## INTRODUCCIÓN

La ingeniería del software es una disciplina de la ingeniería cuya meta es el desarrollo costeable de sistemas de software. Éste es abstracto, intangible y fácil de modificar. Que el software tenga estas características simplifica la ingeniería del software ya que no existen limitaciones físicas del potencial del software, pero la dificulta en el sentido que el software puede llegar a ser extremadamente complejo y difícil de entender, debido a no tener restricciones naturales.

La noción de **ingeniería del software** surgió en el año 1968, debido a la "**crisis del software**".

| CRISIS DEL SOFTWARE | |
|---|---|
| **CAUSA** | Introducción de nuevas computadoras hardware basadas en circuitos integrados. Esto provocó la posibilidad de desarrollar software más grande y más complejo. |
| **SINTOMAS** | • Los proyectos se atrasaban.<br>• Se sobrepasaban los presupuestos.<br>• Muchos proyectos se cancelaban.<br>• Se comenzaba a desarrollar software sin especificaciones. |
| **CONSECUENCIAS** | Software de mala calidad y de desempeño pobre, lo cual derivaba en la realización de intensas actividades de mantenimiento, lo cual degrada el software. |

Entonces fue evidente que para crear software de esta magnitud tomar un enfoque informal no era lo adecuado. Se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los sistemas grandes. Sin embargo, cuanto más crezca nuestra capacidad para producir software, también lo hará la complejidad de los sistemas de software solicitados.

### Software

Un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan a los usuarios descargar la información de productos recientes. Existen 2 tipos:

- **Productos genéricos:** Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente. La especificación es controlada por la organización que desarrolla.
- **Productos personalizados (o hechos a medida):** Son sistemas requeridos por un cliente en particular. La especificación de los productos personalizados, por lo general, es desarrollada y controlada por la organización que compra el software.

### Ingeniería del Software

La ingeniería del software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza. En esta definición, existen dos frases clave:

- **Disciplina de la ingeniería:** Los ingenieros aplican teorías, métodos y herramientas donde sean convenientes, pero las utilizan de forma selectiva y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías y métodos aplicables para resolverlos. Los ingenieros también saben que deben trabajar con

restricciones financieras y organizacionales, por lo que buscan soluciones tomando en cuenta estas restricciones.

- **Todos los aspectos de producción de software**: La ingeniería comprende disciplinas:
  - **Técnicas:** Ayudan a construir el producto. Éstas son: Rq´s, análisis, diseño, implementación, prueba.
  - **De Gestión:** Planificación de proyectos, Monitoreo y control.
  - **De Soporte:** Gestión de Configuración, métricas, aseguramiento de la calidad.

**Diferencia entre ingeniería del software y ciencia de la computación**

- La **ciencia de la computación** se refiere a las teorías y métodos subyacentes a las computadoras y los sistemas de software→Vendría a ser la TEORIA.
- **La ingeniería del software** se refiere a los problemas prácticos de producir software. Lo ideal sería que todos los ingenieros de software conocieran las teorías de la ciencia de la computación, pero a veces éstos se basan en enfoques *ad hoc* para desarrollar el software, ya que no todas las teorías son aplicables a problemas reales→Vendría a ser la PRÁCTICA.

**Diferencia entre ingeniería del software y ingeniería de sistemas**

No entiendo… pero el libro dice:
La ingeniería de sistemas se refiere a todos los aspectos del desarrollo y de la evolución de sistemas complejos donde el software desempeña un papel principal. Por lo tanto, la ingeniería de sistemas comprende el desarrollo de hardware, políticas y procesos de diseño y distribución de sistemas, así como la ingeniería del software. Los ingenieros de sistemas están involucrados en la especificación del sistema, en la definición de su arquitectura y en la integración de las diferentes partes para crear el sistema final. Están menos relacionados con la ingeniería de los componentes del sistema (hardware, software, etc.).
La ingeniería de sistemas incluye a la ingeniería de software.

**Proceso de Software**

Es un conjunto de actividades y resultados asociados que producen un producto de software. Existen cuatro actividades fundamentales que son comunes para todos los procesos y son:

- **Especificación de Software:** Los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
- **Desarrollo de Software:** El software se diseña y programa.
- **Validación de Software:** El software se valida para asegurar que es lo que el cliente realmente quiere.
- **Evolución del Software:** El software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

El proceso de desarrollo se elegirá de acuerdo al tipo de sistema que se vaya a desarrollar. Además estas actividades genéricas pueden organizarse de diferentes formas y describirse en diferentes niveles de detalle para diferentes tipos de software. Sin embargo, el uso de un proceso inadecuado del software puede reducir la calidad o la utilidad del producto de software que se va a desarrollar y/o incrementar los costos de desarrollo.

## Modelo de procesos del software

Es una descripción simplificada de un proceso del software que presenta una visión de ese proceso. Estos modelos pueden incluir:

- Actividades.
- Productos.
- Papel de las personas involucradas.

Algunos ejemplos de estos tipos de modelos son los modelos de flujo de trabajo, modelos de flujo de datos o de actividad, y modelos de rol acción.
Estos modelos se basan en uno de los tres modelos generales o paradigmas de desarrollo de software:

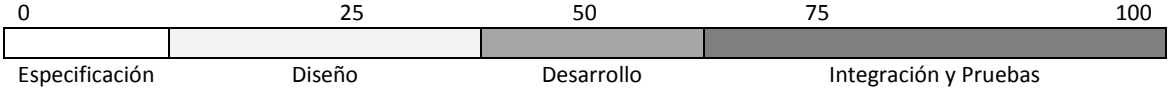
- **Enfoque Cascada:** Considera las actividades anteriores y las representa como fases de procesos separados. Después de que cada etapa queda definida «se firma» y el desarrollo continúa con la siguiente etapa.
- **Desarrollo iterativo:** Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones muy abstractas y luego se refina basándose en las peticiones del cliente, de manera de producir un sistema que satisfaga sus necesidades.
- **Ingeniería del software basada en componentes (CBSE):** Esta técnica supone que las partes del sistema ya existen, por lo cual se enfoca en la integración de estas partes más que en desarrollarlas desde el principio.

