Introducción

En esta unidad, nos proponemos explicarte el proceso de desarrollo de software. Te contaremos qué es el ciclo de vida y te presentaremos los principales modelos de desarrollo que se utilizan actualmente. También te contaremos las disciplinas que intervienen en la Ingeniería de software.
Finalmente te contaremos qué es la configuración de software y cómo planear y realizar las pruebas del software.

Como hemos visto en el módulo **Técnicas de Programación**, un sistema puede llegar a ser muy complejo en funcionalidad y en cantidad de código fuente. Para poder administrar la complejidad de tales sistemas, es necesario contar con modelos de procesos y tecnologías de software apropiadas.
En esta unidad, te presentaremos distintos modelos de procesos y veremos cuáles son sus características, ventajas y desventajas.

¡Comenzamos!

El proceso de desarrollo de software

Seguramente habrás oído hablar muchas veces de proceso histórico, de proceso productivo, proceso lógico, proceso natural, etc.

Entonces comenzamos preguntándonos....

¿Qué es para vos un proceso?

Hacé clic en el botón para ver la respuesta.

Habrá muchas formas de definir este concepto, pero seguramente tu definición se asemejará a ésta que te proponemos:

“Un proceso es un conjunto de actividades planificadas que implican la participación de un número de personas y de recursos materiales coordinados para conseguir un objetivo previamente identificado”.

Un proceso entonces define **quién hace qué, cuándo y cómo** lo hace para poder alcanzar un objetivo.

En este módulo estudiaremos el proceso de desarrollo ya que, en general, el éxito de las organizaciones depende en gran medida de la definición y el seguimiento adecuado de sus procesos.

El proceso de desarrollo de software

En nuestro caso, será de interés una empresa que se dedique al desarrollo de software y para eso requerirá procesos especializados que abarquen desde la creación hasta la administración de un sistema de software.

El Ciclo de Vida de Desarrollo de Software

Como te comentamos en la Unidad 1 de la materia Técnicas de Programación, el desarrollo de un sistema se realiza durante todo el ciclo de vida, que es el período de tiempo que se extiende desde la idea original del problema a resolver hasta el mantenimiento y desarrollo de las mejoras.

(hacé clic sobre cada etapa para ver su descripción)

Análisis del problema

En esta etapa se debe determinar cuál es el problema a resolver y los límites y alcances que tendrá el software que lo resolverá. Es el momento de reunirse con quien nos solicita el programa para saber cuáles son los requerimientos.

Especificación del software

En este momento los profesionales de sistemas se encargan de definir las entradas y las salidas del software y, qué restricciones tendrán los datos. También se describen los componentes que se deberán desarrollar, qué características y comportamiento tendrán y cómo estarán relacionados.

Desarrollo del software

Corresponde al proceso de construcción de software propiamente dicho. Es en esta etapa en donde los programadores escriben el código fuente utilizando algún lenguaje de programación.

Verificación del software

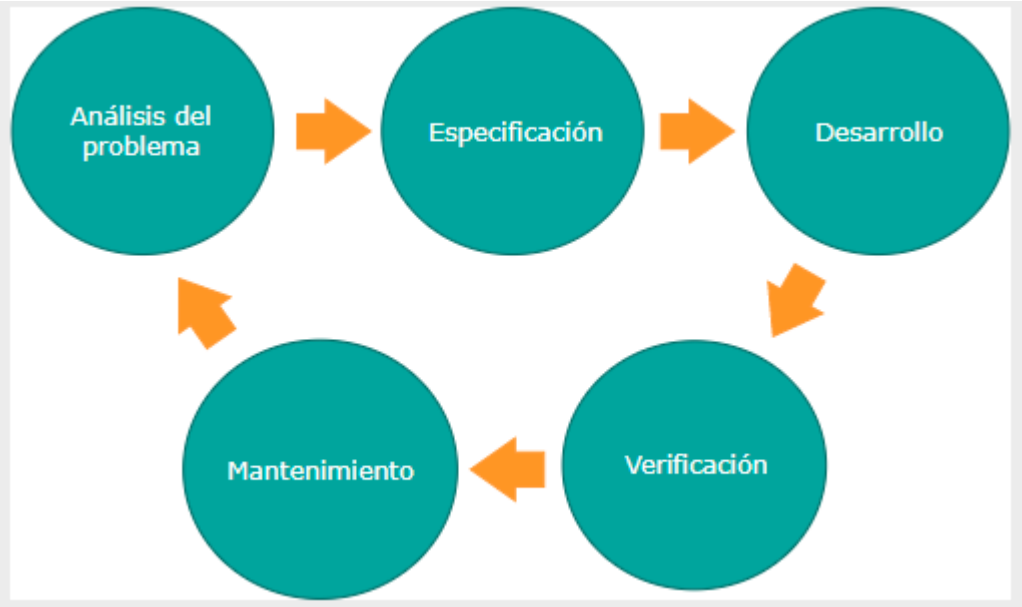
Una vez que el software está desarrollado se debe probar para verificar que responde a las definiciones y no tiene errores.

