# INGENIERIA DE SOFTWARE 2 – RESUMEN TEORIA 1

- **IEEE 1362-1998**

  Ofrece un formato y contenidos para la confección de las descripciones de sistema en los desarrollos y modificaciones del sistema.

  Este estándar no especifica técnicas exactas, sino que proporciona las líneas generales que deben respetarse. Es una guía de referencia.

  Las partes esenciales de un ConOps son:

  Punto 3: Descripción del sistema existente.

  Punto 4: Descripción del sistema propuesto.

  El estándar identifica los elementos que al menos debe incluir una descripción del sistema. El usuario puede incorporar otros elementos, agrandando clausulas y sub-clausulas.

- **SRS - IEEE 830**

  En los SRS se debe evitar incluir requerimientos de diseño o de proyecto. Los aspectos básicos que debe cubrir son: Funcionalidad. Descripción de lo que el software debe hacer. Interfaces externas. Como debe interactuar el software con las personas, el hardware o con otros sistemas. Rendimiento. Indicación de la velocidad, disponibilidad, tiempos de respuesta, de recuperación, de determinadas funciones. Atributos. Consideraciones de portabilidad, corrección, mantenibilidad, seguridad, etc. Restricciones de diseño en la implementación, entre otras.

  - Alcance: Brindar una colección de buenas prácticas para escribir especificaciones de requerimientos de software (SRS). Se describen los contenidos y las cualidades de una buena especificación de requerimientos.
  - Naturaleza: El SRS es una especificación para un producto de software particular. El SRS es escrito por uno o más representantes del equipo de desarrollo y uno o más representantes de la parte cliente o ambos.
  - Ambiente: El software puede contener toda la funcionalidad del proyecto o puede ser parte de un sistema más grande. En el último caso habrá un SRS que declarará las interfaces entre el sistema y su software desarrollado, y pondrá qué función externa y requerimientos de funcionalidad tiene con el software desarrollado.
  - Características:
    - Correcto: Un SRS es correcto si, y sólo si, cada requisito declarado se encuentra en el software.
    - No ambiguo: Un SRS es inequívoco si, y sólo si, cada requisito declarado tiene sólo una interpretación.
    - Completo: Un SRS está completo si, y sólo si, se reconoce cualquier requisito externo impuesto por una especificación del sistema.
    - Consistente: La consistencia se refiere a la consistencia interior. Si un SRS no está de acuerdo con algún documento del nivel superior, como una especificación de requerimientos de sistema, entonces no es consistente.
    - Priorizado: Un SRS es priorizado por la importancia de sus requerimientos particulares.
    - Comprobable: Un SRS es comprobable si, y sólo si, cada requisito declarado es comprobable. Un requisito es comprobable si, y sólo si, existe algún proceso con que una persona o máquina puede verificar que el producto del software reúne el requisito. En general cualquier requisito ambiguo no es comprobable.
    - Modificable: Un SRS es modificable si, y sólo si, su estructura y estilo son tales que puede hacerse cualquier cambio a los requerimientos fácilmente, completamente y de forma consistente mientras conserva la estructura y estilo.
    - Trazabilidad: Claridad del origen de cada requerimiento.
  - Consideraciones para un buen SRS:
    - Preparación conjunta del SRS: El SRS se debe preparar en conjunto con las partes intervinientes para lograr un buen acuerdo entre las partes.

- Evolución de SRS: El SRS debe evolucionar conjuntamente con el software, registrando los cambios, los responsables y aceptación de los mismos.
- Prototipos: El uso de prototipos se utiliza frecuentemente para la definición de requerimientos.
- Diseño incorporado en el SRS: El SRS puede incorporar los atributos o funciones externos al sistema, en particular las que describen el diseño para interactuar entre los subsistemas.

- <u>Partes de un SRS:</u>
  - Sección 1: Introducción
    - 1.1 Propósito: Se define el propósito del documento y se especifica a quién va dirigido el documento
    - 1.2 Alcance o ámbito del sistema: Se da un nombre al futuro sistema. Se explica lo que el sistema hará y lo que no hará. Se describen los beneficios, objetivos y metas que se espera alcanzar con el futuro sistema.
    - 1.3 Definiciones, siglas y abreviaciones: Glosario
    - 1.4 Referencias: Esta subdivisión debe proporcionar una lista completa de todas las referencias de los documentos mencionados o utilizados para escribir el SRS.
  - Sección 2: Descripción General
    - Esta sección del SRS debe describir los factores generales que afectan el producto y sus requerimientos. Esta sección no declara los requerimientos específicos. En cambio, mantiene una mención general de esos requerimientos que se definen en detalle en Sección 3 del SRS y los hacen más fáciles de entender.
    - 2.1 Perspectiva del producto: Si el producto es independiente y totalmente autónomo, debe declararse que así es. Si el SRS define un producto que es un componente de un sistema más grande entonces esta subdivisión debe relacionar los requerimientos de ese sistema más grande a la funcionalidad del software y debe identificar las interfaces entre ese sistema y el software.
    - 2.2 Funciones del sistema: Se debe presentar un resumen, a grandes rasgos, de las funciones del futuro sistema.
    - 2.3 Características del Usuario:
    - 2.4 Restricciones: a) Las interfaces: del Sistema, del Usuario, del Hardware; de las de Comunicaciones; b) Acceso y uso de la Memoria; c) Los requerimientos de adaptación del Sitio d) Políticas de la empresa e) Limitaciones del hardware f) Interfaces con otras aplicaciones g) Operaciones paralelas h) Funciones de auditoría i) Lenguaje(s) de programación. Bases de Datos. j) Protocolos de comunicación k) Req. de fiabilidad l) Consideraciones acerca de la seguridad.
    - 2.5 Suposiciones y dependencias: Se describen aquellos factores que, si cambian, pueden afectar a los requerimientos.
    - 2.6 Evoluciones previsibles del sistema: Se identifican requerimientos que serán implementados en futuras versiones.
  - Sección 3: Requerimientos específicos.
    - Debe contener todos los requerimientos del software a un nivel de detalle suficiente para permitirles a los diseñadores diseñar un sistema para satisfacer esos requerimientos, y a los auditores probar que el sistema satisface esos requerimientos. A lo largo de esta sección, cada requisito declarado debe ser externamente perceptible por los usuarios, operadores u otros sistemas externos.
    - 3.1 Requerimientos comunes de las interfaces.
    - 3.1.1 Interfaces de usuario: Describir los requerimientos del interfaz de usuario para el producto. Esto puede estar en la forma de descripciones del texto o pantallas del interfaz.
    - 3.1.2 Interfaces de hardware: Especificar las características lógicas para cada interfaz entre el producto y los componentes de hardware del sistema. Se incluirán características de configuración.

