métrica**
    - El método de estimación está basado en dos modelos: uno aplicable al comienzo de los proyectos (Diseño preliminar) y otro aplicable luego del establecimiento de la arquitectura del sistema (Post Arquitectura).
        - **Diseño Preliminar:** contempla la exploración de las arquitecturas alternativas del sistema y los conceptos de operación. No se sabe lo suficiente del proyecto como para hacer una estimación final. El modelo propone la utilización de los puntos de función como medida del tamaño y conjunto de 7 factores que afectan al esfuerzo del proyecto.
        - **Post Arquitectura:** contempla el desarrollo y el mantenimiento de un producto software. Es más precisa si se ha desarrollado una arquitectura del sistema, la cual haya sido validada y establecida como base para la evolución del producto. El modelo propone la utilización de las líneas de código fuente y/o puntos de función como medidores del tamaño , modificadores para indicar el grado de reutilización del software, 17 estimadores de costo y un conjunto de 5 factores que afectan de manera exponencial en el esfuerzo del proyecto.

    - **Métrica:** en los dos modelos, la estimación del esfuerzo se realiza tomando como base la siguiente ecuación:

      $$PM_{nominal} = A \times (Size)^{B}$$

      PMnominal: es el esfuerzo nominal requerido en meses-hombre
      Size: tamaño estimado del software en miles de líneas de código o en puntos de función sin ajustar.
        - **Valoración del factor escalar B:** se calcula a partir de la sumatoria de los aportes de distintas variables escalares, que indican las

características que el proyecto presenta como complejidad y entorno de desarrollo:

- PREC: variable de precedencia u orden secuencial del desarrollo
- FLEX: flexibilidad del desarrollo
- RSEL: fortaleza de la arquitectura, métodos de estimación y reducción de riesgos
- TEAM: cohesión y madurez del equipo de trabajo
- PMAT: proceso de madurez del software

Luego de la ponderación de éstas variables, el Factor escalar se calcula mediante la siguiente ecuación:

$$B = 0.91 + 0.01 \times \Sigma(W_i)$$

Wi: factor escalar

- **Ajuste del esfuerzo nominal:** los multiplicadores de esfuerzo se cuantifican con una escala de Muy Bajo a Extra alto y cada multiplicador tiene asociado un valor a cada nivel de escala.

$$PM_{ajustado} = PM_{nominal} \times \Pi(EMi)$$

Estimación del esfuerzo – COCOMO II

Ecuación de tiempo (schedule equation)

Predice el número de meses requeridos para completar el proyecto. La duración del proyecto está basada en el esfuerzo calculado por la ecuación del esfuerzo:

$$Duración = 3.67 * (PM_{ajustado})^{[0.28 + 0.2 * (B-0.91)]}$$

Unidad 5-Planificación de Proyectos de Software

Causas de retrasos de los proyectos

- Fecha límite de entrega poco realista
- Cambio en los requisitos del cliente que no se reflejan en los cambios de la planificación temporal
- Subestimación honesta de la cantidad de esfuerzo y número de recursos que serán necesarios para hacer el trabajo.
- Riesgos predecibles y no predecibles no considerados al comienzo del proyecto.
- Falta de comunicación entre el equipo que causa retrasos

Planificación excesivamente optimista

- En 1967, Gene Bylinsky informó que todos los problemas de programación significativos se vuelven urgentes
- En 1970, Fred Brooks apuntó que hay más proyectos software que han salido mal por falta de tiempo que por todas las otras causas combinadas.
- Scott Castello observó que la presión de la fecha límite es el principal enemigo de los proyectos de software
- Capers Jones informa que la presión excesiva en la planificación es el más común de todos los problemas serios de ingeniería de software.
- **Trabajar bajo una presión extrema de la fecha límite se ha convertido en una tradición. El 15 % de los proyectos establecen sus planificaciones antes que los requerimientos, y no los establecen con tiempo suficiente.**