Mantenimiento del software

Todo sistema debería tener mantenimiento ya que siempre habrá que realizar alguna modificación, agregando nueva funcionalidad o bien cambiando alguna característica porque se ha modificado alguna especificación.

El siguiente esquema nos recuerda el ciclo de vida del software:

El Ciclo de Vida de Desarrollo de Software



Modelos de Desarrollo de Software

Como seguramente recordarás, no existe un único modelo de **Ciclo de Vida**. Más allá de que los distintos modelos tendrán diferentes características, todo proceso debe cubrir los siguientes objetivos.

- Definir las actividades a realizar y en qué orden.
- Establecer criterios de transición para pasar a la fase siguiente.
- Proporcionar puntos de control para la gestión del proyecto.
- Asegurar la consistencia con el resto de los sistemas de información.

A continuación, te presentamos algunos de los modelos de desarrollo más importantes.

1. Modelo de cascada

El **modelo de cascada** original se desarrolló entre las décadas de los años 60 y 70. Se define como una secuencia de actividades o etapas, donde la estrategia principal es seguir el progreso del desarrollo de software hacia puntos de revisión bien definidos mediante entregas programadas con fechas precisas. En este modelo, no se muestra una etapa específica de documentación, ya que ésta se lleva durante todo el proceso de desarrollo.

En el modelo original se planteaba que cada actividad debía completarse antes de poder continuar con la siguiente actividad. Sin embargo, en una revisión posterior, se extendió el modelo permitiendo regresar a actividades anteriores.

Este modelo fue muy aceptado debido a que las etapas son lógicas y se comprenden fácilmente por los usuarios no técnicos. Pero este modelo no explica cómo modificar un resultado, en especial teniendo en cuenta lo difícil que puede llegar a especificar toda la funcionalidad, y los requisitos de un sistema desde el comienzo y que estos se mantengan estables durante todo el proceso.

1. Modelo de cascada

Otra desventaja es que toma demasiado tiempo en tener un resultado tangible para el usuario y retrasa la detección de errores hacia las etapas finales del desarrollo.

El modelo establece las siguientes etapas para el desarrollo de software.

(hacé clic sobre cada etapa para ver su descripción)

Análisis de requisitos

Consiste en la recopilación de los requisitos del software. Se debe comprender el ámbito de información del software, así como la función, el rendimiento y las interfaces requeridas. Estos requisitos se deben documentar y revisar de tal manera que los entiendan tanto los usuarios como el equipo de desarrollo del software. En esta fase se desarrollará el documento de requisitos del software que consistirá en una especificación precisa y completa de lo que debe hacer el sistema.

Diseño

Consiste en descomponer y organizar el sistema en elementos componentes que puedan ser desarrollados por separado. El resultado del diseño es la colección de especificaciones de cada elemento componente. En esta fase se desarrollará el Documento del diseño del Software que será una descripción de la estructura global del sistema.

Implementación

En esta fase se traduce el diseño a un lenguaje legible para una computadora. También se harán las pruebas o ensayos necesarios para garantizar que dicho código funciona correctamente. La documentación de esta fase será el Código fuente.

Verificación

Consiste en probar el sistema completo para garantizar el funcionamiento correcto del conjunto antes de ser puesto en producción. Aquí tendremos el Sistema Software ejecutable.