- 3.1.3 Interfaces de software: Indicar si hay que integrar el producto con otros productos de software. Para cada producto de software debe especificarse lo siguiente: Descripción del producto software utilizado, Propósito del interfaz, Definición del interfaz.
- 3.1.4 Interfaces de comunicación: Describir los requerimientos de interfaces de comunicación.
- 3.2 Requerimientos funcionales: Serán desarrollados utilizando Historias de Usuarios
- 3.3 Requerimientos no funcionales
- 3.3.1 Requerimientos de rendimiento: Especificación de los requerimientos relacionados con la carga que se espera que tenga que soportar el sistema. Por ejemplo, el número de terminales, el número esperado de usuarios simultáneamente conectados, etc. Todos estos requerimientos deben ser mensurables. Por ejemplo, indicando "el 95% de las transacciones deben realizarse en menos de 1 segundo", en lugar de "los operadores no deben esperar a que se complete la transacción".
- 3.3.2 Seguridad: Especificación de elementos que protegerán al software de accesos, usos y sabotajes maliciosos, así como de modificaciones o destrucciones maliciosas o accidentales. Los requerimientos pueden especificar: Empleo de técnicas criptográficas, etc.
- 3.3.4 Disponibilidad: Especificación de los factores de disponibilidad final exigidos al sistema. Normalmente expresados en % de tiempo en los que el software tiene que mostrar disponibilidad.
- 3.3.5 Mantenibilidad: Identificación del tipo de mantenimiento necesario del sistema. Especificación de quién debe realizar las tareas de mantenimiento, por ejemplo, usuarios o un desarrollador. Especificación de cuándo deben realizarse las tareas de mantenimiento. Por ejemplo, generación de estadísticas de acceso semanales y mensuales.
- 3.3.6 Portabilidad: Especificación de atributos que debe presentar el software para facilitar su traslado a otras plataformas u entornos. Pueden incluirse: Porcentaje de componentes dependientes del servidor.
- 3.4 Otros requerimientos: Cualquier otro requisito que no encaje en ninguna de las secciones anteriores. Por ejemplo: requerimientos culturales y políticos, requerimientos legales.

▪ Sección 4 – Apéndices
- Pueden contener todo tipo de información relevante para la SRS pero que, propiamente, no forme parte de la SRS. Por ejemplo: Casos de Uso

- **<u>GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE (GCS)</u>**

Es el proceso de identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos.
Es una actividad de autoprotección que se aplica durante el proceso del software. El resultado del proceso de Software se puede dividir en: Programas (códigos y ejecutables). Documentos. Datos.
El cambio se puede producir en cualquier momento, las actividades de la GCS sirven para: Identificar el cambio. Controlar el cambio. Garantizar que el cambio se implemente adecuadamente. Informar del cambio a todos aquellos que puedan estar afectados.

- <u>Línea Base:</u> es un concepto de GCS que nos ayuda a controlar los cambios.
  Definición de la IEEE: Una especificación o producto que se ha revisado formalmente y sobre el que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio.
- <u>Proceso de la GCS:</u>

        - 1. Identificaciones de los objetos: Nombre: cadena de caracteres sin ambigüedad. Descripción: lista de elementos de datos que identifican: Tipo de ECS (documento, código fuente, datos), Identificador del proyecto, información de la versión y/o cambio.
        - 2. Control de versiones: Permite al usuario especificar configuraciones alternativas del sistema mediante la selección de versiones adecuadas (por ejemplo, asociando atributos que la identifican). Combinación de procedimientos y herramientas para gestionar las versiones de los ECS.
        - 3. Control de cambios: A lo largo del proyecto los cambios son inevitables y el control es vital para el desarrollo del mismo. Combina los procedimientos humanos y las herramientas adecuadas para proporcionar un mecanismo para el control del cambio.
        - 4. Auditoría de la configuración: La identificación y el control de versiones y el control de cambio, ayudan al equipo de desarrollo de software a mantener un orden, pero sólo se garantiza hasta que se ha generado la orden de cambio. Cómo aseguramos que el cambio se ha realizado correctamente: Revisiones técnicas formales y auditorías de configuración.
        - 5. Generación de informes de estado de la configuración: Responde ¿Qué pasó? ¿Quién lo hizo? ¿Cuándo pasó? ¿Qué más se vio afectado? La generación de informes de estado de la configuración desempeña un papel vital en el éxito del proyecto.

- **GESTION DE PROYECTOS**
    - Proyecto: es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único.
        - Características:
            - Temporal: Tiene un comienzo y fin definido. El fin se alcanza cuando se han logrado los objetivos del proyecto o cuando queda claro que esos objetivos no serán o no podrán ser alcanzados.
            - Resultado: Productos, servicios o resultados únicos.
            - Elaboración gradual: Desarrollar en pasos e ir aumentando mediante incrementos.
    - Aspectos:
        - Personal (RRHH): Es el elemento más importante. El equipo de dirección del proyecto debe identificar a los interesados, determinar sus requisitos y expectativas y, gestionar su influencia en relación con los requisitos para asegurar un proyecto exitoso.
        - Producto: El producto de software es intangible. A veces es difícil ver el progreso del proyecto. Los proyectos pueden crear: Un producto o artículo producido, que es cuantificable, y que puede ser un elemento terminado o un componente. La capacidad de prestar un servicio como, por ejemplo, las funciones del negocio que respaldan la producción o la distribución. Un resultado
        - Procesos: Un proceso de software proporciona el marco de trabajo desde el cual se puede establecer un plan detallado para el desarrollo del software.
        - Proyecto: Los proyectos deben ser planeados y controlados para manejar su complejidad.
    - Manifestación de una mala gestión:
        - Incumplimiento de plazos.
        - Incremento de los costos.
        - Entrega de productos de mala calidad.
    - Elementos claves: Métricas. Estimaciones. Calendario temporal. Organización del personal. Análisis de riesgos. Seguimiento y control.

- **PLANIFICACION**

  La planificación especifica que debe hacerse, con qué recursos y en qué orden. Establece una secuencia operativa.

  ¿Por qué un software se retrasa? Fecha límite de entrega poco realista. Cambios en los requisitos. Subestimación de los recursos necesarios. Riesgos no considerados. Dificultades humanas. Falta de comunicación.

¿Qué hacer? Realizar una estimación detallada, determinando esfuerzo y duración. Establecer la funcionalidad critica. Realizar reuniones con el cliente.

