



# ASEGURAMIENTO DE LA CALIDAD

# BIENVENIDA

Bienvenido(a) a la asignatura *Aseguramiento de la Calidad*, con la cual comprenderás los principios básicos del proceso de aseguramiento de la calidad en la industria del software. Esto es importante porque evaluarás la calidad de un producto intangible, con requerimientos dinámicos y que puede ser desarrollado de muchas maneras.

Este curso será la base para que puedas contestar estas preguntas: ¿Cómo desarrollar un software de calidad? o ¿cómo evaluar la calidad de un software? De tal manera, este curso te permitirá conocer los tipos de pruebas que existen, y cuándo utilizarlas en la industria del desarrollo de software.



# LIBROS RECOMENDA DOS

- Laporte, C. Y. & April, A. (2018) Software Quality Assurance. Wiley.
- Galin, D. (2018) Software Quality: concepts and practice. Wiley.



# CASO DISRUPTOR



Te invitamos a revisar el Caso de aplicación:

- **Mozilla implementa una solución para 2 errores explotados activamente en el navegador Firefox**

A través del siguiente video:



Si gustas conocer más acerca de este tipo de casos te invitamos a entrar a la sección de *disruptor* desde tu plataforma de estudio.

# UNIDAD 1

# CALIDAD EN LA INDUSTRIA DEL SOFTWARE



# TEMARIO UNIDAD 1



**1.1**

Atributos de la calidad del software

**1.2**

Revisiones formales e informales



# INTRODUCCIÓN

La asignatura *Aseguramiento de la calidad* tiene como objetivo que el alumno conozca el conjunto de acciones sistemáticas y planificadas, necesarias para garantizar que un producto o servicio cumplirá con las exigencias de calidad.

En esta primera unidad aprenderás los conceptos básicos, su aplicación, y cómo afectan a la calidad de los productos de software.



# COMPETENCIAS A DESARROLLAR

**El alumno será capaz de explicar los conceptos básicos utilizados en el área de calidad de software.**



**El alumno será capaz de aplicar técnicas que solucionen, con calidad, problemáticas de software.**

# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

Para hablar de la calidad del software, el primer concepto a clarificar es el de calidad.

De acuerdo con la Sociedad Americana para el Control de la Calidad, la calidad es “el conjunto de características de un producto, proceso o servicio que le confieren aptitud para satisfacer las necesidades del usuario o cliente”.

La siguiente diapositiva nos aclara con qué se relaciona y con qué no se relaciona el concepto de calidad.



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

NO X

Tengo el mejor producto, el más caro, el más durable.

SÍ ✓

Tengo el producto que cumple con las expectativas de los clientes.

Mi empresa está certificada, hago los mejores productos.

Mi empresa está certificada, hago productos con la calidad especificada.

Mi empresa trabaja con calidad total, estoy satisfecho con mis productos.

Mi empresa trabaja con calidad total, siempre estoy mejorando mis productos.



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

Desde el punto de vista del cumplimiento de los requerimientos, Roger Pressman define la calidad de software como:

“Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente.” [Pressman, 2005: 135].



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

Sin embargo, la calidad es **subjetiva**, porque depende de los atributos elegidos para medirla; y es **circunstancial**, porque el conjunto de atributos elegidos puede variar en situaciones diferentes.

Cuando aplicamos el concepto de calidad al software, deja de ser subjetiva, porque se determinan los atributos de calidad; pero no deja de ser circunstancial, ya que, en ciertas situaciones, un determinado conjunto de características de calidad puede ser más importante que en otras.

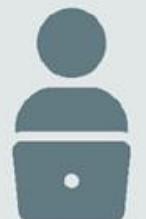


# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

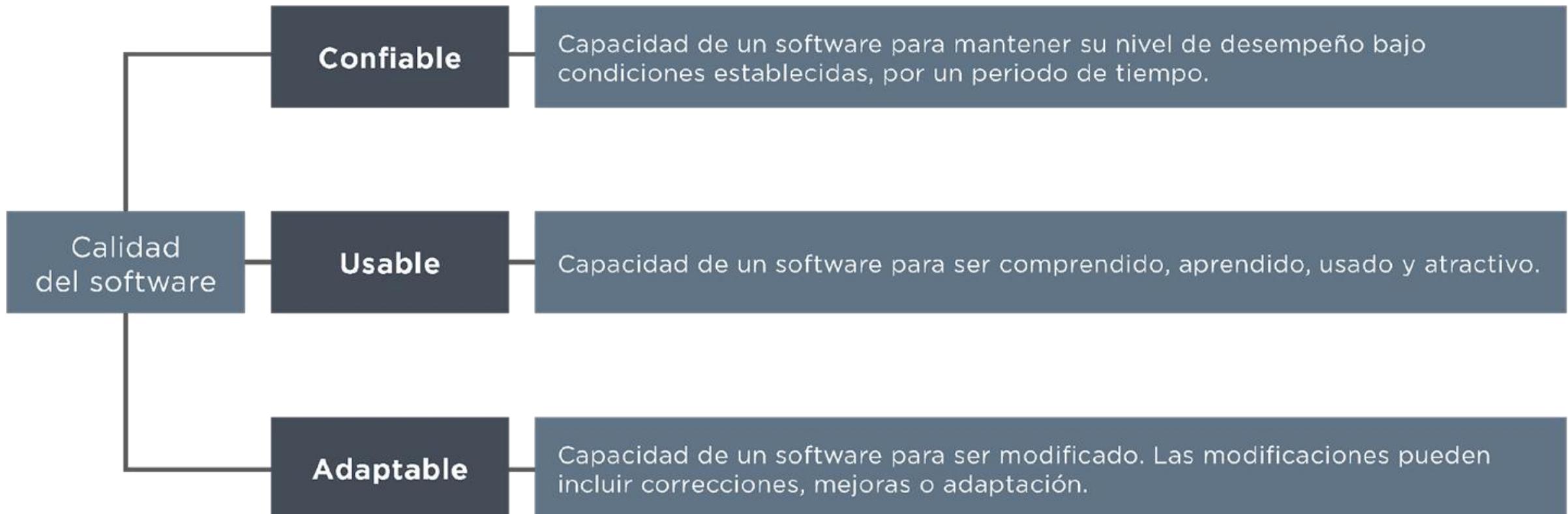
En este sentido, es importante definir atributos que permitan medir la calidad del software. Estos son los siguientes:

- ▶ Confiabilidad
- ▶ Usabilidad
- ▶ Adaptabilidad
- ▶ Funcionalidad
- ▶ Eficiencia
- ▶ Portabilidad

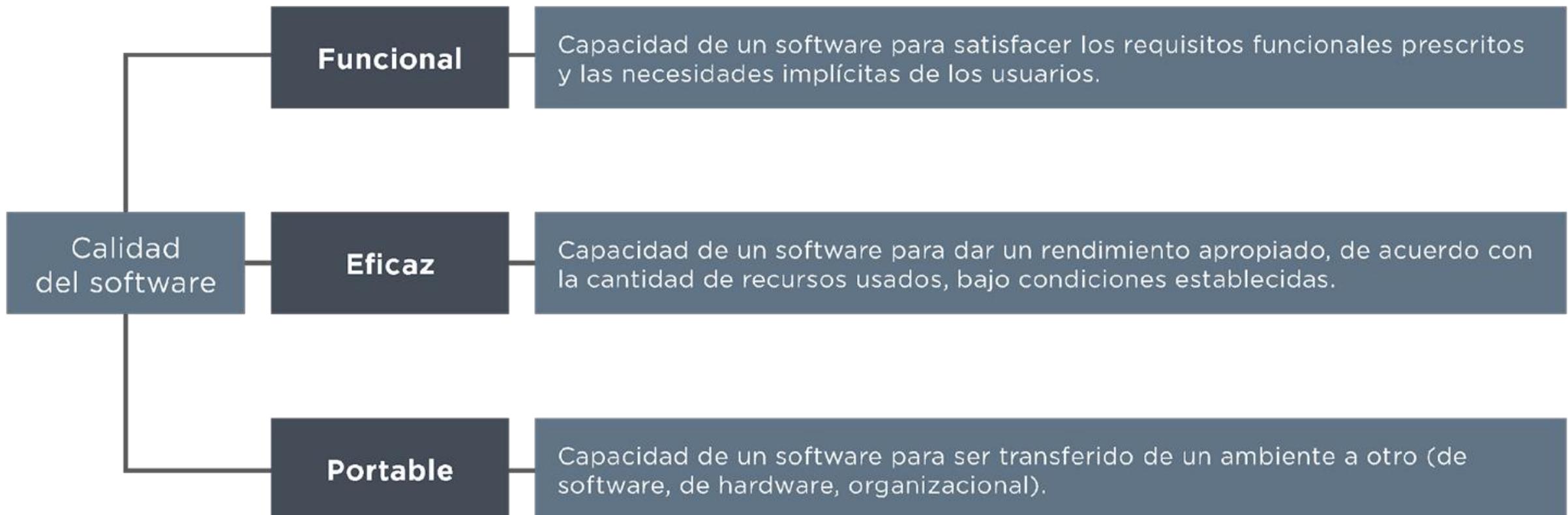
Y se definen a continuación:



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

Además de la calidad del software, es importante abordar el SQA (Aseguramiento de la Calidad de Software, por sus siglas en inglés). No puede existir uno sin la presencia del otro.

El **aseguramiento de la calidad del software** es un conjunto de actividades planificadas y ejecutadas sistemáticamente que garantiza que el software que se está construyendo es de alta calidad.



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

La obtención de un software con calidad implica el uso de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software. Estas deben permitir uniformar la filosofía de trabajo para lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba; a la vez, estos elevan la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

**El grupo de SQA se encarga de:**

- Preparar un plan de SQA para un proyecto.
- Participar en el desarrollo de la descripción del proceso de software del proyecto.
- Revisar las actividades de ingeniería del software.
- Auditarse productos de trabajo de software seleccionados.



# ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

- Garantizar que las desviaciones en el trabajo de software y en los productos de trabajo estén documentadas.
- Registrar cualquier falta de ajuste.
- Corroborar la confiabilidad del software.
- Evaluar continuamente la seguridad del software.

Las revisiones del software son un medio efectivo para descubrir errores, defectos y mejoras.



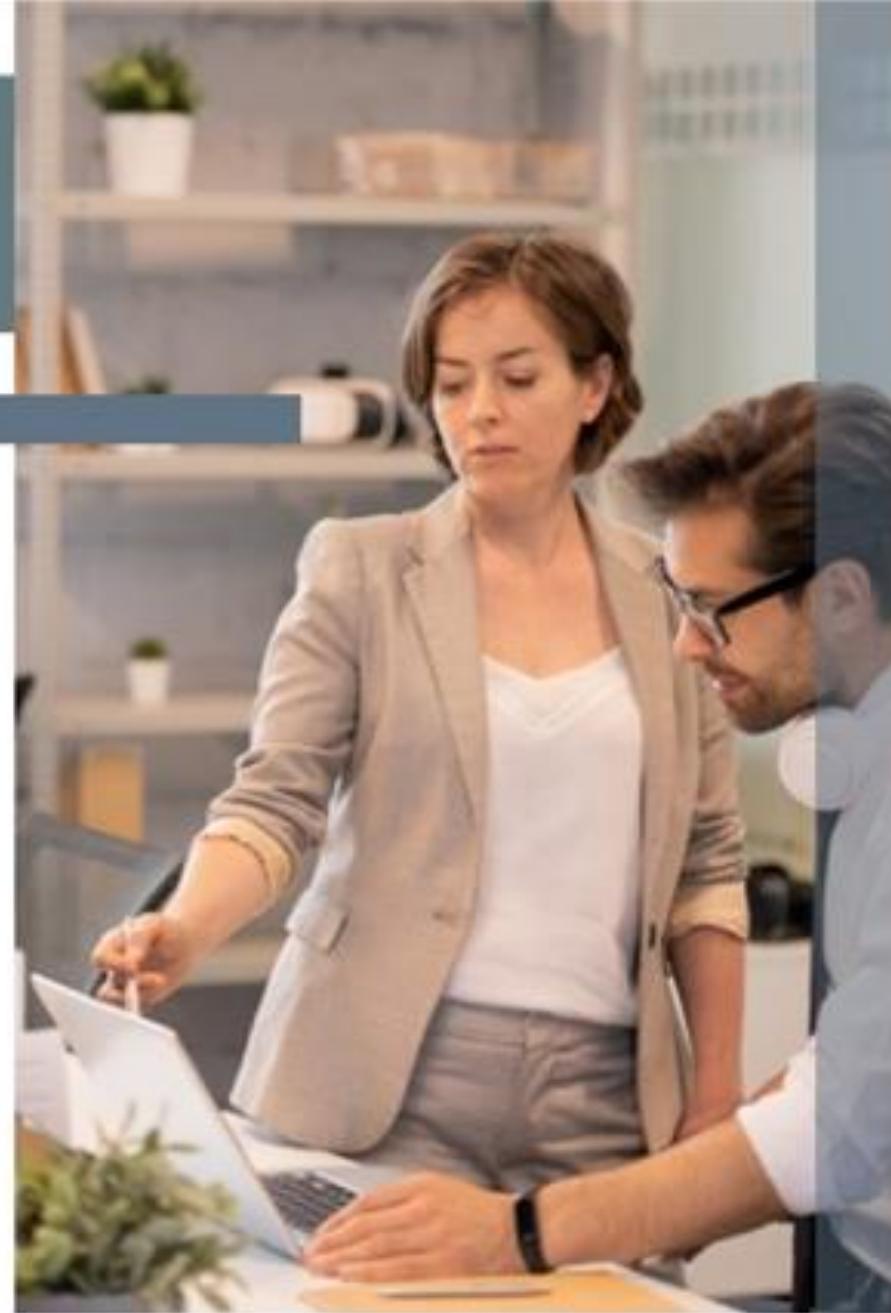
# VIDEOS



Te invitamos a ver el siguiente video:



Programa del proceso



# REVISIONES FORMALES E INFORMALES



Debemos tener claros los conceptos **error**, **defecto** y **fallo**. Estos son términos que utilizamos a menudo cuando el sistema o la aplicación actúa anormalmente. Sus definiciones y ejemplos son los siguientes:

- **Error.** Es una acción humana que produce un resultado incorrecto o una idea equivocada de algo. Es una equivocación de parte del desarrollador o del analista. Un error puede llevarnos a generar uno o más defectos.

# REVISIONES FORMALES E INFORMALES



Algunos ejemplos de errores son los siguientes:

- Inconsistencia en la lógica de la programación.
- Un requerimiento mal especificado.