Mantenimiento

Puede ocurrir que durante el uso del sistema sea necesario realizar cambios para corregir errores que no han sido detectados en las fases anteriores o bien para introducir mejoras. Tendremos que hacer un Documento de cambios ante cualquier modificación. En todas estas fases la verificación y validación se han de tener en cuenta. La verificación consiste en comprobar que el software que se está desarrollando cumple los requisitos y la validación lo que hace es comprobar que las funciones del software son las que el usuario desea.

2. Modelo incremental

El **modelo incremental** consiste en el desarrollo inicial de la arquitectura completa del sistema, seguida de incrementos y versiones parciales. Cada incremento tiene su propio ciclo de vida, típicamente siguiendo el modelo de cascada. Los incrementos pueden construirse de manera serial o paralela dependiendo de la naturaleza de la dependencia entre versiones y recursos.

Cada incremento agrega funcionalidad adicional o mejorada sobre el sistema. Conforme se completa cada etapa, se verifica e integra la última versión con las demás versiones ya completadas del sistema. Durante cada incremento, el sistema se evalúa con respecto al desarrollo de versiones futuras.

Las actividades se dividen en procesos y subprocesos, dando lugar al término fábrica de software (en inglés, software factory).

¿Ventajas?

Este modelo tiene como ventaja que permite que la administración del proyecto sea más simple en incrementos pequeños. Además, al tener una menor funcionalidad por incremento, cada etapa es más simple de comprender y de probar. Otra ventaja es que el usuario tendrá una versión más temprana del sistema, ya que se encontrará con una pequeña funcionalidad para ir probando y verificando.

Con respecto a los cambios, este modelo permite resolverlos en un mismo período de tiempo. De esta manera el modelo es tolerante a modificaciones en los requerimientos, algo que hoy en día sucede con mucha frecuencia.

3. Modelo evolutivo

Ahora te presentamos el **modelo evolutivo**, el cual es una extensión del modelo incremental. En este modelo, los incrementos se hacen de manera secuencial, en lugar de en paralelo.

Desde el punto de vista del cliente, el sistema evoluciona según vayan siendo entregados los incrementos. Desde el punto de vista del desarrollador, los requerimientos que son claros al principio del proyecto determinan el incremento inicial, mientras que los incrementos para cada uno de los siguientes ciclos de desarrollo se definirán a través de la experiencia de los incrementos anteriores. Este modelo considera que el desarrollo de sistemas es un proceso de cambios progresivos mediante deltas (cambios) de especificación de requerimientos.

Este modelo también es conocido como **Desarrollo rápido de aplicaciones** (RAD en inglés, rapid application development) basado en prototipos.

Te contamos que un prototipo de software es un medio para especificar los requerimientos y comunicarlos al usuario. El prototipo será una representación limitada de un producto (en nuestro caso, el software) que permitirá que el usuario lo pueda probar en situaciones reales.
De esta manera se crea un proceso de diseño de iteración que genera calidad.

3. Modelo evolutivo

Un prototipo puede ser cualquier cosa, desde un trozo de papel con sencillos dibujos a un módulo de software.

Resumiendo, te podemos decir que:

Los prototipos apoyan la evaluación del producto, clarifican los requisitos del usuario y definen alternativas.

Ahora te contamos los beneficios e inconvenientes de aplicar este modelo basado en prototipos:

Como ventajas te podemos decir que:

- Los requisitos de los usuarios son más fáciles de determinar y la Implementación del sistema será más sencilla debido a que los usuarios conocen lo que esperan.
- Los sistemas se desarrollan más rápidamente
- Facilita la comunicación con los usuarios ya que los prototipos son comprendidos tanto por el usuario como por el analista.

(Hacé clic para conocer las ventajas...)

Dentro de los inconvenientes te podemos mencionar que:

- Puede crear falsas expectativas en el usuario ya que puede ver el prototipo como si fuera el producto final.
- Puede darse una fuerte intromisión de los usuarios finales en el momento del desarrollo.
- Pueden producirse inconsistencias entre el prototipo y el sistema final.
- El paradigma de prototipos no es apropiado para proyectos grandes y de larga duración ni para aplicaciones pequeñas (menos de un mes), siendo óptimo en aplicaciones y proyectos cuya duración esté fijada entre 3 y 5 meses.

(Hacé clic para conocer los inconvenientes...)

4. Modelo espiral

Este modelo fue desarrollado en la década del 80 y surgió como una extensión del modelo de cascada. El **modelo espiral** se basa en una estrategia para reducir el riesgo del proyecto en áreas de incertidumbre, como por ejemplo tener requerimientos iniciales incompletos e inestables.