- Planificación temporal: Es una actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto. La precisión de la planificación temporal es muy importante para no generar clientes insatisfechos, costos adicionales, reducción del impacto en el mercado, etc.
  - Perspectivas: Con fecha final establecida por el cliente (obligados a distribuir el esfuerzo dentro del plazo previsto). Con fecha final fijada por los desarrolladores (el esfuerzo se distribuye para conseguir un uso óptimo de los recursos y se define una fecha de fin luego de un cuidadoso análisis).
  - Principios Básicos:
    - Compartimentación: dividir el proyecto en actividades y tareas manejables.
    - Interdependencia: determinar la interdependencia de cada actividad o tarea. Algunas pueden realizarse en paralelo, otras necesitan de la terminación de una anterior.
    - Asignación de tiempo: es decir, una fecha inicial y final, además de los recursos.
    - Validación del esfuerzo: Evita la sobreasignación.
    - Responsabilidades definidas: Asignación de responsables a cada tarea.
    - Resultados definidos: Cada tarea debe tener un resultado.
    - Hitos definidos.
  - Conjunto de tareas: es una colección de tareas, hitos y entregas que se deben cumplir para completar el proyecto.
    - Tarea: Secuencia de acciones a realizar en un plazo determinado.
    - Tarea Crítica: Es aquella cuyo retraso genera un retraso en todo el proyecto.
    - Hito: Es “algo” que se espera que esté hecho para alguna fecha, como por ejemplo, un módulo testeado o una característica del funcionamiento, un logro que sea objetivo, fácil de evaluar y notable.
    - Entregas: Componente (fuentes, documentación, etc.) formalizados a través de un documento.
  - Tarea: Puede describirse con cuatro parámetros:
    - Precursor: evento o conjunto de eventos que deben ocurrir antes de que la actividad pueda comenzar.
    - Duración: cantidad de tiempo necesaria para completar la actividad.
    - Fecha de entrega: fecha para la cual la actividad debe estar completada.
    - Punto final: Hito o componente listo.
  - Red de tareas: Es una representación gráfica del flujo de las tareas desde el inicio hasta el fin de un proyecto. En algunos casos los conjuntos de tareas permiten realizar algunas actividades en paralelo. Representan la secuencia de las tareas y su interdependencia.
  - Calendarización: Separar todo el trabajo de un proyecto en actividades complementarias y considerar el tiempo requerido para completar dichas actividades.
  - Métodos de planificación temporal:
    - PERT (Técnica de evaluación y revisión de programas): Creado para proyectos del programa de defensa del gobierno norteamericano entre 1958 y 1959. Se utiliza para controlar la ejecución de proyectos con gran número de actividades que implican investigación, desarrollo y pruebas. Red de tareas (Fechas tempranas y tardías. Camino crítico)
    - CPM (Método del camino crítico): Desarrollado para dos empresas americanas entre 1956 y 1958. Se utiliza en proyectos en los que hay poca incertidumbre en las estimaciones. Diagrama de barras que muestra las tareas del proyecto (Tiempo de inicio temprano y tardío. Las barras representan la duración de la actividad. Los “ * ” el camino crítico. Los “ - ” que no son parte del camino crítico. Las “ F ” tiempo flotante).

- Gantt: Representación gráfica de las tareas sobre una escala de tiempos. Las tareas se representan en forma de barra sobre dicha escala manteniendo la relación de proporcionalidad entre sus duraciones y su representación gráfica, y su posición respecto del punto origen del proyecto.
- PERT+CPM: Actualmente se ha tomado lo mejor de ambos métodos y se han vuelto uno solo, conocido como Método del Camino Crítico. Establecer lista de tareas. Fijar dependencia entre tareas y duración. Construir la red. Numerar los nodos. Calcular la fecha temprana y tardía de cada nodo (Tei = Fecha temprana del nodo i. Tai = Fecha tardía del nodo i). Calcular el camino crítico que une las tareas críticas ==> Tei = Tai.
    - Fechas Tempranas: TeJ = TeI + tIJ. Donde TeJ = fecha más temprana del nodo destino. TeI = fecha más temprana del nodo origen. tIJ = duración de la tarea desde el nodo I hasta el nodo J. Si hay más de un camino ... Max (TeJ1, TeJ2..)
    - Fechas Tardías: TaI = TaJ – tIJ. Donde TaI = fecha más tardía del nodo origen. TaJ = fecha más tardía del nodo destino. tIJ = duración de la tarea desde el nodo I hasta el nodo J. Si hay más de un camino ... Min (TaJ1, TaJ2..)
    - Margen Total: Mt = TaJ – TeI – tIJ. Donde TaJ = fecha tardía del nodo destino. TeI = fecha temprana del nodo origen. tIJ = duración de la tarea desde el nodo I hasta el nodo J.

- Seguimiento y control de proyectos: Realizar reuniones periódicas para informar progresos y problemas. Evaluar las revisiones. Controlar que los "hitos" del proyecto se hayan alcanzado en la fecha programada. Comparar fecha estimada de inicio y real. Análisis del valor ganado (medida de progreso). Permite evaluar cuantitativamente el % de realización.

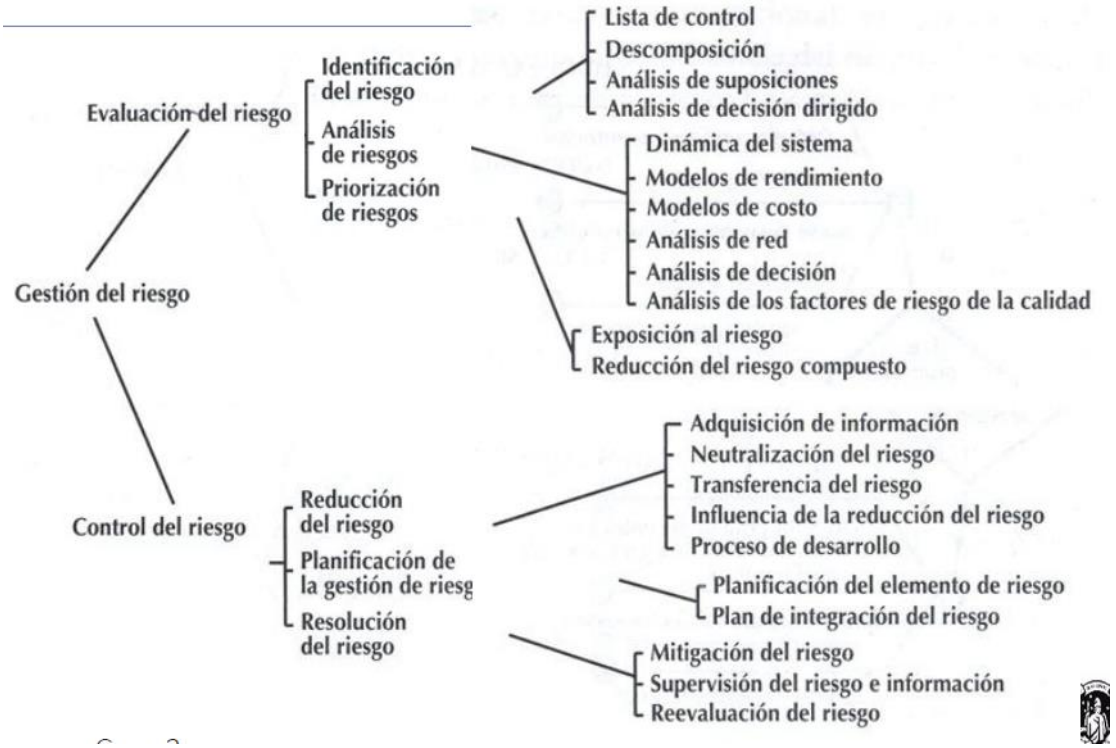
- Planificación organizativa: El personal que trabaja en una organización de software es el activo más grande, representa el capital intelectual. Una mala administración del personal es uno de los factores principales para el fracaso de los proyectos.
    - Participantes: Gestores ejecutivos (definen los temas empresariales). Gestores técnicos (planifican, motivan, organizan y controla a los profesionales). Profesionales (aportan habilidades técnicas). Clientes (especifican los requerimientos). Usuarios finales (interactúan con el software).
    - Líderes de equipo:
        - Modelo MOI de Liderazgo:
            - Motivación al personal - Habilidades para alentar al personal técnico para producir a su máxima capacidad.
            - Organización del equipo - Habilidades para modelar procesos que peritan que el concepto inicial se traduzca en un producto final.
            - Incentivación de Ideas e Innovación. - Habilidad para alentar a las personas a crear y sentirse creativas.
        - Rasgo de un líder eficaz:
            - Resolución de problemas - Diagnosticar los conflictos técnicos y organizativos, estructurar una solución sistemática, aplica lecciones aprendidas, flexible.
            - Identidad administrativa - Asumir el control cuando es necesario.
            - Logro – recompensar las iniciativas
            - Influencia y construcción del equipo - Comprender señales verbales y no verbales y reaccionar ante las necesidades.
    - Estructura de proyectos:
        - Proyecto: Integración de un equipo de personas que llevan a cabo el proyecto de principio a fin.
        - Funcional: Equipos distintos de personas que realizan cada fase del proyecto.

