IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

Desarrollo de contenido

Unidad 2

Ingeniería de Software

Unidad 2: Lenguaje de Modelado Unificado y Pruebas de Software

Tecnología en Desarrollo de Software

ORD No 74 de 2017 · VIGILADA MinEducación

Tabla de Contenido

Glosario
Unidad 2

A

- **AGP**
  Puerto acelerador de gráficos. Permite correr velozmente archivos gráficos tridimensionales.

- **AI**
  Artificial Intelligence - Inteligencia Artificial.

- **Algoritmo**
  Conjunto de reglas bien definidas para la resolución de un problema. Un programa de software es la trascripción, en lenguaje de programación, de un algoritmo.

- **Amazon**
  Librería mundial online. Ofrece más de un millón de títulos a través de Internet.

- **AMIBIOS**
  Una de las marcas de BIOS más usadas.

- **ASCII**
  American Standard Code of Information Interchange: código normalizado estadounidense para el intercambio de la información. Código que permite definir caracteres alfa- numéricos; se lo usa para lograr compatibilidad entre diversos procesadores de texto. Se pronuncia "aski".

- **Attachement**
  Archivo adjunto.

- **AVI**
  Formato de Microsoft para archivos de audio y video.

B

- **BASIC**
  Beginner's All-Purpose Symbolic Instruction Code: Código de Instrucción Simbólica Multipropósito para Principiantes. Lenguaje de programación, creado en 1963, sencillo y muy difundido.

- **Blind carbon copy**
  Función que permite mandar un mensaje de e-mail a más de un destinatario. A diferencia de la función c.c, el nombre de los

destinatarios no aparece en el encabezado.

- BIOS
  Sistema básico de ingreso/salida de datos. Conjunto de procedimientos que controla el flujo de datos entre el sistema operativo y dispositivos tales como el disco rígido, la placa de video, el teclado, el mouse y la impresora.

- Bit
  Abreviatura de binary digit (dígito binario). El bit es la unidad más pequeña de almacenamiento en un sistema binario dentro de una computadora.

- Bookmark
  Anotación, en el navegador, de una dirección de Internet que se almacena para agilizar su uso posterior. En el programa Internet Explorer, se llama "Favoritos".

- Boot (butear)
  Cargar el sistema operativo de una computadora.

- Bps
  Bits por segundo.

- Browser
  Navegador.

- Buffer
  Area de la memoria que se utiliza para almacenar datos temporalmente durante una sesión de trabajo.

C

- Carbon copy
  Copia de papel carbónico, como alusión al antiguo método para copiar un documento. Función que permite mandar un mensaje de e-mail a más de un destinatario. Véase bcc.

- Caso de uso
  Descripción de los requisitos de comportamiento de un sistema y su interacción con un usuario.

- Chip
  Abreviatura de "microchip". Circuito muy pequeño, compuesto por miles de millones de transistores impresos sobre una oblea de silicio.

- Ciberespacio
  Espacio virtual, no geográfico, determinado por la interconexión de personas a través de redes telemáticas. El término fue acuñado por el escritor norteamericano William Gibson en su novela de ficción científica Neuromante, publicada en 1984. Gibson inició el movimiento llamado "cyberpunk".

- Clipboard
  Portapapeles.

- Clave pública y clave privada
  Esquema de encriptación en el que cada persona tiene dos claves: la pública y la privada. Los mensajes se encriptan usando la clave pública del destinatario y sólo pueden ser descifrados usando su clave privada.

- Cursor
  Símbolo en pantalla que indica la posición activa: por ejemplo, la posición en que aparecerá el próximo carácter que entre.

D

- Data
  Datos, información.

- Data entry
  Ingreso de datos. Proceso de ingresar datos a una computadora para su procesamiento.

- Database
  Base de datos.

- Delete
  Borrar; eliminar; anular.

- Directorio (directory)
  Grupo de archivos relacionados entre sí que se guardan bajo un nombre.

- DirectX
  Recurso para mejorar el rendimiento en gráficos, sonidos, 3D, sitios web y juegos.

- Disco rígido
  Soporte giratorio de almacenamiento en forma de placa circular revestida por una película magnética. Los datos se graban en pistas concéntricas en la película.

- Display
  Unidad de visualización; monitor; pantalla.
  disquete: cartucho de plástico rígido para el almacenamiento de datos. Hay de 3 ½ pulgadas de lado, que almacena hasta 1,44 MB, y de5 ¼ pulgadas (en desuso). Véase también zip drive.

- DVD
  Digital Versatile Disc: Disco Versátil Digital. Disco que posee gran capacidad de al-macenamiento y sirve también para almacenar películas.

- Dynamic HTML
  Variante del HTML (Hyper TextMark-up Language) que permite crear páginas web más animadas.

E

- Emulación
  Emulation. Proceso de compatibilización entre computadoras mediante un software.

- Encapsulamiento
  Como proceso, la encapsulación significa el acto de encerrar uno o más elementos dentro de un contenedor. Una técnica de desarrollo de software que consiste en aislar una función del sistema o un conjunto de datos y operaciones en esos datos dentro de un módulo y proporcionar especificaciones precisas para el módulo.

- Encoder
  Programa que convierte un archivo wave en un archivo MP3. El programa que reproduce los archivos MP3 se llama player.

- Encriptar
  Proteger archivos expresando su contenido en un lenguaje cifrado. Los lenguajes cifrados simples consisten, por ejemplo, en la sustitución de letras por números.

- Enlace
  Link.

- Ethernet
  Tecnología para red de área local. Fue desarrollada originalmente por Xerox y posteriormente por Xerox, DEC e Intel. Ha sido aceptada como estándar por la IEEE.

F

- FAQ
  Frequently-asked questions. Las preguntas más frecuentes (y sus respuestas) sobre el tema principal de un sitio web.

- Fibra óptica
  Tecnología para transmitir información como pulsos luminosos a través de un conducto de fibra de vidrio. La fibra óptica transporta mucha más información que el cable de cobre convencional. La mayoría de las líneas de larga distancia de las compañías telefónicas utilizan la fibra óptica.

- Flexibilidad
  La capacidad del sistema para reestructurarse en respuesta a cambios o presiones externas.

- Freeware
  Software de distribución libre. A diferencia del shareware, es total-mente gratuito.

- FTP

File Transfer Protocol: Protocolo de Transferencia de Archivos. Sirve para enviar y recibir archivos de Internet.

- Función
  Un sistema de resultados que contribuyen a las metas u objetivos. Para tener una función, un sistema debe ser capaz de proporcionar el resultado a través de dos o más combinaciones diferentes de comportamiento elemental. Una acción, una tarea o una actividad realizada para lograr el resultado deseado. Un área de trabajo amplia que abarca múltiples disciplinas relacionadas (por ejemplo, Ingeniería, Finanzas, Recursos Humanos, etc.). Una función se define mediante la transformación de flujos de entrada en flujos de salida, con un rendimiento definido.

G

- Gestión de programas
  La aplicación de conocimientos, habilidades, herramientas y técnicas a un programa para cumplir con los requisitos del programa y para obtener beneficios y control que no están disponibles al administrar proyectos individualmente.

- GPS
  Global Positioning System. Sistema de localización global compuesto por 24 satélites. Se usa, por ejemplo, en automóviles, para indicarle al conductor dónde se encuentra y sugerirle rutas posibles.

- Gusano
  Programa que se copia a sí mismo hasta ocupar toda la memoria. Es un virus que suele llegar a través del correo electrónico, en forma de archivo ad- junto.

H

- Hosting
  Alojamiento. Servicio ofrecido por algunos proveedores, que brindan a sus clientes (individuos o empresas) un espacio en su servidor para alojar un sitio web.

I

- Internet2
  Proyecto de interconexión de más de 100 universidades estadounidenses. El objetivo es desarrollar una red de altísima velocidad para la educación y la investigación.

- Intranet
  Red de redes de una empresa. Su aspecto es similar al de las páginas de Internet.

J

- Jitter
  Variación en la cantidad de latencia entre paquetes de datos recibidos.

- Joystick
  Dispositivo para manejar ciertas funciones de las computadoras, especialmente en juegos.

M

- Modelo
  Una representación de similitudes en un conjunto o clase de problemas, soluciones o sistemas. Cada modelo describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar esta solución un millón de veces, sin tener que hacerlo. De la misma manera dos veces.

O

- Obtención
  La función general que describe las actividades y procesos para adquirir bienes y servicios.

P

- Plan
  Item de información que presenta un curso de acción sistemático para lograr un propósito declarado, incluyendo cuándo, cómo y por quién se realizarán actividades específicas.

- Proyecto
  Un esfuerzo temporal realizado para crear un producto, servicio o resultado único. Un esfuerzo de desarrollo que consiste en actividades técnicas y de gestión con el propósito de diseñar un sistema. Esforzarse con criterios definidos de inicio y finalización para crear un producto o servicio de acuerdo con los recursos y requisitos especificados.

S

- Socket
  (Soporte) conector eléctrico, toma de corriente, enchufe. / Un socket es el punto final de una conexión. / Método de comunicación entre un programa cliente y un programa servidor en una red (véase cliente/servidor).

- Suite
  Serie, conjunto. Conjunto de programas que se comercializan en un solo paquete.