Los errores se llaman así cuando se detectan antes de haberse entregado el software al usuario.

# REVISIONES FORMALES E INFORMALES



- **Defecto.** Comúnmente se encuentra en algún componente del sistema. Es la imperfección de un componente causado por un error.

El analista de pruebas o **tester** reporta el defecto, ya que es el encargado de ejecutar los casos de prueba y encontrar los mismos. Los defectos son conocidos como **bugs**.

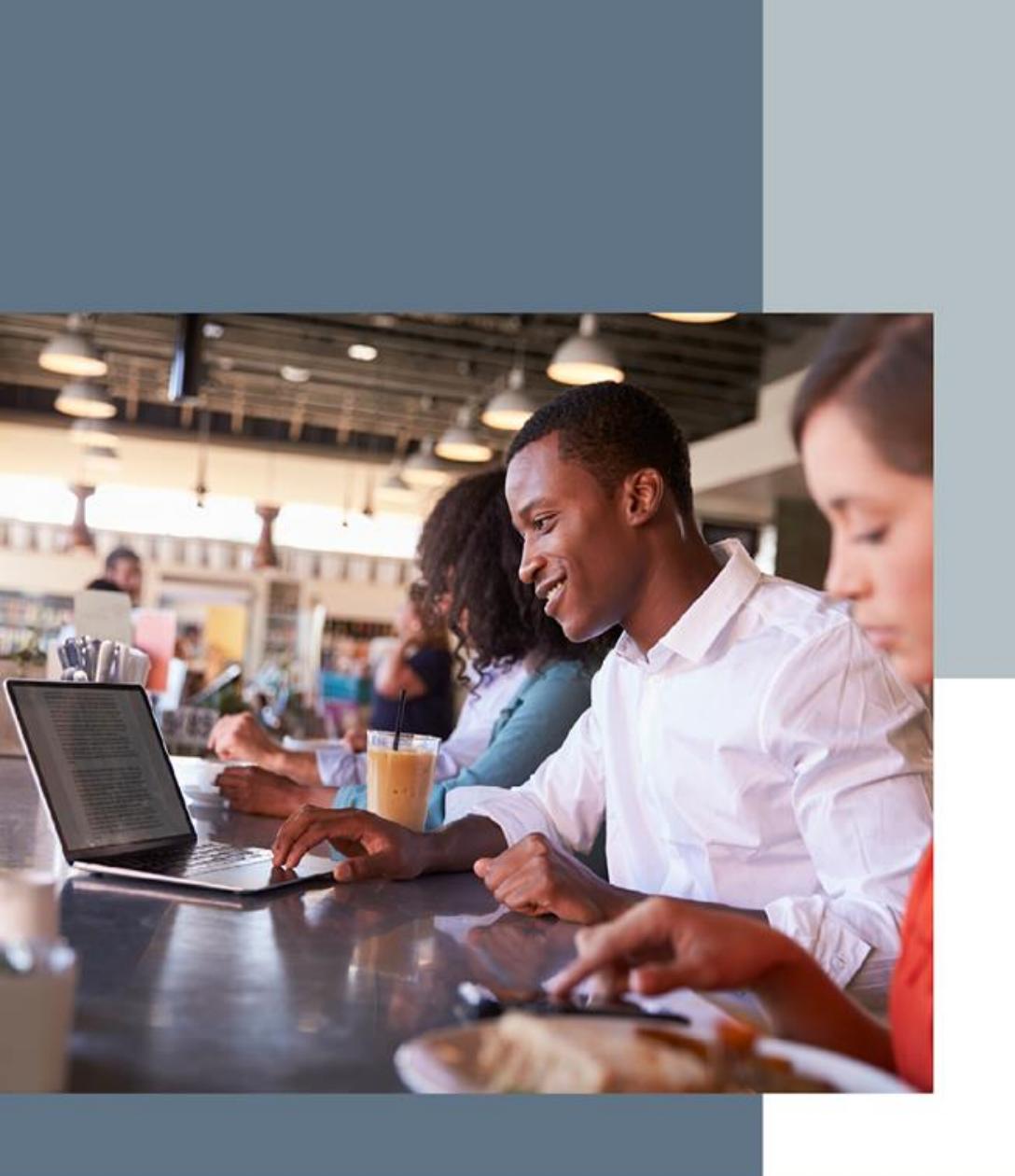
# REVISIONES FORMALES E INFORMALES



Algunos ejemplos de defectos son los siguientes:

- El módulo de facturación tiene mal la configuración en la función de cálculo mensual.
- La función de suma no cuenta con las variables de usuario necesarias.

Los **fallos**, por su parte, se detectan después de haber entregado el software al usuario.



# REVISIONES FORMALES E INFORMALES

- **Fallo.** La manifestación visible de un defecto. Es decir que, si un defecto es encontrado durante la ejecución de una aplicación, entonces va a producir un fallo.

Algunos ejemplos de fallos son los siguientes:

- Cuando nos aparece un mensaje de alerta.
- En una página de captura de datos, el botón de editar no funciona.



# REVISIONES FORMALES E INFORMALES



Un error puede generar uno o más defectos, y un defecto va a causar un fallo.

Por ejemplo, si un desarrollador comete el error de colocar el valor máximo de temperatura, diferente al de los requerimientos, y al momento de realizar las pruebas del sistema el analista de pruebas de software coloca la temperatura correcta dentro del parámetro definido en el requerimiento, se genera un defecto en el sistema que provoca, a su vez, un fallo, que podría ser un mensaje en pantalla, el cual indicaría que la temperatura no es válida.

# REVISIONES FORMALES E INFORMALES



Si nosotros arrastramos los errores de etapas anteriores, pueden surgir tres tipos de errores más:

- **Errores inadvertidos:** aquellos que pasamos por alto.
- **Errores amplificados:** aquellos que eran pequeños, y ya se hicieron más grandes.
- **Errores nuevos:** aquellos errores que no existían y, a causa del error anterior, se generaron.

# REVISIONES FORMALES E INFORMALES

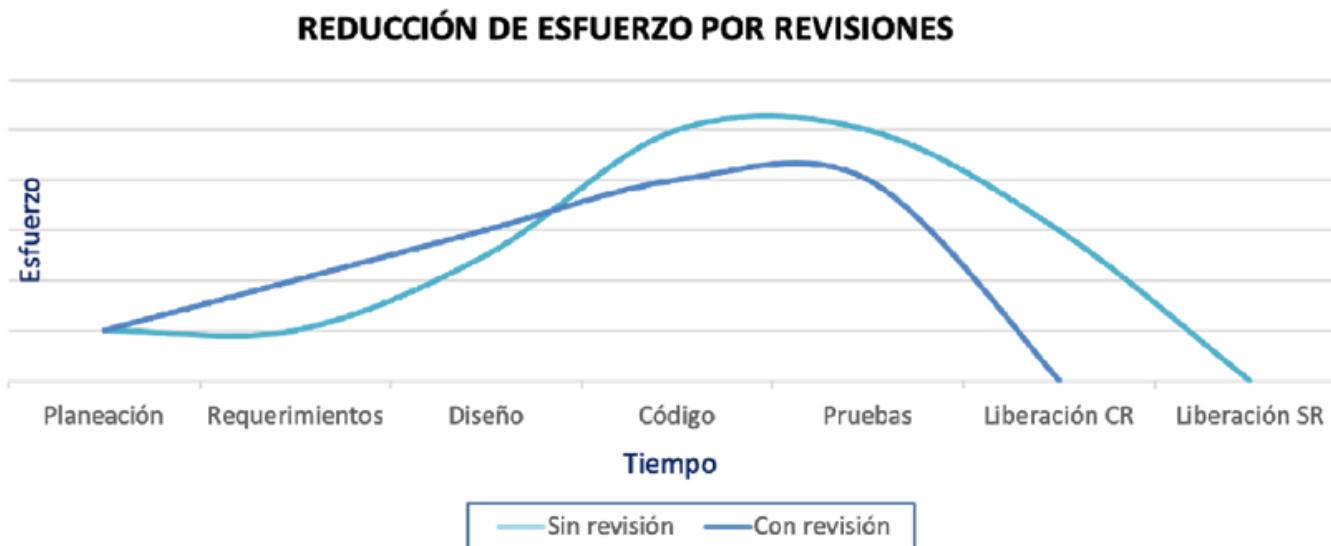


Si no se realizan revisiones, los errores se ampliarán y/o multiplicarán, causando problemas al usuario e incrementando el costo de la reparación. **Las revisiones proveen los siguientes beneficios:**

- Reducción de los defectos en el uso del software.
- Reducción de los recursos de desarrollo, sobre todo en las etapas de codificación y prueba.
- Reducción en los costos de mantenimiento correctivo.

# REVISIONES FORMALES E INFORMALES

Las revisiones técnicas deben aplicarse con un nivel de formalidad apropiado para el producto, el plazo y el personal que realiza el trabajo. La siguiente gráfica indica cómo se pueden reducir los recursos al hacer revisiones:

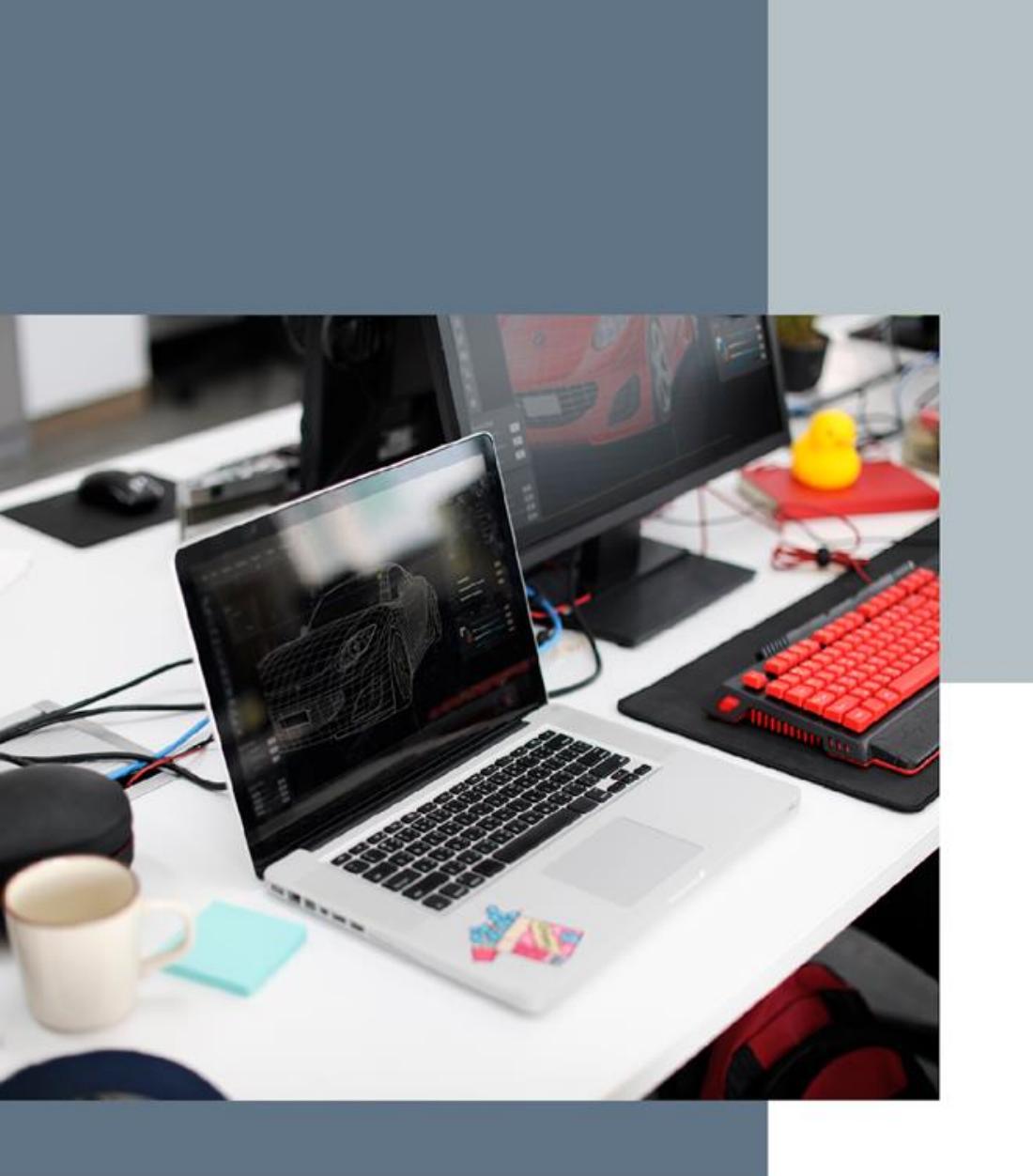


# REVISIONES FORMALES E INFORMALES



Las **cuatro características** más utilizadas, y que contribuyen al incremento de formalidad con la que se efectúa una revisión, son las siguientes:

- **Roles de los participantes.** Se definen explícitamente roles distintos para los revisores.
- **Planeación y preparación.** Se cuenta con una buena planeación y preparación para la revisión del software.



# REVISIONES FORMALES E INFORMALES

- **Estructura de la reunión.** Se define una estructura distinta para la revisión (incluso tareas y productos internos del trabajo).
- **Corrección y verificación.** El seguimiento por parte de los revisores tiene lugar para cualquier corrección que se efectúe.

Por su parte, las **revisiones informales** pueden ser una simple verificación de escritorio o una reunión rápida (de dos o más personas).



# REVISIONES FORMALES E INFORMALES



Sin embargo, como no hay una planeación o preparación por adelantado, ni agenda o estructura de la reunión, y no se da seguimiento a los errores descubiertos, la eficacia de tales revisiones es mucho menor que la de los enfoques más formales. En cambio, una verificación de escritorio sencilla descubre errores que de otro modo se propagarían en el proceso del software, por lo que no deja de ser una revisión.

# REVISIONES FORMALES E INFORMALES



Una forma de mejorar la eficacia de una verificación de escritorio es desarrollar un conjunto de listas de revisión para cada producto grande del trabajo generado por el equipo de software. Las preguntas que se plantean en la lista son generales, pero servirán para guiar a los revisores en la verificación del producto.

# REVISIONES FORMALES E INFORMALES



Una **Revisión Técnica Formal (RTF)** es una actividad del control de calidad del software realizada por ingenieros de software *testers*, o encargados del control de calidad del proyecto.

**Los objetivos de una RTF son:**

- Detectar errores en el funcionamiento, lógica o implementación de cualquier representación del software.
- Verificar que el software que se revisa cumple los requerimientos especificados.