Causas fundamentales

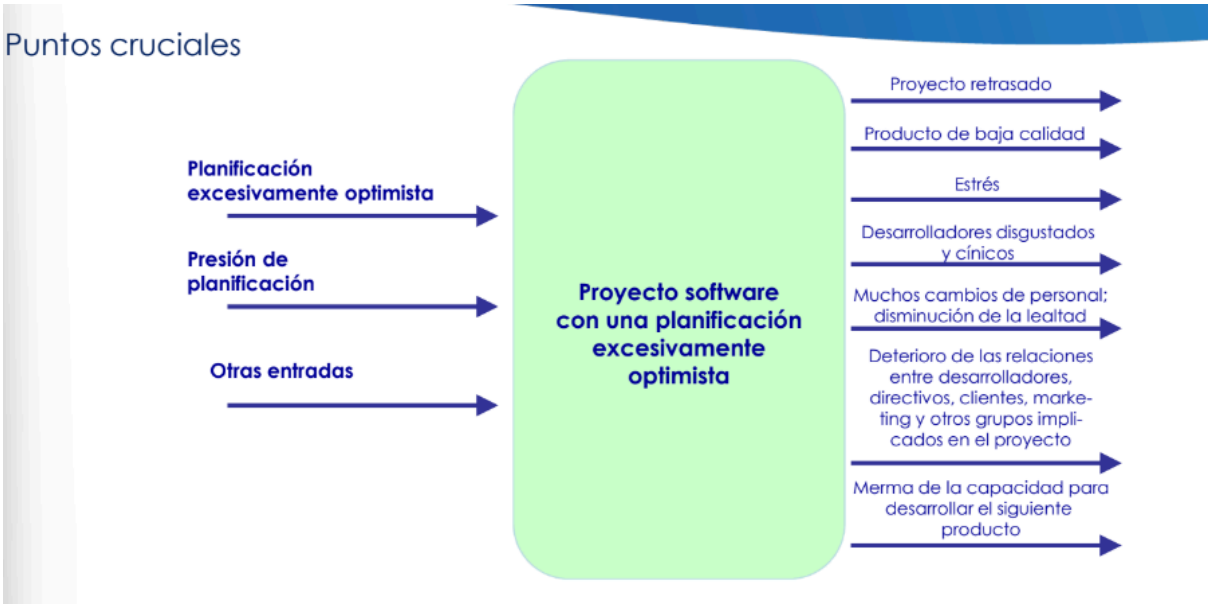
- Hay un plazo límite externo e inmutable.
- Los responsables se niegan a aceptar un rango de estimación y hacen los planes basándose en una estimación puntual del mejor caso.
- Los responsables y desarrolladores subestiman deliberadamente el proyecto porque desean un incentivo o les gusta trabajar bajo presión.
- El proyecto es subestimado deliberadamente por la directiva o el responsable del área comercial para proponer una oferta insuperable.

Presión excesiva en la planificación

Ocurre aproximadamente en un 75% de los proyectos de mediana envergadura y cerca del 100% en todos los proyectos grandes.

- **Calidad:** se ha determinado que alrededor de un 40% de todos los errores de software son causa de la tensión.
- **Azar:** los directivos y desarrolladores sienten la necesidad de apostar al azar en vez de correr riesgos calculados.
- **Motivación:** un poco de presión en la planificación resultante de una planificación ligeramente optimista pero factible puede motivar.
- **Creatividad:** muchos aspectos del desarrollo requieren creatividad. La motivación externa excesiva reduce la motivación interna.
- **Agotamiento:** si se abusa de las horas extras en un proyecto los desarrolladores se verán afectados en el próximo proyecto.
- **Cambio de personal:** las planificaciones excesivamente optimistas y la presión resultante en la planificación tienden a causar un cambio voluntario excesivamente alto de personal.
- **Relación entre desarrolladores y directivos:** la presión en la planificación aumenta las diferencias entre los desarrolladores y directivos.

En Quality Software Management, Gerald Weinberg sugiere pensar en los proyectos del software como en sistemas (Weinberg). Cada sistema tiene unas entradas y genera unas salidas



Principios básicos

1. **Compartimentación**
2. **Interdependencia**
3. **Asignación de tiempo**
4. **Validación de Esfuerzo**
5. **Responsabilidades definidas**
6. **Resultados definidos**
7. **Hitos definidos**

- El objetivo del gestor del proyecto es definir todas las tareas del proyecto, construir una red que describa sus interdependencias, identificar las tareas que son críticas dentro de la red y después hacerles un seguimiento para asegurarse de que el retraso se reconoce «de inmediato».