V

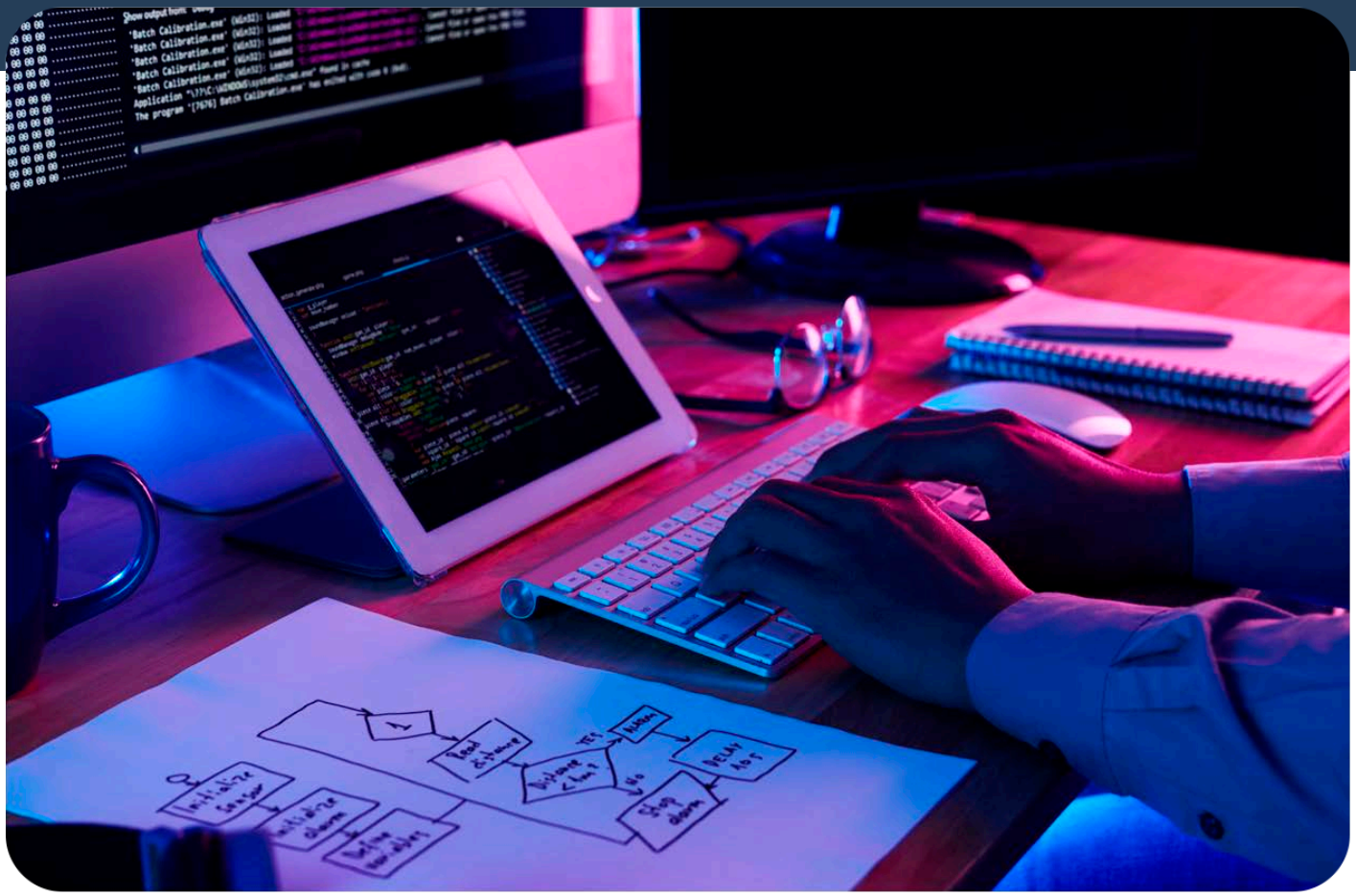
- Viable
  Un sistema viable es cualquier sistema organizado de tal manera que satisfaga las demandas de sobrevivir en un entorno cambiante.

Z

- Zip
  Formato de los archivos comprimidos.

Unidad 2.

Lenguaje de Modelado Unificado y Pruebas de Software



Introducción unidad 2

Dentro del desarrollo de productos de software y hardware, la documentación cumple un rol importante. Es la información que describe el producto a sus usuarios. Si se piensa en el desarrollo de estos productos se debe tener para ello unos buenos manuales técnicos y la información en línea (incluidas las versiones en línea de los manuales técnicos y las

descripciones de las instalaciones de ayuda). El término también se usa a veces para referirse a la fuente de información sobre el producto contenida en documentos de diseño, comentarios detallados de código, documentos técnicos y notas de la sesión de pizarra.

En esta unidad trabajaremos el UML, el cual en la actualidad es uno de los lenguajes de modelado de sistemas de software más conocidos, siendo un grupo de especificaciones en notas orientadas a Objetos, las cuales se ensamblan por distintos diagramas, que incorporan las diferentes etapas del desarrollo de un proyecto de software.

También abordaremos, la documentación que se requiere para el diseño de aplicaciones, las herramientas para la documentación de código y sobre pruebas de software. Se describen las pruebas de software, la necesidad de estas, los objetivos de pruebas de software y los principios. Todos los involucrados con las pruebas deben estar familiarizados con estos temas, por lo cual se explica diferentes técnicas de pruebas de software, como pruebas de corrección, pruebas de rendimiento, pruebas de confiabilidad y pruebas de seguridad.

Por último, analizaremos los principios básicos de las pruebas de caja negra, pruebas de caja blanca y pruebas de caja gris, examinando algunas de las estrategias que apoyan estos paradigmas y la comparación entre depuración y pruebas.

Desarrollo los temas de
la actividad de aprendizaje 2

1. TEMA 1.
UML

Unified Modeling Language (UML) es un lenguaje de modelado de propósito general. El objetivo principal de UML es definir una forma estándar de visualizar la forma en que se ha diseñado un sistema. Es bastante similar a los planos utilizados en otros campos de la ingeniería.

UML no es un lenguaje de programación, es más bien un lenguaje visual. Utilizamos diagramas UML para representar el comportamiento y la estructura de un sistema. UML ayuda a los ingenieros de software, hombres de negocios y arquitectos de sistemas con el modelado, diseño y análisis. El Object Management Group (OMG) adoptó Unified Modeling Language como estándar en 1997. Desde entonces ha sido administrado por OMG. La Organización Internacional de Normalización (ISO) publicó UML como una norma aprobada en 2005. UML se ha revisado a lo largo de los años y se revisa periódicamente.

¿Realmente necesitamos UML?

- Las aplicaciones complejas requieren la colaboración y la planificación de varios equipos y, por lo tanto, requieren una forma clara y concisa de comunicarse entre ellos.

- Los hombres de negocios no entienden el código. Por lo tanto, UML se convierte en esencial para comunicarse con los requisitos, funcionalidades y procesos del sistema que no son programadores.

- Se ahorra mucho tiempo en la línea cuando los equipos pueden visualizar los procesos, las interacciones de los usuarios y la estructura estática del sistema.

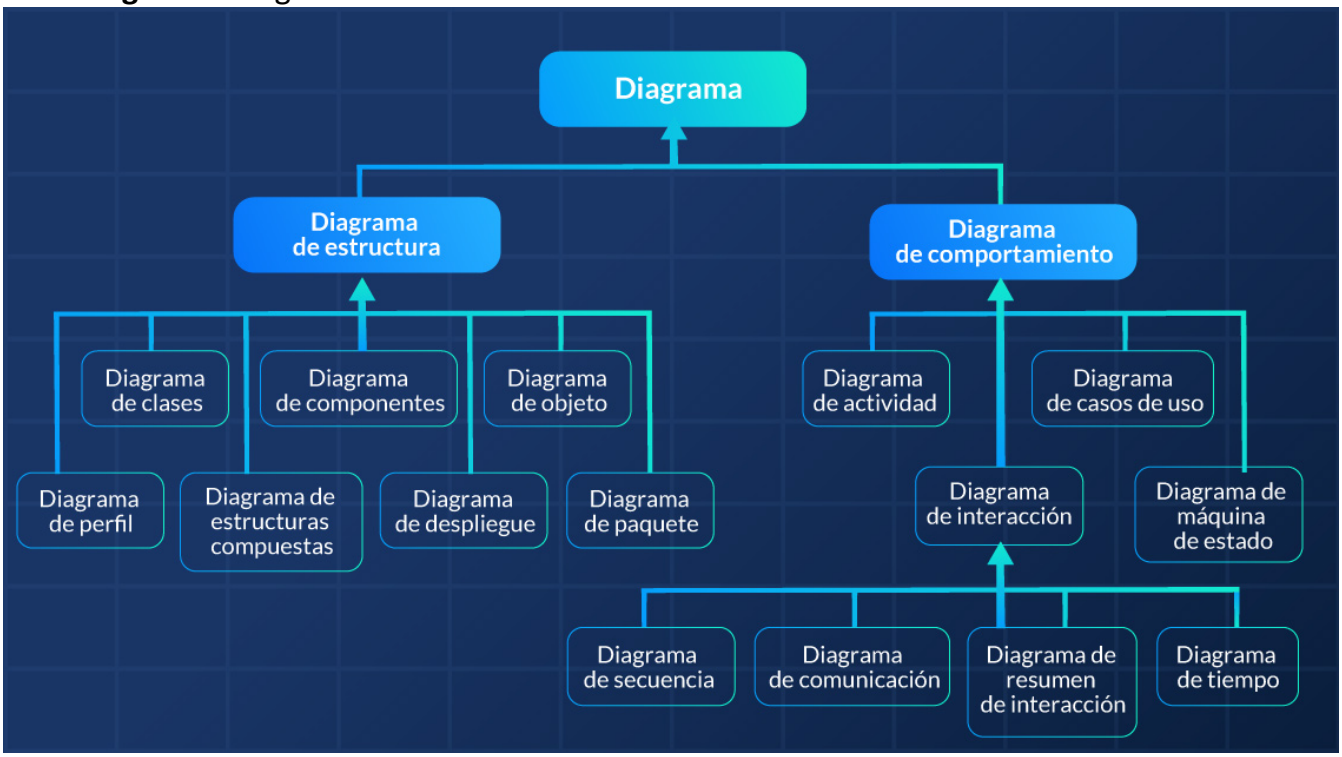
UML está vinculado con el diseño y análisis orientado a objetos. UML hace uso de elementos, formas y asociaciones entre ellos para formar diagramas. Los diagramas en UML se pueden clasificar en términos generales como:

1. **Diagramas estructurales**: captura aspectos estáticos o la estructura de un sistema. Los diagramas estructurales incluyen: Diagramas de componentes, Diagramas de objetos, Diagramas de clases y Diagramas de implementación.

2. **Diagramas de comportamiento:** captura aspectos dinámicos o el comportamiento del sistema. Los diagramas de comportamiento incluyen: Diagramas de casos de uso, Diagramas de estado, Diagramas de actividad y Diagramas de interacción.