# REVISIONES FORMALES E INFORMALES



- Garantizar que el software está representado de acuerdo con estándares predefinidos.
- Obtener software desarrollado de manera homogénea.
- Realizar proyectos más manejables.
- Capacitar, ya que permite que los ingenieros principiantes observen distintos enfoques de análisis, diseño e implementación del software.

# REVISIONES FORMALES E INFORMALES

Para estructurar una revisión técnica formal se sugiere lo siguiente:

- Un revisor produce la lista de pendientes, o minuta de la revisión. Además, se elabora un reporte técnico formal de la revisión, el cual responde tres preguntas:
  1. ¿Qué fue lo que se revisó?
  2. ¿Quién lo revisó?
  3. ¿Qué se descubrió y a qué conclusiones se llegó?



# REVISIONES FORMALES E INFORMALES



Los lineamientos básicos de una revisión técnica formal son los siguientes:

- **Revisa el producto, no al productor.** Los errores deben señalarse en forma amable; el tono de la reunión debe ser relajado y constructivo; el trabajo no debe apenar o menospreciar a nadie. El líder de la revisión debe conducir la reunión en tono y actitud apropiados, y debe detenerla de inmediato si se sale de control.

# REVISIONES FORMALES E INFORMALES



- **Establece una agenda y síguela.** Una de las fallas clave de las reuniones de todo tipo es la dispersión. El líder de la revisión tiene la responsabilidad de mantener la sesión encarrilada y no debe sentir temor de llamar al orden cuando se dispersen.
- **Limita el debate y las contestaciones.** Cuando el revisor plantea un asunto, quizá no haya acuerdo unánime acerca de su efecto. En lugar de perder tiempo en debatir la cuestión, esta debe registrarse para discutirla después.

# REVISIONES FORMALES E INFORMALES



- **Enuncia áreas de problemas, pero no intentes resolver cada uno.** Una revisión no es una sesión para resolver problemas. La solución de los problemas debe posponerse para después de la reunión de revisión.
- **Toma notas por escrito.** A veces, es buena idea que algún miembro del equipo tome notas, a fin de que la redacción y prioridades sean evaluadas por los demás revisores a medida que la información se registra.

# REVISIONES FORMALES E INFORMALES



- **Limita el número de participantes e insiste en la preparación previa.** Dos cabezas piensan más que una, pero 15 no son necesariamente mejor que 5. Mantenga limitado el número de personas involucradas. Sin embargo, todos los miembros del equipo de revisión deben prepararse. El líder de la revisión tiene que solicitar comentarios por escrito (lo que proporciona un indicador de que el revisor ha inspeccionado el material).

# REVISIONES FORMALES E INFORMALES



- **Desarrolla una lista de verificación para cada producto que sea probable que se revise.** Una lista de verificación ayuda al líder del proyecto a estructurar la RTF, y a cada revisor a centrarse en los aspectos importantes.
- **Asigna recursos y dispón tiempo para las RTF.** Deben programarse como tareas del proceso de software. Además, debe programarse tiempo para hacer las inevitables modificaciones que ocurrirán como resultado de la RTF.

# REVISIONES FORMALES E INFORMALES



- **Da una capacitación significativa a todos los revisores.** Para que una revisión sea eficaz, todos los revisores deben recibir cierta capacitación formal. Esta debe hacer énfasis, tanto en aspectos relacionados con el proceso como en el lado de la psicología humana de la revisión.
- **Revisa las primeras revisiones.** Volver a revisar puede ser benéfico para descubrir problemas con el proceso de revisión en sí mismo.

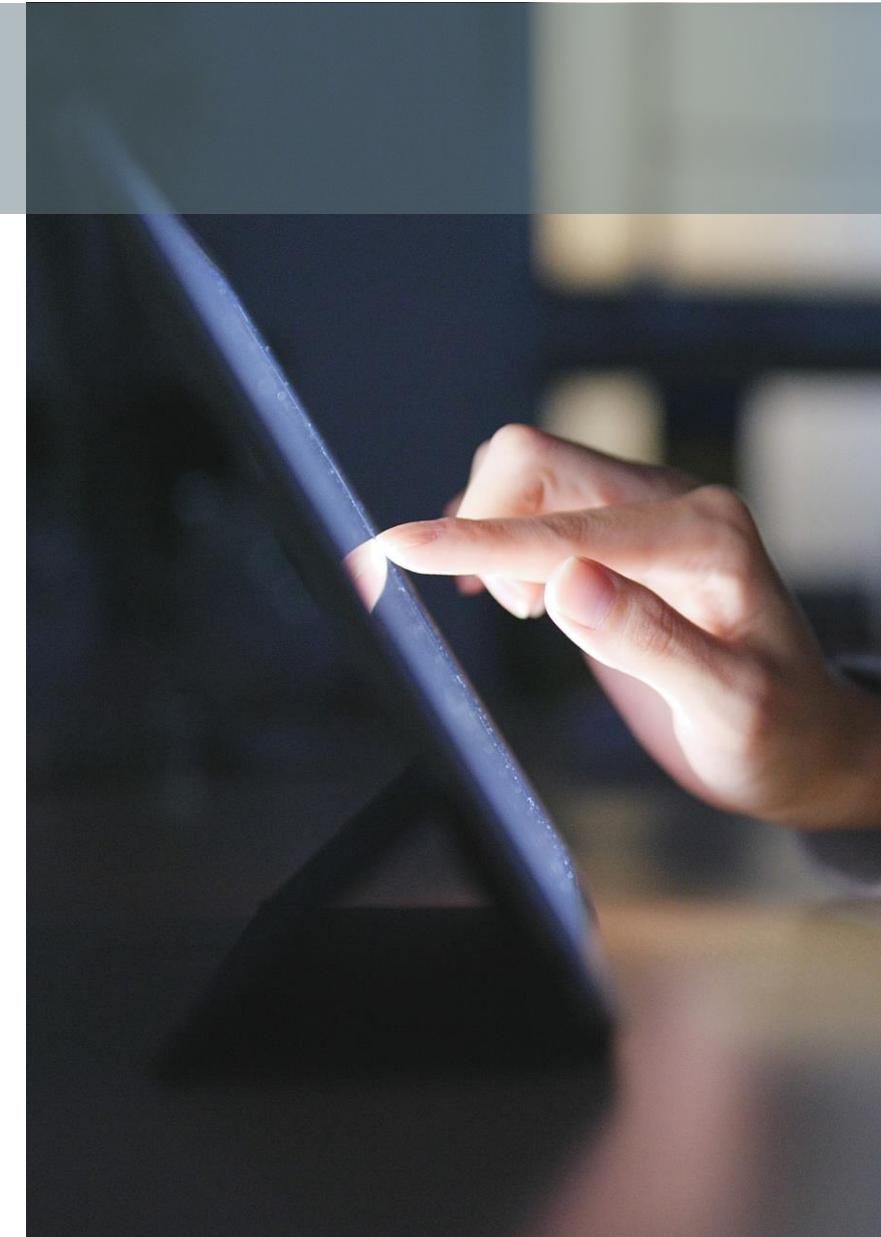
# FORO 1

De acuerdo al caso de aplicación **Mozilla implementa una solución para 2 errores explotados activamente en el navegador Firefox**, el cual puedes ver mediante el siguiente enlace:



- ¿Consideras que la calidad del navegador web mejoró gracias a la solución implementada?
- ¿Qué solución hubieras implementado tu?

Presiona el botón para participar en el foro.



# CONCLUSIÓN

El control de calidad permite ahorrar en el costo, cuando este se realiza desde el principio del proceso de desarrollo de software. El objetivo de toda revisión técnica es detectar errores y descubrir aspectos que podrían tener un efecto negativo en el software que se va a desarrollar. Entre más pronto se descubra y corrija un error, menos probable es que se propague a otros productos del trabajo de ingeniería de software y que se amplifique. Con esto se evita, en su mayoría, que estos errores se conviertan en defectos (y estos, a su vez, en fallos), lo que provocaría un mayor esfuerzo y costo para corregirlos.



# **ASEGURAMIENTO DE LA CALIDAD**

## **UNIDAD 1.**

### **CALIDAD EN LA INDUSTRIA DEL SOFTWARE**

**¡Felicitaciones!**

Acabas de concluir el **primer módulo** de tu curso **Aseguramiento de la Calidad**. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

## UNIDAD 2

# VISIÓN GENERAL DE LAS PRUEBAS DE SOFTWARE



# TEMARIO UNIDAD 2



**2.1**

Pruebas de aplicaciones  
convencionales

**2.2**

Aspectos estratégicos



# INTRODUCCIÓN

En esta segunda unidad veremos diversos tipos de pruebas de software, como lo son las pruebas de caja blanca y caja negra, las pruebas de ruta básica y las de estructura de control. Además, se discutirán patrones para realizar pruebas de software.



# COMPETENCIAS A DESARROLLAR

**El alumno será capaz de identificar y explicar los distintos tipos y patrones de pruebas.**



**El alumno será capaz de integrar aspectos tácticos en su planeación de pruebas.**

# PRUEBAS DE APLICACIONES CONVENCIONALES

Una **prueba de software** es un proceso que se realiza al ejecutar un programa de software, con la intención de encontrar errores. La prueba de software tiene limitantes, tanto teóricos como prácticos.

Desde el punto de vista teórico, la prueba es un problema que llamamos **no decidable**; esto implica, de manera general, que no podemos escribir un programa que pruebe los programas sin intervención humana. Sin embargo, la prueba es automatizable en muchos aspectos.



# PRUEBAS DE APLICACIONES CONVENCIONALES

Desde el punto de vista práctico, la cantidad de posibilidades para probar exhaustivamente un sistema es sencillamente inmanejable; es necesario utilizar técnicas adecuadas para maximizar la cantidad de fallas importantes encontradas con los recursos asignados.

Cada método que se utilice para detectar defectos deja un residuo de defectos más sutiles contra los cuales ese método es ineficaz. Esto es lo que se conoce como la **“Paradoja del pesticida”**.



# PRUEBAS DE APLICACIONES CONVENCIONALES

**Las características de una prueba son:**

- Alta probabilidad de encontrar un error.
- No debe ser redundante.
- No debe ser ni tan simple ni tan compleja.
- Tiene que seleccionarse como la mejor prueba propuesta.



# PRUEBAS DE APLICACIONES CONVENCIONALES



Un caso de prueba, en ingeniería del software, es un conjunto de condiciones o variables bajo las cuales un analista o *tester* determinará si una aplicación, un sistema, un software, o una característica de estos es parcial o completamente satisfactoria.

Un buen caso de prueba tiene una alta probabilidad de encontrar un error aún no descubierto; pero un caso de prueba exitoso es aquel que tiene la certeza de encontrar un error no descubierto.

# PRUEBAS DE APLICACIONES CONVENCIONALES

Se requiere que el *tester* deseche nociones preconcebidas sobre lo correcto, para diseñar casos de prueba a fin de romper el software. El objetivo de probar es encontrar todos los errores. [Pressman, 2010: 95].

Un software, por su parte, debe cumplir con las siguientes características para facilitar las pruebas:



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Operable.** Mientras mejor funcione, más eficientemente puede probarse.
- **Estable.** Mientras menos cambios, menos perturbaciones para probar.
- **Observable.** Lo que ve es lo que prueba.
- **Controlable.** Mientras mejor pueda controlar el software, más podrá automatizar y optimizar las pruebas.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Fácil de descomponerse.** Al controlar el ámbito de las pruebas, es posible aislar más rápidamente los problemas y realizar pruebas nuevas y más inteligentes.
- **Comprensible.** Mientras más información se tenga, se probará con más inteligencia.
- **Simple.** Mientras haya menos que probar, más rápida es la prueba.





# PRUEBAS DE APLICACIONES CONVENCIONALES

Los **objetivos de la prueba de software** son los siguientes:

- **Conocer el nivel de calidad de productos intermedios para actuar a tiempo:** esto permite una administración realista del tiempo de elaboración del producto en cuestión.
- **No pagar por un producto de software sino hasta que tenga el nivel de calidad pactado:** esto eleva la certidumbre en el comprador de software y minimiza riesgos.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Disminuir la penosa y costosa labor de soporte a usuarios insatisfechos:** a consecuencia de liberar un producto inmaduro, y mejorar la imagen de la organización desarrolladora.
- **Reducir costos de mantenimiento:** mediante el diagnóstico oportuno de los componentes del sistema.
- **Obtener información concreta acerca de fallas:** para ser usada como apoyo en la mejora de procesos y en la de los desarrolladores.



# PRUEBAS DE APLICACIONES CONVENCIONALES

A continuación, se presentan los **tipos de pruebas de software** más conocidos: de caja blanca y de caja negra:

- **La prueba de caja blanca.** Asegura la operación interna del programa, revisa las rutas lógicas a través del software y las colaboraciones entre componentes, es decir, se comprueban los caminos lógicos del programa, así como las condiciones y ciclos, examinando el estado del programa en varios puntos.



# PRUEBAS DE APLICACIONES CONVENCIONALES



Las funciones de una prueba de caja blanca son las siguientes:

- Garantizar que, al menos una vez, se revisaron todas las rutas independientes dentro de un módulo.
- Ejecutar todos los bucles en sus fronteras operativas.
- Revisar todas las decisiones lógicas en sus lados verdadero y falso.
- Revisar estructuras de datos internas para garantizar su validez.

# PRUEBAS DE APLICACIONES CONVENCIONALES

Existen dos tipos de pruebas de caja blanca:

- **De estructura de datos locales.** Estudia las variables del programa. Generalmente, se hace una búsqueda de cada variable y se verifica que esté declarada y que no exista otra con el mismo nombre, ni declarada local o globalmente; o bien que haya referencias a todas y cada una de las variables; finalmente, analiza su comportamiento en comparaciones.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **De cobertura lógica.** Se subdivide, a su vez, en las siguientes:
  - ▶ **De cobertura de sentencias:** comprueba que todas las sentencias se ejecuten al menos una vez.
  - ▶ **De cobertura de decisión:** ejecuta casos de prueba, de modo que cada decisión se pruebe al menos una vez a “verdadero” (true) y otra a “falso” (false).



# PRUEBAS DE APLICACIONES CONVENCIONALES

- ▶ **De cobertura de condición:** ejecuta un caso de prueba para “verdadero” y otro para “falso”, por cada condición, teniendo en cuenta que una decisión puede estar formada por varias condiciones.
- ▶ **De cobertura de condición de decisión:** se realizan las pruebas de cobertura de condición y las de decisión a la vez.