- Para conseguirlo, el gestor debe tener una planificación temporal que se haya definido con un grado de resolución que le permita supervisar el progreso y controlar el proyecto.
- La planificación temporal para proyectos informáticos puede verse desde dos perspectivas bastante diferentes. En la primera se ha establecido ya (irrevocablemente) una fecha final de entrega del proyecto. La organización está limitada a distribuir el esfuerzo dentro del marco de tiempo previsto.
- El segundo punto de vista de la planificación temporal asume que se han estudiado unos límites cronológicos aproximados pero que la fecha final será establecida por los gestores del proyecto

Considere cuatro ingenieros del software, cada uno capaz de producir 5000 LDC/año cuando trabajan en proyectos individuales. Cuando se pone a estos cuatro ingenieros en un proyecto de equipo, son posibles seis vías de comunicación potenciales. Cada vía de comunicación requiere un tiempo que podría de otra manera emplearse en desarrollar software. Asumiremos que la productividad del equipo (medida en LDC) se verá reducida en 250 LDC/año por cada vía, debido al gasto indirecto asociado con la comunicación. Por tanto, la productividad del equipo es 20.000 -(250 x 6) = 18.500 LDC/año, un 7,5 por ciento menos de lo que podríamos esperar.

- El proyecto de un año en el que está trabajando el equipo anterior se retrasa ya dos meses de la fecha de entrega se agregan dos personas adicionales al equipo. El número de vías de comunicación se dispara a 15.
- La productividad de la nueva plantilla es la equivalente a 840 x 2 = 1.680 LDC para los dos meses restantes antes de la entrega.
- La productividad del equipo es ahora

20.000 + 1.680- (250 x 15) = 17.930 LDC/año.

Tipos de proyectos

- **Desarrollo del concepto:** se inician para explorar algún nuevo concepto de negocios o aplicación de alguna tecnología
- **Desarrollo de una nueva aplicación:** se aceptan como consecuencia del encargo de un cliente específico
- **De mejora de aplicaciones:** que ocurren cuando un software existente sufre grandes modificaciones de su funcionamiento rendimiento o interfaces que son observables por el usuario final.
- **Mantenimiento de aplicaciones:** que corrigen, adaptan o amplían un software existente.
- **Proyectos de reingeniería:** que se lleva a cabo con la intención de reconstruir un sistema existente (heredado) en su totalidad o parte.

Grados de Rigor

Es una función de muchas características del proyecto. Para un proyecto de un tipo en particular, el grado de rigor con el que se aplica el proceso del software puede variar significativamente.

- **Casual:** se aplican todas las actividades estructurales del proceso pero solo se requiere un conjunto de tareas minimo

- **Estructurado:** se aplicará la estructura del proceso a este proyecto. Se aplicaran actividades estructurales y tareas relativas apropiadas para el tipo de proyecto.
- **Estricto:** se aplicará el proceso completo para este proyecto con un grado de disciplina tal que garantice una alta calidad.

Red de tareas

Es una representación gráfica del flujo de tareas de un proyecto. Se emplea a veces como el mecanismo a través del cual se introduce la secuencia de tareas y las dependencias en una herramienta de programación automática de la planificación temporal de un proyecto.
La tecnica de evaluacion y revision del programa PERT y el método del camino crítico CPM son dos métodos de la planificación temporal de un proyecto y ambas son dirigidas por informaciones anteriores sobre la planificación del proyecto:

- Estimaciones de esfuerzo
- Descomposición de la función del producto
- Selección del modelo de proceso adecuado y del conjunto de tareas
- Descomposición de tareas.

Gráficos de tiempo o Gantt

Cuando se crea una planificación temporal de un proyecto de software el planificador empieza un conjunto de tareas. Si se emplean herramientas automáticas, la descomposición del trabajo es introducida como una red de tareas o esquema de tareas. El esfuerzo, duración y fecha de inicio son las entradas de cada tarea. Además se asignan tareas a individuos específicos y se genera un gráfico de tiempo.

Plan de Proyecto

Se produce a la culminación de las tareas de planificación. Proporciona información básica de costos y planificación temporal que será empleada a lo largo del proceso de software. Es un documento relativamente breve dirigido a una audiencia diversa y debe:

1. Comunicar el ámbito y recursos a los gestores, personal técnico y al cliente.
2. Definir los riesgos y sugerir técnicas de aversión al riesgo.
3. Definir los costos y planificación temporal para la revisión de la gestión.
4. Proporcionar un enfoque general del desarrollo para todo el personal relacionado con el proyecto
5. Describir cómo se garantizará la calidad y se gestionan los cambios.

