INGENIERÌA DE SOFTWARE.

Javier Gerard

- “Notas importantes”:

Antes de comenzar con el desarrollo del presente manual de “Ingeniería de software” considero pertinente decir que esta obra no es totalmente de mi autoría, sino que configura una recopilación de resúmenes del libro “Ingeniería del Software” de Ian Somerville (Séptima Edición), de las diapositivas presentadas en clase por el profesor Raúl González, y de conclusiones y visiones personales.
Desde ya, a disfrutarlo.

Índice:

6. "Proceso (relevamiento)"

.7. Proceso (otros medios de especificación de requerimientos)

.8. Plan de proyecto

.9. Aseguramiento y estándares

.10. Pruebas del software

.1. Anteproyecto:

Al hablar de lo que comprende un anteproyecto haremos alusión a los siguientes puntos:

.1.1. ¿Qué es el anteproyecto?:

El anteproyecto es la descripción del destinatario (cliente) y del sistema que se pretende desarrollar; por medio de este se pretende destacar las ventajas que se obtendrán por intermedio de este sistema, al igual que el/los programador/es que desarrollarán el proyecto en cuestión.

.1.2. ¿Por qué y para qué?:

Para poder responder esta interrogante es que recurrimos al concepto de "análisis estratégico"; este consiste en determinar los valores, visión, misión, fuerzas porter (empresas que configuran nuestra competencia en el mercado, futuras empresas en el rubro en el mercado, productos similares que puedan cumplir con la presente necesidad, capacidad de negociación de los clientes, y la capacidad de negociación de los proveedores), cadena de valor, y el análisis FODA (herramienta consistente en la elección de buenas estrategias a tomar, para las decisiones adoptadas por el gerente o jefe administrativo).
Con todo esto lo que queremos decir es que el análisis estratégico se realiza desde el ámbito interno de la empresa hasta el sistema a desarrollarse.

.1.3. ¿Qué hará?:

Para dar una respuesta a esta interrogante es que surge el concepto de "Proyecto"; lo que se procurará es:
* Definición clara acerca del proyecto: lo que se pretende aquí es entender el problema, ya que: "Un buen entendimiento equivale a una buena explicación del problema a resolver" – Raúl González.
* Estudio de factibilidad: aquí lo que se busca es realizar un amplio análisis acerca de la viabilidad económica y financiera del proyecto a realizar.
* Análisis de riesgos: aquí se hace referencia a la realización de un análisis acerca de las posibles amenazas y riesgos que pueden ocurrir a lo largo del proyecto, y por ende, las consecuencias y daños que estos podrían causar (ya que si se prevén estos casos, se reducirían los riesgos a una posible pérdida del proyecto).
* Especificación de los requerimientos: lo que se pretende es lograr explicar qué es lo que el destinatario (cliente) pretende con las diferentes funcionalidades del sistema, con esto me refiero a las diferentes funcionalidades que deberá tener el sistema, y el para qué debo desarrollar

estas funcionalidades (ya que estas deberán cumplir o satisfacer las diferentes necesidades – problemas que los clientes plantean). Teniendo en claro los problemas a resolver, la funcionalidad logrará desempeñarse de forma eficaz.
* Casos de uso (generales y/o principales): también en esta etapa se deben establecer y/o presentar al cliente los casos de uso generales y/o principales, de esta forma ya antes de comenzar con el desarrollo del proyecto, ya podremos tener validadas por el cliente las principales funcionalidades del sistema a desarrollar.

.1.4. ¿Cómo y con qué?:

Para responder esta interrogante es que recurrimos al concepto del "Plan de proyecto". Este consiste en la elección de las herramientas (tecnologías, lenguaje, motor de base de datos, etc.), metodologías (scrum, en cascada, evolutiva, en espiral, etc.) a utilizar.
Dentro del plan de proyecto es también que encontraremos dos nuevos conceptos, que son el de: SQA (control de calidad de software: esto consiste en las metodologías empleadas para cumplir con la necesidad de efectividad del sistema, cumplir con las normativos ISO, etc.), y SCM (gestión de configuración de software: aquí se hace referencia a la configuración, versionado, roles, etc.).
Considero importante decir que dentro de este punto también incluiremos a la capacitación del personal en cuestión.

.1.5. ¿Cuándo?:

Para responder a esto punto es que recurriremos al concepto de "planificación". Este punto dependerá mucho de la metodología empleada. En nuestro caso particular se nos pide que utilicemos la metodología de "Scrum", por tal motivo es que recurriremos a la planificación, y por ende a herramientas como el MS Project, por medio del cual podremos desarrollar los gráficos de Gantt correspondientes, que nos permitan realizar una clara planificación de nuestro proyecto.

.2. “Ingeniería de Software”:

.2.1. Introducción:

En este punto trataremos los primeros conceptos que pueden surgir de las palabras “Ingeniería de software”, al igual que dar un primer punto de vista sobre esta disciplina.

.2.2. Concepto:

Es importante decir que no existe un único concepto acerca de la “Ingeniería de Software”, en base a esto daré conceptos que se han dado a lo largo de la historia por diferentes autores:

- Bauer, 1972: es la disciplina que trata los principios y métodos de la ingeniería a fin de obtener sw de modo rentable que sea fiable y trabaje en máquinas reales.
- Bohem, 1976: es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadoras y la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de sw o producción de sw.
- Zelkowits, 1978: la IdeS es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas de software.
-IEEE, 1993: es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de sw.

.2.3. ¿Qué es la ingeniería de software?:

La ingeniería de software es la disciplina de la ingeniería que comprende todos los aspectos de la producción de software, estos aspectos abarcan desde sus etapas iniciales (de la especificación del sistema), su desarrollo, e inclusive el mantenimiento de ella luego de que la misma es terminada.
En esta definición encontraremos dos factores claves:

a) Disciplina de la ingeniería: los ingenieros son los encargados de hacer que las cosas funcionen, para ello aplicarán teorías, métodos, herramientas, etc. También tendremos que tener en cuenta que para elaborar una aplicación existen riesgo financieros y organizacionales, razón por la cual los ingenieros serán los responsables de las decisiones a tomar, tomando en especial consideración estas restricciones.

b) Todos los aspectos de producción de software: de la definición también se desprende que la ingeniería no sólo comprende los procesos técnicos del software, sino también la gestión de proyectos del software, el desarrollo de herramientas, de métodos, etc.

.2.4. ¿Qué es software?:

Muchas personas asocian al término software con "programa de computadora", sin embargo el software no es sólo un programa de computadora, sino que también es el conjunto de documentos asociados a él, y la configuración de datos que se necesitan para hacer que el programa funcione de forma correcta.
Los ingenieros de software se concentran en el desarrollo de productos de software, productos que se venderán a los clientes, en base a esto diremos que existen dos tipos de productos de software:
a) Productos genéricos: estos son productos aislados, los cuales son vendidos en mercados abiertos a cualquier cliente, como por ejemplo: procesadores de texto, motores de base de datos, herramientas de gestión de proyectos, etc.
b) Productos personalizados (o hechos a medida): estos son sistemas operativos hechos a medida, en base a requisitos específicos de un cliente. Como por ejemplo: sistemas de control para instrumentos electrónicos, sistemas de control de tráfico aéreo, sistemas para un putero, etc.

La principal diferencia entre estos dos productos es que en los productos personalizados, es el cliente quién controla la especificación del producto; hecho sumamente contrario a lo que ocurre en los productos genéricos. Sin embargo debemos decir que la línea de separación entre estos dos tipos de productos, cada vez es más borrosa, y difícil de diferenciar.

.2.5. Cuál es la diferencia entre ingeniería del software e ingeniería de sistemas?:

La ingeniería de sistemas se refiere a todos los aspectos del desarrollo y de la evolución de sistemas complejos, sistemas en el cual el software desarrolla un papel fundamental. En base a esto diremos que la ingeniería de sistemas comprende: el desarrollo de hardware, políticas y procesos de diseño, e inclusive a la ingeniería de software.

.2.6. Mitos:

Al hablar de la ingeniería de software diremos que existen mitos tanto por el lado del cliente, el de los gerentes del proyecto, y el los desarrolladores; por esta razón que aquí haremos alusión al mito, y a lo que el mismo representa en la realidad.

- Mitos del cliente:

1) Mito: los detales de los requerimientos se definirán a lo largo del desarrollo.
Realidad: los requisitos deben establecerse antes.
2) Mito: el software flexible se adapta a continuos cambios de requerimientos.
Realidad: los costos no son los mismos en todas las fases.

- Mitos de los gerentes de proyectos:

1) Mito: existen estándares de trabajo definidos.
Realidad: estos no se adaptan o no se usan.
2) Mito: tenemos excelentes ambientes de desarrollo.
Realidad: la tecnología existente no resuelve los problemas.
3) Mito: se pueden incorporar desarrolladores sin impacto.
Realidad: la incorporación no es sencilla.

- Mitos de los desarrolladores:

1) Mito: el trabajo finaliza cuando se entrega el producto.
Realidad: el trabajo finaliza cuando se deja de usar el producto.
2) Mito: el único control de calidad es el testing.
Realidad: es necesaria la existencia de validaciones tempranas antes de llegar a la etapa del testing.
3) Mito: el único entregable es el sistema funcionando.
Realidad: es necesaria la documentación técnica y de usuarios del sistema.

.2.7. Conclusiones del capítulo:

Como conclusiones de este capítulo diremos que:

1) Los productos de sw son programas y la documentación asociada. Sus atributos principales son: mantenibilidad, confiabilidad, eficiencia y usabilidad.
2) Las actividades básicas relativas al desarrollo sw que forman el proceso del sw son: especificación, desarrollo, validación y evolución.
3) Los métodos son formas organizadas de producir sw (proceso, notación, reglas y lineamientos de diseño).

.3. “Proceso del software”:

.3.1. Concepto:

Un proceso del software es un conjunto coherente de actividades y resultados que producen un producto de software (desde cero o no). Estas actividades son llevadas a cabo por los ingenieros de software. Existen cuatro actividades fundamentales de procesos, que son comunes a todos los procesos del software:

a) Especificación del software: donde los clientes e ingenieros definen el software a producir, al igual que las restricciones sobre su operación.

b) Diseño e implementación del software: donde el software se diseña y se programa.

c) Validación del software: donde el software se valida para asegurar que es lo que el cliente quiere.

d) Evolución del software: donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

- Ventajas: considero importante decir que la adopción de un proceso de software traerá consigo los siguientes beneficios: dan lugar a la reducción de tiempo, con ellos se obtiene una mayor organización, se disminuyen los costos del proceso, se logra llegar a una estandarización, se logra una capacitación y comunicación, se hace del software algo predictible, se logra mejorar con la experiencia de cada proyecto, etc.

- Objetivo: con el proceso de software lo que se busca es el conseguir una secuencia de pasos a seguir para la resolución de los problemas en forma ordenada.

- Determinar si un proceso es pobre: existen diferentes factores para determinar si un proceso de software es pobre, como por ejemplo: costos innecesarios, retrabajo descontrolado, desentendimiento, equipos poco cohesionados, pérdida de motivación, etc.

.3.2. Características del proceso de software:

Las principales características del proceso del software son:

- Precisión del proceso: aquí se hace referencia a que los diferentes elementos del proceso son tratados con un importante nivel de detalle.

- Valor del proceso: con esto hacemos referencia al valor que agregan las actividades del proceso a los clientes.

- Alcance del proceso: esta característica determina las etapas del ciclo de vida del proyecto que cubre el proceso.

- Tamaño del proceso: este es el número de elementos que el proceso poseerá (actividades, estándares, productos, técnicas).

- Densidad del proceso: con esto hacemos referencia al grado de detalle que se poseerá sobre los elementos del proceso.
- Tolerancia del proceso: este es el grado de variabilidad que podrá permitir para el cumplimiento del proceso.

.3.3. Elementos del proceso del software:

Los elementos del proceso de software serán: las actividades, los insumos y productos, personas, herramientas, etc. El proceso de software definirá las técnicas, prácticas, roles, etc., para convertir los requisitos de un cliente en producto.

A continuación daré una explicación de los principales elementos del proceso del software:

- Tamaño del proceso: con este elemento se hace referencia al número de elementos que el proceso posee (actividades, estándares, productos, técnicas).
- Densidad del proyecto: a este elemento también se lo denomina como "granularidad", esto es el grado de detalle, consistencia y control que se tiene sobre los elementos.
- Peso del proceso: también llamado "grado de flexibilidad", con este elemento se hace referencia a la combinación de los dos elementos anteriores, esto es al tamaño con la densidad del proyecto.
- Tolerancia del proceso: con este se hace alusión al grado de variabilidad permitido para que el proceso pueda ser cumplido.

.3.4. Modelo del proceso de software:

El modelo de software es una descripción simplificada de un proceso de software (estos a veces son llamados paradigmas del proceso), estos modelo tendrán definidas las actividades que son parte de los procesos y productos de software, además del papel de las personas involucradas en la ingeniería del software.

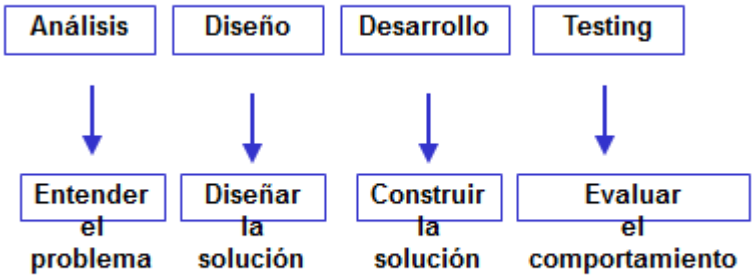
- Nota: considero importante decir que no existe un proceso ideal, ya que la realidad varía en el tiempo, lo que lleva a que los procesos también lo hagan.

El modelo del proceso del software será el encargado de:

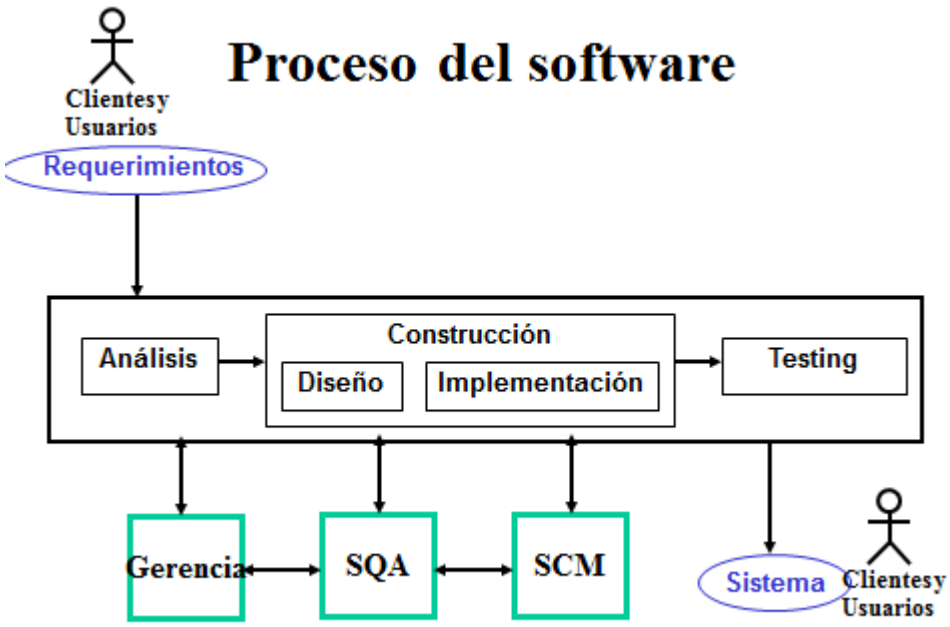
* Describir las principales fases del desarrollo.
* Describir las funciones de cada fase.
* Describir las secuencias de las fases.
* Establecer los criterios para la transición entre fases.