La siguiente imagen muestra la jerarquía de diagramas de acuerdo con UML 2.2

Figura 1. Diagramas canónicos de UML



Fuente: Kirichenko. D (2014). Cómo Desarrollar Un Asesor Experto Usando Las Herramientas De Uml [Figura]. Recuperado de https://www.mql5.com/es/articles/304

Conceptos orientados a objetos utilizados en UML

- **Clase:** una clase define la impresión azul, es decir, la estructura y las funciones de un objeto.
- **Objetos:** los objetos nos ayudan a descomponer grandes sistemas y nos ayudan a modularizar nuestro sistema. El modularidad ayuda a dividir nuestro sistema en componentes comprensibles para que podamos construir nuestro sistema pieza por pieza. Un objeto es la unidad fundamental (bloque de construcción) de un sistema que se utiliza para representar una entidad.
- **Herencia:** herencia es un mecanismo por el cual las clases secundarias heredan las propiedades de sus clases primarias.
- **Abstracción:** mecanismo mediante el cual los detalles de la implementación están ocultos para el usuario.
- **Encapsulación:** el enlace de datos y la protección del mundo exterior se conoce como encapsulación.
- **Polimorfismo:** mecanismo por el cual las funciones o entidades pueden existir en diferentes formas.

Adiciones en UML 2.0

Se han incorporado metodologías de desarrollo ágil de software y se ha ampliado el alcance de la especificación UML original.

Originalmente UML especificaba nueve diagramas. UML 2.x ha aumentado el número de diagramas de 9 a 13. Los cuatro diagramas que se agregaron son: diagrama de tiempo, diagrama de comunicación, diagrama de visión general de interacción y diagrama de estructura compuesto. UML 2.x cambió el nombre de los diagramas de estado a diagramas de máquina de estado.

UML 2.x agregó la capacidad de descomponer el sistema de software en componentes y subcomponentes.

Diagramas UML Estructurales

- **Diagrama de clase:** el diagrama UML más utilizado es el diagrama de clase. Es el bloque de construcción de todos los sistemas de software orientados a objetos. Utilizamos diagramas de clase para representar la estructura estática de un sistema al mostrar las clases del sistema, sus métodos y atributos. Los diagramas de clase también nos ayudan a identificar la relación entre diferentes clases u objetos.

- **Diagrama de estructura compuesta:** utilizamos diagramas de estructura compuesta para representar la estructura interna de una clase y sus puntos de interacción con otras partes del sistema. Un diagrama de estructura compuesto representa la relación entre las partes y su configuración que determina cómo se comporta el clasificador (clase, componente o nodo de implementación). Representan la estructura interna de un clasificador estructurado que hace uso de partes, puertos y conectores. También podemos modelar colaboraciones utilizando diagramas de estructura compuesta. Son similares a los diagramas de clase, excepto que representan partes individuales en detalle, en comparación con toda la clase.

- **Diagrama de objetos:** se puede hacer referencia a un diagrama de objetos como una captura de pantalla de las instancias en un sistema, y la relación que existe entre ellas. Dado que los diagramas de objetos representan el comportamiento cuando los objetos se han instanciado, podemos estudiar el comportamiento del sistema en un instante particular. Un diagrama de objetos es similar a un diagrama de clases, excepto que muestra las instancias de las clases en el sistema. Representamos los clasificadores reales y sus relaciones haciendo uso de diagramas de clase. Por otro lado, un diagrama de objetos representa instancias específicas de clases y relaciones entre ellas, en un momento determinado.

- **Diagrama de componentes:** los diagramas de componentes se utilizan para representar la forma en que se han organizado los componentes físicos de un sistema. Los usamos para modelar los detalles de la implementación. Los diagramas de componentes representan la relación estructural entre los elementos del sistema de software y nos ayudan a comprender si los requisitos funcionales han sido cubiertos por el desarrollo planificado. Los diagramas de componentes se vuelven esenciales para usar cuando diseñamos y construimos sistemas complejos. Los componentes del sistema utilizan las interfaces para comunicarse entre sí.

- **Diagrama de implementación:** los diagramas de implementación se utilizan para representar el hardware del sistema y su software. Nos dice qué componentes de

hardware existen, y qué componentes de software se ejecutan en ellos. Ilustramos la arquitectura del sistema como distribución de artefactos de software sobre destinos distribuidos. Un artefacto es la información generada por el software del sistema. Se utilizan principalmente cuando se usa, distribuye o implementa un software en múltiples máquinas con diferentes configuraciones.

- **Diagrama de paquetes:** utilizamos diagramas de paquetes para describir cómo se han organizado los paquetes y sus elementos. Un diagrama de paquete simplemente nos muestra las dependencias entre diferentes paquetes y la composición interna de los paquetes. Los paquetes nos ayudan a organizar los diagramas UML en grupos significativos, y hacen que el diagrama sea fácil de entender. Se utilizan principalmente para organizar clases y usar diagramas de casos.

Diagramas de comportamiento

- **Diagramas de máquina de estado:** un diagrama de estado se utiliza para representar la condición del sistema o parte del sistema en casos de tiempo finitos. Es un diagrama de comportamiento y representa el comportamiento mediante transiciones de estados finitos. Los diagramas de estado también se conocen como máquinas de estado y diagramas de diagrama de estado. Estos términos a menudo se usan indistintamente. Así que, simplemente, se utiliza un diagrama de estado para modelar el comportamiento dinámico de una clase en respuesta al tiempo y al cambio de estímulos externos.

- **Diagramas de actividad:** utilizamos diagramas de actividad para ilustrar el flujo de control en un sistema. También podemos usar un diagrama de actividad para referirnos a los pasos involucrados en la ejecución de un caso de uso. Modelamos actividades secuenciales y concurrentes utilizando diagramas de actividad. Por lo tanto, básicamente representamos los flujos de trabajo visualmente, utilizando un diagrama de actividad. Un diagrama de actividad se centra en la condición del flujo y la secuencia en la que ocurre. Describimos o describimos las causas de un evento en particular usando un diagrama de actividad.

- **Diagramas de casos de uso:** los diagramas de casos de uso se utilizan para representar la funcionalidad de un sistema o una parte de un sistema. Se utilizan ampliamente para ilustrar los requisitos funcionales del sistema y su interacción con agentes externos (actores). Un caso de uso es básicamente un diagrama que representa diferentes escenarios donde se puede utilizar el sistema. Un diagrama de casos de uso nos da una

vista de alto nivel de lo que hace el sistema o una parte del sistema sin entrar en los detalles de la implementación.

- **Diagrama de secuencia:** un diagrama de secuencia simplemente representa la interacción entre objetos en un orden secuencial, es decir, el orden en el que tienen lugar estas interacciones. También podemos utilizar los términos diagramas de eventos o escenarios de eventos para referirse a un diagrama de secuencias. Los diagramas de secuencia describen cómo y en qué orden están los objetos en una función del sistema. Estos diagramas son ampliamente utilizados por los empresarios y desarrolladores de software para documentar y comprender los requisitos de los sistemas nuevos y existentes.

- **Diagrama de comunicación:** se utiliza un diagrama de comunicación (conocido como Diagrama de colaboración en UML 1.x) para mostrar los mensajes secuenciados intercambiados entre objetos. Un diagrama de comunicación se centra principalmente en los objetos y sus relaciones. Podemos representar información similar utilizando diagramas de secuencia, sin embargo, los diagramas de comunicación representan objetos y enlaces de forma libre.

- **Diagrama de tiempo:** el diagrama de tiempo es una forma especial de los diagramas de secuencia que se utilizan para representar el comportamiento de los objetos en un período de tiempo. Los usamos para mostrar las restricciones de tiempo y duración que rigen los cambios en los estados y el comportamiento de los objetos.

- **Diagrama de información general de interacción:** un diagrama de información general de interacción modela una secuencia de acciones y nos ayuda a simplificar interacciones complejas en situaciones más simples. Es una mezcla de actividad y diagramas de secuencia.

2. TEMA2.
Documentación del diseño de aplicaciones (Software)

Todos disfrutan de los beneficios de tener la documentación adecuada. Los proyectos de software se ejecutan sin problemas y las mejoras futuras son más fáciles cuando hay documentación escrita que acompaña al código.

Ciertos tipos de documentos, de una forma u otra, se encuentran en los mejores proyectos de software ejecutados, sin embargo, el hecho de que aquí se mencione un documento en particular no implica que sea necesario para todos los proyectos. Además, el hecho de que un recurso particular (plantilla, ejemplo o lista de verificación) incluya cierto contenido, no significa que este contenido sea necesario para todos los proyectos todo el tiempo. Los tipos de documentos creados y su contenido dependen de las necesidades del proyecto.

Para cada tipo de documento hay:

- Una breve descripción que presenta el contenido y el propósito del documento.
- Una plantilla que muestra los encabezados de las secciones principales y una breve explicación de lo que se necesita en cada sección.
- Uno o más ejemplos que ilustran cómo se vería una instancia real del documento. Los ejemplos son buenos para aclarar e inspirar, no la última palabra sobre lo que se necesita o se permite. Como es de esperar, la experiencia muestra que lo que es apropiado para un proyecto puede no serlo para otro.
- Una lista de verificación de garantía de calidad (QA) que enumera los criterios, generalmente en forma de preguntas breves, para evaluar un documento. Una lista de verificación puede proporcionar orientación durante el desarrollo o servir como un estándar para evaluar el trabajo completado.

Por conveniencia, los documentos están organizados en dos grupos:

1. Documentos de proceso.
2. Documentos del producto.

1. Documentos de proceso

Declaración de visión

La declaración de visión establece la dirección de un proyecto al especificar lo que se logrará. Define el alcance de un proyecto. Una buena declaración de visión lo ayuda a tomar decisiones con respecto a las prioridades, y qué incluir o excluir.

Carta del proyecto

La carta del proyecto define el alcance del proyecto y proporciona una justificación racional para llevarlo a cabo. Si se aprueba el proyecto, gran parte de la información contenida en los estatutos del proyecto se ampliará y se perfeccionará en el plan del proyecto.

Especificación de requisitos de software (SRS)

La especificación de requisitos de software enumera los requisitos funcionales y no funcionales junto con cualquier restricción de implementación. El documento de requisitos sirve a una audiencia diversa que va desde clientes no técnicos hasta programadores. Para satisfacer las necesidades de este grupo diverso, los requisitos se expresan comúnmente en niveles progresivos de detalle. La mayoría de los documentos de requisitos incluirán una lista de características generales del producto, así como el comportamiento detallado del sistema necesario para ofrecer estas características. El comportamiento detallado del sistema a menudo se expresa con casos de uso o escenarios de uso.

Plan de gestión de proyectos de software (SPMP)

La planificación del proyecto es el proceso de definir los resultados esperados del proyecto y diseñar un curso de acción para lograrlos. El plan del proyecto documenta los resultados del proceso de planificación.