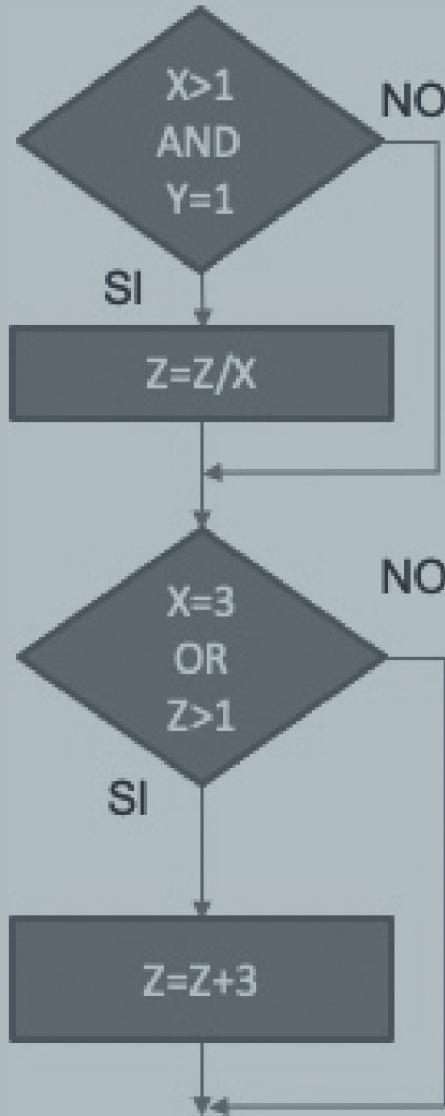


# PRUEBAS DE APLICACIONES CONVENCIONALES



- ▶ **De condición múltiple:** cada decisión multicondición se traduce a una condición simple, aplicando posteriormente la cobertura de decisión.
- ▶ **De cobertura de caminos:** se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Se entiende “camino” como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

# PRUEBAS DE APLICACIONES CONVENCIONALES



Algunos ejemplos de esto, son los siguientes:

- **De cobertura de sentencias:** dado el diagrama de flujo, se propone el siguiente caso de prueba:

$X=3, Y=1, Z=5$

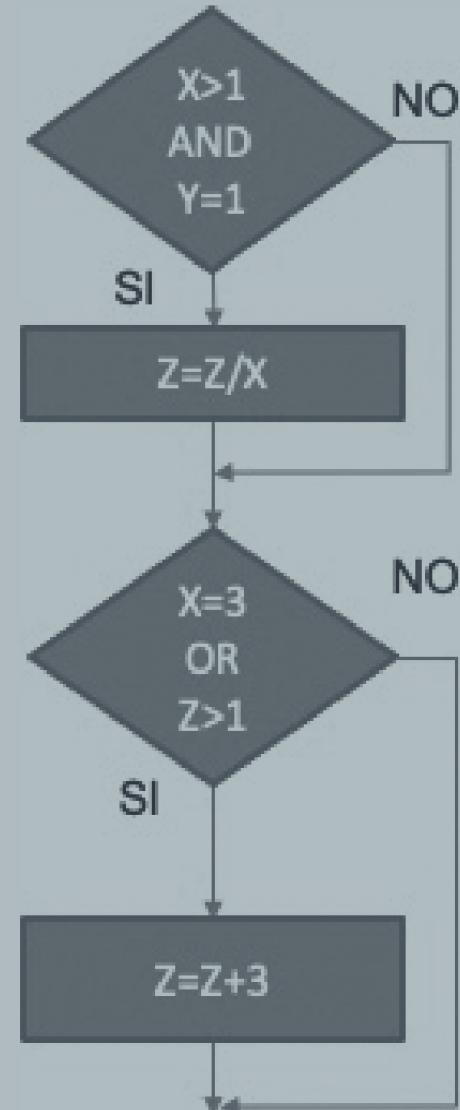
Esto nos muestra que cada sentencia es ejecutada al menos una vez.

# PRUEBAS DE APLICACIONES CONVENCIONALES

- **De cobertura de decisión:** dado el diagrama de flujo, se propone el siguiente caso de prueba:

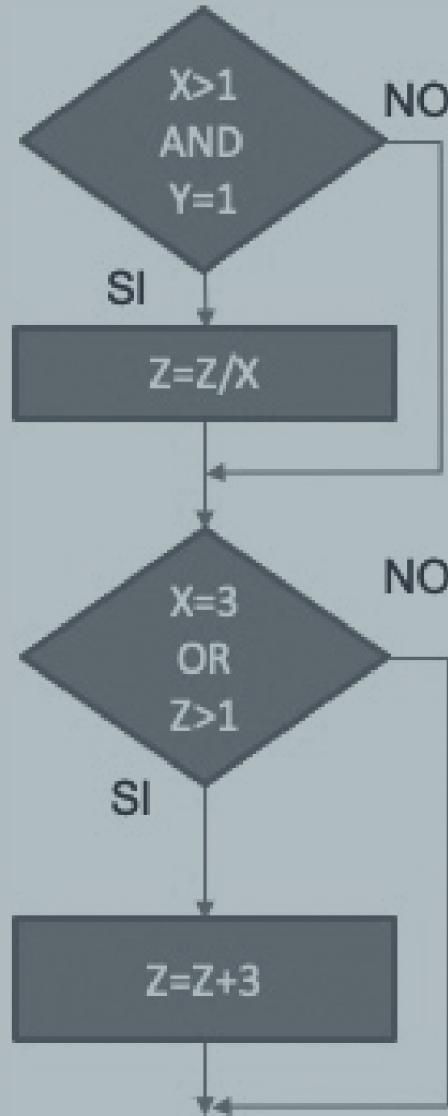
$X=4, Y=1, Z=5$   
 $X=3, Y=2, Z=1$

Al aplicar los valores en los casos de prueba, se muestra que cada decisión ejecutada tenga valores de “verdadero” y “falso”.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **De cobertura de condición:** dado el diagrama de flujo, se propone el siguiente caso de prueba:



X	Y	Z	X>1	Y=1	X=3	Z>1
1	1	5	F	V	F	V
3	2	1	V	F	V	F

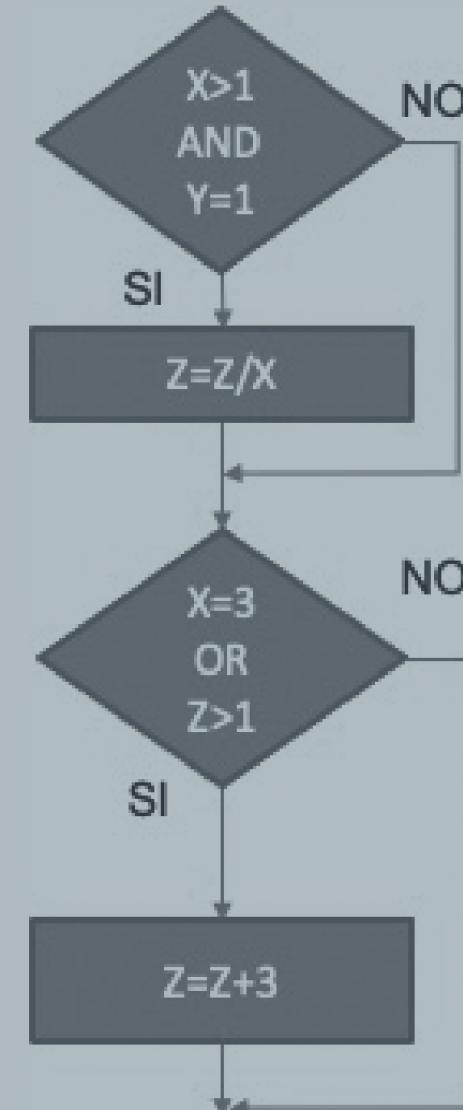
Esto nos muestra que cada condición toma todos los posibles valores al menos una vez. Suele ser mejor a las anteriores.

# PRUEBAS DE APLICACIONES CONVENCIONALES

- **De cobertura de condición de decisión:** dado el diagrama de flujo, se propone el siguiente caso de prueba:

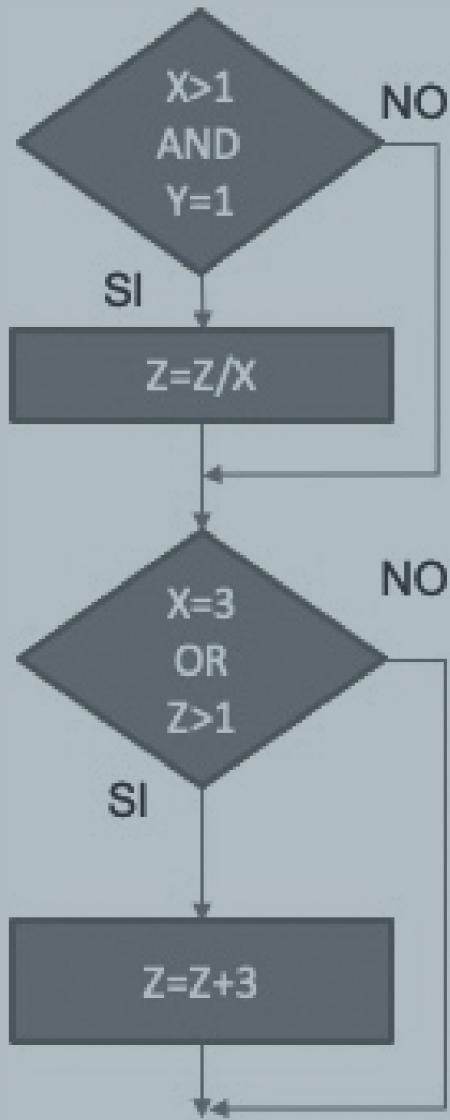
X	Y	Z	X>1	Y=1	X=3	Z>1
3	1	6	V	V	V	V
1	0	1	F	F	F	F

Se muestra que se deben cumplir ambos criterios. En la primera decisión del segundo caso, se toma la alternativa de “falso”, siempre por fallo de las dos premisas a la vez.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **De condición múltiple:** dado el diagrama de flujo, se propone el siguiente caso de prueba:



X	Y	Z	X>1	Y=1	X=3	Z>1
3	1	6	V	V	V	V
1	0	1	F	F	F	F
3	2	1	V	F	V	F
1	1	4	F	V	F	V

Al aplicar los valores en los casos de prueba, se muestra que incluyen todas las anteriores.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **La prueba de caja negra.** En ocasiones, son conocidas como pruebas de comportamiento. Se enfocan en revisar por completo todos los requerimientos funcionales para un programa.

Cabe aclarar que no son una alternativa para las técnicas de caja blanca, sino un complemento. Es probable que se descubra una clase de errores diferentes. Suelen aplicarse durante las últimas etapas de las pruebas.



# PRUEBAS DE APLICACIONES CONVENCIONALES

Las pruebas de caja negra están orientadas a descubrir los siguientes tipos de errores:

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores en las estructuras de datos o en el acceso a datos.
- Errores de comportamiento o rendimiento.
- Errores de inicialización y terminación.



# PRUEBAS DE APLICACIONES CONVENCIONALES



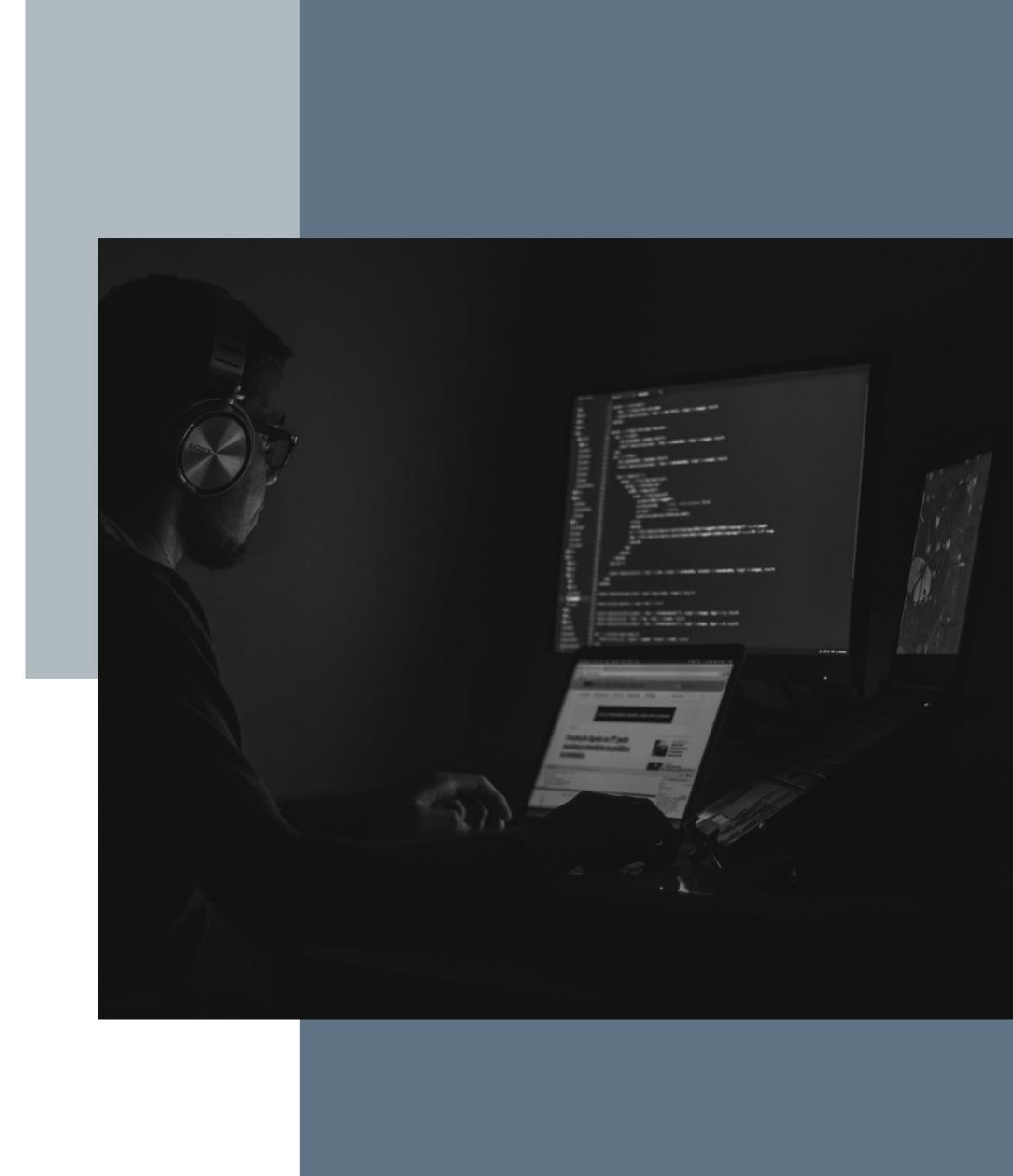
Las pruebas de caja negra deben contestar a las siguientes preguntas:

- ¿Cómo se prueban el comportamiento y el rendimiento del sistema?
- ¿El sistema es particularmente sensible a ciertos valores de entrada?
- ¿Qué tasas y volumen de datos puede tolerar el sistema?
- ¿Qué efecto tendrán sobre la operación del sistema algunas combinaciones de datos?