Es importante señalar que el Plan del Proyecto no es un documento estático.
Esto es, el equipo del proyecto consulta el plan repetidamente actualizando riesgos, estimaciones, planificaciones e información relacionada a la vez que el proyecto avanza y es más conocido.

Unidad 6-Desarrollo Rápido del Software

Objetivos de la Ingeniería de Software

- Maximizar calidad
- Maximizar productividad
- Minimizar riesgos

Dificultades en Producción de Software

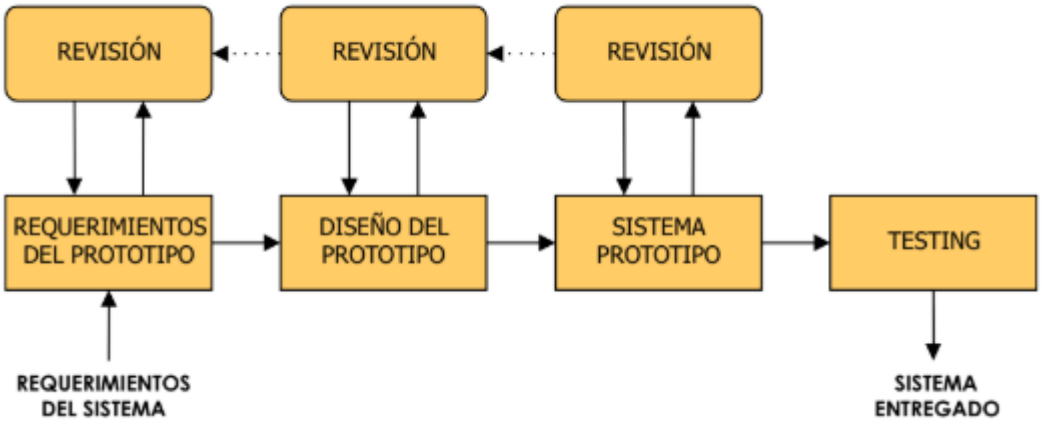
- Esencia
- Complejidad
- Conformidad
- Necesidad de cambios
- Invisibilidad

Proceso de Software

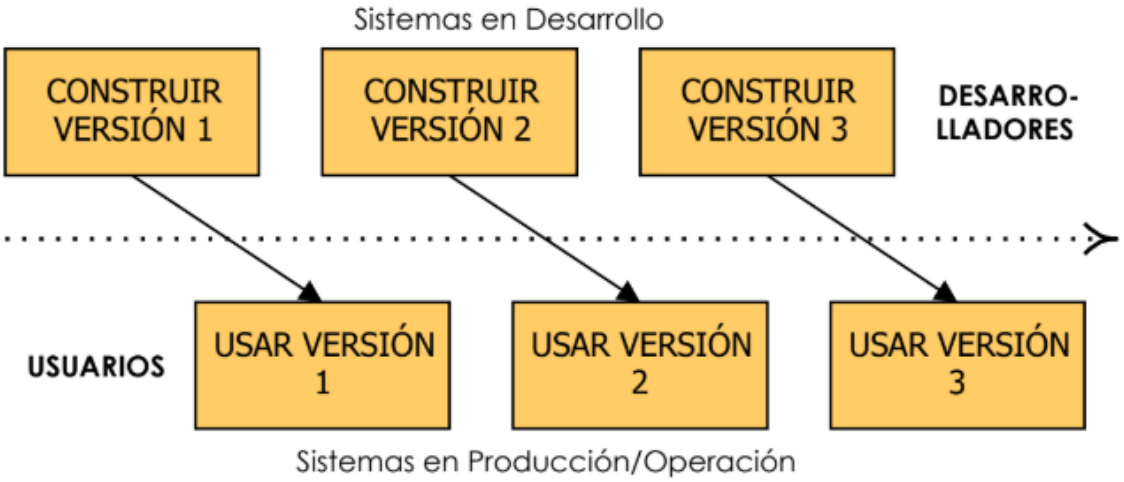
Conjunto relacionado de actividades y tareas implicadas en el desarrollo y evolución de un sistema software.