## Costos de la ingeniería del software

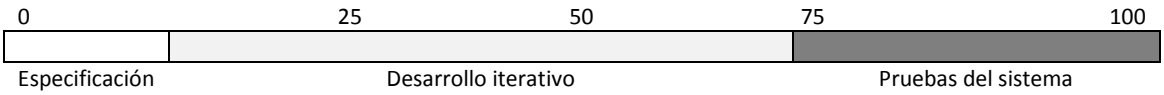
La distribución de costos a través de las diferentes actividades en el proceso del software depende del proceso utilizado y del tipo de software que se vaya a desarrollar.

**Costos del desarrollo para sistemas a medida:**

- **Utilizando un enfoque en cascada:** Los costos de especificación, diseño, implementación e integración se miden de forma separada. La integración y pruebas del sistema son las actividades de desarrollo más caras. Normalmente, suponen alrededor del 40%, 50% o más del costo del desarrollo total.

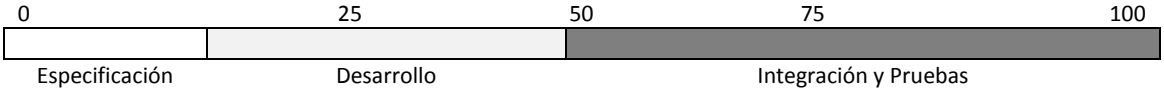



- **Utilizando un enfoque iterativo:** Los costos de la especificación se reducen debido a que sólo se produce la especificación de alto nivel antes que el desarrollo. La especificación, el diseño, la implementación, la integración y las pruebas se llevan a cabo en paralelo dentro de una actividad de desarrollo. Sin embargo, aún se necesita una actividad independiente de pruebas del sistema una vez que la implementación inicial esté completa.




- **Utilizando ingeniería del software basada en componentes:** Los costos de desarrollo se reducen en relación a los costos de integración y pruebas. Los costos de integración

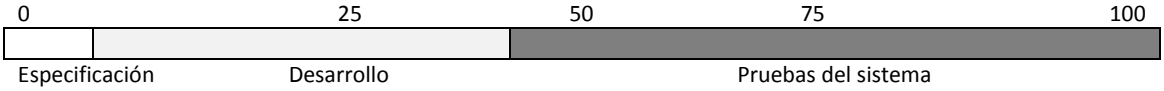
y pruebas se incrementan porque tenemos que asegurarnos de que los componentes que utilizamos cumplen realmente su especificación y funcionan como se espera.




**Costos de evolución para sistemas a medida:**

- Para sistemas software de larga vida, que pueden ser usados durante 10 años o más, estos costos exceden a los de desarrollo por un factor de 3 o 4.
- Para sistemas de negocio más pequeños, que tienen una vida mucho más corta, los costos de evolución son más reducidos.

**Costos del desarrollo para sistemas genéricos:** Los costos de la especificación son relativamente bajos. Sin embargo, debido que se pretende utilizarlos en diferentes configuraciones, deben ser probados a fondo.




**Costos de evolución para sistemas a medida:** En muchos casos, existe poca evolución de un producto. Una vez que una versión de producto se entrega, se inicia el trabajo para entregar la siguiente y, por razones de mercadotecnia, ésta se presenta como un producto nuevo más que como una versión modificada de un producto que el usuario ya adquirió. Por lo tanto, los costos de evolución no se consideran de forma separada, sino que son sencillamente los costos del desarrollo para la siguiente versión del sistema.

## Métodos de la ingeniería del software

Es un enfoque estructurado para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable. Algunos de ellos son:

- Métodos orientados a funciones: Por ejemplo Análisis Estructurado y JSD.
- Métodos orientados a objetos.

No existe un método ideal, y métodos diferentes tienen distintas áreas donde son aplicables.
Todos los métodos se basan en la idea de modelos gráficos de desarrollo de un sistema y en el uso de estos modelos como un sistema de especificación o diseño. Los métodos incluyen varios componentes diferentes.

## Atributos de un buen software

Los atributos reflejan la calidad del software. No están directamente asociados con lo que el software hace. Más bien, reflejan su comportamiento durante su ejecución y en la estructura y organización del programa fuente y en la documentación asociada.
El conjunto específico de atributos que se espera de un sistema de software depende obviamente de su aplicación.

| ATRIBUTOS ESENCIALES DE UN BUEN SOFTWARE | |
|---|---|
| **MANTENIBILIDAD** | El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de Cambio de los dientes. Éste es un atributo crítico debido a que el cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocios. |
| **CONFIABILIDAD** | La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, protección y seguridad. El software confiable no debe causar daños físicos o económicos en el caso de una falla del sistema. |
| **EFICIENCIA** | El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento. La eficiencia incluye tiempos de respuesta y de procesamiento, utilización de memoria, etc. |
| **USABILIDAD** | El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada. |

### Retos que afronta la ingeniería del software

- **Reto de la heterogeneidad:** Consiste en desarrollar técnicas para construir software confiable que sea lo suficientemente flexible, es decir, que pueda operar como un sistema distribuido en redes con distintos tipos de computadoras, que se integre con sistemas heredados, con sistemas desarrollados en otros lenguajes, etc.
- **Reto de la entrega:** Reducir los tiempos de entrega para sistemas grandes y complejos sin comprometer la calidad del sistema.
- **Reto de la confianza:** Desarrollar técnicas que demuestren que los usuarios pueden confiar en el software.

## GESTIÓN DE PROYECTOS

### Características de un Proyecto

- Están orientados a objetivos:
  - Los proyectos están dirigidos a obtener resultados y ello se refleja a través de objetivos.
  - Los objetivos guían al proyecto.
  - Los objetivos deben ser ambiguos.
  - Los objetivos deben ser claros y alcanzables.
- Tienen una duración limitada en el tiempo, tienen principio y fin:
  - Los proyectos son temporarios, cuando se alcanzan los objetivos, el proyecto termina.
  - Una línea de producción no es un proyecto.
- Implican tareas interrelacionadas basadas en esfuerzos y recursos.
- Son únicos: Todos los proyectos por similares que sean tienen características que los hacen únicos, por ejemplo, que tienen cronogramas diferentes.