Este nos servirá para "saber donde estamos parados en un momento dado en el proyecto".

- Actividades clásicas para producir software: a continuación podremos ver un diagrama de las principales actividades para producir software, y el objetivo que se pretende con cada una de ellas.



La forma en que las actividades recientemente nombradas se desarrollan es la siguiente:

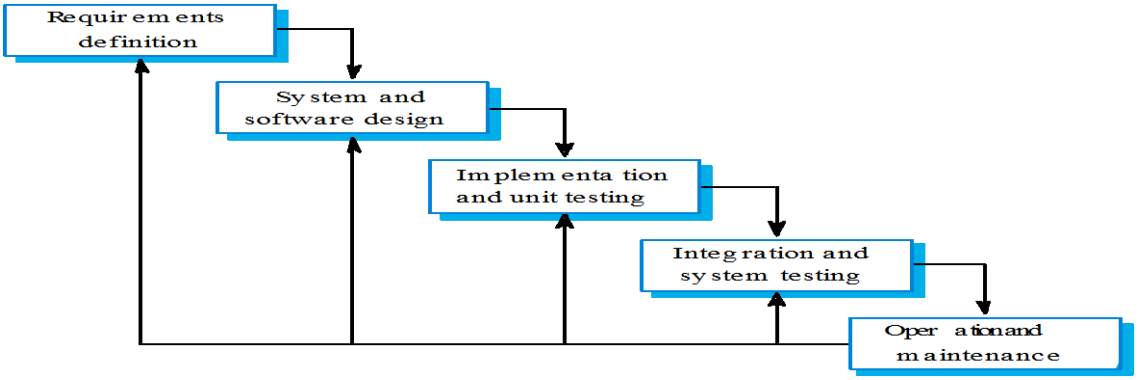


.3.5. Paradigmas de los procesos del software:

Existen diferentes paradigmas de procesos del software, y dentro de ellos diferentes modelos de procesos del software, a continuación desarrollaremos los más importantes:

1) Paradigma secuencial: dentro de este paradigma podremos encontrar a los siguientes procesos del software:

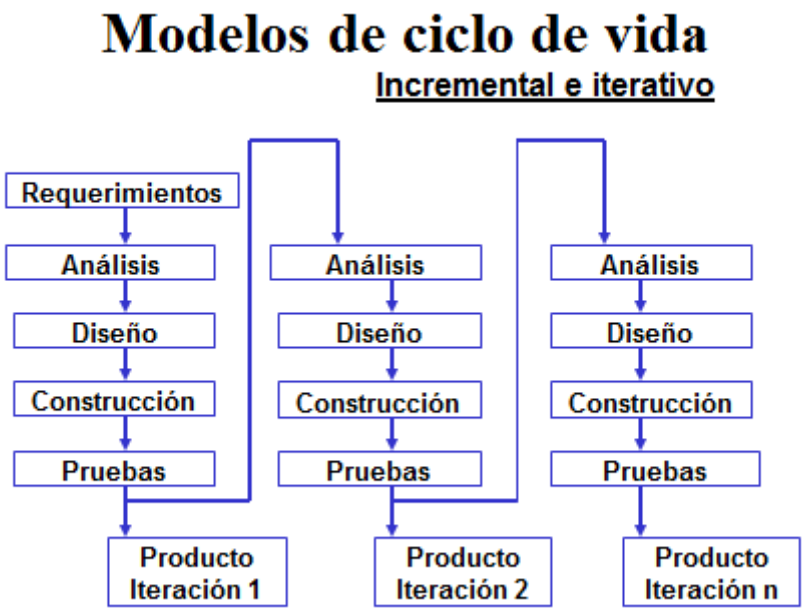
a) Modelo cascada: en este modelo de trabajo el orden de todas las actividades en las transiciones entre las fases es lineal o secuencial.
En este modelo el producto se entrega todo junto al final, antes de llegar al final lo único que se posee es la documentación del mismo. Aquí podremos encontrar fases bien conocidas (cabe decir que este será un modelo eficiente siempre y cuando el sistema sea conocido y los requisitos sean estables).



b) Modelo RAD (Rapid Application Development): este modelo configure una adecuación al modelo clásico de cascada, este proceso dispara varios procesos en paralelos intentando acortar el proceso de desarrollo.
- Ventajas: acelera el desarrollo, optimiza los recursos, estabiliza la arquitectura, el cliente recibe el producto antes.
- Desventajas: requiere de un alto compromiso de los clientes, posee un mayor costo de integración, no se administra de forma correcta los riesgos tecnológicos, mayor apoyo tecnológico.

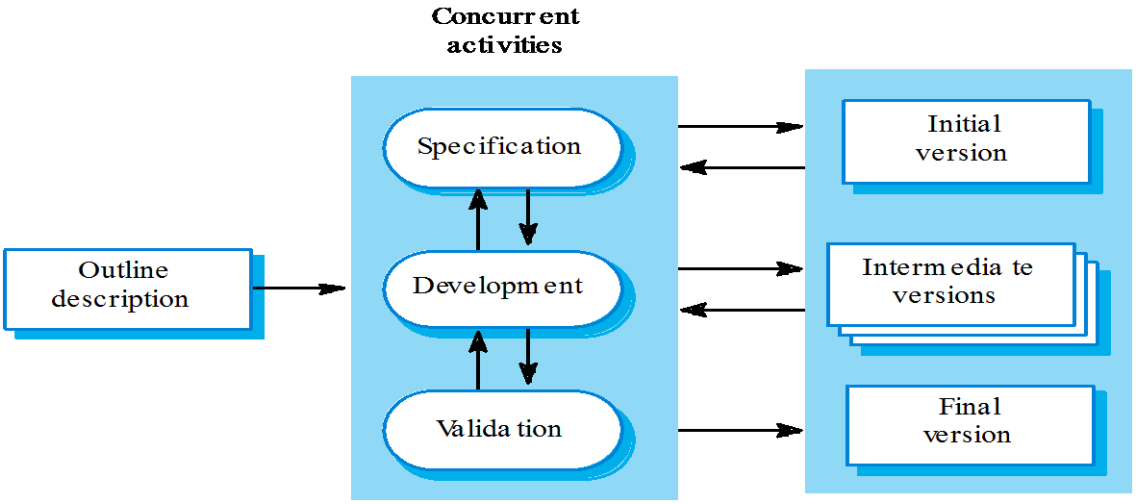
2) Paradigma evolutivo: dentro de los paradigmas evolutivos los principales modelos de software que encontraremos serán:

a) Modelo incremental: este modelo combina el modelo lineal secuencial con el de la prototipación. Aquí se prototipa una fase y se cumple de forma lineal secuencial, luego de terminada esta fase, se planea otra de la misma forma.

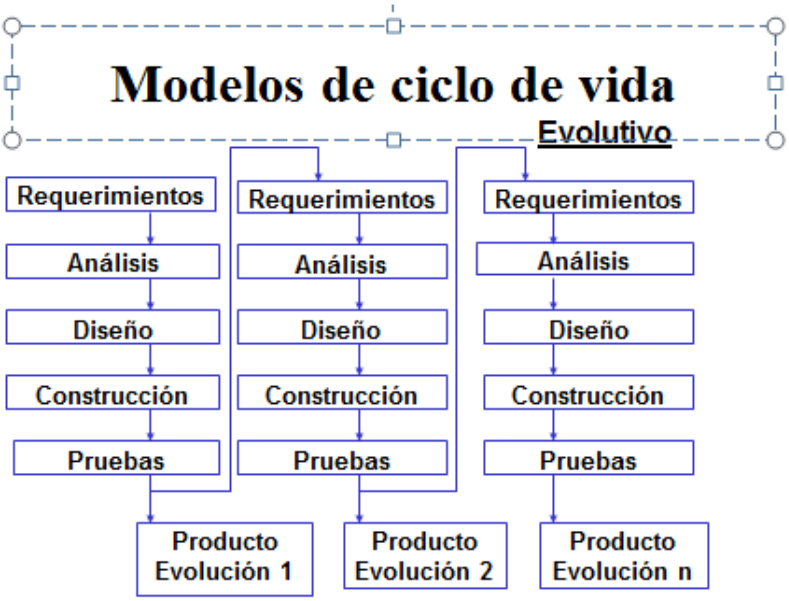


b) Modelo incremental e iterativo: en este modelo la funcionalidad se incorpora por versión, aquí la concurrencia se da entre actividades.
La mayor ventaja de este modelo es la obtención de productos intermedios, al igual que una mejor visibilidad del proceso de desarrollo.
Por otro lado tenemos que decir que la desventaja de este modelo es que se posee un mayor costo de planificación e integración.

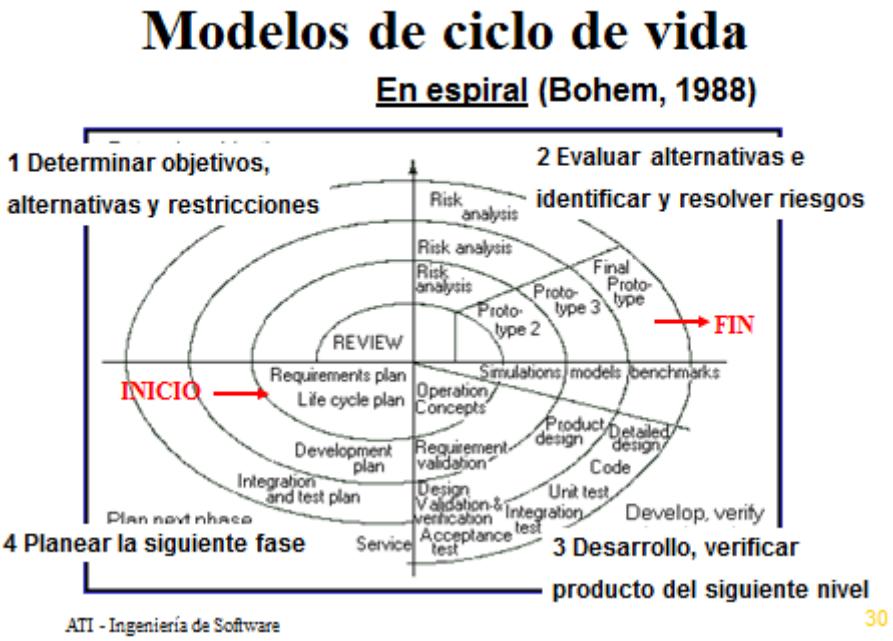
c) Modelo evolutivo: este modelo se basa en la idea de desarrollar una implementación inicial, para luego exponerla a los comentarios de los usuarios, y en base a estas opiniones ir refinándola a través de versiones, hasta la obtención de un sistema adecuado.
Este modelo cuenta con un bosquejo de la descripción, las actividades que comprenderá este modelo serán: especificación, desarrollo y validación, las cuales se desarrollarán de forma concurrente.



Al hablar del desarrollo evolutivo, diremos que existen dos tipos:
* Desarrollo exploratorio**:** aquí se trabaja con el cliente para explorar sus requerimientos y entregar un sistema final. Aquí se arranca con las partes del sistema que mejor se comprenden, para luego ir agregando atributos.
* Prototipos desechables: en este modelo se hacen prototipos de lo que no se comprende, para así poder obtener una definición mejorada de los requerimientos del sistema.



d) Modelo en espiral: este modelo consiste en la asociación de los modelos lineal + el de la prototipación. Aquí se ataca el problema en función de los riesgos, ya que es la primera actividad que se haces. Cada ciclo que termina pregunta ¿Seguimos? Aquí lo que se busca es integrar la gestión de riesgos con el desarrollo, en este modelo cada ciclo produce algo para ser evaluado.



e) Modelo ensamblaje de componentes: en este modelo lo que se busca es especificar los requerimientos, identificar los componentes candidatos, se buscan los componentes en bibliotecas, se extraen los componentes para: reutilizarlos, adaptarlos y ensamblarlos.
- Ventajas: incrementa la reutilización, reduce los costos y el proceso de desarrollo.
- Desventajas: posee mayores costos de planificación e integración, posee costos por la búsqueda de componentes.

3. Paradigma prototipado:

El prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar muchos recursos.
El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará. La interacción ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

4. Paradigma métodos formales:

En este paradigma se suele utilizar el análisis matemático como el elemento conductor de la construcción y fundamentalmente de la verificación. Estos métodos son muy costosos en el desarrollo, y cabe decir que no son de fácil comunicación con el cliente.

5. Paradigma: Agile Software Development: en los años 80 y a principios de los años 90 existía la opinión general de que la mejor forma para el desarrollo de un software esa una planificación rigurosa, utilizando procesos de desarrollo controlados y rigurosos, basados en herramientas CASE (opinión de la comunidad de ingenieros que en ese momento desarrollaban software de larga vida, los cuales se componían por varios programas individuales). Un ejemplo de estos sistemas puede ser el de un sistema de software de un avión, sistema en el cual pueden pasar más de 10 años desde su especificación inicial, a su implementación práctica.
Sin embargo a principios de los 90, cuando se empezó a planificar software con un negocio más pequeño, la implementación de estos métodos ya no era útil, porque se gastaba más tiempo en el análisis y diseño, en vez que en el desarrollo; y cabe decir que un cambio de requerimientos en la marcha del proyecto, daba lugar un nuevo análisis del software q desarrollar.
Toda esta situación dio lugar al surgimiento de los denominados métodos ágiles, los cuales estaban pensados para entregar software funcional de forma más rápida a los clientes, métodos tales como: programación extrema (Beck 1999 - 2000), Scrum (Schwaber y Beedle 2001), Cristal, etc.