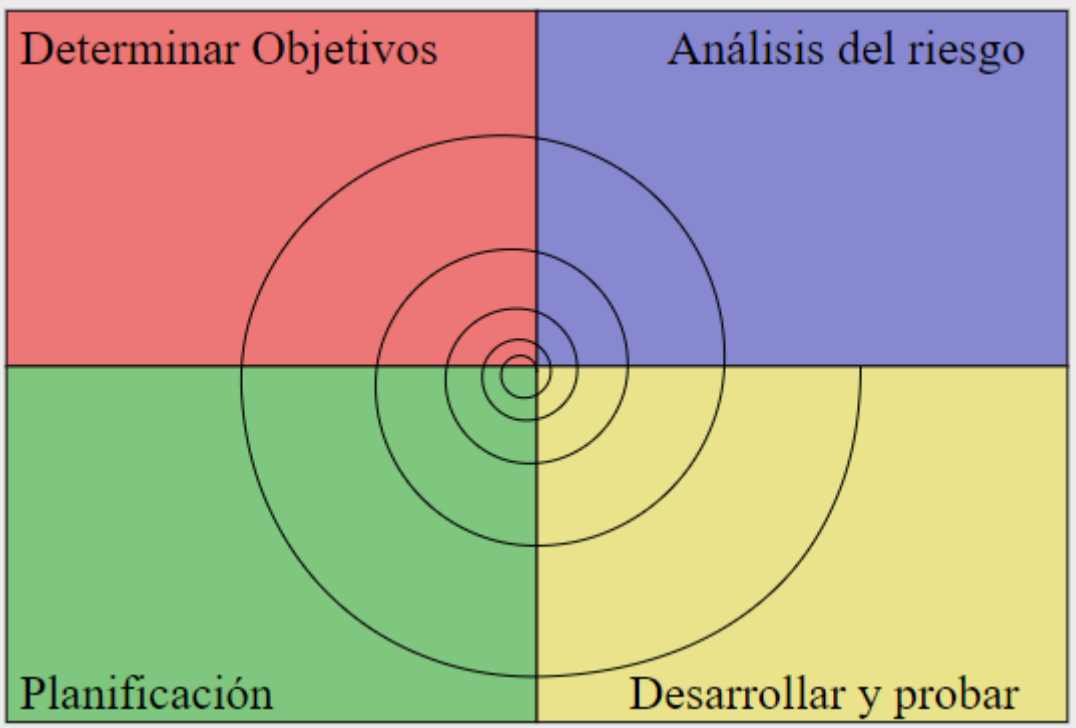
El modelo enfatiza ciclos de trabajo, cada uno de los cuales estudia el riesgo antes de proceder al siguiente ciclo. Cada ciclo comienza con la identificación de los objetivos, soluciones alternas y restricciones asociadas con cada alternativa y, finalmente, se procede a su evaluación.

4. Modelo espiral

Cuando se encuentra que existe cierta incertidumbre, se utilizan diversas técnicas para reducir el riesgo de las distintas alternativas. Cada ciclo del modelo espiral termina con una revisión que discute los logros actuales y los planes para el siguiente ciclo.

Esta secuencia de pasos se compone de **4 actividades**:

1. Determinar Objetivos.
2. Análisis del riesgo.
3. Desarrollar y probar.
4. Planificación.



Los **ciclos se repiten en forma de espiral**, comenzando desde el centro. Se suele interpretar como que dentro de cada ciclo de la espiral se sigue un Modelo Cascada, pero no necesariamente debe ser así.

Como ventajas te podemos decir que

- Este modelo se puede ajustar a un amplio rango de opciones.
- Su orientación al riesgo evita, y hasta elimina, muchas de las posibles dificultades.
- Se centra en la eliminación de errores.
- Permite incorporar objetivos de calidad en el desarrollo software.
- Se adapta bien al diseño Orientado a Objetos.

(Hacé clic para conocer las ventajas...)

4. Modelo espiral

Dentro de los inconvenientes podemos mencionar que

• Depende de manera excesiva de la experiencia que se tenga en identificar y evaluar riesgos.