### Administración de Proyectos

Es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.
Administrar un proyecto incluye:

- Identificar los requerimientos.

- Establecer objetivos claros y alcanzables.
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders).

**Factores de Calidad**

- Objetivos de proyecto.
- Tiempo.
- Costos.

El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estos tres objetivos (que a menudo compiten entre ellos y por lo general casi nunca se cumplen simultáneamente).

**ERRORES CLASICOS EN PROYECTOS Y ORGANIZACIONES**

El desarrollo de software es una actividad complicada. Un proyecto de software típico puede presentar muchas oportunidades para aprender de lo errores originados en la aplicación frecuente de prácticas inefectivas. Estos errores se clasifican en:

- Gente.
- Proceso.
- Producto.
- Tecnología.

**Gente**

1. **Motivación débil:** Se ha demostrado que la motivación tiene mayor efecto sobre la productividad y la calidad que ningún otro factor.
2. **Personal mediocre:** La capacidad individual de los miembros del equipo, así como sus relaciones como equipo, probablemente tienen la mayor influencia en la productividad.
3. **Empleados problemáticos incontrolados:** Amenazan la velocidad del desarrollo.
4. **Hazañas:** Algunos desarrolladores ponen un gran énfasis en la realización de hazañas en sus proyectos. Pero lo que hacen tienen más de malo que de bueno. Este tipo de actitudes convierten pequeños contratiempos en auténticos desastres.
5. **Añadir mas personal a un proyecto retrasado:** Es este el más clásico de los errores. El aprendizaje agrega más tiempo.
6. **Oficinas repletas y ruidosas:** La mayoría de los desarrolladores consideran sus condiciones de trabajo como insatisfactorias. Los entornos repletos y ruidosos alargan los planes de desarrollo.
7. **Fricciones entre clientes y desarrolladores:** El principal efecto de esta fricción es la mala comunicación y los efectos secundarios son pobres entendimientos de los requerimientos, pobre desarrollo de la interfaz, rechazo del cliente por el producto terminado.
8. **Expectativas poco realistas:** Una de las causas más comunes de fricciones entre los desarrolladores y sus clientes o los directivos son las expectativas poco realistas.
9. **Falta de un promotor efectivo del proyecto:** Sin un promotor ejecutivo del proyecto el resto de los gerentes de la empresa puede presionar para que se acepten fechas poco realistas o hacer cambios que afecten seriamente al proyecto.
10. **Falta de la participación de los implicados.**
11. **Falta de la participación del usuario.**

12. **Política antes que desarrollo:** Primar la política en vez de los resultados. Es fatal para el desarrollo orientado a la velocidad.
13. **Ilusiones:** Las ilusiones no son sólo optimismo. Realmente consisten en cerrar los ojos y esperar que todo funcione cuando no se tienen las bases razonables para pensar que será así. Impiden llevar a cabo una planificación coherente y pueden ser la raíz de más problemas en el software que todas las otras causas combinadas.

**Proceso**

14. **Planificación excesivamente optimista:** Esto hace que el proyecto falle por infravalorar el alcance del proyecto, reduciendo las actividades críticas. También afecta a los desarrolladores, quienes se ven afectados en su moral y productividad.
15. **Gestión de riesgos insuficientes:** No hacerlo ya es un error clásico y muchos de los errores resaltados aquí son consecuencia de no llevar a cabo esta etapa en un proyecto.
16. **Fallos de los contratados:** Si las gestiones con terceros no se gestionan oportunamente, la utilización de contratados puede demorar el proyecto en vez de acelerarlo.
17. **Planificación Insuficiente:** Sino se planifica un desarrollo rápido no se puede esperar obtenerlo.
18. **Abandonar la planificación bajo presión:** Los equipos desarrollan planes y los abandonan cuando tropiezan con un problema de planificación. El problema no está en el abandono del plan sino en no tener un plan alternativo y entonces se cae en el modo de codificar y corregir.
19. **Pérdida de tiempo en el inicio difuso:** Es el tiempo que transcurre antes del inicio del proyecto.
20. **Escatimar las actividades iniciales:** Muchos proyectos se aceleran intentando acotar las actividades no esenciales, y por lo general estas actividades son el análisis de requerimientos, arquitectura y diseño, ya que no generan código. Los resultados de este error también son conocidos como “saltar a la codificación” y son predecibles los resultados.
21. **Diseño Inadecuado.**
22. **Escatimar control de calidad**: Proyectos apurados recortan las revisiones de código, los testeos y las revisiones de diseño. Acortar un día estas actividades presupone de 3 a 10 días de retraso al final del proyecto.
23. **Control Insuficiente de la Dirección**.
24. **Convergencia prematura o excesivamente frecuente:** Bastante antes de entregar el producto hay un esfuerzo por entregarlo y entonces se preparan varias actividades que irían al final, esto quita tiempo a otras actividades y no beneficia en nada, puesto que el producto todavía esta siendo desarrollado.
25. **Omitir tareas necesarias en la estimación:** Sino se mantienen registro de anteriores proyectos, se olvidan las tareas menos visibles y son las que luego agregan tiempos. Esfuerzo omitido en un plan agrega de un 20% a 30% al esfuerzo de desarrollo.
26. **Planificar ponerse al día mas adelante.**
27. **Programación a Destajo.** Algunas organizaciones creen que la programación rápida, libre, tal como salga es el camino hacia el desarrollo rápido. Esto rara vez funciona.

**Producto**

28. **Exceso de Requerimientos:** Algunos proyectos tienen más requerimientos de los que necesitan, desde el mismo inicio, lo que alarga innecesariamente el plan de proyecto.
29. **Cambio de las prestaciones:** Un cambio de este calibre pude producir un aumento en el plan de un 25% o más, lo que puede ser fatal para un proyecto de desarrollo rápido.