Espere que el plan del proyecto se actualice y refine continuamente durante todo el proyecto a medida que cambian las condiciones y se sabe más sobre el proyecto. Pocos proyectos comienzan con toda la información necesaria para planificarlo en detalle desde el principio. Con mayor frecuencia, el plan del proyecto comienza con un desglose de alto nivel del trabajo conocido, un cronograma general del curso y solo estimaciones

aproximadas de costo y duración. Con el tiempo, a medida que se entienda más sobre el proyecto, estos y otros componentes del plan del proyecto se irán perfeccionando progresivamente.

Plan de lanzamiento

El plan de lanzamiento es un cronograma de alto nivel que se extiende por la duración del proyecto. Hay uno por proyecto y especifica el momento de las iteraciones y una asignación aproximada de las características del producto a las iteraciones.

Plan de iteración

Un plan de iteración define las actividades que se realizarán durante una iteración. Hay uno por iteración y especifican las tareas detalladas para una iteración, y en algunos casos una asignación de tareas a individuos.

Memo de entendimiento

Un memorando de entendimiento es una forma menos formal de documentar suposiciones e intenciones. Los acuerdos verbales a menudo se documentan en un memorando de entendimiento.

Criterios de éxito del proyecto

Los criterios de éxito del proyecto describen cómo se medirán los resultados del proyecto. Si el proyecto está programado para completarse el 1 de julio, ¿se considerará un fracaso si se termina el 2 de julio? Lo hará si los resultados del proyecto estuvieran destinados a un calendario de ferias comerciales para el 1 de julio. En otros casos, llegar una semana tarde podría ser tolerable. Los criterios de éxito del proyecto definen de manera medible y verificable lo que constituye el éxito del proyecto.

2. Documentos del producto

Arquitectura y Diseño

El propósito del documento de arquitectura / diseño es explicar la organización del código. Un documento de arquitectura bien escrito reducirá la cantidad de tiempo que

le toma a los programadores nuevos en un proyecto leer y comprender el código al nivel necesario para realizar modificaciones y mejoras.

El documento de arquitectura / diseño debe identificar los principales componentes del sistema y describir sus atributos estáticos y patrones dinámicos de interacción.

La arquitectura y los diseños de software generalmente se expresan con una combinación de modelos UML (los diagramas de clase y secuencia son los dos más comunes). Los diagramas de flujo de datos también son útiles para comprender la interacción entre los componentes y el flujo general de datos a través del sistema.

La arquitectura es un diseño de alto nivel, por lo que los principios que se encuentran en ambos tipos de documentos son los mismos, pero la forma en que se expresan puede variar según el nivel del diseño que se capture. Por ejemplo, los protocolos de comunicación entre componentes de arquitectura son relevantes para un documento de arquitectura, pero probablemente no sea un documento de diseño, donde la comunicación entre componentes generalmente se logra a través de variables compartidas / memoria o llamadas a procedimientos.

Guía del usuario

La guía del usuario explica cómo usar el software desde la perspectiva del usuario. Una guía de usuario bien escrita dará la bienvenida a los usuarios primerizos al proporcionar información básica sobre cómo comenzar rápidamente, pero también incluirá información más detallada para los usuarios avanzados que desean comprender cómo usar las funciones más avanzadas del software.

Documentación del sistema

La documentación del sistema (también conocida como guía de instalación, Manual del administrador, etc.) explica cómo instalar y configurar el software.

3. TEMA3.
Herramientas para la documentación de código

Una de las primeras consideraciones a tener en cuenta, cuando se piensa en las herramientas de documentos de API, es quién va a escribir. Si los escritores técnicos crean toda la documentación, la elección de las herramientas puede no importar tanto. Pero si los desarrolladores van a contribuir a los documentos, generalmente es ventajoso integrar sus herramientas de creación y publicación en la cadena de herramientas y el flujo de trabajo del desarrollador. Las herramientas centradas en el desarrollador para la documentación a menudo se denominan herramientas de documentos como código. Las herramientas de docs-as-code son mucho más comunes que las herramientas de creación de ayuda tradicionales (HAT) con documentación de API.

Integración en herramientas de ingeniería y flujos de trabajo

Riona Macnamara, escritora técnica de Google, dice que hace varios años, la documentación interna de Google estaba dispersa en wikis, Google Sites, Google Docs y otros lugares. En las encuestas internas de Google, muchos empleados dijeron que la incapacidad de encontrar documentación precisa y actualizada era uno de sus puntos débiles más significativos. A pesar de la excelencia de Google en la organización de la información externa del mundo en línea, organizarla internamente resultó ser difícil.

Riona dice que ayudaron a resolver el problema integrando documentación en el flujo de trabajo del ingeniero. En lugar de tratar de forzar herramientas centradas en el escritor, en los ingenieros, se ajusta la documentación en herramientas centradas en desarrolladores. Los desarrolladores ahora escriben documentación en los archivos Markdown, en el mismo repositorio que su código. Los desarrolladores también tienen un script para mostrar estos archivos Markdown en un navegador directamente desde el repositorio de código.

El método ganó rápidamente tracción, con cientos de proyectos de desarrolladores adoptando el nuevo método. Ahora, en lugar de crear documentación en un sistema separado (utilizando herramientas centradas en el escritor), los desarrolladores simplemente agregan el documento en el mismo repositorio que el código. Esta ubicación garantiza que cualquier persona que esté utilizando el código también puede encontrar la documentación. Los ingenieros pueden leer la documentación directamente en la fuente Markdown, o puede leerlo que se muestra en un navegador.

Si planeas que los desarrolladores escriban, definitivamente echa un vistazo a la presentación de Riona Macnamara:

Macnamara, R. (2015). Documentation, Disrupted: How Two Technical Writers Changed Google Engineering Culture. Recuperado de: https://www.youtube.com/watch?v=EnB8GtPuauw

Qué significan las herramientas de código de documentos

Hacer que los desarrolladores escriban o contribuyan a la documentación debe informar a su elección de herramienta con la documentación de la API. Si planea involucrar a los desarrolladores en la escritura y edición, naturalmente elegirá más de un enfoque de herramientas de Docs-as-Docs-as-Código, que significa tratar los documentos al igual que los desarrolladores tratan el código. Tratar documentos como código generalmente significa hacer algo como esto:

- Trabajar en archivos de texto sin formato (en lugar de formatos de archivo binarios como Adobe FrameMaker o Microsoft Word).
- Uso de un generador de sitios estáticos de código abierto como Sphinx, Jekyll o Hugo para construir los archivos localmente a través de la línea de comandos (en lugar de utilizar uno como FrameMaker o Word).
- Trabajar con archivos a través de un editor de texto como Atom o Sublime Text (en lugar de depender de herramientas comerciales con sistemas patentados y cerrados que funcionan como cajas negras).
- Almacenamiento de documentos en un repositorio de control de versiones (normalmente un repositorio Git) similar a cómo se almacena el código de programación (en lugar de mantener documentos en otro espacio como SharePoint o una unidad compartida); también se usa, potencialmente, almacenar los documentos en el mismo repositorio que el código.
- Colaborar con otros escritores utilizando el control de versiones como Git para bifurcar, fusionar, insertar y lograr actualizaciones de extracción (en lugar de colaborar a través de grandes sistemas de administración de contenido o SharePoint- como sitios de check-in / check-out).
- Automatización del proceso de compilación del sitio con entrega continua para crear la salida web desde el servidor cuando se actualiza una rama en particular (en lugar de publicar y transferir archivos manualmente de un lugar a otro).
- Ejecución de comprobaciones de validación utilizando scripts personalizados para comprobar si hay enlaces rotos, términos / estilos incorrectos, y errores de formato (en lugar de comprobar el contenido manualmente).

- Gestión de documentos mediante procesos similares a los ingenieros (por ejemplo, agile scrum), como fragmentar documentos en un gestor de problemas (como JIRA), la presentación de informes a las partes interesadas sobre el trabajo del documento completado (mostrando demostraciones). (Para obtener más información sobre este punto, consulte Siguiendo scrum ágil con proyectos de documentación.)

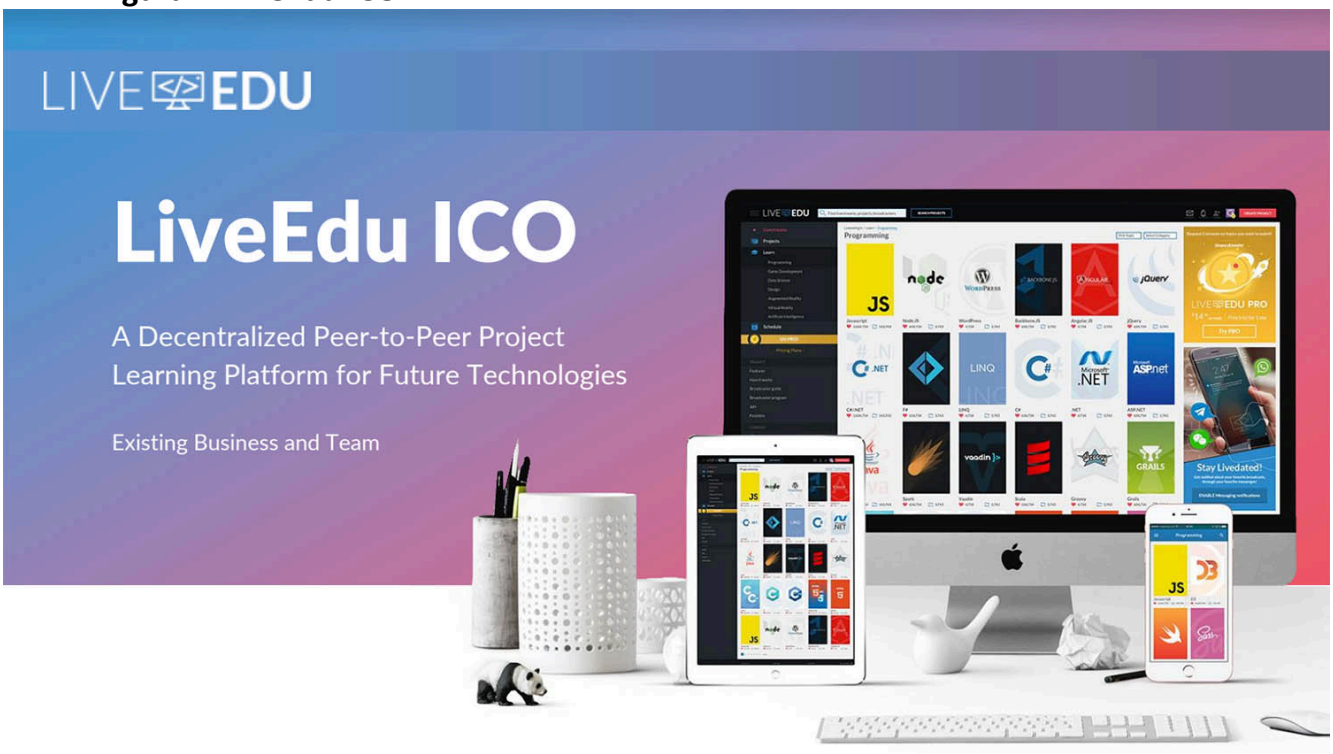
  En resumen, tratar documentos como código significa utilizar los mismos sistemas, procesos y flujos de trabajo con documentos como lo hace con el código de programación.