• Es difícil adaptarlo al software contratado debido a la poca flexibilidad y libertad que posee un software de estas características.

(Hacé clic para conocer los inconvenientes...)

Disciplinas que conforman la Ingeniería de Software

Durante el proceso de desarrollo de software, como te habrás dado cuenta, intervienen muchas personas con distintos roles que realizan actividades diferentes.

¿Qué son las disciplinas entonces? ¿Por qué hablamos de disciplinas que conforman el desarrollo de Software?

Las **disciplinas** son precisamente eso: u**n conjunto de actividades relacionadas con un área de atención dentro de todo el proyecto.**

De esta manera podemos observar que durante el ciclo de desarrollo intervienen más de una disciplina. Entre las principales disciplinas que intervienen podemos mencionarte las siguientes:

(hacé clic sobre cada disciplina para ver su descripción)

Modelado de Negocios

El propósito de esta disciplina es entender el contexto del cliente, y la aplicación que tendrá el sistema dentro del mismo. Las actividades del modelado de negocios más comunes son la realización de:

• Modelado de contexto. Esto es, mostrar cómo tiene cabida tu sistema dentro del ambiente general.

• Modelado de requerimientos de negocio de alto nivel (casos de uso).

• Glosario definiendo los términos críticos del negocio.

• Modelado de dominio, describiendo las entidades principales del negocio.

Requerimientos

Aquí se aplican métodos de ingeniería a los requerimientos del proyecto, incluyendo la identificación, modelado y documentación de dichos requerimientos. El entregable principal de esta disciplina es el documento donde se especifican los Requerimientos del Software.

4. Modelo espiral

El rol principal lo cubre el Analista de Sistemas, quien se encarga de descubrir todos los casos de uso de requerimientos.

Análisis y Diseño

El propósito de esta disciplina es crear una arquitectura robusta para el sistema basada en los requerimientos del cliente, transformar dichos requerimientos en un diseño, y asegurarse de que los problemas del ambiente dentro del cual se implementará el sistema se reflejen en el diseño.

En esta disciplina interviene el Arquitecto de Software, quien decide las tecnologías para toda la solución. Y el Diseñador, quien detalla el análisis y diseño de un grupo de casos de uso.

Implementación

Esta disciplina define la organización del código, implementa el diseño de elementos, prueba los componentes desarrollados como unidades e integra los resultados individuales en un sistema ejecutable. En esta disciplina es donde aparece el Desarrollador, quien tiene el plan de construcción que muestra qué entidades o clases se integrarán unas a otras. También se encargará de codificar un grupo de clases u operaciones de clases.

Pruebas

Esta disciplina es la encargada de evaluar todos los componentes del sistema y de asegurar la calidad del producto desarrollado. Las actividades que se llevan a cabo son:

- Encontrar fallas de calidad en software y documentarlas.
- Dar retroalimentación sobre la calidad percibida en el software.
- Validar y probar que se han cubierto los requerimientos y el diseño del sistema.

Es acá donde interviene el Administrador de Pruebas, quien se asegura de que las pruebas se completen y sean realizadas bajo las especificaciones correctas. También interviene el Analista de Pruebas, quien selecciona qué probar; el Diseñador de Pruebas, quien decide qué pruebas deben ser automatizadas y cuáles manuales; y el tester, quien ejecuta una prueba específica.

Transición

En esta disciplina se describen las actividades asociadas con el aseguramiento de la entrega y disponibilidad del producto de software hacia el usuario final.

Interviene el Administrador de transición, quien se encarga de prever que todas las unidades del sistema tengan una transición exitosa. Además, intervienen otros roles como Escritor Técnico, Desarrollador de Curso, Artista Gráfico, quienes crean los materiales detallados para asegurarse de una transición exitosa.

Administración y configuración del cambio

4. Modelo espiral

Esta disciplina consiste en controlar los cambios y mantener la integridad de los productos que incluye el proyecto. Interviene el Administrador de Configuración, quien prepara el ambiente, las políticas; el Administrador de Control de Cambio, quien establece un proceso de control de cambio; y el Administrador de Control de Cambio, quien revisa y administra las solicitudes de cambios.

Administración de Proyectos

Es la disciplina que provee un marco de trabajo para administrar los proyectos intensivos de software, además de guías prácticas para el desarrollo del mismo. El Administrador del Proyecto es quien crea los casos de negocio y un plan general para todo el proyecto. También planea, rastrea y administra los riesgos para cada iteración.