30. **Desarrolladores meticulosos:** Los desarrollares a menudo se muestran fascinados por una nueva tecnología y quieren implantar estas nuevas características sea que el producto las necesite o no. El esfuerzo requerido por diseñar, codificar, testear y documentar estas prestaciones alarga el plan.
31. **Tires y aflojes en la negociación:** Un directivo que aprueba un retraso en el plan lo hace sabiendo que el plan estaba equivocado, pero una vez que lo corrige realiza acciones explícitas para volver a equivocarse.
32. **Desarrollo orientado a la investigación:** Si el proyecto fuerza los límites de la informática porque necesita la creación de nuevos algoritmos o de nuevas técnicas de computación, no estamos desarrollando software, estamos investigando. Los planes de desarrollo son razonablemente predecibles, los planes en la investigación sobre software ni siquiera son predecibles teóricamente.

**Tecnología**

33. **Síndrome Silver-bullet (de las balas de plata):** Cuando el equipo del proyecto se aferra sólo a una nueva técnica, una nueva tecnología o un proceso rígido y espera resolver con ello sus problemas de planificación, está inevitablemente equivocado.
34. **Sobreestimación de las ventajas del empleo de nuevas herramientas o métodos:** Las organizaciones mejoran raramente sus productividad a grandes saltos, sin importar cuantas nuevas herramientas o método empleen. Los beneficios de las nuevas técnicas son parcialmente desplazados por las curvas de aprendizaje y también se suponen nuevos riesgos que solo se descubren utilizando dichas técnicas.
35. **Cambiar de herramientas a mitad del proyecto:** Es un viejo recurso que funciona raramente. Quizás convenga cambiar de la versión 3 a la 3.1 o quizás a la 4. Pero la curva de aprendizaje, el retrabajo y los errores cometidos con la nueva herramienta, normalmente anulan cualquier beneficio.
36. **Falta de control automático de código fuente:** Sin él se expone el proyecto a riesgos innecesarios y dos programadores que trabajan sobre un mismo trozo de código deben coordinar manualmente sobre que versión se hacen las actualizaciones.

**Habilidades (skills) de un Líder de Proyecto**

- Sentirte cómodo con los cambios.
- Entender a la organización y su estructura.
- Ser capaz de guiar al equipo y motivarlo para satisfacer los objetivos del proyecto.
- Necesidad de dos tipos de skills:
    - Hard skills: Conocimiento del producto, conocer las herramientas y técnicas de administración de proyectos.
    - Soft skills: Ser capaz de trabajar con gente. Esto incluye:
        - Comunicación: saber escuchar, persuadir.
        - Organizacional: Planificar, plantear de objetivos, análisis.
        - Team Building: Empatía, motivación, espíritu de cuerpo.
        - Liderazgo: Dar ejemplos, enérgico, tener el big picture, delegar, ser positivo.
        - Flexibilidad, creatividad, paciencia, persistencia.
        - Tecnológicos: experiencia, conocimiento del proyecto.

## ETAPAS DEL PROYECTO (según lo que dijo el profe en la clase del viernes 7)

- Iniciación del proyecto.
- Planificación del proyecto y armado del equipo.
- Ejecución.
- Tracking y control.
- Post mortem.

### Antes del proyecto

Antes de iniciar un proyecto se debe hacer:

- **Identificación del cliente y del mercado:** Es decir, identificar el dominio, el soporte que se debe proveer, si se debe cumplir con algún marco de calidad y las líneas de reporte.
- **Estudio de factibilidad:** Es un estudio que se realiza a fin de verificar si el proyecto puede realizarse o no. Se llevan a cabo 3 tipos de estudios (factibilidad técnica, económica y operativa). Se evalúa si el proyecto realmente se necesita, cuáles son los requerimientos iniciales, si se dispone del capital necesario y además si el proyecto funcionará y garantizará un éxito mínimo.
- **Propuesta de Proyecto:**
  - Resumen ejecutivo: Cuáles son las funciones básicas y los beneficios al cliente.
  - Arquitectura del sistema: Es decir, qué se va a construir.
  - Plan de Proyecto: Cuáles son los entregables y sus fechas de entrega.
  - Costos: Se pueden dividir por perfiles de los empleados.
- **Negociaciones:**
  - Características del producto: Funcionalidad más importante que tiene el producto. Las características deben estar priorizadas, ser realistas, claramente entendidas y medibles.
  - Cronograma o Schedule: Determinar fechas de comienzo y fin, milestones o Hitos, entregas de productos.
  - Presupuesto: Del Staff, del software y hardware.
  - Estructura del equipo: Curva de Staffing, cantidad de personas por rol.
  - Consideraciones: Si los riesgos se van a compartir, quién será el dueño del código fuente, etc.

### 1- Inicio del Proyecto

Debo Elegir un nombre para el proyecto y anunciarlo, esto dará inicio oficial al proyecto. Se debe definir:

- Estructura del equipo.
- Necesidades de soporte.
- Necesidades financieras.
- Objetivos de calidad.

### 2- Planificación del proyecto y armado del equipo.

- **Definición del proyecto:** Determinar los problemas, las oportunidades, la misión, los objetivos. Es importante que éstos últimos sean:
  - Críticos.
  - Observables.
  - Distinguibles.
  - Medibles.

Es importante también definir las asunciones, los riesgos y las condiciones de aceptación del producto. Esto último permite marcar cuándo el proyecto de desarrollo termina, y cuándo comienza otro de mantenimiento, es decir, marcar los límites del proyecto.

- **Creación del plan:** El **Plan de Proyecto** fija los recursos disponibles, divide el trabajo y crea el calendario de trabajo. Debe utilizarse como un conductor para el proyecto y debe ser lo mejor posible de acuerdo a la información de la que se dispone. Este plan provee una herramienta de comunicación estándar a través del ciclo de vida de un proyecto.
  La planificación es un proceso iterativo, que solamente se completa cuando el proyecto se termina. Conforma la información se hace disponible, el plan debe revisarse y adaptarse.

- **Infraestructura del proyecto:** Los proyectos necesitan empezar con una infraestructura básica, que puede incluir:
  - Administración de Configuración.
  - Un directorio para el proyecto.
  - Lista de Mails.
  - Sitio Web para realizar la comunicación, por ejemplo para postear novedades de l proyecto.
  - Herramientas varias, como ser los entornos de desarrollo y testing.

- **Formación del equipo:** Determinar quiénes serán los miembros iniciales del proyecto, los que provienen de otros proyectos, y cuáles serán sus roles.