Modelos tradicionales

1. **Modelo en Cascada:** es el más antiguo, debe completarse un estado antes de comenzar el siguiente. Es útil para que el desarrollador visualice lo que va a hacer y su principal problema es que no refleja la realidad.
2. **Modelo de Prototipación:** permite la construcción del sistema. Usuario y desarrollador tienen una visión común y se reduce el riesgo y la incertidumbre del desarrollo.

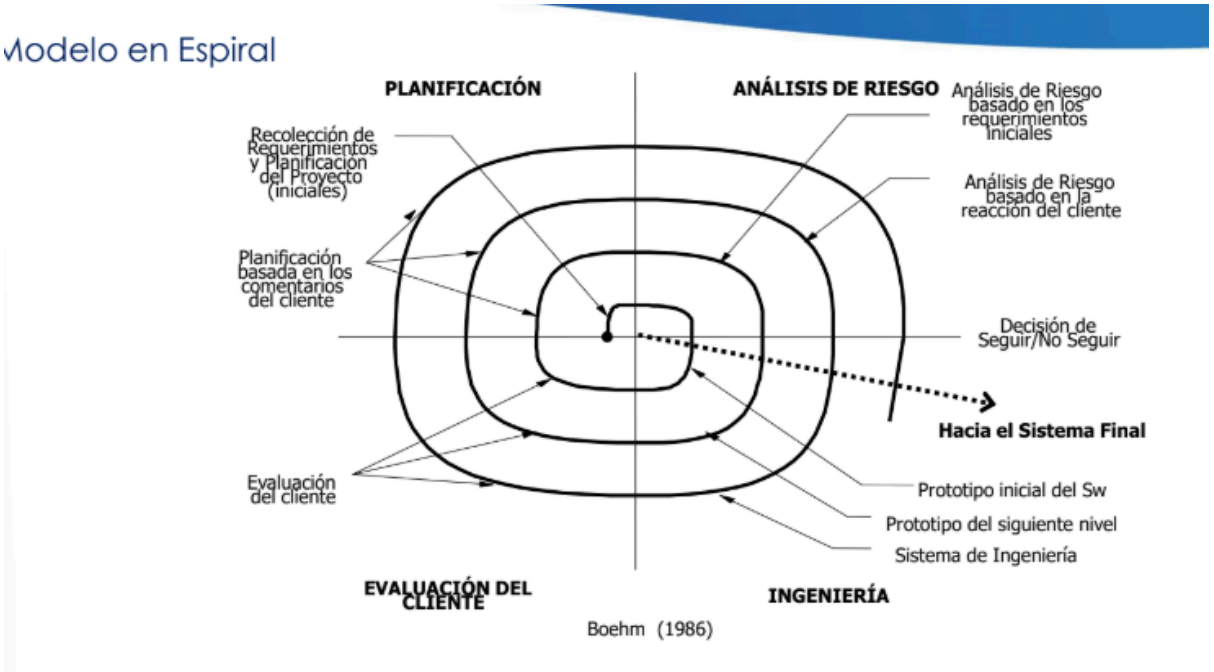


3. **Modelo de Desarrollo en Fases:** una forma de reducir el retardo. El sistema se diseña de manera a que pueda ser entregado por partes y así el usuario tiene algo de funcionalidad mientras se desarrolla el resto y hay dos sistemas funcionando en paralelo: el de producción, utilizado por el cliente y el de desarrollo, la siguiente versión que reemplazara a la actual de producción



4. **Modelo en Espiral:** se combinan las actividades de desarrollo con análisis de riesgo. El modelo es de tipo iterativo y tiene las siguientes fases:
    a. Planificación
    b. Análisis de Riesgo
    c. Ingeniería
    d. Evaluación

En cada iteración se evalúan las diferentes alternativas y se elige una y los gestores del proyecto intentan eliminar o minimizar el riesgo.



RUP-Rational Unified Process

Corresponde a un framework que puede ser usado para describir procesos de desarrollo específicos. Cada ciclo de vida del software abarca 4 fases en el orden siguiente:

- Planificación
- Elaboración
- Construcción
- Transición

La esencia es la iteración y cada una de estas resulta en un entregable preferentemente ejecutable

Problemas de los modelos tradicionales

- Incumplimiento de los plazos de entrega
- Reducción de las funcionalidades previstas inicialmente
- Difícil adaptación a los cambios
- Excesiva documentación

Métodos Ágiles-Historia

En Febrero de 2001 nace el término ágil aplicado al desarrollo de software. El objetivo de los expertos fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