# PRUEBAS DE APLICACIONES CONVENCIONALES

Por lo general, el desafío de una prueba de caja negra es buscar las combinaciones de entradas y estados del sistema que tengan la más alta probabilidad de encontrar defectos dentro de un conjunto inmenso que no podemos probar exhaustivamente. Para lograrlo, existen **dos técnicas de prueba de caja negra:**

- Partición equivalente y
- Valores límite.



# PRUEBAS DE APLICACIONES CONVENCIONALES

La **prueba de partición equivalente** tiene las siguientes características:

- Se dividen los parámetros de entrada en un conjunto de clases de datos denominado clases de equivalencia.
- Se parte de la premisa de que cualquier elemento de una clase de equivalencia es representativo del resto del conjunto.
- A partir de las clases de equivalencia se obtienen las clases de equivalencia válidas e inválidas.





# PRUEBAS DE APLICACIONES CONVENCIONALES

Para comprender mejor, definamos el término “clase de equivalencia”:

**Clase de equivalencia.** Conjunto de datos de entrada donde el comportamiento del software es igual para todos los datos del conjunto. Las clases pueden ser: **válida**, la cual genera un valor esperado; e **inválida**, que genera un valor inesperado.

Las clases de equivalencia se obtienen de las condiciones de entrada descritas en las especificaciones del software a desarrollar.



# PRUEBAS DE APLICACIONES CONVENCIONALES

La **prueba de valores límite** tiene las siguientes características:

- Se basa en la evidencia experimental de que los errores suelen aparecer con mayor probabilidad en los extremos de los campos de entrada.
- Un análisis de las condiciones límite de las clases de equivalencia aumenta la eficiencia de las pruebas.



# PRUEBAS DE APLICACIONES CONVENCIONALES

Para exemplificar las pruebas de caja negra se plantea lo siguiente:

Una aplicación posee una pantalla para el ingreso de un valor numérico cuyo valor debe estar entre 1 y 1 000. Por lo tanto, todo valor menor que 1, y mayor a 1 000 es inválido.

## Caso 1

Datos de entrada: 765.

Resultado esperado: la aplicación permite el ingreso del dato.



# PRUEBAS DE APLICACIONES CONVENCIONALES

## Caso 2

Datos de entrada: 1 (valor límite).

Resultado esperado: la aplicación permite el ingreso del dato.

## Caso 3

Datos de entrada: 1 000 (valor límite).

Resultado esperado: la aplicación permite el ingreso del dato.





# PRUEBAS DE APLICACIONES CONVENCIONALES

## Caso 4

Datos de entrada: 0.

Resultado esperado: la aplicación no permite el ingreso del dato y muestra un mensaje de error.

## Caso 5

Datos de entrada: 1001.

Resultado esperado: la aplicación no permite el ingreso del dato y muestra un mensaje de error.



# PRUEBAS DE APLICACIONES CONVENCIONALES

Finalmente, además de las pruebas de caja blanca y de caja negra, existen **otros tipos de prueba**. Los siguientes son algunos ejemplos:

- **Pruebas de interfaces gráficas de usuario.** Son conocidas también como GUI (por sus siglas en inglés). Plantean desafíos interesantes a los ingenieros de software. Estas pruebas utilizan listas de chequeo, como las siguientes:



# PRUEBAS DE APLICACIONES CONVENCIONALES

## Para ventanas:

- ¿Se abre la ventana mediante órdenes basadas en el teclado o en un menú?
- ¿Se puede ajustar el tamaño, mover y desplegar la ventana?
- ¿Se regenera adecuadamente cuando se escribe y se vuelve a abrir?





# PRUEBAS DE APLICACIONES CONVENCIONALES

**Para menús emergentes y operaciones con el ratón:**

- ¿Se muestra la barra de menú apropiada en el contexto apropiado?
- ¿Es correcto el tipo, tamaño y formato del texto?
- ¿Si el ratón tiene varios botones, están apropiadamente reconocidos en el contexto?



# PRUEBAS DE APLICACIONES CONVENCIONALES

## Entrada de datos:

- ¿Se repiten y son introducidos adecuadamente los datos alfanuméricos?
- ¿Funcionan adecuadamente los modos gráficos de entrada de datos? (por ejemplo: barra deslizante)?
- ¿Se reconocen adecuadamente los datos no válidos? ¿Son inteligibles los mensajes de entrada de datos?



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Prueba de arquitectura cliente/servidor.**

Esta representa un importante desafío para quienes prueban el software. La naturaleza distribuida de los entornos cliente/servidor, los aspectos de desempeño relacionados con el proceso de transacción, la posible presencia de varias plataformas de hardware diferentes, la complejidad de la comunicación en red, etc., son la mayor complejidad de este software.



# PRUEBAS DE APLICACIONES CONVENCIONALES



Un enfoque de prueba para aplicaciones cliente/servidor es la **prueba de funcionalidad de la aplicación**. Debido a la complejidad del sistema, serán necesarias varias fases:

- **Pruebas de integridad de datos:** son especialmente importantes en el caso de bases de datos distribuidas.
- **Prueba de servidor:** se prueban las funciones de coordinación y manejo de datos del servidor.

# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Prueba de base de datos:** se prueba la exactitud e integridad de los datos almacenados en el servidor.
- **Prueba de transacción o pruebas de red:** se evalúa que cada clase de transacciones se procesa de acuerdo con sus requisitos.
- **Prueba de comunicación de red:** con esta prueba se verifica que el paso de mensajes, las transacciones y el tráfico de la red relacionado se realiza sin errores.





# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Prueba de la documentación y de las funciones de ayuda:** los errores en la documentación son tan devastadores para la aceptación del programa como los errores en los datos o el código fuente. Se aborda en dos fases: en la primera, de revisión e inspección, se examina la claridad editorial del documento. En la segunda fase, la prueba en vivo, se emplea la documentación junto con el programa real.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- **Prueba de sistemas de tiempo real:** el diseñador de caso de prueba no solo debe considerar los casos de prueba convencionales, sino también el manejo de eventos, la temporización de los datos y el paralelismo entre las tareas o procesos que manejan los datos.

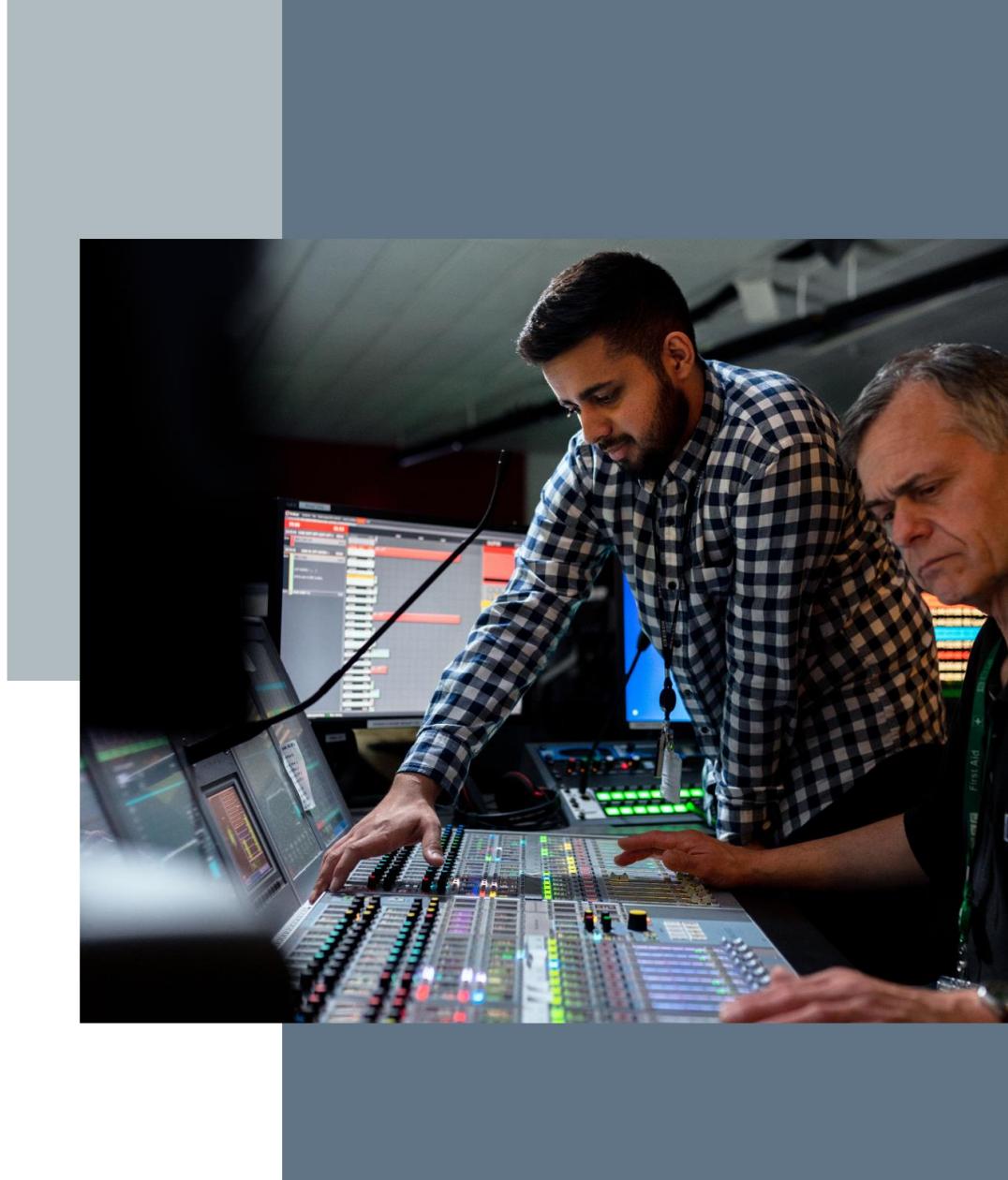
Se puede seguir una **estrategia de cuatro pasos:**



# PRUEBAS DE APLICACIONES CONVENCIONALES

**1. Prueba de tareas.** Consiste en probar cada tarea de manera independiente. Se aplican pruebas de caja blanca y caja negra a cada tarea. Pretende descubrir errores en la lógica y en el funcionamiento.

**2. Prueba de comportamiento.** Con el empleo de modelos del sistema automatizados, se simula el comportamiento del sistema en tiempo real, así como la consecuencia de sucesos externos.





# PRUEBAS DE APLICACIONES CONVENCIONALES

3. **Prueba Inter tareas.** Se prueban las tareas asincrónicas de las cuales se sabe que se comunican entre sí, empleando diferentes tasas de datos y cargas de procesamiento para determinar si ocurrirán errores de sincronización.
4. **Prueba del sistema.** Se aplica un rango completo de pruebas del sistema para tratar de descubrir errores en la interfaz software/hardware.



# PRUEBAS DE APLICACIONES CONVENCIONALES

Una última herramienta para evaluar la calidad del software son los **patrones de prueba**. Estos se describen de manera muy parecida a los de análisis y diseño.

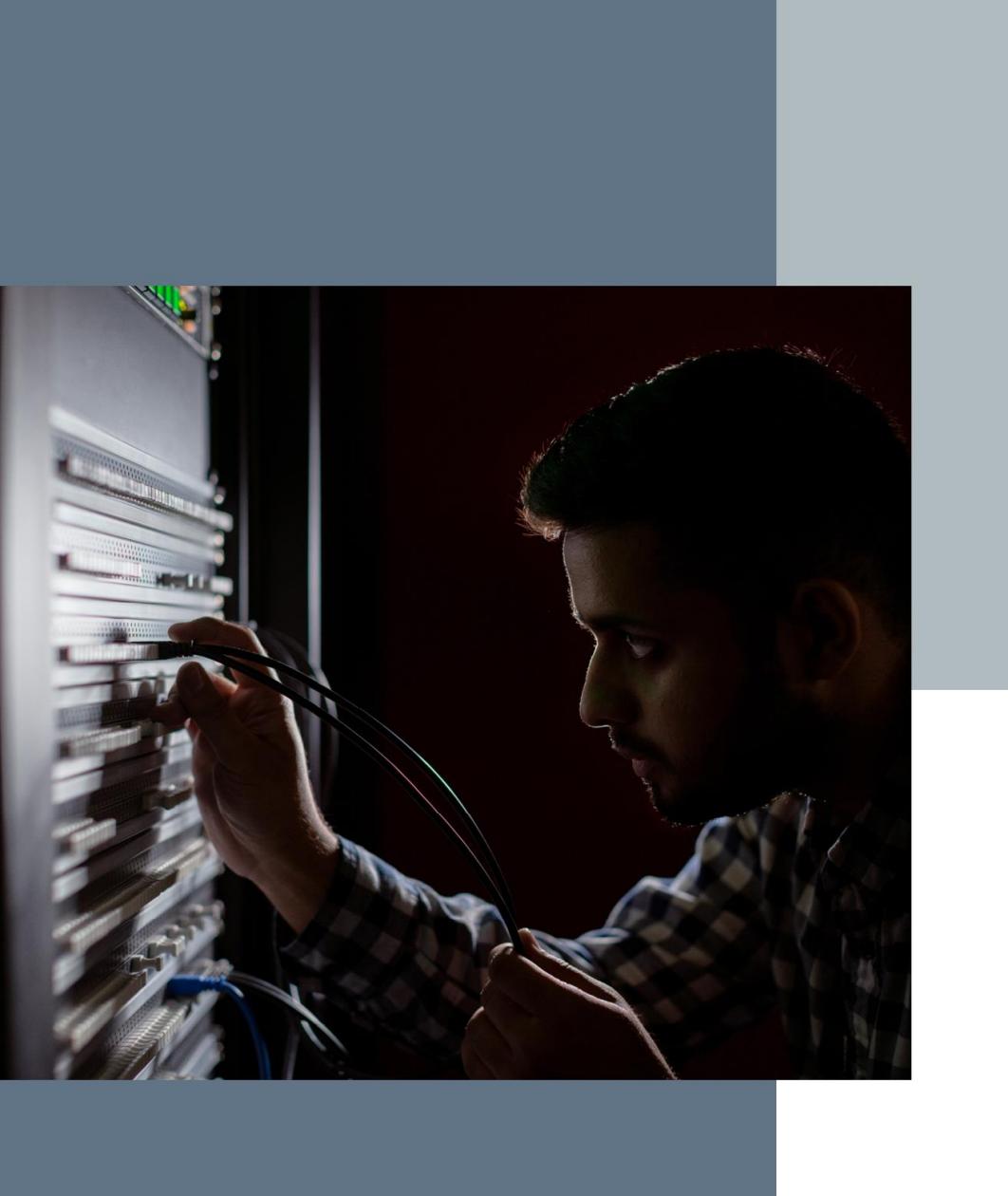
Los patrones de prueba no solo ofrecen a los *testers* una directriz útil cuando empiezan las actividades de prueba, sino que también proporcionan varios beneficios adicionales:



# PRUEBAS DE APLICACIONES CONVENCIONALES

- Proporcionan una terminología a los encargados de la resolución de problemas.
- Concentran la atención en las fuerzas que se encuentran detrás del problema; eso permite a los diseñadores de los casos de prueba comprender mejor las instrucciones.
- Aprender las reglas, algoritmos, estructuras de datos, lenguajes de programación, etc.