- **Actividades Principales:**
  - Armar la infraestructura del proyecto y su entorno.
  - Asegurar el entrenamiento que haga falta: Capacitación General, Capacitación basada en roles específicos, Capacitación de dominios específicos.
  - Producir los documentos “claves” del proyecto:
    - ✓ **REQB**: Libro de requerimientos (Una ERS por ejemplo)
    - ✓ **SPMP:** Plan de Proyecto.
    - ✓ **SQAP** Plan de Aseguramiento de Calidad: Describe los procedimientos y estándares de calidad que se utilizarán en el proyecto.
    - ✓ **SCMP** Plan de Gestión de Configuración: Describe los procedimientos para la gestión de configuraciones y su estructura.
    - ✓ **Directorio del proyecto o repositorio.**

## WORK BREAKDOWN STRUCTURE (WBS)

Una WBS o Estructura de Desglose de Trabajo identifica las actividades que necesitan ser realizadas durante el proyecto. Sus objetivos son:

- Desglosar hasta llegar al nivel de actividades. El límite es que cada actividad debe producir un “entregable” (código, modelos).
- Hacer un bosquejo de cómo se van a estructurar las actividades:
  - En partes lógicas.
  - De acuerdo a las etapas del proyecto.
  - De acuerdo a los paquetes o módulos, etc.

El líder de proyecto es quien decide sobre el enfoque de arquitectura y el nivel de detalle de la WBS. A veces se sigue el siguiente criterio:

- Organizar por etapas → Para empresas que no desarrollan software, o se si va a tercerizar alguna etapa.
- Organizar por módulos → Conviene si hay grupos de trabajo por módulo, o cuando algunos módulos tienen que estar listos antes que otros.

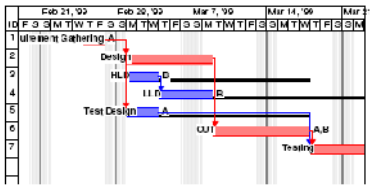
Se dice que una WBS está lista cuando:

- Los elementos del WBS (tareas) son medibles.
- Los eventos que marcan el Comienzo / Fin están claramente definidos.
- Cada "actividad" tiene un entregable.
- Tiempos / costos pueden ser estimados.
- La duración de las actividades tienen limites aceptables (por lo general se recomiendan actividades con duración mínima de 1 semana pero no mayor a las 8 semanas).
- Las asignaciones pueden ser hechas independientemente.

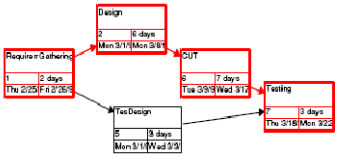
## SCHEDULE

Determina qué tareas deben ser hechas secuencialmente, cuáles pueden ser hechas en paralelo, y luego estima el tiempo de realización de las tareas. Debe tenerse en cuenta también la asignación de recursos a cada una de las actividades.
Un problema en la calendarización surge cuando no se posee información acerca de la duración de actividades similares de proyectos anteriores, lo cual hace que se posea cierta incertidumbre al realizar dichas estimaciones.
Al Schedule podemos representarlo a través de 2 gráficos:

- **Grafico de Gantt:** Es un gráfico de barras, cuyo largo es proporcional a la duración de cada actividad. Es la forma más fácil e intuitiva para visualizar un Schedule y es útil para el tracking.



- **Grafico de Pert:** Es un diagrama de red, que muestra todas las actividades y sus dependencias y hace evidente el camino crítico. Es útil para comprender cuando un set de tareas debe ser completado, pero no es bueno para el tracking.



El camino crítico esta constituido por aquellas actividades que no pueden atrasarse sin aumentar el tiempo total del proyecto.

## HITOS Y ENTREGAS

Un hito es el punto final de una actividad del proceso de software. Cuando se planifica un proyecto, se debe establecer una serie de **hitos.** Éstos deben representar el fin de una etapa lógica en el proyecto y para cada uno de ellos debe existir una salida formal. Una entrega es el resultado del proyecto que se entrega al cliente al final de una fase principal del proyecto. Como regla general, las entregas son hitos, pero éstos no necesariamente son entregas. Dichos hitos pueden ser resultados internos del proyecto que son utilizados por el gestor del proyecto para verificar el progreso del mismo pero que no se entregan al cliente.

## GESTIÓN DE RIESGOS

Es el proceso de identificación de riesgos y manejo de aquellos más importantes para el proyecto de forma tal que no se conviertan en problemas, afectando a la programación del proyecto o a la calidad del software a desarrollar.
El proceso de gestión de riesgos es un proceso iterativo, que se aplica a lo largo de todo el proyecto. A medida que se tenga más información, los riesgos deben analizarse y priorizarse, y deben modificarse los planes de mitigación y contingencia si fuera necesario.

**Riesgo:** Evento que podría comprometer el éxito del proyecto. Probabilidad de que una circunstancia adversa ocurra.

- Toda actividad lleva implícita un riesgo.
- El riesgo acompaña a todo cambio.
- El riesgo implica elección e incertidumbre.
- Los riesgos están caracterizados por la incertidumbre.

Es importante saber diferenciar los riesgos, de los problemas. Los riesgos tienen condiciones y consecuencias inciertas y pueden ser dinámicos. Los problemas son ciertos.

### Categorías de los Riesgos

- **Riesgos del Proyecto:** Afectan la calendarización o los recursos del proyecto.
- **Riesgos del producto:** Afectan a la calidad o al rendimiento del software que se está desarrollando.
- **Riesgos del negocio:** Afectan a la organización que desarrolla o suministra el software.

Los riesgos que pueden afectar a un proyecto dependen del propio proyecto y del entorno organizacional donde se desarrolla. Sin embargo, muchos riesgos son universales.
La gestión de riesgos es importante particularmente para los proyectos de software debido a las incertidumbres inherentes con las que se enfrentan muchos proyectos.

### Estrategias frente a los riesgos

- **Estrategias reactivas:** Se evalúan las consecuencias del riesgo cuando éste ya ha ocurrido. Esto pone en peligro el proyecto.
- **Estrategias proactivas:** Se caracteriza por una evaluación previa y sistemática de riesgos y sus consecuencias, creando planes de mitigación (para reducir el efecto o la probabilidad del riesgo) y planes de contingencia.