  Para habilitar el proceso de documentación más rápido y la consistencia del estilo, debes usar herramientas de documentación de códigos.

  A continuación, podrás ver algunas:

1. LiveEdu

Figura 2. LiveEdu ICO



Fuente: LiveEdu ICO (2019). Blockchain company building YouTube for online education [Figura]. Recuperado de https://vator.tv/news/2018-01-10-liveedu-launching-ico-a-decentralized-project-learning-platform-for-future-technologies

Puedes transmitir o almacenar tu trabajo de proyecto directamente en Livecoding. Al hacer esto, podrás permitir fácilmente a los miembros de tu equipo acceder a secciones importantes del proyecto. Existen múltiples beneficios al usar Livecoding como una herramienta para documentar tu código. Algunos de ellos se mencionan a continuación:

1. Mejora la documentación de texto escrito puro y proporciona un mejor contexto y comprensión al lector.
2. Los equipos ágiles pueden realizar fácilmente un seguimiento de los cambios del proyecto.
3. Los escritores técnicos pueden utilizar la documentación del código de video para comprender mejor el proyecto.
4. Los desarrolladores pueden invertir su tiempo ahorrado en la implementación de otras funcionalidades del proyecto.

2. Doxygen

Doxygen es una gran herramienta para generar documentación a partir del código fuente. La herramienta está dirigida a C ++, pero también se puede utilizar con PHP, Java, Python, etc. Con la ayuda de Doxygen, puedes crear documentación HTML en línea. También se puede utilizar para generar resultados en múltiples formatos, incluidas páginas de manual de Unix, PostScript, etc.

La mayor ventaja de usar Doxygen es que tendrás coherencia en toda la documentación del código fuente. También puede ayudarte a generar una estructura de código utilizando los archivos de origen no documentados. Todo lo que necesitas hacer es configurarlo en consecuencia.

3. Esfinge

Sphinx es una herramienta de documentación popular para los programadores. Está disponible bajo la licencia BSD y es compatible con múltiples lenguajes de programación como Python, C y C ++. Sphinx es ideal para desarrolladores que desean organizar su documentación. Puede usarse tanto para la documentación del proyecto como para la documentación del código. Algunas características de Sphinx incluyen extensas referencias cruzadas, múltiples formatos de salida, automática Índices, soporte de extensión, etc.

Figura 3. Sphinx

Fuente: Sphinx. (2007). Python Documentation Generator. Recuperado de http://www.sphinx-doc.org/en/master/

4. Pandoc

Pandoc no es como otras herramientas de documentación de código que hay. Actúa como una navaja suiza y permite a un desarrollador convertir rápidamente un formato de marca a otro. Si te gusta escribir tu propia documentación de código en el marcado, y deseas convertir rápidamente a otro formato, Pandoc es para ti. Tiene una amplia gama de soporte de documentos, incluyendo texto, texto recortado, LaTex, ePUB, etc.

Además, ofrece múltiples extensiones de sintaxis de rebajas, incluyendo listas de definición, tablas, notas al pie, etc. Puedes consultar la página oficial (https://pandoc.org/) para obtener una lista completa de las extensiones admitidas y el formato del documento.

5. Dr. Explique

El desarrollo Frontend también requiere documentación hasta cierto punto. Para esa tarea se tiene la herramienta, Dr. Explain, que te permite documentar la interfaz de usuario de la aplicación. Filtra los elementos clave de la interfaz y luego extrae la metainformación asociada a cada elemento. Una vez hecho esto, puedes modificar la información extraída para crear rápidamente una documentación de interfaz.

6. LaTex

LaTex es el estándar de facto para documentar proyectos científicos. Sin embargo, también se puede utilizar para otros tipos de proyectos, incluidos el código y la documentación del proyecto.

LaTex es bien conocido como un sistema de composición tipográfica de alta calidad con un enfoque en la producción de documentación científica y técnica.

7. Markdown

Markdown una creación de John Gruber, es un lenguaje simple que te ayuda a escribir código de alta calidad y documentación del proyecto. Técnicamente, Markdown es una herramienta de texto a HTML para escritores web, pero también puede usarse para fines de documentación. Como desarrollador, puedes escribir la documentación en Markdown y luego usar Pandoc para convertirla en el formato que desees.

8. GhostDoc

Con GhostDoc, una extensión de Visual Studio, puedes generar fácilmente tu documento XML de comentarios. La herramienta genera comentarios basados en múltiples factores, incluidos el nombre, los parámetros, la información contextual, el tipo, etc.

9. Natural Docs

Natural Docs es otro generador de documentos de código abierto que funciona con muchos lenguajes de programación. Te ayuda a automatizar la generación de documentación de código y convertirla en formato HTML. Actualmente, Natural Docs admite diecinueve lenguajes de programación, incluyendo Python, C ++, PL / SQL, Actionscript, etc.

10. phpDocumentor

Los desarrolladores de PHP, que desean generar documentación de código a partir del código fuente, tienen phpDocumentor que es un generador que ayudará a manejar la documentación de código y actúa como una referencia a la documentación adecuada. Las características clave de phpDocumentor son: compatibilidad con framework PHP, arquitectura conectable, etc. El trabajo interno es administrado por un sistema de plantillas poderoso y flexible. La herramienta también puede ayudar a generar informes, gráficos y a mejorar la calidad general del código.

4. TEMA4.
Herramientas para diseñar y documentar Bases de datos

A continuación, se describen algunas herramientas prácticas para la documentación de bases de datos.

1. Procesadores de textos

Cuando deseas crear un documento imprimible, los procesadores de texto (como MS Word o LibreOffice Writer) suenan como una opción obvia.

Beneficios

- Es muy probable que ya lo estés usando.
- Buenas capacidades de formateo, marca e impresión.

Desventajas

- Es un duro y arduo trabajo preparar el informe.
- Requiere de mucho tiempo.

2. Hojas de cálculo

Una opción ligeramente mejor para una herramienta sería una hoja de cálculo (como MS Excel o LibreOffice Calc). Facilita la navegación, búsqueda y filtrado de datos. También se siente más natural ya que la documentación suele tener estructura tabular.

Beneficios

- Es muy probable que ya lo estés usando.
- Fácil de buscar y filtrar metadatos.

Desventajas

Capacidades de impresión pobres.
La documentación es difícil de organizar, por la cantidad de documento requeridos.

3. Desarrollo de bases de datos y herramientas de gestión

Las consolas de bases de datos estándar como SQL Server Management Studio (SQL Server), Oracle SQL Developer (Oracle) o MySQL Workbench (MySQL) admiten algunos esquemas básicos de bases de datos y documentación y generación de modelos.

Esas características pueden incluir:

- Comentarios de elementos de datos (tablas, columnas, vistas, etc.)
- Generación de documentación HTML o PDF.
- Esquema de base de datos de ingeniería inversa a diagramas ER

Beneficios

- La mayoría de ellos son gratuitos o enviados con el motor de base de datos.
- DBAs, desarrolladores y arquitectos ya usan herramientas de administración de bases de datos.

Desventajas

- No hay documentación global para ambos - descripciones de objetos y diagramas.
- Pocas o ninguna capacidad de edición y creación (sólo funciones de anotación básicas).
- No hay repositorio para diagramas.
- No hay capacidad para generar documentación integrada que consiste en diagramas y un diccionario de datos detallado.

Ejemplos de herramientas
SQL Server: SQL Server Management Studio.
Oracle: Oracle SQL Developer.
MySQL: MySQL Workbench.

4. Generadores de documentación (solo lectura)

Existe una serie de herramientas cuyo único propósito es generar documentación a partir del esquema de su base de datos. Esta categoría de herramientas no tiene ninguna capacidad de edición y solo puede generar una documentación a partir de metadatos extraídos de la base de datos.

Beneficios

- Fácil de usar.
- Mejor formato que las herramientas de base de datos estándar.

Desventajas

- Sin capacidades de edición, se requiere una herramienta separada para esto (por ejemplo, una herramienta de administración de base de datos).

- Las capacidades de anotación están limitadas por la plataforma de base de datos (no puede proporcionar descripciones de texto enriquecido ni describir elementos si DBMS no lo admite).

Ejemplos de herramientas

- dbForge Documenter para SQL Server.
- Elasoft SqlSpec.
- Documentador Core Espectral.

5. Herramientas de documentación (lectura / escritura).

Hay una serie de herramientas que le permiten realizar ambas funciones de documentación:

Describir tablas y columnas (diccionario de datos) u otros objetos de la base de datos y genera documentos convenientes para compartir.

Esta categoría es diferente de la anterior en cuanto a capacidades de edición, que es una gran diferencia ya que proporcionar las descripciones es una actividad de documentación clave. Aquí es donde se crea el valor agregado. La generación de documentos hace que sea más fácil acceder y compartir, mientras que, al proporcionar descripciones de las estructuras de datos, se obtiene el conocimiento de las personas y se guarda para futuras referencias.

Estas herramientas almacenan en su mayoría metadatos (descripciones) en la base de datos, en Propiedades extendidas en el caso de SQL Server o comentarios en el caso de MySQL y Oracle.

Beneficios

- Capacidad para describir / anotar elementos de datos (tablas, columnas, etc.).

Desventajas

- Alcance de metadatos limitado por las capacidades de la base de datos (no puede agregar más información de la que el motor tiene).

Ejemplos de herramientas

- Redgate SQL Doc.
- ApexSQL Doc.
- Elasoft SqlSpec.

6. Herramientas avanzadas de documentación (lectura / escritura + metadatos + creación)

Existen herramientas que van más allá de los editores de propiedades / comentarios extendidos y los generadores de documentación, llamémosles repositorios de metadatos. Esas herramientas mantienen los datos en un repositorio separado que les permite recopilar y almacenar muchos más metadatos y los hace independientes en la plataforma de la base de datos. Proporcionan más funcionalidades: creación avanzada, más metadatos, diagramas de bases de datos.