        - Matricial: Cada proyecto de desarrollo tiene un administrador. Cada persona trabaja en uno o más proyectos bajo la supervisión del administrador correspondiente.
    - Estructura del grupo: La "mejor" estructura de equipo depende del estilo de gestión de una organización, el número de personas que compondrá el equipo, sus niveles de preparación y la dificultad general del problema. Además, depende de su naturaleza y del producto.
    - Organigramas de equipos genéricos:
        - Descentralizado democrático (DD): Este equipo no tiene un jefe permanente. Se nombran coordinadores de tareas a corto plazo y se sustituyen por otros para diferentes tareas. Las decisiones se toman por consenso. La comunicación entre los miembros del equipo es horizontal.
        - Descentralizado controlado (DC): Este equipo tiene un jefe definido que coordina tareas específicas y jefes secundarios que tienen responsabilidades sobre subtareas. La resolución de problemas sigue siendo una actividad del grupo, pero la implementación de soluciones se reparte entre subgrupos por el jefe de equipo. La comunicación entre subgrupos e individuos es horizontal. También hay comunicación vertical a lo largo de la jerarquía de control.
        - Centralizado controlado (CC): El jefe del equipo se encarga de la resolución de problemas a alto nivel y la coordinación interna del equipo. La comunicación entre el jefe y los miembros del equipo es vertical.
- <u>El equipo de software:</u> Factores a considerar cuando se planea una estructura de equipo
        - 1.Dificultad de problema a resolver
        - 2.Tamaño de programa resultante
        - 3.Tiempo que el equipo permanecerá unido
        - 4.Grado en el que puede dividirse en módulos el problema
        - 5.Calidad y confiabilidad requerida por el sistema a construir
        - 6.Rigidez de la fecha de entrega
        - 7.Grado de sociabilidad requerido en para el proyecto
    - Gestión de Grupos: Los grupos de desarrollo de software deberían ser pequeños y cohesivos.
        - Ventajas de los grupos cohesivos: Los intereses del grupo son más importantes que los personales. Se pueden desarrollar estándares por consenso. Se fomenta el aprendizaje de unos con otros. Se garantiza la continuidad aún si un miembro abandona el equipo. Los programas son una "propiedad" del grupo.
        - Desventajas de los grupos cohesivos: Resistencia al cambio por un liderazgo externo a los miembros del grupo. Decisiones por mayoría sin estudiar alternativas.
    - Comunicación Grupal: Las comunicaciones en un grupo se ven influenciadas por factores como: status de los miembros del grupo, tamaño del grupo, composición de hombres y mujeres, personalidades y canales de comunicación disponible. Los programadores pueden mejorar la productividad si cuentan con un entorno de trabajo provisto con recursos necesarios y áreas de comunicación adecuadas.

- **<u>RIESGOS</u>**

Un riesgo es un evento no deseado que tiene consecuencias negativas.
Los gerentes deben determinar si pueden presentarse eventos no deseados durante el desarrollo o el mantenimiento, y hacer planes para evitar estos eventos, o, si son inevitables, minimizar sus consecuencias negativas.
El riesgo concierne lo que ocurrirá en el futuro. A como afectarán los cambios al desarrollo. A las elecciones.
-Peter Drucker define que "Mientras es inútil intentar eliminar el riesgo y cuestionable poder minimizarlo, es esencial que los riesgos que se tomen sean los adecuados".
Estrategias de riesgos: Reactivas (reaccionar ante el problema y "gestionar la crisis". Proactivas (tener estrategias de tratamiento).

- Según Pfleeger:

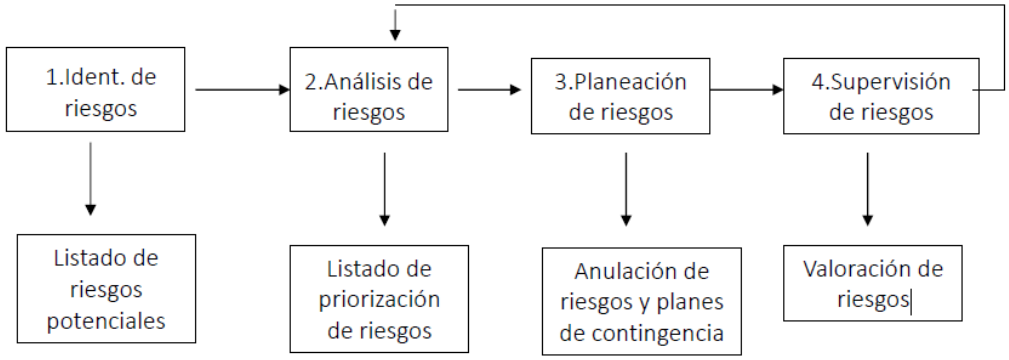



- Según Boehm:
  - Deficiencias del personal.
  - Cronogramas y presupuestos no realistas.
  - Desarrollo de funciones de software incorrectas.
  - Desarrollo de interfaces de usuario incorrectas.
  - Expectativas imposibles de satisfacer.
  - Corriente incesante de cambios a los requerimientos.
  - Deficiencias en tareas ejecutadas externamente.
  - Deficiencias del funcionamiento en tiempo real.
- Riesgos de software: El riesgo siempre implica dos características:
  - Incertidumbre: el acontecimiento que caracteriza al riesgo puede o no puede ocurrir no hay riesgos de un 100 % de probabilidad.
  - Pérdida: si el riesgo se convierte en una realidad, ocurrirán consecuencias no deseadas o pérdidas.

  Al analizar los riesgos, es importante cuantificar el nivel de incertidumbre y el grado de pérdidas asociado con cada riesgo.
- Categorización (clasificación):
  - Del proyecto: Amenazan el plan del proyecto. Identifican los problemas potenciales de presupuesto, planificación temporal, personal, recursos, cliente y requisitos.
  - Del producto: Afectan la calidad o rendimiento del software que se está desarrollando.
  - Del negocio: Afectan a la organización que desarrolla o suministra el software.
- Clasificación por categorías:
  - Riesgos conocidos: son todos aquellos que se pueden descubrir después de una cuidadosa evaluación del proyecto. Ejemplo: fechas de entrega poco realistas.
  - Riesgos predecibles: se extrapolan de la experiencia en proyectos anteriores. Ejemplo: mala comunicación con el cliente.
  - Riesgos: impredecibles pueden ocurrir, pero son extremadamente difíciles de identificar por adelantado.
- Tipos:
  - Riesgos genéricos: son una amenaza potencial para todos los proyectos. Ejemplo: entender mal los requerimientos.