### Etapas de la Gestión de Riesgos

#### Identificación de riesgos

Comprende el descubrimiento de los posibles riesgos del proyecto. En esta etapa no hay que valorarlos o darles prioridad, aunque por lo general no se consideran los riesgos con consecuencias menores o con baja probabilidad. Tipos de riesgos (o categorías):

- **Riesgos de tecnología:** Se derivan de las tecnologías de software o de hardware utilizadas en el sistema que se está desarrollando.
- **Riesgos de personal (o relacionados con la experiencia y tamaño del equipo):** Riesgos asociados con las personas del equipo de desarrollo.

- **Riesgos organizacionales (o relacionados con el impacto en la organización):** Se derivan del entorno organizacional donde el software se está desarrollando.
- **Riesgos de herramientas (o Relacionados con el entorno de desarrollo):** Se derivan de herramientas CASE y de otro software de apoyo utilizado para desarrollar el sistema.
- **Riesgos de requerimientos:** Se derivan de los cambios de los requerimientos del cliente y el proceso de gestionar dicho cambio.
- **Riesgos de estimación:** Se derivan de los estimados administrativos de las características del sistema y los recursos requeridos para construir dicho sistema.

**Otros** (estaban en la filmina):

- **Asociados con el tamaño del producto.**
- **Relacionados con el cliente.**
- **Asociados a la definición del proceso de producción.**

**Análisis de Riesgos**

Durante este proceso se considera por separado cada riesgo identificado y se decide acerca de la probabilidad y el impacto del mismo. No se hace una valoración con números precisos sino en intervalos. La medición se realiza de la siguiente manera:

- La probabilidad del riesgo se puede valorar como muy bajo « 10%), bajo (10-25%), moderado (25-50%), alto (50-75%) o muy alto (>75%).
- El impacto del riesgo puede ser valorado como catastrófico, serio, tolerable o insignificante. El impacto del riesgo se mide de acuerdo a 2 aspectos:
  - Alcance: Cuán serio es y cuánto del proyecto se ve afectado.
  - Temporalización de los efectos: Cuándo y por cuánto tiempo.
- La exposición se calcula como el producto entre la probabilidad y el impacto. Representa a la amenaza total del riesgo.

Luego se colocan los riesgos en una tabla ordenados de acuerdo a la exposición de los mismos. Como no pueden gestionarse absolutamente todos los riesgos asociados al proyecto, ya que se necesitaría demasiada información, es necesario discernir cuáles son los más importantes que se deben tratar (se recomiendan 10, pero este numero puede variar). Por lo general se desprecian riesgos poco probables y los medianamente probables con poco impacto.

**Planificación de Riesgos**

En esta etapa se definen las estrategias para gestionar cada riesgo, las cuales pueden dividirse en tres categorías:

- **Estrategias de prevención:** Están orientadas a reducir la probabilidad del riesgo.
- **Estrategias de minimización:** Intentan reducir el impacto del riesgo.
- **Planes de contingencia:** Definen cuál es el plan a seguir en el caso de que el riesgo ocurra. A veces se implementan porque es más barato solucionar el problema que evitarlo.

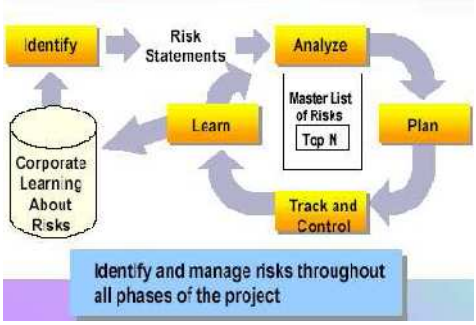
**Supervisión de Riesgos**

En esta etapa se valora cada uno de los riesgos identificados para decidir si éste es más o menos probable y si han cambiado sus efectos. La supervisión de riesgos debe ser un proceso continuo y, en cada revisión, cada uno de los riesgos debe ser analizado por separado ya que éstos pueden desaparecer, ocurrir o mantenerse, y también pueden aparecer nuevos riesgos.

## Estimación de riesgos

En esta etapa se debe determinar cuándo desistir y finalizar el proyecto. Se debe marcar la relación entre cada factor de riesgo enumerado y el punto de referencia y definir el área de incertidumbre, donde será tan válido continuar como interrumpir el trabajo y predecir cómo la combinación de riesgos afectará a los niveles de referencia.

### Proceso de Gestión de Riesgos Proactiva




- Identificación de los Riesgos.
- Análisis (Definir probabilidad e impacto).
- Planificación (contingencia, mitigación).
- Monitoreo y Control (Supervisión).
- Aprendizaje.

Este proceso se realiza continuamente, y permite obtener una base de datos que sirva como base de referencia para los próximos proyectos.

# SOFTWARE E INGENIERIA DEL SOFTWARE

## Software

El software es un transformador de información, realiza la producción, el manejo, la adquisición, la modificación, el despliegue o la transmisión de la información que puede ser tan simple como un solo bit o tan compleja como una presentación multimedia. El software entrega el producto más importante de nuestro tiempo: la información.
Desde el punto de vista del ingeniero de software, el producto obtenido la forman los programas, el contendido (datos) y los documentos que constituyen el software. Pero desde el enfoque del usuario, el producto obtenido es la información resultante que de alguna manera mejora el mundo del usuario.

## Características del software

El software es un elemento lógico, en lugar de físico, de un sistema. Por lo tanto, el software tiene características muy diferentes a las del hardware:

- El software se desarrolla o construye, no se manufactura en el sentido clásico.
- El software no se desgasta, pero si se deteriora: El software es inmune a los males ambientales que desgastan al hardware. La curva de la tasa de fallas para el software debería tener la forma de la “curva idealizada”. Los defectos sin descubrir causan tasas de falla altas en las primeras etapas de vida de un programa. Sin embargo, los errores se corrigen y la curva se aplana. Sin embargo, el software se deteriora: Durante su vida, el software sufre cambios, lo cual puede introducir errores, y ocasiona que la curva de fallas tenga un pico. Antes de que la curva pueda volver a su estado original, se requiere otro cambio lo que ocasiona que la curva tenga otro pico. Por lo tanto, el mantenimiento del software es más complejo que el del hardware.
- A pesar de que la industria tiene una tendencia hacia la construcción por componentes, la mayoría del software aún se construye a medida: Los componentes reutilizables se han creado para que el ingeniero se pueda concentrar en los elementos

que en realidad son innovadores en el diseño. Esto es muy utilizado en el diseño de hardware, pero en el ámbito del software recién se esta comenzando a extender.