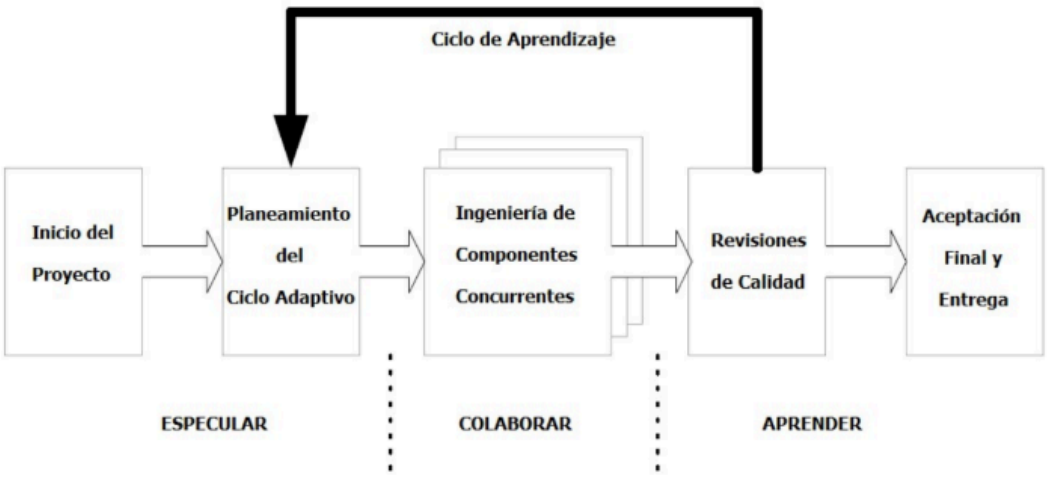
Concepto: son un conjunto de estrategias del desarrollo que promueven prácticas adaptativas, en lugar de predictivas. Promueven la adaptabilidad a los cambios por sobre el planteamiento estricto.

Según el manifiesto ágil, se valora:

- **Al individuo y a las interacciones del equipo de desarrollo sobre el proceso y las herramientas:** es mucho más importante construir un buen equipo que construir el entorno.
- **Desarrollar software que funciona más que conseguir una buena documentación:** deben ser breves y centrarse en lo fundamental.
- **La colaboración con el cliente más que la negociación de un contrato:** interacción constante entre el cliente y el equipo de desarrollo.
- **Responder a los cambios más que seguir estrictamente un plan:** la planificación no debe ser estricta sino flexible y abierta

Tipos de Metodologías ágiles

- Adaptive Software Development
    - Desarrollado por J. Highsmith en el 2000
    - La idea fundamental es **balancearse en el borde del caos**
    - Es decir, buscar un equilibrio entre crear un ambiente que favorezca la creatividad e innovación y administrar lo que se está haciendo.



Uso actual

- Casi no es utilizado porque no se especifican en detalle las pautas a seguir para su uso
- Debe complementarse con otra metodología mejor definida
- Provee conceptos sobre la filosofía mismas de las Metodologías Ágiles con ideas sobre desarrollo de sistemas complejos

Programación Extrema XP

Desarrollado por K. Beck en 1999. Es un conjunto de prácticas ya conocidas pero combinadas de manera innovadora para lograr una nueva metodología. Se la conoce como extrema por el hecho de llevar cada una de sus prácticas a niveles extremos de aplicación.

- **Planificación incremental:** los requerimientos son grabados en tarjetas de Historia y las historias a ser incluidas en una versión son determinadas por el tiempo disponible y su prioridad relativa.
- **Entregas pequeñas:** el mínimo grupo de funcionalidades útiles que proveen valor al negocio son las que se desarrollan primero
- **Diseño simple:** solo el necesario es llevado a cabo para cubrir los requerimientos
- **Desarrollo previamente probado:** un plan de trabajo de unidades de pruebas automatizadas es utilizado para escribir pruebas para una nueva funcionalidad antes de que sea implementada.
- **Refactorización:** se espera que todos los desarrolladores inspeccionen continuamente tan pronto como mejoras al código sean identificadas.
- **Programación en pares:** los desarrolladores trabajan en pares chequeando el trabajo del otro
- **Propiedad colectiva:** los pares trabajan en todas las áreas de manera que no existan expertos en determinadas áreas, todos son propietarios del código
- **Integración continua:** tan pronto una tarea se completa, es integrado dentro del sistema

- **Ritmo sostenible:** grandes cantidades de horas extras no son aceptables ya que reduce la calidad y la productividad.
- **Cliente presente:** un representante del usuario final debería estar disponible a tiempo completo para el equipo de XP.