- Riesgos específicos: sólo los pueden identificar los que tienen una clara visión de la tecnología, el personal y el entorno específico del proyecto en cuestión. Ejemplo: no contar con equipamiento específico.

- Proceso de Gestión de Riesgos




1- Identificación de riesgos:
Enumerar los "verdaderos riesgos".
Elaborar una "lista de comprobación de elementos de riesgo" para estimar el impacto del riesgo.
La actividad puede llevarse a cabo utilizando un enfoque de tormenta de ideas o basarse en la experiencia.

- Otra clasificación:
  - Riesgos de tecnología. Ej.: tiempos de respuesta inaccesibles.
  - Riesgos de personas. Ej.: no tienen habilidades requeridas
  - Riesgos organizacionales. Ej.: reducción en el presupuesto
  - Riesgos de herramientas. Ej.: CASE generan código ineficiente
  - Riesgos de requerimientos. Ej.: cambios en los requerimientos
  - Riesgos de estimación. Ej.: de tiempo, de tamaño, etc.
- Características identificatorias de los Riesgos: Existe una pérdida asociada con el evento (tiempo, calidad, etc.) → IMPACTO
  Probabilidad de que el evento pueda ocurrir → PROBABILIDAD=1 → PROBLEMA
  Grado en que se puede cambiar el resultado → CONTROL

2- Análisis de riesgos:
Se considera por separado cada riesgo identificado y se decide la probabilidad y el impacto.
Se construye la tabla de riesgos:

| Riesgos | **Categoría** | **Probabilidad** | **Impacto** |
|---|---|---|---|
| **El cliente cambiará los requisitos** | | | |
| **Falta de formación en las herramientas** | | | |

- Establecer una escala que refleje la PROBABILIDAD observada de un riesgo:

Bastante improbable → < 10%
Improbable → 10-25%
Moderado → 25-50%
Probable → 50-75%
Bastante probable → >75%

- Estimar el IMPACTO (depende de la naturaleza del riesgo, del alcance y de la duración):

Catastrófico → cancelación del proyecto.
Serio → reducción de rendimiento, retrasos en la entrega, excesos importantes en costo.
Tolerable → reducciones mínimas de rendimiento, posibles retrasos, exceso en costo.
Insignificante → incidencia mínima en el desarrollo.

- Generación de la tabla: 1ra columna se listan todos los riesgos en desorden. 2da columna se pone la categoría del riesgo. 3ra columna se pone la probabilidad estimada del riesgo. Puede

ser estimada por consenso, o individualmente y sacar un promedio. 4ta columna se pone el impacto del riesgo. Luego de todo esto se ordena la lista por probabilidad e impacto y se traza una línea de corte. Los riesgos que queden encima de la línea serán los que se les preste atención. Los que queden debajo de la línea serán reevaluados y tendrán una prioridad de segundo orden.

Un factor de riesgo que tenga gran impacto, pero poca probabilidad de que ocurra, no debería absorber un tiempo significativo.

Los riesgos de gran impacto con una probabilidad de moderada a alta y los riesgos de poco impacto, pero con gran probabilidad deberían tomarse en cuenta.

3- Planeación:

Dichas estrategias a seguir son las siguientes:

- Evitar el riesgo: el sistema se diseña de modo que no pueda ocurrir el evento.
- Minimizar el riesgo: la probabilidad que el riesgo se presente se reduce.
- Plan de contingencia: se está preparado para lo peor. Se acepta la aparición del riesgo y es tratado de manera de minimizar las consecuencias.

Para la toma de decisión acerca del tratado de riesgos, se debe tener en cuenta el costo de la aplicación de las estrategias

INFLUENCIA = (EXPOSICION antes – EXPOSICION después)
COSTO de reducción

EXPOSICION: Probabilidad que ocurra x Costo del Proyecto si sucede el riesgo.

Para que se justifiquen las acciones de reducción del riesgo el valor de INFLUENCIA debe ser alto.

4- Supervisión:

- Evaluar si ha cambiado la probabilidad de cada riesgo
- Evaluar la efectividad de las estrategias propuestas.
- Detectar la ocurrencia de un riesgo que fue previsto
- Asegurar que se están cumpliendo los pasos definidos para cada riesgo
- Recopilar información para el futuro
- Determinar si existen nuevos riesgos
- Reevaluar periódicamente los riesgos

Los riesgos deben monitorizarse comúnmente en todas las etapas del proyecto. En cada revisión administrativa, es necesario reflexionar y estudiar cada uno de los riesgos clave por separado. También hay que decidir si es más o menos probable que surja el riesgo, y si cambiaron la gravedad y las consecuencias del riesgo.

- **DISEÑO DE LA INTERFAZ DEL USUARIO**

Es la categoría de diseño que crea un medio de comunicación entre el hombre y la máquina.

Con un conjunto de principios, crea un formato de pantalla.

De un buen diseño depende en parte el éxito de un sistema.

Una interfaz difícil de utilizar provoca que los usuarios cometan errores o incluso que se rehúsen a utilizar el sistema.

Partimos de la base de que personas diferentes pueden tener estilos diferentes de percepción, comprensión y trabajo.

La interfaz debe contribuir a que el usuario consiga un rápido acceso al contenido de sistemas complejos, sin pérdida de la comprensión mientras se desplaza a través de la información.

- Conceptos iniciales:
  - Variedad de tecnologías que deben adaptarse al usuario: Hipertexto, sonido, presentaciones tridimensionales, video, realidad virtual, etc.
  - Configuraciones de hardware: Teclado, mouse, dispositivos de presentación gráfica, lápices, anteojos de realidad virtual, reconocimiento de voz, etc.
  - Variedad de Dispositivos: PC, equipos específicos, celulares, televisores, etc.
- Análisis y modelos de interfaces:

- El encargado del software establece un **modelo de usuario**: establece el perfil de los usuarios finales del sistema. Habitualmente los diseñadores y desarrolladores piensan con frecuencia en los usuarios finales. Sin embargo, en ausencia de un modelo mental fuerte de usuarios específicos, lo sustituyen con los modelos de diseñadores y desarrolladores (modelo de diseño). Esto no es centrarse en el usuario, sino en uno mismo.
  Para construir una interfaz de usuario eficaz, todo diseño debe comenzar con la comprensión de los usuarios que se busca, lo que incluye los perfiles de edad, género, condiciones físicas, educación, antecedentes culturales o étnicos, motivación, metas y personalidad.
- El ingeniero de software crea un **modelo de diseño**, este debe haberse desarrollado de manera que incluya la información contenida en el modelo del usuario, y el modelo de la implementación debe reflejar de manera exacta la información sintáctica y semántica de la interfaz.
- El usuario final desarrolla una **modelo mental** o percepción del sistema, es la imagen del sistema que los usuarios finales llevan en la cabeza. Un usuario que entienda bien los procesadores de texto, pero que haya trabajado con el procesador específico una sola vez, tal vez esté más preparado para hacer una descripción más completa de su funcionamiento que el principiante que haya pasado semanas tratando de entender el sistema.
- Los implementadores del sistema crean un **modelo de implementación**, esta combina la manifestación externa del sistema basado en computadora (la vista y sensación de la interfaz) con toda la información de apoyo (libros, manuales, videos, archivos de ayuda, etc.) que describe la sintaxis y semántica de la interfaz.