**Mitos del software**

| | |
|---|---|
| **MITOS DE LA ADMINISTRACIÓN** | |
| **Mito** | Ya se tiene un libro de estándares y procedimientos para la construcción del software. Esto proporcionará todo el conocimiento necesario. |
| **Realidad** | El libro puede existir, pero lo importante es garantizar que se sepa de su existencia, que se use, que sirva, etc. |
| **Mito** | Si se está atrasado en el itinerario es posible contratar más programadores para así terminar a tiempo. |
| **Realidad** | Agregar gente a un proyecto de software atrasado lo atrasa más. Esto se debe a que la gente que estaba trabajando debe invertir su tiempo en la enseñanza a los nuevos, lo cual reduce el tiempo dedicado al desarrollo. |
| **Mito** | Si decido subcontratar el proyecto de software a un tercero, puedo relajarme y dejar que esa compañía lo construya. |
| **Realidad** | Si una organización no entiende cómo administrar y controlar internamente los proyectos del software, de manera invariable entrará en conflicto al subcontratar este tipo de proyectos. |
| **MITOS DEL CLIENTE** | |
| **Mito** | Un enunciado general de los objetivos es suficiente para comenzar a escribir programas, los detalles se pueden afinar después. |
| **Realidad** | A veces no es factible un enunciado completo y estable de los requerimientos, pero un enunciado ambiguo hará que se corran muchos riesgos. |
| **Mito** | Los requerimientos del proyecto cambian de manera continua, pero el cambio puede ajustarse con facilidad porque el software es flexible. |
| **Realidad** | Cuando los cambios en los requerimientos se solicitan en etapas tempranas, el impacto es pequeño. Pero conforme pasa el tiempo, el impacto en el costo crece con rapidez. Se pueden requerir muchos cambios en el diseño y en los recursos. |
| **MITOS DEL DESARROLLADOR** | |
| **Mito** | Una vez que el programa ha sido escrito y puesto a funcionar, el trabajo esta terminado. |
| **Realidad** | Datos de la industria indican que entre el 60% y el 80% de todo el esfuerzo se realizará después de que el sistema haya sido entregado al cliente por 1º vez. |
| **Mito** | Mientras el programa no se esté ejecutando, no existe forma de evaluar su calidad. |
| **Realidad** | Uno de los mecanismos más efectivos para el aseguramiento de la calidad del software se puede aplicar desde el inicio de un proyecto: la revisión técnica formal. |
| **Mito** | El único producto del trabajo que puede entregarse para tener un proyecto exitoso es el programa en funcionamiento. |
| **Realidad** | El programa en funcionamiento es sólo una parte. La documentación provee el fundamento para la ingeniería exitosa y es una guía para el mantenimiento. |
| **Mito** | La ingeniería del software obligará a emprender la creación de una documentación voluminosa e innecesaria y el proceso se volverá más lento. |
| **Realidad** | La ingeniería del software no se refiere a la elaboración de documentación. Está relacionada con la creación de calidad. Ésta conduce a la reducción de los trabajos redundantes y esto último deriva en menores tiempos de entrega. |

## MÉTRICAS DEL PRODUCTO PARA EL SOFTWARE

Las medidas se emplean para comprender mejor los atributos de los modelos que se crean y para evaluar la calidad de los productos de la ingeniería o de los sistemas que los construyen. Las métricas de producto se recopilan a medida que se realizan las tareas técnicas y proporcionan al ingeniero de software una indicación en **tiempo real** de la eficacia de los modelos de análisis, diseño y código, pero también aportan indicativos de la efectividad de los casos de prueba y la calidad general del software que se construirá. Es importante el hecho de que provean información en tiempo real, ya que permiten descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos.

### Calidad

**Calidad del software:** Es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se esperan del software desarrollado profesionalmente.

### Factores de Calidad de McCall

Los factores que afectan a la calidad del software se concentran en tres aspectos importantes de un producto de software: sus características operativas, su capacidad para experimentar cambios y su capacidad de adaptarse a nuevos entornos.

| **OPERACIÓN DEL PRODUCTO** | |
|---|---|
| **Corrección** | El grado en que el programa cumple con su especificación y satisface los objetivos que propuso el cliente. |
| **Confiabilidad** | El grado en que se esperaría que un programa desempeñe su función con la precisión requerida. |
| **Eficiencia** | La cantidad de código y de recursos de cómputo necesarios para que un programa realice su función. |
| **Integridad** | El grado de control sobre el acceso al software o los datos por parte de las personas no autorizadas. |
| **Facilidad de uso** | El esfuerzo necesario para aprender, operar y preparar los datos de entrada de un programa e interpretar una salida. |
| **REVISIÓN DEL PRODUCTO** | |
| **Facilidad de mantenimiento** | El esfuerzo necesario para localizar y corregir un error en un programa. |
| **Flexibilidad** | El esfuerzo necesario para modificar un programa en operación. |
| **Facilidad de prueba** | El esfuerzo que demanda probar un programa con el fin de asegurar que realiza su función. |
| **TRANSICIÓN DEL PRODUCTO** | |
| **Portabilidad** | El esfuerzo necesario para transferir el programa de un entorno de hardware o software a otro. |
| **Facilidad de reutilización** | El grado en que un programa (o partes de él) puede reutilizarse en otras aplicaciones. |
| **Interoperabilidad** | El esfuerzo necesario para acoplar un sistema con otro. |

Puede encontrarse otra lista de factores de calidad que corresponden al estándar ISO 9126

A veces el esfuerzo por desarrollar medidas precisas de la calidad del software se ve frustrado por la naturaleza subjetiva de la actividad. En todos los casos las métricas representan medidas indirectas , es decir, nunca se mide realmente la calidad, sino alguna manifestación de ésta.