- Conceptos bases: al hablar de paradigma flexible tenemos que hacer alusión a dos conceptos bases que forman parte del mismo, estos serán:

* Honestidad del código: con esto hacemos referencia a que el código que funciona (el que ejecuta el sistema) sea real, con esto queremos decir a que el mismo pueda ser visto por el cliente; y además que el código pueda encontrarse sujeto a futuras modificaciones.
* Efectividad de personas trabajando en equipos cooperativos: con esto se hace referencia a que los equipos de desarrollo deben hacer un muy buen uso de la comunicación, con esto nos referimos a ejemplos tales como: que las personas del equipo compartan todos juntos un desayuno de 10 minutos entre todos; con esto hacemos alusión a que los miembros del equipo deben conocerse las caras, saber con quiénes trabajan.

a) Agile (ASD) Manifesto: el agile manifestó consiste en un trabajo conceptual de la ingeniería de software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto.
Las principales ideas que surgirán en este paradigma serán:
* Individuos e interacciones - sobre - procesos y herramientas: se deben reconocer y explotar las habilidades del equipo de desarrollo. Se les debe dejar desarrollar sus propias formas de trabajar, sin procesos formales.
* Software funcionado - sobre - documentación exhaustiva: el software se desarrolla en incrementos, donde el cliente especificará los requerimientos a incluir en cada incremento. Siempre se buscará entregar un software en funcionamiento (aunque el mismo consista en un entregable), en vez de una documentación exhaustiva de lo que es el software.
* Colaborar con el cliente - sobre - negociación de contratos: aquí lo que se busca es que los clientes estén fuertemente implicados en todo el proceso de desarrollo. Su papel es el de proporcionar y priorizar nuevos requerimientos del sistema, al igual que el evaluar las iteraciones del sistema. La idea es hacer participar e implicar al cliente, en vez de estar en todas las reuniones con un abogado o un contrato en mano que establece las reglas de lo que se debe desarrollar.
* Responder al cambio - sobre - seguir un plan: lo que se busca en este paradigma es que el software en desarrollo pueda adaptarse a nuevos cambios, y que este no tenga que ajustarse a un plan en el cual un simple cambio pueda causar un riesgo importante para la culminación del desarrollo del sistema.

- Principios: en base a todo lo desarrollado en los puntos anteriores es que se desprenden los siguientes principios del paradigma flexible:
* Participación del cliente: los clientes deben estar fuertemente implicados en todo el proceso de desarrollo. Su papel es el de proporcionar y priorizar

nuevos requerimientos del sistema, al igual que el evaluar las iteraciones del sistema.
* Entrega incremental: el software se desarrolla en incrementos, donde el cliente especifica los requerimientos a incluir en cada incremento.
* Personas, no procesos: se deben reconocer y explotar las habilidades del equipo de desarrollo. Se les debe dejar desarrollar sus propias formas de trabajar, sin procesos formales.
* Aceptar el cambio: se debe contar con que los requerimientos del sistema cambian por lo que el sistema debe ser diseñado para soportar estos cambios.
* Feedback continuo: con esto se hace referencia a la retroalimentación, esto consiste en la capacidad del emisor para recoger lo que pretende decir el cliente, y para así saber que es lo que se pretende (a lo largo de la vida del desarrollo del sistema) de los diferentes requerimientos del mismo.
* Mantener la simplicidad: el sistema debe centrarse en la simplicidad, tanto en el software como en el proceso de desarrollo. Con esto se hace alusión a que se debe eliminar la complejidad del sistema.
* Foco en el trabajo en equipo: se busca que las personas sepan con quiénes trabajan, para así lograr una buena cooperatividad en el equipo.

b) Programación extrema: la programación extrema es un ejemplo de metodología de este paradigma flexible; aquí todos los requerimientos se expresan como escenarios (llamados historias de usuario), los cuales se implementan como una seria de tareas. Aquí los programadores trabajan en parejas y desarrollan pruebas para cada tarea antes de escribir el código, de forma tal que todas las pruebas se ejecuten de forma correcta cuando el código nuevo se integre al sistema.

.3.6. Conclusión:

* ¿Qué ciclo de vida tendremos que elegir?
En definitiva a la hora de elegir cuál es el ciclo de vida más adecuado para un sistema de software concreto, tendremos que prestar especial atención a los siguientes puntos:

- Cultura de la organización.
- Deseo de asumir riesgos.
- Áreas de aplicación del software.
- Volatilidad de los requerimientos.
- Comprensión de los requerimientos.

- Disponibilidad de recursos.
- Experiencia previa.

*¿Cuál es el proceso correcto?
En base a lo dado diremos que para elegir el proceso correcto tendremos que tomar en cuenta los siguientes rasgos:
- Tamaño del proyecto
- Tamaño del problema.
- Complejidad del problema.
- Tipo de organización.
- Requerimientos.
- Calidad del equipo de trabajo.
- Lenguaje de programación.
- Criticidad.
- Ciclo de vida elegido.
- Dominio del producto.

Cabe decir que en general para proyectos grandes existe la necesidad de utilizar diferentes enfoques de paradigmas y procesos, razón por la cual terminaremos utilizando un modelo híbrido para el desarrollo del software; mientras que si pensamos en procesos iterativos, en general la especificación del software se realizará junto con el desarrollo del mismo.

.4. “Fases del proceso de la Ingeniería de Software:”

.4.1. Introducción:

Como ya hemos dicho en otros capítulos, la ingeniería de sistemas es una actividad interdisciplinaria que incluye a la ingeniería de software (la ingeniería de software respeta los principios de la ingeniería de sistemas, y además aporta su “granito de arena” a ella).

Para comenzar con el desarrollo de este punto tendremos que decir que podremos encontrar 7 fases del proceso de la Ingeniería de software, ellas serán:

1) Definición de requerimientos.
2) Diseño del sistema.
3) Desarrollo de los sub – sistemas.
4) Integración.
5) Instalación y operación.
6) Evolución.
7) Desmantelamiento.

.4.2.1. Fase 1 – Definición de requerimientos:

La definición de los requerimientos del software es realizada por los clientes y usuarios finales del sistema. Por medio de esta definición es que obtendremos:

a) Los requerimientos funcionales (estas serán las funcionalidades en concreto, o sea, las funcionalidades pensando en un nivel abstracto).
b) La propiedad del sistema, esto vendría a ser los requisitos no funcionales (un ejemplo de requisito no funcional sería: querer que una funcionalidad “x” sea desarrollada en 6 semanas, o en un lenguaje determinado).
c) Las características que pretendemos que no sean mostradas por el sistema (por ejemplo: si una persona va a pedir un préstamo a un banco, en caso de que se le deniegue, que el sistema no muestre un mensaje diciendo que no se habilita el pago porque la persona es deudora, dando detalles de estas deudas).

Como conclusión diremos que esta fase lo que pretende es lograr determinar un conjunto completo de los objetivos que el software debe cumplir.

.4.2.2. Fase 2 – Diseño del sistema:

El objetivo de esta fase será el de proporcionarle al sistema la funcionalidad, y ello se logrará a través de sus componentes. Las actividades que se desarrollarán en esta fase serán:

a) Dividir requerimientos.
b) Identificar sub – sistemas.
c) Asignar requerimientos a los sub – sistemas.
d) Especificar funcionalidad de los sub – sistemas.
e) Definir las interfaces.

En esta fase todo se realiza mediante la retroalimentación e iteración. En cuanto al diseño de solución diremos que existen varios modelos aplicables, simplemente tendremos que tener en cuenta que la solución elegida para el desarrollo debe ser aquella que resulta más apropiada para cumplir con los requerimientos del sistema.

.4.3. Proceso del software:

Como ya hemos dicho en el capítulo anterior el proceso del software es "un conjunto coherente de actividades para la producción de software (desde cero o no) " – Ian Somerville.

Las actividades básicas que podremos encontrar en los diferentes procesos serán:
- Especificación: esta etapa también es llamada "Ingeniería de requerimientos", esta configura un momento particularmente crítico debido a que los errores cometidos aquí darán lugar a un gran impacto posterior en el desarrollo del sistema.
Por intermedio de esta etapa es que se lograremos:
* Llegar al producto final: esto es un documento de requerimientos.
* Destino y diferentes niveles: usuarios finales y clientes de alto nivel.
* Desarrolladores del sistema.
* Etc.

- Diseño e implementación: en esta fase es cuando la especificación se convierte en un sistema ejecutable, esto se logra mediante la entrega de versiones sucesivas. Cabe decir que un diseño de software es la descripción de la estructura del software que se va a implementar, los datos, las interfaces entre componentes, e inclusive a veces, los algoritmos utilizados. En esta etapa a medida que vamos profundizando en el desarrollo del sistema, podremos encontrar errores y omisiones, razón por la cual por

intermedio de la retroalimentación y la repetición del trabajo lograremos solucionarlos.
*Depuración: la depuración consiste en llevar a cabo las pruebas del código desarrollado; en base a esta definición es que logramos diferencia al concepto de prueba del concepto de depuración, esto es así porque: la prueba establece la existencia de defectos, mientras que la depuración comprende la localización y corrección de estos defectos. La depuración se configura por intermedio de las siguientes etapas:

1	Localizar el error
2	Diseñar la reparación del error
3	Reparar el error
4	Volver a probar el programa

- Validación: la validación y verificación del software (V & V) nos servirá para demostrar que el sistema se encuentra acorde a su especificación y que además cumple con las expectativas del usuario.
Esta fase incluye los procesos de comprobación, como ser las inspecciones en cada etapa del proceso de software (desde la definición de requerimientos hasta el desarrollo del programa).

- Evolución: esta es la fase en donde el software es modificado para adaptarlo a los cambios requeridos por el cliente.

.5. “Ingeniería de requerimientos”:

.5.1. Introducción:

Para comenzar con el desarrollo de este tema debemos preguntarnos ¿Cuál es el objetivo de la ingeniería de requerimientos?, y la respuesta a esta pregunta es la de lograr expresar los requerimientos del sistema, esto es: comprender y enunciar el problema con la finalidad de enunciar y validar la solución.
Los principales aspectos a desarrollar aquí serán: a) Requerimientos del usuario y del sistema, b) Requerimientos funcionales y no funcionales, y c) Como organizar estos requerimientos en un documento de requerimientos.

La gran dificultad en la ingeniería de requerimientos se encuentra en el poder establecer que es lo que el sistema debe hacer, por ende podremos decir que la ingeniería de requerimientos es el proceso que consta en el: descubrir, analizar, documentar y verificar los requerimientos. Pero ¿qué son los requerimientos? Los requerimientos son las descripciones y las restricciones de los servicios que el sistema ofrecerá.

- Definición: “un requerimiento es una declaración abstracta de alto nivel de un servicio que debe proveer el sistema, o también lo será, una restricción del servicio”.
Otra definición que se puede dar es: “una función matemática detallada y final, de una función del sistema”.
- Importancia: en base a un estudio realizado por el autor Bohem es que se resalta la gran importancia de los requisitos, este autor en su estudio publicó que:

> * El 45% de los errores detectados en los sistemas, se deben a una mala o poca especificación de los requerimientos.
> * Estos errores fueron encontrados en etapas más tardías del desarrollo de los sistemas.
> * Cuanto antes se encuentre un error, más barato será su corrección.

- Concepto de la ingeniería de requerimientos: en base a lo anteriormente dicho es que podemos decir que la ingeniería de requerimientos es un conjunto de actividades, en las cuáles utilizando técnicas y herramientas, se analizará un problema para concluir con una especificación de la solución (cabe decir que la “especificación de la solución” es: el conjunto de requerimientos del sistema que indican **QUE** hace el sistema y no el **COMO** lo hace).

En la ingeniería de requerimientos podremos encontrar las siguientes actividades:

* Extracción de requerimientos: esto es la definición del problema.
* Especificación de requerimientos: esto es la definición del producto.
* Validación de requerimientos: esto consiste en el "asegurar" completitud y correctitud.
* Administración de requerimientos.

.5.2. Ingeniería de requerimientos:
Como hemos dicho al comienzo de este tema, podremos encontrar diferentes clases de requerimientos, y a continuación daremos un breve acercamiento a cada uno de ellos:
a) Requerimientos del usuario: estos son las declaraciones en lenguaje natural y en diagramas, de los servicios que se espera que el sistema provea (además de las restricciones sobre las cuáles el sistema deberá operar).
b) Requerimientos del sistema: estos son aquellos que establecen con mayor detalle los servicios y restricciones del sistema.
c) Especificación del diseño del software: si bien este enunciado no fue nombrado en la introducción del presente punto, aquí se suele agregar el detalle a la especificación de los requerimientos del sistema, este configurará el puente entre la ingeniería de requerimientos y las actividades de diseño.

- Ejemplo de diferencia entre requerimientos del usuario y del sistema:
* Usuario: a) el sistema debe proveer un medio para representar y acceder a archivos externos creados por otras herramientas.
* Sistema: a) al usuario se lo proveerá con los recursos para definir el tipo de archivos externos, b) cada tipo de archivo externo tendrá una herramienta asociada que será aplicada al archivo, c) Cada tipo de archivo externo se representará como un ícono específico sobre la pantalla del usuario, d) etc.

- Diferentes lectores de requisitos: también es importante decir que existen diferentes tipos de lectores de requerimientos, y estos lectores variarán dependiendo del tipo de requerimiento, por ejemplo: requerimientos del usuario: administradores, clientes, usuarios finales del sistema, etc.; requerimientos del sistema: usuarios finales del sistema, ingenieros clientes, etc.; especificación del diseño del software: ingenieros clientes, arquitectos del sistema, desarrolladores, etc.

.5.3. Características de los requerimientos:
Las principales características de los requerimientos serán:
a) Sin ambigüedad: que un requerimiento no sea ambiguo quiere decir que este tiene una única y posible interpretación para cualquier lector (usuario, cliente, constructor, analista, etc.).
Por ejemplo: los intereses se calcularán mensualmente, y se pagarán trimestralmente según calendario comercial.

b) Correctitud: un requerimiento será correcto sí y solo sí "cae" dentro del espacio del producto.
Por ejemplo: un sistema de contabilidad no es correcto si realiza liquidación de haberes.

c) Completitud: que el requerimiento sea completo quiere decir que el mismo cuenta con: 1) todas las necesidades del usuario se ven reflejadas, y 2) existe un mapeo completo entre las entradas y salidas del sistema.

d) Consistencia: un requerimiento será consistente sí y solo sí no contradice otro requerimiento.
Por ejemplo: R1: para el cálculo de todos los haberes generados se considerará el calendario comercial. // R2: para el cálculo del salario vacacional se considerará el calendario civil.

e) Verificabilidad: un requerimiento será verificable sí y solo sí es posible determinar sin ambigüedad, a lo largo de un proceso, cuando una implementación lo satisface.

.5.4. Tipos de requerimientos:
Como veremos en este punto, existen diferentes tipos de requerimientos, los tipos de requerimientos que podremos encontrar serán:

a) Requerimientos funcionales: los requerimientos funciones son declaraciones de los servicios que proveerá el sistema, explicarán el cómo se comportará el sistema frente a situaciones especiales. Estos también declararán de forma expresa lo que el sistema no deberá hacer.
- Definición: un requisito funcional es la condición o capacidad de un sistema requerido por el usuario, para poder resolver un problema o alcanzar un objetivo. Este determina lo funcional, esto es (desde el punto de vista del usuario) el **QUE** debe hacer el sistema.
- Función: estos deberán describir la funcionalidad o los servicios que se espera que el sistema provea. Estos siempre van a depender del tipo de

sistema y de los usuarios, ya que: a) si son requerimientos de USUARIO, estos se describirán de forma general, b) pero si son requerimientos de SISTEMA, estos serán descritos en forma detallada (cada función, sus entradas, salidas, etc.).
- Ejemplos: el sistema debe emitir un informe de estado de situación patrimonial. || el sistema debe realizar la liquidación de haberes a destajistas, personal contratado, etc.

b) Requerimiento no funcional: los requisitos no funcionales son las restricciones a los servicios o funciones, como pueden serlo: el tiempo, el proceso de desarrollo, estándares, etc.
- Definición: estos requisitos determinan lo **NO FUNCIONAL**. Estos configuran la condición o capacidad que debe poseer un sistema para satisfacer un contrato, estándar, especificación, y otro documento formalmente impuesto.
Estos refieren a las propiedades emergentes del sistema, como ser: la fiabilidad, la respuesta en el tiempo, la capacidad de almacenamiento. Esta clase de requerimientos de forma alternativa definirán las restricciones del sistema como ser la capacidad de los dispositivos de entrada y salida, y la representación de los datos en las interfaces del sistema.
- Clasificación: a los requerimientos no funcionales los podremos clasificar en:
* Requerimientos del producto: estos son aquellos que especifican su comportamiento, como por ejemplo: desempeño, fiabilidad, portabilidad, etc.
* Requerimientos organizacionales: son aquellos que refieren a estándares, implementación de determinados lenguajes, etc.
* Requerimientos externos: estos son los requerimientos que provienen fuera del sistema y del ambiente de desarrollo, como ser: la interoperabilidad con otros sistemas, requerimientos éticos, legales, etc.
- Ejemplos: el sistema deberá correr bajo Windows Xp || el sistema deberá ser desarrollado en JAVA || el sistema deberá desarrollarse en 1 año.

c) Requerimientos de dominio: estos requisitos se derivan del dominio del sistema más que de las necesidades específicas de los usuarios. Estos pueden ser requerimientos funcionales nuevos, pueden restringir los ya existentes, o pueden establecer como se deben ejecutar ciertos cálculos particulares.
Por ejemplo: a) uso de estándares en interfaz de usuario para todas las bases de datos || b) borrado de datos inmediatamente después de su llegada.