Beneficios

- Repositorio global.
- Metadatos ricos.
- Mejores capacidades de autoría.
- Soporte para múltiples motores de bases de datos.

Desventajas

- Un poco más difícil de configurar y aprender.

7. Herramientas de modelado de datos

Existe una gran categoría de herramientas que están diseñadas específicamente para el modelado de datos lógico y físico independiente del motor. Proporcionan funcionalidades de ingeniería directa e inversa. Son buenos para crear diagramas de ER, pero mucho menos para describir elementos de datos (crear diccionarios de datos). Peor aún en el mantenimiento de la documentación.

Beneficios

- Repositorio global.
- Metadatos ricos.
- Mejores capacidades de autoría.
- Apoya a más de uno.
- Soporte para múltiples motores de bases de datos.

Desventajas

- Muy complicado y sobrecargado de características.
- Diseñado para propósitos de ingeniería avanzada, no es realmente ideal para ingeniería inversa.

- Edición de descripciones de columna en su mayoría no convenientes (Diccionario de datos).
- Documentos de exportación predeterminados Clunky.
- Exportar a veces requiere programación / personalizaciones.

Ejemplos de herramientas

- Erwin.
- SAP PowerDesigner.
- Idera ER / Studio Data Architect.

8. Herramientas de diagramación

Si solo desea crear diagramas de base de datos (no es una documentación completa de la base de datos) puede usar la herramienta de diagramación genérica.

Beneficios

- Buena diagramación y capacidades visuales.

Desventajas

- No hay soporte para el diccionario de datos (una descripción de cada elemento de datos).
- La mayoría de estas herramientas no admiten conexiones a la base de datos.
- Incluso si admiten la conexión a una base de datos, no es fácil mantener dichos modelos cuando se modifica el esquema de la base de datos.
- No es compatible con otros elementos de la base de datos: procedimientos almacenados, funciones, desencadenadores, etc.

Ejemplos de herramientas

- MS Visio.
- Gliffy (en línea).
- LucidChart (en línea).

5. TEMA5.
Pruebas de *Software*

Fundamentos

Las pruebas de software se refieren al proceso de evaluación del software con la intención de averiguar un error en él. Las pruebas de software son una técnica destinada a evaluar un atributo o capacidad de un programa o producto y determinar que cumple con su calidad. Las pruebas de software también se utilizan para probarlo en busca de otros factores de calidad, como fiabilidad, usabilidad, integridad, seguridad, capacidad, eficiencia, portabilidad, mantenibilidad, compatibilidad, etc.

Desde hace muchos años todavía estamos utilizando las mismas técnicas de prueba, algunas de los cuales son un método hecho a mano en lugar de buenos métodos de ingeniería. Las pruebas pueden ser costosas, pero no probar software puede ser aún más costoso. Las pruebas de software tienen como objetivo lograr ciertos propósitos y principios que deben seguirse.

Necesidad de pruebas de software

El desarrollo de software implica el desarrollo de software con un conjunto de requisitos. Las pruebas de este son necesarias para verificar y validar que el software que se ha creado se ha creado sirva para cumplir con las especificaciones. Así que, con el fin de asegurar que proporcionamos a nuestro cliente una solución de software adecuada, vamos a probar. Las pruebas garantizan que lo que obtienes al final es lo que querías crear. Comprobamos si hay algún problema, cualquier error en el sistema que puede hacer que el software sea inutilizable por el cliente. Esto ayuda en la prevención de errores en un sistema.

Objetivos para las pruebas de software

Los objetivos son la salida del proceso de software. Las pruebas de software tienen los siguientes objetivos:

- Verificación y validación: Las pruebas también se pueden utilizar para verificar que el producto, o el software funciona según lo deseado, y validar si el software cumple las condiciones establecidas.

- **Cobertura prioritaria:** las pruebas deben realizarse de manera eficiente y eficaz dentro de los límites presupuestarios y de programación.
- **Equilibrado:** el proceso de prueba debe equilibrar los requisitos, la limitación técnica y la expectativa del usuario.
- **Trazable:** los documentos deben estar preparados tanto del éxito como de los fracasos de los procesos de prueba. Así que no hay necesidad de probar lo mismo de nuevo.
- **Determinista:** debemos saber lo que estamos haciendo, lo que estamos apuntando, cuál será el posible resultado.

Principios de prueba

Principio es la regla o método en acción que tiene que ser continua. Los diferentes principios de prueba son los siguientes:

- **Pruebe un programa para intentar que falle:** las pruebas son el proceso de ejecutar un programa con la intención de encontrar errores. Debemos exponer los errores para hacer que el proceso de prueba sea más eficaz.
- **Comience a probar temprano:** esto ayuda a corregir enormes errores en las primeras etapas de desarrollo, reduce la reelaboración de la búsqueda de los errores en las etapas iniciales.
- Las pruebas dependen del contexto: Las pruebas deben ser apropiadas y diferentes para diferentes momentos.
- **Definir plan de pruebas:** el plan de pruebas generalmente describe el alcance de la prueba, los objetivos de prueba, la estrategia de prueba, el entorno de prueba, los resultados de la prueba, los riesgos y la mitigación, la programación, los niveles de pruebas que se van a aplicar, los métodos, las técnicas y las herramientas que se van a utilizar. El plan de pruebas debe satisfacer eficientemente las necesidades de una organización y los clientes también.
- Diseñar casos de prueba eficaces.
- **Prueba de condiciones válidas e inválidas:** además de las entradas válidas, también debemos probar el sistema para entradas/condiciones no válidas e inesperadas.
- **Las pruebas deben ser realizadas por diferentes personas a diferentes niveles:** diferente propósito abordado en diferentes niveles de pruebas por lo que la persona diferente debe realizar pruebas de manera diferente, utilizando diversas técnicas de prueba en diferentes niveles.

- **Fin de las pruebas:** las pruebas tienen que ser detenidas en algún lugar. La prueba se puede detener cuando el riesgo está por debajo de algún límite, o si hay limitación.

Técnicas de prueba de software

En esta sección el enfoque se centra en las diferentes técnicas de prueba de software. Las técnicas de prueba de software se pueden dividir en dos tipos:

1. Pruebas manuales (pruebas estáticas)

Es un proceso lento y laborioso donde las pruebas se hacen en estática y se realizan en la fase temprana del ciclo de vida. También se denomina prueba estática y lo hace el analista, desarrollador y el equipo de pruebas.

Diferentes técnicas de prueba manual son las siguientes:

- Caminar a través de.
- Revisión informal.
- Revisión técnica.
- Inspección.

2. Pruebas de mate automático (pruebas dinámicas)

En este probador se ejecuta el script en la herramienta de prueba, y se realiza la prueba. Las pruebas automatizadas también se denominan pruebas dinámicas. Las pruebas automatizadas se clasifican en cuatro tipos:

1. **Pruebas de corrección.**
2. **Pruebas de rendimiento.**
3. **Pruebas de fiabilidad.**
4. **Pruebas de seguridad.**

1. Pruebas de corrección

La corrección es el requisito mínimo de software. Las pruebas de corrección necesitarán algún tipo de guía para distinguir el comportamiento correcto del equivocado. El probador puede o no conocer los detalles interiores del módulo de software bajo prueba.

Por lo tanto, se pueden utilizar pruebas de caja blanca o pruebas de caja negra (las cuales veremos a continuación). Las pruebas de corrección tienen tres formas:

- iPruebas de caja blanca.
- Pruebas de cajas negras.
- Pruebas de cajas grises.

- Pruebas de caja blanca

Las pruebas de caja blanca son muy efectivas para detectar y resolver problemas, ya que a menudo se pueden encontrar errores antes de que causen problemas. La prueba de caja blanca es el proceso de dar la entrada al sistema y verificar cómo el sistema procesa esa entrada para generar la salida requerida. La prueba de caja blanca también se llama análisis de caja blanca, prueba de caja clara o análisis de caja clara. La prueba de caja blanca es aplicable a los niveles de integración, unidad y sistema del proceso de prueba de software. Las pruebas de caja blanca se consideran un método de prueba de seguridad que se puede utilizar para validar si la implementación del código sigue el diseño previsto, para validar la funcionalidad de seguridad implementada y para descubrir vulnerabilidades explotables.

Algunos tipos diferentes de pruebas de caja blanca son las siguientes:

- Pruebas de ruta de base.
- Pruebas de bucle.
- Pruebas de estructura de control.

Ventajas de las pruebas de caja blanca:

- Todas las rutas independientes de un módulo se ejercerán al menos una vez.
- Se ejercerán todas las decisiones lógicas.
- Se ejecutarán todos los bucles en sus bondades.
- Se ejercerán estructuras de datos internas para mantener su validez.
- Se revelan errores en códigos ocultos.
- El particionamiento realizado por igualdad en la ejecución del código, es decir, el tiempo de respuesta de los bucles.
- El desarrollador da cuidadosamente la razón sobre la implementación.

Desventajas de las pruebas de caja blanca:

1. Se perdieron los casos omitidos en el código.
2. Como el conocimiento del código y la estructura interna es un requisito previo, se necesita un probador experto para llevar a cabo este tipo de testing, lo que aumenta el costo.
3. Y es casi imposible examinar cada bit de código para descubrir errores ocultos, lo que puede crear problemas y provocar la falla de la aplicación.

Pruebas de cajas negras

Las pruebas de cajas negras son un software de prueba basado en los requisitos de output y sin ningún conocimiento de la estructura interna o codificación en el programa.

Básicamente, las pruebas de cajas negras son una parte integral de "Pruebas de corrección", pero sus ideas no se limitan a las pruebas de corrección solamente. El objetivo es probar cómo el componente se ajusta al requisito publicado para el componente. Las pruebas de cajas negras tienen poca o ninguna consideración a la estructura lógica interna del sistema, sólo examina el aspecto fundamental de este. Se asegura de que la entrada sea debidamente aceptada y la salida se produzca correctamente.

Algunos tipos diferentes de técnicas de prueba de caja negra son los siguientes:

- Partición equivalente.
- Análisis del valor límite.
- Técnicas gráficas de causa-efecto.
- Pruebas de comparación.
- Pruebas de pelusa.
- Pruebas basadas en modelos.