**Medidas, métricas e indicadores**

- **Medida:** Proporciona una indicación cuantitativa de la extensión, la cantidad, la dimensión, la capacidad o el tamaño de algún atributo de un producto o proceso.
- **Medición:** Es el acto de determinar una medida.
- **Métrica (Según el IEEE):** Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado.
- **Indicador:** Es una métrica o una combinación de métricas que proporciona conocimientos que permiten al líder ajustar el proceso, el proyecto o el producto para que las cosas mejoran.

Una métrica de software relaciona de alguna manera las medidas individuales.

**Proceso de medición**

Las actividades que deben desarrollarse son:

- **Formulación:** La derivación de medidas y métricas apropiadas para la representación del software que se considera.
- **Recolección:** El mecanismo con el que se acumulan los datos necesarios para derivar las métricas formuladas.
- **Análisis:** El cálculo de las métricas y la aplicación de herramientas matemáticas.
- **Interpretación:** La evaluación de las métricas para conocer mejor la calidad de la representación.
- **Retroalimentación:** Recomendaciones derivadas de la interpretación de las métricas del producto transmitidas al equipo de software.

Las métricas del software sólo serán útiles si están caracterizadas de manera efectiva y si se validan para probar su valor.

**Principios para caracterizar y validar las métricas**

- **Una métrica debe tener propiedades matemáticas deseables:** Es decir, el valor de la métrica debe estar en un rango significativo (de 0 a 1 por ejemplo), y realizar la medición en la escala que se mide el componente.
- **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 a la toma de decisiones:** Una métrica debe "crecer" para aplicarse en sistemas grandes y funcionar en diversos lenguajes de programación y dominios de sistemas.

Aunque la formulación, caracterización y validación son críticas, la recopilación y el análisis son actividades que dirigen el proceso de medición.

**Directrices para la recopilación y el análisis**

- Siempre que sea posible, deben automatizarse la recopilación de datos y su análisis.
- Debe aplicarse técnicas estadísticas válidas para establecer relaciones entre los atributos internos del producto y las características de calidad externas.
- Para cada métrica deben establecerse recomendaciones para la interpretación.

### Paradigma OPM

Es una técnica para identificar métricas significativas aplicables en cualquier parte del proceso de software. El OPM (Objetivo – Pregunta – Métrica) destaca la necesidad de:

- Establecer un **objetivo** de medición explícito que sea específico para la actividad del proceso o las características del producto que se está evaluando.
- Definir un conjunto de **preguntas** que deben responderse con el fin de alcanzar el objetivo.
- Identificar **métricas** bien formuladas que ayuden a responder esas preguntas.

### Atributos de las métricas efectivas del software

- **Simples y calculables:** Debe ser relativamente fácil aprender a derivar la métrica y su cálculo no debe consumir demasiado tiempo o esfuerzo.
- **Empírica e intuitivamente persuasivas:** La métrica debe satisfacer las nociones intuitivas del ingeniero acerca del atributo del producto que se está construyendo.
- **Consistentes y objetivas:** La métrica siempre debe arrojar resultados que no permitan ninguna ambigüedad.
- **Consistentes en el uso de unidades y dimensiones:** El cálculo debe emplear medidas que no lleven a combinaciones extrañas de unidades.
- **Independientes del lenguaje de programación:** Las métricas deben basarse en le modelo de análisis o diseño, o en la estructura del programa.
- **Proporcionar retroalimentación efectiva:** Es decir, la métrica debe llevar a un producto final de la más alta calidad.

### Tipos de métricas

- **Métricas para el modelo de análisis:** Examinan el modelo de análisis con la intención de predecir el "tamaño" del sistema resultante, lo cual a veces sirve como indicador de la complejidad del diseño y del esfuerzo de codificación, integración y prueba.
  Estas métricas incluyen:

  - **Funcionalidad entregada:** Proporciona una medida indirecta de la funcionalidad que se empaqueta con el software.
  - **Tamaño del sistema:** Mide el tamaño general del sistema, definido desde el punto de vista de la información disponible como parte del modelo de análisis.
  - **Calidad de la especificación:** Proporciona una indicación del grado en que se ha completado la especificación de los requerimientos.

- **Métricas para el modelo de diseño:** Cuantifican los atributos del diseño de manera tal que le permiten al ingeniero de software evaluar la calidad del diseño. La métrica incluye:

  - **Métricas arquitectónicas:** Proporcionan un indicio de la calidad del diseño arquitectónico, teniendo en cuenta la estructura arquitectónica y la efectividad de módulos o componentes dentro de la arquitectura.
  - **Métricas al nivel de componente:** Se concentran en las características internas de un componente de software e incluyen medidas de cohesión, acoplamiento y complejidad del módulo.
  - **Métricas de diseño de la interfaz:** Se concentran en la calidad, la cohesión y la facilidad de uso.

  - **Métricas especializadas en diseño orientado a objetos:** Miden características de clases además de las correspondientes a comunicación y colaboración.

- **Métricas para el código fuente:** Miden el código fuente y se usan para evaluar su complejidad, además de la facilidad con que se mantiene y prueba.

  - **Métricas de Halstead:** Son muy buenas aunque son controversiales. Proporcionan medidas únicas de un programa de cómputo.
  - **Métricas de complejidad:** Miden la complejidad lógica del código fuente.
  - **Métricas de longitud:** Proporcionan un indicio del tamaño del software.

- **Métricas para pruebas:** Casi todas estas métricas se centran en el proceso de prueba, no en las características técnicas de las mismas. En general quienes aplican las pruebas deben depender de las métricas de análisis, diseño y código como guía para el diseño y la ejecución de casos de prueba. Incluyen:

  - **Métricas de cobertura de instrucciones y ramas:** Lleva al diseño de casos de prueba que proporcionan cobertura del programa.
  - **Métricas relacionadas con los defectos:** Se concentran en encontrar defectos y no en las propias pruebas.
  - **Efectividad de la prueba:** Proporcionan un indicio en tiempo real de la efectividad de las pruebas aplicadas.
  - **Métricas en el proceso:** Son métricas relacionadas con el proceso que se determinan a medida que se aplican las pruebas.