- <u>Diseño de la interfaz del usuario:</u>
  - Análisis y modelado: Definir objetos y acciones de la interfaz (operaciones) con el uso de la información desarrollada en el análisis de la interfaz
  - Diseño de la interfaz: Definir eventos (acciones del usuario) que harán que cambie el estado de la interfaz de usuario. Hay que modelar este comportamiento.
  - Construcción de la interfaz: Ilustrar cada estado de la interfaz como lo vería en la realidad el usuario final.
  - Validación: Indicar cómo interpreta el usuario el estado del sistema a partir de la información provista a través de la interfaz.
- <u>Diseño de experiencias de usuario (Udx):</u> es un conjunto de métodos aplicados al proceso de diseño que buscan satisfacer las necesidades del cliente y proporciona una buena experiencia a los usuarios destinatarios (Allanwood & Beare 2015).
  - Fuentes: Se toma información de diferentes fuentes que permiten estudiar al usuario:
    - Encuestas. Información de ventas. Información de mercadotecnia. Información de charlas de apoyo al usuario.
  - Información para crear el perfil del usuario:
    - Franja de edad. Etnia. Experiencia. Género. Nivel de ingresos. Idioma. Nivel de Estudios. Localización. Ocupación o profesión. Religión.
- <u>Estilos de interfaces:</u>
  - Interfaz de comandos: Es la interfaz más elemental. Solo se interactúa con texto. Generalmente se interactúa desde una línea de comando de una consola de una aplicación en particular con el teclado
    Características: Poderoso y Flexible. Administración de errores pobre. Difícil de aprender.
  - Interfaz de menú simple: Se presentan un conjunto de opciones, que pueden ser seleccionadas por el usuario. Solo se interactúa con los caracteres indicados.
    Características: Evita errores del usuario. Lento para usuarios experimentados.
  - Interfaz gráfica de usuarios: Se caracterizan por la utilización de todo tipo de recursos visuales para la representación e interacción con el usuario.
    Ventajas: Son relativamente fáciles de aprender y utilizar. Los usuarios cuentan con pantallas múltiples (ventanas) para interactuar con el sistema. Se tiene acceso inmediato a cualquier punto de la pantalla.

- Interfaz por reconocimiento de voz.
- Interfaz inteligente: Tienen la capacidad de captar la secuencia de acciones que el usuario repite con frecuencia para luego adelantarse y brindar la posibilidad de completar la secuencia de acciones en forma automática. Dentro de este tipo se encuentran: Las adaptativas que brindan diferentes modos de interacción que se pueden seleccionar automáticamente de acuerdo al tipo de usuario en cuestión y son sensibles a los perfiles individuales de los usuarios y a sus estilos de interacción. Las evolutivas que tienen la propiedad de cambiar y evolucionar con el tiempo, junto con el grado de perfeccionamiento que el usuario va adquiriendo con el sistema. Y las interfaces accesibles que son las que respetan las normas del diseño universal para que puedan ser accedidas.

- <u>Aspectos de diseño:</u>
  - Principios de Nielsen: son utilizados para el diseño de nuevas interfaces y, como métricas de evaluación de interfaces ya desarrolladas.
    - 1.- Diálogo simple y natural: Forma en que la interacción con el usuario debe llevarse a cabo. Realizar una escritura correcta, sin errores de tipeo. No mezclar información importante con la irrelevante. Distribución adecuada de la información. Prompts lógicamente bien diseñados. Evitar el uso excesivo de mayúsculas y de abreviaturas. Unificar el empleo de las funciones predefinidas.
    - 2.- Lenguaje del usuario: Emplear en el sistema un lenguaje familiar para el usuario Usar el lenguaje del usuario. No utilizar palabras técnicas ni extranjeras. Evitar el truncamiento excesivo de palabras. Diseñar correctamente las entradas de datos. Emplear un grado adecuado de información (ni excesivo ni escaso).
    - 3.- Minimizar el uso de la memoria del usuario: Evitar que el usuario esfuerce su memoria para interactuar con el sistema. Brindar Información de contexto. Brindar información de la navegación y sesión actual. Visualización de rangos de entrada admisibles, ejemplos, formatos.
    - 4.- Consistencia: Que no existan ambigüedades en el aspecto visual ni tecnológico en el diálogo o en el comportamiento del sistema. La consistencia es un punto clave para ofrecer confiabilidad y seguridad al sistema. Debe existir una consistencia terminológica y visual.
    - 5.- Feedback: Es una respuesta gráfica o textual en la pantalla, frente a una acción del usuario. El sistema debe mantener al usuario informado de lo que está sucediendo. Brindar información de los estados de los procesos. Brindar información del estado del sistema y del usuario. Utilización de mensajes de aclaración, validaciones, confirmación y cierre. Realizar validaciones de los datos ingresados por el usuario.
    - 6.- Salidas evidentes: Que el usuario tenga a su alcance de forma identificable y accesible una opción de salida. Brindar salidas de cada pantalla. Salidas para cada contexto. Salidas para cada acción, tarea o transacción. Brindar salidas en cada estado. Visualización de Opciones de Cancelación, Salidas, de Suspender, de Deshacer y Modificación.
    - 7.- Mensajes de error: Feedback del sistema ante la presencia de un error. De qué forma se ayuda al sistema para que salga de la situación en la que se encuentra. Deben existir mensajes de error para ser usados en los momentos que corresponda. Brindar Información del error, explicar el error y dar alternativas a seguir. Se deben categorizar los diferentes tipos de mensajes. No deben existir mensajes de error intimidatorios. Manejar adecuadamente la forma de aparición de los mensajes.
    - 8.- Prevención de errores: Evitar que el usuario llegue a una instancia de error. Brindar rangos de entradas posibles para que el usuario seleccione y no tipee. Mostrar ejemplos, valores por defecto y formatos de entrada admisibles. Brindar mecanismos de corrección automática en el ingreso de los datos. Flexibilidad en las entradas de los usuarios.

- 9.- Atajos: La interfaz debería proveer de alternativas de manejo para que resulte cómodo y amigable tanto para usuarios novatos como para usuarios experimentados. Brindar mecanismos alternativos para acelerar la interacción con el sistema. Brindar la posibilidad de reorganizar barras de herramientas, menús, de acuerdo a la necesidad del usuario. Brindar mecanismos de Macros, atajos, definición de teclas de función.
- 10.- Ayudas: Componentes de asistencia para el usuario. Un mal diseño de las ayudas puede llegar a entorpecer y dificultar la usabilidad. Deben existir las ayudas. Se deben brindar diferentes tipos de ayuda: generales, contextuales, específicas, en línea. Las ayudas deben proveer diferentes formas de lectura. Se deben brindar diferentes mecanismos de asistencia como búsquedas, soporte en línea, e-mail del soporte técnico, acceso a las preguntas frecuente.