Ventajas de las pruebas de caja negra:

- El número de casos de prueba se reduce para lograr pruebas razonables.
- Los casos de prueba pueden mostrar presencia o ausencia de clases de errores.
- El probador de caja negra no tiene "vínculo" con el código.
- Tanto el programador como el probador son independientes entre sí.
- Más eficaz en unidades de código más grandes que en las pruebas de caja transparente.

Desventajas de las pruebas de caja negra:

1. Los casos de prueba son difíciles de diseñar sin especificaciones claras.
2. Solo se pueden probar pequeños números de entradas posibles.
3. Algunas partes del back-end no se prueban en absoluto.
4. Posibilidades de tener rutas no identificadas durante esta prueba.
5. Posibilidades de que se repitan las pruebas que ya realiza el programador.

Pruebas de caja gris

La metodología de pruebas caja gris (Graybox) es un método de prueba de software utilizado para probar aplicaciones del mismo. La metodología es independiente de la plataforma y el lenguaje. La implementación de current de la metodología Graybox depende en gran medida del uso de un depurador de plataforma host para ejecutar y validar el software bajo prueba. Estudios recientes han confirmado que el método Graybox se puede aplicar en tiempo real utilizando la ejecución de software en la plataforma de destino.

Las técnicas de prueba de cajas grises combinaron la metodología de prueba de la caja blanca y la caja negra. La técnica de prueba de caja gris se utiliza para probar una pieza de software en función de sus especificaciones, pero utilizando algunos conocimientos de su trabajo. La comprensión de los internos del programa en las pruebas de caja gris es más que las pruebas de caja negra, pero menos que las pruebas de caja clara.

La metodología Graybox (Caja Gris) es un proceso de diez pasos para probar software informático:

1. Identifican las entradas.
2. Identificar salidas.
3. Identificar las rutas principales.
4. Identificar subfunción (SF)X.
5. Desarrollar entradas para SF X.
6. Desarrollar salidas para SF X.
7. Ejecutar caso de prueba para SF X.
8. Verifique el resultado correcto para SF X.
9. Repita los pasos 4:8 para otros SF.
10. Repita los pasos 7 y 8 para la regresión.

La metodología Graybox utiliza herramientas automatizadas de pruebas de software para facilitar la generación de software único de prueba. Los controladores de módulo y los trozos son creados por el conjunto de herramientas para aliviar que los ingenieros de software de prueba tienen que generar manualmente este código. El conjunto de herramientas también verifica la cobertura del código instrumentando en el código de prueba, las herramientas de instrumentación ayudan con la inserción de código de instrumentación sin incurrir en los errores que se producirían a partir de la instrumentación manual.

Al operar en un depurador o emulador de destino, el conjunto de herramientas Graybox controlaba el funcionamiento del software de prueba. La metodología Graybox se ha movido de un depurador al mundo real y en tiempo real. La metodología se puede utilizar en tiempo real mediante la modificación de la premisa básica de que las entradas se pueden enviar al software de prueba a través de mensajes y salidas normales del sistema, se verifican utilizando los mensajes de salida del sistema.

2. Pruebas de rendimiento

Las pruebas de rendimiento implican todas las fases como el ciclo de vida de las pruebas principales como una disciplina independiente que implica tanto la estrategia como el plan, el diseño, la ejecución, el análisis y la generación de informes.

No todo el software tiene especificación de rendimiento explícitamente. Pero cada sistema tendrá requisitos de rendimiento implícitos.

El rendimiento siempre ha sido una gran preocupación y fuerza motriz de la evolución de la informática. Los objetivos de las pruebas de rendimiento pueden ser realizar la identificación de cuellos de botella del código, la comparación de rendimiento y la evaluación.

Mediante pruebas de rendimiento podemos medir las características de rendimiento de cualquier aplicación. Uno de los objetivos más importantes de las pruebas de rendimiento es mantener una latencia baja de un sitio web, un alto rendimiento y una baja utilización.

Las pruebas de rendimiento tienen dos formas:

Pruebas de carga: las pruebas de carga son el proceso de someter un ordenador, periférico, servidor, red o aplicación a un nivel de trabajo que se acerca a los límites de

sus especificaciones. Las pruebas de carga se pueden realizar en condiciones de laboratorio controladas para comparar las capacidades de diferentes sistemas o para medir con precisión las capacidades de un solo sistema. En esto podemos comprobar si el software puede manejar la carga de cualquier usuario o no.

Pruebas de estrés: las pruebas de esfuerzo son una prueba, que se lleva a cabo para evaluar un sistema o componente en o más allá de los límites de sus requisitos especificados para determinar la carga bajo la cual falla y cómo.

3. Pruebas de confiabilidad

El punto de las pruebas de fiabilidad es descubrir posibles problemas con el diseño tan pronto como sea posible y, en última instancia, proporcionar confianza en que el sistema cumple con sus requisitos de confiabilidad. Las pruebas de fiabilidad están relacionadas con muchos aspectos del software en los que se incluye el proceso de prueba; este proceso de prueba es un método de muestreo eficaz para medir la fiabilidad del software. En el sistema después de que el software desarrolla técnicas de prueba de confiabilidad como analizar, o técnicas de corrección, se pueden llevar a cabo para comprobar si se utiliza el software.

4. Pruebas de seguridad

La calidad del software, la fiabilidad y la seguridad están estrechamente acoplados. Los defectos en el software pueden ser explotados por intrusos para abrir agujeros de seguridad.

Las pruebas de seguridad se aseguran de que solo el personal autorizado pueda acceder al programa, y solo el personal autorizado pueda acceder a las funciones disponibles para su nivel de seguridad. Las pruebas de seguridad se realizan para comprobar si hay alguna fuga de información, en el sentido mediante el cifrado de la aplicación o el uso de una amplia gama de software y hardware y firewall, etc.

Estrategias de prueba de software

Una estrategia para pruebas de software integra métodos de diseño de casos de prueba de software en una serie bien planificada de pasos que dan como resultado la construcción exitosa de software. Las estrategias de prueba de software proporcionan la hoja de ruta para las pruebas. Una estrategia de pruebas de software debe ser lo suficientemente flexible como para promover un enfoque de pruebas personalizado, al mismo tiempo que debe ser lo suficientemente correcto. La estrategia es generalmente desarrollada por los gerentes de proyecto, ingeniero de software y especialista en pruebas.

Hay cuatro estrategias de prueba de software diferentes.

1. Pruebas unitarias.
2. Pruebas de integración.
3. Pruebas de aceptación/validación.
4. Pruebas del sistema.

1. Pruebas unitarias:

La unidad es el módulo más pequeño, es decir, la colección más pequeña de líneas de código que se pueden probar. Las pruebas unitarias son solo uno de los niveles de pruebas que van juntas para hacer el panorama general de la prueba de un sistema. Te complementa la integración y las pruebas a nivel de sistema. También debe complementar las revisiones de código y tutoriales.

Las pruebas unitarias se ven generalmente como una clase de prueba de caja blanca. Es decir, está sesgado a mirar y evaluar el código tal como se implementa. En lugar de evaluar la conformidad con algún conjunto de requisitos.

Beneficios de las pruebas unitarias:

1. Las pruebas de nivel unitario son muy rentables.
2. Proporciona una mejora de confiabilidad mucho mayor para los recursos expandidos que las pruebas a nivel del sistema en particular, tiende a revelar errores que son insidiosos y son a menudo catastróficos, como los extraños accidentes del sistema que se producen en el campo cuando algo inusual sucede.
3. Ser capaz de probar partes de un proyecto sin esperar a que las otras partes estén disponibles.
4. Lograr el paralelismo en las pruebas por ser capaz de probar y solucionar problemas simultáneamente por muchos ingenieros.

5. Ser capaz de detectar y eliminar defectos a un costo mucho menor en comparación con otras etapas posteriores de prueba.
6. Ser capaz de aprovechar una serie de técnicas de prueba formales disponibles para pruebas unitarias.
7. Simplifique la depuración limitando a una unidad pequeña las posibles áreas de código en las que buscar errores.
8. Ser capaz de probar las condiciones internas a las que no alcanzan fácilmente las entradas externas en los sistemas integrados más grandes.
9. Ser capaz de lograr un alto nivel de cobertura estructural del código.
10. Evite largos ciclos de compilación al depurar problemas difíciles.

Técnicas de pruebas unitarias

Una serie de técnicas de prueba eficaces se pueden utilizar en la etapa de prueba unitaria. Las técnicas de ensayo pueden dividirse ampliamente en tres tipos:

- Pruebas funcionales.
- Pruebas estructurales.
- Pruebas heurísticas o intuitivas.

2. Pruebas de integración:

Las pruebas de integración son una técnica sistemática para construir la estructura del programa y, al mismo tiempo, realizar pruebas para descubrir errores asociados con la interconexión. El objetivo es tomar componentes probados unitariamente y construir una estructura de programa como ha sido dictada por el diseño.

A continuación, se analizan diferentes estrategias de pruebas de integración:

- Pruebas de integración de rematar.
- Pruebas de integración de abajo hacia arriba.

Integración de arriba hacia abajo

Las pruebas de integración descendentes son un enfoque incremental para construir el programa. Los módulos se integran moviéndose hacia abajo a través de la estructura, comenzando con el módulo de control principal. Los módulos subordinados al módulo de control principal se incorporan a la estructura de una manera de profundidad primero o de amplitud primero.

El proceso se realiza en una serie de cinco pasos:

- El módulo de control principal se utiliza como controlador de prueba y los trozos se sustituyen por todos los componentes directamente subordinados al módulo de control principal.
- Dependiendo del enfoque de integración, los trozos subordinados seleccionados se reemplazan uno por uno con componentes reales.
- Las pruebas se realizan a medida que cada componente está integrado.
- Al completar cada conjunto de pruebas, otro código auxiliar se sustituye por el componente real.
- Se pueden realizar pruebas de regresión para garantizar que no se hayan introducido nuevos errores.

No es tan simple como parece, el problema surge cuando se prueba el módulo de bajo nivel que requiere pruebas del nivel superior. Es reemplazar el módulo de bajo nivel al principio de las pruebas de arriba hacia abajo. Por lo tanto, ningún dato puede fluir hacia arriba.

Integración de abajo hacia arriba

Las pruebas de integración de abajo hacia arriba, como su nombre simple, comienzan la construcción y las pruebas con módulos atómicos. Dado que los componentes se integran de abajo hacia arriba, el procesamiento necesario para los componentes subordinados a un nivel determinado siempre está disponible, y se elimina la necesidad de trozos.