.5.5. Requerimientos de usuarios y de sistema:

Como ya hemos dicho con anterioridad a lo largo de este tema, podremos encontrar requerimientos de usuarios, y requerimientos de sistema.

- Requerimientos de usuario: estos describen los requerimientos funcionales y no funcionales de forma tal que sean comprensibles para un usuario que no posea conocimiento técnico.

Estos especifican el comportamiento externo del sistema, dejando de lado las características de diseño del sistema. Estos son requerimientos que deberán ser redactados en lenguaje natural, con representaciones y diagramas sencillos e intuitivos.

Si bien estos requerimientos deben escribirse en lenguaje natural, esta característica da lugar a problemas como:

* Falta de claridad: razón por la cual uno debe evitar ser ambiguo a la hora de detallarlos; estos deben ser claros y fáciles de leer.

* Confusión de requerimientos: se debe ser preciso evitando así toda posible confusión acerca de si un requisito es funcional o no funcional.

* Conjunción de requerimientos: debe evitarse expresar varias requerimientos como si configurasen uno sólo.

Como posibles soluciones a este problema se recomienda:

* Inventar un formato estándar, y utilizarlo siempre.

* Utilizar el lenguaje de forma consistente, para así lograr diferenciar los requerimientos deseables (esto son los “debería”), de los requerimientos obligatorios.

* Resaltar el texto en negrita o itálica para aquellas partes claves del documento.

* Evitar el lenguaje de computación o programación en la redacción de los mismos.

- Requerimientos del sistema: estos son las descripciones detalladas de los requerimientos del usuario. Estos servirán como base para poder definir el contrato de la especificación del sistema.
Será a partir de estos requerimientos que los ingenieros de software tomarán como punto de partida para el desarrollo del sistema; en esta clase de requerimientos se realizarán diagramas de objetos y de flujo de datos.
* Función: estos requisitos deberán establecer lo que el sistema hará, y no la manera de como el sistema lo resolverá. Al igual que en los requerimientos de usuario, se nos pide que estos requerimientos sean escritos en lenguaje natural, pudiendo dar lugar a los siguientes problemas:

* Malas interpretaciones en base a la ambigüedad de la redacción.

* Puede darse lugar a la generación de requerimientos flexibles.

Como posible solución a estos problemas se plantean las siguientes alternativas:

* Utilizar un lenguaje natural estructurado: como ser planillas o estándares en la redacción.

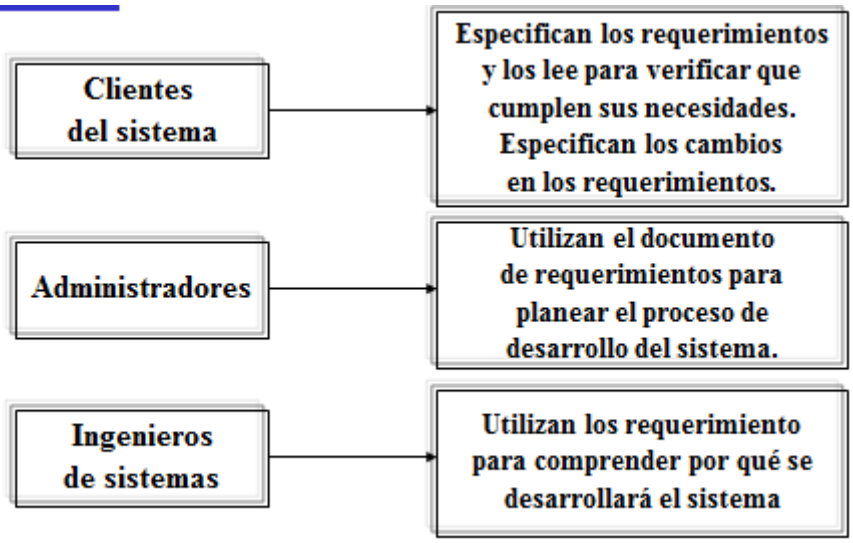
* Utilizar lenguaje de descripción de diseño: como por ejemplo CdeU.

* Utilizar notaciones gráficas: esto es utilizar lenguaje gráfico con Anotaciones de texto.

* Especificación de interfaces: esto nos sirve para cuando el sistema debe operar con otros sistemas ya implementados e instalados en el Entorno. Cabe decir que podremos encontrar dos tipos de interfaces:
a) De procedimientos: como ser aquellos subsistemas que ofrecen variedad de requerimientos.
b) De estructuras de datos que pasan de un subsistema a otro.

.5.6. El documento de requerimientos:

El documento de requerimientos es la declaración oficial de lo que requieren los desarrolladores del sistema. Este documento contemplará tanto a los requerimientos del usuario, como a los requerimientos del sistema.



Al hablar del contenido que debe brindar este documento, podremos encontrar diferentes opiniones, por ejemplo:

a) Autor Haninger (año 1980): este autor considera que este documento debe contar con la siguiente información:

1) Debe especificar únicamente el comportamiento externo del sistema.
2) Debe especificar las restricciones de la implementación.
3) Será fácil y abierto para posibles modificaciones.
4) Servirá como herramienta de referencia para los encargados del mantenimiento del sistema.
5) Deberá contener las previsiones del ciclo de vida del sistema.
6) Deberá caracterizar las respuestas aceptables para los eventos no deseados del sistema.

b) Documento de la IEEE: este documento recomienda que el documento de requerimientos debe contar con las siguientes partes:

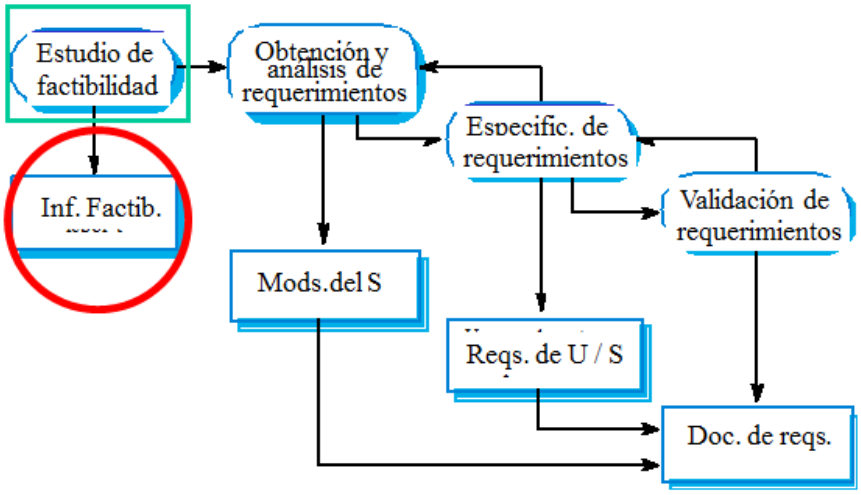
* Introducción: en esta parte debe establecerse el propósito del documento, el alcance del producto, las definiciones, abreviaturas, resumen del resto del documento, etc.

* Descripción general: aquí se debe dar una perspectiva del producto, se deben especificar las funciones del producto, las características del usuario, las restricciones generales, etc.
* Requerimientos específicos: estos son aquellos requerimientos funcionales, no funcionales y de interfaz. Esta será la parte más importante del documento.
* Apéndice: el cual proveerá información detallada y precisa que estará relacionada con la aplicación que se desarrolla. Algunos ejemplos de apéndices que pueden incluirse son las descripciones del hardware y la base de datos. Aquí los requerimientos del hardware definirán las configuraciones mínimas, y las óptimas del sistema; mientras que los datos definirán la organización lógico de los datos utilizados por el usuario.
* Índice: se deberán incluir varios índices en el documento, como por ejemplo: alfabéticos, de diagramas, de funciones, etc.

- “Ingeniería de requerimientos – Parte 2: Proceso”

.5.2. Introducción:

La ingeniería de requerimientos es un proceso que comprende todas las actividades requeridas para la elaboración y mantenimiento de un ***Documento de requerimientos del sistema***.
Existen cuatro actividades genéricas de alto nivel que desembocarán en la obtención de este documento: a) estudios de factibilidad, b) obtención y análisis de requerimientos, c) validación de requerimientos, y d) administración de requerimientos.



a) Estudio de factibilidad: siempre que se vayan a desarrollar sistemas nuevos se debe inicializarse conociendo la viabilidad del mismo, esto es, sabiendo si el mismo es viable (posible) de desarrollar o cumplir. Para ello siempre se parte de un descripción resumida, aquí es una etapa basada en la recolección y evaluación de la información. El resultado de esta etapa será la elaboración de un informe que contiene una recomendación del seguir o no con el proyecto.
- Preguntas a realizarse: en esta fase debemos respondernos preguntas como: ¿Qué pasa si no se hace?, ¿Problemas actuales y como se resolverán?, ¿Cómo enganchar esto con otras tecnologías?, ¿Se utilizará tecnología nueva?, etc.
- Objetivo: el informe de factibilidad deberá recomendar el cuándo y cómo se debe hacer el proyecto, este además podrá proponer cambios en el alcance, presupuesto y calendarización del proyecto.

b) Obtención y análisis de requerimientos: una vez que se realiza el estudio de factibilidad es que ingresamos a la actividad de la obtención y análisis de requerimientos. Aquí el personal del desarrollo técnico del software trabajará con clientes y usuarios para determinar la aplicación, estos es: que servicios proveerá, el desempeño requerido, las restricciones de hardware, etc.
Por intermedio de esto es que llegaremos a los siguientes resultados:

Extracción de requerimientos = definición del problema.

Especificación de requerimientos = definición del producto.

- StakeHolder (ST): el ST es cualquier persona que posee una influencia directa o indirecta sobre los requerimientos del sistema.

Esta segunda actividad es una actividad difícil porque los ST no conocen lo que desean desarrollar, tampoco utilizan términos propios, diferentes ST poseen diferentes requerimientos, puede darse lugar al cambio de políticas o inclusive a la aparición de nuevos ST.

En base a esto diremos que las tareas que componen el proceso de la ingeniería de requerimientos serán:

1) Comprensión del dominio: en esta etapa el analista debe desarrollar su propia comprensión del dominio de la aplicación. Por ejemplo: si se requiere de un sistema para un supermercado, el analista debe averiguar como operan los supermercados.
2) Recolección de requerimientos: esta tarea consiste en el interactuar con los ST del sistema para así descubrir los requerimientos (como veremos, la comprensión del dominio se daría en esta tarea).
3) Clasificación: una vez que se descubren los requerimientos, llegamos a la tarea de la clasificación, esta reside en organizar los requerimientos en grupos coherentes.
4) Resolución de conflictos: cuando nos encontramos en proyectos que poseen muchos ST, lo más probable es que se de lugar a conflictos en los requerimientos, esta tarea se basa en la resolución de estos conflictos, para así no dar lugar a requerimientos (por ejemplo) ambiguos.
5) Priorización: ahora nos encontramos en una etapa en la cual tenemos todos los requerimientos del software, pero no los tenemos priorizados. Esto es que tendremos que ordenar los requerimientos a desarrollar en un orden determinado, priorizando aquellos más esenciales para el desarrollo

del sistema; podemos ver que en esta etapa será de vital importancia la interacción de los desarrolladores técnicos con los ST.
6) Verificación de requerimientos: y por último se llegará a la tarea de verificación, aquí lo que se busca es descubrir si los requerimientos son completos, esto es, si son consistentes y acordes con lo que los ST pretendían.

- Alcance de esta fase: esta fase posee 3 alcances:
1) Relevamiento de requerimientos: esta segunda tarea busca aprender (conocer, analizar y describir) el problema a resolver. También busca identificar todo lo involucrado (accionistas, fuentes de requerimientos, procesos, datos y recursos), entender las necesidades de los usuarios, las restricciones del problema, etc.
En este primer alcance surgen conceptos como:
*** Espacio del producto:** este es el rango del problema que cumplen todas las restricciones.
*** Fuentes de restricciones de nuestro problema:** usuarios (ya que estos definirán la funcionalidad, los tiempos de respuesta, la interfaz, etc.), el cliente (ya que serán los encargados de definir la funcionalidad, tiempos de desarrollo, costos, confiabilidad, etc.), el constructor (quién definirá los costos, atributos a tomar en cuenta, etc.), tecnologías (capacidades del cliente, trabajo de migración de sistemas, conversión de datos, etc.), leyes y restricciones gubernamentales (como por ejemplo: impuestos).
*** Técnicas de relevamiento:** aquí podremos encontrar técnicas de comunicación, como ser: entrevistas, tormentas de idea, cuestionarios, etc.; y entrevistas de descripción, como ser: diagramas de flujo de datos, modelos de entidad relación, diagramas de clases, etc.
- Conclusión: como podemos ver todas estas etapas darán lugar a la especificación de requerimientos para la elaboración del "Documento de requerimientos".

2) Especificación de requerimientos: para especificar los requerimientos se recomiendan dos técnicas de caja negra del sistema (esto es, del comportamiento del sistema), ellas son: ESRE y Casos de Uso.
*** ESRE:** el ESRE es un documento de especificación de requerimientos, este documento describe de forma exacta lo QUE el producto es y lo que se debe CONSTRUIR. Este es un documento que captura los resultados del relevamiento y las características de soluciones aceptables al problema. Ejemplos: el sistema debe emitir un informe de estado de situación patrimonial; el sistema debe correr sobre WINDOWS XP o superior; el sistema deberá ser desarrollado en C++.

Un ejemplo de organización de un ESRE es:
1) Introducción; 2) Descripción; 3) Especificación de requerimientos, 4) Referencias; 5) Anexos.

Cabe decir que como contenido mínimo de un ESRE tendremos que tener: objetivos del producto, identificación de actores, requerimientos funcionales, y requerimientos no funcionales.