- Reglas básicas del diseño: Theo Mandel (1997) indica como reglas:
  - Dar control al usuario: El usuario busca un sistema que reaccione a sus necesidades y lo ayude a hacer sus tareas. Por ejemplo: Definir modos de interacción de forma que el usuario no realice acciones innecesarias. Proporcionar una interacción flexible. Incluir las opciones de interrumpir y deshacer. Depurar la interacción a medida que aumenta la destreza del usuario. Ocultar al usuario ocasional los elementos técnicos internos. Diseñar interacción directa con los objetos que aparecen en pantalla.
  - Reducir la carga de memoria del usuario: Reducir la demanda a corto plazo. Definir valores por defecto que tengan significado. Definir accesos directos intuitivos. El formato visual de la interfaz debe basarse en una metáfora de la realidad. Desglosar la información de manera progresiva.
  - Lograr una Interfaz consistente: Permitir que el usuario incluya la tarea actual en un contexto que tenga algún significado (el usuario debe tener la capacidad de determinar de dónde viene y hacia donde puede ir) Mantener consistencia en toda la familia de aplicaciones (utilizar las mismas reglas de diseños para las mismas interacciones). Mantener modelos que son prácticos para el usuario, a menos que sea imprescindible cambiarlos.
  - Factores Humanos: Percepción visual/auditiva/táctil. Memoria humana. Razonamiento. Capacitación. Comportamiento/Habilidad personales. Diversidad de usuarios (Usuarios casuales: Necesitan interfaces que los guíen. Usuarios experimentados: Requieren interfaces ágiles).
- Presentación de la información: Mantener separada la lógica del software de la presentación y la información misma (enfoque MVC).
  Manejo de los colores: Limitar el número de colores utilizados. No asociar solamente colores a significados. (10% de los humanos no perciben el color. Acompañarlos de algún otro tipo de identificación). Usar los colores consistentemente. Usar cambio de color para mostrar cambios en el estado del sistema. Combinar los colores cuidadosamente.
- Soporte al usuario: Mensajes del sistema por acciones del usuario. Ayudas en línea. Documentación del sistema.
- Usabilidad: se obtiene cuando la arquitectura de la interfaz se ajusta a las necesidades de las personas que la emplearán.
  Donahue la define: “La usabilidad es una medida de cuán bien un sistema de cómputo facilita el aprendizaje, ayuda a quienes lo emplean a recordar lo aprendido, reduce la probabilidad de cometer errores, les permite ser eficientes y los deja satisfechos con el sistema.”
  La forma de determinar si existe “usabilidad” en un sistema que se construye es evaluarla o probarla.

- **METRICAS**

Las métricas son la clave tecnológica para el desarrollo y mantenimiento exitoso del software. (Briand et al., 1996)

En general, la medición persigue los siguientes objetivos fundamentales (Fenton y Pfleeger, 1997): Entender qué ocurre durante el desarrollo y el mantenimiento. Controlar qué es lo que ocurre en nuestros proyectos. Mejorar nuestros procesos y nuestros productos. Evaluar la calidad.

  - Métrica: medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. El ingeniero de software recopila medidas y desarrolla métricas para obtener indicadores.
  - Medida: indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto.
  - Medición: es el acto de determinar una medida.
  - Indicador: combinación de métricas. Proporciona una visión profunda que permite al gestor de proyectos o a los ingenieros de software ajustar el producto, el proyecto o el proceso para que las cosas salgan mejor.

Las métricas pueden ser utilizadas para que los profesionales e investigadores puedan tomar las mejores decisiones.

Existen dos formas en que pueden usarse las mediciones de un sistema de software: Para asignar un valor a los atributos de calidad del software. Para identificar los componentes del sistema cuya calidad está por debajo de un estándar.

  - Clasificación:
    - Métricas de control: Apoyan la gestión del proceso. Ej.: esfuerzo promedio, tiempo requerido para reparar defectos.
    - Métricas de predicción (del producto): Ayudan a predecir las características del software. Se conocen también como métricas del producto. Por ej.: Tamaño, complejidad.
  - Métricas del producto: son métricas de predicción para medir los atributos internos de un sistema de software. Se dividen en dos clases:
      - Métricas dinámicas: que se recopilan de un programa en ejecución y ayudan a valorar la eficiencia y fiabilidad de un programa. Ej. Nro. de reportes de bugs.
      - Métricas estáticas: que se recopilan mediante mediciones hechas de representaciones del sistema y ayudan a valorar la complejidad, comprensibilidad y mantenibilidad de un sistema de software o sus componentes. Ej.: tamaño del código.
    - Principios: se pueden usar para caracterizar y validar las métricas. Una métrica debe tener propiedades matemáticas deseables (rango significativo). Cuando una métrica representa una característica de software que aumenta cuando se presentan rasgos positivos o que disminuye al encontrar rasgos indeseables, el valor de la métrica debe aumentar o disminuir en el mismo sentido. Cada métrica debe validarse empíricamente en una amplia variedad de contextos antes de publicarse o aplicarse en la toma de decisiones.
    - LDC – Líneas de código: Medida directa del software y del proceso. Medida discutida porque depende del lenguaje y es post-mortem. Es para saber en qué tiempo voy a terminar el software y cuántas personas voy a necesitar. Si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño.
    - Métricas Post-Mortem: Conformar una línea base para futuras métricas. Ayudar al mantenimiento conociendo la complejidad lógica, tamaño, flujo de información, identificando módulos críticos. Ayudar en los procesos de reingeniería.
    - Métricas orientadas al tamaño: Métricas derivadas del proceso de desarrollo: KLDC (miles de líneas de código. Productividad: relación entre KLDC / Persona mes. Calidad: relación entre Errores / KLDC. Costo: relación entre $ / KLDC.
      Exigen explicar el manejo de: líneas en blanco, líneas de comentarios, declaraciones de datos, líneas con varias instrucciones separadas.
    - Propuesta Fenton/Pfleeger: Medir: CLOC = Cantidad de líneas de comentarios.

Luego: long total (LOC) = NCLOC + CLOC. Surgen medidas indirectas: CLOC/LOC mide la densidad de comentarios.

- Métricas de control: Un ejemplo de métrica de control es la llamada Métrica de Punto Función (FP), que examina el modelo de análisis con la intención de predecir el tamaño del sistema. Mide la cantidad de funcionalidad de un sistema descripto en una especificación.
  Métricas derivadas: Productividad: relación entre PF y Persona_mes. Calidad: relación entre Errores y PF. Costo: relación entre $ y PF.
- Métrica GQM (OPM): está orientado a lograr una métrica que "mida" cierto objetivo. El mismo nos permite mejorar la calidad de nuestro proyecto. Estructura:
  - Nivel Conceptual (Goal / Objetivo). Se define un objetivo (en nuestro caso, para el proyecto).
  - Nivel Operativo (Question / Pregunta). Se refina un conjunto de preguntas a partir del objetivo, con el propósito de verificar su cumplimiento.
  - Nivel Cuantitativo (Metric / Métrica). Se asocia un conjunto de métricas para cada pregunta, de modo de responder a cada una de un modo cuantitativo.

  Es útil para decidir qué medir. Debe estar orientado a metas. Es flexible.