Se puede implementar una estrategia de integración ascendente con los siguientes pasos:

- Los componentes de bajo nivel se combinan en clústeres que realizan una subfunción de software específica.
- Se escribe un controlador para coordinar la entrada y salida del caso de prueba.
- Se prueba el clúster.
- Los controladores se eliminan y los clústeres se combinan moviéndose hacia arriba en la estructura del programa.

3. Pruebas de aceptación:

Las pruebas de aceptación (también conocidas como pruebas de aceptación del usuario) son un tipo de prueba realizada con el fin de verificar si el producto se desarrolla de acuerdo con las normas y criterios especificados, y cumple con todos los requisitos

especificados por el cliente. Este tipo de pruebas generalmente se llevan a cabo por un usuario / cliente donde el producto es desarrollado externamente por otra parte.

Las pruebas de aceptación entran a la metodología de pruebas de caja negra en la que el usuario no está muy interesado en el trabajo interno/codificación del sistema, pero evalúa el funcionamiento general del sistema y lo compara con los requisitos especificados por ellos. Las pruebas de aceptación del usuario se consideran una de las pruebas más importantes por el usuario, antes de que el sistema sea finalmente entregado o entregado al usuario final.

Las pruebas de aceptación también se conocen como pruebas de validación, pruebas finales, pruebas de control de calidad, pruebas de aceptación de fábrica y pruebas de aplicaciones, etc. Una atravesada en ingeniería de software, las pruebas de aceptación pueden llevarse a cabo en dos niveles diferentes; uno a nivel de proveedor de sistema, y otro a nivel de usuario final.

Tipos de pruebas de aceptación

Pruebas de aceptación del usuario: aceptación del usuario en ingeniería de software se consideran un paso esencial antes de que el sistema sea finalmente aceptado por el usuario final. En términos generales, las pruebas de aceptación del usuario son un proceso de prueba del sistema antes de que sea finalmente aceptado por el usuario.

Pruebas Alpha Testing & Beta: las pruebas Alfa son un tipo de pruebas de aceptación realizadas en el sitio del desarrollador por los usuarios. En este tipo de pruebas, el usuario continúa probando el sistema y el resultado es observado y observado por el desarrollador simultáneamente.

Las pruebas beta son un tipo de prueba realizada en el sitio del usuario. Los usuarios proporcionan sus comentarios al desarrollador para el resultado de las pruebas. Este tipo de pruebas también se conoce como pruebas de campo. Los comentarios de los usuarios se utilizan para mejorar el sistema/producto antes de que se publique a otros usuarios/clientes.

Pruebas de Aceptación Operacional: este tipo de pruebas también se conoce como pruebas de preparación/preparación operativas. Es un proceso de asegurar que todos los componentes requeridos (procesos y procedimientos) del sistema están en su lugar con el fin de permitir que el usuario / probador para utilizarlo.

Pruebas de aceptación de contactos y regulaciones: en las pruebas de aceptación de contratos y regulaciones, el sistema se prueba con los criterios especificados como se menciona en el documento del contrato, y también se prueba para comprobar si cumple/obedece a todo el gobierno, y a las normas y leyes de las autoridades locales y también todas las normas básicas.

4. Pruebas del sistema:

Las pruebas del sistema de software o hardware se realizan en un sistema completo e integrado para evaluar el cumplimiento del sistema con sus requisitos especificados. Las pruebas de sistema están comprendidas en el ámbito de las pruebas de la caja negra, y como tal, no deben requerir ningún conocimiento del diseño interno del código o de la prueba lógica del sistema. Es en realidad una serie de pruebas diferentes cuyo objetivo principal es un sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todos funcionan para verificar que los elementos del sistema se han integrado correctamente y realizar funciones asignadas.

Algunos de los diferentes tipos de pruebas del sistema son los siguientes:

- Pruebas de recuperación.
- Pruebas de seguridad.
- Pruebas gráficas de la interfaz de usuario.
- Pruebas de compatibilidad.

Pruebas de recuperación: Las pruebas de recuperación son una prueba del sistema que obliga al software a fallar de diversas maneras y verifica que la recuperación sea realizada. Si la recuperación es automática, la reinicialización, la verificación de los mecanismos de apuntado, la recuperación de datos y el reinicio se evalúan para su corrección. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación para determinar si está dentro de los límites aceptables.

Pruebas de seguridad: las pruebas de seguridad intentan verificar que los mecanismos de protección integrados en un sistema lo protegerán de una penetración inadecuada.

Durante las pruebas de seguridad, el probador desempeña el papel de la persona que desea penetrar en el sistema. ¡Todo vale! El evaluador puede intentar adquirir contraseñas a través de medios administrativos externos; puede atacar el sistema con software personalizado diseñado para descomponer cualquier defensa que se han construido; puede abrumar el sistema, negando así el servicio a otros; puede causar intencionalmente errores del sistema, con la esperanza de penetrar durante la

recuperación; puede navegar a través de la recuperación; inseguro data, con la esperanza de encontrar la clave para la entrada del sistema.

Pruebas gráficas de la interfaz de usuario: las pruebas gráficas de la interfaz de usuario son el proceso de probar la interfaz gráfica de usuario de un producto para asegurarse de que cumple con sus especificaciones escritas. Esto se hace normalmente a través del uso de una variedad de casos de prueba.

Pruebas de compatibilidad: las pruebas de compatibilidad que forman parte de las pruebas no funcionales de software, son pruebas realizadas en la aplicación para evaluar la compatibilidad de la aplicación con el entorno informático.

Análisis Final

Las pruebas de software son un proceso que se puede planificar y especificar sistemáticamente. Se puede llevar a cabo el diseño de casos de prueba, se puede definir una estrategia y los resultados se pueden evaluar según las expectativas prescritas.

La depuración se produce como consecuencia de las pruebas correctas. Es decir, cuando un caso de prueba descubre un error, la depuración es el proceso que da como resultado la eliminación del error.

El propósito de la depuración es localizar y corregir el código infractor responsable de un síntoma que infrinja una especificación conocida.

Lecturas
y Material Complementario

Te invitamos a explorar el siguiente material para que amplíes los temas estudiados hasta este momento. Lee y analiza cada uno de ellos, pues contienen información que te será de gran ayuda en el desarrollo del curso.

- Solution Center. (2015). Cómo documentar bases de datos SQL automáticamente (Traductor Calbimonte, D.). Recuperado de: https://solutioncenter.apexsql.com/es/como-documentar-bases-de-datos-sql-automaticamente/

- it-Mentor. (s.f.). Capacitación y guía para el desarrollo de software. Pruebas de software. Recuperado de: http://materias.fi.uba.ar/7548/PruebasSoftware.pdf

- Castaño V. (2014). Tendencias de la Teoría General de Sistemas. Prezi. Recuperado de: https://prezi.com/nai6jdnqjk1y/tendencias-de-la-teoria-general-de-sistemas/

- García, A. (1995). Notas sobre la teoría general de sistemas. Revista General de Información y Documentación, 5(1), 197-213. Recuperado de https://dialnet.unirioja.es/servlet/articulo?codigo=903087

Bibliografía

- ISO / IEC / IEEE. (2011). Ingeniería de sistemas y software: desarrollo de documentación de usuario en un entorno ágil. Ginebra, Suiza.
- Parnas, D. (1972). Sobre los criterios que se utilizarán en la descomposición de sistemas en módulos, Communications of the ACM, 5: (12) 1053-1058.
- IEEE. (1990). Glosario estándar de ingeniería de software. Washington, DC, EE. UU. Instituto de Ingenieros Eléctricos y Electrónicos (IEEE). IEEE 610.12-1990.
- PMI (2013). Una guía para el cuerpo de conocimiento de gestión de proyectos (Guía PMBOK®). 5ta ed. Newtown Square, PA, EE. UU: Project Management Institute (PMI).
- PMI. (2013). Una guía para el cuerpo de conocimiento de gestión de proyectos (Guía PMBOK®). 5ta ed. Newtown Square, PA, EE. UU.: Project Management Institute (PMI).
- ANSI / EIA. (2003). Procesos para diseñar un sistema. Filadelfia, Pensilvania, EE. UU.: American National Standards Institute (ANSI) / Electronic Industries Association (EIA), ANSI / EIA-632-1998.
- ISO / IEC / IEEE. (2015). Ingeniería de sistemas y software - Procesos del ciclo de vida del sistema. Ginebra, Suiza. ISO / IEC / IEEE 15288.
- Ackoff, RL (1971). Hacia un sistema de conceptos de sistemas. Ciencias de gestión. 17 (11).
- Hitchins, D. (2007). Ingeniería de sistemas: una metodología de sistemas del siglo XXI. Hoboken, Nueva Jersey, EE. UU.: John Wiley & Sons.
- ISO / IEC / IEEE. (2011). Ingeniería de sistemas y software - Contenido de los productos de información del ciclo de vida (documentación). Sección 5.16. Ginebra, Suiza. ISO / IEC / IEEE 15289.
- Alexander, C. (1979). La forma intemporal de construir. Nueva York, NY, EE. UU.: Oxford University Press.

- Beer, S. (1967). Cybernetics and Management, 2ª edición. Londres, Reino Unido: English Universities Press.

- Woods, D. D. (2006). Características esenciales de la resiliencia, en Ingeniería de resiliencia: conceptos y preceptos, Ashgate Publlishing Limited.

- Ackoff, RL (1971). Hacia un sistema de conceptos de sistemas. Ciencias de gestión 11: 11.

- INGRESO. (1998). Términos Glosario. INCOSE Conceptos y términos, Grupo de trabajo (ed.). Seattle, WA, EE. UU.: Consejo Internacional de Ingeniería de Sistemas.

- Wasson, C. S. (2006). Análisis, diseño y desarrollo de sistemas. Editado por AP Sage, Serie Wiley en Ingeniería y Gestión de Sistemas. Hoboken, Nueva Jersey, EE. UU.: John Wiley & Sons.

- Jackson, S. (2016). Evaluación de principios de resiliencia para sistemas de ingeniería. Doctorado en Investigación, Ingeniería, Universidad del Sur de Australia.

- INGRESO (2011). Manual de ingeniería de sistemas: una guía para los procesos y actividades del ciclo de vida del sistema. Versión 3.2.2. San Diego, CA, EE. UU.: Consejo Internacional de Ingeniería de Sistemas (INCOSE), INCOSE-TP-2003-002-03.2.2

IU Digital de Antioquia

INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

ORD No 74 de 2017 · VIGILADA MinEducación

www.iudigital.edu.co