# PRUEBAS DE APLICACIONES CONVENCIONALES

- Aprender los principios de programación estructurada, programación modular, programación orientada a objetos, programación genérica, etc.
- Capturan la experiencia y la hacen accesible a los no expertos.
- El conjunto de sus nombres forma un vocabulario que ayuda a que los desarrolladores se comuniquen mejor.



# PRUEBAS DE APLICACIONES CONVENCIONALES

- Los lenguajes de patrones ayudan a comprender un sistema más rápidamente cuando está documentado con los patrones que usa.
- Los patrones pueden ser la base de un manual de ingeniería de software.
- Estimulan el razonamiento iterativo.



# VIDEOS



Te invitamos a ver el siguiente video:

Estrategia de prueba



# ASPECTOS ESTRATÉGICOS



La **prueba** es un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente.

Una **estrategia** de prueba del software debe incluir pruebas de bajo nivel que verifiquen que todos los pequeños segmentos de código fuente se han implementado correctamente, así como pruebas de alto nivel que validen las principales funciones del sistema frente a los requisitos del cliente.

# ASPECTOS ESTRATÉGICOS



La prueba del software es un elemento de un tema más amplio que, a menudo, se le conoce como **verificación y validación (V&V)**. Bohem lo define de la siguiente forma:

- **Verificación.** ¿Estamos construyendo el producto correctamente?
- **Validación.** ¿Estamos construyendo el producto correcto?

# ASPECTOS ESTRATÉGICOS



En cualquier proyecto de software existe un conflicto de intereses evidente; este puede aparecer en cuanto comienza la prueba. Se pide a la gente que ha construido el software que lo pruebe. Sin embargo, los programadores tienen un gran interés en demostrar que el programa está libre de errores, que funciona de acuerdo con las especificaciones del cliente y que estará listo conforme a los plazos y el presupuesto.

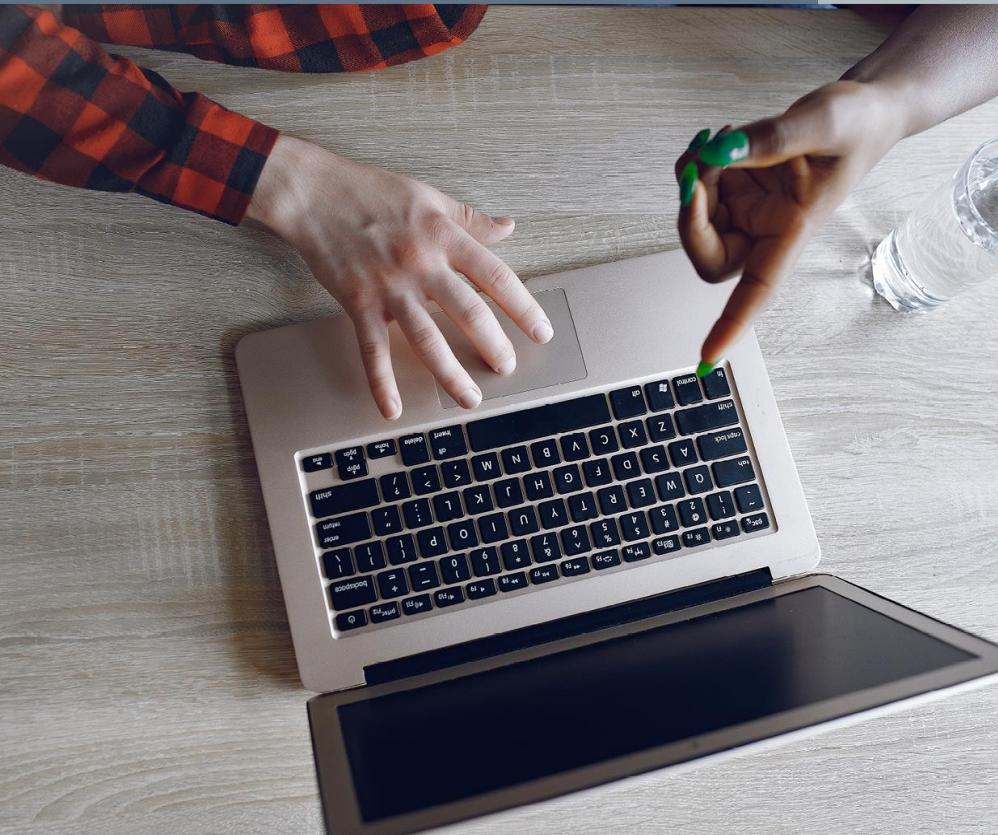
# ASPECTOS ESTRATÉGICOS

La función del grupo independiente de prueba (*testers*) es eliminar los problemas asociados con el hecho de permitir al desarrollador que pruebe lo que ha construido.

Una prueba independiente elimina el conflicto de intereses que, de otro modo, estaría presente. Después de todo, al personal del equipo que forma el grupo independiente se le paga para que encuentre errores.



# ASPECTOS ESTRATÉGICOS



Una **estrategia de pruebas** debe consistir en los siguientes cuatro tipos de pruebas, las cuales deben seguirse de manera secuencial:

- Prueba de unidad
- Prueba de integración
- Prueba de validación
- Prueba de sistema

# ASPECTOS ESTRATÉGICOS

Las pruebas unitarias se enfocan a una sección de código. En el paradigma estructural puede ser una función; o en el paradigma orientado a objetos, una clase. Prueban todo lo que se puede probar en esa sección de código.

- Interfaz
- Estructuras de datos locales
- Condiciones de frontera
- Rutas independientes
- Rutas de manejo de error

Módulo

Casos de prueba



# ASPECTOS ESTRATÉGICOS

Las **pruebas de integración** se ejecutan después de la corrección de defectos. Prueban lo siguiente:

- ▶ Funcionalidad del software.
- ▶ Funcionalidad afectada por el cambio.
- ▶ Componentes que cambiaron.

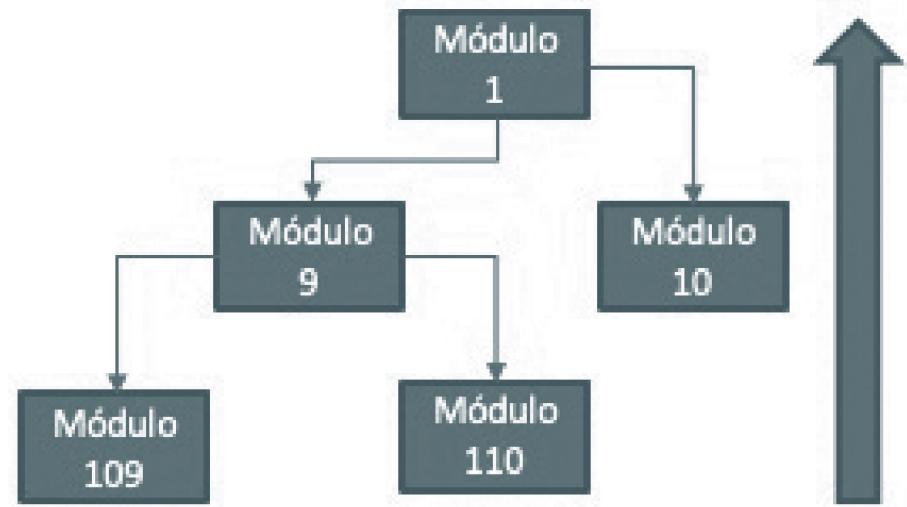
Tal como se muestra en la siguiente diapositiva, pueden ser:

- a) **Ascendentes**, también llamadas atómicas o
- b) **Descendentes**, también llamadas de control.

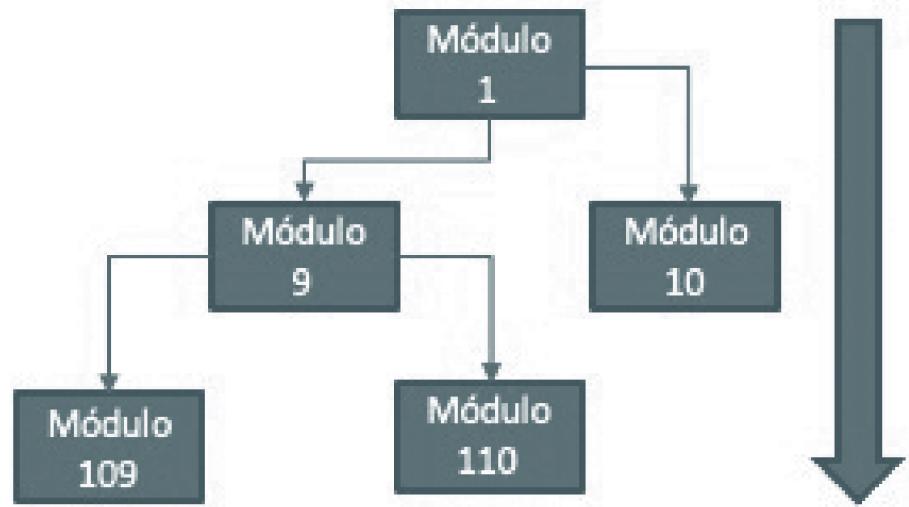


# ASPECTOS ESTRATÉGICOS

**Prueba de integración ascendente (atómica):**



**Prueba de integración descendente (control):**



# ASPECTOS ESTRATÉGICOS



Una vez terminada la prueba de integración, el software está completamente ensamblado como un paquete. Se puede comenzar otra serie de pruebas del software:

- Las **pruebas de validación** se consiguen cuando el software funciona de acuerdo con las expectativas razonables del cliente. Las expectativas razonables están definidas en los requerimientos del software, específicamente en la sección “criterios de validación”.

# ASPECTOS ESTRATÉGICOS



Al finalizar las pruebas de validación, puede darse una de las dos condiciones siguientes:

- Las características están de acuerdo con las especificaciones y son aceptables; o
- Se descubre una desviación de las especificaciones. Las desviaciones descubiertas en esta fase rara vez se pueden corregir antes de la fecha de entrega. Por tanto, se negocia con el cliente el cómo resolverlas.

# ASPECTOS ESTRATÉGICOS

Las pruebas de validación pueden dividirse de la siguiente manera:

## Alfa

- Incluyen usuarios finales representativos.
- Hay un ambiente controlado.
- Se pueden realizar pruebas con colaboración del desarrollador.

## Beta

- Incluyen usuarios finales en sitio con los clientes.
- Hay un ambiente de calidad.
- Se pueden realizar pruebas sin presencia del desarrollador.





# ASPECTOS ESTRATÉGICOS

La **prueba del sistema** verifica todos los elementos del sistema en su conjunto. Para ello, tenemos los siguientes tipos de pruebas dentro del sistema:

- Prueba de recuperación
- Prueba de seguridad
- Prueba de resistencia
- Prueba de rendimiento



# ASPECTOS ESTRATÉGICOS



La **prueba de recuperación** fuerza el fallo y verifica que la recuperación se lleva a cabo apropiadamente. Si la recuperación es automática, hay que evaluar la corrección de la inicialización, de los mecanismos de recuperación del estado del sistema, de la recuperación de datos y del proceso de arranque. Si la recuperación requiere la intervención humana, se evalúan los tiempos medios de reparación para determinar si están dentro de límites aceptables.

# ASPECTOS ESTRATÉGICOS

Durante la **prueba de seguridad**, el responsable desempeña el papel de un individuo que desea entrar en el sistema. ¡Todo vale! Con tiempo y recursos suficientes, una buena prueba de seguridad terminará por acceder al sistema. El papel del diseñador del sistema es hacer que el costo de la entrada ilegal sea mayor que el valor de la información obtenida.



# ASPECTOS ESTRATÉGICOS



En la **prueba de resistencia** se ejecuta un sistema, de manera que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo, diseñar pruebas que generen diez interrupciones por segundo, cuando las normales son una o dos, o búsquedas excesivas de datos residentes en disco, etc. Esencialmente, el responsable de la prueba intenta “romper el programa”.

# ASPECTOS ESTRATÉGICOS



La **prueba de rendimiento** prueba el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Se involucra el hardware y software. Mide la utilización de recursos (por ejemplo: ciclos de procesador) de un modo exacto. Puede monitorizar los intervalos de ejecución, los sucesos ocurridos (por ejemplo: interrupciones) y muestras de los estados de la máquina en un funcionamiento normal.

# ACTIVIDAD 1



Te invitamos a realizar la siguiente actividad:

Presiona el botón para descargar la actividad:



Presiona el botón para entregar la actividad:



# CONCLUSIÓN

El principal objetivo de la prueba en aplicaciones convencionales es que sea capaz de sacar a la luz diferentes clases de errores, minimizando la cantidad de tiempo y esfuerzo invertido. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto. Es decir, una prueba tiene éxito si descubre un error no detectado hasta entonces. Para ser más eficaces, deben ser aplicadas por un equipo independiente.

La experiencia parece indicar que, donde hay un defecto hay otros; es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto. Siempre se ha considerado que las pruebas son una labor destructiva y rutinaria, pero, en realidad, son una tarea tan o más creativa que el desarrollo de software.