Utilización

- El 30% del mercado agil utiliza XP ya que tiene bien definidas sus prácticas

SCRUM

Es una manera de trabajar en equipo donde el resultado se produce de forma incremental y el resultado es un entregable. Se establecen periodos cortos de trabajo en los que se sigue un mismo patrón, denominados sprint y son la base del método.

Actividades

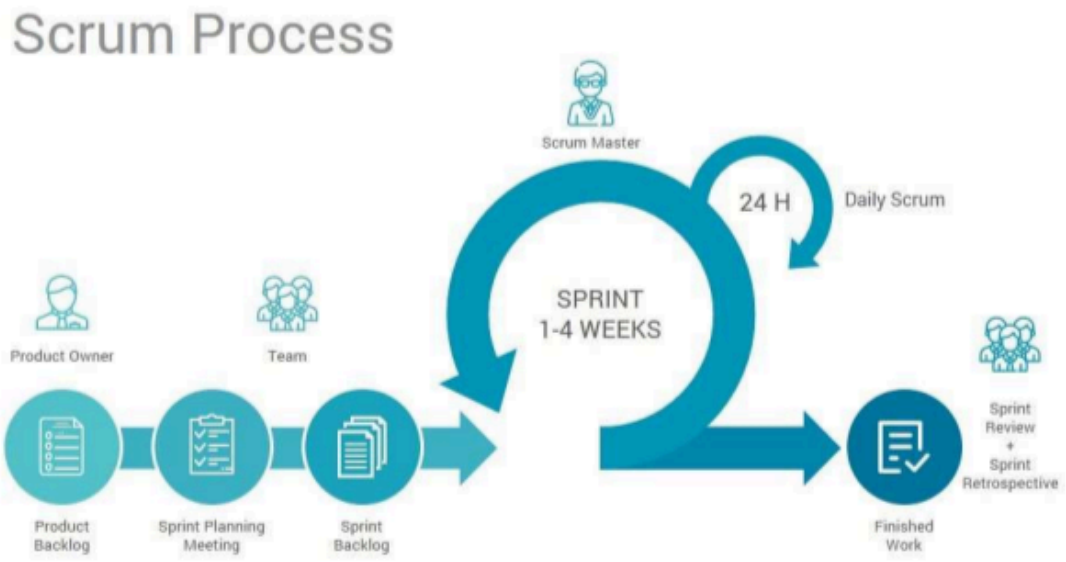
- **Sprint**
- **Sprint Planning**
- **Daily**
- **Sprint review**
- **Sprint retrospective.**

Herramientas

- Product backlog
- Sprint BackLog
- Burn Down

Roles

- Product Owner
- Scrum Master
- Team



Uso actual: es uno de los más utilizados y a nivel de investigación se están realizando esfuerzos para integrar XP y SCRUM argumentando que SCRUM provee un marco de gestión del proyecto sobre las prácticas bien definidas de XP.

Ventajas de las Metodologías Ágiles

- Iteraciones en ciclos cortos que permiten correcciones y verificaciones más rápidamente
- Límites de tiempo para cada ciclo de dos a ocho semanas
- Adaptable al surgimiento de nuevos riesgos
- Orientación hacia las personas
- Estilo de trabajo en equipo.

Desventajas

- Puede llevar a caos en el proceso de desarrollo ya que no sigue un plan estricto
- Difícil de mantener el interés de los clientes
- La definición de contratos puede ser un problema
- Mantener la simplicidad significa un trabajo extra

Cuadro comparativo

Ágiles	Tradicionales
Basadas en heurísticas provenientes de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Preparados para cambios durante el proyecto	Resistencia a cambios
Impuestas internamente	Impuestas externamente
Proceso menos controlado	Proceso mucho más controlado
No existe contrato tradicional o es flexible	Contrato prefijado
Cliente parte del equipo de desarrollo	Cliente interactúa con el equipo mediante reuniones
<10 integrantes en el mismo sitio	Grandes grupos y distribuidos
Menos énfasis en la arquitectura de software	La arquitectura es esencial