Configuración de Software

Evolución del software

Con la experiencia que habrás tenido en el cursado de los otros módulos, habrás notado que el software es un producto que se realiza en base a una especificación y que luego puede ser modificado, ya sea porque esas especificaciones cambian o bien por algún otro motivo incluso ajeno al cliente.

De esta manera, debemos agregar una **nueva etapa de Mantenimiento al proyecto** una vez que el producto ha sido terminado. Esta Fase de Mantenimiento consiste en repetir o realizar parte de las actividades de las fases anteriores para introducir cambios en una aplicación del software ya entregada al cliente.

El mantenimiento lo podemos clasificar en **tres tipos distintos.** Te contamos a continuación cuáles son:

Mantenimiento Correctivo

Tiene como finalidad corregir errores en el software que no han sido detectados y eliminados durante el desarrollo inicial del mismo.

Mantenimiento Adaptativo

Se produce en aplicaciones que se están utilizando desde hace mucho tiempo, de manera que los elementos básicos HW y SW que constituyen la plataforma o entorno en que se ejecutan evolucionan, modificándose parcialmente la interfaz ofrecida a las aplicaciones que corren sobre ellos.

Mantenimiento Perfectivo

Es necesario para obtener versiones mejoradas del producto que permitan mantener o aumentar su éxito.

Configuración de Software

El **mantenimiento**, por lo tanto, es una tarea que puede durar mucho tiempo y abarcar distintas etapas, es por eso que se necesita una buena gestión sobre los cambios y el testing. En las siguientes secciones te contaremos en más detalle cómo realizar estas gestiones.

Gestión de la configuración

Seguramente te estarás preguntando **qué es configurar software.**
Te contamos que la configuración es la manera en que diversos elementos se combinan para construir un software bien organizado.

Para mantener bajo control la configuración del software nos vamos a apoyar en dos técnicas que son:

Control de versiones: consiste en almacenar de forma organizada las sucesivas versiones de cada elemento de la configuración.

Control de cambios: consiste en garantizar que las diferentes configuraciones del software se componen de elementos compatibles entre sí.

Recordemos que estamos trabajando bajo una metodología de desarrollo. Debido a eso es preciso llevar un control y registro de los cambios con el fin de reducir errores, aumentar la calidad y productividad.

El objetivo de la **gestión de la configuración** es el de mantener la integridad de los productos que se obtienen a lo largo del desarrollo del sistema, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan.

Esta gestión de la configuración se realiza durante todas las actividades asociadas al desarrollo del sistema.

Desde el punto de vista del usuario y del desarrollador se consideran como elementos componentes de la configuración todos los que intervienen en el desarrollo. Estos elementos serán, por tanto:

- **Documentos del desarrollo.**
- **Código fuente de los modelos.**
- **Programas, datos y resultados de las pruebas.**
- **Manuales del usuario.**
- **Documentos de mantenimiento.**
- **Prototipos.**
- **Código fuente de los modelos.**
- **Programas, datos y resultados de las pruebas.**
- **Manuales del usuario.**
- **Documentos de mantenimiento.**
- **Prototipos.**

Gestión de cambios

Más allá del objetivo concreto del mantenimiento, las actividades a realizar durante el mismo son básicamente la realización de cambios significativos sobre el software ya realizado.

Podemos distinguir **2 enfoques diferentes** en cuanto a la gestión de cambios, en función del mayor o menor grado de modificación del producto. De esta manera tenemos:

- Si el cambio a realizar afecta a la mayoría de los componentes de software se puede plantear como un nuevo desarrollo y aplicar un nuevo ciclo de vida utilizando lo ya desarrollado (reutilización).

- Si el cambio afecta a una parte localizada del producto entonces se puede organizar como una simple modificación de algunos elementos.

Normas y estándares

Al igual que muchas otras actividades de ingenierías que están reguladas, en la ingeniería de software también hay normas, algunas han sido recogidas por organizaciones internacionales y establecidas como estándares.

Te mostramos ejemplos:

- **ISO:** Son las siglas del organismo internacional de normalización. El organismo español que lo integra es AENOR. Entre sus normas de Ingeniería del software de mayor nivel se encuentran las ISO-9001 que establecen los criterios que han de cumplir las empresas que desarrollen software para obtener certificaciones de determinados niveles de garantía de calidad para su producción.