# **ASEGURAMIENTO DE LA CALIDAD**

## **UNIDAD 2.**

### **VISIÓN GENERAL DE LAS PRUEBAS DE SOFTWARE**

**¡Felicitaciones!**

Acabas de concluir el **segundo módulo** de tu curso **Aseguramiento de la Calidad**. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

# UNIDAD 3

# PRUEBAS NO CONVENCIONALES



# TEMARIO UNIDAD 3



**3.1**

Pruebas para aplicaciones  
orientadas a objetos

**3.2**

Pruebas para aplicaciones web



# INTRODUCCIÓN

En esta tercera unidad se sigue persiguiendo el objetivo de encontrar errores en un sistema de software. Sin embargo, cambian las estrategias. Esta unidad enfoca su contenido a sistemas orientados a objetos y a aplicaciones web.



# COMPETENCIAS A DESARROLLAR

El alumno podrá implementar pruebas para evaluar la calidad de aplicaciones orientadas a objetos.



El alumno podrá implementar pruebas para evaluar la calidad de aplicaciones web.

# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

Con la creación de modelos de requerimientos (análisis) y de diseño comenzamos la construcción de software orientado a objetos. Estos, a su vez, se tornan como representaciones relativamente informales de los requisitos de sistema, y evolucionan hacia modelos detallados de clases, relaciones de clase, diseño y asignación de sistema, y diseño de objetos (que incorpora un modelo de conectividad de objetos mediante mensajería).



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

El **Análisis Orientado a Objetos** (AOO) es una serie de técnicas y métodos para entender un problema; estas técnicas ayudan a entender el problema que se desea resolver. Además, nos ayudan a diseñar una solución a este problema, en donde empiezan a funcionar aspectos del diseño orientado a objetos.

El AOO implementa dos modelos principales para plantear la estructura de un problema, los cuales son: el **modelo de casos de uso** y el **modelo del dominio**.



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS



- **Los modelos de casos de uso** nos ayudan a identificar la interacción entre el usuario y el sistema; describiendo el rol que debe tener cada usuario en los escenarios del sistema.
- Mientras que **el modelo del dominio** se encarga de generalizar el vocabulario implementado en el sistema; lo que permite tener, en términos entendibles, todas las definiciones del sistema.

# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

Después de pasar por el AOO, se inicia el proceso de desarrollo del software, en donde implementamos el **Diseño Orientado a Objetos (DOO)**. A partir de este, se construye un modelo de objetos, que implementa **tarjetas CRC (Clase - Responsabilidad -Colaboración)**, en las que se especificarán las colaboraciones y las responsabilidades de las clases.

A partir de aquí, es posible iniciar las pruebas. Algunos tipos son los siguientes:



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

- **Prueba de unidad en el contexto OO.** En lugar de módulos individuales, la menor unidad a probar es la clase u objeto encapsulado.
- **Prueba de integración en el contexto OO.** Debido a que el software orientado a objetos no tiene una estructura de control jerárquica, las estrategias convencionales de integración ascendente y descendente poseen un significado muy pequeño.



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS



Por lo que, en este contexto, examina un conjunto de clases que se requieren para responder a un evento dado. Utiliza dos nuevas pruebas: las basadas en hilos o hebras y las basadas en uso.

- **Prueba de validación en un contexto OO.**  
La validación del software se enfoca en las acciones visibles al usuario, y salidas del sistema, reconocidas por él. Se recurre mucho a los casos de uso para poder apoyarse en esta prueba.

# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

- **Implicaciones de los conceptos OO para el diseño de casos de prueba.** La clase OO es el objetivo para el diseño de los casos de prueba. Debido al encapsulamiento de atributos y operaciones, se complica un poco la elaboración de dichas pruebas.
- **Aplicabilidad de métodos convencionales de diseño de casos de prueba.** Los métodos de caja blanca y caja negra pueden aplicarse a las operaciones que se definen en una clase.

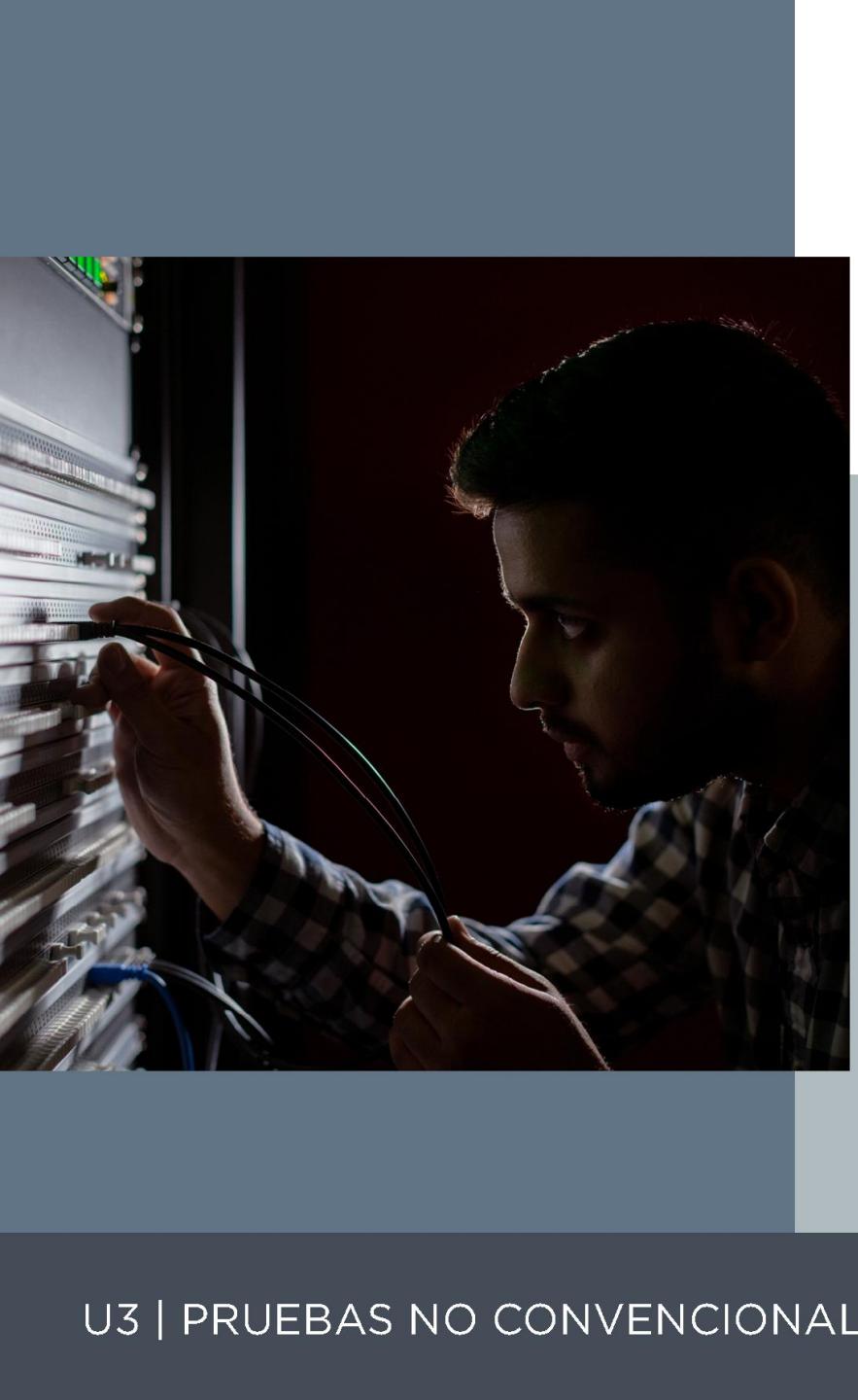


# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

- **Pruebas basadas en fallo.** Su objetivo es diseñar pruebas que posean una alta probabilidad en la detección de errores posibles.

Además de lo anterior, existen varias formas en las que la Programación Orientada a Objetos (POO) impacta en la realización de las pruebas:



A photograph of a man with dark hair and a beard, wearing a plaid shirt, working on a server rack in a dimly lit room. He is holding a black cable and appears to be connecting it to a server unit. The server rack has multiple drive bays and cables visible.

# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

- **Dependiendo del enfoque de la POO.** Algunos tipos de errores se tornan menos posibles, algunos se tornan mas posibles y aparecen nuevos tipos de errores.
- **Casos de prueba y jerarquía de clases.** La herencia no obvia la necesidad de ejecución de pruebas completas en todas las clases derivadas. De hecho, esto puede complicar el proceso de prueba.



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

## Diseño de pruebas basadas en escenarios.

Los resultados de pruebas basadas en errores no capturan dos tipos principales de errores: primero, especificaciones incorrectas; y, segundo, interacciones entre subsistemas.

Finalmente, hay dos métodos de prueba aplicables en el nivel clase, los cuales son los siguientes:



# PRUEBAS PARA APLICACIONES ORIENTADAS A OBJETOS

- **Pruebas aleatorias para clases OO.** Estas pruebas deben atacar una clase específica, a la vez que identifican todos los métodos asociados.
- **Pruebas de partición a nivel clase.** Las pruebas de partición reducen el número de casos de prueba necesarios para ejercitar la clase. Las particiones basadas en estados categorizan las operaciones de clases basándose en su habilidad para cambiar de estados.



# PRUEBAS PARA APLICACIONES WEB



En el caso de las aplicaciones web, además de las pruebas tradicionales, se **evalúan otros seis elementos:**

- Contenido
- Interfaz
- Navegación
- Configuración
- Seguridad
- Rendimiento

# PRUEBAS PARA APLICACIONES WEB

La **prueba de contenido** tiene como objetivos descubrir:

- Errores sintácticos (ortografía y gramática de contenido).
- Errores semánticos (exactitud de la información presentada y consistencia entre objetos de contenido y objetos relacionados).
- Errores en la organización o estructura del contenido que se presenta al usuario final.



# PRUEBAS PARA APLICACIONES WEB



*E/ tester* debe responder a las siguientes preguntas:

- ¿La información realmente es precisa, es decir: concisa y exacta?
- ¿La plantilla del objeto de contenido es fácil de entender para el usuario?
- ¿La información anidada en un objeto de contenido se encuentra con facilidad?

# PRUEBAS PARA APLICACIONES WEB



- ¿La información presentada internamente es consistente con la información de otros objetos de contenido?
  - ¿El contenido es ofensivo, engañoso o abre la puerta a pleitos?
  - ¿El contenido infringe derechos de autor o marcas registradas?
- ¿El estilo estético del contenido entra en conflicto con el estilo estético de la interfaz global?

# PRUEBAS PARA APLICACIONES WEB

La **prueba de la interfaz** de usuario tiene como objetivo descubrir errores que resultan de una pobre implementación de interacción, omisiones, inconsistencias o ambigüedades. Se realiza en tres puntos del proceso:

- En el análisis (formulación y análisis de requisitos).
- En el diseño (al diseñar la interfaz que garantice la calidad).
- Durante las pruebas donde se ejecuta la aplicación.



# PRUEBAS PARA APLICACIONES WEB



Las pruebas de interfaz buscan:

- Asegurar que las reglas del diseño, la estética y el contenido visual no tengan error.
- Descubrir errores de semántica y facilidad de uso por cada caso de uso.
- Probar la compatibilidad del software en una diversidad de ambientes.

# PRUEBAS PARA APLICACIONES WEB



La **prueba de navegación** busca garantizar que todos los mecanismos que permiten al usuario navegar a través de la aplicación web funcionen correctamente. Dentro de los mecanismos que se prueban están:

- Vínculos de navegación
- Redirección
- *Bookmarks*
- Mapas del sitio
- Motores de búsqueda internos

# PRUEBAS PARA APLICACIONES WEB

Las **pruebas de configuración** se aplican para descubrir errores de desempeño que se presentan debido a la falta de recursos del lado del servidor, por el ancho de banda de red inapropiado, por las capacidades inadecuadas de bases de datos y por las debilidades en el sistema operativo. Se realizan con la finalidad de comprender cómo responde el sistema ante las cargas como: número de usuarios, número de transacciones y volumen de datos globales.



# PRUEBAS PARA APLICACIONES WEB



Las **pruebas de seguridad** están diseñadas para probar las vulnerabilidades del lado del cliente, así como en las comunicaciones de red que ocurren mientras se pasan los datos del cliente al servidor, y viceversa.

Existe un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.



# PRUEBAS PARA APLICACIONES WEB

Se trata del “proyecto abierto de seguridad de aplicaciones web” (OWASP: *Open Web Application Security Project*, en inglés), el cual divide las pruebas en 10 subcategorías, que se mencionan a continuación:

- Recopilación de información.
- Pruebas de gestión de la configuración.



# PRUEBAS PARA APLICACIONES WEB

- Pruebas de la lógica de negocio.
- Pruebas de autenticación.
- Pruebas de autorización.
- Pruebas de gestión de sesiones.
- Pruebas de validación de datos.
- Pruebas de denegación de servicio.
- Pruebas de servicios web.
- Pruebas de AJAX.



# PRUEBAS PARA APLICACIONES WEB



Finalmente, las **pruebas de rendimiento** sirven para:

- Validar y verificar atributos de la calidad del sistema: escalabilidad, fiabilidad, uso de los recursos.
- Comparar dos sistemas para saber cuál de ellos funciona mejor.
- Medir qué partes del sistema o de carga de trabajo provocan que el conjunto rinda mal.

Las dos principales son las pruebas de **carga** y **estrés**.

# PRUEBAS PARA APLICACIONES WEB



Una **prueba de carga** se realiza, generalmente, para observar el comportamiento de una aplicación bajo una cantidad esperada de peticiones. Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación. Si también se monitorizan otros aspectos como la base de datos, el servidor de aplicaciones, etc., entonces esta prueba puede mostrar el cuello de botella en la aplicación.

# PRUEBAS PARA APLICACIONES WEB

Una **prueba de estrés** se utiliza, normalmente, para buscar “romper la aplicación”. Se va doblando el número de usuarios que se agregan a esta, y se ejecuta una prueba de carga, hasta que se “rompe”. Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema.



# LECTURAS



Pruebas de seguridad en aplicaciones web según OWASP



Plan de pruebas

# ACTIVIDAD 2



Te invitamos a realizar la siguiente actividad:

Presiona el botón para descargar la actividad:



Presiona el botón para entregar la actividad:



# CONCLUSIÓN

Recalcamos que el objetivo de las pruebas orientadas a objetos sigue siendo el mismo que el de las pruebas convencionales. Sin embargo, en el entorno orientado a objetos, el foco de las pruebas ya no está en los módulos, sino en las clases.