- **ESTIMACIONES**

Técnica que permiten dar un valor aproximado.
Para obtener estimaciones confiables generalmente se usan varias técnicas y se comparan y concilian resultados. La estimación no es una ciencia exacta. Modificaciones en la especificación hacen peligrar las estimaciones. Requieren experiencia, acceso a información histórica y decisión para convertir información cualitativa en cuantitativa.
El riesgo de la estimación decrece con la disponibilidad de historia. Se realizan estimaciones de recursos, costos y tiempos. Los factores que influyen son la complejidad, el tamaño, la estructuración del proyecto.

- Estimaciones de costos: Existen tres principales parámetros que se deben usar al calcular los costos de un proyecto:
  - Costos de esfuerzo (pagar desarrolladores e ingenieros)
  - Costos de hardware y software, incluido el mantenimiento
  - Costos de viaje

  Para la mayoría de los proyectos, el mayor costo es el primer rubro.
  Debe estimarse el esfuerzo total (meses-hombre), sin embargo, se cuenta con datos limitados para esta valoración. Es posible que se deba licenciar el middleware y la plataforma, o que se requieran mayor cantidad de viajes cuando se desarrolla en distintos lugares. Se debe iniciar con un bosquejo de Plan de Proyecto y se debe contar con una especificación de los requerimientos.
- Fijación de precio: En principio el precio es simplemente el costo de desarrollo, sin embargo, en la práctica, la relación entre el costo y el precio al cliente no es tan simple. Cuando se calcula un precio hay que considerar temas de índole organizacional, económica, política y empresarial. Debe pensarse en los intereses de la empresa, los riesgos y el tipo de contrato. Esto puede hacer que el precio suba o baje.
- Estimaciones de recursos: Recursos humanos. Recursos de software reutilizables. Recursos de hardware y herramientas de software.
  Cada recurso requiere: Descripción. Informe de disponibilidad. Fecha en que se lo requiere. Tiempo que se lo necesita.
- Técnicas de estimación:
  - Juicio experto: Se consultan varios expertos. Cada uno de ellos estima. Se comparan y discuten.
  - Técnica Delphi: Consiste en la selección de un grupo de expertos a los que se les pregunta su opinión. Las estimaciones de los expertos se realizan en sucesivas rondas, anónimas, con el objeto de tratar de conseguir consenso, pero con la máxima autonomía por parte de los participantes.
  - División de trabajo: Jerárquica hacia arriba.
- Modelos empíricos de estimación: Utilizan fórmulas derivadas empíricamente para predecir costos o esfuerzo requerido en el desarrollo del proyecto. Ej: MODELO COCOMO de Boehm (1981).

(COnstructive COst MOdel, modelo constructivo de costos) se obtuvo recopilando datos de varios proyectos grandes. Las fórmulas que utiliza el COCOMO vinculan el tamaño del sistema y del producto, factores del proyecto y del equipo con el esfuerzo necesario para desarrollar el sistema.

- COCOMO 81: El modelo inicial consideraba tres tipos de proyectos:
  - Orgánicos: Proyectos pequeños y de poca gente.
  - Semiacoplados: Proyectos intermedios.
  - Empotrados: Proyectos con restricciones rígidas.
- COCOMO II (2000): Reconoce que las líneas de código son difíciles de estimar tempranamente. Considera diferentes enfoques para el desarrollo, como la construcción de prototipos, el desarrollo basado en componentes, desarrollo en espiral y engloba varios niveles que producen estimaciones detalladas de forma incremental.
  Está compuesto por 4 niveles:
  - De construcción de prototipos: La fórmula para el cálculo del esfuerzo para el prototipado del sistema es:

    PM = (NAP x (1-%reutilización/100))/PROD

    PM = esfuerzo estimado en personas/mes
    NAP = total de puntos de aplicación en el sistema a desarrollar.
    PROD = productividad medida en puntos objeto.
    Productividad de puntos de objeto, se basa en la siguiente tabla para obtener el PROD de la formula anterior.

| Experiencia y capacidad de los desarrolladores | Muy baja | Baja | Media | Alta | Muy alta |
|---|---|---|---|---|---|
| Madurez y capacidad de las herramientas CASE | Muy baja | Baja | Media | Alta | Muy alta |
| PROD (NOP/mes) | 4 | 7 | 13 | 25 | 50 |

  - De diseño inicial: La fórmula para las estimaciones en este nivel es:

    Esfuerzo = A x Tamaño(elevado a la B) x M

    Boehm propone que A = 2.94. (Otros autores proponen: 2.45)
    Tamaño = KLDC (miles de líneas de código fuente).
    B = 0.91 X Σ SFj (ver tabla)
    M = PERS x RCPX x RUSE x PDIF x PREX x FCIL x SCED (ver más adelante).
    Tabla de Valores de escala SF para calcular B:

| | Muy bajo | bajo | nominal | alto | Muy alto | Extra alto |
|---|---|---|---|---|---|---|
| PREC | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 |
| FLEX | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 |
| RESL | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 |
| TEAM | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 |
| PMAT | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 |

    **PREC**: experiencia de proyectos precedentes. Desde totalmente sin precedentes hasta totalmente familiar
    **FLEX**: flexibilidad de desarrollo. Desde requerimientos muy rígidos hasta solamente metas generales
    **RESL**: Nivel de riesgos y tiempo dedicado a arquitectura, desde casi nada a total
    **TEAM**: cohesión del equipo. Desde dificultades graves en interacción hasta interacción suave.
    **PMAT**: Madurez de acuerdo a CMM: Desde 1 hasta 5 (uno se repite)

    Aclaración siglas de M: M: Multiplicador.
    Son siete características/atributos del proyecto y del proceso que influyen en la estimación. Éstas hacen que aumente o disminuya el esfuerzo requerido.
    RCPX = Fiabilidad y complejidad del producto. RUSE = Reutilización requerida.
    PDIF = Dificultad de la plataforma. PERS = Capacidad del personal.
    PREX = Experiencia del personal. SCED = Calendario.
    FCIL = Facilidades de apoyo.

Se pueden estimar directamente en una escala de 1 (valor muy bajo) a 6 (valor muy alto).

- De reutilización: Para el código generado automáticamente, el modelo estima el número de persona/mes necesarias para integrar este código.

  PMauto = (ASLOC x AT/100)/ATPROD.

  AT = porcentaje de código adaptado que se genera automáticamente.
  ATPROD = productividad de los ingenieros que integran el código
  ASLOC = Nro de líneas de código en los componentes que deben ser adaptadas
- De post-arquitectura: Las estimaciones están basadas en la misma fórmula básica

  PM = A x TamañoB x M

  pero se utiliza un conjunto más extenso de atributos (17 en lugar de 7) de producto, proceso y organización para refinar el cálculo del esfuerzo inicial.
  La estimación del número de líneas de código se calcula utilizando tres componentes: Una estimación del número total de líneas nuevas de código a desarrollar. Una estimación del número de líneas de código fuente equivalentes (ESLOC) calculadas usando el nivel de reutilización. Una estimación del número de líneas de código que tienen que modificarse debido a cambios en los requerimientos.
  Estas estimaciones se añaden para obtener el tamaño del código (KLDC).
  El exponente B se calcula considerando 5 factores de escala, como Productividad de desarrollo, Cohesión del equipo, Entre otros