- **MÉTRICA 3:** Ofrece a los organismos oficiales un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software.

Testing de Software

Como te hemos estado contando durante toda esta Unidad, la calidad de un software se determina por la etapa que sigue a su desarrollo. Un sistema no finaliza cuando termina de desarrollarse, ya que, en el modelo de ciclo de vida, las actividades de revisión y pruebas tienen como objetivo controlar la calidad del producto.

Testing de Software

Factores de calidad

Existe un esquema de mediciones de la calidad del software establecido en 1977. Este modelo se utiliza para especificar software de calidad y está basado en valoraciones a tres niveles diferentes.

Factores. Constituyen el nivel superior y son la valoración propiamente dicha. Describen la visión externa del software, es decir, cómo es visto por el usuario.

Criterios. Describen la visión interna del software, es decir, cómo es visto por el desarrollador.

Métricas. Están en el nivel inferior, son mediciones puntuales de determinados atributos o características del producto.

Entre los factores de calidad te podemos mencionar los siguientes:

• **Corrección**: Es el grado en que un software cumple con sus especificaciones. Se puede definir como el porcentaje de requisitos que se cumplen adecuadamente.

• **Fiabilidad**: Es el grado de ausencia de fallos durante la operación del software; puede estimarse como la cantidad de fallos producidos o tiempo durante el que permanece inutilizable.

• **Eficiencia**: Relación entre la cantidad de resultados suministrados y los recursos requeridos durante la operación.

• **Seguridad**: Dificultad para el acceso a los datos u operaciones por parte de personal no autorizado

• **Facilidad de uso**: Es la inversa del esfuerzo requerido para aprender a usar un producto software y poder utilizarlo adecuadamente.

• **Mantenibilidad**: Facilidad para corregir el software.

• **Flexibilidad**: Facilidad para modificar el software.

• **Facilidad de prueba**: Es la inversa del esfuerzo requerido para probar un software y comprobar su corrección o fiabilidad

• **Transportabilidad**: Facilidad para adaptar el software a una plataforma de hardware y Sistema Operativo diferente a aquella para la que fue desarrollado inicialmente.

• **Reusabilidad**: Facilidad para emplear parte del software en otros desarrollos posteriores.

• **Interoperatividad**: Facilidad del software de trabajar en combinación con otros productos.

Testing de Software

Plan de garantía de la calidad

La calidad del software se consigue mediante una buena organización del desarrollo. Como parte del proceso de desarrollo se deberá generar un documento llamado Plan de garantía de calidad del software. Este documento deberá contemplar los siguientes aspectos.

- Organización de los equipos de personas y la dirección y seguimiento del desarrollo.
- El modelo de ciclo de vida a seguir.
- Documentación requerida.
- Revisiones y auditorías que se llevarán a cabo durante el desarrollo para garantizar que las actividades y documentos son correctos y aceptables.
- Organización de las pruebas que se realizarán sobre el producto software a distintos niveles.
- Organización de las etapas de mantenimiento.

Revisiones

Te contamos que una revisión consiste en inspeccionar el resultado de una actividad para determinar si es aceptable o no.

A continuación, te mencionamos algunas recomendaciones que podrás seguir para formalizar las revisiones.

- Deben ser realizadas por un grupo de personas, no por una sola
- El grupo que realice la revisión debe ser reducido (3-5 personas).
- No deben ser realizadas por los autores del software y así poder garantizar la imparcialidad del criterio.
- Se debe revisar el producto, pero no al productor ni al proceso de producción.
- Si la revisión tiene que decidir la aceptación o no del producto, se debe establecer previamente una lista formal de comprobaciones a realizar
- Debe levantarse acta o minuta de la reunión de la revisión. Este documento puede considerarse como el producto de la revisión.

Pruebas

Testing de Software

Las pruebas consisten en hacer funcionar un software o una parte de él en condiciones determinadas y comprobar si los resultados son los correctos. Por lo tanto, el objetivo de las pruebas será descubrir los errores que pueda contener el software probado.

Es importante que comprendas que las pruebas no permiten garantizar la calidad de un producto, sino la funcionalidad que se está probando.