Las principales características del ESRE serán: Correcto (debe representar algo que necesita), no ambiguo (debe existir una única interpretación para todos), completo (debe cumplir con todos los objetivos), verificable (con esto queremos decir que todos sus requisitos son verificables), consistente (que ningún requerimiento contradiga a otro), claro (que no deje dudas al lector), anotado (debe clasificar sus requerimientos en base a prioridad de implementación).

*** Casos de uso:** por otro lado encontraremos a los casos de uso, estos tienen objetivo el especificar los requerimientos, también busca establecer la base para el diseño, busca la validación del diseño, la base para el testing, y la base para el manual de usuario del sistema.

b.1) ¿Cómo realizar la obtención y el análisis de requerimientos?

Como ya hemos dicho anteriormente, la fase de la obtención y el análisis de requerimientos se desenvuelve mediante de las siguientes 6 tareas:
1) Comprensión del dominio, 2) Recolección de requerimientos, 3) Clasificación, 4) Resolución de conflictos, 5) Priorización, y 6) Verificación de requerimientos. Y como también ya dijimos, todas estas tareas se realizan con el fin de lograr la especificación de requerimientos para el "Documento de requerimientos".

Pero llegado este punto nos preguntaremos: ¿Cómo hacemos esto?

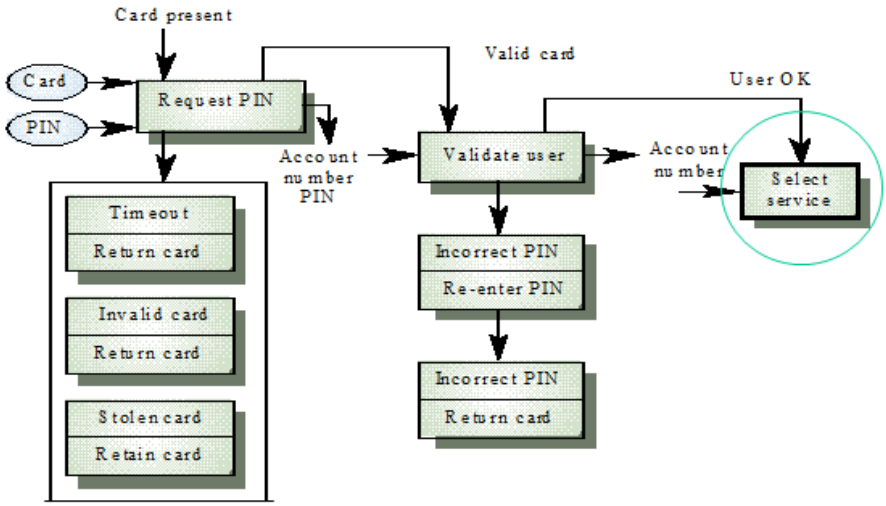
1) Obtención orientada a puntos de vista: frente a la determinación de requerimientos siempre existirán diferentes puntos de vista, esto es así porque cada interlocutor puede tener una perspectiva diferente sobre un mismo requerimiento. Lo que logramos por intermedio de los puntos de vista es el encontrar los conflictos, y las partes en común.
- Concepto: un punto de vista puede ser tanto una fuente (de lo que emerge) o un consumo de datos (esto implica identificar que datos se producirán y

que datos se van a consumir). Otra definición que se ha dado de punto de vista es la de que el punto de vista es un marco de trabajo de la representación, y cabe decir que también se lo ha definido como un receptor de servicios.

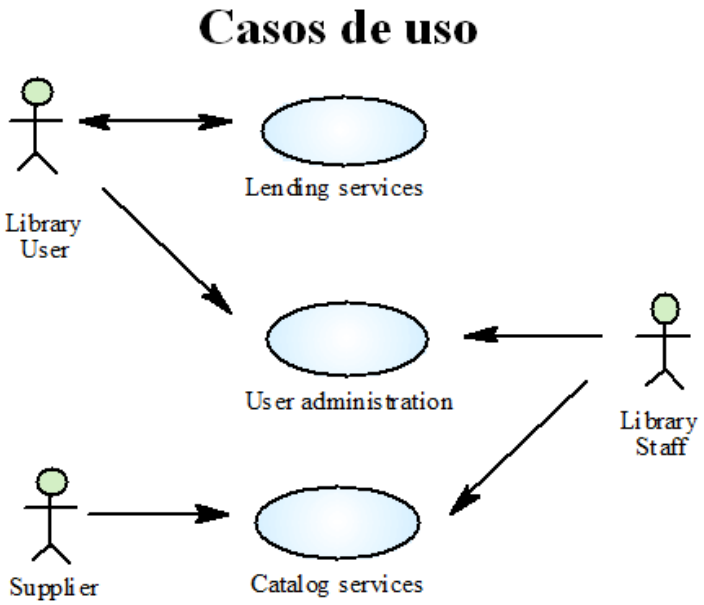
2) Escenarios: a las personas le resulta mucho más fácil el dar ejemplos de la vida real, que tener que desarrollar descripciones abstractas. Estos ejemplos es a los que se les puede denominar como "escenarios", y cabe decir que podría ser muy útil para agregar detalles al bosquejo de los requerimientos.
Estos escenarios incluirán: una descripción del estado del sistema al inicio, una descripción del flujo normal de eventos, una descripción de lo que puede ir mal y como se debe manejar, información de otras actividades concurrentes, descripción del estado final, etc.
- Funcionamiento: en la vida real, los ingenieros trabajarán con los ST (stakeholders) para lograr identificar los escenarios y los detalles de los mismos. Cabe decir que de forma alternativa se puede utilizar un enfoque mas estructurado, como ser los escenarios de eventos o los casos de uso.

* Escenarios de eventos: los escenarios de eventos incluyen una descripción del flujo datos y las acciones del sistema (en estos también se documentan las excepciones que puedan surgir). Aquí cada evento de interacción distinto se documenta como un escenario de eventos diferente (ejemplo: inserción de tarjeta en ATM, selección de un servicio en ATM, etc.).

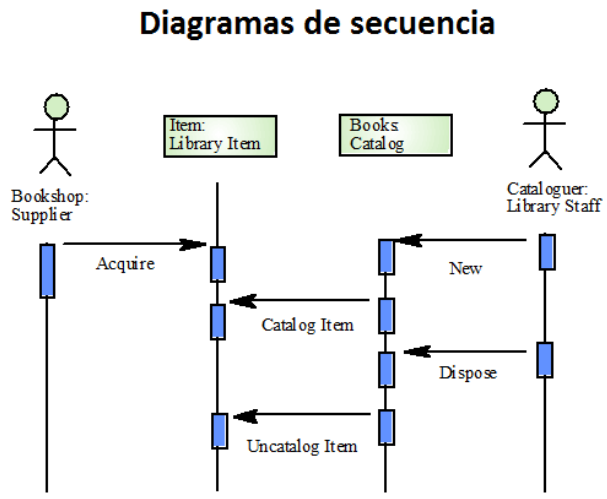
Ejemplo: ESCENARIO DE EVENTOS – TRANSACCION DE INICIO



* Casos de uso: los casos de uso son una técnica que se basa en escenarios para la obtención de requerimientos. La introducción de esta herramienta fue realizada por Jacobson en el año 1993, cabe decir que hoy en día estos configuran la base de la notación UML que se utilizar para describir los modelos de sistemas orientados a objetos. Los casos de uso tienen como objetivo la identificación de los actores involucrados en una interacción.



Como hemos dicho, el conjunto de casos de uso representa todas las posibles interacciones que ocurrirán en los requerimientos del sistema.
Para agregar información a los casos de uso, es que muchas veces se recurre a los diagramas de secuencia. Estos diagramas serán los encargados de mostrar: los actores involucrados, los objetos del sistema, y las operaciones asociadas con dichos objetos.



3) Etnografía: la etnografía consiste en una técnica de observación para poder obtener los requerimientos. Esta técnica se funda en la premisa de que los sistemas de software no existen de forma aislada, sino que se utilizan en un contexto social y organizacional. De esta forma concluye estableciendo que los requerimientos de sistemas de software se derivan y se restringen acorde al contexto social y organizacional en el que se utilizan.

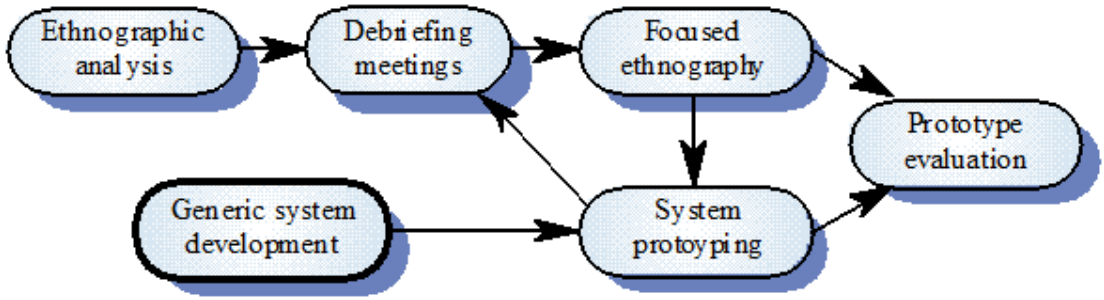
- Tipos de requisitos: la etnografía apunta a describir dos tipos de requisitos:

a) Aquellos que se derivan de la forma en que trabaja la gente, más que en las definiciones de los procesos.

b) Aquellos que surgen de la cooperación y el conocimiento de las actividades de la gente.

- Nota: considero importante decir que normalmente no basta sólo con la etnografía, sino que también será necesaria la utilización de otras técnicas para la obtención de los requerimientos (como ser por ejemplo: casos de uso).

- Ejemplo:



c) Validación de requerimientos: para la validación de requerimientos nos encontramos con técnicas como: JAD (Joint Application Design), grupos motivacionales, prototipación, encuestas, etc.

- Objetivo de la validación: el objetivo de la validación es el de corregir las omisiones y malas interpretaciones, antes de continuar con el desarrollo del sistema. Tendremos que tener en cuenta, que el descubrimiento de un error aquí tendrá un costo y un daño mucho mayor, que en una etapa mucho más avanzada.

- Preguntas que nos podemos realizar: en esta fase debemos realizarnos preguntas como:

* ¿Qué se valida?: aquí validamos los requerimientos funcionales, de eficiencia, de ingeniería humana, etc. Aquí se valida desde el punto de vista de sus características, esto es: ambigüedad, correctitud, completitud, consistencia, etc.
* ¿Quiénes lo validan?: la validación va a ser desarrollada por dos grupos de personas: a) los que lo ven de afuera (clientes, usuarios), y b) los que lo desarrollaron (analistas de requerimientos y desarrolladores).
* ¿Cómo se validan?: estos son validados mediante la revisión de casos de uso, revisión de otros requerimientos, mediante la utilización de prototipos, JADs, etc.
* ¿Cuándo se validan?: estos son validados una vez que se haya terminado de especificar los requerimientos (o al menos un conjunto de ellos).
* ¿Dónde se validan?: se debe crear un ambiente propicio para la validación.
* ¿Por qué se validan?: se validan para poder saber con exactitud y precisión lo que se va a construir.
* ¿Para qué se validan?: se valida para poder continuar con las siguientes etapas del sistema.

- Revisión de requerimientos: la revisión de requerimientos puede desarrollarse de dos formas: a) informal: esta la desarrolla el proveedor con tantos stakeholders como se pueda; y b) formal: aquí el proveedor de desarrollo deberá “conducir” al cliente a través de los requerimientos del usuarios, explicándole las limitaciones de cada requerimientos.
Una vez que el documento esta “OK” se proceda a terminar el “Documento de especificaciones”.

.5.3. Administración de requerimientos:

- Introducción y concepto: los requerimientos para sistemas de softwares "grandes" y/o "complejos" siempre son *cambiantes*, con esto queremos decir que una vez que se definen los mismos podrán ir variando, esto puede darse tanto por situaciones internas (como ser el cambio de mentalidad del cliente) o por situaciones externas (como ser la aprobación de una nueva ley). Debido a que los requisitos son cambiantes, estos también serán *incompletos*, debido a que un requisito no podrá definirse de forma completa.
Durante el proceso del software la comprensión del problema por el desarrollador irá cambiando, y estos cambian retroalimentarán a los requerimientos.
- Objetivo: por ende diremos que el principal objetivo de la administración de requerimientos será el de comprender y controlar los posibles cambios en los requerimientos del sistema. En base a lo anteriormente dicho surgirá también el concepto de planeación.
La planeación comenzará al mismo tiempo de la obtención de los requerimientos iniciales, y la administración activa de los requerimientos deberá comenzar tan pronto esté listo el documento de requerimientos.
- Requerimientos duraderos y volátiles: como ya hemos dicho anteriormente, con el paso de tiempo se modifica el entorno del sistema y los objetivos del negocio, dando lugar a nuevos cambios; en base a esto a que podremos dividir a los requisitos en:

    * Duraderos: estos son aquellos más o menos estables, estos son los que se derivan de la actividad principal de la organización y que están relacionados directamente con el dominio del sistema.
    * Volátiles: estos son los requerimientos que variarán a los largo del desarrollo del sistema. Estos poseen las siguientes características: son mutantes (debido a los cambios en el ambiente que opera la organización), son emergentes (surgen al incrementarse la comprensión con el cliente), son consecutivos (ya que al introducirse se da un cambio en el proceso de la organización, y se produce un cambio en la forma de trabajo), y son de compatibilidad (si cambian otros sistemas, esto deberán cambiar también).

-Planeación: Como hemos dicho anteriormente la planeación comenzará al mismo tiempo de la obtención de los requerimientos iniciales. En esta etapa de la planeación se tomarán decisiones sobre:

* Identificación de requerimientos: aquí se buscará identificar a los requerimientos de forma única, sin que se de lugar a la ambigüedad entre requerimientos.
* Proceso de administración del cambio: se buscará planear el conjunto de actividades que evalúan el impacto, y el costo del cambio.
* Políticas de rastreo: estas serán las políticas que definirán la relación entre requerimientos, y además la relación entre requerimientos y sistema.
* Ayuda de herramientas CASE: aquí se buscará ver que herramientas CASE se podrán utilizar, la cuales permitan sacar provecho a tanta información.

- Administración del cambio de requerimientos: utilizando un proceso formal, la administración del cambio de requerimientos asegura que frente a un cambio propuesto, los requerimientos sean tratados de forma consistente, y que además los cambios en el documento de especificaciones se realice de forma controlada.
Aquí a partir de un cambio identificado se hace: 1) se realiza un análisis del problema y la especificación del cambio, 2) se realiza un análisis y costeo del cambio, y 3) se realiza la implementación del cambio para obtener los requerimientos revisados.

.6. “Proceso (relevamiento)”:

.6.1. Introducción:

En este punto se hace referencia a las técnicas utilizadas para la obtención y análisis de requerimientos (considero importante decir que estas técnicas también son utilizadas para la realización de estudios de factibilidad).
El relevamiento de requerimientos sirve para definir el espacio del producto, por espacio hacemos referencia al: aprender (conocer, analizar y describir) el problema a ser resuelto, el identificar todo lo involucrado (accionistas, fuentes de requerimientos, procesos, datos, recursos, etc.), entender las necesidades de los usuarios, entender las restricciones del problema, etc.
Como ya hemos dicho en otros temas, para el relevamiento de requerimientos existen dos tipos de técnicas, y ellas son:

a) **De comunicación:** entrevistas, tormenta de ideas, grupos motivacionales, cuestionarios, Jad, observación, prototipación, etc.
b) **De descripción:** diagramas de flujos de datos, diagrama de transición de estados, modelos de entidad relación, diagrama de clases, etc.

.6.2. Técnicas de comunicación:

Las principales técnicas de comunicación son:

a) Entrevista: (COPIAR TRES TRIÁNGULOS DE APUNTES).
- Principales problemas: los principales problemas de esta técnica de comunicación son: el entrevistar a las personas equivocada en el momento equivocado; el hacer preguntas equivocadas y luego obtener respuestas equivocadas; y por último el crear fricciones entre ambas partes (entrevistador - entrevistado).
- Reglas generales: las reglas generales en la entrevista son: a) desarrollar un plan global de entrevistas; b) contar con la aprobación previa para llegar a los entrevistados; c) Planear bien la entrevista para aprovechar mejor el tiempo; d) Conocer los temas de mayor interés para el entrevistado (esto configura el “comenzar” de la entrevista); y e) usar un estilo apropiado de entrevista.
- Posibles resistencias del entrevistado: a la hora de entrevistar a una persona, la misma puede presentar las siguientes resistencias: que la entrevista ocupa demasiado espacio de su tiempo; que la misma amenaza

su empleo; que se pretende cambiar la forma en que la persona trabaja; que la persona no quiera un nuevo sistema, etc.
Otros posibles problemas que se deben tener en cuenta son: usuarios que no saben lo que quieren, desacuerdos entre colegas, subordinados y jerarcas, la confusión entre síntomas y problemas, etc.

b) Observación: la técnica de la observación consiste en el estudio de antecedentes de un tema determinado, en el tratar de revivir la/s situación/es reales, y en el análisis de documentación.
c) Cuestionarios: por otro lado podremos encontrar a la técnica de los cuestionarios, estos son un buen medio para la obtención de información; cabe decir que existen diferentes tipos de cuestionarios, como ser: abiertos – cerrados, formulación – evaluación, elección de la muestra, etc.
d) Joint Application Design (JAD): el principal objetivo del JAD es el del diseñar el sistema entre un conjunto de usuarios y relevadores. La actividad central del JAD reside en la sesión de relevamiento estructurada. Los participantes que encontraremos en esta técnica son: usuarios y relevadores, y el desarrollo de esta se utiliza llevando a cabo una agenda detallada la cual va generando un documento del sistema a construir.
- Fases del JAD: el JAD se configura por 5 fases bien definidas:
*** Fase 1- Definición del Proyecto:** en esta fase lo que se busca es determinar el espacio del producto, y por ende definir el equipo de trabajo. Los pasos a seguir en esta fase serán: entrevistar al Director / Gerente, entrevistar a los Responsables de cada área, después a los responsables del proyecto, para así después pasar a la selección del equipo, y por último a la confección del cronograma del JAD.
Conclusión de la fase: en esta fase se llega a la definición gerencial, a la lista y roles de JAD, y por último a la generación del cronograma del JAD.
Roles JAD: a la hora de seleccionar el equipo para el JAD, podremos encontrar roles bien diferenciados, estos serán:

a) Esponsor: este es un integrante de los usuarios del sistema, que va a poseer mayor autoridad respecto de los demás; esta persona tendrá la función de facilitar el conocimiento acerca de lo que busca el compromiso del resto del grupo.
b) Líder JAD: este será el encargado de llevar a cabo todo el proceso del JAD. Este será un miembro imparcial del JAD, este no corresponde ni a un miembro de los usuarios, ni a un miembro de desarrollo.
c) Escriba: este será el encargado de registrar todas las decisiones de requerimientos tomadas en la sesión del JAD.

d) Participantes: y dentro de los participantes del JAD podremos encontrar aquellos que son full – time, Ad hoc y Observadores.

*** Fase 2 - Investigación:** el principal objetivo de esta fase será el de conocer los procesos de la organización, al igual que el conocer los sistemas actuales que funcionan en la misma (en el caso de que estos últimos existan).
Los pasos a seguir en esta fase serán: en primer lugar debemos relevar los procesos actuales, luego especificarlos para así poder pasar a un nuevo esbozo de los nuevos procesos a implementar. Y sería a partir de este punto que nos encontramos en condiciones de confeccionar la agenda del JAD.

*** Fase 3 – Preparación:** en esta fase se tiene por objetivo el clasificar y documentar la información que se relevó en las fases 1 y 2, además se busca preparar la sesión JAD.
Los pasos a seguir en esta fase son: en primer lugar confeccionar un documento de trabajo, en segundo lugar confeccionar un guión JAD, confeccionar el material de trabajo, y por último distribuir productos.
Conclusión: la conclusión de esta fase es la de la obtención de un documento de trabajo, un guión del JAD, y también del material de trabajo.
Documento de trabajo: el documento de trabajo es un documento que contiene: la identificación del documento, el prefacio (propósitos y audiencias), generalidades del JAD (metodología JAD), agenda, supuestos, cuestiones abiertas, etc.

*** Fase 4 – La sesión JAD:** el objetivo de esta fase será el de negociar y aprobar requerimientos. Los pasos a seguir en este punto serán: discutir supuestos, cerrar cuestiones abiertas, determinar quienes son los que reciben el documento final, evaluar sesión vía cuestionario.

*** Fase 5 – Producción del documento final:** el objetivo de esta fase será el de la obtención de un documento con requerimientos de los usuarios, ya aprobados y negociados.
Es en esta fase que se cierran las cuestiones que estaban abiertas, y se distribuye el documento final.

- Beneficios: los principales beneficios del JAD son:
* Esta técnica contribuye a crear sentido de grupo, tanto de relevadores como de usuarios.
* Se logra el compromiso de los usuarios.

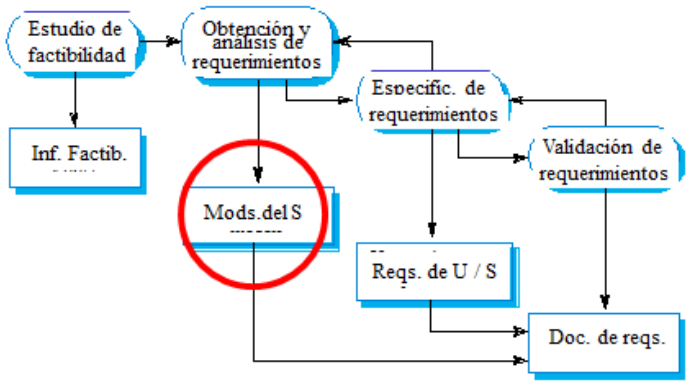
* Se minimizan las visiones encontradas o en conflicto.
* Se disminuye tiempo de reuniones y por ende de relevamiento.

- Diez mandamientos del JAD: los diez mandamientos de esta técnica serán:
* Los participantes full – time deben asistir a todas las sesiones.
* Los derechos de todos los participantes son iguales.
* La preparación del JAD es tan importante como la sesión del JAD.
* Evitar la jerga técnica.
* Ser puntual.
* Confeccionar una buena agenda y administrar el tiempo de acuerdo a ella.
* Asegurarse de incluir a la gente correcta.
* Buscar un espacio fuera del habitual de trabajo para mejorar la productividad.
* Un JAD exitoso requiere de compromiso gerencial al más alto nivel.
* Tener todas las especificaciones preparadas antes de la sesión del JAD.

.7. Proceso (otros medios de especificación de requerimientos):

.7.1. Introducción:

En el punto anterior hemos visto a la especificación de requerimientos, ahora veremos los modelos de sistema.



.7.2. Modelos de sistemas:

El modelo de sistema es una vista abstracta de un sistema que ignora algunos detalles del mismo. Considero importante decir que se pueden desarrollar modelos de sistemas complementarios que muestren información pero desde una perspectiva diferente, de esta forma podremos encontrar las siguientes perspectivas:
* Perspectiva externa: esta es la perspectiva que modela el contexto o entorno. Los modelos de sistema de contexto serán los encargados de definir los límites del sistema y las dependencias con el entorno.
* Perspectiva de comportamiento: la cual modela el comportamiento el sistema. Dentro de estos modelos podremos encontrar a los modelos de flujo de datos (modelo en cual se explica el como se procesan los datos a través de las diferentes funciones que actúan sobre ellos), modelos de máquinas de estado (estos son los modelos que se utilizan para modelar el comportamiento del sistema en respuesta a los eventos internos o externos).
* Perspectiva estructural: la cual modela la arquitectura del sistema, o la estructura de datos procesados por éste. Estos modelos describen la estructura lógica de los datos que importan o que son exportados por el sistema, estos muestras entidades del sistema, sus atributos y la relación en la que participan. Cabe decir que estos modelos se complementar con diccionarios de datos (los cuales poseen una descripción más detallada de estos).
* Perspectiva lógica: dentro de esta perspectiva podremos encontrar a los modelos de sistema de objetos, estos modelos describen la lógica del sistema, su clasificación y su agregación. Estos serán los encargados de representar los datos del sistema y su procesamiento. Los posibles modelos de objetos que se pueden desarrollar incluyen: modelos de herencia, de agregación de objetos y de comportamiento de objetos.

.7.3. Construcción de prototipos:

En la construcción de prototipos se desarrollan prototipos los cuales son brindados a los usuarios para que éstos adquieran una visión completa de las capacidades del sistema. Los prototipos ayudarán a establecer y validar requerimientos.
- Tipos: considero importante decir que existen 2 clases de prototipos:
a) Evolutivos: que son aquellos en los cuales a partir de una versión de un prototipo se van generando otras, siempre evolucionando sobre la anterior hasta culminar en un producto final (cabe decir que en este tipo de

prototipo siempre se comienza con las partes que mejor se comprenden del prototipo).
b) Desechables: estos son los que se desarrollan para comprender mejor los requerimientos a desarrollar (en este tipo de prototipos se suele comenzar con que menos se entienden.)
- Construcción de prototipos: en la construcción de prototipos se suelen utilizar lenguajes de alto nivel, programación de base de datos, y cabe decir que la construcción se realiza en base a componentes reutilizables.
La construcción se basa en el concepto de desarrollo rápido, dejando de lado las restricciones no funcionales.

.8. “Plan de proyecto”:

.8.1. Introducción:

En la primera parte del curso hemos aprendido a cómo fijar objetivos para un proyecto, a como obtenerlos (esto es el QUE), al igual que a conocer el entorno y a conocer el alance y limitaciones del mismo. También hemos llevado adelante un estudio de alternativas a seguir (COMO Y CON QUE). Lo que queremos decir con esto, es que en la primera parte del semestre hemos analizado el “QUÉ HAREMOS”; en esta sección veremos el “COMO HACERLO”.

.8.2. Administración de proyectos:

Los proyectos de software nunca son totalmente iguales a otros, razón por los cuál se deben idear técnicas especiales de administración, para poder realizar una buena administración de los mismos. A la hora de la realización de estas técnicas especiales se tendrá que tomar en cuenta tanto a las restricciones de presupuesto, como a las de calendario.
Los principales puntos a tomar en cuenta en esta sección serán: las actividades, planeación, cronograma, riesgos, personal, costo, y calidad.

.8.2.1 Actividades:

En este punto podremos decir que existen un conjunto de actividades que son comunes a la administración de todos los proyectos, ellas son:
a) Redacción de la propuesta: en la redacción de la propuesta se van a describir los objetivos del proyecto, y el cómo se hará para llevarlos adelante. En general en esta etapa se suelen incluir estimados de costo y calendario, y se justifica el porque se deben contratar a los desarrolladores (esta configura una práctica crítica, y es una habilidad que se adquiere con la experiencia).
b) Planeación y cronograma del proyecto: aquí se identifican las actividades, hitos y entregas producidas por el proyecto. Aquí se tendrá listo el bosquejo del plan que guía a las metas del proyecto.
c) Costeo del proyecto: esta configura una fase muy difícil del proyecto.
d) Supervisión y revisión del proyecto: la supervisión configura una actividad continua, aquí el administrador debe tener conocimiento del progreso del proyecto y debe comprar avance y costos con lo planeado. Mientras que la revisión apunta al prestar atención con los cambios “permanentes” a los que el proyecto se enfrenta.

e) Selección y evaluación del personal: en esta actividad se debe establecer el equipo ideal para llevar adelante el proyecto, las dificultas a enfrentar aquí serán: sueldos altos, encontrar gente con mucha experiencia, etc.
f) Redacción y presentación de informes: esta actividad deberá ser desarrollada tanto por los usuarios como por el proveedor, estos deberán ser concisos y coherentes y deberán resumir los detalles de los mismos en forma clara y precisa.

.8.2.2. Planeación del proyecto:

Una buena administración de un proyecto está basada en el saber planear el progreso del propio proyecto. El administrador del proyecto deberá poder anticiparse a los posibles problemas que puedan surgir, al igual que preparar posibles soluciones para ellos. Y cabe decir que el plan hecho al inicio del proyecto deberá funcionar como hilo conductor del proyecto. Además del plan general del proyecto, se deberán preparar otros tipos de plantes, como ser: planes de calidad, validación, administración de la configuración, de mantenimiento, etc.
- Camino a seguir: el camino a seguir para la planeación del proyecto será: 1) se debe iniciar con la valoración de las restricciones que afectan el proyecto, 2) se debe hacer una estimación de la estructura, tamaño y distribución de las funciones, 3) se deben definir los hitos y los productos a entregar, 4) posteriormente el plan debe tener actualizaciones sucesivas, y así dar lugar a la renegociación.
Siempre tendremos que tener en cuanta que las bases a tomar serán más "pesimistas" que "positivistas", esto nos permitirá manejarnos con cierta holgura, y por ende, tranquilidad.

.8.2.3. Cronograma:

Los administradores deben estimar el tiempo y los recursos requeridos para el desarrollo de una actividad, y para así poder ordenarlas de forma coherente (es por esta razón que se prepara un calendario del proyecto). Las estimaciones que se realizan al comienzo en general son inciertas, y terminan representándose como un conjunto de gráficos que muestran la división del trabajo y la asignación del personal al mismo. En general dentro de esta tarea podremos encontrar a la herramienta Ms Project, y dentro de ellas a la realización de Gráficos de barras (Gantt) y Redes de actividades (PERT).

.8.2.4. Administración de Riesgos:

Otra actividad muy importante es la de la anticipación de los riesgos que pueden afectar al proyecto o a la calidad del software. En esta fase se deben identificar los riesgos y se deben crear plantes para minimizar sus efectos en el proyecto.
- Concepto de riesgo: el riesgo es una probabilidad de que una circunstancia adversa ocurra.
El análisis de los riesgos es una actividad muy importante debido a las incertidumbres por las que un proyecto atraviesa, por ejemplo: una pobre definición de requerimientos, la dificultad en la estimación de tiempos y recursos, las dependencias de habilidades individuales, etc.
- Categorías: considero importante decir que existen 3 tipos de categorías de riesgos, ellos son: a) de Proyecto: estos son los que afectan al calendario o a los recursos del proyecto, b) del Producto: estos son los que afectan la calidad o el desempeño del software que se está desarrollando, y c) del Negocio: que son los que afectan a la organización que desarrolla el software.
- Objetivo: el principal objetivo en esta fase, será el de tener desarrollados planes de contingencia para el caso de que estos riesgos ocurran.
Algunos ejemplos universales de riesgos son: rotación de personal, cambio de una administración, no disponibilidad del hardware, bajo desempeño del CASE, cambio de tecnología, etc.
- Etapas del proceso: considero necesario decir que en la administración de riesgos podremos encontrar las siguientes etapas:
* Identificación de riesgos: aquí se hace un análisis para descubrir los posibles riesgos del proyecto. Esta es una fase que se hace a partir de lluvia de ideas o por experiencias anteriores. El resultado de esta fase será el de la obtención de una larga lista de riesgos que puedan ocurrir y afectar.
Los tipos de riesgos a identificar serán: riesgos de tecnología, de personas, organizaciones, de herramientas, de requerimientos, etc.
* Análisis de riesgos: en esta fase a los riesgos se los considerará por separado, y se decidirá acerca de su probabilidad y efectos. En la probabilidad se suelen utilizar escalas, por ejemplo: muy baja <= 10%, baja 10% al 25%, moderada 25% al 50%, alta 50% al 75% o muy alta > 75%. Mientras que los efectos (o seriedad) se miden en: catastrófico, serio, tolerable, o insignificante.
El resultado de esta fase será el de la obtención de una tabla que será ordenada por efecto, y cabe decir que esta tabla no será estática, sino que será constantemente actualizada.

* Planeación: en la planeación de riesgos se trabajo con los riesgos obtenidos en las etapas 1 y 2, aquí se trata de definir el "qué hacer con ellos", aquí se pretende desarrollar: 1) una estrategia de anulación (se busca intentar reducir la probabilidad de que estos ocurran), 2) estrategia de disminución (se trata de reducir el impacto de ellos en caso de acontecimiento) y 3) planes de contingencia (esto es la elaboración de caminos alternativos a seguir en caso de acontecimiento de alguno de los riesgos previstos).
* Supervisión: los riesgos no son estáticos, sino que estos pueden ir variando en el tiempo, es por ello que deben ir supervisando a lo largo de la vida del proyecto para ver si estos varían. Considero importante decir que es muy difícil observar los riesgos, y que no es recomendable la utilización de factores de riesgos para dar indicios de posibles problemas.

.8.2.5. Administración del personal:

Los humanos tenemos una memoria rápida de corto plazo, una de trabajo y otra de largo plazo. La resolución de problemas comprenderá la integración de información a partir de la memoria de largo plazo con la nueva información de la memoria a corto plazo.
Los principales aspectos a tener en cuenta en la administración del personal serán: a) Personas y su motivación: como se cubrirán sus necesidades, el tipo de profesionales existentes; b) Trabajo en grupo: la composición del grupo, la cohesión del grupo, la organización, y la comunicación en el grupo.

.8.3. Estimación del costo:

.8.3.1. Introducción:

En esta fase se pretende lograr estimar el costo y el esfuerzo requerido para la producción de software, aquí se busca: fundamentar el precio y el costo del software, generar métricas para medir la productividad, generar técnicas para estimar costos y duración.
En esta fase tendremos que realizarnos preguntas como: ¿cuánto esfuerzo se requiere para completar cada actividad?, ¿cuánto tiempo calendario se necesitar para completar cada actividad?, ¿cuál es el costo total de cada actividad?
- Componentes del costo de desarrollo: el costo se configura por diferentes componentes, estos son: a) Hardware y software (más su mantenimiento), b) viajes y capacitación, c) de esfuerzo ("dominante"), d) de apoyo al

esfuerzo (oficinas, su clima, iluminación, redes y comunicaciones, recursos compartidos como ser bibliotecas, recreativos, etc.).
- Precio y costo: las estimaciones de costo se hacen para obtener el costo de desarrollo en la producción del software. Los factores que se tomarán para determinar el precio serán: oportunidad de mercado, incertidumbre en la estimulación de costos, términos contractuales, salud financiera, etc.
- Medidas de productividad: en la actividad manufacturera la productividad se mide en unidades/ personas – horas. Pero en software eso es difícil porque se podrán encontrar factores como seguridad, eficiencia, mantenibilidad, etc.; que no son factibles de ser medidos (conceptos que tienen una gran influencia en la definición de la "calidad" de software). Sin embargo debemos estimar la productividad del software basándonos en algunos atributos del software.
Los criterios que se han utilizados para medir la productividad del software son:
a) Relacionadas con el tamaño: por ejemplo se toman las líneas de código fuentes entregadas, o de páginas de la documentación.
Por ejemplo: líneas de código LCD. Este criterio se relaciona con el tamaño, antes era más fácil de hacerlo, pero ahora es difícil porque con los lenguajes de alto nivel el esfuerzo es menor, y esto lleva a que la productividad pareciera que fuera menor, pero todo lo contrario, es mejor.
b) Relacionadas con la función: se expresa en términos de cantidad de funcionalidad ÚTIL producida en un tiempo dado.
Sin embargo es necesario decir que podremos encontrar problemas a las hora de determinar las cosas a medir, ya que: se debe estimar el tamaño de lo que se va a medir, se debe estimar el número total de personas, se debe estimar la productividad de equipos tercerizados como por ejemplo la documentación para luego incorporarla a la estimación total.
Por ejemplo: puntos de función. En esta técnica se miden los puntos de función producidos por una persona / mes independientemente del lenguaje, aquí los puntos de función se calculan midiendo varias características del programa, como por ejemplo: entradas y salidas, iteraciones con el usuario, interfaces externas, etc.
Otro ejemplo son los puntos de objeto. Los puntos de objeto configuran una alternativa a los puntos de función (estos son utilizados en el modelo de estimación de COCOMO 2). Para estimar los PDO nos basaremos en: a) cantidad de pantallas independientes que se despliegan (1 = simple, 2 o 3 = complejas), b) cantidad de informes que se producen (2 = simple, 5 u 8 = muy difíciles), c) cantidad de módulos 3GL que se deben desarrollar para completar el código 4GL.

Los factores que pueden afectar la productividad de un proyecto serán: la experiencia en el dominio de la aplicación, la calidad del proceso, el tamaño del proceso, el apoyo técnico, el ambiente de trabajo, etc.

.8.4. Técnicas de estimación de esfuerzo:

.8.4.1. Introducción:

A partir de pocos datos de requerimiento de alto nivel, desconociendo equipos, a los usuarios, etc.; es muy difícil poder hacer estimaciones de costos precisas (con esto queremos decir que es difícil hacer estimaciones de costos precisas en etapas tempranas del proyecto).
A la hora de realizar estimaciones de esfuerzo, las mismas se pueden hacer por medio de dos enfoques: **a)** Ascendente: aquí se divide el proyecto en componentes, y se va haciendo estimaciones por etapas; **b)** Descendente: se parte del sistema total, y se va estimando hacia abajo.

.8.4.2. Técnicas de estimación de esfuerzos:

Las principales técnicas para la estimación de esfuerzos (según el autor Bohem año 1981) son:
- Modelado del algorítmico de costos: esta técnica se basa en información histórica que se posee, utilizando métricas (como por ejemplo el tamaño del proyecto); aquí mediante la métrica seleccionada se predice el esfuerzo requerido (cabe decir que como una de estas técnicas podremos encontrar al COCOMO en sus dos versiones.
Esta técnica parece ser el enfoque más sistemático, sin embargo no es el más preciso, aquí se utiliza una fórmula matemática para predecir costos basados en estimaciones de tamaño del proyecto.
En el modelado algorítmico se deben realizar tres estimaciones: la peor, la esperada y la mejor.
*** Técnica COCOMO:** dentro del modelado algorítmico de costos podremos encontrar a la técnica del COCOMO; este modelo es un modelo empírico, este modelo se obtuvo mediante la recolección de datos de muchos proyecto grandes, y a partir de estos, se descubrieron fórmulas utilizadas en ellos.
Desarrollo: esta técnica supone un proceso en cascada y de desarrollo desde cero, esta se basa en 3 niveles: a) Básico: aquí se provee una estimación inicial burda; b) Segundo: se modifica la estimación inicial utilizando multiplicadores del proyecto y del proceso; c) Más detallada: se producen estimaciones en cada fase del proceso.
*** Técnica COCOMO 2:** dentro del modelado algorítmico de costos es que también podremos encontrar a la técnica evolucionada del COCOMO 2;

esta es una técnica que posee 3 niveles de avance según el proyecto: a) Construcción inicial de prototipos: aquí se realizan estimaciones basadas en Puntos De Objeto, y con el uso de fórmulas simples para estimar el esfuerzo; b) De diseño inicial: aquí se termina de determinar los requerimientos y se realiza algo de diseño inicial, aquí las estimaciones se realizan en Puntos de Función que se convierten a LDC; c) Post arquitectónico: aquí una vez diseñada la arquitectura del sistema, se competan los nuevos atributos y se trabaja basado en LDC.

En el COCOMO 2 para cada nivel se emplean fórmulas específicas, además en el nivel 3 se puede contar con 2 factores más, ellos son: a) la volatilidad de los requerimientos, y b) la amplitud de la posible reutilización. Considero importante decir que en este tercer nivel además se incorpora: la capacidad del personal, el conocimiento del producto, y las características del proyecto.

- Opinión de expertos: en esta técnica se busca a uno o más expertos y se comparan sus estimaciones, se discuten y luego se llega a una estimación.
- Estimación por analogía: en este técnica se busca un proyecto similar, y si se encuentra se toma la estimación hecha para ese proyecto.
- Ley de Parkinson: esta ley establece que el trabajo se extiende para hasta llenar todo el tiempo disponible. Aquí el costo se calcula por los recursos disponibles, en todo el tiempo que se dispone para el desarrollo del proyecto (ejemplo: si el proyecto se debe entregar en 12 meses y tengo 5 personas el costo del proyecto será el de 60 personas por mes).
- Asignar precios para ganar: en esta técnica el esfuerzo estimado se hace en proporción al presupuesto del cliente, y no en base a la funcionalidad del software. Esta es un técnica de estimación poco apropiada y ética, pero es en la realidad se utiliza con mucha frecuenta. La ventaja de esta técnica es que el alcance del proyecto se determina en base al poder financiero del cliente.

.8.4.3. Duración y personal del proyecto:

Dividir el esfuerzo requerido por la duración del proyecto no da para dar una indicación útil del número de personas. Generalmente el número de personas crece desde un número pequeño hasta un número máximo y después se reduce.
Al comienzo de las tareas de planeación y especificación se requiere de un número pequeño de personas, y al progresar en el proyecto se requerirá un de un trabajo más detallado, llegándose a un máximo de personas Y luego

de la implementación y de las pruebas unitarias, el número de personas decaerá hasta alcanzar 1 o 2 para la entrega final del producto.
- Modelo Putnam: el modelo Putnam es un modelo que incluye el tiempo de desarrollo como un factor clave para la estimación del esfuerzo; por ende diremos que si el tiempo para el desarrollo decrece, el esfuerzo requerido para terminarlo será mayor.

.9. "Aseguramiento y estándares:"

.9.1. Introducción:

Los aseguramientos y estándares serán los encargados de fijar el marco de trabajo, los procesos de calidad serán los encargados de definir o seleccionar los estándares que se aplicarán, y cabe decir que estos estándares podrían estar incrustados en procesos aplicables en el desarrollo (asimismo estos procesos pueden apoyarse en herramientas que capturan el conocimiento de estándares de calidad).

.9.2. Tipos de estándares:

Al hablar de los tipos de estándares diremos que existen dos tipos, ellos son: **a)** Estándares del producto: estos son los estándares que serán aplicables al producto de software a desarrollar, estos incluyen estándares a nivel de: documentación, documentos y codificación; **b)** Estándares del proceso: estos contienen la definición de los procesos a seguir en el desarrollo del software, estos incluyen: la definición de procesos de especificación, diseño y validación; y la descripción de los documentos a generar en el transcurso de estos procesos.

- Importancia: la importancia de la utilización de estándares reside en que estos proveen un conjunto compacto de las mejoras prácticas a seguir; además estos proveen el marco de trabajo para implementar el proceso de calidad; y por último diré que ayudan a mantener la continuidad del trabajo.

A través de los estándares el equipo de QA deberá crear su manual con los estándares apropiados para la organización, en general estos estándares se ven como burocráticos e irrelevantes, y para la mayoría de los casos los formularios que los configuran son vistos como algo tedioso.

La relación entre ambos tipos de estándares es muy cercana, como ejemplo de ellos podremos encontrar: **a)** Estándares del producto: formulario de revisión, formato del encabezado o del procedimiento, estilo de programación en un lenguaje determinado, formato del plan de proyecto, etc., **b)** Estándares del proceso: conducto para la revisión del diseño, sometimiento de documentos, proceso de entrega de las versiones, proceso de control de cambio, etc.

.9.3. Calidad del proceso y del producto:

La calidad de un producto está influida por la calidad del proceso que lo obtuvo, los atributos de calidad del producto son difíciles de evaluar, la relación entre ambos tipos de calidades es compleja y la gran diferencia con otras disciplinas está dada por la estandarización y supervisión.

La administración de la calidad del proceso comprende tres tareas: 1) definir estándares del proceso, 2) Supervisar el proceso de desarrollo, y 3) Hacer informes del proceso.

- Planeación de la calidad: la planeación de la calidad se inicia siempre en etapas tempranas del proyecto, el plan de calidad consta en: definir la calidad deseada para el producto, seleccionar los estándares apropiados, tener en cuenta métodos, herramientas, etc.

- Atributos de la calidad: a nivel de ejemplo de atributos de calidad de software podremos encontrar: seguridad, comprensión, portabilidad, protección, experimentación, usabilidad, fiabilidad, adaptabilidad, reutilización, complejidad, robustez, etc.

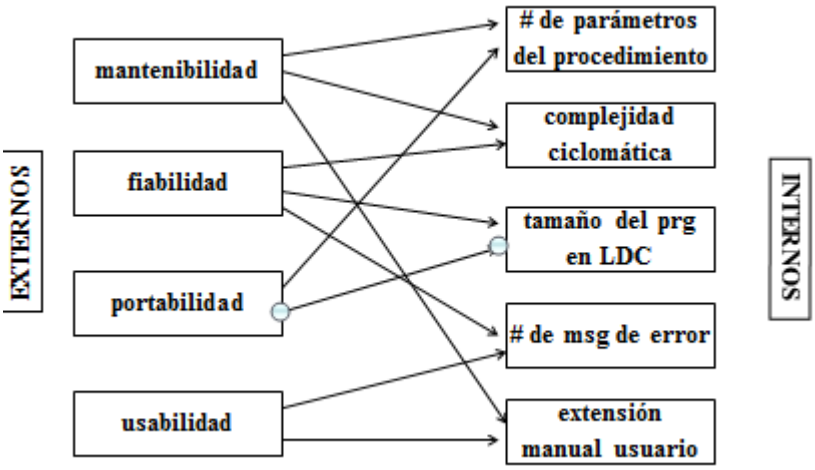
- Revisiones de la calidad: el objetivo de las revisiones de la calidad es el de detectar errores e inconsistencias, y obvio comunicarlas. A la hora de hablar de las revisiones de la calidad podremos encontrar tres tipos para realizarlas, ellos son:

* Inspecciones de diseño o programas: en este tipo de revisión lo que se busca es el detectar errores finos en los requerimientos, el diseño o el código. Aquí la revisión se realización mediante una lista de verificación de posibles errores.
* Revisiones de progreso: en la revisión de progreso se provee información para administrar el progreso completo del proyecto, esta es una revisión tanto del proceso como del producto y se refiere a los costos, plan y calendarización.
* Revisiones de calidad: y por último, las revisiones de calidad consisten en el llevar a cabo un análisis técnico de los componentes del producto o documentación para encontrar diferencias entre la especificación y el diseño del componente, código y documentación, y para asegurar que se están siguiendo los estándares de calidad definidos.

- Medición de métricas de software: para cada atributo elegido del proceso del producto, se determina un valor a alcanzar y luego se lo compara con lo obtenido. La métrica de software es cualquier tipo de medida relacionada con el sistema, el proceso o la documentación del software.

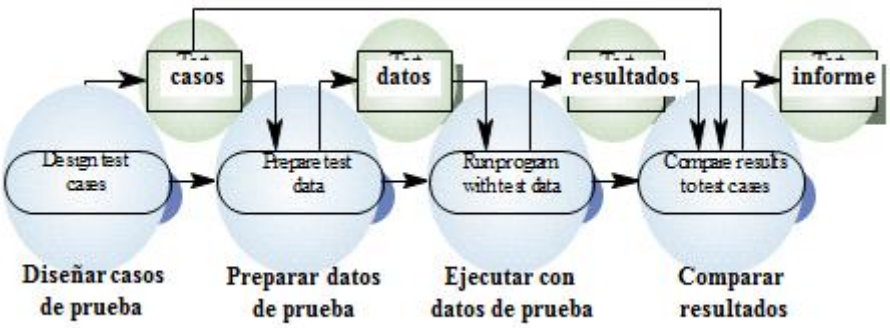
* Medida: la medida es el número de errores descubiertos en la revisión de un módulo. Esta proporcionará una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos de los atributos de un proceso o producto.
* Medición: la medición es la recopilación de medidas (o sea de los errores encontrados). Esta es el acto de determinar una medida, esta permitirá además a planificar la planificación del software, y de poder evaluar la calidad del producto que se produce.

* Métrica: la métrica es el número medio de errores encontrados por la revisión o por persona/hora. Según la IEEE es una medida cuantitativa del grado en que un sistema posee un atributo dado.
- Relación entre los atributos: medir los atributos de calidad del software de forma directa (los atributos externos) no configura una tarea sencilla, razón por lo cual se suele recurrir a atributos internos del proyecto para lograrlo. La relación entre los atributos internos y externos no es fácil de determinar, y para hacerlo se suele recurrir a supuestos basados en la experiencia de la persona que lo realiza.



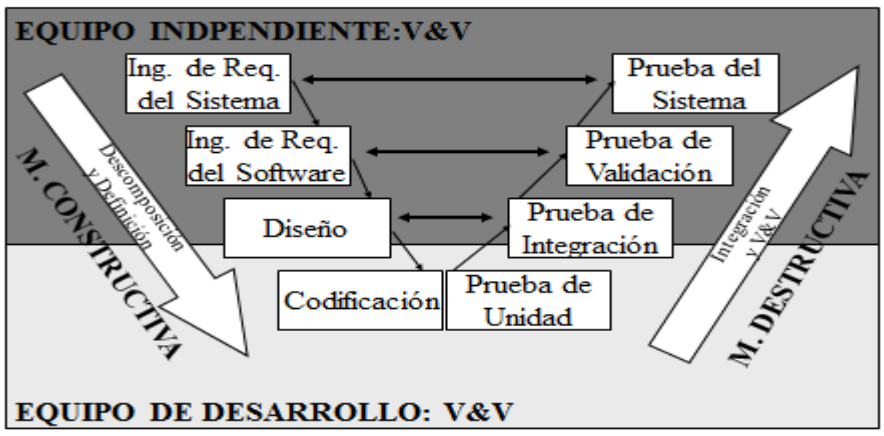
.10. Pruebas del software:
Según el autor Glen Myers la prueba de software es el proceso de ejecución de un programa con la intención de descubrir errores. Este autor estableció que un buen caso de prueba es aquél con alta probabilidad de descubrir un error no encontrado hasta el momento. El caso de prueba tendrá éxito sí y solo sí se descubren errores.
Como supuesto básico tendremos que partir de la premisa que "la ausencia de errores no es evidencia de que éstos no existan", con esto queremos decir que si con la prueba no encontramos errores, no se pueden afirmar la inexistencia de los mismos.
- Principios: los principios de las pruebas del software serán: 1) Deberá existir trazabilidad de ida y vuelta entre los requerimientos y los casos de prueba, 2) se deberá planificar la prueba antes de construir el sistema, 3) Las pruebas deberán ir de lo pequeño a lo grande, 4) Las pruebas deberán ser realizadas por un equipo independiente al de las personas que desarrollan el software.

- Pruebas de defectos (PD): el objetivo de estas pruebas será el de exponer los defectos latentes de un sistema de software antes de que este sea entregado. Una prueba de defectos exitosa será aquella en la cual el sistema se desempeña de forma incorrecta, dando lugar a un defecto del sistema, esta prueba demostrará la existencia de fallas en el software.



- Técnicas de prueba: al hablar de técnicas de prueba podremos encontrar a:
*** Pruebas de caja negra:** estas pruebas se realizan ejecutando la aplicación, son realizadas a partir de la interfaz de usuario, y con ellas se pretende demostrar que las entradas y salidas del programa son correctas. En estas pruebas se aplican pruebas de bajo nivel del software, estas son aplicadas en las primeras fases de la prueba.
En las pruebas de caja negra se parte de requerimientos como los ve el usuario, estos requerimientos se derivan de su especificación, y los casos de prueba generados ayudan a encontrar: funciones incorrectas o ausentes, errores en la interfaz, errores en acceso a base de datos, errores de desempeño, etc.
*** Pruebas de caja blanca:** en estas pruebas se analiza la lógica interna de la aplicación, aquí lo que se pretende demostrar es que los componentes internos de la aplicación se comportan de forma adecuada.
Estas se aplican a las pruebas de alto nivel (desde el punto de vista del usuario), se aplican en las fases más tardías de la prueba.
Los casos de prueba de caja blanca que se generan en esta técnica de pruebas deben garantizar: que se ejecutan todos los posibles caminos, que se ejecutan todas las alternativas, que se ejecutan todos los loops, y que se ejecutan todas las estructuras internas de datos.
*** Pruebas de componentes:** en las pruebas de componentes se prueban las funciones o grupos de métodos que forman parte de un módulo u objeto.

*** Pruebas de integración:** mientras que en las pruebas de integración los componentes se integran para formar un subsistema o un sistema completo, y por ende se probará el subsistema o sistema completo. Las pruebas de integración se realizan una vez que se probaron todos los componentes de forma individual, estas se desarrollan a partir de la especificación del sistema, la idea aquí es el detectar la fuente del error para así lograr un enfoque incremental.
*** Pruebas ascendentes y descendentes:** a) Ascendentes: en estas pruebas los componentes de bajo nivel se integran y se prueban antes de que se desarrollen los componentes de alto nivel; b) Descendentes: mientras que n las pruebas descendentes primeramente se integran los componentes de alto nivel, y se prueban antes de que se complete su diseño e implementación.
*** Pruebas de esfuerzo:** una vez que el sistema está integrado de forma completa, se pueden probar sus propiedades emergentes, como ser el desempeño y fiabilidad. En estas pruebas se pretende probar el comportamiento erróneo si la carga es alta, lo que se busca es ver que el programa "no explote".
<u>- Niveles de prueba:</u> considero importante decir que los diferentes tipos de pruebas que hemos mencionado se realizan en diferentes niveles, en base a esto explicaremos que:
* Prueba de Unidad: aquí se prueban los componentes de forma individual, las pruebas se basan en los requerimientos de la unidad.
* Prueba de Integración: en estas se prueba un conjunto de componentes, las pruebas se basan en los requerimientos del conjunto.
* Prueba de Validación: aquí se prueba toda la aplicación de software, estas se basan en los requerimientos del software.
* Pruebas del sistema: aquí se prueba toda la aplicación en el contexto del sistema. Estas pruebas se basan en los requerimientos del sistema.



.10.1. Administración de la configuración – SCM:

.10.1.1. ¿Qué es SCM?:

A la hora de determinar que es SCM podremos encontrar diferentes opiniones, algunas de ellas son:

a) Babich: este autor estableció que SCM es el arte de identificar, organizar y controlar las modificaciones al software que está siendo construido por un equipo de programadores, con el objetivo de maximizar la productividad, minimizando los errores.

b) ISO/IEC 12207: por otro lado, la normativa ISO/IEC 12207 ha establecido que SCM es el proceso de aplicar procedimientos técnicos y administrativos a lo lardo del ciclo de vida del proyecto para poder identificar y definir las piezas de software.

c) Penderson: este autor estableció que el propósito principal de SCM es asegurar un control ordenado de los productos de software producidos en el proceso de desarrollo de software, y el de proporcionar un mecanismo efectivo para incorporar cambios en el software durante su desarrollo y su uso operacional.

d) Pressman: SCM es un conjunto de actividades desarrolladas durante el ciclo de vida del proyecto, para gestionar cambios a lo largo de todo el ciclo de vida (la SCM es una actividad de garantía de calidad del software).

e) Sommerville: SCM es el desarrollo y aplicación de estándares y procedimientos para administrar un producto evolutivo de sistemas.

.10.1.2. ¿Porqué surge la administración de configuración - SCM?

En base a problemas como:

* Actualizaciones simultáneas: situación en la cual dos personas han trabajado separadamente sobre el mismo objeto, luego ambos los suben al repositorio, eliminándose uno de los trabajos de una de las personas.
* Objetos compartidos: situación en la cual se encuentra un error en un objeto que comparten varios desarrolladores y uno de ello no es comunicado.
* Versiones: situación en la cual se desarrollan versiones evolutivas o versiones variantes, dándose lugar a la propagación de errores entre ellas.

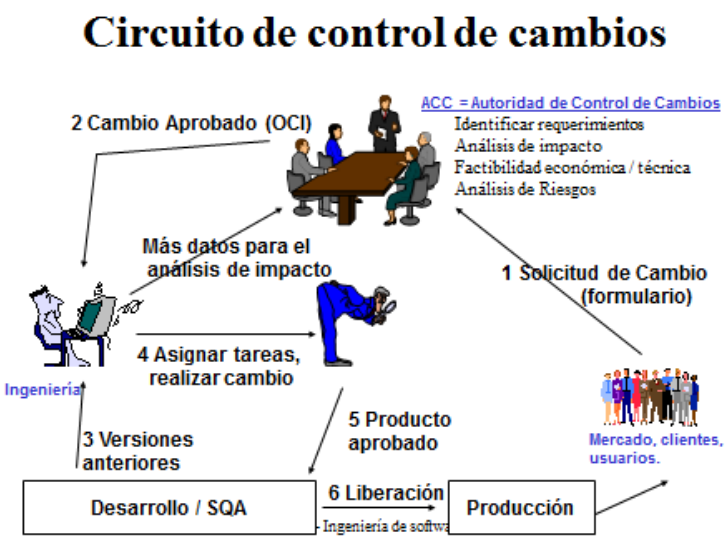
Es que se da lugar al surgimiento da la administración de configuración.

Los procedimientos de la SCM definirán el cómo se deben registrar y procesar los cambios propuestos, además de el cómo se deben relacionar

los cambios con los componentes y métodos usados para identificar las versiones.

- Línea base: a menudo la SCM es realizada por la misma persona de SQA, el proceso es que los desarrolladores pasan su producto a SQA, una vez que SQA verifica su calidad, esto proceso lo pasan a SCM que es el responsable de controlar los cambios al software.
Algunas veces los sistemas controlados se denominan línea base (baselines) puesto que son el inicio para la evolución controlada.
- Elementos de configuración de software: existen tres tipos de elementos creados durante el desarrollo del software, estos tres tipos son: a) Programas (código fuente y ejecutables), b) Documentos (técnicos, administrativos y de usuario), c) Estructuras de datos (DB, archivos, etc.).
- Elementos de la configuración de Software: es muy importante la asignación de nombres a los documentos, siempre existirán relaciones entre los mismos, por ejemplo: entre documentos de diseño y los programas. Los documentos de configuración requieren que sean identificados en forma única.
En general es posible definir ECS compuesto que agrupen varios ECS simples, y otros ECS compuestos. Como por ejemplo: un diagrama de clases.

.10.2. La base de datos de la configuración:
La base de datos de la configuración es utilizada para registrar toda la información relevante relacionada con las configuraciones. Las funciones principales de la base de datos de configuración son las de ayudar a la evaluación del impacto de los cambios en el sistema, y de proveer información de la administración acerca del proceso de SCM.
- Organización del repositorio: el repositorio se organiza en varios elementos:
a) Línea base: un punto de referencia en el desarrollo del software quedará marcado por la aprobación del elemento de configuración del software, y mediante la revisión técnica formal, y su envío al área de producción.
b) Repositorio: el repositorio es una estructura de directorios en disco, en la cual se almacenan los elementos componentes de software producidos a lo largo del proyecto. El objetivo de la administración del cambio será el de proporcionar mecanismos para evitar el caos ante cambios no controlados.



- Check - out: por intermedio del check – out se busca obtener una copia de algún objeto del repositorio, sobre el que trabajará la persona. El objeto del repositorio quedará bloqueado y nadie más podrá realizar modificaciones sobre éste, hasta que el mismo no sea liberado.
- Check – in: por intermedio del check – in se busca ingresar un nuevo objeto al repositorio, o una copia de un objeto que se encuentra bloqueado por un check – out. A partir de este momento el objeto quedaría librado para cualquier modificación.
- Identificación de versiones: para realizar una correcta identificación de versiones, se sugiere seguir los siguientes lineamientos:

* Numeración de las versiones: al componente se le asigna un # de versión explícito y único.
* Identificación basada en atributos: aquí se identifica cada componente por combinación de un conjunto de nombres y atributos.
* Identificación orientada al cambio: aquí se identifica al componente asociando por los cambios implementados en el componente.

- Administración de las liberaciones: una liberación es una versión del sistema que se distribuye a los clientes. Aquí se debe decidir cuando se realiza, se debe determinar la creación de la entrega y el medio de distribución como la documentación asociada (además el código ejecutable debe incluir: archivos de configuración y de datos, programa de instalación, documentación electrónica, etc.).
- Generación de informes: los informes van a ayudar a evitar el síndrome de “la mano izquierda no sabe lo que hace la mano derecha”, que es muy común cuando en un proyecto se encuentran muchas personas involucradas

en una misma tarea. Un ejemplo de ineficiencia es: dos programadores modifican secuencialmente un ECS de forma tal que lo objetivos quedarán como contradictorios.

c) Asignación de permisos: en base a este punto diremos que siempre tendremos que tener presente que el acceso al repositorio tendrá que estar controlado por un mecanismos de permisos, ya que no sería correctamente apropiado que cualquier perfil de la empresa pueda acceder a cualquier instancia del proyecto de software.