Por otra parte, una aplicación web se implementa en diversas configuraciones ambientales, por lo que debe probarse la compatibilidad con cada configuración, además de la seguridad y capacidad de transacciones. Por lo general, la aplicación web debe probarse en una población controlada y monitoreada de usuarios finales. Esto para descubrir errores relacionados con la facilidad de uso, compatibilidad, confiabilidad y desempeño.



# **ASEGURAMIENTO DE LA CALIDAD**

## **UNIDAD 3.**

### **PRUEBAS NO CONVENCIONALES**

**¡Felicitaciones!**

Acabas de concluir el **tercer módulo** de tu curso **Aseguramiento de la Calidad**. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

# UNIDAD 4

# PLANEACIÓN Y AJUSTE



# TEMARIO UNIDAD 4



**4.1**  
Plan de pruebas

**4.2**  
Incidentes y control de cambios



# INTRODUCCIÓN

En esta cuarta unidad, se explica la importancia de realizar un plan de pruebas, así como la estructura y proceso para realizar los casos de prueba.

Posteriormente, analizaremos el ciclo de vida de una incidencia, su administración y la gestión de cambios asociada a esta. Todo con el objetivo de que el proyecto no se salga de control.



# COMPETENCIAS A DESARROLLAR

El alumno será capaz de realizar un plan de pruebas e integrar casos de prueba bien estructurados.



El alumno podrá gestionar el control de cambios a partir de los resultados de un plan de pruebas.



# PLAN DE PRUEBAS

Para desarrollar software de calidad y libre de errores, el plan de pruebas y los casos de prueba son muy importantes. El **Software Test Plan (STP)** se diseña para determinar el ambiente de aplicación de los recursos, así como el calendario de las actividades de las pruebas. Se debe identificar el dominio y sus características a probar, lo mismo que el tipo de pruebas a realizar.

El **estándar 829 de la IEEE** es el indicado a seguir al diseñar un plan de pruebas.



# PLAN DE PRUEBAS

Un **caso de prueba** es un conjunto de acciones con resultados y salidas previstas. Se basa en los requisitos de especificación del sistema. Sus componentes son:

- **Propósito** de la prueba o descripción del requisito que se está probando.
- **Método** o forma como se probará.
- **Versión** o configuración de la prueba, versión de la aplicación en prueba, el hardware, el software, el sistema operativo, los archivos de datos, entre otros.



# PLAN DE PRUEBAS

- **Resultados y acciones** esperados, o entradas y salidas.
- **Documentación** de la prueba y sus anexos.

Un caso de prueba debe cumplir con los siguientes **factores de calidad**:

- **Correcto.** Debe ser apropiado para los probadores y el entorno. Si teóricamente es razonable, pero exige algo que ninguno de los probadores tiene, caerá por su propio peso.



# PLAN DE PRUEBAS

- **Exacto.** Debe demostrar que su descripción se puede probar.
- **Económico.** Debe tener solo los pasos o los campos necesarios.
- **Confiable y repetible.** Debe ser un experimento controlado con el que se obtiene el mismo resultado cada vez que se ejecute.
- **Rastreable.** Debe saber qué requisitos del caso de uso se prueban.
- **Medible.** Debe verificar si se cumple un estándar.



# PLAN DE PRUEBAS

Un caso de prueba bien diseñado tiene gran posibilidad de llegar a resultados más fiables y eficientes, mejorar el rendimiento del sistema, así como reducir los costos, en **tres categorías**:

- 1. Productividad:** menos tiempo para escribir y mantener los casos.
- 2. Capacidad de prueba:** menos tiempo para ejecutarlos.
- 3. Programar la fiabilidad:** estimaciones más fiables y efectivas.



# PLAN DE PRUEBAS

Las **pruebas automatizadas** tienen como objetivo detectar fallas en el software. Sin embargo, evitan que una persona tenga que ejecutar las pruebas manualmente.

En este caso, el experto en *testing* genera un caso a probar utilizando una herramienta, para que luego la misma se realice automáticamente.



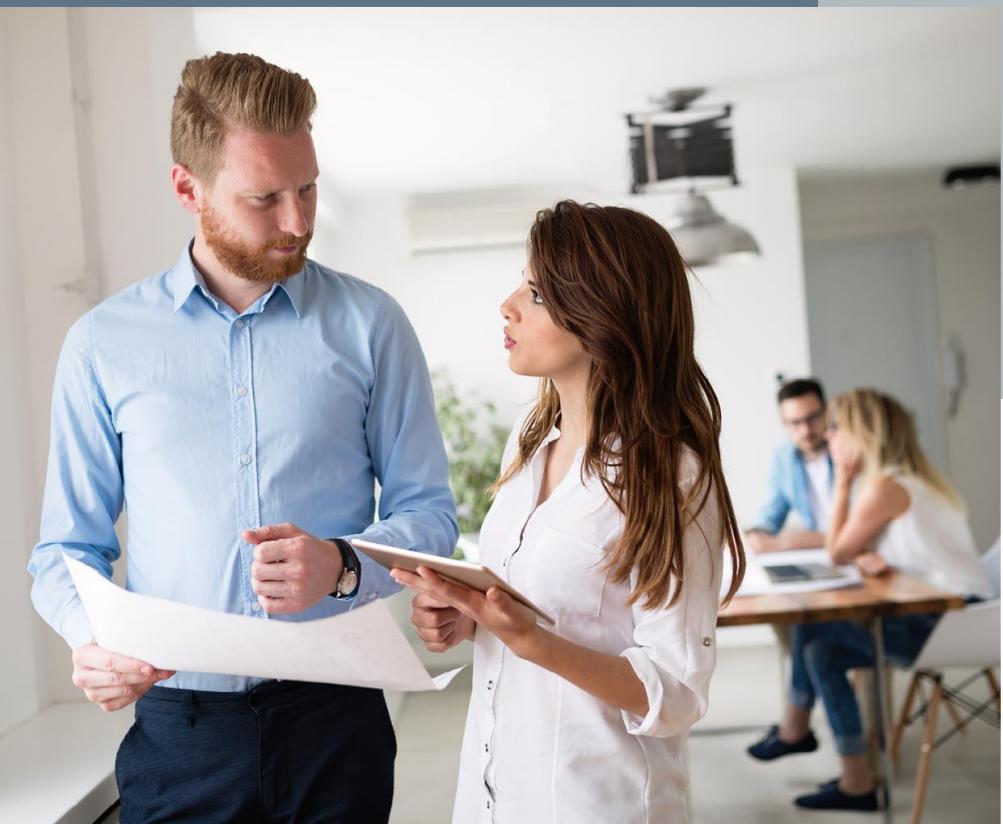
# PLAN DE PRUEBAS

Algunos ejemplos de herramientas para automatizar pruebas son los siguientes:

- Selenium
- Watir
- SoapUI
- Junit
- TestNG
- Nunit
- Testoob
- CSUnit
- HTMLUnit
- PHPUnit
- Telerik
- Microsoft Test Manager



# INCIDENCIAS Y CONTROL DE CAMBIOS



Una **incidencia** se define como un acceso, intento de acceso, uso, divulgación, modificación o destrucción no autorizada de información. Así como un impedimento en la operación normal de las redes, sistemas o recursos informáticos; o una violación de la política de seguridad, entre otras.

Es decir, una incidencia es cualquier ocurrencia de un suceso que requiere investigación [Según IEEE 1008].

# INCIDENCIAS Y CONTROL DE CAMBIOS



**El ciclo de vida de una incidencia se divide en diferentes fases:**

- Fase inicial (preparación y prevención; y detección y preanálisis).
- Contención, erradicación y recuperación (notificación, análisis, contención y erradicación).
- Recuperación del incidente (recuperación).
- Actividad después del incidente (reflexión y documentación).



# INCIDENCIAS Y CONTROL DE CAMBIOS

Los modelos de incidencia permiten optimizar el proceso de resolución para incidencias que se repiten.

## Un modelo de incidencia debería incluir:

- Los pasos a seguir para la resolución de la incidencia.
- El orden cronológico de los pasos y sus dependencias, si los hubiera.
- Quién debe hacer qué.
- Plazos para las actividades.
- Quién debe contactarse y cuándo.

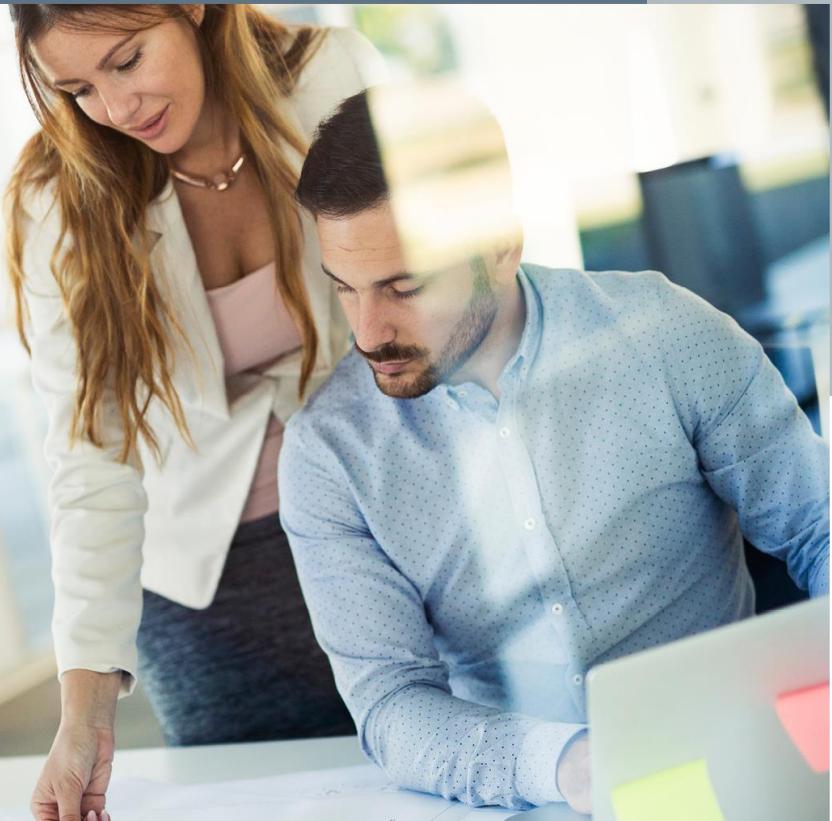


# INCIDENCIAS Y CONTROL DE CAMBIOS

Por otra parte, el objetivo del proceso de gestión de cambios es controlar el ciclo de vida de todos los cambios.

**ITIL** (*IT Infrastructure Library*) define un cambio como:

“La adición, modificación o eliminación de cualquier cosa que pueda tener un efecto en los servicios de TI. El alcance debe incluir cambios en todas las arquitecturas, procesos, herramientas, métricas y documentación, así como cambios en TI, servicios y otros elementos de configuración”.



# INCIDENCIAS Y CONTROL DE CAMBIOS



Los **cambios** pueden generarse en todas las formas y tamaños diferentes. Se dividen en cambios estándar, normales y de emergencia.

Por su parte, las aprobaciones necesarias y las tareas ejecutadas pueden variar significativamente según los elementos de configuración que se modifiquen. Existen cambios menores de rutina que pueden implementarse rápidamente y con una supervisión mínima.



# INCIDENCIAS Y CONTROL DE CAMBIOS



Además, hay cambios de emergencia que deben implementarse lo más rápido posible y, por lo tanto, están sujetos a un proceso de aprobación acelerado y condensado.

# INCIDENCIAS Y CONTROL DE CAMBIOS



De igual manera, se utilizan tipos especiales de documentos en gestión de cambios. En ITIL se proporciona una plantilla de **solicitud de cambio (Request for Change, RFC)** estandarizada, y se utiliza un registro de cambios para documentar el ciclo de vida de un solo cambio, desde la solicitud hasta el cierre y la revisión.



# INCIDENCIAS Y CONTROL DE CAMBIOS



La **gestión de cambios organizativos** es una disciplina que aborda los cambios de gestión en la estructura, estrategia, políticas, procedimientos o cultura de una empresa.

Sin embargo, la gestión de cambios enfocada al área de desarrollo de software es diferente. **ITIL Change Management** se enfoca en gestionar las implicaciones de servicio de los cambios técnicos.



# INCIDENCIAS Y CONTROL DE CAMBIOS



La gestión de cambios se entiende mejor a través de los **cinco objetivos para la gestión eficaz del proceso de cambio, proporcionados por ITIL:**

- El proceso de gestión de cambios debe responder a los requisitos comerciales cambiantes del cliente, al tiempo que maximiza el valor y reduce los incidentes, las interrupciones y la repetición del trabajo.

# INCIDENCIAS Y CONTROL DE CAMBIOS



- Responder a las solicitudes de cambio del negocio y de TI que alinearán los servicios con las necesidades del negocio.
- Asegurar que los cambios se registran y evalúan; y que los cambios autorizados se priorizan, planifican, prueban, implementan, documentan y revisan de manera controlada.
- Asegurar que todos los cambios en los elementos de configuración se registren en el sistema de gestión de la configuración.

# INCIDENCIAS Y CONTROL DE CAMBIOS



- Optimizar la gestión de cambios para el riesgo empresarial, aceptando, ocasionalmente, niveles razonables de riesgo, debido a los posibles beneficios asociados.

Los cambios no controlados o mal administrados pueden tener consecuencias importantes en TI, especialmente si estos conducen a una interrupción que afecte a la empresa, o un incidente importante.

# LECTURAS



Gestión de incidencias y sus principales actividades según ITIL



Pruebas de seguridad OWASP



Plan de pruebas

# CONCLUSIÓN

En esta unidad revisamos las diferencias entre una incidencia y un cambio. Entre los temas más relevantes que se trataron destacan el plan de pruebas, los casos de prueba y la automatización de pruebas. A su vez, se recomendaron herramientas para realizar una automatización de pruebas. Además, se plantea por qué es importante y cómo se gestiona el proceso de solicitud de cambios. Finalmente, se recomiendan fuentes externas, como ITIL, que servirán en el proceso de aprendizaje continuo.

# ASEGURAMIENTO DE LA CALIDAD

## UNIDAD 4.

### PLANEACIÓN Y AJUSTE

**¡Felicitaciones!**

Acabas de concluir el **cuarto módulo** de tu curso **Aseguramiento de la Calidad**. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.



# PROYECTO FINAL

# PROYECTO FINAL



Te invitamos a realizar el siguiente proyecto final:

Presiona el botón para descargar el proyecto final:



Presiona el botón para entregar el proyecto final:

