

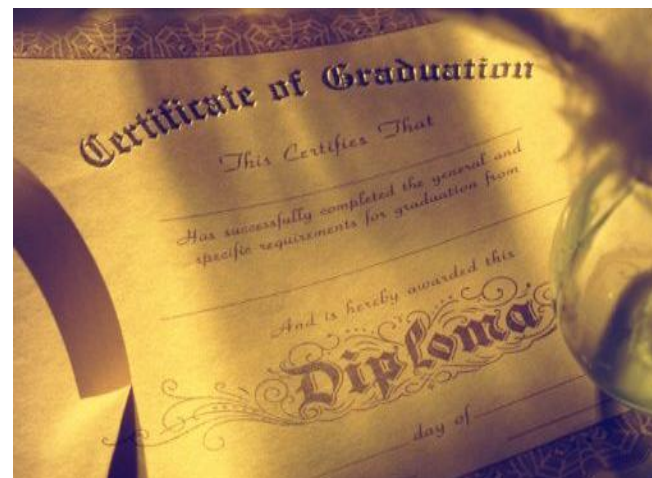


Seminarios y Formación

Tester Certificado Nivel Básico

Formación para el “Tester Certificado – Nivel Básico” de acuerdo al programa de estudios 2005 del ISTQB

V1.93_español



0 – Introducción

01 - Presentación

■ Organización del Curso

- Agenda del Curso
- Por favor, mantenga su celular apagado

■ Presentación del Profesor y de los Asistentes

- Nombre y Formación
- Áreas de trabajo en la empresa
- Experiencia laboral
- Experiencia en ingeniería de software y en pruebas de software

■ Información sobre SQA

- Antecedentes de la empresa y hechos relevantes.
- Experiencia en formación.

0 – Introducción

02 – Tabla de Contenidos

El presente curso se ha desarrollado de acuerdo con el programa de estudios del años 2005 del Certificado de Tester – Nivel Básico, que consta de siete capítulos:

- **Capítulo 0** Introducción.
- **Capítulo I** Fundamentos de Pruebas.
- **Capítulo II** Pruebas a través del Ciclo de Vida del Software.
- **Capítulo III** Técnicas Estadísticas.
- **Capítulo IV** Técnicas de Diseño de Pruebas.
- **Capítulo V** Gestión de Pruebas.
- **Capítulo VI** Herramientas de Pruebas.

0 – Introducción

03 – Organizaciones Internacionales

Programa de Capacitación del ISTQB

- En 1998, se desarrolla en Gran Bretaña un programa de capacitación de múltiples niveles,
- Los fundamentos del proceso de pruebas de software son formulados en el programa de estudios para el nivel básico (“Syllabus for Foundation Level”), edición actual: Octubre de 2005.
- Desde 2004 también se cuenta con certificaciones para el Nivel Avanzado (“Advanced Level”):
 - {Jefe de Pruebas (“Test Manager”), Tester Técnico (“Technical Tester”), Tester Funcional (“Functional Tester”)}
- Se encuentra preparación el Nivel Experto.
- Los Consejos/Comités de Pruebas (“Testing Boards”) (locales) de cada país conforman la estructura de la organización del “International Software Testing Qualification Board” (www.istqb.org)
- Por ejemplo en España: Spanish Software Testing Board (SSTQB), en Alemania: The German Testing Board (GTB)

* ISTQB = International Software Testing Qualification Board

0 – Introducción

04 – Programa de Estudios y Evaluación

El conjunto de diapositivas está basado en el Programa de Estudios de Tester Certificado – Nivel Básico del ISTQB, versión 2005 (Octubre)

- A continuación del curso de formación tendrá lugar un examen al que se puede asistir para obtener el certificado de **Tester Certificado, Nivel Básico** (“**Certified Tester, Foundation Level**”)
- La evaluación es realizada por un examinador perteneciente a una organización independiente (por ejemplo ISQI* o GASQ**)
- Los temas de la evaluación están extraídos de las secciones correspondientes a las dadas en el curso de formación. El examen es de tipo selección múltiple, con una duración de 60 minutos.
- Cada pregunta tiene una (de cuatro) única respuesta correcta. Tienen que ser respondidas un total de 40 preguntas, de las cuales 24 (60%) deben ser respondidas en forma correcta con el objeto de aprobar el examen.

*ISQI = International Software Quality Institute

**GASQ = Global Association for Software Quality

0 – Introducción

05 – Objetivos

■ **Objetivos principales para la formación de Tester Certificado.**

- Aprender las técnicas básicas para planificar las pruebas.
- Aplicar las técnicas de prueba de software en proyectos.
- Seleccionar las técnicas y objetivos de pruebas apropiados.
- Aprender una terminología común.

■ **Audiencia**

- El curso está dirigido a testers de software, desarrolladores y jefes de proyecto en un entorno de producción de software, así como en un entorno de producción industrial que deseen dar a su conocimiento un fundamento de mayor solidez.

■ **Documentos**

- Conjunto de diapositivas propias de la presentación
- Ejercicios y sus soluciones
- Referencias y Glosario.

I – Fundamentos de Pruebas de Software

Agenda

- Capítulo I – Fundamentos de Pruebas de Software
- I/01 Comprendiendo el proceso de pruebas de software.
- I/02 Los Siete Principios Generales del Proceso de Pruebas de Software.
- I/03 Proceso de Pruebas Básico.
- I/04 Psicología en el Proceso de Pruebas

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

■ La Importancia Económica del Software

- El funcionamiento de maquinaria y equipamiento depende en una gran medida al software.
- No es posible imaginar grandes sistemas, en el ámbito de las finanzas ni en el control de tráfico, funcionando sin software.

■ Calidad del Software

- Cada vez más, la calidad del software se ha convertido en un factor determinante del éxito técnico o comercial de sistemas y productos.

■ Pruebas para la Mejora de la Calidad

- Las pruebas y revisiones aseguran la mejora de la calidad de productos de software así como de la calidad del proceso de desarrollo en sí.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Ejemplo de fallo 1: Lanzamiento del Ariane 5

Vuelo 501, tuvo lugar el 4 de junio de 1996, fue la primera prueba de vuelo del sistema desechable de lanzamiento del Ariane 5.

No fue un éxito, la lanzadera se destruyó 37 segundos después del lanzamiento debido a un mal funcionamiento en el software de control, haciendo del mismo defecto de software uno de los más costosos de la historia.

El software del Ariane 5 reutilizó las especificaciones del Ariane 4, pero la trayectoria de vuelo del Ariane 5 era considerablemente distinta y superaba el rango para el cual el código reutilizado había sido diseñado. En particular, la mayor aceleración del Ariane 5 provocó un fallo en los ordenadores de respaldo (“back up”) y navegación inercial primarios, tras lo cual toberas de la lanzadera fueron dirigidas por datos ilegítimos. Las pruebas previas al vuelo nunca fueron ejecutadas sobre el código reajustado bajo condiciones de vuelo simuladas del Ariane 5, por lo tanto el error no fue descubierto antes del lanzamiento.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Ejemplo de fallo 2: Rayos X Letales

Un gran número de pacientes recibieron una dosis letal de rayos gamma debido a un fallo de software.

El **Therac-285** era una máquina para radiación terapéutica producida por la empresa Atomic Energy of Canada Limited.

Estuvo involucrada con, al menos, seis accidentes conocidos entre 1985 y 1987, en los cuales los pacientes fueron objeto de una sobredosis masiva de radiación, que en algunos casos fueron del orden de centenas de “gray”. Al menos cinco pacientes murieron por la sobredosis. Estos accidentes destacan los riesgos del control de software de sistemas críticos en términos de seguridad (“Safety Critical Systems”).

Fuente: Wikipedia.com

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

■ Causas de los Fallos (“Failures”) de Software

■ Error Humano

- Un defecto es introducido en el código de software, en los datos o en los parámetros de configuración.
 - Causas del error humano
 - Plazos, demandas excesivas debidas a la complejidad, distracciones.

■ Condiciones Ambientales

- Cambios en las condiciones ambientales.
 - Causas de condiciones ambientales negativas/adversas.
 - Radiación, magnetismo, campos electromagnéticos, polución, manchas solares, fallo de disco duros, fluctuaciones en el suministro de energía solar.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Error (“Error”), Defecto (“Defect”), Fallo (“Failure”)

- **Error (“Error”) (IEEE 610):**
 - Acción humana que produce un resultado incorrecto, por ejemplo un error de programación.
- **Defecto (“Defect”)**
 - Desperfecto en un componente o sistema que pueda ser la causa por la cual el sistema o componente no logre llevar a cabo su función específica, por ejemplo; sentencia o definición de datos incorrectas.
- **Fallo (“Failure”)**
 - Manifestación física o funcional de un defecto. Si un defecto es encontrado durante la ejecución de una aplicación puede producir un fallo.
 - Desviación de un componente o sistema respecto de la prestación, el servicio o resultado esperados. (Fenton).

Un error introduce un defecto, un defecto causa un fallo

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Proceso de Pruebas durante el proceso de desarrollo, mantenimiento y Operaciones

- **Mejora de la Calidad de un Producto de Software:**

- El proceso de pruebas ayuda a proveer al software de los atributos deseados, i.e. retirar defectos que conducen a fallos.

- **Reducción del Riesgo de Detectar Errores:**

- Actividades de Pruebas de Software adecuadas reducirán el riesgo de encontrar errores durante la fase de operaciones de software.

- **Satisfacer Compromisos**

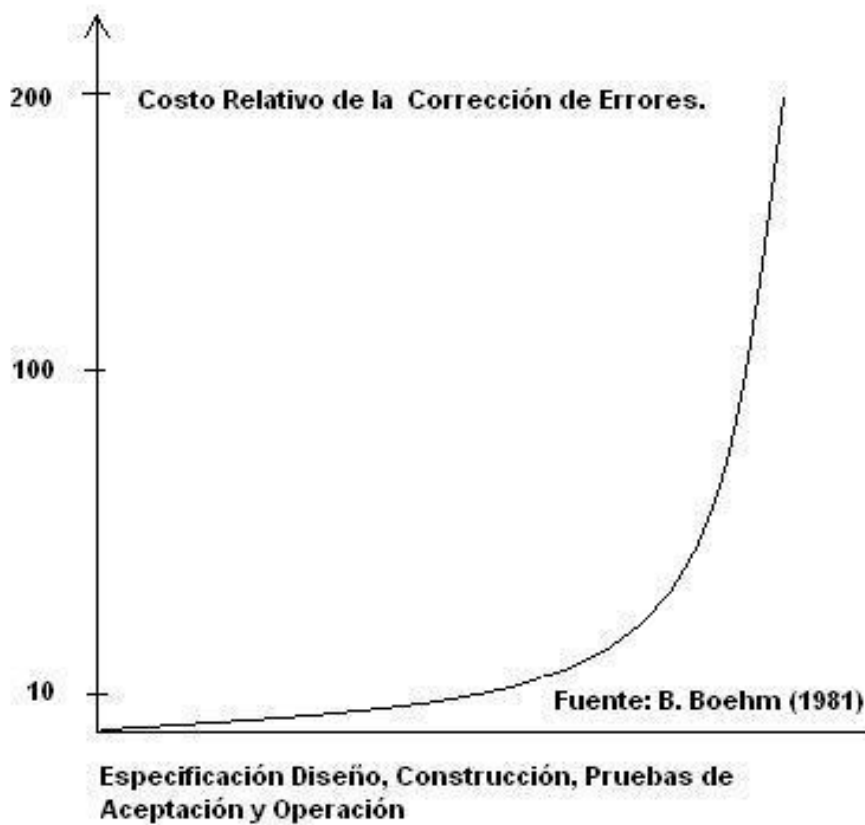
- La ejecución de pruebas puede ser un requisito obligatorio por parte del cliente o debido a normas legales así como el cumplimiento de estándares propios de la industria específica.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

El costo de los defectos

- El costo de eliminar defectos se incrementa con el tiempo durante el cual el defecto permanece en el sistema.
- La detección de errores en etapas tempranas permite la corrección de los mismos a costos reducidos.



I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Calidad en el Software

- **Definición: Calidad en el Software (según IEEE610)**
 - Programas de computadora, procesamientos y posiblemente documentación asociada así como los datos relacionados con la operación de un sistema informático.

- **Definición: Calidad en el Software (según ISO/IEC 9126)**
 - La totalidad de las funcionalidad y características de un producto de software que contribuyen a su habilidad de satisfacer necesidades especificadas o implícitas.

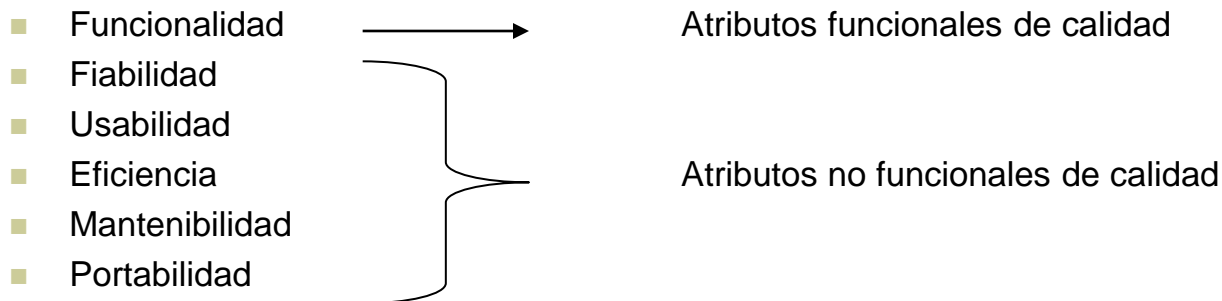
- **Definición: Calidad en el Software (según IEEE Std 610)**
 - Grado en el cual un componente, sistema o proceso satisface los requisitos especificados y/o necesidades del usuario/cliente y sus expectativas.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Calidad en el Software

■ **De acuerdo a la norma ISO/IEC 9126 la calidad de software está constituida por:**



■ **Tipos de Aseguramiento de Calidad (QA):**

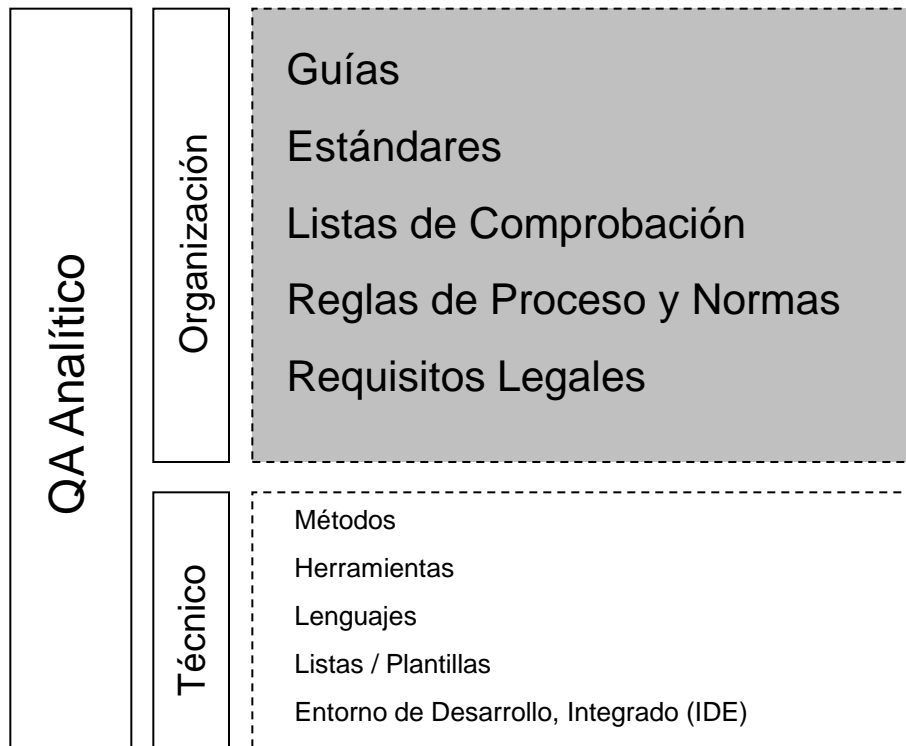
- **Actividades constructivas** con el objeto de prevenir defectos, por ejemplo a través de la aplicación de métodos apropiados de ingeniería de software.
- **Actividades analíticas** con el objeto de detectar defectos, por ejemplo a través de pruebas conducentes a la corrección de defectos y prevención de fallos, incrementando así la calidad del software.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Aseguramiento de la Calidad Constructivo

■ Proceso de Calidad – Gestión de la Calidad



Consigna

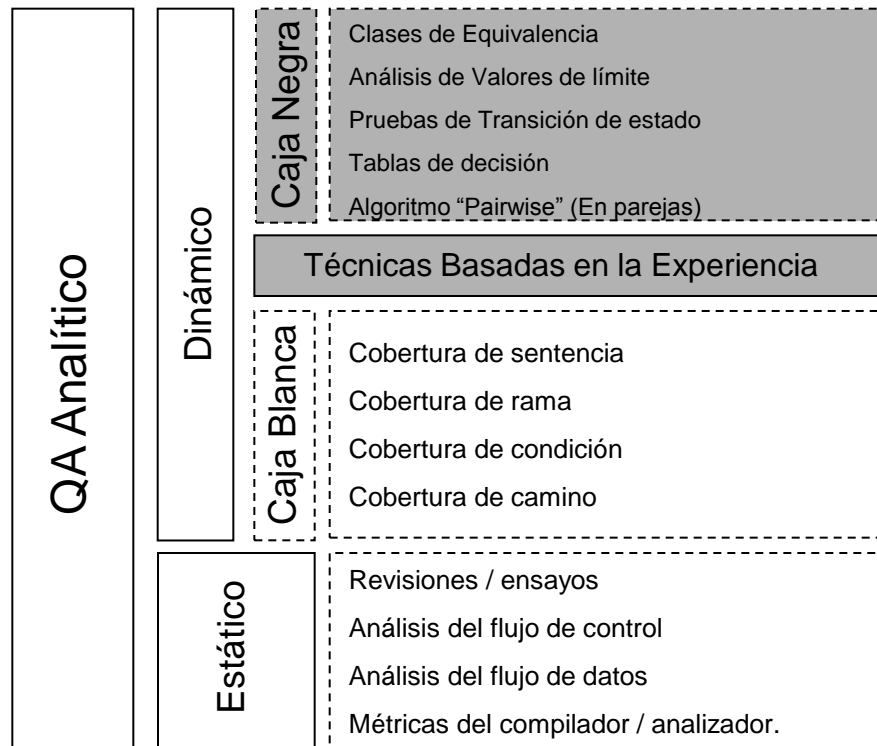
- Los defectos evitados no requieren ser reparados.
- Los errores cometidos en el pasado no deben ser repetidos.
- Prevenir defectos.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Aseguramiento de la Calidad Constructivo

■ Calidad del Producto – Procedimientos de Verificación y Pruebas



Consigna:

- Los defectos deben ser detectados tan temprano en el proceso como sea posible.

•Pruebas Estáticas

Examen sin la ejecución del programa

•Pruebas Dinámicas

Incluye la ejecución del programa.

I – Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Calidad Software – Atributos funcionales de Calidad

■ Funcionalidad significa:

1. Correctitud: La funcionalidad satisface los atributos/capacidades requeridos.
2. Completitud: La funcionalidad satisface todos los requisitos.

■ Funcionalidad incluye (ISO/IEC 9126):

1. Adecuación
2. Precisión
3. Conformidad
4. Interoperabilidad
5. Seguridad

I– Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Calidad Software – Atributos no funcionales de Calidad

- **Fiabilidad:**
 - Madurez, tolerancia a defectos, recuperación tras fallos.
 - Características: Bajo ciertas condiciones el software mantiene su capacidad por un tiempo determinado.
 - $\text{Fiabilidad} = \text{Calidad} / \text{tiempo}$

- **Usabilidad:**
 - Facilidad para ser utilizado, aprendido, comprendido. Atractivo
 - Características: intuitivo, fácil de usar y de aprender.

I– Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Calidad Software – Atributos no funcionales de Calidad

- **Eficiencia:**
 - Funcionalidad y respuesta temporal.
 - Características: el sistema requiere la utilización de un mínimo de recursos para ejecutar una tarea determinada.

- **Mantenibilidad:**
 - Verificabilidad, estabilidad, permite ser analizado.
 - Características: Medida del esfuerzo requerido para realizar cambios en un sistema.

- **Portabilidad:**
 - Capacidad del software de ser transferido a un nuevo entorno
 - Características: Fácil de instalar y desinstalar. Parametrizable.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Atributos de la calidad

- Algunos atributos se influyen entre sí. Debido a esta interdependencia y dependiendo del objeto de la prueba, los atributos deben ser priorizados.
- Para medir cada tipo de atributo se requiere diferentes tipos de pruebas.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Objetivos de las pruebas

- Adquirir conocimiento sobre los defectos en un objeto de prueba
- Comprobar la funcionalidad
- Generar información de riesgos
- Generar confianza

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Cuántas pruebas son suficientes?

Criterio para finalización de las pruebas

- Pruebas basadas en riesgo: responsabilidad y probabilidad de fallos
- Pruebas basadas en plazos y presupuestos: disponibilidad de recursos.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Caso de prueba (“test case”), base de una prueba (“test bases”)

- **Caso de prueba:**

- Precondiciones.
- Conjunto de valores de entrada.
- Conjunto de resultados esperados.
- Forma en la cual se debe ejecutar el caso de prueba y verificar los resultados.
- Poscondiciones esperadas.

- **Base de la prueba:**

- Conjunto de documentos que definen los requisitos de un componente o sistema. Utilizado como el fundamento para el desarrollo de casos de prueba.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

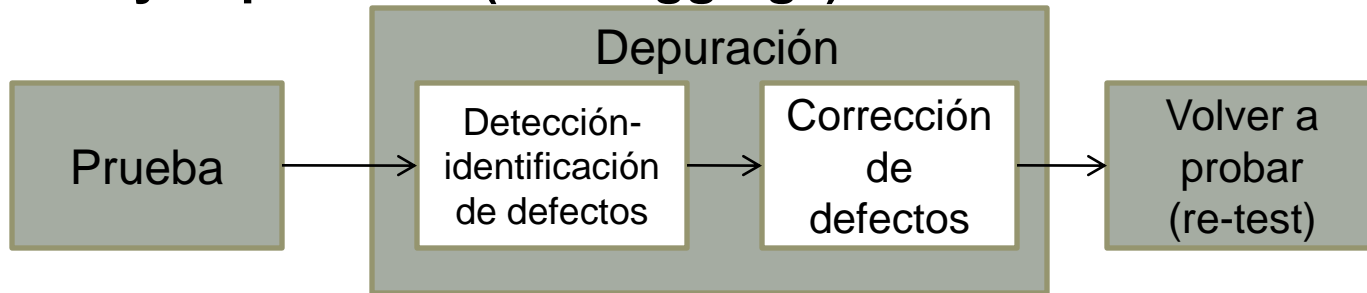
Desarrollo software y revisiones

- **Código (“code”), código fuente (“source code”):**
 - Programa de computadora escrito en un lenguaje de programación que puede ser leído por una persona.
- **Depuración (“debugging”):**
 - Localización y corrección de defectos en el código fuente.
- **Desarrollo software (“software development”):**
 - Es un proceso cuyo objetivo es desarrollar un sistema basado en un computador, normalmente sigue un modelo de desarrollo software.
- **Requisito (“requirement”):**
 - Un requisito describe un atributo funcional o no funcional deseado o considerado obligatorio.
- **Revisión (“review”):**
 - Evaluación de un proyecto con el fin de detectar discrepancias con respecto a los resultados esperados y para recomendar mejoras.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Pruebas y depuración (“debugging”)



- **Probar y volver a probar (“re-test”) son actividades propias del proceso de prueba.**
 - Las pruebas muestran los fallos.
 - Volver a probar verifica que el defecto ha sido corregido.
- **La depuración y la corrección de defectos son actividades propias del desarrollo.**
 - A través de la depuración los desarrolladores pueden reproducir los fallos, analizar el estado del programa y detectar el defecto correspondiente con el objetivo de corregirlo.

I - Fundamentos de Pruebas de Software

01 – Comprendiendo el Proceso de Pruebas del Software

Resumen:

- **Los fallos de software** pueden causar importantes perjuicios.
- **La calidad de software** es la suma de los atributos que se refieren a la capacidad del software de satisfacer un conjunto de requisitos dados.
- El aseguramiento de la calidad **constructivo** se ocupa de la prevención de defectos.
- El aseguramiento de la calidad **analítico** se ocupa de detectar y corregir defectos.
- **Los atributos de la calidad** funcionales y no funcionales definen la calidad total del sistema.
- Cada prueba debe contar con un **criterio para la finalización de pruebas**. Al alcanzar el criterio de finalización de pruebas finalizan las actividades del proceso de pruebas.
- Los testers buscan fallos en el sistema e informan sobre los mismos (proceso de pruebas). Los desarrolladores buscan defectos y los corrigen (depuración).

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 1: El proceso de pruebas demuestra la presencia de defectos.

- **El proceso de pruebas puede probar la presencia de defectos.**
- Las desviaciones identificadas a lo largo del proceso de pruebas demuestran la presencia de un fallo.
- La causa de un fallo puede no ser obvia.

- **El proceso de pruebas no puede demostrar la ausencia de defectos.**
- Las pruebas reducen la probabilidad de la presencia de defectos que permanecen sin ser detectados. La ausencia de fallos no demuestran la corrección de un producto software.
- El mismo proceso de pruebas puede contener errores.
- Las condiciones de las pruebas pueden ser inapropiadas para detectar errores.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 2: No es posible realizar pruebas exhaustivas

- **Pruebas exhaustivas (“exhaustive testing”)**
 - Enfoque del proceso de pruebas en el cual el juego de pruebas abarca todas las combinaciones de valores de entrada y precondiciones.
- **Explosión de casos de prueba (“test case explosion”)**
 - Define el incremento exponencial de esfuerzo costo en el caso de pruebas exhaustivas.
- **Prueba de muestra (“sample test”)**
 - La prueba incluye solamente a un subconjunto de todos los posibles valores de entrada.
 - En condiciones normales, se utilizan generalmente pruebas de muestra. Probar todas las combinaciones posibles de entradas y precondiciones solo es económicamente viable en casos triviales.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 3: pruebas tempranas (“early testing”)

- La **corrección de un defecto es menos costosa en la medida en la cual su detección se realiza en fases mas tempranas del proceso de software.**
- Se obtiene una máxima rentabilidad cuando los errores son corregidos **antes de la implementación.**
- Los **conceptos y especificaciones** pueden ser probados.
- Los defectos detectados en la **fase de concepción** son corregidos con menor **esfuerzo y costos.**
- La **preparación de una prueba también consume tiempo.**
- El **proceso de pruebas** implica mas que solo la **ejecución de pruebas.**
- Las actividades de pruebas pueden ser preparadas antes de que el desarrollo se haya completado.
- Las **actividades de pruebas** deben ser ejecutadas en **paralelo** a la especificación y diseño software.



I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 4: Agrupamiento de defectos (defect clustering)

- **¡Encuentre un defecto y encontrara mas defectos “cerca”!**
- Los defectos aparecen agrupados como hongos o cucarachas.
- Cuando se detecta un defecto es conveniente investigar el mismo modulo en el que ha sido detectado.
- **Los testers deben ser flexibles**
- Habiendo sido detectado un defecto es conveniente volver a considerar el rumbo de las pruebas siguientes.
- La identificación de un defecto puede ser investigada con un mayor grado de detalle, por ejemplo, realizando pruebas adicionales o modificando pruebas existentes.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 5: Paradoja del pesticida

- **Repetir pruebas en las mismas condiciones no es efectivo**
- Cada caso de prueba debe contar con una **combinación única** de parámetros de entrada para un objeto de pruebas particular, de lo contrario no se podrá obtener información adicional.
- Si se ejecutan las mismas pruebas de forma reiterada no se podrán encontrar nuevos defectos.
- **Las pruebas deben ser revisadas/modificadas regularmente para los distintos módulos (código).**
- Es necesario repetir una prueba tras una modificación del código (corrección de defectos, nueva funcionalidad).
- La automatización de pruebas puede resultar conveniente si un conjunto de casos de prueba se debe ejecutar frecuentemente.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 6: Las pruebas dependen del contexto

- **Las pruebas se desarrollan de forma diferente en diferentes contextos.**
- **Objetos de prueba diferentes son probados de forma diferente.**
- El controlador del motor de un coche requiere pruebas **diferentes** respecto de aquellas para una aplicación de “e-commerce”.
- **Entorno de pruebas (“test bed”) vs. Entorno de producción.**
- Las pruebas tienen lugar en un entorno distinto del entorno de producción. El entorno de pruebas debe ser **similar** al entorno de producción.
- siempre habrá diferencias entre el entorno de pruebas y el entorno de producción. Estas diferencias introducen incertidumbre con respecto a las conclusiones que se pudieran obtener tras las pruebas.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Principio 7: La falacia de la ausencia de errores

- **Un proceso de pruebas adecuado detectara los fallos mas importantes.**
- En la mayoría de los casos el proceso de pruebas no encontraran todos los defectos del sistema (ver principio 2), pero los defectos mas importantes deberían ser detectados.
- **Esto en si no prueba la calidad del sistema software.**
- La funcionalidad del software puede no satisfacer las necesidades y expectativas de los usuarios.
- No se puede introducir la calidad a través de las pruebas, ella tiene que construirse desde el principio.

I - Fundamentos de Pruebas de Software

02 – Principios del proceso de pruebas software

Resumen:

- **Las pruebas** pueden ayudar a detectar defectos en el software, sin embargo las mismas no pueden demostrar la **ausencia de defectos**.
- Salvo en casos triviales las **pruebas exhaustivas** son imposibles, las pruebas de muestra son necesarias.
- Las **pruebas tempranas** ayudan a reducir costos dado que los defectos descubiertos en fases tempranas del proceso software son corregidas con menor esfuerzo.
- Los defectos se presentan **agrupados**,. El encontrar un defecto en una ubicación determinada significa que probablemente se encontrara otro defecto a su alrededor.
- **Repetir** pruebas idénticas no genera nueva información.
- Cada **entorno particular** determina la forma en la cual se ejecutaran/desarrollaran las pruebas.
- Un software libre de errores no implica que sea **adecuado para el uso**.

I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

El proceso de pruebas como proceso dentro del proceso de desarrollo software

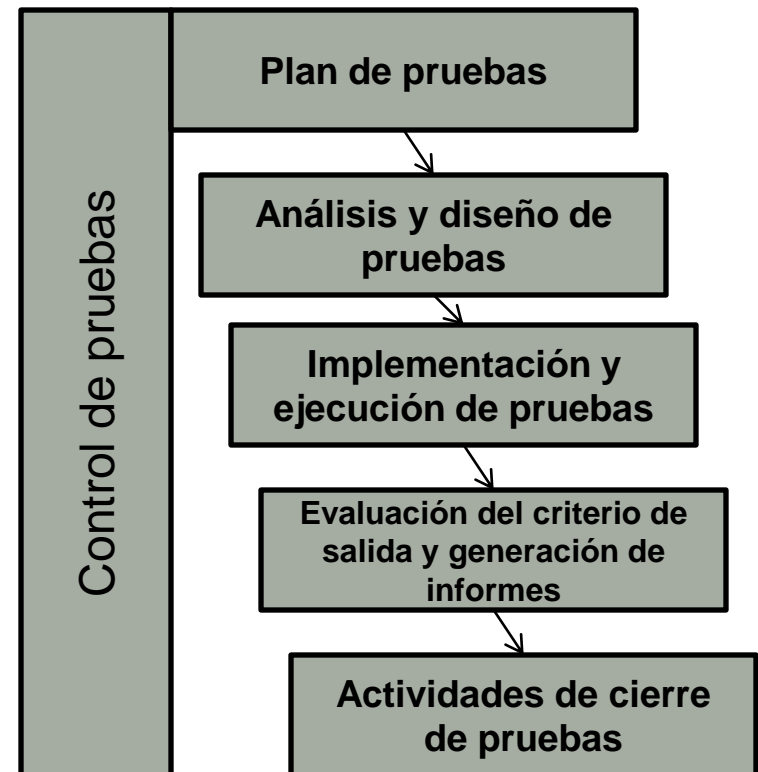
- Dependiendo del enfoque seleccionado, el proceso de pruebas tendrá lugar en diferentes puntos del proceso de desarrollo.
 - Las pruebas constituyen un proceso en si mismas.
 - El proceso de pruebas esta determinado por las siguientes fases.
 - **Planificación de pruebas.**
 - **Análisis y diseño de pruebas.**
 - **Implementación y ejecución de pruebas.**
 - **Evaluación del criterio de finalización de pruebas y generación de informes de pruebas.**
 - **Actividades de cierre de pruebas.**
 - Así también
 - **Control de las pruebas (todas las fases).**
- Las fases del proceso de pruebas se podrán superponer.

I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

El proceso de pruebas a lo largo del proceso de desarrollo software

- Incluye superposición y vuelta atrás (“backtracking”).
- ¡El proceso de pruebas es mas que la ejecución de pruebas!
- Cada fase del proceso de pruebas tiene lugar de forma concurrente con las fases del proceso de desarrollo software.

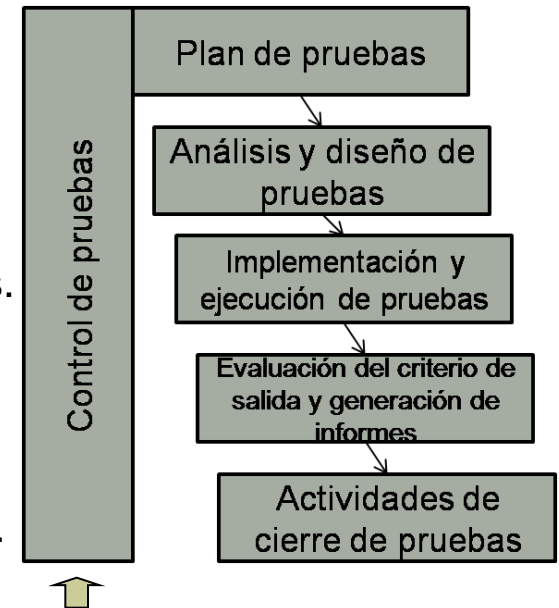


I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Control de pruebas – tareas principales

- El control de pruebas es una actividad continua que **influye** en la planificación de las pruebas. El plan de pruebas puede ser modificado en función de la información adquirida a partir del control de pruebas.
- El estado del proceso de pruebas se determina comparando el **progreso** logrado con respecto al plan de pruebas. Se inician aquellas actividades que se consideraran consecuentemente necesarias.
- Se **miden** y **analizan** resultados.
- El **progreso** de las pruebas, la **cobertura** de las pruebas y el cumplimiento del **criterio de finalización de pruebas** son objeto de seguimiento y son documentados.
- Se inician **medidas correctivas**.
- **Preparar** y **tomar** decisiones.

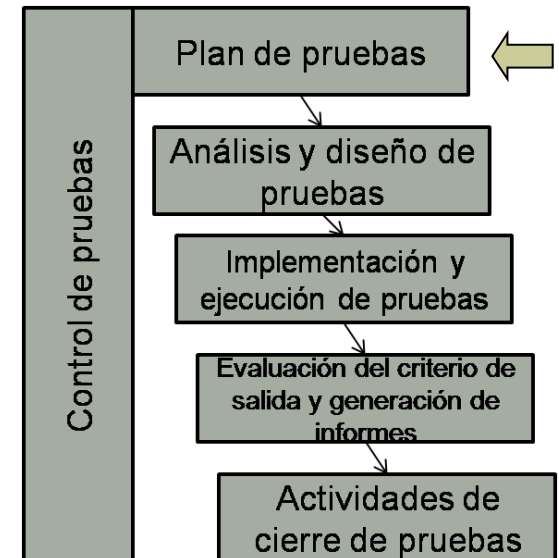


I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Planificación de pruebas – tareas Principales

- Determinar el alcance y los riesgos.
- Identificar los objetivos de las pruebas y el criterio de finalización de pruebas.
- Determinar el enfoque: técnicas de pruebas, cobertura de pruebas, equipo de pruebas.
- Implementar el método/estrategia de pruebas, planificación del tiempo disponible para las actividades a seguir.
- Adquirir/obtener y programar recursos requeridos para las pruebas: personal, entorno de pruebas, presupuesto de pruebas.



I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

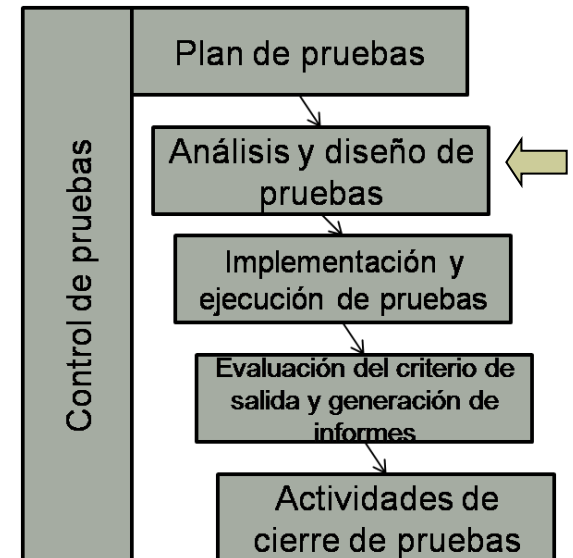
- **Estrategia de pruebas:**
 - (1) Descripción a alto nivel de los niveles de prueba a llevar a cabo y las pruebas asociadas a ellos para una organización o programa (uno o mas proyectos).
 - (2) De acuerdo con el enfoque global, los esfuerzos aplicados a las pruebas se reparten entre los objetos de prueba y los diferentes objetivos de las pruebas: la elección del método de prueba, como y cuando las actividades asociadas a las pruebas deberán ser ejecutadas y cuando se debe detener el proceso de pruebas (criterio de finalización).
- **Criterio de salida** (según Gilb and Graham):
 - Conjunto de condiciones genéricas y específicas, acordadas con los implicados, para permitir que un proceso sea considerado formalmente finalizado. El propósito de los criterios de salida es evitar que una tarea se considere finalizada habiendo partes destacadas de la misma sin completar. Los criterios de salida son utilizados como fuente para elaborar informes y planificar la suspensión de las pruebas.
- **Lo anterior debe ser realizado para cada nivel de pruebas.**

I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Análisis y diseño de pruebas – tareas principales

- Revisión de las bases de las pruebas (requisitos, arquitectura del sistema, diseño, interfaces).
 - Análisis de la arquitectura del sistema, diseño del sistema incluyendo las interfaces entre los objetos de prueba.
- Identificar condiciones de prueba específicas y los **datos de prueba** necesarios.
 - Evaluar la disponibilidad de datos de prueba y/o la viabilidad de generar datos de prueba.
- Diseños de pruebas / casos de prueba.
 - **Crear** casos de prueba lógicos (casos de prueba con datos de prueba sin valores específicos), **y establecer un orden de prioridad** para los mismos.
 - Los **casos de prueba positivos** comprueban la funcionalidad, los **negativos** comprueban situaciones en las cuales se debe realizar un tratamiento de errores.
- Análisis de la testabilidad.

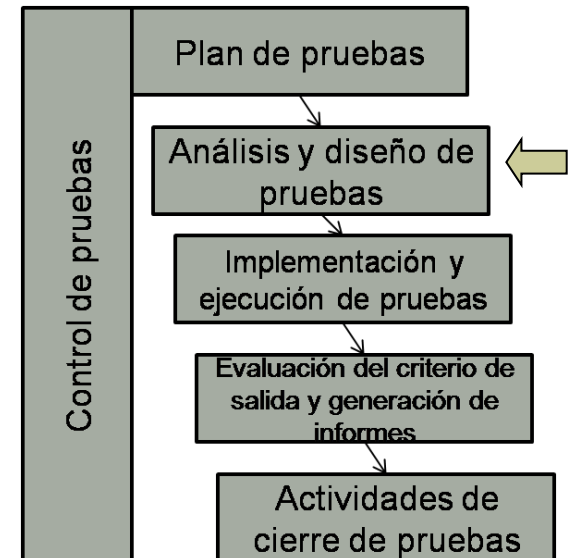


I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Análisis y diseño de pruebas – tareas principales

- Organización del entorno de pruebas (test bed).
 - Disponibilidad (exclusiva), del entorno de pruebas, ventanas de tiempo, etc.
 - Definir el modo de operación del entorno de pruebas, incluida la administración de usuarios.
 - Carga de datos de prueba y parámetros del sistema.
 - Conexión del entorno de pruebas con los sistemas adyacentes.
- Infraestructura de pruebas y herramientas de pruebas (si fuera necesario).
 - Procesos, procedimientos y responsabilidades.
 - Elección, suministro, instalación y operación de herramientas de pruebas.



I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

- **Plan de pruebas (“test plan”):**

- Documento que describe el alcance, enfoque, recursos y planificación (temporal), de las actividades previstas del proceso de pruebas. Este documento incluye, pero no esta limitado a, los objetos de prueba, las características a verificar (a probar), los recurso y un plan de contingencias.

- **Datos de prueba (“test data”):**

- Datos que existen en el sistema antes de que una prueba sea ejecutada, y que afecta o es afectado por el componente o sistema sujeto a pruebas.

- **Datos de entrada (“input data”):**

- Variable que es leída por un componente (tanto almacenada dentro del sistema como fuera del mismo).

- **Cobertura de pruebas (“test coverage”):**

- Grado en el cual un elemento específico ha sido ejecutado por conjunto de pruebas, utilizado con mayor frecuencia en pruebas de caja blanca con el objeto de determinar la cobertura del código.

- **Oráculo de pruebas (“test oracle”):**

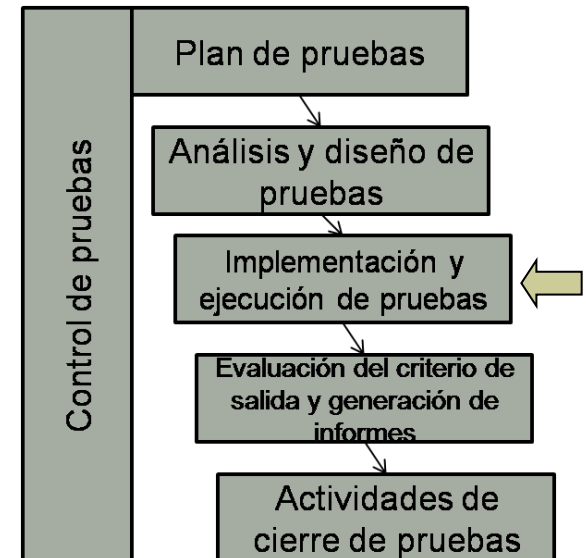
- Fuente que permite determinar los resultados esperados de un software sujeto a pruebas: comparativas (benchmarks)(también resultado de pruebas previas), manuales de usuario o conocimiento especializado. No debe ser el código.

I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Ejecución de pruebas tareas principales

- **Desarrollo y asignación de un orden de prioridad** a los casos de prueba.
 - Creación de datos de prueba, desarrollo de procedimientos de prueba.
 - Creación de secuencias de prueba (juegos de prueba – test suites).
- Creación de **guiones de automatización** de pruebas, si fuera necesario.
- Configuración del **entorno de pruebas** (“test bed”).
- **Ejecución de pruebas** (de forma manual o automática).
 - Seguimiento de secuencias de pruebas establecidas en el plan de pruebas (juegos de pruebas, orden de los casos de pruebas).
- **Registro** y análisis de los resultados de pruebas.
- Reiteración de pruebas (“retest”) / pruebas de regresión (“regression testing”).
 - Tras una corrección: repetición de pruebas con el objeto de asegurar que los defectos han sido **corregidos** y que la corrección no ha **descubierto** otros o **introducido nuevos defectos**.



I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

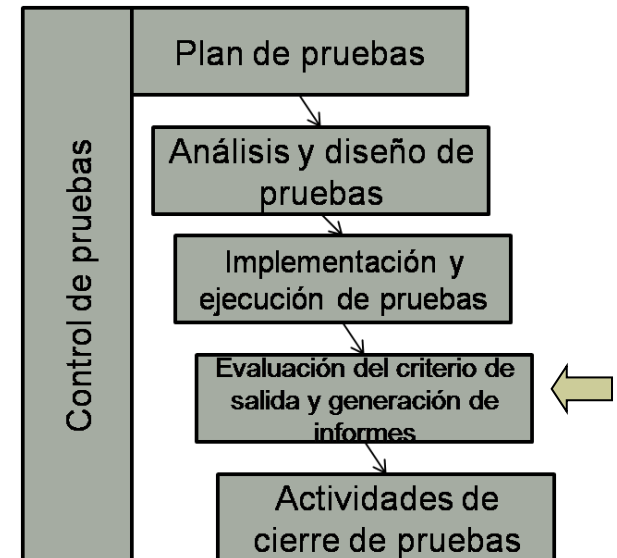
- **Juego de pruebas (test suit) / secuencia de pruebas (“test sequence”):**
 - Conjunto de casos prueba para un componente o sistema, donde la poscondición de un caso de prueba es utilizada como precondition para el caso siguiente.
- **Especificación de procedimiento de pruebas (escenario de pruebas):**
 - Documento en el cual se especifica una secuencia de acciones para la ejecución de una prueba. También es conocido como guión de pruebas o guión de pruebas manuales.
- **Ejecución de pruebas:**
 - Es el proceso de llevar a cabo una prueba aportando/generando los resultados reales.
- **Registro de pruebas (informe de pruebas):**
 - Relación cronológica de los detalles relevantes relativa a la ejecución de pruebas; cuando se desarrollaron las pruebas, que resultados fueron generados
- **Pruebas de regresión:**
 - Pruebas, tras la modificación de un programa previamente probado, con el objeto de asegurar que no se han introducido o descubierto defectos en áreas que no hubieran sido objeto de modificación como resultado de los cambios realizados. Se realizan cuando el software o su entorno a sido modificado.
- **Pruebas de confirmación, reiteración de pruebas (“retest”):**
 - Repetición de una prueba tras la corrección de un defecto con el objeto de confirmar que el defecto ha sido eliminado con éxito.

I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Evaluación del criterio de salida – tareas principales

- Evaluación de la ejecución de pruebas con respecto a objetivos definidos.
- Evaluación de los registros de pruebas (resumen de las actividades de pruebas, resultado de prueba, comunicar criterio de salida).
- Proporcionar información con el objeto de permitir la decisión con respecto a si tendrán lugar pruebas adicionales.

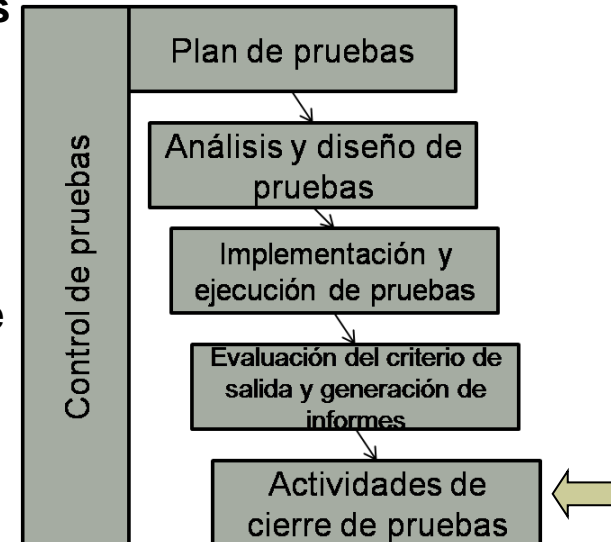


I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Actividades de cierre de pruebas – tareas principales

- Recopilar datos de las actividades del proceso de pruebas finalizadas con el objeto de consolidar la experiencia, utensilios de pruebas, hechos y resultados.
- **Cierre de informes de incidencias** o generación de solicitudes de cambio para cualquier punto que permaneciera abierto.
- Comprobar que entregables planificados han sido entregados y probados.
- Documentar la aceptación del sistema.
- Finalizar y **archivar los utensilios de pruebas**, el entorno de pruebas y la infraestructura de pruebas para un uso posterior, transferencia al entorno de operaciones.
- Analizar las **lecciones aprendidas** para futuros proyectos.



I - Fundamentos de Pruebas de Software

03 – Proceso de pruebas básico.

Resumen:

El proceso de pruebas se puede dividir en diferentes fases.

- **Planificación de pruebas:** abarca las actividades como la definición de la estrategia de pruebas para todas las fases, así como la planificación de los recursos (tiempo, personal, maquinas).
- **Diseño de pruebas (especificación):** abarca el diseño de casos de prueba y sus resultados esperados.
- **Ejecución de pruebas:** abarca la definición de datos de pruebas, la ejecución de pruebas y la comparación de resultados.
- **Informes de pruebas (“logging”):** registro de los resultados de prueba, en forma de tabulada, en base de datos o de forma escrita.
- **Evaluación de pruebas:** abarca el análisis de defectos y la evaluación del criterio de salida.
- **Control de pruebas:** consiste en las actividades de control que abarcan todas las fases anteriores.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Roles y responsabilidades

Rol: Desarrollador

- Implementa requisitos
- Desarrolla estructuras
- Desarrolla el software
- Crea un producto

Rol: Tester

- Planifica actividades de pruebas
- Diseña casos de prueba
- Su única preocupación es encontrar defectos
- encontrar errores producidos por un desarrollador es su éxito

Percepción

¡la actividad del desarrollador es Constructiva!

¡la actividad del tester es destructiva!

¡Error!

¡Las pruebas también constituyen una actividad constructiva, su propósito es la eliminación de defectos en un producto!

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Características personales de un buen tester /1

- Curioso, perceptivo, atento a los detalles.
 - Con el objeto de comprender los escenarios prácticos del cliente.
 - Con el objeto de poder analizar la estructura de una prueba.
 - Con el objeto de descubrir donde se pueden manifestar fallos.

- Escéptico y con actitud crítica.
 - Los objetos de prueba contienen defectos. Usted solo debe encontrarlos.
 - No se debe tener temor al hecho de que pueden ser descubiertos defectos serios que tengan un impacto sobre la evolución del proyecto.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Características personales de un buen tester /2

- Aptitudes para la comunicación.
 - Necesarias para llevar malas noticias a los desarrolladores.
 - Necesarias para vencer estados de frustración.
 - Tanto cuestiones técnicas como prácticas. Relativas al uso del sistema. Deben ser entendidas y comunicadas.
 - Una comunicación positiva puede ayudar a evitar o facilitar situaciones difíciles.
 - Facilitan establecer una relación de trabajo con los desarrolladores a corto plazo.
- Experiencia.
 - Factores personales influyen en la ocurrencia de errores.
 - La experiencia ayuda a identificar donde se pueden acumular errores.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Diferencias: diseñar – desarrollar – probar

- Las pruebas requieren una perspectiva distinta a la del diseño y desarrollo de sistemas de software.
 - Cometido del diseño: ayudar al cliente a proveer/suministrar los requisitos correctos.
 - Cometido de los desarrolladores: convertir los requisitos en funciones.
 - Cometido de los testers: evaluar la correcta implementación de los requisitos del cliente.
 - Objetivo común: proveer un buen producto software.
- En principio, una persona puede asumir los tres roles en su trabajo.
 - Se deben tener en cuenta las diferencias en objetivos y modelos de conducta.
 - Es difícil pero posible.
 - A veces otras situaciones (pruebas independientes), pueden ser mas sencillas y aportar mejores resultados.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Pruebas independientes

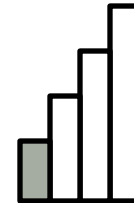
- La separación de las responsabilidades en el proceso de pruebas apoya/promueve la evaluación independiente de los resultados de las pruebas.
- En el siguiente diagrama se representa el grado de independencia a través de un grafico de barras.



I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Organización de pruebas – tipos /1



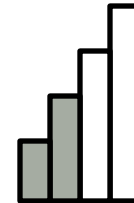
■ Pruebas del desarrollador

- El desarrollador nunca analizara su “creación” de forma imparcial (apego afectivo).
 - Sin embargo el conoce el objeto de pruebas mejor que nadie.
 - Habrá costos adicionales debido a la formación/información de otras personas respecto del objeto de pruebas.
- Las personas tienden a pasar por alto sus propios defectos.
 - Los desarrolladores corren el riesgo de no reconocer defectos evidentes.
- Errores cometidos como consecuencia de una mala interpretación de los requisitos se mantendrán sin ser detectados.
 - Establecer grupos de prueba donde los desarrolladores prueben los productos de otros ayuda a evitar o, al menos, reducir la posibilidad de ocurrencia de este tipo de anomalía (producida por negligencia).

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Organización de pruebas – tipos /2



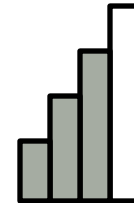
■ Equipos de desarrollo

- Los desarrolladores hablan el mismo lenguaje.
- Los costos de formación/información en lo relativo a objetos de prueba se mantienen a un nivel moderado, especialmente cuando los equipos intercambian objetos de prueba.
- Peligro de generación de conflictos entre equipos de desarrollo.
 - Un desarrollador que busca y encuentra un defecto no será el mejor amigo del autor del objeto de prueba analizado.
- Mezcla de actividades de desarrollo y pruebas.
 - Cambios frecuentes en la forma de pensar.
 - Dificultad en el control del presupuesto del proyecto.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Organización de pruebas – tipos /3



■ Equipos de pruebas

- La creación de equipos de pruebas que den servicio a diferentes áreas del proyecto mejora la calidad de las pruebas.
- Es importante que los equipos de pruebas de diferentes áreas del proyecto trabajen de forma independiente

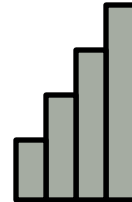
I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Organización de pruebas – tipos /4

■ Subcontratación (externalización) de pruebas

- La separación de las actividades de pruebas y desarrollo aportan la máxima independencia entre los objetos de prueba y el tester.
- Las actividades de pruebas subcontratadas son ejecutadas por personal con conocimiento relativamente pequeño de los objetos de prueba y de los antecedentes del proyecto.
 - La curva de aprendizaje implica altos costos, por lo tanto deberían ser incorporados expertos independientes en etapas tempranas del proyecto.
- Los expertos externos cuentan con un alto nivel de conocimiento (know how), del proceso de pruebas.
 - Esta asegurado un diseño de pruebas apropiado.
 - Se alcanza la optimización en el uso de métodos y herramientas.



■ Diseño de casos de prueba de forma automática.

- Generación de casos de prueba asistida por ordenador, por ejemplo casos de prueba basados en documentos de especificaciones formales.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Pruebas dirigidas por objetivos

- Los testers fijan sus actividades a los efectos de alcanzar los objetivos establecidos para ellos.
 - Objetivo de prueba “**detección de defectos**”: los testers encontraran defectos (y posiblemente encontraran desviaciones que no son defectos).
 - Objetivo de prueba “**verificar la funcionalidad**”: los testers comprobaran la funcionalidad (y posiblemente pasaran por alto algunos defectos).
- Los objetivos de pruebas deberían ser establecidos de forma clara, de lo contrario podrían influir negativamente en el curso del proyecto.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Dificultades /1

- Incapacidad de comprensión mutua.
 - Los desarrolladores deberían contar con un conocimiento básico de pruebas.
 - Los testers deberían contar con un conocimiento básico de desarrollo software.
- Especialmente en situaciones de tensión, la detección de errores cometidos por alguien frecuentemente conduce a conflictos.
 - La forma de documentar los defectos y la forma en la cual el defecto es descrito determinara como se desarrollaran los hechos.
 - Las personas no deberían ser criticadas, los defectos deben ser descritos en términos objetivos.
 - La descripción de los defectos debería ayudar al desarrollador a encontrar el error.
 - Los objetivos comunes siempre deben ser la cuestión principal.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Dificultades /2

- **La comunicación** entre testers y desarrolladores es **insuficiente** o inexistente. Este hecho puede hacer imposible el trabajo conjunto.
 - Los testers son vistos únicamente como “portadores de **malas noticias**”.
 - Mejora: intente ponerse en el rol de la otra persona. ¿ha llegado mi mensaje? ¿la respuesta es suficiente?
- Un proceso de pruebas sólido requiere la **distancia apropiada** con respecto al **objeto de prueba**.
 - Se adquiere un punto de vista independiente e imparcial a través de la distancia con respecto al desarrollo.
 - Sin embargo, una distancia muy grande con respecto al objeto de prueba y el equipo de desarrollo conducirá a mayores esfuerzos y tiempo para las pruebas.

I - Fundamentos de Pruebas de Software

04 – Psicología en el proceso de pruebas

Resumen

- Las personas cometen **errores**, toda implementación tiene defectos.
- La naturaleza humana dificulta la posibilidad de hacer frente a los defectos propios (**ceguera a los errores**).
- Desarrollador y tester implican el encuentro de dos mundos distintos.
 - El **desarrollador constructivo** – algo que no estaba ahí previamente es creado.
 - El **proceso de pruebas** resulta destructivo a primera vista - ¡se detectarán defectos!
 - Juntos, el **desarrollo** y las **pruebas** son constructivas en su objetivo de obtener un producto software con la menor cantidad de defectos posible.
- **Las pruebas independientes aumenta la calidad del proceso de pruebas.**
 - En lugar de equipos de desarrolladores utilice equipos de prueba (testers), o equipos de prueba con testers externos.

II – Pruebas a lo largo del ciclo de vida software

Agenda

Capitulo II – Pruebas a lo largo del ciclo de vida software

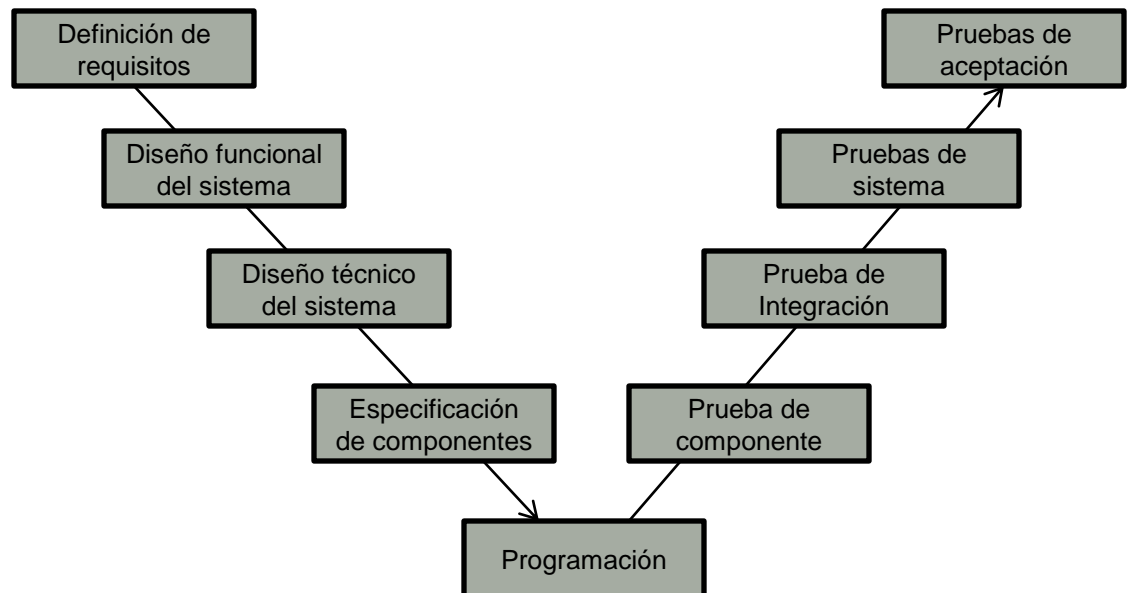
- II/01 Modelos de desarrollo software.
- II/02 Niveles de prueba del modelo – V.
- II/03 Tipos de prueba – objetivos de las pruebas.

II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Pruebas a lo largo del modelo-V general

- El modelo-V general es el modelo de desarrollo software **mas utilizado**.
- Desarrollo y pruebas son **dos ramas iguales**
 - Cada nivel de desarrollo tiene su correspondiente nivel de pruebas.
- Las pruebas (rama derecha), se diseñan en paralelo al desarrollo software (rama izquierda).
- Las actividades del proceso de pruebas tienen lugar a lo largo de todo el ciclo de vida software.



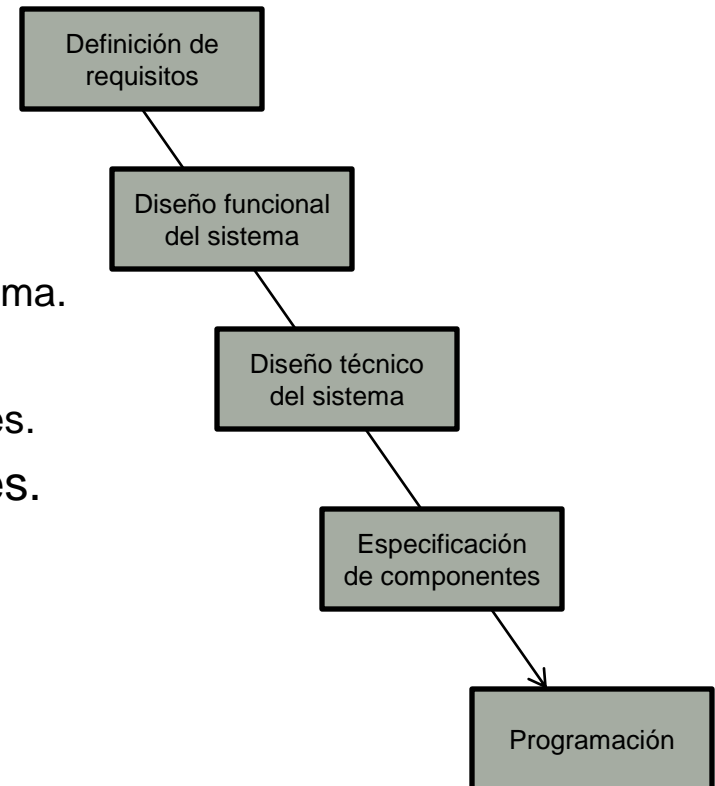
II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Pruebas a lo largo del modelo-V general

■ Rama: desarrollo software.

- Definición de requisitos.
 - Documentos de especificación.
- Diseño funcional del sistema.
 - Diseño del flujo funcional del programa.
- Diseño técnico del sistema.
 - Definición de arquitectura / interfaces.
- Especificación de los componentes.
 - Estructura de los componentes.
- Programación.
 - Creación de código ejecutable



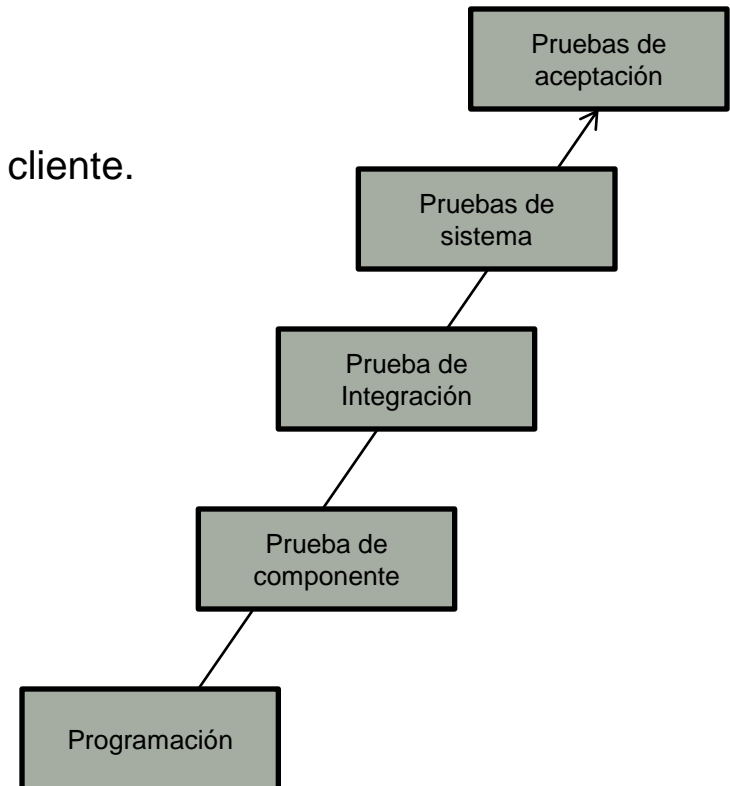
II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Pruebas a lo largo del modelo-V general

■ Rama: pruebas de software.

- Pruebas de aceptación.
 - Pruebas formales de los requisitos del cliente.
- Pruebas de sistema.
 - Sistema integrado, especificaciones.
- Pruebas de integración.
 - Interfaces de componentes.
- Pruebas de componente.
 - Funcionalidad del componente.



II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Verificación vs. Validación

■ Verificación

- Comprobación de la conformidad con los requisitos establecidos (definición según ISO 9000).
- Cuestión clave: ¿se ha procedido correctamente en la construcción del sistema?
¿hemos sumado 1 mas 1 correctamente?

■ Validación

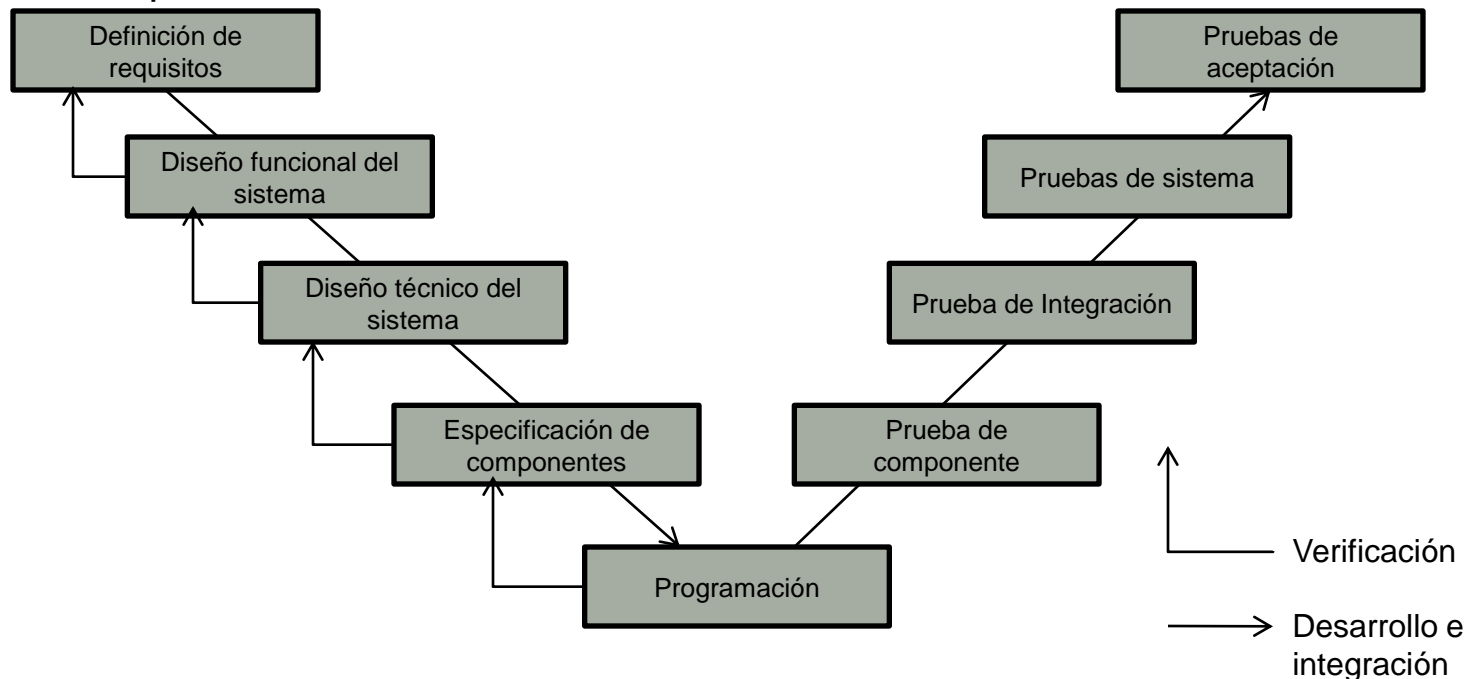
- Comprobación de la idoneidad para el uso esperado (definición según ISO 9000).
- Cuestión clave: ¿Hemos construido el sistema software correcto?
¿El objetivo era sumar 1 mas 1 o deberíamos haber restado?

II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Verificación dentro del modelo-V general

- Cada nivel de desarrollo se verifica respecto de los contenidos del nivel que le precede.
 - Verificar: comprobar la evidencia, substanciar.
- Verificar significa comprobar si los requisitos y definiciones de niveles previos han sido implementados correctamente.

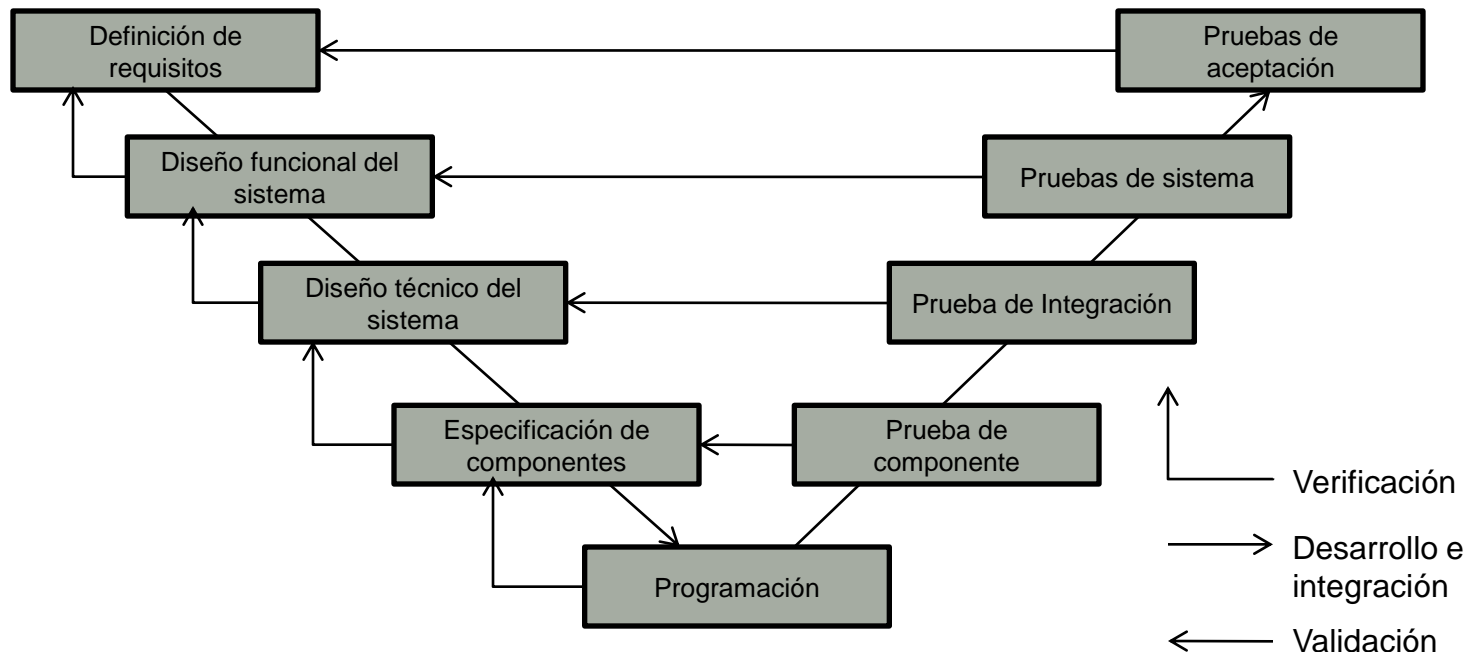


II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Validación dentro del modelo-V general

- La validación se refiere a la corrección de cada nivel de desarrollo.
 - Validar: dar prueba de la aportación de valor.
- Validar significa comprobar lo adecuado de los resultados de un nivel de desarrollo.



II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Otros modelos de prueba: modelos iterativos /1

- Desarrollo software iterativo.
 - Las actividades: definición de requisitos, diseño, desarrollo y pruebas se segmentan en pasos reducidos y se ejecutan de forma continua.
 - Se debe alcanzar el consentimiento con el cliente tras cada iteración con el objeto de modificar el rumbo del proyecto si fuera necesario.
- Ejemplos de modelos iterativos:
 - **Modelo prototipado:** desarrollo rápido de una representación del sistema que pudiera ser objeto de uso, seguida de modificaciones sucesivas hasta que el sistema sea finalizado.
 - **Desarrollo rápido de aplicaciones (“rapid application developmen – RAD”):** la interfaz de usuario se implementa utilizando una funcionalidad hecha (out of the box), simulando la funcionalidad que posteriormente será desarrollada.
 - **Proceso unificado (“rational unified process – RUP”):** modelo orientado a objeto y producto de la compañía rational / IBM. Principalmente aporta el lenguaje de modelo UML y soporte al proceso unificado.
 - **Programación extrema (“extreme programming – XP”):** el desarrollo y las pruebas tienen lugar sin una especificación de requisitos formalizada.

II – Pruebas a lo largo del ciclo de vida software

01 – Modelos de desarrollo software

Otros modelos de prueba: modelos iterativos /2

- Características de los modelos iterativos:
 - Cada iteración contribuye con una **característica adicional** del sistema bajo desarrollo.
 - Cada iteración puede ser **probada por separado**.
 - Las **pruebas de regresión** son de gran relevancia.
 - En cada iteración, la verificación (relación con el nivel precedente), y la validación (grado de corrección del producto dentro del nivel actual), se pueden efectuar por separado.

II – Pruebas a lo largo del ciclo de vida Software

01 – Modelos de Desarrollo de Software

Principios de todos los modelos

- Cada actividad de desarrollo debe ser probada.
 - Ninguna porción del software puede quedar sin ser probada, tanto si ha sido desarrollada “en una única fase” o de forma iterativa.
- Cada nivel de pruebas debería ser probado de forma específica.
 - Cada nivel de pruebas cuenta con sus propios objetivos de prueba.
 - Las pruebas llevadas a cabo en cada nivel deben reflejar estos objetivos.
- El proceso de pruebas comienza con mucha antelación a la ejecución de pruebas.
 - Tan pronto como el desarrollo comienza puede comenzar la preparación de las pruebas correspondientes.
 - También es el caso de las revisiones de documentos comenzando por los conceptos, especificación y el diseño en conjunto.

II – Pruebas a lo largo del ciclo de vida Software

01 – Modelos de Desarrollo de Software

Resumen

- El desarrollo de software utiliza modelos tanto para el propio **desarrollo de software** como para las actividades del proceso de pruebas.
- El modelo más conocido es el **modelo-V**, el cual describe los niveles de desarrollo y niveles de prueba como dos ramas relacionadas.
- Los **modelos iterativos** más relevantes son RUP y XP.
- **Las actividades de pruebas** están recomendadas en todos los niveles de desarrollo.

II – Pruebas a lo largo del ciclo de vida Software

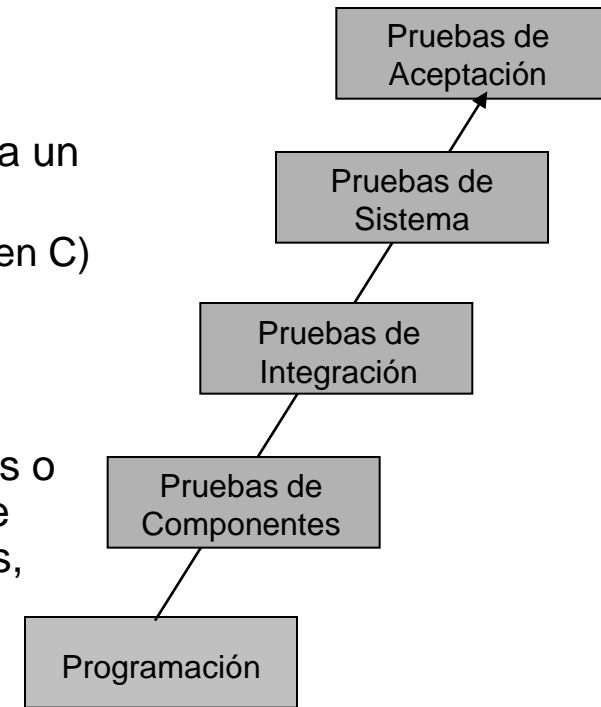
02 – Niveles en el Proceso de Pruebas

Definición

- Pruebas de componente
 - Pruebas de cada componente tras su realización

- Dadas las convenciones de cada lenguaje de programación para la asignación de nombres a sus respectivas componentes, se podrá hacer referencia a un componente como:
 - **Pruebas de módulo (“module test”)** (por ejemplo en C)
 - **Prueba de clase** (por ejemplo en Java o C++)
 - **Prueba de unidad** (por ejemplo en Pascal)

- Los componentes son referidos como módulos, clases o unidades. Dado que los desarrolladores posiblemente pueden estar involucrados en la ejecución de pruebas, éstas también son denominadas **pruebas de desarrollador (developer’s test)**



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Componente: Alcance

- Sólo se prueban componentes individuales.
 - Un componente puede estar constituido por un conjunto de unidades más pequeñas
 - Los objetos de prueba no siempre pueden ser probados en solitario (de forma autónoma).
- Cada componente es probado de forma independiente
 - Describiendo defectos internos.
 - Los efectos cruzados entre componentes quedan fuera del alcance de estas pruebas
- Notas
 - En algunos casos incluso puede ser útil efectuar pruebas de estrés a nivel de componente.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Componente: Motivación

■ Probar la funcionalidad.

- **Toda funcionalidad** debe ser **probada**, por lo menos., con un caso de prueba.
 - Las funciones: ¿Se realizan **correctamente**?
 - La funcionalidad: ¿Cumplen **todas** las especificaciones?
- **Defectos** descubiertos habitualmente:
 - Defectos en el **tratamiento de datos**, normalmente en el entorno de los **valores en las fronteras (“boundary values”)**
 - **Funciones ausentes (“missing functions”)**

■ Probar la robustez (“testing robustness”) (resistencia a datos de entrada inválidos)

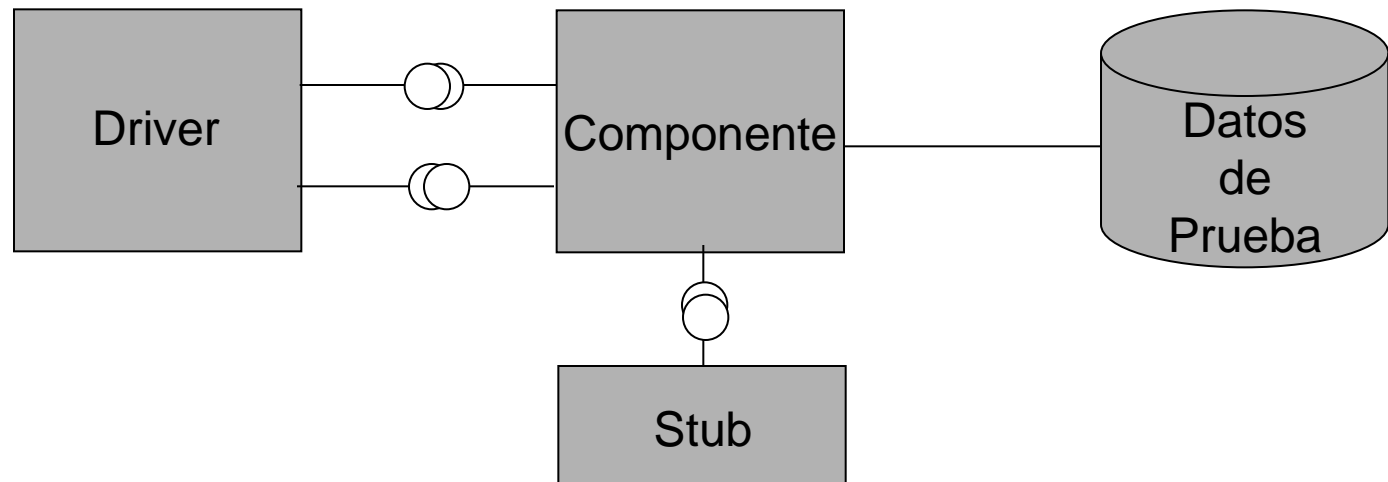
- Los casos de prueba que comprueban datos de entrada inválidos se denominan **pruebas negativas**.
- Un sistema robusto aporta un tratamiento apropiado para **datos de entrada erróneos**.
 - La aceptación por parte del sistema de datos de entrada erróneos puede producir fallos en un futuro procesamiento de los mismos (datos de salida erróneos, fallo del sistema (“system crash”)).

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Componente: Arnés de Pruebas

- La ejecución de pruebas de componente requiere, frecuentemente, de “drivers” y “stubs”.
 - Un “**driver**” procesa la interfaz de un componente.
 - Los drivers simulan datos de entrada, registran datos de salida y aportan un arnés de pruebas (“test harness”)
 - Los drivers utilizan herramientas de programación.
 - Un “**stub**” reemplaza o simula un componente que aún no se encuentra disponible.



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Componente: Métodos

- El código fuente se encuentra disponible para el tester
 - Caso tester = desarrollador
 - Las pruebas se desarrollan con una fuerte orientación hacia el desarrollo.
 - Se podrá aplicar al diseño de casos de prueba el conocimiento de la funcionalidad, estructura de componentes y variables.
 - Las pruebas funcionales serán pertinentes (con frecuencia).
 - Adicionalmente, el uso de depuradores (“debuggers”) y otras herramientas de desarrollo permitirán acceso directo a las variables del programa.
- El conocimiento del código fuente permitirá la aplicación de métodos de caja blanca (“white box”) para pruebas de componente.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Resumen: Pruebas de Componente

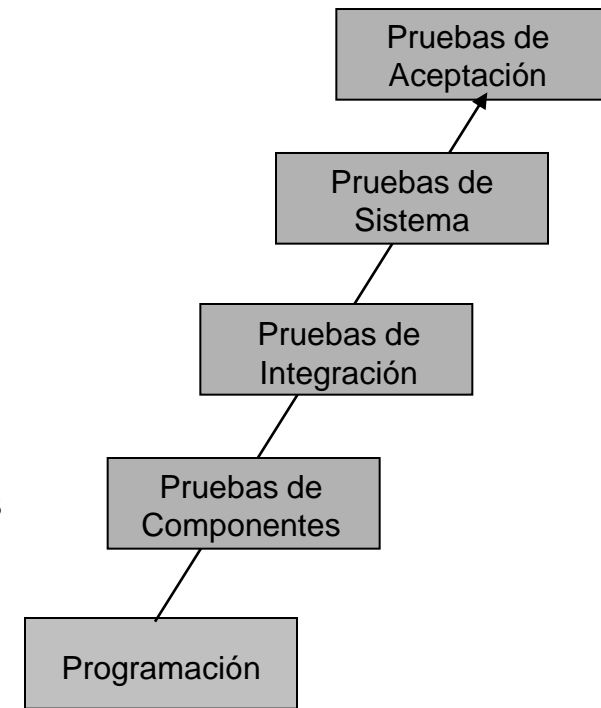
- Un **componente** es la unidad más pequeña especificada de un sistema.
- **Prueba de módulo, de clase, de unidad y de desarrollador** son los utilizados como sinónimos.
- Los “**drivers**” ejecutarán las funciones de un componente y las funciones adyacentes que son reemplazadas por “**stubs**”.
- Las pruebas de componente podrán comprobar características **funcionales** y **no funcionales** de un sistema.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Definición

- Pruebas de Integración (también: Pruebas de Interfaz)
 - Comprueba la **interacción** entre elementos de software (componentes) tras la integración del sistema.
 - La integración es la actividad en la cual se combinan componentes de software individuales en **subsistemas más amplios**.
 - La integración **adicional** de **subsistemas** también es parte del proceso de integración del sistema
 - Cada componente ya ha sido probado en lo referente a su **funcionalidad interna** (prueba de componente). Las pruebas de integración comprueban las **funciones externas**.



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

- Pruebas de Integración: Alcance /1
 - Las pruebas de integración asumen que los componentes **ya han sido probados**.
 - Las pruebas de integración comprueban la **interacción** mutua entre componentes (**subsistemas**) software:
 - Interfaces con otros **componentes**
 - Interfaces **GUIs / MMIs**
 - Las pruebas de integración comprueban las interfaces con el **entorno del sistema**.
 - En la mayoría de los casos la interacción probada es el comportamiento del componente y el **entorno simulado**.
 - En condiciones reales factores adicionales del entorno pueden **influir** en el comportamiento del componente. Por lo tanto las pruebas de integración **no pueden garantizar** un comportamiento correcto en el entorno real del sistema.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

■ Pruebas de Integración: Alcance /2

- Será probado un **(subsistema) sistema** compuesto de componentes individuales.
 - Cada componente tiene una interfaz externo y/o una interfaz que interactúa con otro componente dentro del mismo (subsistema) sistema.
- Son necesario **“drivers” de prueba** (los cuales aportan el entorno al proceso del sistema o subsistema)
 - Con el objeto de tener en cuenta o producir entradas y salidas del (subsistema)
 - Con el objeto de registrar datos.
- Los “drivers” de prueba propios de las pruebas de componente pueden ser **reutilizadas** aquí.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

- Pruebas de Integración: Alcance /3
 - **Herramientas de control (“monitoring tools”)** pueden apoyar las actividades de pruebas registrando datos y controlando las misma pruebas.
 - Un **“stub”** reemplaza un componente faltante.
 - Un “stub” programado reemplazará datos o funcionalidad de un componente que aún no ha sido integrado
 - Un “stub” asumirá las tareas elementales de un componente faltante.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

■ Pruebas de Integración: Enfoque

- El propósito de las pruebas de integración es detectar defectos en las interfaces. Las pruebas de integración comprueban la correcta **interacción entre componentes**.
 - Con el objeto de comprobar, entre otros, aspectos relativos al rendimiento y la seguridad, requiriendo pruebas adicionales (no funcionales).
- Al reemplazar “drivers” y “stubs” de prueba por **componentes reales** se pueden generar nuevos defectos tales como:
 - **Pérdida de datos**, manipulación errónea de datos o entradas erróneas.
 - El componente involucrado **interpreta** los datos de entrada de una manera distinta.
 - Los datos son transferidos en un **momento** incorrecto: antes de tiempo, después de tiempo, a una frecuencia distinta de la requerida.

II – Pruebas a lo largo del ciclo de vida Software

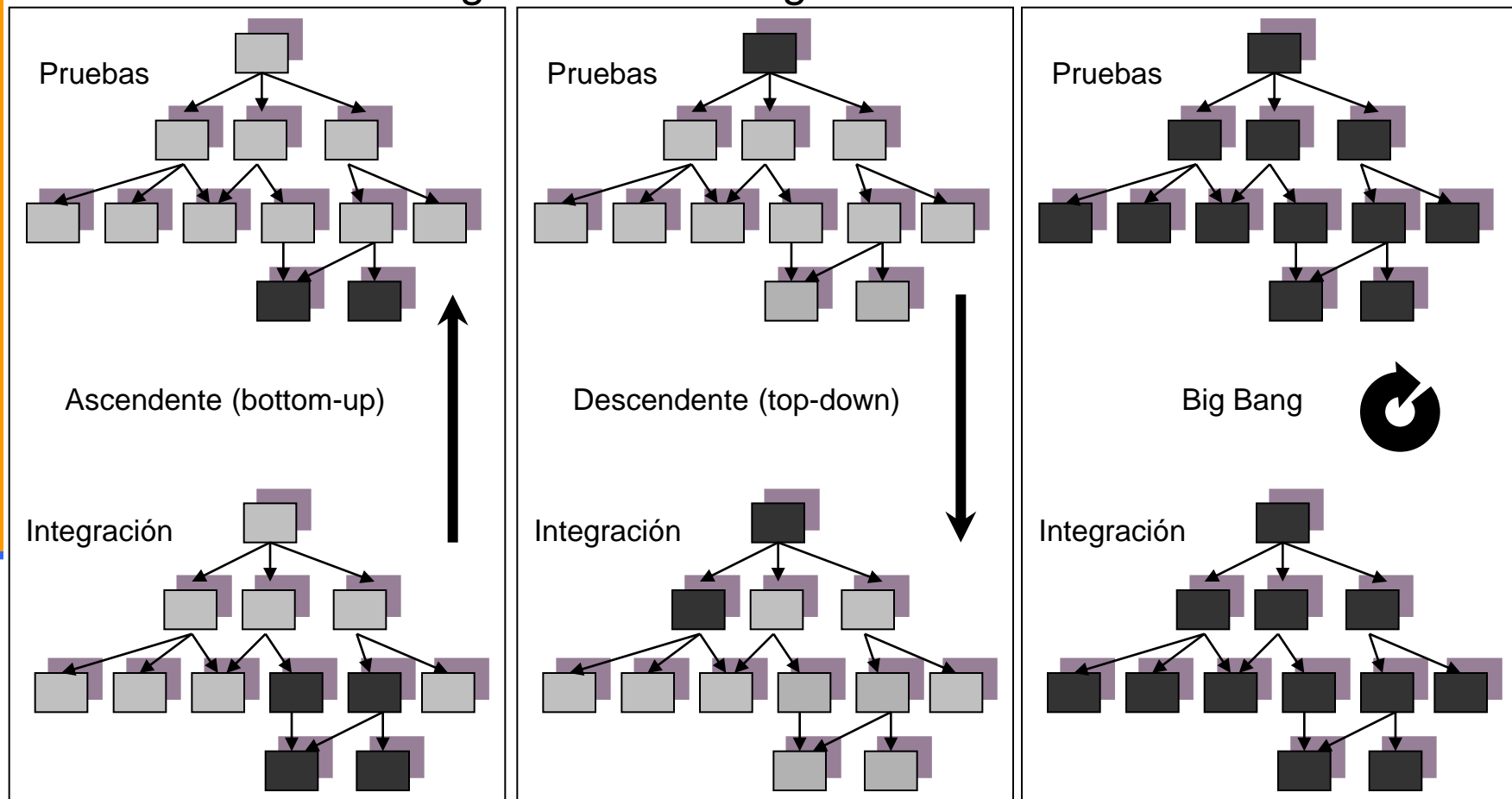
02 – Niveles en el Proceso de Pruebas

- Pruebas de Integración: Estrategias /1
 - Hay diferentes estrategias para las pruebas de integración
 - Enfoque **incremental** es un elemento común a la mayoría de las estrategias (excepción: estrategia “Big Bang”).
 - Las estrategias **ascendente** (“**bottom - up**”) y **descendente** (“**top-down**”) son las utilizadas con mayor frecuencia.
 - La elección de la **estrategia** debe considerar también aquellos aspectos relativos a la eficiencia de las pruebas.
 - La estrategia de integración determina la magnitud del **esfuerzo** requerido para las **pruebas** (por ejemplo el uso de herramientas, programación de “drivers” y “stubs”, etc.).
 - La **finalización de la construcción** de componentes determina, para todos los tipos de estrategias, el intervalo temporal en el cual componente estará disponible. Por lo tanto, la **estrategia de desarrollo** influye en la estrategia de integración.
 - En cada proyecto específico se debe considerar el compromiso entre la reducción de tiempo y la reducción esfuerzo en pruebas:
 - Probar aquello que se encuentra finalizando: mayores costos en pruebas y menor tiempo ocioso.
 - Seguir un plan de pruebas de integración estricto: menores costos y mayor tiempo ocioso.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

■ Pruebas de Integración: Estrategias /2



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

- Pruebas de Integración: Estrategias /3
 - Integración “ad hoc”.
 - Los componentes serán probados, si es posible, inmediatamente después de haber sido finalizada su construcción y se hayan finalizado las pruebas de componente.
 - Características de la integración “ad hoc”.
 - Inicio temprano de las actividades de prueba, dando lugar a un proceso de desarrollo software más corto en términos globales.
 - Uso de la Integración “ad hoc”.
 - Es una estrategia que puede ser aplicada en cualquier etapa del proyecto.
 - Es una estrategia que, normalmente, se aplica combinada con otras.

II – Pruebas a lo largo del ciclo de vida Software

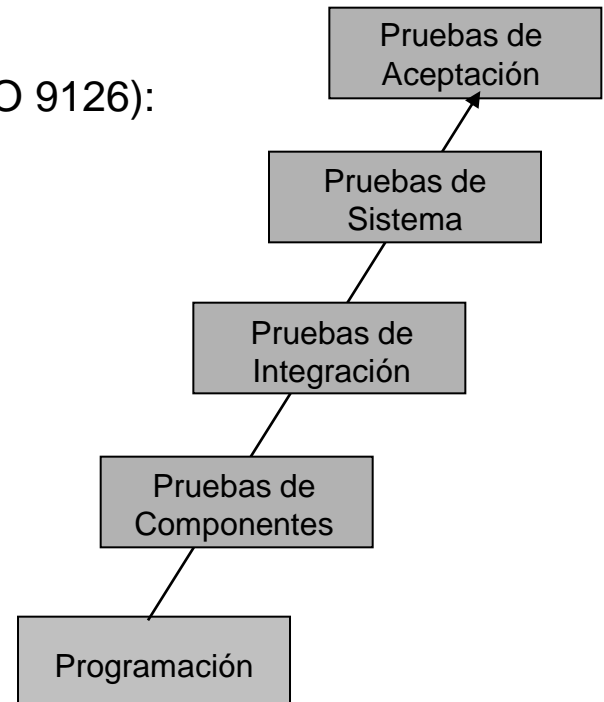
02 – Niveles en el Proceso de Pruebas

- Resumen: Pruebas de Integración
- **Integración** significa construir grupos de componentes.
- Las **pruebas de integración** comprueban la interacción entre componentes respecto de la especificación de interfaces.
- La integración ocurre de forma **ascendente** (“**bottom-up**”), **descendente** (“**top-down**”) o en forma de “**Big Bang**”.
- **Integración de subsistemas** (están constituidos por componentes integrados) también es una forma de integración.
- Cualquier estrategia de integración distinta a las anteriores es integración “**ad hoc**”.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

- Definición
- Pruebas de Sistema
- Pruebas del sistema de software integrado con el objeto de comprobar la conformidad con los requisitos especificados.
 - La calidad de software es observada desde el punto de vista del usuario.
- Las pruebas de sistema se refieren a (según ISO 9126):
 - Requisitos Funcionales.
 - Funcionalidad.
 - Requisitos no funcionales.
 - Fiabilidad.
 - Usabilidad.
 - Eficiencia.
 - Modificabilidad.
 - Portabilidad.



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Sistema: Alcance

- Prueba de un sistema integrado desde el punto de vista del usuario.
 - Implementación completa y correcta de los requisitos.
 - Despliegue en el entorno real del sistema con datos recientes.
- El entorno de pruebas debería coincidir con el entorno real.
 - No son necesarios “stubs” o “drivers”.
 - Todas las interfaces externas son probadas en condiciones reales.
 - Emulación próxima al futuro entorno real del sistema.
- **¡No se realizan pruebas en el entorno real!**
 - Los defectos inducidos podrían dañar el entorno real.
 - Un software objeto de despliegue se encuentra en un estado de cambio constante. Muchas pruebas no serán reproducibles.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Sistema: Requisitos Funcionales/1

- Objetivo: comprobar que la funcionalidad implementada expone las características requeridas.
- Las características a ser probadas incluyen (según ISO 9126):
 - Adecuación/idoneidad (“suitability”)
 - ¿Las funciones implementadas son adecuadas para su uso esperado?
 - Exactitud (“accuracy”)
 - ¿Las funciones presentan los resultados correctos (acordados)?
 - Interoperatibilidad (“interoperability”)
 - ¿Las interacciones con el entorno del sistema presentan algún problema?
 - Conformidad (“compliance”)
 - ¿El sistema cumple con normas y reglamentos aplicables?
 - Seguridad (“security”)
 - ¿Están protegidos los datos/programas contra acceso no deseado o pérdida?

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Sistema: Requisitos Funcionales/2

- Tres enfoques para probar requisitos funcionales:
 - **Pruebas basadas en requisitos.**
 - Los casos de prueba se derivan de la especificación de requisitos.
 - El número de casos de prueba variará en función del tipo/profundidad de las pruebas basadas en la especificación de requisitos.
 - **Pruebas basadas en procesos de negocio**
 - Cada proceso de negocio sirve como fuente para derivar/generar pruebas.
 - El orden de relevancia de los procesos de negocio pueden ser aplicados para asignar prioridades a los casos de prueba.
 - **Pruebas basadas en casos de uso:**
 - Los casos de prueba se derivan/generan a partir de las secuencias de usos esperados o razonables.
 - Las secuencias utilizadas con mayor frecuencia reciben una prioridad más alta.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Sistema: Requisitos No Funcionales

- La **conformidad** con requisitos no funcionales es **difícil de lograr**:
 - Con frecuencia la definición de estos requisitos es **muy vaga** (por ejemplo fácil de operar, interfaz de usuario bien estructurada, etc.).
 - Los requisitos no funcionales **no son establecidos de forma explícita**, son una parte implícita de la descripción del sistema. Sin embargo se espera que sean satisfechos
 - La **cuantificación** de requisitos no funcionales es **difícil** debido al hecho de que se dispone de pocas o ninguna métrica objetiva para su medición: las personas exponen su punto de vista subjetivo (por ejemplo que sea bonito, muy seguro, fácil de aprender, etc.).
- Ejemplo: Prueba / inspección de documentación.
 - ¿Está actualizada la documentación de programas con la versión actual del sistema? ¿La documentación es completa, concisa y fácil de entender?
- Ejemplo: Prueba de mantenibilidad
 - ¿Los programadores han cumplido con los estándares de codificación respectivos?
 - ¿El sistema ha sido diseñado de una forma estructurada y modular?

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Resumen: Pruebas de Sistema

- Las pruebas de sistema se desarrollan utilizando casos de prueba **funcionales** y **no funcionales**.
- Las pruebas de sistema funcionales **confirman** que los requisitos para un uso específico previsto han sido satisfechos (validación).
- Las pruebas de sistema no funcionales verifican los atributos de calidad no funcionales, por ejemplo **usabilidad, eficiencia, portabilidad, etc.**
- Con frecuencia, los atributos de calidad no funcionales son una parte **implícita** de los requisitos, esto hace difícil validarlos.

II – Pruebas a lo largo del ciclo de vida Software

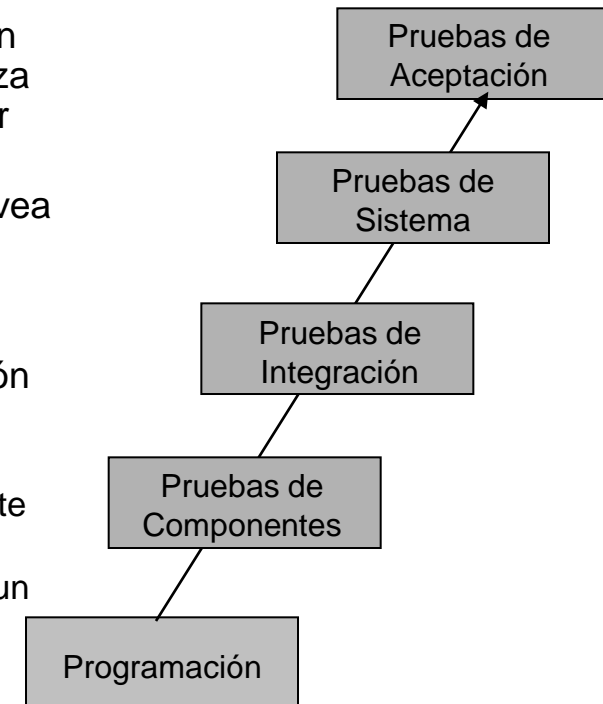
02 – Niveles en el Proceso de Pruebas

Definición

Pruebas de Aceptación.

- Las pruebas de aceptación son pruebas formales llevadas a cabo con el objeto de verificar la conformidad del sistema con los requisitos. El objetivo es aportar justificación a la confianza en el sistema para que pueda ser aceptado por el cliente (ver IEEE610).
- Es habitual que sean las primeras pruebas en las cuales se vea involucrado el cliente (es recomendable involucrar al cliente durante el mismo proceso de desarrollo, por ejemplo dar soporte al desarrollo de prototipos).
- El grado de implicación de los clientes puede variar en función del tipo de programa (software individual o un producto de software comercial (COTS*)).
 - Normalmente, el software individual es probado directamente por el cliente.
 - Un producto de software comercial puede ser probado por un grupo seleccionado (particular) de clientes.

*COTS = Commercial off the shelf



II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Aceptación: Aceptación Contractual

- ¿El software satisface todos los requisitos **contractuales**?
 - Con la aceptación formal se cumplen hitos legales: comienzo de fase de **garantía**, hitos de **abono (pago)**, acuerdos de **mantenimiento**, etc.
 - **Criterios de aceptación verificables** definidos en el momento del acuerdo contractual constituyen una garantía para ambas partes
 - Las pruebas de aceptación deben tener en cuenta normas y reglamentos gubernamentales, legales, industriales y de otro tipo (por ejemplo reglamento de seguridad “FMVSS 2008: Federal Motor Vehicle Safety Standards”).
- Normalmente el cliente selecciona **casos de prueba** para las pruebas de aceptación.
 - Es posible que surjan malas interpretaciones con respecto a los requisitos y pueden ser discutidos, “El cliente conoce su negocio”.
- Las pruebas se realizan utilizando el **entorno del cliente**.
 - El entorno del cliente puede producir nuevo fallos.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Aceptación: Pruebas de aceptación operativas

- Requiere que el software sea adecuado para su uso en un entorno operativo.
 - Integración del software en la infraestructura TI del cliente (copias de seguridad/restauración de sistemas, reinicio, instalación, capacidad de ser desinstalado, recuperación en caso de desastres, etc.).
 - Gestión de usuarios, interacción con ficheros y estructuras de directorios en uso.
 - Compatibilidad con otros sistemas (otros ordenadores, servidores de bases de datos, etc.).
- Con frecuencia, las pruebas de aceptación operativas son realizadas por el administrador de sistemas del cliente.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Pruebas de Aceptación: Pruebas Alpha y Pruebas Beta

- Una versión preliminar y estable del software (**versión beta**) se entrega a un conjunto específico (seleccionado) de clientes.
 - El **cliente** ejecuta las pruebas en **dependencias propias** (pruebas beta, también denominadas pruebas de campo), los problemas son registrados e informados a los desarrolladores para su corrección.
 - El cliente utiliza el software para hacer el tratamientos de sus procesos cotidianos o ejecuta un juego de pruebas seleccionado.
 - El software se prueba mejor en distintos entornos del cliente.
- Previo a las pruebas beta el cliente pueden probar el nuevo software en las dependencias del desarrollo.
 - Las **pruebas alpha** se desarrollan en las dependencias de la organización que desarrolla.
- Ventajas de estas pruebas.
 - Reducen el costo de las pruebas de aceptación, se utilizan distintos entornos de cliente, involucran a un alto número de usuarios.

II – Pruebas a lo largo del ciclo de vida Software

02 – Niveles en el Proceso de Pruebas

Resumen: Pruebas de Aceptación

- Las pruebas de aceptación son las pruebas de sistema por parte del **cliente**.
- La prueba de aceptación es una actividad de carácter **contractual**, se verificará entonces que el software satisface los requisitos del cliente.
- Las pruebas **alpha** y **beta** son pruebas ejecutadas por clientes reales o potenciales, en las dependencias del desarrollador (alpha) o en las dependencias del cliente (beta).

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Tipos de pruebas y niveles de pruebas

- Niveles en el proceso de pruebas.
 - En la sección previa se han explicado los distintos niveles de pruebas, es decir, pruebas de componente, pruebas de integración, etc.
 - ¡En cada nivel de prueba los objetivos de las pruebas tienen un foco diferente!
 - Por lo tanto, se aplican distintos tipos de pruebas durante los distintos niveles de pruebas.
- Tipos de pruebas
 - Pruebas funcionales (Objetivo: probar la función).
 - Pruebas no funcionales (Objetivo: probar las características del producto).
 - Pruebas estructurales (Objetivo: probar la estructura/arquitectura software).
 - Pruebas de confirmación/regresión (Objetivo: probar después de cambios.)

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas Funcionales

■ **Objetivo: la función del objeto de prueba.**

- La funcionalidad puede ser vinculada a los datos de entrada y salida de un objeto de prueba.
- Los métodos de caja negra (“black box”) se utilizan en el diseño de caso de prueba relevantes.
- Las pruebas se desarrollan teniendo en cuenta los requisitos funcionales (establecidos en las especificaciones, conceptos, casos de estudio, reglas de negocio o documentos relacionados).

■ **Ámbito de Aplicación**

- Las pruebas funcionales se pueden llevar a cabo en todos los niveles de prueba.

■ **Ejecución.**

- El objeto de prueba es ejecutado utilizando combinaciones de datos de prueba derivados/generados a partir de los casos de prueba.
- Los resultados de la ejecución de la prueba son comparados con los resultados esperados.

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas No Funcionales

- **Objetivo: la función del objeto de prueba.**
 - ¿De qué forma el software lleva a cabo la función?
 - Las características de calidad no funcionales (ISO 9126: fiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad) a menudo son vagas, incompletas o inexistentes, dificultando las pruebas asociadas a las mismas.
- **Ámbito de Aplicación**
 - Las pruebas no funcionales se pueden llevar a cabo en todos los niveles
 - Pruebas no funcionales típicas:
 - Pruebas de carga (“load testing”) / Pruebas de rendimiento (“Performance Testing”) / Pruebas de volumen (“Volumen Testing”) / Pruebas de Estrés (“Stress Testing”).
 - Pruebas de las características de seguridad para el software (“Testing of Security Features”) / Pruebas de las características de seguridad de software (“Testing of Safety Features”)
 - Pruebas de estabilidad y robustez (“Stability and Robustness Testing”) / Pruebas de Compatibilidad (“Compatibility Testing”).
 - Pruebas de usabilidad (“Usability Testing”) / Pruebas de Configuración (“Configuration Testing”).
- **Ejecución**
 - La conformidad con los requisitos no funcionales se miden utilizando requisitos funcionales (seleccionados).

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas No Funcionales (Pruebas de Sistema)

■ Prueba de Carga (“Load Test”)

- Sistema bajo carga (carga mínima, más usuarios/transacciones).

■ Prueba de Rendimiento (“Performance Test”)

- Rapidez con la cual un sistema ejecuta una determinada función.

■ Prueba de Volumen (“Volume Test”)

- Procesamiento de grandes cantidades de datos / ficheros.

■ Prueba de estrés (“stress test”)

- Reacción a la sobrecarga / recuperación tras el retorno a una carga normal.

■ Prueba de estabilidad (“stability test”)

- Rendimiento en “modo de operación continua”

■ Prueba de Robustez (“test for robustness”)

- Reacción a entradas erróneas o datos no especificados.
- Reacción a fallos hardware / recuperación ante situaciones de desastre.

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas No Funcionales

- **Pruebas de Seguridad para el Software (de datos) (“test for (data) security”)**
 - Protección contra accesos no autorizados.
 - Protección contra el robo y daño de datos.
- **Pruebas de compatibilidad (conversión de datos) (“compatibility test (data conversion)”)**.
 - Cumplimiento de normas y reglamentos (internos / externos).
 - Reacción a distintos entornos (H/W, O/S, etc.)
- Pruebas de usabilidad (“test for usability”)
 - Estructurado, comprensible, fácil de aprender para el usuario.
- **Otros aspectos no funcionales de la calidad:**
 - Portabilidad (“portability”): capacidad de reemplazar (“replaceability”), capacidad de ser instalado (“installability”), conformidad/cumplimiento (“conformance/compliance”), adaptabilidad (“adaptability”).
 - Mantenibilidad (“maintainability”): verificabilidad (“verifiability”), estabilidad (“stability”), analizabilidad (“analyzability”); capacidad de ser modificado (“changeability”).
 - Fiabilidad (“Reliability”): madurez (“maturity”), robustez (“robustness”), capacidad de ser recuperado (“recoverability”).

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas Estructurales

■ **Objetivo: cobertura**

- Análisis de la estructura de un objeto de prueba (enfoque: caja blanca).
- La finalidad de las pruebas es medir el grado en el cual la estructura del objeto de prueba ha sido cubierto por los casos de prueba.

■ **Ámbito de aplicación.**

- Las pruebas estructurales son posibles en todos los niveles de pruebas, se realizan de forma conjunta a las pruebas de componente y de integración mediante el uso de herramientas.
- El diseño de pruebas estructurales se finaliza tras haber sido diseñados las pruebas funcionales, con el propósito de obtener un alto grado de cobertura.

■ **Ejecución.**

- Se probará la estructura interna de un objeto de prueba (por ejemplo el control de flujo en el interior de un componente, el flujo a través de la estructura de un menú).
- Objetivo: todos los elementos estructurales identificados deberán estar cubiertos por casos de prueba.

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas de confirmación/regresión/1

■ **Objetivo: objeto de pruebas después de cambios.**

- Después de que un objeto de pruebas o el entorno de su sistema ha sido objeto de modificación todos los resultados de pruebas resultan inválidos, por lo que las pruebas deben ser repetidas. Repetir una prueba de funcionalidad que ha sido verificada previamente se denomina prueba de **regresión**.
- Es necesario volver a probar (“retest”) zonas adyacentes debido a efectos colaterales no deseados de una funcionalidad extendida o nueva.
- El alcance de las pruebas de regresión depende del riesgo que la nueva funcionalidad implementada (extensión o corrección de errores) impone al sistema.

■ **Ámbito de aplicación.**

- Las pruebas de confirmación/regresión pueden ser realizadas en todos los niveles.
- Las pruebas típicas tras una modificación son las siguientes:
 - Pruebas de confirmación (“retest”) (= pruebas después de la corrección de errores).
 - Pruebas de regresión (= pruebas para descubrir nuevos defectos introducidos en funcionalidad previamente sin fallos).

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Pruebas de confirmación/regresión/2

■ Ejecución

- Básicamente la ejecución tiene lugar de la misma forma en la cual se han ejecutado las pruebas en iteraciones previas.
- En la mayoría de los casos, una prueba de regresión completa no es viable dado sus altos costos y duración.
- Un alto grado de modularidad en el software permite **reducir** las pruebas de regresión.
- Criterios para la selección de casos de prueba de regresión:
 - Casos de prueba de prioridad alta.
 - Probar solamente la funcionalidad estándar, saltarse casos y variaciones especiales
 - Probar solamente la configuración utilizada con mayor frecuencia.
 - Probar solamente subsistemas / zonas seleccionadas del objeto de pruebas.
- Si durante fases tempranas del proyecto resulta evidente que ciertas pruebas son adecuadas para las pruebas de regresión, se deberá considerar la automatización de estas pruebas.

II – Pruebas a lo largo del ciclo de vida Software

03 – Tipos de Pruebas: Objetivos del Proceso de Pruebas

Resumen

- En **niveles de pruebas** distintos se utilizan **tipos de pruebas** distintos.
- Los tipos de pruebas son: **funcionales, no funcionales, estructurales y pruebas relacionadas a cambios.**
- Las **pruebas funcionales** comprueban el **comportamiento** entrada / salida de un objeto de pruebas.
- Las **pruebas no funcionales** comprueban las **características de un producto.**
- Las pruebas no funcionales incluyen, pero no están limitadas a, **pruebas de carga, pruebas de estrés, pruebas de rendimiento, pruebas de robustez.**
- Las **pruebas estructurales** habituales son pruebas que comprueban el flujo de control y datos midiendo la cobertura en el objeto de prueba.
- Pruebas importantes después de un cambio: **pruebas de confirmación (“confirmation tests or retests”)** y **pruebas de regresión (“regression tests”).**

II – Pruebas a lo largo del ciclo de vida Software

04 – Pruebas de Mantenimiento

Pruebas tras aceptación del producto/1

- El cliente ha aprobado el producto y es puesto en producción.
 - El ciclo de desarrollo inicial, incluidas las pruebas asociadas, ha sido completado.
- El mismo software se encuentra al comienzo del ciclo de vida:
 - Será utilizado por muchos años, será ampliado.
 - Es muy probable que aún contenga defectos, por lo tanto será modificado y corregido.
 - Necesitará adaptarse a nuevas condiciones y deberá integrarse a nuevos entornos.
- ¡Cualquier nueva versión del producto, cada nueva actualización y cada nueva actualización y cada cambio del software requiere pruebas adicionales!

II – Pruebas a lo largo del ciclo de vida Software

04 – Pruebas de Mantenimiento

Pruebas tras aceptación del producto/2

- Pruebas de software modificado.
- Pruebas de regresión (“regression testing”)
 - Las pruebas de regresión consisten en la repetición de pruebas después de una modificación (por ejemplo la corrección de un defecto) con el objeto de comprobar que no se han introducido o descubierto defectos enmascarados como resultado de la modificación.
- Nota:
 - Las pruebas de regresión son necesarias y realizadas ya antes de que tenga lugar la aceptación del producto.
 - Las pruebas de regresión se realizan siempre que el software o su entorno haya sido modificados.

II – Pruebas a lo largo del ciclo de vida Software

04 – Pruebas de Mantenimiento

Pruebas tras aceptación del producto/3

- El alcance de las pruebas de regresión puede variar en función de las circunstancias (condiciones de entorno):
 - Prueba de confirmación (“retest”):
 - Repetición de los casos de prueba que han detectado un defecto con el objeto de comprobar que ha sido corregido.
 - Prueba de una funcionalidad modificada:
 - Prueba de todas la partes del software que han sido objeto del cambio.
 - Prueba de nueva funcionalidad:
 - Se prueba solamente el nuevo componente del sistema. El nuevo componente ha sido probado durante las pruebas de componente.
 - Prueba completa de regresión:
 - Se ejecuta/repite una prueba completa del sistema.

III – Técnicas estáticas

00 – Agenda

Capítulo III – Técnicas estáticas de pruebas

- III/01 Revisiones y el proceso de pruebas.
- III/02 Análisis estático basado en herramientas.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Enfoque básico

- Las técnicas estáticas de pruebas comprenden varios métodos que **no ejecutan** el componente o sistema objeto de la prueba.
- Las pruebas estáticas incluyen:
 - **Revisiones (“reviews”)** (actividad manual)
 - **Análisis Estático (“static analysis”)** (actividad basada en herramientas).
- Las técnicas estáticas complementan los métodos dinámicos.
 - Las pruebas estáticas detectan **defectos** en lugar de **fallos**.
 - Los **conceptos** también son analizados, no sólo el código ejecutable.
 - Los defectos / desviaciones son detectados en una fase temprana, antes de que sean implementadas en el código.
- **Documentos de alta calidad** conducen a productos de alta calidad.
 - Incluso si una especificación revisada no contiene ningún defecto, la **interpretación** de la especificación y creación del diseño pueden ser **defectuosas**.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Objetivos de las revisiones

- Las revisiones se realizan con el objeto mejorar la calidad del producto.
 - Las revisiones se utilizan para verificar la correcta transición de una fase a la siguiente, según está definido en el lado izquierdo del modelo -V.
- La detección **temprana** de errores ahorra **costos**.
- En el transcurso de las revisiones se podrían detectar los **siguientes defectos**:
 - Defectos en las especificaciones.
 - Defectos en el diseño y arquitectura del software.
 - Desviaciones con respecto a estándares acordados (por ejemplo guías de programación).
 - Defectos en las especificaciones de interfaces.
 - Mantenibilidad insuficiente.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Ventajas y desventajas de las revisiones /1

■ **Ventajas:**

- Costos más bajos y un potencial de ahorro relativamente alto.
- Defectos en la documentación son detectados y corregidos temprano
- Los documentos de alta calidad mejoran el proceso de desarrollo.
- Mejora el índice de comunicación / intercambio de conocimiento (“know - how”)

■ **Desventajas:**

- Se podrían presentar situaciones de tensión en el caso de enfrentamientos directos con el autor.
- Los expertos involucrados en las revisiones deben adquirir conocimientos específicos del producto, es necesario una buena preparación.
- Inversión considerable de tiempo (del 10% - 15% del presupuesto total).
- Moderador y participantes influyen directamente en la calidad de la revisión

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Fases de una revisión

- Fase de planificación.
 - Organización de la revisión, selección de los miembros de a revisión.
- Preparación de la organización e inicio (“kick – off”).
 - Distribución de los objetivos de la revisión e información adicional.
- Preparación individual.
 - Los evaluadores inspeccionan los objetos, comentan los elementos en caso de necesidad de aclaraciones.
- Reunión de revisión.
 - Reunión de los miembros de la revisión, los evaluadores presentan sus resultados.
- Seguimiento (“follow up”)
 - El resultado de la revisión es distribuido al responsable de la revisión estableciendo
 - El objeto sujeto a pruebas, participantes y roles.
 - Recomendaciones realizadas por los evaluadores.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Roles y responsabilidades

- Director – Responsable – Jefe de Proyectos
 - Inicia la revisión, decide respecto de los participantes y asigna recursos.
- Moderador
 - Difige la reunión / discusión, hace de mediador, concluye resultados.
- Autor.
 - Expone su trabajo a la crítica, lleva a cabo los cambios recomendados.
- Evaluador (Tambié: Inspector o revisor).
 - Detecta defectos, desviaciones, áreas problemáticas, etc.
- Secretario (También: Escriba-Escribano)
 - Documenta todos los asuntos, problemas y puntos que hubieran sido identificados.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /1

- El proceso básico de una revisión – como se esquematiza aquí – se aplica a las siguientes **variaciones de las revisiones**
 - Inspección, “Walkthrough”, revisión técnica, revisión informal
 - Estas variaciones difieren en algunos aspectos de la práctica básica esbozada.
- Una diferencia adicional de las revisiones se presenta dependiendo de la naturaleza del objeto de la revisión.
- **Proceso de desarrollo sw** o proceso del proyecto
 - También denominada **revisión de gestión (“management review”)**. Estas revisiones no interfieren directamente en el proceso de pruebas, no forman parte de este curso
- Documentos / productos del proceso de desarrollo.
 - Estas son las revisiones tratadas en el presente curso.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /2

■ Inspección: características relevantes

- Proceso formal basado en reglas, uso de roles definidos
- Los evaluadores inspeccionan el objeto de revisión utilizando listas de comprobación y métricas.
- Un moderador capacitado (formación específica) e independiente dirige la revisión.
- La viabilidad de la revisión del objeto es valorada de forma previa a la revisión.
- Proceso formal para las actividades de preparación, ejecución, documentación y seguimiento.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /3

■ Inspección: ventajas y desventajas

- Sesiones formales y organizadas con roles claramente definidos.
- Requiere actividades intensivas de preparación y seguimiento.
- Son necesarios el moderador y el secretario (alguien que toma nota).
- Propósito principal: detección de defectos utilizando un método estructurado.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /4

- **“Walkthrough”:** características relevantes.
 - Opcionalmente puede haber una preparación de los evaluadores previa a la reunión.
 - La reunión es dirigida por el autor, que explica el objeto en revisión.
 - No es necesario un moderador distinto (el auto modera).
 - A lo largo de la presentación por parte del autor los evaluadores tratan de localizar desviaciones y/o áreas que representen un problema.
 - Ejemplos de “Walkthrough”:
 - “Walkthrough” de documentos.
 - “Walkthrough” de un diseño preliminar de interfaz de usuario.
 - “Walkthrough” de modelo de datos del proceso de negocio.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /5

■ “Walkthrough”: ventajas y desventajas.

- Esfuerzo reducido en la preparación de la sesión de revisión, pero es una sesión cuyo resultado queda abierto.
- Una sesión puede ser iniciada a través de notificaciones realizadas con poca antelación.
- El autor tiene una gran influencia sobre el resultado: dado que él mismo modera la revisión, hay un riesgo de dominación por su parte (puntos críticos no abordados en profundidad).
- Posibilidad limitada de control, dado que el autor también se encuentra a cargo de cualquier actividad de seguimiento.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /6

■ **Revisión técnica: características relevantes.**

- La meta del examen es un aspecto técnico del objeto de revisión: es apto para el uso?.
- Son necesarios expertos, preferiblemente externos.
- La reunión puede tener lugar sin la presencia del autor.
- La revisión se realiza utilizando especificaciones técnicas y otros documentos.
- El panel de expertos presenta sus recomendaciones con carácter unánime.
- Son necesarios preparativos intensivos.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /7

■ Revisiones informales: características relevantes

- Es la forma de revisión más simple.
- Frecuentemente iniciada por el autor.
- Solamente estarán involucrados evaluadores (uno o más).
- No es necesaria ninguna reunión.
- Los resultados pueden ser registrados en forma de una lista de acción.
- Frecuentemente desarrolladas a través de la solicitud de revisión de un documento a un compañero de trabajo.
- También denominada: revisión por pares (Peer review)

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Tipos de revisiones (IEEE 1028) /8

■ Revisiones informales: ventajas y desventajas

- Fácil de efectuar, incluso de los casos de notificaciones realizadas con poca antelación.
- Rentable
- No requiere protocolo.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Factores de éxito de una revisión

- Las revisiones se deben desarrollar **orientada al logro de los objetivos**, es decir, las desviaciones en el objeto revisado deben ser establecidas de forma imparcial.
- El autor del objeto revisado deberá ser motivado de una **forma positiva** por la revisión (“Su documento será aún mejor ”en lugar de“ su documento es de baja calidad”).
- Uso sistemático (entre otras) de las **técnicas** y plantillas implantadas.
- Para la ejecución apropiada de las revisiones será necesario un **presupuesto** suficiente (**10% al 15%** del costo total del desarrollo).
- Hacer uso del efecto de las lecciones aprendidas, utilizar la retroalimentación para implementar un proceso de **mejora** continua.
- Duración de una reunión de revisión: **2 horas máximo**.

III – Técnicas estáticas

01 – Revisiones y el proceso de pruebas

Resumen

- En el transcurso de las **pruebas estáticas** no se ejecuta el objetivo de prueba.
- Las revisiones pueden tener lugar en **fases tempranas** del proceso de desarrollo, ellas complementan / extienden los métodos de pruebas dinámicas.
- **Fases de una Revisión:**
 - Planificación – Preparación – (Preparación Individual) – Reunión - Seguimiento.
- **Roles y Tareas para una Revisión:**
 - Director – Moderador – Autor – Evaluador – Secretario (Toma Nota)
- **Tipos de Revisiones:**
 - Inspección - “Walkthrough” - Revisión Técnica – Revisión Informal.

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Terminología y definiciones

■ Análisis estático (definición):

Es aquella actividad que consiste en el análisis de un objeto de prueba (por ejemplo código fuente, guión (“script”), requisito llevado a cabo sin ejecutar el producto software (si esto fuera posible).

Posibles aspectos a ser comprobados con análisis estático:

- Reglas y estándares de programación.
- Diseño de un programa (análisis del flujo de control).
- Uso de datos (análisis del flujo de datos)
- Complejidad de la estructura de un programa.
 - (Métricas, por ejemplo número ciclomático).

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Aspectos generales /1

- Todos los objetos de prueba deben tener una estructura formal.
 - Esto es especialmente importante cuando se utilizan herramientas de pruebas
 - Con mucha frecuencia no se generan documentos formalmente.
 - En la práctica, lenguajes de modelado, programación y de guión (“scripting”) cumplen con la regla, de la misma forma que algunos diagramas.
- En el análisis estático de un programa mediante el uso de herramientas se desarrolla con un esfuerzo menor que el necesario en una inspección.
 - Por lo tanto con mucha frecuencia, el análisis estático se ejecuta antes de que tenga lugar una revisión.

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Aspectos generales /2

- Herramientas a utilizar: Compiladores y herramientas de análisis (analizadores).
 - Compilador
 - Detecta errores sintácticos en el código fuente de un programa
 - Crea datos de referencia del programa (por ejemplo lista de referencia cruzada, llamada jerárquica, tabla de símbolos),
 - Comprueba la consistencia entre los tipos de variables.
 - Detecta variables no declaradas y código inaccesible (código muerto)
 - Analizador: Trata aspectos adicionales tales como;
 - Convenciones y estándares
 - Métricas de complejidad
 - Acoplamiento de objetos

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

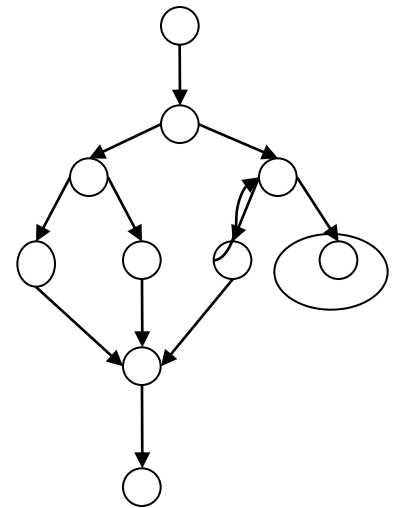
Análisis del flujo de control /1

■ Propósito

- Detectar defectos causados por un desarrollo anómalo del código (ramas muertas, código muerto etc)

■ Método

- La estructura del código se representa como un diagrama de control de flujo.
- Grafo dirigido
 - Los nodos representan sentencias o secuencias de sentencias.
 - Las aristas representan la transferencia del flujo de control, como decisiones y bucles.
 - Construcción mediante herramientas.



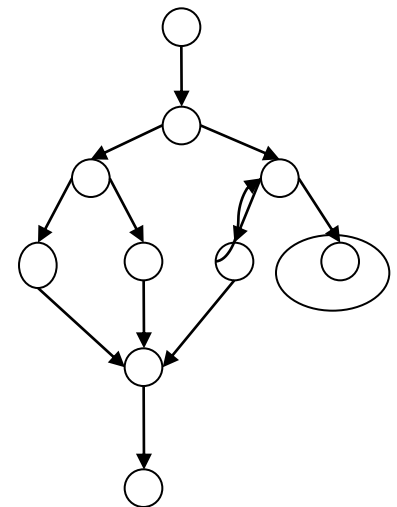
III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Análisis del flujo de control /2

■ Resultados

- Visión del conjunto del código del programa comprensible.
- Las anomalías, pueden ser fácilmente detectadas, los defectos que se hacen evidentes.
 - Bucles abandonados por saltos.
 - Ramas muertas
 - Retornos múltiples.



III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Análisis del flujo de datos /1

■ Propósito

- Detección de anomalías en el flujo de datos con la asistencia de los diagramas de control de flujo y conjeturas racionales respecto de las secuencias del flujo de datos.

■ Beneficios

- Detección fiable de anomalías que den los flujos de datos.
- Se puede detectar fácilmente la localización exacta de defectos.
- Es un buen complemento para otros métodos de pruebas.

■ Desventajas

- Limitado a un rango reducido de tipos de defectos.

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Análisis del flujo de datos /2

■ Método

- Una variable x puede tener los siguientes estados a lo largo de la ejecución de un programa:
 - X se encuentra definida (**d**): ha sido un valor asignado a la variable x (por ejemplo $x=1$).
 - X se encuentra definida (**u**): no ha sido asignado valor alguno a la variable x .
 - X está referenciada (**r**): ha sido tomada una referencia, el valor de x no cambia (por ejemplo $(x>0)$).
 - X no ha sido utilizada (**e**): x no ha sido referenciada (ni en lectura ni en escritura).
- El flujo de datos de una variable puede ser expresado como una secuencia de estados: **d**, **u**, **r** y **e**.
- Si una de estas secuencias contiene una sub-secuencia que no tiene sentido, entonces ha tenido lugar una anomalía en el flujo de datos.

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Análisis del flujo de datos /2

- Anomalía en el flujo de datos.
 - Ejemplo III/03-1

Para este ejemplo, los valores de dos variables son intercambiados a través de una variable auxiliar, si no están ordenados por valor.

```
Void MinMax (int Min, int Max)
{
    int Help;
    if (Min > Max)
    {
        Max = Help;
        Max = Min;
        Help = Min;
    }
}
End MinMax
```

El analista del flujo de datos muestra>

- La variable **Help** se encuentra indefinida (undelined) cuando es referenciada ("referenced"), anomalía – ur.
- La variable **Max** se define ("defined") dos veces sin ninguna referencia entre ambas definiciones: anomalía – dd
- El valor definido ("defined") para la variable **Help** se vuelve indefinido (undefined) final del programa sin referencia: anomalía - du.

Para este ejemplo, los valores de dos variables son intercambiados a través de una variable auxiliar, si no están ordenados por valor.

```
Void MinMax (int Min, int Max)
{
    int Help;
    if (Min > Max)
    {
        Help = Max;
        Max = Min;
        Min = Help;
    }
}
End MinMax
```

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Las métricas y su cálculo.

- Ciertos aspectos de la calidad de un programa pueden ser medidos utilizando métricas.
 - La métrica sólo tiene relevancia para el aspecto medido (considerado).
- La complejidad estática de un programa puede ser medida.
 - Actualmente hay aproximadamente 100 métricas diferentes disponibles.
- Métricas diferentes tratan aspectos diferentes de la complejidad de programa.
 - Tamaño del programa (por ejemplo líneas de código (“lines of code – LOC”)).
 - Estructuras de control del programa (por ejemplo número ciclomático).
 - Estructuras de control de datos (por ejemplo métrica de Halstead (“Haistead - Metric”))
 - Es difícil comparar dos métricas diferentes, incluso cuando ambas abordan el mismol atributo del programa!.

III – Técnicas estáticas

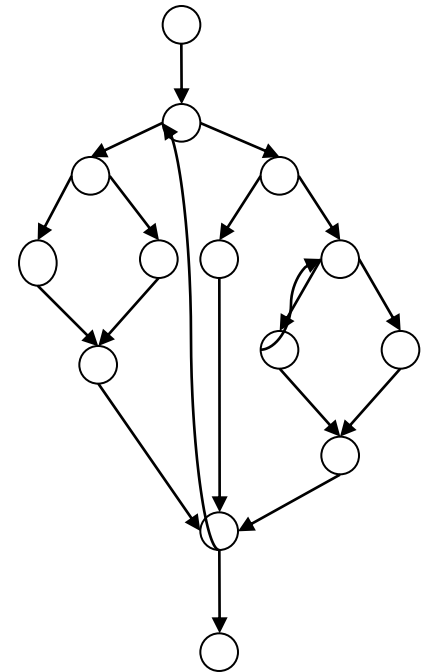
02 – Análisis estático mediante herramientas.

Las métricas y su implicación /1

■ Número ciclomático $v(G)$

- Métrica que mide la complejidad estática de un programa basada en su grafo de flujo de control.
- Mide los caminos linealmente independiente, como índice de la testabilidad y mantenibilidad.
- El número ciclomático se define de la siguiente forma:
 - Número de aristas: **e**
 - Número de nodos: **n**
 - Número de partes del programa independientes inspeccionadas: **p** (normalmente 1).
- Valores hasta 10 son aceptables. Para valores superiores el código debe ser “reworked” /mejorado (buena práctica. McCabe).

$$v(G) = e - n + 2p$$



III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Las métricas y su implicación /2

■ Ejemplo III/03-2

(Número Ciclomático).

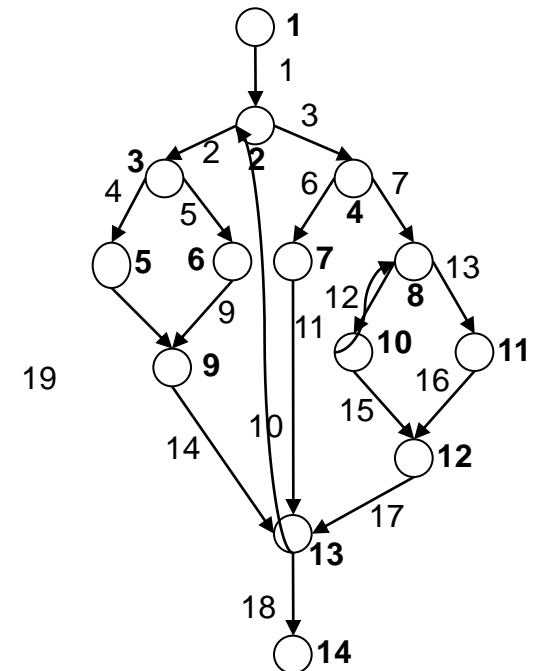
- El grafo de la derecha tiene:
 - 1 parte independiente
 - 14 nodos
 - 19 aristas.
- Esto conduce al número ciclomático:

$$v(G) = e - n = 2p$$

$$v(G) = 7$$

$$\begin{aligned} p &= 1 \\ n &= 14 \\ e &= 19 \end{aligned}$$

$$v(G) = e - n = 2p$$



III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Las métricas y su implicación /3

- Número ciclomático (por McCabe) – implicación.
 - El número ciclomático puede ser utilizado como un valor objetivo para una revisión de código.
 - El número ciclomático también puede ser calculado como el número de decisiones independientes más uno. Si las dos formas de cálculo aportan resultados diferentes se puede deber a:
 - Ramas superfluas.
 - Ramas faltantes.
 - El número ciclomático también aporta un índice del número de casos de prueba necesarios (para alcanzar cobertura de decisión).

III – Técnicas estáticas

02 – Análisis estático mediante herramientas.

Resumen

Análisis estático

- Con el uso de herramientas para la realización del análisis estático (compiladores, analizadores) el código del programa puede ser objeto de inspección **sin** ser ejecutado.
- Con el uso de **herramientas** se puede realizar el **análisis estático** de un programa con un **esfuerzo menor** que el necesario para una **inspección**

Resultado del análisis

- El **diagrama de flujo de control** presenta el flujo del programa y permite la detección de “ramas muertas” y código inalcanzable.
- Las anomalías en los datos se detectan utilizando el **análisis del flujo de datos**.
- Las **métricas** pueden ser utilizadas para evaluar la complejidad estructural conduciendo a una estimación del esfuerzo en pruebas a esperar.

IV – Técnicas de diseño de pruebas

00 – Agenda

Capítulo III – Técnicas estáticas de pruebas

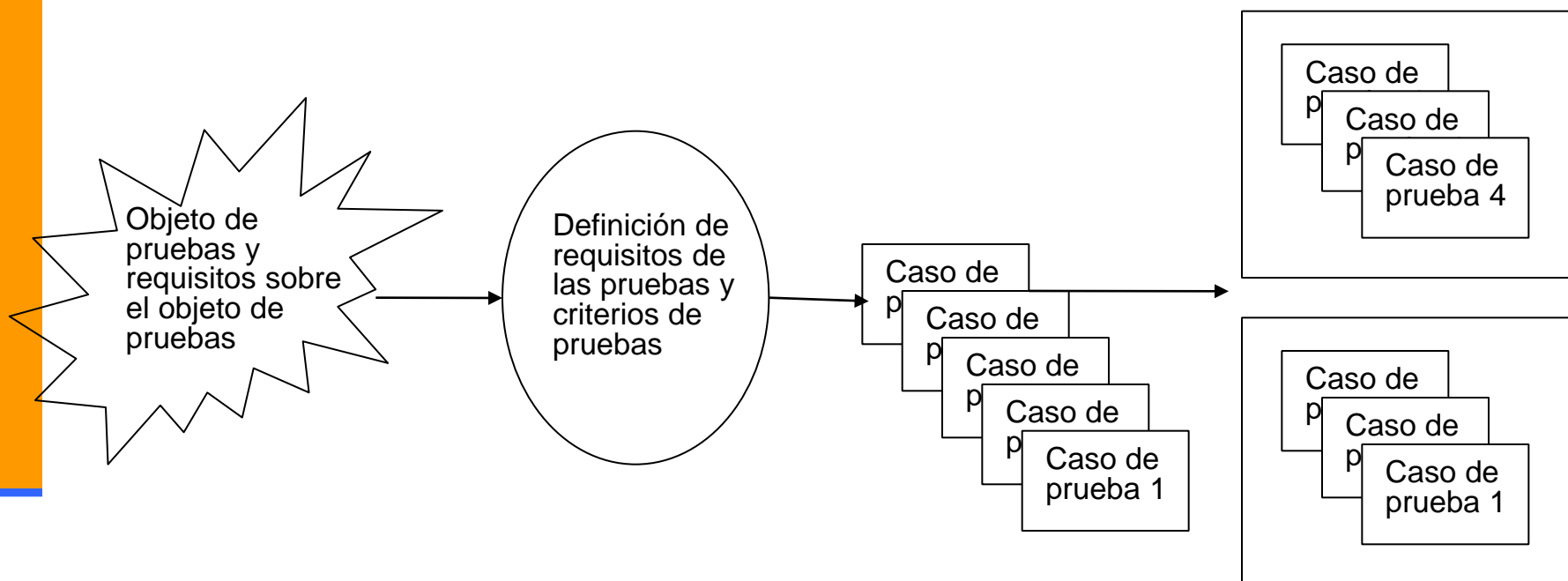
- IV/01 Diseño de casos de prueba.
- IV/02 Categorías de las técnicas de diseño de pruebas.
- IV/03 Técnicas de caja negra ("black box").
- IV/04 Técnicas de caja blanca ("white box").
- IV/05 Técnicas basadas en la experiencia.
- IV/06 Selección de las técnicas de pruebas

IV – Técnicas de diseño de pruebas

01 – Diseño de casos de prueba

Obtención de casos de prueba a partir de requisitos

El diseño de casos de prueba debe ser un proceso controlado



Los casos de prueba pueden ser **creados formal** o **informalmente**, dependiendo de las características del proyecto y la madurez del proceso de uso

IV – Técnicas de diseño de pruebas

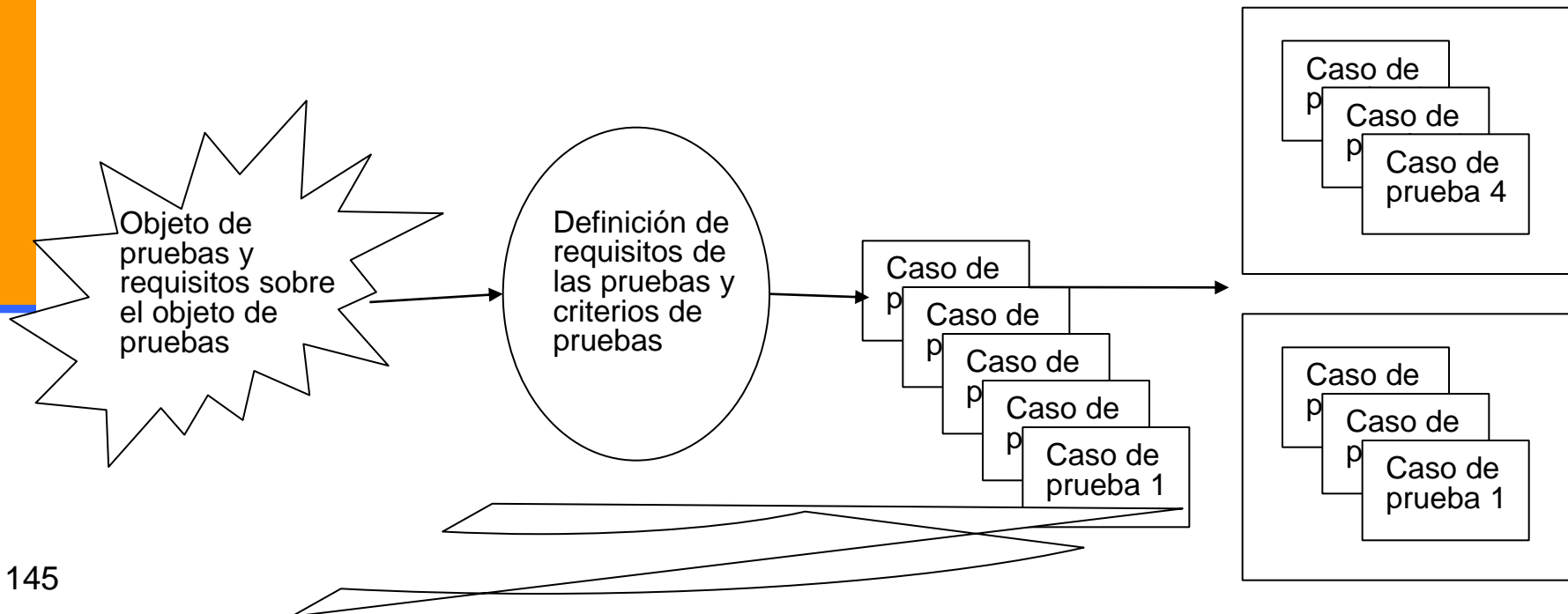
01 – Diseño de casos de prueba

Trazabilidad

Las pruebas deben ser **trazables**: qué casos de prueba han sido incluidos en el catálogo de pruebas basados en qué requisitos?.

Las **consecuencias** sobre las pruebas **de los cambios** a realizar en los requisitos pueden ser identificados directamente.

La trazabilidad también ayuda a determinar la **cobertura de requisitos**.



IV – Técnicas de diseño de pruebas

01 – Diseño de casos de prueba

Definiciones

- **Objeto de pruebas (“Test Object”):**
 - El elemento a ser revisado: un documento o pieza de software en el proceso de desarrollo de software.

- **Condición de la prueba (“Test Condition”):**
 - Un elemento o evento: una función, una transacción, un criterio de calidad o un elemento en el sistema.

- **Criterios de prueba (“Test Criteria”):**
 - El objeto de prueba debe cumplir los criterios de prueba con el objeto de superar la prueba.

IV – Técnicas de diseño de pruebas

01 – Diseño de casos de prueba

Descripción de un caso de prueba según el estándar IEEE 829

- **Valores de entrada (“Input values”)**: descripción de los datos de entrada de un objeto de pruebas.
- **Precondiciones (“Preconditions”)**: situación previa a la ejecución de pruebas o características de un objeto de pruebas antes de llevar a la práctica (ejecución) un caso de prueba.
- **Resultados Esperados (“Expected Results”)**: datos de salida que se espera que produzca un objeto de pruebas (resultado esperado - “expected result”)
- **Poscondiciones (“Post Conditions”)**: características de un objeto de pruebas tras la ejecución de pruebas, descripción de su situación tras la ejecución de las pruebas.
- **Dependencias (“Dependencies”)**: orden de ejecución de casos de prueba, razón de las dependencias.
- **Identificador Distinguido (“Distinct Identification”)**: Identificador o código con el objeto de vincular, por ejemplo, un informe de errores al caso de prueba en el cual ha sido detectado.
- **Requisitos (“Requirements”)**: características del objeto de pruebas que el caso de prueba debe evaluar.

IV – Técnicas de diseño de pruebas

01 – Diseño de casos de prueba

Combinación de casos de prueba

Los casos de prueba se pueden cambiar en los (**“test suites”**) y escenarios de pruebas.

Una especificación de procedimiento de prueba: define la secuencia de acciones para la ejecución de un caso de prueba individual o un test suites. Es un guión o esquema de pruebas describiendo los pasos, el tratamiento y/o las actividades necesarios para la ejecución de pruebas.

Los test suites pueden ser codificados y ejecutados de forma automática con el uso de herramientas adecuadas.

El **plan de pruebas** (**“test plan”**) establece la **secuencia** de las pruebas planificadas, **quién** debe ejecutarlas y **cuándo**. Las restricciones a considerar son las prioridades, la disponibilidad de recursos, la infraestructura de pruebas, etc.

IV – Técnicas de diseño de pruebas

01 – Diseño de casos de prueba

Resumen

- Los casos de prueba y los (“test suites”) son obtenidos a partir de los requisitos o características de los objetos de pruebas.

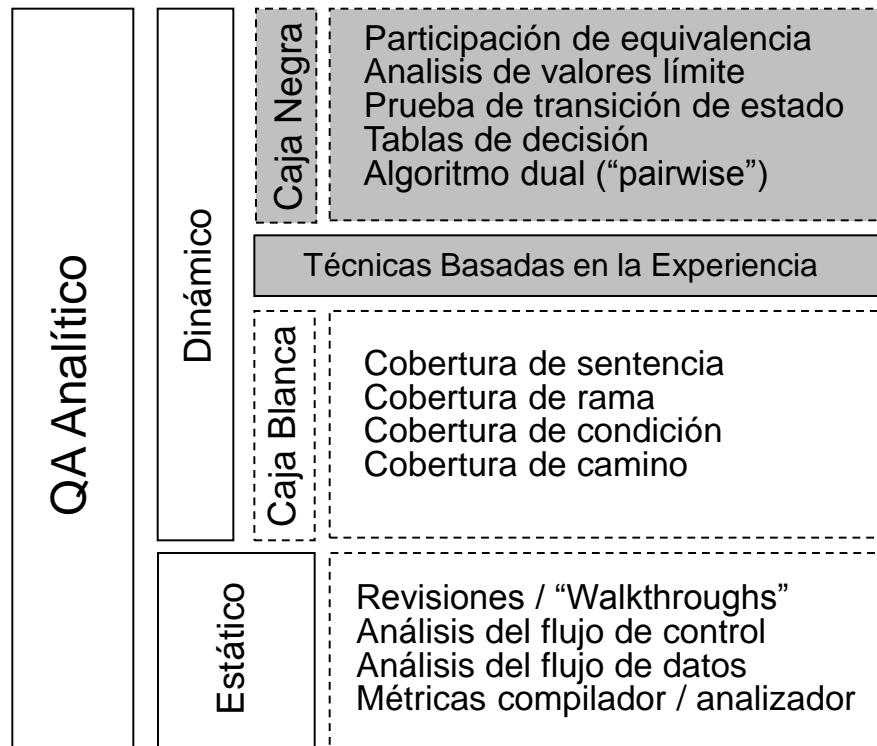
- Componentes de la descripción de un caso de prueba:
 - Código / identificador.
 - Valores de entrada (“input values”)
 - Precondiciones (“pre-conditions”)
 - Resultados esperados (“expected results”)
 - Poscondiciones (“post-conditions”)
 - Dependencias (“dependencies”)
 - Requisitos a partir de los cuales se ha obtenido el caso de prueba.

IV – Técnicas de diseño de pruebas

02 – Categorías de las técnicas de diseño de pruebas

Pruebas de caja negra (“black box”) y caja blanca (“white box”)

- Las pruebas dinámicas se dividen en dos categorías / grupos



- La agrupación se realiza en función del carácter básico del método utilizado para obtener los casos de prueba .



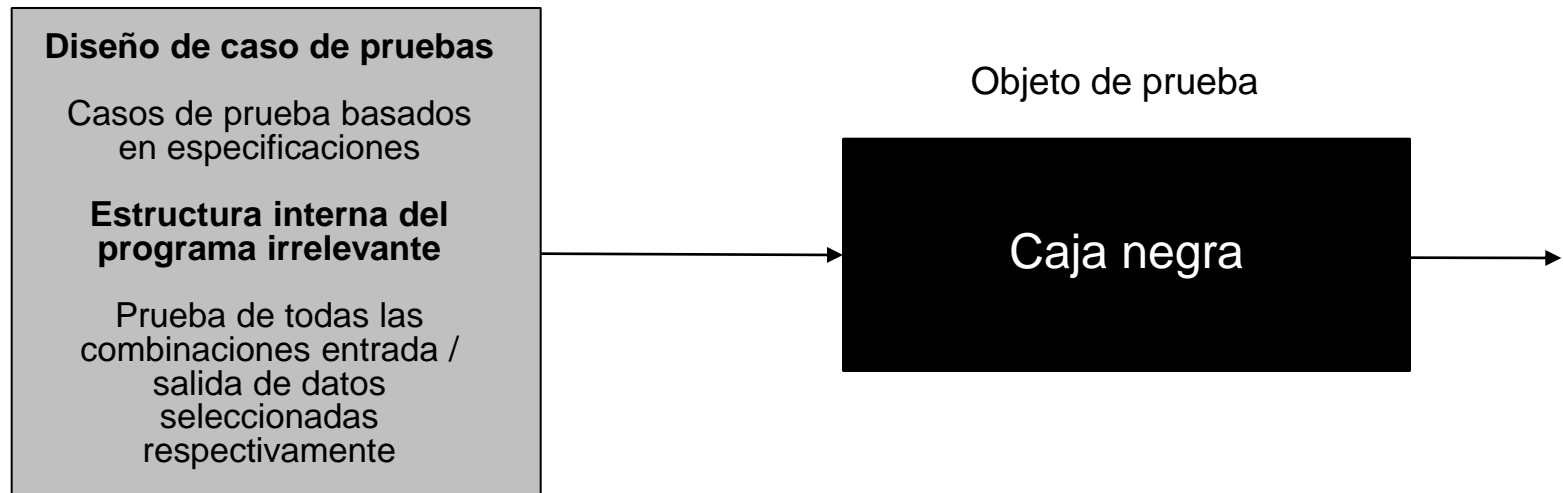
- Cada grupo tiene sus propios métodos para diseñar casos de prueba.

IV – Técnicas de diseño de pruebas

02 – Categorías de las técnicas de diseño de pruebas

Técnica: caja negra (“Black box”)

- El tester observa el objeto de prueba como una caja negra.
 - La estructura interna del objeto de prueba es irrelevante o desconocida.
- Los casos de prueba se obtienen a partir del análisis de la especificación (funcional y no funcional) de un componente o sistema.
 - Prueba del comportamiento entrada / salida (input / output).
- La funcionalidad es el foco de atención!
 - La técnica de caja negra también se denomina prueba **funcional** o prueba **orientada a la especificación**.



IV – Técnicas de diseño de pruebas

02 – Categorías de las técnicas de diseño de pruebas

Técnica: caja blanca (“White box”)

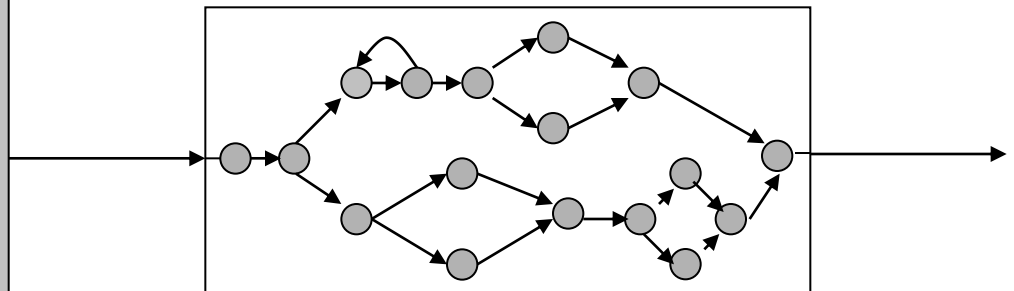
- El tester conoce la estructura interna del programa / código.
 - Por ejemplo jerarquía de los componentes, flujos de control, flujo de datos, etc.
- Los casos de prueba son seleccionados en base a la estructura interna del programa / código
 - A lo largo de las pruebas es posible que se interfiera con la ejecución de las pruebas.
- ¡La estructura del programa es el foco de atención!
 - La técnica de caja blanca también es conocida como **prueba estructural** o **pruebas basadas en el flujo de control**.

Diseño de caso de pruebas

Los casos de prueba están basados en la estructura del programa

El proceso de pruebas es controlado externamente.

Análisis del flujo de control dentro del objeto de prueba a lo largo de la ejecución de pruebas.



IV – Técnicas de diseño de pruebas

02 – Categorías de las técnicas de diseño de pruebas

Categoría de las técnicas de diseño de pruebas- visión general

- Métodos basados en la especificación.
 - El objeto de prueba ha sido seleccionado de acuerdo con el modelo funcional software.
 - La cobertura de la especificación puede ser medida (por ejemplo, el porcentaje de la especificación cubierta por casos de prueba).
- Métodos basados en la estructura.
 - La estructura interna del objeto de prueba es utilizada para diseñar los casos de prueba (código / sentencias, menús, llamadas, etc.)
 - El porcentaje de cobertura es medido y utilizado como fuente para la creación de casos de prueba adicionales.
- Métodos basados en la experiencia.
 - El conocimiento y experiencia respecto de los objetos de prueba y su entorno son las fuentes para el diseño de casos de prueba.
 - El conocimiento y experiencia respecto a posibles puntos débiles, posibles errores y errores previos son utilizados para determinar/definir casos de prueba.

IV – Técnicas de diseño de pruebas

02 – Categorías de las técnicas de diseño de pruebas

Resumen.

- Los casos de prueba pueden ser diseñados utilizando diferentes métodos.
 - Si la **funcionalidad** especificada es el objetivo de las pruebas, los métodos utilizados se denominan métodos basados en la especificación o métodos de **caja negra (“black box”)**.
 - Si la estructura interna de un objeto es investigada, los métodos utilizados se denominan métodos basados en la estructura o métodos de **caja blanca (“white box”)**.
 - Los métodos basados en la **experiencia** utilizan el conocimiento y la habilidad del personal involucrado en el diseño de casos de prueba.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Visión general

- En el presente apartado se explicaran en detalle los siguientes métodos de caja negra:
 - Partición equivalente o clase de equivalencia.
 - Análisis de valores límites.
 - Tablas de decisión & gráficos causa – y - efecto.
 - Pruebas de transición de estado.
 - Pruebas de caso de uso.
- La lista anterior da cuenta de los métodos más importantes y conocidos.
- Otros métodos de caja negra son:
 - Pruebas estadísticas.
 - Pruebas duales (algoritmo dual – “pairwise”).
 - Pruebas de humo.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

General

- Las pruebas funcionales están dirigidas a verificar la correctitud y la completitud de una función.
 - ¿Están disponibles en el módulo todas las funciones especificadas?
 - ¿Las funciones ejecutadas presentan resultados **correctos**?
- La ejecución de casos de pruebas deberían de ser ejecutados con una baja redundancia, pero sin embargo con carácter integral.
 - Probar lo **menos** posible, pero.
 - Probar **tanto** como sea necesario.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación de equivalencia o clase de equivalencia (CE)

- La participación en clases de equivalencia es lo que la mayoría de los tester hacen en forma intuitiva: dividen los posibles valores en clases, mediante lo cual observan.
 - Los valores de **entrada (“input values”)** de un programa (uso habitual del método CE).
 - Los valores de **salida (“output values”)** de un programa (uso poco habitual del método CE).
- El rango de valores definido se agrupa en **clases de equivalencia** para las cuales se aplican las siguientes reglas:
 - Todos los valores para los cuales **se espera** que el programa tenga un **comportamiento común** se agrupan en una clase de equivalencia (CE).
 - Las clases de equivalencia pueden **no superponerse** y pueden **no presentar ningún salto (discontinuidad)**.
 - Las clases de equivalencia pueden consistir en un **rango** de valores (por ejemplo, $0 < x < 10$) o en un valor **aislado** (por ejemplo, $x = \text{Verdadero}$).

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación de equivalencia o clase de equivalencia (CE)

- Las pruebas se ejecutan utilizando un **único representante** de cada CE.
 - Para todo otro valor perteneciente a la CE se espera el mismo comportamiento que para el valor seleccionado.
- Las clases de equivalencia se escogen para entradas (“inputs”) **válidas y no válidas**.
 - Si el valor x se define como $0 < x < 10$ entonces, inicialmente, se pueden identificar tres clases de equivalencia:
 1. $x \leq 0$ (valores de entrada no válidos).
 2. $0 < x < 10$ (valores de entrada válidos).
 3. $x \geq 10$ (valores de entrada no válidos).
- Se pueden definir CE adicionales, conteniendo, pero no limitadas a:
 - Entradas no numéricas.
 - Números muy grandes o muy pequeños.
 - Formatos numéricos no admitidos.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/1

- Todas las **variables de entradas** (“input variables”) del objeto de prueba son identificados, por ejemplo:
 - Campos de una interfaz Gráfica de Usuario (“GUI”).
 - Parámetros de una función.
- Se **define** un **rango** para cada valor de entrada (“input”).
 - Este rango define la suma de todas las clases de equivalencia **válidas** (CEv).
 - Las clases de equivalencia **inválidas** (CEi) están constituidas por aquellos valores no pertenecientes al rango.
 - Aquellos valores que deben ser tratados de una forma **diferente** (conocidos o sospechosos) son asignados a una **clase de equivalencia aparte**.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/2

Ejemplo 1:

- Un programa espera el valor de un peso expresado en Kg (todos los valores serán redondeados internamente).
- Todos los valores ≥ 0 son válidos y todos los valores negativos o no numéricos (“fred”) son no válidos.
 - Clase de equivalencia válida $x \geq 0$
 - Clase de equivalencia no válida $x < 0$
 - Clase de equivalencia no válida $x = \text{no numérico}$.
- Aparte de los CE no válidos establecidos previamente, habrá otras clases inválidas que serán ignoradas en estos ejemplos:
 - $x > 0$, pero mayor que el número más grande que el sistema involucrado puede aceptar.
 - $x > 0$, pero menor que el menor de los números que el sistema involucrado puede aceptar.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/3

- La clase de equivalencia para cada variable (elemento) será dividida adicionalmente.
 - **CE válida:** todos los valores pertenecientes al rango de definición se combinan en una única clase de equivalencia si son tratados de forma idéntica por el objeto de prueba.
 - Este es el caso de todos los valores numéricos positivos y el valor cero en el ejemplo 1.
 - **CE no válida:** se distinguen dos casos que se encuentran fuera del rango de definición.
 - Valores con un formato correcto pero fuera del rango pueden ser combinados en una o más clases de equivalencia.
 - Valores expresados en un formato erróneo pero, generalmente contruidos en una CE independiente.
 - En el ejemplo 1 resultan dos clases de equivalencia resultan en : $x < 0$ y x es un valor numérico.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/4

- En un último paso, se identifica un representante de cada CE, así como del resultado esperado para éste.
 - Para el ejemplo IV/03-1 puede ser:

| Variable | Clase de equivalencia | Representante |
|----------|-------------------------------------|---------------|
| Peso | EC ₁ : $x \geq 0$ | +1,00 |
| | EC ₂ : $x < 0$ | -1,00 |
| | EC ₃ : x no es un número | fred |

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método / 5

Ejemplo 2:

- Por parte del código de un programa trata el precio final de un artículo en base a su precio de venta al público, un descuento en % y el precio del porte (6, 9 ó 10 euros, dependiendo del tipo de porte).

| Variable | Clase de equivalencia | Estado | Representante |
|------------------|---|-----------|---------------|
| Peso | EC ₁₁ : $x \geq 0$ | Válido | 1000,00 |
| | EC ₁₂ : $x < 0$ | No válido | -1000,00 |
| | EC ₁₃ : x valor no numérico | No válido | fred |
| Descuento | EC ₂₁ : $0\% \leq x < 100\%$ | Válido | 10% |
| | EC ₂₂ : $x < 0\%$ | No válido | -10% |
| | EC ₂₃ : $x > 100\%$ | No válido | 200% |
| | EC ₂₄ : x valor no numérico | No válido | fred |
| Precio del porte | EC ₃₁ : $x = 6$ | Válido | 6 |
| | EC ₃₂ : $x = 9$ | Válido | 9 |
| | EC ₃₃ : $x = 12$ | Válido | 12 |
| | EC ₃₄ : $x \in \{6, 9, 12\}$ | No válido | 4 |
| | EC ₃₅ : x valor no numérico | No válido | fred |

Suposiciones:

- El precio de venta al público de un artículo está dado por un número con dos decimales.
- El descuento es el valor porcentual sin decimales entre 0% y 100%.
- El precio del porte puede ser 6, 9 ó 12.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/6

Ejemplo V/03-2:

- Las clases de equivalencia válidas aportan las siguientes combinaciones o casos de prueba: T01, T02 y T03.

| Variable | Clase de equivalencia | Estado | Representante | T01 | T02 | T03 |
|------------------|--|-----------|---------------|-----|-----|-----|
| Peso | EC ₁₁ : $x \geq 0$ | Válido | 1000,00 | * | * | * |
| | EC ₁₂ : $x < 0$ | No válido | -1000,00 | | | |
| | EC ₁₃ : x valor no numérico | No válido | fred | | | |
| Descuento | EC ₂₁ : $0\% \leq x \leq 100\%$ | Válido | 10% | * | * | * |
| | EC ₂₂ : $x < 0\%$ | No válido | -10% | | | |
| | EC ₂₃ : $x > 100\%$ | No válido | 200% | | | |
| | EC ₂₄ : x valor no numérico | No válido | fred | | | |
| Precio del porte | EC ₃₁ : $x = 6$ | Válido | 6 | * | | |
| | EC ₃₂ : $x = 9$ | Válido | 9 | | * | |
| | EC ₃₃ : $x = 12$ | Válido | 12 | | | * |
| | EC ₃₄ : $x \notin \{6, 9, 12\}$ | No válido | 4 | | | |
| | EC ₃₅ : x valor no numérico | No válido | fred | | | |

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/7

Ejemplo IV/03-2:

- Los siguientes casos de prueba han sido generados utilizando CE no válidas, cada una en combinación con CE válidas de otros elementos.

| Variable | Clase de equivalencia | Estado | Representante | T04 | T05 | T06 | T07 | T08 | T09 | T10 |
|------------------|--|-----------|---------------|-----|-----|-----|-----|-----|-----|-----|
| Peso | EC ₁₁ : $x \geq 0$ | Válido | 1000,00 | | | * | * | * | * | * |
| | EC ₁₂ : $x < 0$ | No válido | -1000,00 | * | | | | | | |
| | EC ₁₃ : x valor no numérico | No válido | fred | | * | | | | | |
| Descuento | EC ₂₁ : $0\% \leq x < 100\%$ | Válido | 10% | * | * | | | | * | * |
| | EC ₂₂ : $x < 0\%$ | No válido | -10% | | | * | | | | |
| | EC ₂₃ : $x > 100\%$ | No válido | 200% | | | | * | | | |
| | EC ₂₄ : x valor no numérico | No válido | fred | | | | | * | | |
| Precio del porte | EC ₃₁ : $x = 6$ | Válido | 6 | * | * | * | * | * | | |
| | EC ₃₂ : $x = 9$ | Válido | 9 | | | | | | | |
| | EC ₃₃ : $x = 12$ | Válido | 12 | | | | | | | |
| | EC ₃₄ : $x \notin \{6, 9, 12\}$ | No válido | 4 | | | | | | * | |
| | EC ₃₅ : x valor no numérico | No válido | fred | | | | | | | * |

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Participación en clases de equivalencia: método/8

Ejemplo IV/03-2:

- Se obtienen 10 casos de prueba: 3 casos de prueba positivos (valores válidos) y 7 casos de prueba negativos (valores no válidos).

| Variable | Estado | Representante | T01 | T02 | T03 | T04 | T05 | T06 | T07 | T08 | T09 | T10 |
|------------------|-----------|---------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Peso | Válido | 1000,00 | * | * | * | | | * | * | * | * | * |
| | No válido | -1000,00 | | | | * | | | | | | |
| | No válido | fred | | | | | * | | | | | |
| Descuento | Válido | 10% | * | * | * | * | * | | | | * | * |
| | No válido | -10% | | | | | | * | | | | |
| | No válido | 200% | | | | | | | * | | | |
| | No válido | fred | | | | | | | | * | | |
| Precio del porte | Válido | 6 | * | | | * | * | * | * | * | | |
| | Válido | 9 | | * | | | | | | | | |
| | Válido | 12 | | | * | | | | | | | |
| | No válido | 4 | | | | | | | | | * | |
| | No válido | fred | | | | | | | | | | * |

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia: basado en la salida (“out based”)

- Las clases de equivalencia también pueden ser generadas a partir de los valores de salida esperados.
 - El método utilizado es análogo al anterior, aplicando los valores de salida.
 - La variable (elemento) es entonces la salida (“output”) (por ejemplo, el valor de un campo en la “GUI”).
 - Las clases de equivalencia son generadas para todos los posibles valores de la salida definidos.
 - Se determina un representante para cada clase de equivalencia de los valores de salida.
 - Entonces, el valor de entrada, que conduce al valor representante, es obtenido/identificado.
- Los mayores costos y esfuerzo dado que los valores de entrada deben ser obtenidos para una salida determinada de forma recursiva

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia- en general / 1

- Participación.
 - La calidad de la prueba depende de la segmentación precisa de variables/elementos en clases de equivalencia.
 - CE que no hubieran sido identificadas presentan el riesgo de posibles omisiones, dado que los representantes utilizados no cubren todas las posibilidades.
- Casos de prueba.
 - El método de la clase de equivalencia aporta casos de prueba para los cuales aún debe ser seleccionado un representante.
 - Las combinaciones de datos de prueba son seleccionados definiendo el o los representantes de cada clase de equivalencia.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia- en general / 2

- Sección de representantes.
 - Cada valor perteneciente al CE puede ser un representante. Los óptimos son:
 - Valores característicos (“typical values”) (utilizados con frecuencia).
 - Valores problemático (“problem values”) (sospechosos de producir fallos)
 - Valores límite (“boundary values”) (en la frontera de la CE).
 - Durante la creación de casos de prueba los representantes de CE inválidos deben combinarse siempre con los mismos valores de otros CE válidos (combinaciones estándar).
 - Dado el posible enmascaramiento de errores, se debe evitar las combinaciones de representantes de CE inválidas con representantes de otras CE inválidas en un mismo caso de prueba.
 - La selección de representantes implica que la función implementada por el programa/sistema utiliza comparadores.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia- en general / 3

- La cobertura de clases de equivalencia puede ser utilizada como criterio de salida para finalizar las actividades del proceso de pruebas.

$$\text{Cobertura (CE)} = \frac{\text{Número de CE probados}}{\text{Número de CE definidos}} * 100\%$$

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia- en general / 4

- La transición de la especificación o definición con la funcionalidad a la creación de la clase de equivalencia.
 - Con frecuencia es una tarea difícil debido a la carencia de documentación precisa y completa.
 - Los límites no definidos o las descripciones faltantes hacen difícil la definición de las clases de equivalencia.
 - Con frecuencia, es necesario mantener contacto con el cliente con el objeto de completar la información.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Partición en clases de equivalencia- en general / 5

- Beneficios.
 - Método sistemático para el diseño de casos de prueba por ejemplo, con una mínima cantidad de casos de prueba se pueden esperar un valor de cobertura específico.
 - La partición del rango del valores en clases de equivalencia a partir de las especificaciones cubre los requisitos funcionales.
 - La asignación de prioridades a la clases de equivalencia puede ser utilizada para la asignación de prioridades a los casos de prueba (los valores de entrada utilizados con poca frecuencia deben ser los últimos en ser probados).
 - Las pruebas de las excepciones conocidas está cubierta por los casos de prueba de acuerdo con las clases de equivalencia negativa.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Análisis de valores límite / 1

- El análisis de valores límite amplía la técnica de partición en clases de equivalencia introduciendo una regla para seleccionar a las representantes
- Los valores frontera (valores límite) de la clase de equivalencia deben ser probados de forma intensiva.
- ¿Porqué prestar mas atención a los límites?
 - Frecuentemente los límites del rango de valores no están bien definidos o conducen a distintas interpretaciones
 - Comprobar si los límites han sido implementados (programados) correctamente.
- **Importante:**
 - ¡La experiencia demuestra que, con mucha frecuencia, los errores tienen lugar en los límites del rango de valores!

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Análisis de valores límite / 2

El análisis de valores límite supone que:

- La clase de equivalencia está compuesta de un rango continuo de valores (no por un valor individual o un conjunto de valores discretos).
- Se pueden definir los límites para el rango de valores.
- Adicionalmente a la partición en clases de equivalencia el análisis de valores límite es un método que sugiere la selección de representantes.
 - Partición en clases de equivalencia.
 - Evalúa un valor (típico) de la clase de equivalencia.
 - Análisis de valores límite:
 - Evalúa los valores límite (frontera) y su entorno.
 - Se utilizó el siguiente esquema:

| | | |
|--|-----------------|---------------------------|
| | | |
| Valor mínimo $-\delta$ | Límite inferior | Límite inferior $+\delta$ |
| Valor máximo $-\delta$ | Límite Superior | Valor máximo $-\delta$ |
| δ es el menor incremento definido para el valor. Por ejemplo: 1 para valores enteros | | |

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Definición de valores límite

- El esquema básico sólo puede ser aplicado cuando el **rango** de valores ha sido definido de conformidad con el mismo esquema.
 - En este caso no son necesarias pruebas adicionales para un valor en el interior del rango de valores.
- Si un CE está definido como un único valor numérico, por ejemplo, $x = 5$, los valores corresponde al entorne también serán utilizados.
 - ¡Los representantes (de la clase y su entorno) son 4, 5 y 6.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Análisis de valores límite para CE no válida

- Los valores límites para clases de equivalencia no validas tienen poco sentido.
 - Los representantes de una CE no válida en la frontera de una CE válida ya se encuentran cubiertas a través del esquema básico.
- Para rangos de valores definidos como un conjunto de valores, en general, no es posible definir los valores límites.
 - Por ejemplo: *Soltero, casado, divorciado, viudo.*

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Análisis de valores límite ejemplo 3a

Ejemplo 3ª:

- Rango de valores para un descuento en %: $0,00 \leq x \leq 100,00$.
- **Definición de CE**
3 clases:
 - 1. CE: $x < 0$
 - 2. CE $0,00 \leq x \leq 100,00$
 - 3. CE: $x > 100$
- **Análisis de valores límite.**
 - *Extiende los representantes a:*
 - 2. EC: -0,01; 0,00; 0,01; 99,99; 100,00; 100,01
 - **Nota importante:**
 - En lugar de un representante para la CE válida, ahora hay seis representantes (cuatro válidos y dos no válidos).

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Análisis de valores límite ejemplo 3b

Ejemplo 3^a:

- Esquema básico: seleccionar tres valores con el objeto de ser probados – el valor límite exacto y dos valores pertenecientes al entorno (dentro y fuera del CE).
- Punto de vista alternativo: dado que el valor límite pertenece a la CE, sólo son necesarios dos valores para las pruebas, uno perteneciente al CE y otro no perteneciente al CE.
- **Ejemplo 3b:**
 - Rango de valores para un descuento en % $0,00 \leq x \leq 100,00$
 - **CE válido:** $0,00 \leq x \leq 100,00$
 - **Análisis de valores límite.**
 - Los representantes adicionales son: -0,01; 0,00; 100,00; 100,01
 - 0,01 – mismo comportamiento que 0,00
 - 99,99 – mismo comportamiento que 100,00
- Un error de programación causado por un operador de comparación erróneo será detectado con los dos valores límites (frontera).

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decisión table testing”)/1

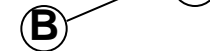
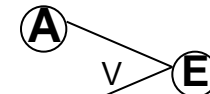
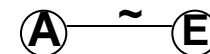
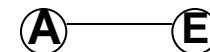
- La participación en clases de equivalencia y el análisis de valores límite tratan **entradas** en condiciones **aisladas**.
- Sin embargo, una condición de entrada puede tener efectos sólo en **combinación** con otras condiciones de entrada.
- Todos los métodos descritos previamente no tienen en cuenta el efecto de **dependencias y combinaciones**.
- El uso del conjunto completo de las combinaciones de todas las clases de equivalencia de entrada conduce, normalmente, a un número muy alto de casos de prueba (explosión de casos de prueba).
- Con la ayuda de **diagramas causa- y - efecto** y tablas de decisión obtenidas a partir de aquellas, la cantidad de combinaciones posibles se puede reducir de forma sistemática a un subconjunto de las mismas.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decision table testing”)/2

- El diagramas causa- y - efecto utiliza un lenguaje formal.
- El diagramas causa- y - efecto se genera traduciendo la especificación (normalmente informal) de un objeto de prueba a un lenguaje formal.
- El objeto de prueba está sometido a una determinada cantidad de efectos que se remontan a sus respectivas causas.
- Elementos 7 símbolos:
 - Aseveración (“Assertion”).
 - (si causa A – entonces efecto E).
 - Negación (“Negation”).
 - (si causa A – entonces no efecto E).
 - O (“or”).
 - (si causa A o B – entonces efecto E).
 - Y (“and”).
 - (si causa A y B – entonces efecto E).



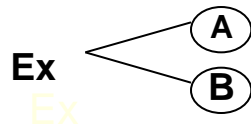
IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

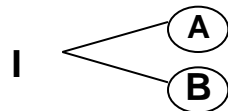
Pruebas de tabla de decisión (“decision table testing”)/3

- Otros elementos del diagrama causa- y – efecto.

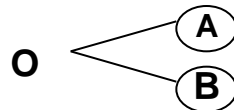
- Exclusivo (“exclusive”) (o causa A o causa B).



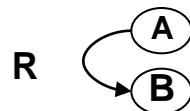
- inclusivo (“inclusive”) (por lo menos una de las dos causas; A or B).



- Uno y sólo uno (Una y exactamente una de las dos causas A or B).



- Requerido (“required”) (si causa A entonces también causa B)



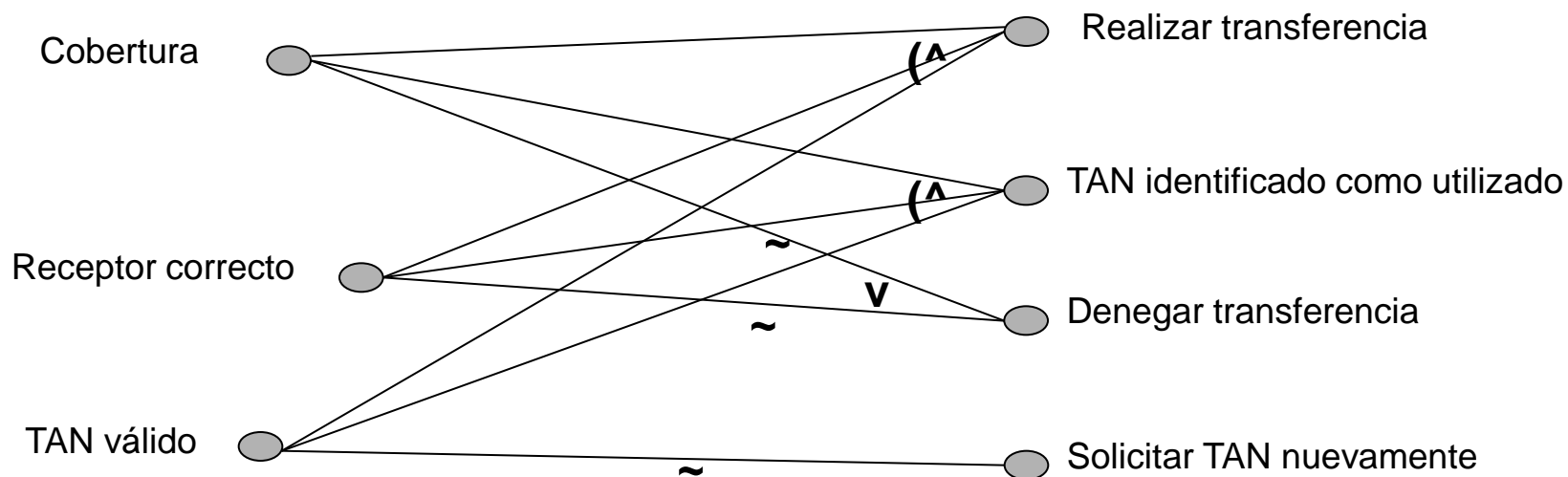
IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decision table testing”)/4

Ejemplo 5: Banca Online (“Online- Banking”) .

- El usuario se identifica a través de su numero de cuenta y PIN. Si tuviera suficiente cobertura podrá realizar una transferencia. Para poder realizar la transferencia desde introducir los datos del receptor y un TAN válido.



IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decisión table testing”)/5

Ejemplo 5: Banca Online (“Online- Banking”) .

| Variable | Clase de equivalencia | T01 T02 | T03 | T04 | T05 |
|----------------------------|---------------------------|------------|-----|-----|-----|
| Precondiciones (causas) | Suficiente cobertura | SI | NO | - | - |
| | Receptor correo | SI | - | NO | - |
| | Tan válido | SI | - | - | NO |
| Actividades (Efectos) | Realizar transferencia | SI | NO | NO | NO |
| | Marcar TAN como utilizado | SI | NO | NO | NO |
| | Denegar transferencia | NO | SI | SI | NO |
| | Solicitar TAN nuevamente | NO | NO | NO | SI |

- Cada columna de la tabla representa un caso de prueba.
- Construcción de la tabla de decisión:
 - Seleccionar un efecto.
 - Retroceder en el diagrama para identificar la causa.
 - Cada combinación de causas está representada por una columna en la tabla de decisión (un caso de prueba).
 - Combinaciones de causas idénticas, conducentes a efectos distintos, se pueden fusionar para formar un único caso de prueba.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decision table testing”)/6

Uso práctico

- La especificación está dividida en partes fáciles de gestionar, por lo que conlleva a una tabla de decisión con un tamaño práctico.
- Es difícil deducir valores límite a partir del diagrama causa-y-efecto o de la tabla de decisión.
- Es recomendable combinar casos de prueba obtenidos a partir de tablas de decisión con los obtenidos a partir de un análisis de valores límite.
- El número de causas y efectos analizados determinarán la complejidad de diagrama causa- y-efecto: para n precondiciones cuyos posibles valores pueden ser verdadero o falso, se pueden generar 2^n casos de prueba.
- Para sistemas de mayor tamaño este método sólo es controlable con el apoyo de herramientas.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de tabla de decisión (“decisión table testing”)/7

■ Beneficios.

- La identificación sistemática de combinaciones de entradas (combinaciones de causas) que no podrían ser identificados.
- Los casos de prueba son fáciles de obtener a partir de la tabla de decisión.
- Facilidad de determinar una cobertura suficiente de casos de prueba, por ejemplo, por lo menos un caso de prueba por cada columna de la tabla de decisión.
- El número de casos de prueba se puede reducir por la fusión sistemática de columnas de la tabla de decisión.

■ Desventajas.

- El establecimiento de un gran número de causas conduce a resultados complejos y extensos.
- Por lo tanto se puede concurrir en muchos errores en la aplicación de este método.
- Esto hace necesario el uso de una herramienta.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de transición de estado

■ Beneficios.

- Muchos métodos sólo tienen en cuenta el comportamiento del sistema en términos de **datos de entrada** (“input data”) y **datos de salida** (“output data”).
- No se tiene en cuenta los diferentes **estados** que pueda tomar el objeto de prueba.
 - Por ejemplo, el resultado de acciones que hubieran ocurrido en el pasado- acciones que hubieran causado que el objeto de prueba adquiriera un determinado estado interno.
- Los distintos estados que puede tomar un objeto de prueba se modelan a través de **diagramas de transición de estado**.
- El **análisis de la transición de estado** se utiliza para definir casos de prueba basados en la transición de estado.

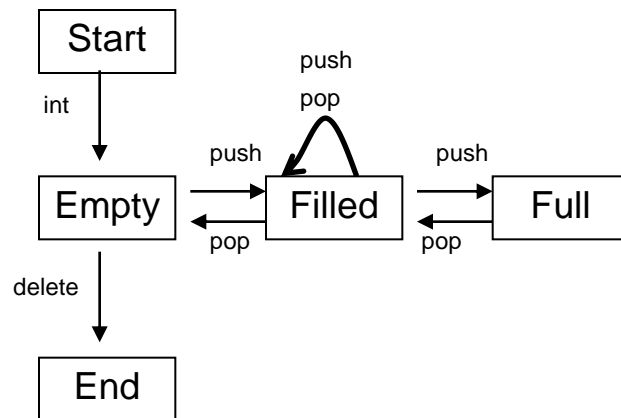
IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de transición de estado – Ejemplo 1/1

■ Ejemplo IV/03-4:

- Ejemplo simplificado de un proceso de tratamiento de una pila (“stack handler”):
 - Considerar una pila con tres entradas (elementos).
 - Una caja representa un estado de la pila, una flecha representa una transición, la descripción de la flecha indica una acción.



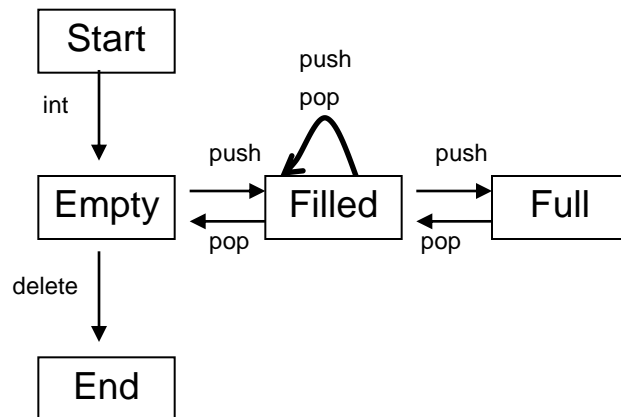
- En el estado inicial (estado= Start) la pila sólo puede estar inicializada.
- Si el estado es “**Empty**” (vacío), las acciones posibles son **push** (“**apilar**”) (que conduce al estado “**Filled**” (lleno) o **delete** (“**borrar**”) (que conduce al estado “**End**” (Fin)).
- Si la primera entrada se encuentra en la pila, es posible rellenar la segunda entrada (“**push**”- **apilar**) o liberar la primera entrada (“**pop**” - **retirar**).

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de transición de estado – Ejemplo 1/1

■ Ejemplo IV/03-4:



- Si la segunda entrada se encuentra en la pila, una entrada puede completar la tercera entrada (“push”) o se puede liberar la segunda entrada (“pop”).

- Si la tercera entrada se encuentra en la pila, sólo se puede ejecutar “pop” para liberarla.

- Desde cada estado, sólo se pueden ejecutar acciones específicas:
 - **Delete (“borrar”)** sólo está permitido si la pila se encuentra “Empty” (vacía)
 - **Push (“apilar”)** no se puede ejecutar si la pila se encuentra “Full” (completa)
 - **Pop (“retirar”)** no se puede ejecutar si la pila se encuentra “Empty” (vacía).
- La pila sólo puede tomar tres estados diferentes : “Empty” (vacía), “Filled” (llena) y “Full”(completa).

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Árbol de transición de estado / 1

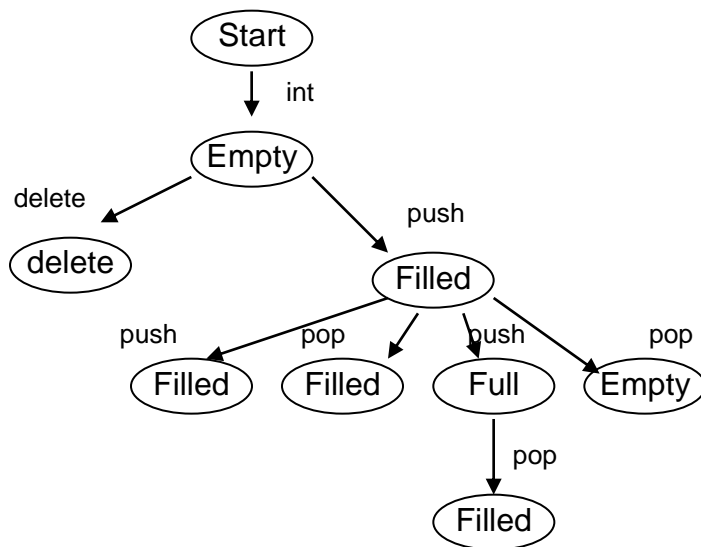
- Para determinar los casos de prueba utilizando un diagrama de transición de estado se puede construir un árbol de transición.
 - Para todos los estados, las posibles transiciones de un estado a otro se representan como ramas.
 - El estado inicial es la raíz del árbol.
 - Para cada estado que puede ser alcanzado desde el estado inicial, se crea un nodo que está conectado a la raíz a través de una rama.
 - Esta operación se repite para todos los estados sucesivos.
 - La creación de las ramas finaliza si:
 - El estado correspondiente al nodo es un estado final (hoja del árbol), o
 - El mismo nodo con el mismo estado ya es parte del árbol
- **Cada camino**, desde la raíz hasta una hoja, representa un caso de prueba para las pruebas de transición de estado

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Árbol de transición de estado/2

- El árbol de transición para la pila tiene la siguiente estructura:



| Paso 1 | Paso 2 | Paso 3 | Paso 4 | Paso 5 | Estado final |
|-----------|-----------|-----------|-----------|-----------|-----------------|
| Init | Delete | | | | Delete |
| Init | Push | Push | | | Filed |
| Init | push | Pop | | | Empty |
| Init | push | Push | Pop | | Filed |
| Init | push | Push | Push | | Full |
| Init | push | Push | Push | Pop | filled |

- La pila del ejemplo anterior tiene un tamaño de tres posiciones, por lo tanto es necesaria la ejecución de tres operaciones “push” (apilar) para llenarla de forma completa.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas de transición de estado- Resumen

- Criterio de salida de la prueba:
 - **Cada estado** debe haber sido alcanzado al menos una vez.
 - **Cada transición** debe haber sido ejecutado al menos una vez
- Beneficios/ Desventajas de éste método.
 - Un buen método de pruebas para aquellos objetos de prueba que pueden ser descritos como una **maquina de estado**.
 - Buen método de pruebas para probar clases, sólo en el caso de disponer del **ciclo de vida del objeto**.
 - Con frecuencia los estados son más bien **complejos**, es decir, es necesario un gran número de parámetros para describir el estado.
 - En estos casos, diseñar casos de prueba y analizar los resultados de las pruebas puede ser difícil y puede implicar un consumo de tiempo considerable.
 - La sola cobertura de todos los estados no garantiza una cobertura completa de las prueba.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas basada en casos de uso / 1

- Los casos de prueba se obtienen directamente a partir de los casos de uso del objeto prueba.
 - El objeto de la prueba es visto como un **sistema** reaccionando con **actores**.
 - Un caso de uso describe la interacción de todos los actores involucrados conduciendo a un resultado final por parte del sistema.
 - Todo caso de uso tiene **precondiciones** que deben ser cumplidas con el objeto de ejecutar el caso de uso (caso de prueba) de forma satisfactoria.
 - Todo caso de uso tiene **poscondiciones** que describen en sistema tras la ejecución del caso de uso (caso de prueba).
- Los casos de uso son elementos del Lenguaje Unificado de Modelado (“Unified Modeling Language”- UML*)
 - El **diagrama de casos de uso** es uno de los 13 diferentes tipos de diagramas utilizado por UML.
 - Un diagrama de casos de uso describe un comportamiento, no describe una secuencia de eventos.
 - Un diagrama de casos de uso muestra la reacción del sistema desde el **punto de vista del usuario**.

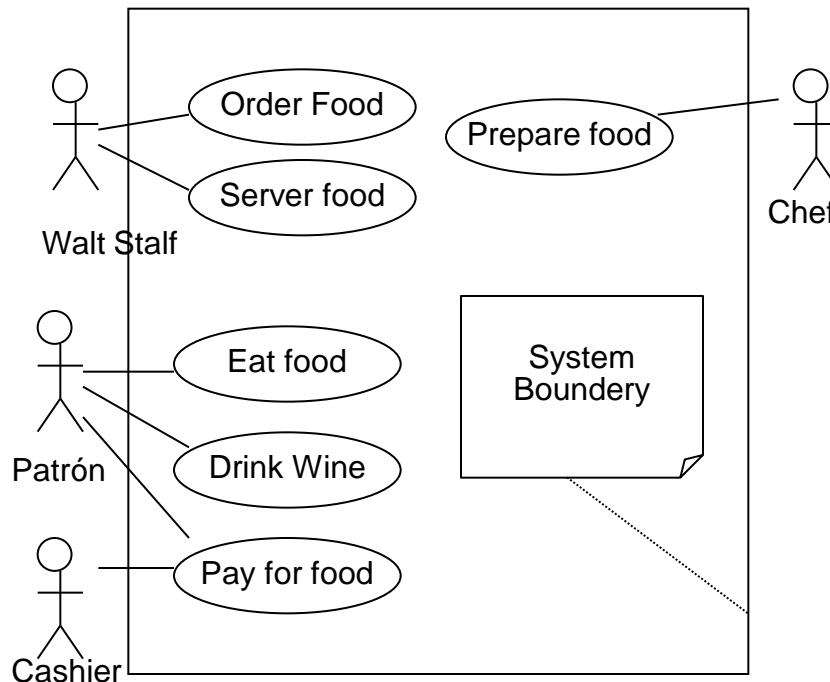
*:UML es un lenguaje de especificación no propietario para el modelado de objetos.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas basada en casos de uso / 2

- Ejemplo de un diagrama de caso de uso sencillo (fuente: Wikipedia).



El diagrama de la izquierda describe la funcionalidad de un sistema “Restaurant” sencillo.

Los casos de uso están representados por óvalos y los actores están representados por figuras de palo.

El actor “Patrón” puede comer comida (“Eat food”), pagar por la comida (“pay for Food”) o beber vino (“Drink Wine”).

El actor cocinero (“Chef”) sólo puede prepara comida. Observar que los actores “Patrón” y “Cashier” estan involucrados en el caso de uso “Pay for food” (pagar por la comida).

La caja define los límites del sistema “Restaurant”, es decir, los casos de uso representados son parte del sistema a modelar y no los actores.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas basadas en casos de uso / 3

- Cada caso de uso describe una cierta tarea (interacción usuario- sistema).
- La descripción de un caso de uso incluye, pero no está limitado a:
 - Precondiciones.
 - Resultados esperados / comportamiento del sistema
 - Poscondiciones.
- Estos elementos descriptivos también son utilizados para definir el **caso de prueba** correspondiente.
- Cada **caso de uso** puede ser utilizado como la fuente para un **caso de prueba**.
- **Cada alternativa** en el diagrama corresponde a un **caso de prueba separado**.
- Normalmente la información aportada por un caso de uso **no tiene suficiente detalle** para definir **casos de prueba**. Son necesarios datos adicionales (datos de entrada, resultados esperados) para construir / desarrollar un caso de prueba

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Pruebas basadas en casos de uso – Resumen

- Beneficios.
 - Pruebas apropiadas para **pruebas de aceptación y pruebas de sistema**, dado que cada caso de uso describe un escenario a probar
 - Pruebas apropiadas si las especificaciones del sistema se encuentran disponibles en UML.
- Desventajas
 - **Nula obtención** de casos de prueba adicionales mas allá de la información aportada por el caso de uso.
 - Por lo tanto éste método debería ser utilizado sólo en **combinación con otros métodos** de diseño sistemático de casos de prueba.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Técnicas de caja negra (“black box”) – Conclusiones generales / 1

- El objetivo principal de las pruebas de **caja negra (“black box”)** es probar la **funcionalidad del sistema**.
 - Por lo tanto el resultado de las pruebas depende de la calidad de la especificación del sistema (por ejemplo, la completitud, especificaciones faltantes o erróneas conducen a malos casos de prueba).
 - Si las especificaciones son erróneas, también serán erróneos los casos de prueba. Las pruebas se desarrollan solamente para las funciones descritas. Una especificación faltante de una funcionalidad requerida no será detectada durante las pruebas
 - Si el objeto de prueba posee funciones que no han sido especificada, éstas no serán evaluadas.
 - Tales funciones superfluas pueden causar problemas en las áreas de la estabilidad y seguridad (por ejemplo, software para un cajero automático.)

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Técnicas de caja negra (“black box”) – Conclusiones generales / 2

- A pesar de estas desventajas, las **pruebas funcionales** constituyen la actividad de pruebas **más importantes**
 - Los **métodos de caja negra (black box)** siempre son utilizados en el proceso de pruebas.
 - Las desventajas pueden ser compensadas con el uso de **métodos adicionales** de diseño de casos de prueba, por ejemplo, pruebas de caja blanca o pruebas basadas en la experiencia.

IV – Técnicas de diseño de pruebas

03 – Técnicas de caja negra (“black box”)

Técnicas de caja negra (“black box”) – Resumen

- Métodos de caja negra (“black box”)
 - Partición equivalencia o clase de equivalencia.
 - Análisis de valores límite.
 - Tablas de decisión & gráficos causa- y- efecto.
 - Pruebas de transición de estado.
 - Pruebas de caso de uso.
- Las pruebas de caja negra (black box) verifican funciones especificadas: si las funciones **no son especificadas**, éstas **no son probadas**.
- El **código adicional** (es decir que no debería estar) no puede ser detectado utilizando pruebas de caja negra (black box)

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Técnicas basadas en la estructura de caja blanca (“white box”)

- En el presente apartado se explicarán en detalle las siguientes técnicas de caja blanca (“white box”)
 - Prueba de sentencia y cobertura
 - Prueba de rama (“branch”) y cobertura.
 - Prueba de condición y cobertura
 - Prueba de camino (“path”) y cobertura .
- Observación:
 - Estas técnicas representan las técnicas de prueba dinámicas más importantes y utilizadas de forma más frecuente.
Estas técnicas están relacionadas con las técnicas de análisis estático descritas anteriormente
- Otras técnicas de caja blanca son las siguientes (no limitada a la enumeración):
 - LCSAJ (Linear Code Sequence And Jump): Secuencia lineal de código y salto.
 - Técnicas basadas en el flujo de datos.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Técnicas de caja blanca (“white box”) - herramientas / 1

- Durante las pruebas de caja blanca, el programa objeto de las pruebas es ejecutado de la misma forma que las pruebas de caja negra. Ambas categorías (caja blanca y caja negra) conforman las **pruebas dinámicas**.
 - La teoría establece que todas las partes de un programa debería ser **ejecutado por lo menos una vez**.
- El grado de cobertura de un programa se mide con el uso de herramientas (por ejemplo, analizadores de cobertura):
 - La **instrumentación del código** se lleva a cabo con el objeto de contar la ejecución de caminos, es decir se insertan contadores en el código del programa del objeto de prueba.
 - Estos **contadores** son **inicializados** en **cero**, cada ejecución del camino específico incrementará el contador correspondientemente.
 - Los contadores que mantienen el valor **cero** tras las pruebas indican las partes del programa que aún **no** han sido **ejecutadas**.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Técnicas de caja blanca (“white box”) - herramientas / 2

- Las técnicas de caja blanca requieren el apoyo de herramientas en muchas áreas, a saber:
 - Especificación de caso de pruebas
 - Generación automática del diagrama del flujo de control a partir del código fuente del programa.
 - Ejecución de la prueba.
 - Herramientas para monitorizar y controlar el flujo del programa dentro del objeto de la prueba.
- El soporte de herramienta asegura la calidad de las pruebas e incrementa su eficiencia.
 - Dada la complejidad de las mediciones necesarias para las pruebas de caja blanca, la ejecución manual de pruebas implica:
 - Consumo de tiempo, consumo de recursos.
 - Dificultad en la implementación y propensión a errores.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Principales tipos de cobertura

- Cobertura de sentencia (“statement coverage”)
 - Porcentaje de sentencias ejecutables que han sido practicadas por los casos de prueba.
 - También puede ser aplicado a módulos, clases, elementos de un menú, etc.
- Cobertura de decisión (= cobertura de rama) (“decision coverage = branch coverage”)
 - Procentaje de resultados de decisión que han sido practicados por los casos de pruebas
- Cobertura de camino (“path coverage”)
 - Porcentaje de camino de ejecución que han sido practicados por casos de pruebas.
- Cobertura de condición (“condition coverage”).
 - Porcentaje de todos los resultados individuales de condición que afectan la forma independiente al resultado de una decisión que ha sido practicada por los casos de prueba.
 - La cobertura de condición presenta distintos grados, por ejemplo, cobertura de condición simple, cobertura de condición múltiple y cobertura de condición múltiple mínima.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia (“Statement coverage”)

- El foco de la atención es la sentencia del código de un programa.
 - ¿Qué casos de prueba son necesarios con el objeto de ejecutar todas (o un porcentaje determinado) las sentencias del código existentes?
- La base de este análisis es el diagrama de flujo de control.
 - Todas las instrucciones están representadas por nodos y el flujo de control entre instrucciones está representado por una arista (flecha).
 - Las instrucciones múltiples se combinan en un nodo independiente si solamente pueden ser ejecutadas en una secuencia particular
- El objetivo de la prueba (criterio de salida) es lograr la cobertura de un porcentaje específico de todas las sentencias, denominado cobertura de sentencia (Co cobertura de código – “code coverage”)

$$\text{Cobertura de sentencia (Co)} = \frac{\text{Número de sentencias ejecutadas}}{\text{Número total de sentencias}} * 100\%$$

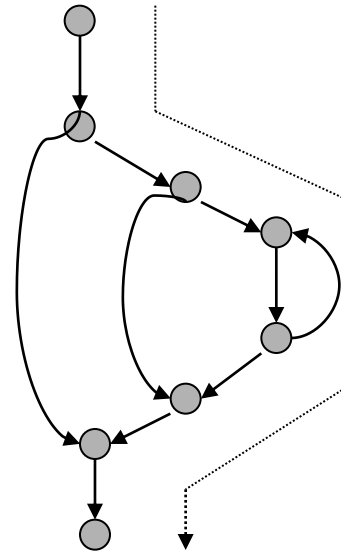
IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia (“Statement coverage”)

- Ejemplo 1 / 1
- Ejemplo IV/02-1
 - Se evalúa el siguiente segmento de código de un programa, que está representado por el diagrama de flujo de control (imagen a la derecha).

```
If (i>0) {  
    j=f (i);  
    if (j>10){  
        for (k=i; k >10; k -- ){  
            .....  
        }  
    }  
}
```



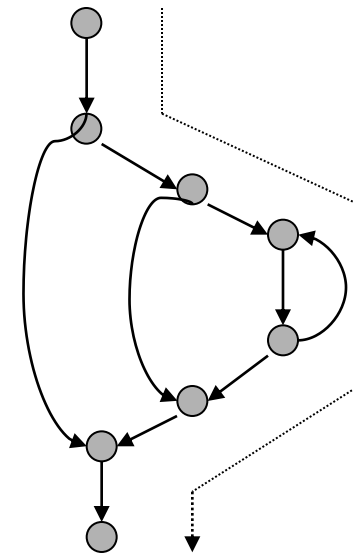
IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia (“Statement coverage”) -

Ejemplo 1/2

- Considerar el programa representado por el diagrama de flujo de control (imagen a al derecha)
 - Contiene dos sentencias “if” y un bucle “do-while” dentro de la segunda sentencia “if”.
- Hay tres “caminos” diferentes a través del segmento de programa.
 - La primera sentencia “if” permite dos direcciones.
 - La dirección de la derecha de la primera sentencia “if” se divide nuevamente a partir de una segunda sentencia “if”.
- Todas las sentencias de este programa pueden ser alcanzadas haciendo uso de este camino a la derecha.
 - Un solo caso de prueba será suficiente para alcanzar el 100% de cobertura de sentencia.



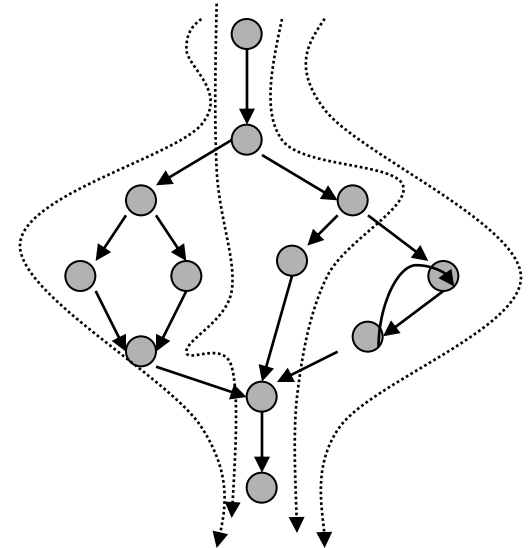
IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia (“Statement coverage”)-

Ejemplo 2

- **Ejemplo IV/02-2.**
- En este ejemplo el diagrama es ligeramente más complejo:
 - El programa contiene las sentencias “if” y un bucle (dentro de una sentencia “if”)
- Cuatro caminos diferentes conducen a través de este segmento de programa.
 - La primera sentencia “if” permite dos direcciones.
 - En cada rama de la sentencia “if” otra sentencia “if” permite nuevamente dos direcciones diferentes.
- Utilizando sólo un caso de prueba, un máximo de 7 de 12 sentencias pueden ser cubiertas, esto resulta en un valor de $C0 = 58,33\%$.
 - Para una cobertura de sentencia del 100% hacen falta cuatro casos de prueba.



IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia (“Statement coverage”)- Conclusiones generales

- La medición de la cobertura se realiza con el uso de herramientas diseñadas de forma específica.
 - Estas herramientas se denominan Herramientas de análisis de Cobertura (“Coverage Analysis Tools”) o Analizadores de Cobertura (“Coverage Analyzers”).
- Beneficios / desventajas de este método.
 - El código muerto, es decir, código constituido por sentencias que nunca se ejecutan, será detectado.
 - Si hay código muerto en el programa, no se podrá lograra una cobertura del 100%.
 - Instrucciones faltantes es decir, código que es necesario con el objeto de cumplir con la especificación, no puede ser detectado.
 - Las pruebas se desarrollan solamente respecto de sentencias ejecutadas: ¿todo el código puede ser alcanzado/ ejecutado?
 - El código faltante no puede ser detectado utilizando técnicas de caja blanca (análisis de cobertura)

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de decisión (“decisión coverage”)

- En lugar de las sentencias, la cobertura de decisión se centra en el flujo de control en un segmento de programa (no los nodos sino las aristas del diagrama de flujo de control)
 - Todas las aristas del diagrama de flujo de control tiene que ser cubiertas al lo menos una vez.
 - ¿Qué casos de prueba son necesarios para cubrir cada arista del diagrama de flujo de control a l menos una vez?
- El propósito de este prueba (criterio de salida) es lograr la cobertura de un porcentaje específico de todas las decisiones, denominado cobertura de decisión (C₁ cobertura de código- “code coverage”).

$$\text{Cobertura de decisión (C}_1\text{)} = \frac{\text{Número de decisiones ejecutadas}}{\text{Número total de decisiones}} * 100\%$$

Sinónimo de:

$$\text{Cobertura de rama (C}_1\text{)} = \frac{\text{Número de ramas cubiertas}}{\text{Número total de ramas}} * 100\%$$

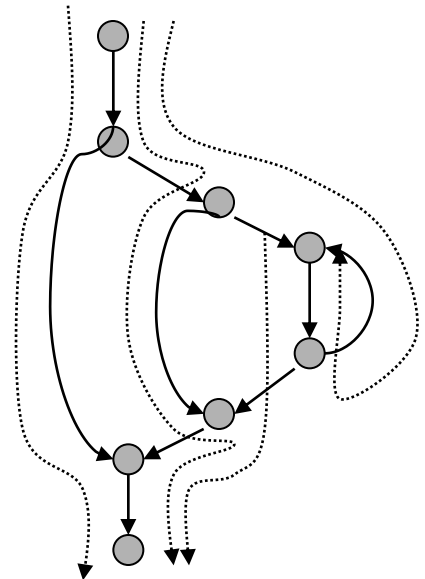
IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de decisión (“decision coverage”) -

Ejemplo 1

- **Ejemplo IV/ 02-3.**
- El diagrama de flujo de control (imagen a la derecha) representa el segmento un programa objeto de la evaluación
- Tres caminos diferentes conducen a través del diagrama de este segmento de programa.
 - La primera sentencia “if” conducen a dos direcciones diferentes.
 - Un camino de la primera sentencia “if” se divide nuevamente en dos caminos diferentes, uno de los cuales contiene un bucle.
 - Solamente se puede alcanzar las aristas a través de una combinación de los tres caminos posibles.
 - Son necesarios tres casos de prueba para alcanzar una cobertura de decisión del 100%.
 - Utilizando solamente las dos direcciones de la derecha pueden ser cubiertas nueve de las diez aristas ($C_1 = 90\%$).

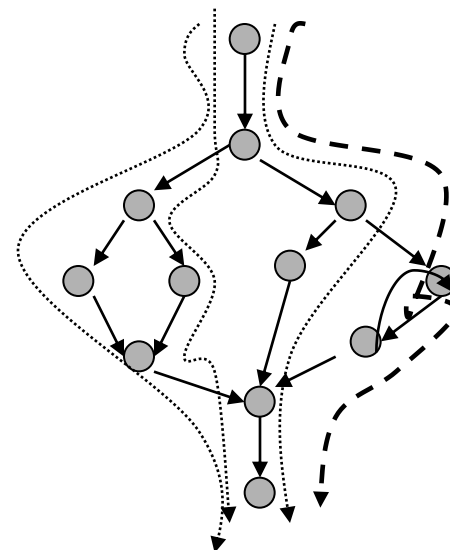


IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de decisión (“decision coverage”) - Ejemplo 2

- **Ejemplo IV /02- 4.**
- En este ejemplo el diagrama es ligeramente más complejo.
- Cuatro “caminos” diferentes conducen a través del segmento de programa.
 - La primera sentencia “if” permite dos direcciones.
 - En ambas ramas de la sentencia “if” otra sentencia “if” permite nuevamente dos direcciones diferentes.
 - En este ejemplo, el bucle no se cuenta como una decisión adicional.
- Utilizando sólo un caso de prueba, pueden ser cubiertas 7 de 15 aristas. Esto resulta en un valor de $C_1 = 46,67\%$.
 - Son necesarios cuatro casos de prueba para lograr una cobertura de decisión de un 100%.
 - ¡en este ejemplo, son necesarios el mismo conjunto de casos de prueba para lograr una cobertura de sentencia del 100%!



IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de decisión (“decisión coverage”)- Conclusiones generales

- Lograr una cobertura de decisión del 100% requiere, al menos, los mismos casos de prueba que requiere la cobertura de sentencia- más en la mayoría de los casos.
 - Una cobertura de decisión del 100% siempre incluye una cobertura de sentencia del 100%.
- La mayoría de las aristas son cubiertas en múltiples ocasiones.
- Desventajas.
 - No se pueden detectar sentencias faltantes.
 - No es suficiente para probar condiciones complejas.
 - No es suficiente para probar bucles de forma extensiva.
 - No se consideran las dependencias entre bucles.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de sentencia y cobertura de decisión

- Ambos métodos se refieren a caminos a través de diagrama de flujo de control.
 - Difieren en la cantidad de casos de prueba necesarios para lograr el 100% de cobertura.
- Sólo se considera el resultado final de una condición, a pesar de que las condición resultante puede estar constituida por múltiples condiciones atómicas.
 - La condición ***if ((a>2) &&(b<6))*** sólo puede ser verdadero o falsa.
 - El camino (del programa) a ejecutar depende solamente del resultado final de la condición combinada.
 - Aquellos fallos debidos a una implementación errónea de las partes de una decisión combinada pueden no ser detectados.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de condición

- Se tiene en cuenta la complejidad de una condición que esté constituida por múltiples condiciones atómicas
 - Una condición atómica no puede ser dividida en sentencias condicionales mas pequeñas.
- Éste método tiene por objetivo detectar defectos que resulten de la implementación de condiciones múltiples (condiciones combinadas).
 - Las condiciones múltiples están constituidas por condiciones atómicas, que se combinan con el uso de operadores lógicos como:
 - Ejemplo: ((a>2) OR (b<6))
 - Las condiciones atómicas no tienen operadores lógico, sólo contienen operadores relacionales y el operador NOT (=, >, <, etc.)
- Hay tres tipos de cobertura de condición.
 - Cobertura de condición simple (“simple condition coverage”).
 - Cobertura de condición múltiple (“multiple condition coverage”).
 - Mínima cobertura de condición múltiple (“minimum multiple condition coverage”).

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de condición simple (“simple condition coverage”)

- Cada sub.-condición atómica de una sentencia condicional combinada tiene que tomar, al menos una vez, los valores lógicos verdadero (“true”) así como falso (“false”).

Ejemplo IV/02-6

Considerar la siguiente condición.

$a > 2$ OR $b < 6$

Los casos de prueba para la cobertura de condición simple podrán ser por ejemplo

| | | |
|-------------|-------------|------------------------------|
| a=6 (true) | b=9 (false) | $a > 2$ OR $b < 6$ (true) |
| a=1 (false) | b=2 (true) | $a > 2$ OR $b < 6$ (true) |

- Este ejemplo se utiliza para explicar la cobertura de condición utilizando una expresión como una condición múltiple.
- Con sólo dos casos de prueba se puede lograr una cobertura de condición simple.
 - Cada sub-condición ha tomado los valores verdadero (“true”) y falso (“false”).
- Sin embargo, el resultado combinado es verdadero (“true”) en ambos casos.
 - True OR false = true
 - False OR true = true

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de condición simple (“simple condition coverage”)

- Cada sub-condición atómica de una sentencia condicional combinada tiene que tomar, al menos una vez, los valores lógicos verdadero (“true”) así como falso (“false”).

Ejemplo IV/02-6

Considerar la siguiente condición.

$$a > 2 \text{ OR } b < 6$$

Los casos de prueba para la cobertura de condición múltiple podrían ser por ejemplo

| | | |
|-------------|-------------|--------------------|
| a=6 (true) | b=9 (false) | a>2 OR b<6 (true) |
| a=6 (true) | b=2 (true) | a>2 OR b<6 (true) |
| a=1 (false) | b=2 (true) | a>2 OR b<6 (true) |
| a=1 (false) | b=9 (false) | a>2 OR b<6 (false) |

- Este ejemplo se utiliza para explicar la cobertura de condición utilizando una expresión como una condición múltiple.
- Con cuatro casos de prueba se puede lograr una cobertura de condición múltiple
 - Se han creado las combinaciones de los valores verdadero (“true”) y falso (“false”).
 - Se han logrado todos los posibles resultados de la condición múltiple.
- El número de casos de prueba se incrementa de forma exponencial:
 - n = número de condiciones atómicas.
 - 2^n = número de casos de prueba.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de condición simple (“simple condition coverage”)

- **Todas las condiciones que puedan ser creadas** utilizando los resultados lógicos de cada sub-condición deben ser parte de las pruebas, sólo si el cambio del resultado de una sub-condición cambia el resultado de la condición combinada.

Ejemplo IV/ 02-6

Considerar la siguiente condición.

$a > 2$ OR $b < 6$

Los casos de prueba para la cobertura de condición múltiple podrían ser por ejemplo

| | | |
|-------------|-------------|----------------------------|
| a=6 (true) | b=9 (false) | $a > 2$ OR $b < 6$ (true) |
| a=6 (true) | b=2 (true) | $a > 2$ OR $b < 6$ (true) |
| a=1 (false) | b=2 (true) | $a > 2$ OR $b < 6$ (true) |
| a=1 (false) | b=9 (false) | $a > 2$ OR $b < 6$ (false) |

- Este ejemplo se utiliza para explicar la cobertura de condición utilizando una expresión con una condición múltiple.
- Los cambios de una sub-condición cambian el resultado global para tres de cuatro casos de prueba.
 - Sólo para el caso nº2 (true OR true = true) el cambio en la sub-condición no resultará en un cambio en la condición global. ¡este caso de prueba puede ser omitido!.
- El número de casos de prueba se puede reducir a un valor entre $n+1$ y $2n$

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de condición (“condition coverage”) – conclusiones generales

- La **cobertura de condición simple** es un instrumento débil para probar condiciones múltiples.
- La **cobertura de condición múltiple** es un método mucho mejor.
 - Asegura cobertura de sentencia y decisión.
 - Sin embargo tiene como resultado un alto número de casos de prueba: 2^n .
 - La ejecución de algunas combinaciones no es posible.
 - Por ejemplo “ $x > 5$ AND $x < 10$ ” ambas sub-condiciones no pueden ser falsas al mismo tiempo.
- La **mínima cobertura de condición múltiple** es incluso mejor, debido a:.
 - Reduce el número de casos de prueba (de $(n+1)$ a $(2n)$).
 - Las coberturas de sentencia y decisión también son cubiertas.
 - Tiene en cuenta la complejidad de las sentencias de decisión.
- **Todas las decisiones complejas deben ser probadas – la mínima cobertura de condición múltiple es adecuada para logara este objetivo.**

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de camino (“path coverage”) / 1

- La cobertura de camino se centra en la ejecución de todos los posibles caminos a través de un programa.
 - Un camino es una combinación de segmentos de programa (en el diagrama de flujo de control: una secuencia alternante de nodos y aristas).
 - Para cobertura de cisión, un solo camino a través de un bucle es suficiente. Para la cobertura de camino hay casos de prueba adicionales.
 - Un caso de prueba no entrante al bucle.
 - Un caso de prueba adicional para cada número de ejecuciones del bucle.
- Esto puede conducir a un número muy alto de casos de prueba.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de camino (“path coverage”) / 2

- El foco de análisis de cobertura es el diagrama de flujo de control.
 - Las sentencias son nodos.
 - El flujo de control está representado por las aristas.
 - Cada camino es una vía única desde el inicio al fin del diagrama de flujo de control.
- El objetivo de esta prueba (criterio de salida) es alcanzar un porcentaje definido de cobertura de camino.

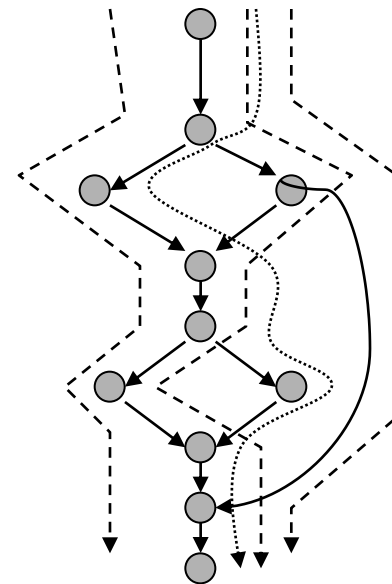
$$\text{Cobertura de camino} = \frac{\text{Número de caminos cubiertos}}{\text{Número total de caminos}} * 100\%$$

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de camino (“path coverage”)- Ejemplo 1

- **Ejemplo IV /02-5:**
- El diagrama de flujo de control de la imagen a la derecha, representa el segmento de programa a ser evaluado. Contiene dos sentencias “if” y un bucle en el interior de la segunda sentencia “if”.
- Tres caminos diferentes conducen a través del diagrama de este segmento de programa logran una cobertura de decisión completa.
- Sin embargo, pueden ser ejecutados cinco posibles caminos distintos.
 - Son necesarios cinco casos de pruebas para lograr un 100% de cobertura de camino.
 - Sólo dos son necesarios para un 100% de cobertura C0, tres son necesarios para un 100% de cobertura C1

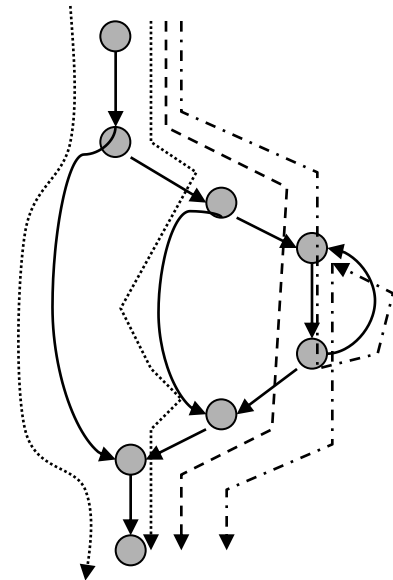


IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de camino (“path coverage”)- Ejemplo 1

- **Ejemplo IV/02-5:**
- El diagrama de flujo de control de la imagen a la derecha, representa el segmento de programa a ser evaluado. Contiene dos sentencias “if” y un bucle en el interior de la segunda sentencia “if”.
- **Tres caminos diferentes** conducen a través del diagrama de este segmento de programa logran una cobertura de decisión completa.
- Si el bucle se ejecuta dos veces son posibles **cuatro caminos diferentes**.
- **Cada incremento en el contador del bucle** añade un nuevo caso de prueba.



IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Cobertura de camino (“path coverage”) – Conclusiones generales

- El 100% de cobertura de camino sólo se puede lograr en programas muy simples.
 - Un solo bucle puede conducir a una explosión de casos de prueba dado que , todo número posible de ejecuciones de un bucle constituye un nuevo caso de prueba.
 - Teóricamente es posible un número indefinido de caminos.
- Cobertura de camino es más exhaustiva que la cobertura de sentencia y de decisión.
 - Cada posible camino a través del programa es ejecutado.
- 100% de cobertura de camino incluye 100% de cobertura de decisión, que a su vez incluye 100% de cobertura de sentencia.

IV – Técnicas de diseño de pruebas

04 – Técnicas de caja blanca (“white box”)

Resumen

- Los métodos de caja blanca y caja negra son métodos dinámicos, el objeto de prueba es ejecutado durante las pruebas.
- El método de caja blanca (white - box) comprende:
 - **Cobertura de sentencia (“Statement coverage”).**
 - **Cobertura de decisión (“decision coverage”).**
 - **Cobertura de camino (“path coverage”).**
 - **Cobertura de condición (“condition coverage”)** (simple (“single”), múltiple (“multiple”), mínimo múltiple (“minimu, multiple”).
- Sólo se puede probar código existente. Habiendo funciones faltantes, este hecho no puede ser detectado. Sin embargo el **código muerto o superfluo** puede ser detectado con las pruebas de caja blanca.
- Principalmente, los métodos de caja blanca son utilizados en pruebas de bajo nivel como **pruebas de componente** o pruebas de integración.
- Los métodos difieren en la intensidad de las pruebas (profundidad de la prueba).
 - Dependiendo del método, el número de caso de prueba es distinto.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Fundamentos / 1

Definición de técnicas basadas en la experiencia

- Práctica para la creación de casos de prueba sin un claro enfoque metodológico, basada en la intuición y experiencia del tester.
- Los casos de prueba se basan en la intuición y experiencia.
 - ¿Dónde se han acumulado defectos en el pasado?
 - ¿Dónde falla el software con más frecuencia ?
- Las pruebas basadas en la experiencia también se denominan pruebas intuitivas (“intuitive testing”) e incluye: predicción de errores (“error guessing”) (pruebas orientadas a puntos débiles) y pruebas exploratorias (pruebas iterativas basadas en el conocimiento adquirido respecto del sistema).

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Fundamentos / 2

- Principalmente aplicadas con el objeto de complementar otros caso de prueba generados con un mayor formalismo.
 - No cumple con los criterios de un proceso de pruebas sistemático.
 - Frecuentemente produce casos de prueba adicionales que no podrían ser creados con otras prácticas, por ejemplo:
 - Pruebas de un año bisiesto posterior al año 2060.
 - Problemas conocidos del pasado.
 - Conjuntos vacíos en valores e entrada.
 - Una aplicación similar ha tenido errores en estas circunstancia.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Diseño intuitivo de casos de prueba – posibles fuentes

- Resultados de pruebas y experiencia práctica con sistemas similares
 - Posiblemente predecesor del software u otro sistema con funcionalidad similar.
- Experiencia del usuario.
 - Intercambio de experiencia con el sistema como usuario.
- Enfoque del despliegue.
 - ¿Qué partes del sistema serán utilizados con mayor frecuencia?
- Problemas de desarrollo.
 - ¿Hay algún punto débil como resultado de dificultades en el proceso de desarrollo?

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Predicción de errores (“error guessing”) en practica

- Lista de comprobación de errores
 - Enumerar posibles errores.
 - Factores ponderados dependientes del riesgo y probabilidad de ocurrencia.
- Diseño de caso de prueba.
 - Creación de casos de prueba dirigidos a producir los errores de la lista.
 - Asignar prioridades a los casos de prueba considerando el valor de su riesgo.
- Actualizar la lista de errores durante las pruebas.
 - Procedimiento iterativo.
 - Es útil una colección estructurada de experiencia cuando se repite el procedimiento en futuros proyectos.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Pruebas exploratorias (“exploratory testing”)

- Es un procedimiento de diseño de casos de prueba especialmente apropiado cuando la información base se encuentra u poco estructurada.
- También es útil cuando el tiempo disponible para pruebas es escaso.
- Procedimiento:
 - Revisar las partes constituyentes (individuales/identificables) del objeto de prueba.
 - Ejecutar un número reducido de casos de prueba, exclusivamente sobre aquellas partes que deben ser probadas, aplicando predicción de errores (“error guessing”).
 - Analizar los resultados, desarrollar un modelo preliminar (“rough model”) de cómo funciona el objeto de prueba.
 - Iteración: diseñar nuevos objetos de prueba aplicando el conocimiento adquirido recientemente.
 - Por lo tanto concentrándose en las áreas relevantes y explorando características adicionales del objeto de prueba.
 - Herramientas de captura pueden ser útiles para registrar las actividades de prueba.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Pruebas exploratorias (“exploratory testing”) - Principios

- Seleccionar objetos pequeños y/o concentrarse en aspectos particulares del objeto de pruebas.
 - Una iteración unitaria no debería llevar mas de 2 horas.
- Los resultados de una iteración constituyen la base de información para la siguiente iteración.
 - Se obtienen casos de prueba adicionales a partir de la situación particular de la prueba.
- El modelado tiene lugar durante el proceso de pruebas.
 - Se genera un modelo del objeto de prueba durante las pruebas.
 - Un objetivo de las pruebas es el refinamiento continuo del modelo.
- Preparación de pruebas adicionales.
 - Con esto, el conocimiento puede ser adquirido para apoyar la elección apropiada de métodos de diseño de casos de prueba.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Diseño intuitivo de casos de prueba vs. diseño sistemático de casos de prueba.

- Diseño intuitivo de casos de prueba es un buen complemento a los enfoques sistemáticos.
 - Aún debe ser tratado como una actividad complementaria.
 - No puede dar constancia de completitud – el número de casos de prueba puede variar de forma considerable.
- Las pruebas son ejecutadas de la misma manera que lo son los casos de prueba definidos de forma sistemática.
- La diferencia es la forma en la cual los casos de prueba han sido diseñados/identificados.
- A través de pruebas intuitivas se pueden detectar defectos que no podrán ser detectados a través de métodos sistemáticos de prueba.

IV – Técnicas de diseño de pruebas

05- Técnicas basadas en la experiencia

Resumen

- Las técnicas basadas en la experiencia complementan las técnicas sistemáticas para determinar casos de prueba.
- Las técnicas basadas en la experiencia dependen en gran medida de la habilidad individual del tester.
- La predicción de errores y las pruebas exploratorias son dos de las técnicas mas ampliamente utilizadas de pruebas basadas en la experiencia.

IV – Técnicas de diseño de pruebas

06- Selección de las técnicas de pruebas

Criterios para seleccionar el diseño apropiado de caso de prueba / 1

- Estado de información respecto del objeto de prueba.
 - ¿Se pueden realizar pruebas de caja blanca de alguna manera?
 - ¿Hay suficiente material de especificación para definir pruebas de caja negra, o son necesarias pruebas exploratorias para comenzar?
- Objetivos de prueba predominante.
 - ¿las pruebas funcionales han sido solicitadas de forma explícita?
 - ¿Qué pruebas no funcionales son necesarias?
 - ¿Son necesarias pruebas estructurales para lograr los objetivos del proceso de prueba?
- Riesgos
 - ¿Se espera un daño/perjuicio serio proveniente de defectos ocultos?
 - ¿Es muy alta la frecuencia de uso del objeto de prueba?
 - ¿Hay algún estándar legal o contractual, respecto a la ejecución de pruebas y cobertura de pruebas, que deban ser cumplidos?

IV – Técnicas de diseño de pruebas

06- Selección de las técnicas de pruebas

Criterios para seleccionar el diseño apropiado de caso de prueba / 2

- Precondiciones del **proyecto**
 - ¿Cuánto **tiempo** y quien está planificado/asignado para las pruebas?
 - ¿Cuál es el **riesgo** de que le proceso de pruebas no sea finalizado según la planificación?
 - ¿Qué **método de desarrollo** es utilizado?
 - ¿Cuáles son los puntos débiles del **proceso** asociado al proyecto?
- Características del **objeto de la prueba**.
 - ¿Qué **posibilidades** para ser probado ofrece el objeto de prueba?
 - ¿Cuál es la **disponibilidad** del objeto de prueba?
- Requisitos **contractuales** y del cliente.
 - ¿ha habido algún **acuerdo específico** entre el cliente / iniciador del proyecto respecto de los procedimientos de prueba?
 - ¿Qué **documentos** deben ser entregados en el momento de despliegue del sistema?

IV – Técnicas de diseño de pruebas

06- Selección de las técnicas de pruebas

Criterios para seleccionar el diseño apropiado de caso de prueba / 3

- Buena práctica
 - ¿Qué enfoques han demostrado ser apropiados para estructuras similares?
 - ¿qué experiencias han sido adquiridas con qué enfoques en el pasado?
- Niveles de prueba
 - ¿En qué niveles de prueba se deben realizar pruebas?
- ¿Se deben aplicar criterios adicionales dependiendo de la situación específica!

IV – Técnicas de diseño de pruebas

06- Selección de las técnicas de pruebas

Intereses distintos causan enfoques diferentes en el diseño de pruebas

- Interés del jefe de proyecto
 - Desarrollar software de la calidad requerida.
 - Cumplir las restricciones de tiempo y presupuesto.
- Intereses del cliente / iniciador del proyecto:
 - Recibir software de la mas alta calidad (funcionalidad, fiabilidad, usabilidad, eficiencia, portabilidad y mantenibilidad).
 - Cumplir las restricciones de tiempo y presupuesto.
- Intereses del jefe de pruebas:
 - Pruebas suficientes e intensivas / despliegue adecuado de las técnicas necesarias desde el punto de vista del proceso de pruebas.
 - Evaluar/valorar el nivel de calidad alcanzado por el proyecto.
 - Asignar y utilizar los recursos planificados para las pruebas de forma óptima.

IV – Técnicas de diseño de pruebas

06- Selección de las técnicas de pruebas

Resumen

- Criterio para seleccionar el enfoque apropiado de diseño de casos de prueba:
 - Fuente de pruebas (Fuente de la información respecto de los objetos de prueba).
 - Objetivos del proceso de pruebas (Qué conclusiones se deben obtener a través de las pruebas).
 - Aspectos asociados al riesgo.
 - Estructura del proyecto / precondiciones.
 - Requisitos contractuales / cliente.

V – Gestión de pruebas

00- Agenda

Capítulo V – Gestión de pruebas (“Test management”)

- V/01 Organización del proceso de pruebas (“Test organization”).
- V/02 Planificación y estimación del proceso de pruebas (“Test planning and estimation”).
- V/03 Seguimiento y control del estado de las pruebas (“Test progress monitoring and control”).
- V/04 Gestión de la configuración (“Configuration management”).
- V/05 Riesgo y proceso de pruebas (“Risk and testing”).
- V/06 Gestión de incidencias (“Incident management”).

V – Gestión de pruebas

01- Organización del proceso de pruebas

La gestión de pruebas como parte del proceso de pruebas

- La gestión de pruebas es la gestión de proyecto de proyectos de pruebas
- El proceso de pruebas es una actividad que cumple por completo el proceso de desarrollo software.
- Las actividades propias de la gestión de pruebas son necesarias a lo largo de todo el proceso de pruebas.

Actividad

Concepción de pruebas
Planificación de pruebas
Control de pruebas
Pruebas de aceptación

Producto resultado del trabajo

Plan de pruebas (estático)
Plan de pruebas (dinámico)
Informe de estado, acción de control
Versión del producto software

V – Gestión de pruebas

01- Organización del proceso de pruebas

Los equipos de prueba deberían ser independientes

Ventajas

- Imparcialidad, no hay vinculación personal con el objeto de prueba.
- Se pueden cuestionar hechos respecto de la base de pruebas (“Test basis”) y verificar las suposiciones hechas durante al diseño de pruebas.

Desventajas.

- Aumenta el esfuerzo dedicado a la comunicación, presentación de conflictos del tipo “tener la última palabra”.

Otras formas de conformar equipos de prueba

- Los testers también son miembros del equipo de desarrollo.
- Los testers también son miembros del equipo del proyecto o estructura de la organización.
- Especialistas para tareas específicas.
- Equipos de prueba externos.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Perfiles del personal de pruebas

- El proceso de pruebas requiere personas con una amplia variedad de competencias y cualificaciones.
- Se explicaran en detalle los siguientes roles asociados al proceso de pruebas:
 - Jefe de pruebas o director de pruebas (“Test manager”).
 - Diseñador de pruebas (“Test designer”).
 - Ingeniero de automatización de pruebas (“Test automation engineer”).
 - Administrador de pruebas (“Test administrator”)/ administrador del sistema de pruebas (“Test system administrator”).
 - Tester
 - Experto técnico (“Technical expert”).
- Nota
 - Se pueden especificar roles adicionales, por ejemplo administrador de base de datos, tester de carga.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Jefe de pruebas (“Test leader”) {También director de pruebas (“Test manager”) o coordinador de pruebas (“Test coordinator”)}

- Planifica, realiza el seguimiento y control del proyecto de pruebas.
- Competencias especiales necesarias:
 - Gestión de pruebas y calidad software.
 - Planificación y control de pruebas.
 - Experiencia como jefe de proyecto.
 - Habilidades de gestor.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Diseñador de pruebas (“Test desinger”)

- Diseña los casos de prueba necesarios y establece el orden en el cual tendrá lugar la ejecución de los casos de prueba.

- Competencias especiales necesarias en el área de :
 - Conocimiento de desarrollo y pruebas.
 - Conocimiento de ingeniería de software.
 - Conocimiento respecto de especificaciones técnicas.
 - Conocimientos de requisitos funcionales.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Ingeniero de automatización de pruebas (“Test automation engineer”)

- Evalúa las posibilidades de la automatización de pruebas y las implementa.
- Competencias especiales necesarias:
 - Experiencia como tester.
 - Conocimiento técnico (“know-how”) en el ámbito de diseño y automatización de pruebas.
 - Conocimientos de programación.
 - Amplios conocimientos en el uso de las herramientas utilizadas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Administrador del sistema de pruebas (“Test system administrator”)

- Prepara y opera el entorno de pruebas.
 - Es responsable de cumplir los requisitos del sistema de pruebas.
- Competencias especiales necesarias:
 - Administrador del sistema (o acceso al administrador del sistema).
 - Conocimiento de herramientas desarrollo y pruebas.
 - Sistemas de bases de datos, si aplica.
 - Redes, si aplica.
 - Instalación de operación de software del sistema (por ejemplo sistemas operativos).

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tester de Software (“Software Tester”)

- Ejecuta las pruebas de acuerdo con la especificación de casos de prueba.
- Competencias especiales necesarias:
 - Conocimiento básico del software.
 - Conocimiento básico de pruebas.
 - Operación y uso de herramientas de pruebas.
 - Experiencia en la ejecución de pruebas.
 - Conocimiento respecto de los objetos de prueba.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Experto técnico (“Technical expert”)

- Asistente al equipo de pruebas cuando es necesario
- Competencias especiales necesarias:
 - Administrador de bases de datos o diseño de bases de datos.
 - Experto en interfaces de usuario.
 - Experto en redes.
- Dependiendo del tipo de problema o del entorno de pruebas puede ser necesario que expertos adicionales en pruebas formen parte del equipo de pruebas.
 - En algunas ocasiones son necesarias competencias especiales que no se encuentren directamente relacionadas con las pruebas, por ejemplo, expertos en usabilidad, expertos en seguridad, ect.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Competencias no técnicas (“Soft skills”)

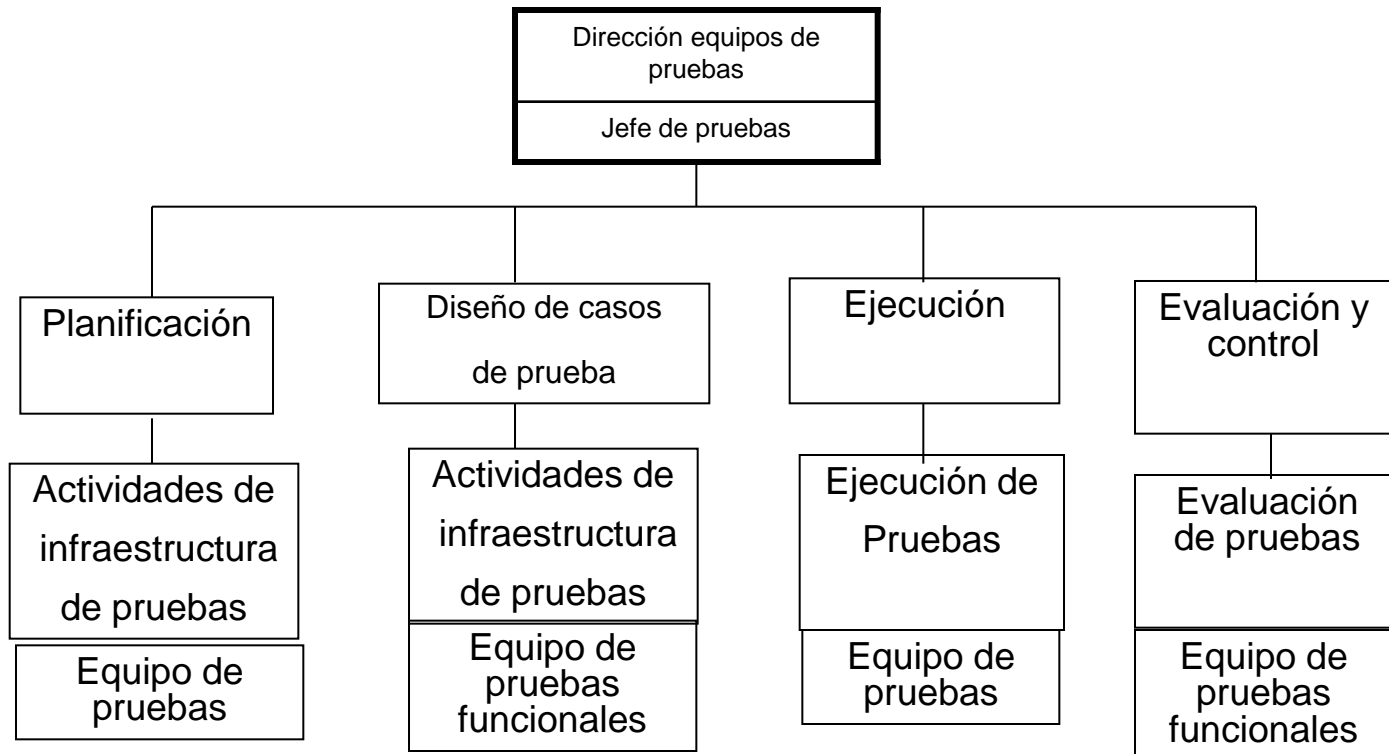
- Con carácter complementario a las competencias técnicas, los miembros del equipo de pruebas requieren de la siguientes competencias y experiencia:
 - Miembro del equipo: instinto (político y) diplomático.
 - Disposición a preguntar sobre hechos aparentemente obvios.
 - Persistencia, fuerte personalidad.
 - Meticulosidad y creatividad.
 - Capacidad para tratar situaciones complejas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Organización de equipos de pruebas.

- Utilizar una organización apropiada para cada proyecto específico.



V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 1

- Organización del equipo de pruebas
- Planificación de pruebas (de acuerdo con el plan de calidad corporativo).
- Planificación de los ciclos de prueba.
- Estrategia de pruebas
- Ejecución y control de pruebas.
- Introducción de un sistema de gestión de incidencias adecuado.
- Introducción de un sistema de gestión de la configuración*.
- Informes de resultado y progreso para la dirección de la organización/compañía

*Esta no es una tarea propia del jefe de pruebas, la gestión de la configuración es necesaria en todas las fases de desarrollo de un producto.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 2

- Gestión de las pruebas.
- Redacción del plan de pruebas.
- Creación de un documento que soporta métodos, recursos y plazos para las actividades del proceso de pruebas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 3

- Pruebas- / Planificación del ciclo de pruebas (“Test cycle planning”).
 - Enfoque de pruebas: implementación de una estrategia de pruebas para un proyecto específico.
 - Ciclo de pruebas: ciclo a través del proceso de pruebas para un objeto de pruebas específico.
 - Actividades del proceso de pruebas (recordatorio):
 - Planificación de pruebas (“test planning”).
 - Especificación de pruebas (“test specification”).
 - Ejecución de pruebas (“test execution”).
 - Información de pruebas (“test evaluation”).
 - Inicialización de solicitudes de cambio y corrección.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 4

- Pruebas- / Planificación del ciclo de pruebas (“Test cycle planning”).
 - Planificación de pruebas.
 - Planificación del proceso de pruebas: debe ser desarrollada en una fase temprana del proyecto. El resultado debe estar reflejado en un documento (plan de pruebas estático – “**static test plan**”).
 - Planificación del ciclo de pruebas.
 - Planificación detallada de un ciclo de pruebas: el plan de pruebas estático será **detallado** para describir un ciclo de prueba específico. Los detalles dependen de la situación particular del proyecto (por ejemplo progreso del desarrollo, resultados de pruebas, disponibilidad de recursos).
- Tareas del jefe de pruebas: **Iniciación, control y supervisión de pruebas y ciclos de pruebas.**

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 5

- Pruebas- / Planificación del ciclo de pruebas (“Test cycle planning”).
 - La **planificación de pruebas** comienza al **inicio del proyecto**. Hitos , presupuesto y prioridades de las diversas actividades del proceso de pruebas requieren ser abordados..
 - Las actividades de planificación deberían dar comienzo tan pronto como sea posible en el proyecto.
 - La planificación detallada, basada en el plan inicial, puede ser desarrollada para cada ciclo de pruebas.
 - Esta es la base para una posterior definición de las reglas para los requisitos de prueba y estrategia de pruebas.
 - El plan de pruebas detallado es confeccionado con el objeto de satisfacer la situación específica del proyecto.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 6

- Pruebas- / Planificación del ciclo de pruebas (“Test cycle planning”).
 - Los recursos deben ser planificados.
 - Estos son escasos y, con frecuencia, deben ser asignados de forma individual.
 - Durante los ciclos de pruebas, pueden ocurrir retrasos de tal forma que la planificación de los recursos de pruebas debe ser revisada.
 - El desarrollo del proyecto debe ser tenido en cuenta.
 - A lo largo del proyecto pueden ocurrir retrasos de tal forma que puedan poner en peligro a la planificación (plazos).
 - En este caso, los casos de prueba planificados tienen que ser “filtrados” con el objeto de cumplir con los hitos. Esta es, con mucha frecuencia, la primera medida tomada cuando los plazos se reducen.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 7

- Pruebas- / Planificación del ciclo de pruebas (“Test cycle planning”).
 - Se deben tener en cuenta las evaluaciones de pruebas anteriores.
 - En cualquier momento, a lo largo del proyecto, los resultados (de las pruebas en curso) de las actividades de pruebas pueden ser tenidas en cuenta de tal forma que la atención se desplaza hacia nuevos requisitos.
 - El control de las actividades de pruebas en curso se realiza utilizando métricas especificadas y acordadas en el plan de pruebas.
 - El cierre (“end-of-test”) de las pruebas se confirma por el jefe de pruebas cuando todos los criterios de salida (medidos utilizando métricas) son cumplidos/ alcanzados.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 8

- Especificación.
 - El objeto principal del proceso de pruebas es detectar la mayor cantidad de defectos relevantes con el menor esfuerzo posible!.
 - Todas las pruebas documentadas en el plan de pruebas son especificadas, es decir, se establece cómo se estructura y cómo debe ser ejecutado. Este proceso es inicializado por el jefe de pruebas.
 - Los casos de pruebas están constituidos por pasos unitarios, cada paso consta de una acción y de un resultado esperado.
 - ¡Los casos de prueba deberían ser obtenidos con la colaboración del personal que cuente con conocimiento de los requisitos funcionales del software!.
 - Los casos de prueba deberían ser diseñados teniendo en mente su carácter repetitivo.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 9

- Ejecución y control.
 - Comparación de los resultados esperados y obtenidos en el proyecto.
 - Cada ciclo de pruebas requiere ser ajustado/adaptado al plan de pruebas.
 - ¿Han ocurrido retrasos o cambios?.
 - ¿Los resultados obtenidos se encuentran dentro del rango esperado?- Número de defectos detectados, tiempo utilizado en correcciones, repetición de pruebas, etc.
 - Todas las desviaciones deben ser informadas y tenidas en cuenta.
 - Habitualmente, las medidas correctivas deben ser tomadas de acuerdo con el plan de pruebas y las actividades de pruebas en curso, por ejemplo:
 - Ajuste de fechas planificadas.
 - Ajuste de recursos para la ejecución de pruebas.
 - Ejecutar ciclos de pruebas adicionales/ omitir ciclos de pruebas.
 - Modificar la prioridad de ciclos de pruebas

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 10

- Evaluación.
 - La gestión de pruebas proporciona transparencia a la evolución del proceso de pruebas y aporta indicadores a la dirección del proyecto.
 - Informes generados durante la ejecución de pruebas (por ejemplo informe de errores, inventario organizado por tipo de defectos, estadísticas), el seguimiento de defecto e informes al cliente son una importante fuente de información para el jefe de proyecto y la dirección de la compañía (por ejemplo como base para la planificación de recursos y plazos).
 - El uso de herramientas y plantillas aumentarán la calidad y pueden reducir la carga de trabajo.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del jefe de pruebas / 11

- Evaluación.
 - La gestión de pruebas incluye la aceptación de resultados del proyecto, significa: el producto debe cumplir con los requisitos definidos y la especificación.
 - El jefe de proyecto, de acuerdo con el jefe de pruebas, decide respecto de la aceptación de los objetos de prueba (por ejemplo pasar a un siguiente nivel de pruebas).
 - Los informes extensivos (documentación) aseguran la ejecución completa de las actividades de pruebas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del Tester

- Asiste en la implementación del plan de pruebas.
- Ejecución de pruebas.
- Informa las pruebas (realización de informes de pruebas).
- Asiste a la implementación de la automatización de pruebas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del Tester / 1

- Asiste en la implementación del plan de pruebas.
 - Especifica y comprueba planes de prueba.
 - Analiza y evalúa bases de pruebas (documentación especificaciones).
 - Desarrolla especificaciones de pruebas.
 - Formula y selecciona casos de prueba y combinaciones de datos de prueba.
 - Formula casos esperados.
 - Prepara, configura y administra el entorno de pruebas (conjuntamente con los administradores de la red y sistema).

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del Tester / 2

- Ejecución de pruebas.
 - Implementación de las pruebas a todos los niveles
 - ejecuta pruebas y registra resultados en un protocolo de pruebas.
 - Evalúa los resultados de las pruebas.
 - Opera herramientas de pruebas.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Tareas del Tester / 3

- Tratamiento después de pruebas (“test post processing”).
 - Creación de protocolo de pruebas.
 - Trazar notas relativas a desviaciones.
 - Ejecutar repetición de pruebas.
 - Preparar documentación de aceptación.
- Asiste en la implementación de la automatización de pruebas.
 - Creación de guiones de pruebas.
 - Creación / operación de herramientas de pruebas en el área de automatización de pruebas.
 - Ejecutar actividades de automatización de pruebas, controlar ejecuciones de captura – repetición (“capture-replay-runs”).
 - Analizar y evaluar resultados.

V – Gestión de pruebas

01- Organización del proceso de pruebas

Resumen

- Los equipos de pruebas deberían ser **independientes**, incluso en el caso de estar constituidos por desarrolladores de los objetos de prueba. Cuanto más independientes, mejor.
- El jefe de pruebas establece el **equipo de pruebas** en una fase temprana y:
 - Planifica y prepara todas las pruebas.
 - Establece una estrategia de pruebas.
 - Organiza la gestión de desviaciones y la gestión de la configuración.
 - Controla la ejecución de pruebas.
 - Evalúa los resultados de las pruebas.
- El **jefe de pruebas informa** a la dirección de la compañía y al jefe de proyecto.
- El **tester apoya** las actividades de preparación de pruebas, ejecuta pruebas, crea la documentación relativa a mensajes de desviación y resultados de prueba. También asiste en la implementación de la automatización de pruebas.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Actividades de planificación de pruebas

- La planificación de pruebas es planificación de proyectos.
 - Todas las tareas y actividades deben ser planificadas con antelación.
 - Para las distintas tareas definidas se deben asignar recursos (personal, presupuesto, herramientas, entornos de prueba, etc.).
 - Concretar las actividades de pruebas en un plan de pruebas y coordinarlas con el plan el plan de proyecto.
 - Definir el nivel de calidad (por ejemplo, profundidad de las pruebas) para los distintos niveles de pruebas.
 - La planificación de pruebas es una actividad continua, debe ser controlada de forma constante.
 - La información proveniente de las actividades de pruebas podrían imponer ajustes en el plan de pruebas con el objeto de afrontar riesgos sujetos a cambios

V – Gestión de pruebas

02- Planificación y estimación de pruebas

La planificación de pruebas es parte de la planificación de la calidad en su conjunto.

- La planificación de pruebas es una parte importante del aseguramiento de calidad, pero no es la única parte.
- La estructura y contenidos de un plan de calidad pueden ser encontrados en el estándar IEEE 730 (con información adicional en el estándar IEEE 983).
- Elementos de un plan de aseguramiento de la calidad de acuerdo con el estándar IEEE 730: planificación y descripción de:
 - Organización del proyecto.
 - Documentos que cubre el ciclo de vida de desarrollo.
 - Estándares, métodos y convenciones de la misma manera que un mecanismo que asegure que aquellos son seguidos (cumplidos).
 - Revisiones y auditorias durante el ciclo de vida de desarrollo.
 - Proceso de pruebas.
 - Documentación de errores, remedios.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

La plan de pruebas (estático)

- Tras definir el rol de l proceso de pruebas en el marco de actividades de aseguramiento de la calidad (QA), el proceso de pruebas comienza su fase de planificación.
- El primer paso de la planificación es la creación de un plan de pruebas estático.
 - El plan de pruebas cubre todas las fases del proceso de pruebas.
 - Las reglas se fijan de acuerdo los objetivos de las pruebas, recursos, actividades de pruebas, hitos, etc.
- El plan de pruebas (estático) es, posteriormente, ampliado con el objeto de cubrir los resultados a partir de la fase de planificación de detalle.
 - La planificación depende del proyecto complementa la primera versión del plan de pruebas.
 - El plan de pruebas cuenta con un extensión dinámica, que será ajustado durante el ciclo de vida del proyecto, si eso fuera necesario.
 - El estándar IEEE 829 aporta una estructura de plan de pruebas acreditada.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

La plan de pruebas de acuerdo al estándar IEEE 829

1. Introducción.
2. Supuestos.
3. Ítems de prueba.
4. Características sujetas a pruebas.
5. Características no sujetas a pruebas.
6. Enfoque.
7. Criterios de éxito/fracaso para un ítem.
8. Criterios de suspensión/reanudación.
9. Entregables de pruebas.
10. Tareas de pruebas.
11. Necesidades relativas al entorno.
12. Responsabilidades.
13. Dotación de personal y formación.
14. Calendario.
15. Riesgos y contingencias.
16. Aprobación.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Actividades a realizar

- La planificación de pruebas comienza al inicio de un proyecto de desarrollo y se ajusta a lo largo del ciclo de vida del proyecto.
- La planificación de pruebas también cubre la creación y actualización del plan de pruebas. Las siguientes actividades se explican con mayor detalle:
 - Estrategia de pruebas (“test strategy”).
 - Planificación de recursos (“resource planning”).
 - Prioridad de las pruebas (“priority of tests”).
 - Soporte de herramientas (“tool support”).

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Actividades a realizar – Estrategia de pruebas (“Test strategy”)

- La estrategia describe los **niveles de prueba** a desarrollar y la **intensidad de las pruebas** en aquellos niveles.
- La estrategia de pruebas también establece los **criterios de entrada y salida** para cada nivel, incluyendo las **métricas** para evaluar estos criterios.
- Es necesaria una estrategia de pruebas dado que probar un sistema de forma completa no es viable. Probar con todas las combinaciones de datos de prueba, estados internos y restricciones temporales es prácticamente imposible.
- La valoración de riesgos ayuda a centrar la atención en aquellas áreas en que las actividades de pruebas presentan un riesgo de fallo más alto.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Actividades a realizar – Planificación de recursos (“resource planning”)

- El objetivo principal de la planificación de recursos es estimar el esfuerzo de los miembros del equipo, incluyendo sus necesidades en términos de tiempo, herramientas, actividades de apoyo, etc. Estas **estimaciones** se convierten en parte del plan de pruebas (dinámico).
- Este plan de pruebas cuenta con un **calendario** (“**time table**”) detallado, incluyendo hitos, asignación de personal a actividades. Este plan es un instrumento para gestionar la tarea global de la ejecución de pruebas con todas sus actividades.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Actividades a realizar – Planificación de pruebas (“test planning”)

- **Gestionar tiempos:** muchos proyectos experimentan intensos **problemas de tiempo** en torno a las fases finales. Esto puede conducir a decisiones sobre de reducción de actividades de pruebas u omitir pruebas de forma completa.
- **Priorizar pruebas:** dado que la distribución de software sin haber sido aprobado suficientemente conlleva un alto riesgo, es necesario asignar prioridades a las actividades de prueba. Esto debe ser realizado de tal forma que los casos de prueba mas importantes sean ejecutados de **forma temprana**. De esta forma **partes críticas** de los programas son probadas incluso en el caso en el que actividades de pruebas sean abortadas de forma prematura.
- **Selección de herramientas:** decidir respecto de qué herramientas deben ser utilizadas para probar, si las herramientas disponibles son suficientes o si hay necesidad de herramientas adicionales.
- **Documentar:** definir el nivel de detalle, estructura y plantillas para la documentación de pruebas.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Criterios de salida de pruebas (“test exit criteria”)

- Los criterios de salida que indican la finalización de una fase de pruebas, deben ser establecidas para cada nivel de pruebas. Son necesarias métricas para controlar estos criterios de salida.

Ejemplos:

- **Cobertura de código (“code coverage”)**
 - x% de código (de un programa) que ha sido ejecutado.
 - x% de todas las funciones / todas las opciones de menú que han sido cubiertas.
- **Cobertura de riesgo (“risk coverage”)**
 - Casos de prueba de una clase de riesgo predefinido (por ejemplo el nivel de riesgo más alto) han sido ejecutados con éxito en su totalidad.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Criterios de salida de pruebas (“test exit criteria”)

- **Aborto de pruebas debido a razones de tiempo, costos o calidad.**
 - Las actividades de pruebas son paralizadas/suspendidas cuando se alcanza la fecha de entrega o el presupuesto se agota. Es muy frecuente que ésta sea la realidad en proyectos, en muchas ocasiones esta circunstancia tiene un alto coste en tiempo y dinero con posterioridad.
 - Si no ha sido alcanzado un mínimo de calidad, las pruebas pueden ser suspendidas o incluso no ser iniciadas (muchos defectos críticos).
- **Tasa de detección de errores (“error finding rate”)**
 - El número de nuevos errores detectados cae por debajo de un valor predeterminado.
 - Ejemplo: las pruebas han sido suspendidas si se han detectado menos de un error por hora.
 - Las economías del proceso de pruebas deben ser tenidas en cuenta. Más allá de una cierta tasa de detección de errores puede resultar más barato corregir que buscar y eliminar errores.

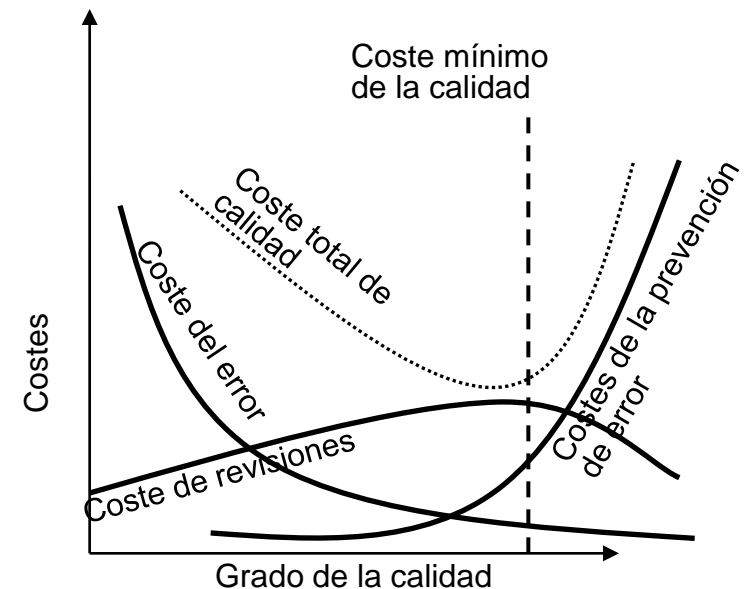
V – Gestión de pruebas

02- Planificación y estimación de pruebas

Criterios de salida de pruebas (“test exit criteria”)

■ Economía del proceso de pruebas.

- Un grado creciente de calidad representa un coste mas bajo del error, pero costes más altos en prevención de errores.
- Inicialmente, el coste de la revisión se incrementa, a continuación se reduce (las revisiones no aportan beneficios en las fases finales del proyecto).
- Inicialmente, el coste total de la calidad se reduce, a continuación se incrementa los costes de la calidad son los mas bajos donde la curva representa un mínimo.
- Frecuentemente es necesario aportar un mínimo de calida con el objeto de poder mantenerse en el negocio.



V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas- factores (síntesis)

- Características del producto (por ejemplo complejidad).
- Calidad de la base de pruebas.
- Requisitos de fiabilidad y seguridad del producto.
- Complejidad del proceso de desarrollo.
- Estabilidad de la organización, madurez del proceso utilizado.
- Personal involucrado, restricciones temporales.
- Métodos para estimar el esfuerzo en el proceso de pruebas.
 - Estimación basada en tareas.
 - Estimación basada en analogías.
 - Estimación basada en porcentajes.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en tareas /1

- Método.
 - Identificar todas las tareas a ejecutar (normalmente utilizando un enfoque descendente (“top down”)).
 - Obtener estimaciones para cada tarea por los responsables (de su ejecución) o por expertos.
 - Sumar todos los valores de las tareas. Incluir los factores de corrección (si hay experiencias respecto de la exactitud de ciertas estimaciones).
 - Incluir elementos amortiguadores (buffers)/ elementos adicionales con el objeto de cubrir tareas omitidas o subestimadas.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en tareas / 2

- Ventajas.
 - Las actividades de estimación pueden estar estrechamente vinculadas a la planificación del proyecto.
 - La estimación da origen a una información de detalle que puede ser controlada y ajustada a lo largo del ciclo de vida del proyecto.
 - Las tareas pueden ser asignadas a grupos (por ejemplo pequeño, mediano, grande) y los esfuerzos son estimados para un representante del mismo.
- Desventajas.
 - Este método es extensivo y caro.
 - Este método requiere una idea clara respecto de la estrategia de pruebas y actividades de pruebas en una fase temprana del proyecto.
 - La experiencia demuestra que las estimaciones son, en la mayoría de los casos, a la baja. Esto podría deberse a la omisión o subestimación grosera de ciertas tareas.
 - Los elementos amortiguadores incorporados son incorporados durante la planificación del proyecto.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en analogías /1

- Método.
 - Clasificar las tareas de pruebas requeridas.
 - Buscar un proyecto que se haya desarrollado en el pasado que contenga una tarea similar a una específica.
 - Utilizar el esfuerzo real de esta tarea como base de la estimación.
 - A través del uso de métricas (líneas de código, número de módulos, número de casos de prueba, etc.) como base, calcular el valor de la estimación total.
 - Considerar factores de corrección.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en analogías / 2

- Ventajas.
 - Este método es simple y efectivo.
 - Se pueden lograr valores muy precisos para la estimación si se cuenta con suficiente experiencia.
- Desventajas.
 - Se requiere personal con experiencia y/o información detallada respecto del proyecto actual y las tareas a estimar.
 - Los criterios para la clasificación de proyectos pueden no cubrir todos los aspectos de un proyecto.
 - Frecuentemente conduce a debates con la dirección respecto de la validez de la estimación.

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en porcentaje /1

- Método.
 - El esfuerzo para las actividades de pruebas se estiman sobre la base de la totalidad de las actividades del proyecto.
 - El valor del porcentaje (fracción) requiere ser determinado basándose en la experiencia.
 - Ejemplo: Spillner / Linz habla de un porcentaje del 50% de actividades de pruebas respecto a la totalidad de actividades del proyecto (véase también “Basiswissen Softwaretest”, dpunkt. Verlag, 3. Auflage, S. 181).
 - Este método también puede ser utilizado para parte del trabajo (por ejemplo estimación para los costes de gestión de proyecto, estimación del esfuerzo de pruebas para las pruebas del sistema)

V – Gestión de pruebas

02- Planificación y estimación de pruebas

Estimación de pruebas: estimación basada en porcentajes / 1

- Ventajas.
 - Técnica de estimación muy simple y potente que no requiere excesiva información de entrada.
- Desventajas.
 - No muy precisa, dado que no tiene en cuenta los hechos particulares del proyecto.
 - Es necesaria mucha experiencia e intuición por parte del estimador con el objeto de obtener estimaciones válidas.
 - La decisión respecto del valor del porcentaje puede conducir a debates difíciles.
 - Tiene en cuenta actividades que ya forman parte de las estimaciones de la planificación de l proyecto (por ejemplo ¿El esfuerzo de pruebas del desarrollador forma parte de la estimación correspondiente al desarrollo o debe formar parte de las estimaciones del proceso de pruebas?).

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Planificación de pruebas, seguimiento de pruebas y control de pruebas

- **La planificación de pruebas (“test planning”)**: las pruebas deben ser iniciadas.
- **Seguimiento de pruebas (“test monitoring”)**: control de las actividades de pruebas con el objeto de detectar desviaciones respecto al plan.
- **Control de pruebas**: corrección del rumbo de las actividades de pruebas cuando sea necesario.
- El seguimiento debe ser realizado en base consideraciones medidles.
 - Métricas en base a errores, por ejemplo, utilizando información procedente del sistema de gestión de incidentes.
 - Métricas en base a casos de prueba, por ejemplo utilizando información procedente del sistema de gestión de pruebas.
 - Métricas en base a objetos de prueba, por ejemplo utilizando información procedente del sistema de gestión de la configuración.
 - Métricas en base a costes , por ejemplo utilizando información procedente del sistema de control del proyecto.
- Los resultados obtenidos de la medición deben ser **informados** de forma regular.
- Deben ser utilizados informes de estado de pruebas (planilla: por ejemplo IEEE 829)

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Informe del estado de pruebas /1

- La información respecto de las actividades de pruebas se consolida a los efectos de la información de pruebas (“test reporting”).
- Ejemplo del contenido de un informe de estado de pruebas (según IEEE 829).
 - Objeto u objetos de prueba.
 - Niveles de prueba, ciclos de prueba, periodo del informe.
 - Estado de pruebas (utilizando métricas, por ejemplo número de defectos documentados, número de casos de prueba ejecutados).
 - Recursos utilizados / presupuesto consumido.
 - Hitos alcanzados (por ejemplo aceptación de objetos de pruebas en niveles de pruebas específicos).
 - Informe de defectos (número de defectos descubiertos, número de defectos corregidos).
 - Evaluación del riesgo (nuevos riesgos / riesgos modificados respecto de informes previos).
 - Pronóstico: actividades planificadas para el próximo periodo de informe.
 - Evaluación general / estado (semáforo).

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Informe del estado de pruebas / 2

- Frecuencia de los informes.
 - Al inicio del proyecto / en la fase de preparación los ciclos de los informes son más largos (quincenal o mensual).
 - Las fases “críticas” de la ejecución de pruebas requieren ciclos cortos (semanales, o incluso diario).
 - El informe de cierre de pruebas al final del proyecto.
 - Evaluación de los informes de pruebas.
 - ¿El desarrollo es apropiado?
 - ¿La ejecución de las pruebas es eficaz y eficiente?
 - ¿Las actividades están alineadas con los objetivos de pruebas?
¿Están siendo alcanzados objetivos de pruebas?
 - ¿Cuál es el grado de riesgo de aquellos defectos que no han sido detectados?.

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Control de pruebas

- El control de pruebas es una tarea de gestión
 - El jefe de pruebas pertenece a los cuadros directivos del proyecto.
 - Las tareas de dirección pueden incluir otras áreas del proyecto principal / tareas ajenas al alcance de pruebas.
- Medidas correctivas como respuesta a desviaciones respecto al plan.
 - El control de pruebas incorpora a todas las medidas emprendidas durante el proceso de pruebas.
 - Ajuste de las actividades planificadas y , cuando sea necesario, iniciar un nuevo ciclo de planificación en el plan de proyecto.
- Evaluación del cierre de pruebas.
 - Los criterios de salida de pruebas también son registrados con las métricas de progreso d pruebas.
 - Los criterios de salida de pruebas que hubieran sido alcanzados son documentados en el informe de pruebas para su aprobación.

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Medidas de control de pruebas

- Provisión de mayores recursos
 - Mas recursos humanos
 - Mas presupuesto.
 - Despliegue de herramientas para la automatización de tareas
- Reducción del esfuerzo aplicado al trabajo.
 - Exclusión de variaciones de casos de pruebas.
 - Simplificación de objetos de prueba complejos / omisión de objetos específicos.
 - Reducción de la cantidad de datos de prueba.
 - Omisión de casos de prueba / juegos de pruebas.
- Las medidas de control de pruebas son documentados con el objeto de informar a la dirección del proyecto / cliente respecto de cambios en riesgos en el despliegue del producto.

V – Gestión de pruebas

03- Seguimiento y control del estado de las pruebas.

Resumen

- El seguimiento del estado de pruebas se basa en **criterios medibles** y aporta la información necesaria para gestionar el proceso de pruebas.
- Las **desviaciones** respecto al plan requieren acciones correctivas.
- La **presentación regular de informes** importa información al proyecto y a la dirección de la compañía respecto al progreso de las pruebas.

V – Gestión de pruebas

04- Gestión de la configuración

Observaciones generales

- La gestión de la configuración presenta un rol de apoyo dentro de un proyecto- todos los cambios deben ser registrados en un recurso común y comunicado haciendo uso de procesos definidos.
- Las expectativas a respecto a la gestión de la configuración puede variar de forma considerable dependiendo del tipo de alcance de proyecto – se debe desarrollar un plan de gestión de la configuración específica.
- El IEEE 829 aporta un estándar para gestión de la configuración y el plan de gestión de la configuración.
- La gestión de la configuración no es una actividad particular del proceso de pruebas, es necesaria durante todas las fases de un proyecto.
- La gestión de la configuración sin una herramienta apropiada solo es posible en proyectos muy pequeños.

V – Gestión de pruebas

04- Gestión de la configuración

Definiciones

Gestión de la configuración GC(“configuration management (CM)”) se refiere a un conjunto de medidas que complementan al desarrollo del software:

- **Gestión del cambio** (“change management”) sigue todas las actividades, por ejemplo cambios en el código fuente para cada solicitud de cambio.
- **Gestión de la construcción** (“build management”) describe todos los pasos para crear una versión de un producto software con el objeto de ser suministrado como un todo o subsistema individuales.
- **Gestión de entregas** (“release management”) permite la definición de versiones aisladas para cada artefacto componente de una versión completa de un producto a ser probado, entregado, etc.
- **Gestión de versiones** (“versions management”) registra toda la información de acceso para cada artefacto: versión actual (número) , ultimo cambio, ultimo usuario, etc.

V – Gestión de pruebas

04- Gestión de la configuración

Problemas tratados por la gestión de la configuración

- **Cuál es la versión actual?** La ambigüedad con respecto a que versiones se corresponden puede resultar en actividades de desarrollo basadas en versiones antiguas (obsoletas) de la especificación.
- **Que ha sido modificado, cuando y quien lo modificó?** Son posibles cambios concurrentes de un fichero: ¿Qué cambios pueden ser sobre escritos?
- **¿Qué versión del fichero ha sido probada?** Es difícil probar y extraer una conclusión de unas pruebas cuando no se tiene conocimiento concreto de la versión de la que se trata.
- **¿Qué aterfactos se corresponde?** ¿Qué versiones han sido agrupadas para crear las distintas entregas (“release”)?

V – Gestión de pruebas

04- Gestión de la configuración

Los requisitos sobre GC conforman el punto de vista del proceso de pruebas

- Control de versiones (“versión control”)
 - Clasificar, guardar y recuperar diferentes versiones de un objeto (V1.0, V1.1, etc.).
- Gestión de la configuración (gestión de entregas – (“release management”))
 - Determinar y administrar toda la información de las versiones correspondientes que conforman un subsistema.
- Protocolos, comentarios y razones para los cambios realizados.
- Mantener un registro del estado.
 - Trazar defectos y cambios, registrar informes de problemas y aportar actividades de retroceso

V – Gestión de pruebas

04- Gestión de la configuración

Auditoria de la configuracion (“configuration audit”)

- Se introduce una auditoria de la configuración con el objeto de comprobar la efectividad de las actividades de la gestión de la configuración
- La auditoria de la configuración comprobara:
 - Si todos los componentes individuales de un sistema son considerados en la gestión de la configuración
 - Si las configuraciones individuales pueden ser identificadas correctamente.

V – Gestión de pruebas

04- Gestión de la configuración

Resumen

- La gestión de la configuración es necesaria para **administrar los cambios** sobre los objeto de prueba y sus respectivas **versiones**.
- Información de la **construcción (build)** y la **entrega (release)** es conservada con el objeto de poder reconstruir versiones antiguas.
- La gestión de la configuración se aplica al **proceso de desarrollo software completo**, no solamente al proceso de pruebas.
- La gestión de la configuración es apenas posible sin las **herramientas apropiadas**

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Riesgo

- Riesgo (from: Deutsche Wikipedia)
 - “Un riesgo calculado es una predicción de un posible daño, respectivamente, la pérdida en caso de un resultado negativo (peligro) o una posible ventaja, respectivamente, ganancia en caso de un resultado positivo (oportunidad)”.
 - El riesgo es la probabilidad de un resultado negativo (matemático), o la probabilidad de la ocurrencia de un suceso negativo multiplicado por el monto del daño económico (económico).
- Riesgo (“Watizing con bears”, Tom Demarco/ Timothy Lister)
 - “Un posible evento futuro que conducirá a un resultado indeseable (causa), respectivamente, este resultado indeseado de la misma (efecto)”
- El riesgo asociado al proyecto y al producto deben ser tenidos en cuenta durante la planificación y el diseño de casos de prueba, cuando se prioricen casos de prueba, cuando se seleccionen métodos y durante la ejecución de pruebas.

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Riesgos asociados al proyecto

- Los riesgos asociados al proyecto afectan al éxito del proyecto y deben ser gestionados.
- Estimación de la probabilidad y daño potencial.
- Implementar medidas apropiadas para tratar los riesgos identificados:
 - Evitar el riesgo (evitar situaciones de riesgo).
 - Mitigación del riesgo (preparación activa de medidas para reducir la probabilidad y/o el daño potencial).
 - Control de riesgo (preparar las medidas necesarias en el caso en el cual el riesgo se convierte en un problema, contar con tiempo y fondos disponibles).
 - Ignorancia por el riesgo (esperar que el riesgo no se convierta en un problema, rezar, cruzar los dedos, etc.).
- Las primeras tres medidas tienen un coste económico – la cuarta medida pueden tener un coste económico muy alto.

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Tipos de riesgos asociados al proyecto (más frecuentes)

- Riesgos asociados a la organización (“organizational risks”)
 - Capacitación y disponibilidad del personal.
 - Problemas personales entre equipos / miembros del equipo.
 - Cooperación insuficiente entre departamentos / conflictos de intereses.
 - Estimaciones no realistas de plazos del proyecto.
- Riesgos tecnológicos.
 - Requisitos defectuosos, incompletos o no realistas.
 - Tecnologías métodos herramientas, etc. Nuevas o que presentan incertidumbres para el desarrollo del software.
 - Déficit de calidad en productos.
 - Disponibilidad de un entorno de prueba complejo.
- Riesgos ambientales (“environmental risks”).
 - Deficiencias por parte de extremos en la provisión de componentes (plazos, calidad, coste).
 - Problemas de aceptación y otros inconvenientes contractuales con proveedores.
 - Acceso concurrente a recursos externos.
 - Cambios en requisitos legales

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Riesgos asociados al producto

- Los **riesgos asociados al producto** son el resultado de problemas relacionados con el **producto suministrado**.
 - **Funcionalidad** insuficiente del producto suministrado.
 - atributos **no funcionales** insuficientes.
 - El producto **no es idóneo** para su **uso** previsto, por lo tanto no puede ser puesto en operación.
 - El producto provoca **daños a la propiedad**.
 - El producto provoca **muerte o lesión** accidentales
- Las **pruebas** se ejecutan para reducir u evitar los riesgos asociados al producto.
 - La **probabilidad de ocurrencia** de un riesgo se reduce.
 - El **daño potencial** por la ocurrencia de un riesgo se reduce.

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Gestión de riesgos asociados al producto / 1

- Gestión de riesgos asociados al producto utilizando pruebas basadas en el riesgo.
 - Identificar, analizar y priorizar riesgos.
 - **Influencia del riesgo** tenido en cuenta durante la planificación de pruebas
 - Seleccionar los **métodos de pruebas** para mitigar riesgos
 - Asignar **alcance de la pruebas** (profundidad) de acuerdo al nivel de riesgo.
 - Adaptar el **orden de ejecución** de casos de prueba (los casos de prueba importantes en primer lugar con el objeto de detectar defectos críticos de forma temprana).
 - Actualizar la **lista de evaluación de riesgos (“risk assessment worksheet”)** de forma regular.
 - Los riesgos pueden desaparecer (el proveedor a entrega en plazo).
 - Pueden aparecer nuevos riesgos (el cliente solicita funciones adicionales)
 - Los riesgos pueden cambiar (epidemia de gripe).

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Gestión de riesgos asociados al producto / 2

- Beneficios de las pruebas basadas en el riesgo.
 - Los **métodos de pruebas** son seleccionados de forma particular con el objeto de mitigar los riesgos identificados.
 - El **alcance de las pruebas** se ocupa de los riesgos identificados.
 - El **alcance del proceso de pruebas** tiene en cuenta los riesgos identificados. De esta forma, el esfuerzo en el `proceso de pruebas se centra en abordar la reducción del riesgo potencial.
 - Los fallos peligrosos son detectados de forma temprana, por lo tanto se hace más **económica** su corrección.
 - Incluso en el caso de un aborto de pruebas, se asegura que los **casos de prueba más importantes** han sido **ejecutados** (asignación de prioridades a pruebas basada en el riesgo)

V – Gestión de pruebas

05- Riesgo y proceso de pruebas

Resumen

- Los **riesgos asociados al proyecto y al producto** ponen en peligro el éxito del proyecto, los riesgos deben ser **gestionados**.
- Los riesgos pueden ser **tecnológicos**, del **entorno** o estar **asociados a la organización**.
- Riesgo (valor) = probabilidad de ocurrencia por daño potencial.
- “La gestión del riesgo es gestión de proyecto para adultos”

V – Gestión de pruebas

06- Gestión de incidencias

Detección de errores durante las pruebas

- El tester ejecuta los casos de prueba y registra los resultados.
- Posteriormente analizan las desviaciones entre los resultados esperados y los obtenidos:
 - Se identifica los fallos (los fallos pueden ocurrir en todo lugar: en documentos, en el código, los datos de salida de un objeto de prueba, en un texto de ayuda)
- En este punto (temporal), las tareas del tester han finalizado por este momento.
 - El tester espera la versión corregida del programa para ejecutar la repetición de pruebas (“retest”).
- Posteriormente el seguimiento de errores se realiza utilizando un sistema de gestión de errores (sistema gestión de errores / defectos)

V – Gestión de pruebas

06- Gestión de incidencias

¿Quién hace qué? / 1

- El tester.
 - Ejecuta los casos de prueba con el objeto de detectar errores.
 - Registra los resultados en un protocolo de pruebas.
 - Introduce los errores en un repositorio (informe de problemas).
- Jefe de pruebas (“test manager”)
 - Evalúa el informe de problemas.
 - Asigna prioridades de los errores (de acuerdo con la dirección del proyecto, cliente, etc.)
 - Redacta el informe de estado en función del estado actual de las labores de corrección.

V – Gestión de pruebas

06- Gestión de incidencias

¿Quién hace qué? / 2

- Concejo de Control de Cambio (CCC) (“Change Control Board (CCB)”)
 - Decide respecto de los cambios de requisitos y sus prioridades.
- Desarrollador
 - Analiza los fallos, localiza la causa del error
 - Corrige la cusa del error de acuerdo con la prioridad asignada.
 - Ejecuta todos los cambios aprobados
- Nota: todas estas tareas son ejecutadas en forma iterativa:
 - Jefe de prueba. (“test manager”)
 - Concejo de Control de Cambio (CCC) (“Change Control Board (CCB)”)
 - Desarrollador (“developer”)
 - Tester.

V – Gestión de pruebas

06- Gestión de incidencias

Estructura de un informe de incidencias (informe de errores)

- ¡El enfoque de incidencias describe el efecto de un error, no su causa!
- La plantilla / estructura de un informe de incidencias puede ser encontrada en el estándar IEEE 829
- Los elementos que puede incluir un informe de incidencia.
 - Datos del error.
 - Número único del error (por ejemplo generado automáticamente)
 - Denominación del objeto de prueba (denominación, versión)
 - Entorno de pruebas
 - Nombre del autor del informe de incidencias.
 - Fecha de la primera ocurrencia.
 - Clasificación de errores.
 - Clase de error (“error class”)
 - Estado del error.(“error state”) (error nuevo, repetición de prueba, etc.)
 - Prioridad. (“priority”) (asignación de la urgencia)

V – Gestión de pruebas

06- Gestión de incidencias

Estructura de un informe de incidencias (informe de errores)

- Elementos que pueden incluir un informe de incidencias:
 - Descripción
 - Caso de prueba (aporta todos aquellos detalles respecto de las precondiciones).
 - Resultado erróneo / modo de fallo (usando un descripción del resultado obtenido y el resultado esperado)
 - Descripción de la desviación para facilitar su resolución.
 - Severidad del error (impacto y afectados/ implicado)
 - Referencias cruzadas a informes relacionados.
 - Comentarios.
 - Acciones correctivas tomadas

V – Gestión de pruebas

06- Gestión de incidencias

Clase de error y prioridad del error

La **severidad** de un error se expresa por la asignación de una clase de error.

- Las clases de errores a utilizar pueden ser: error critico, error mayor, error medio, error menor. Son frecuentes tres u cuatro errores.
- El criterio para la clasificación puede ser la influencia y la usabilidad del producto.

La **prioridad** tiene en cuenta el efecto del error.

- Impacto sobre la funcionalidad del programa.
- Impacto sobre el proyecto, sobre el cliente.
- Posibilidad de aportar una solución (corrección) inmediata al problema en la siguiente entrega.

La prioridad rige la **urgencia** de la corrección.

V – Gestión de pruebas

06- Gestión de incidencias

Estado de un error

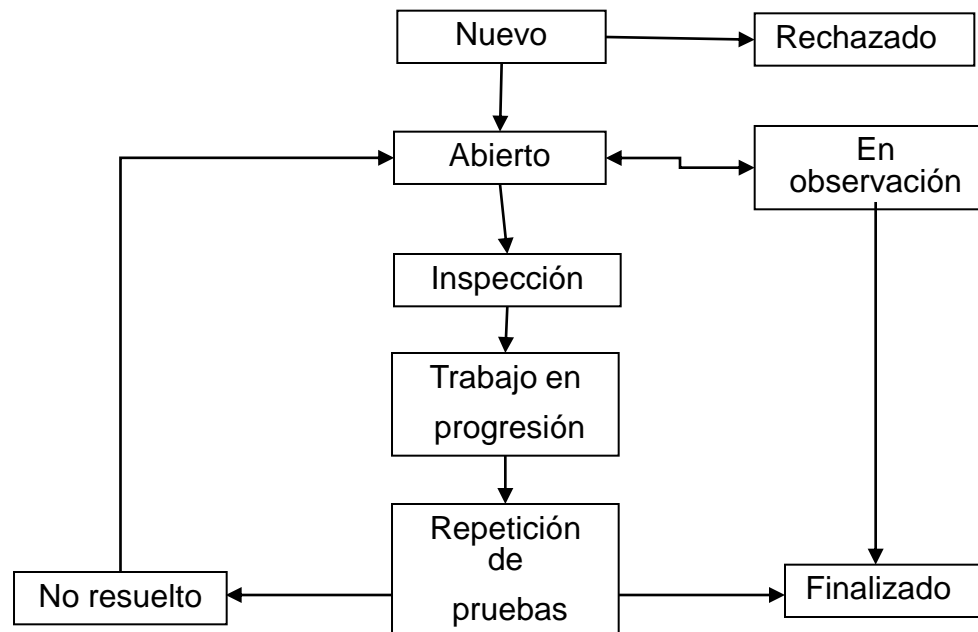
- El estado de un error aporta información relativa al progreso / evolución del trabajo que ha sido desarrollado para este error
- Los posibles estados de un error son, pero no están limitados a las siguientes:
 - **Nuevo (“new”)**– El tester a introducido un error en el sistema.
 - **Abierto (“open”)**- El jefe de pruebas a confirmado el informe del problema.
 - **Rechazado (“rejected”)** - El jefe de prueba ha rechazado el informe del problema.
 - **Inspección (“inspection”)** - El desarrollador intenta identificar el error.
 - **En observación (“surveillance”)** - El error no puede ser reproducido, se encuentra bajo vigilancia
 - **Trabajo en progresión (“WorkInProgress”)** – El error es localizado y preparado / desbloqueado para su correccion
 - **Repetición de pruebas (“retest”)** - El desarrollador a corregido la causa del error
 - **Finalizado (“finalized”)** - El tester a verificado la corrección a través de la repetición de las pruebas.
 - **No resuelto (“NotSolved”)**- El tester no ha podido verificar la corrección, el error aun se encuentra ahí

V – Gestión de pruebas

06- Gestión de incidencias

Estado de un error

- El estado de un error aporta información relativa al progreso / evolución del trabajo que ha sido desarrollado para este error.



V – Gestión de pruebas

06- Gestión de incidencias

Estado de un error

- ¡Solo un tester puede tener un error en estado finalizado!.
- Normalmente el jefe de prueba decide si un error debe ser corregido o rechazado – de forma alternativa el concejo de control de cambio puede decidir sobre la corrección de un error teniendo en cuenta el costo de reparación.
- Todos los cambios (incluidos los comentarios) deben ser registrados en el sistema de gestión de incidencias.
 - El control continuo sobre el estado de corrección de un error esta asegurado.
 - Las actividades de pruebas futuras pueden ser planificadas.
 - En ocasiones, pueden ser generados casos de prueba adicionales con el objeto de localizar la causa de un fallo.

V – Gestión de pruebas

06- Gestión de incidencias

Análisis de informes de errores

- Todos los informes de error son analizados de forma sistemática con el objeto de evaluar el estado de desarrollo de las actividades de corrección de errores, conformidad con el plan de proyecto y la calidad de software
- Elementos de atención característicos.
 - ¿Es perceptible una reducción en el numero de detecciones de nuevos errores?
¿O se está incrementando el numero a lo largo del ciclo de vida del proyecto?
 - ¿Hay objetos de prueba particulares que representen un alto numero de errores? ¿Hay algún objeto de prueba que presente un número de errores más bajo que el numero medio de errores ?
 - ¿Cuántos errores de severidad alta / prioridad alta aún siguen abiertos?
 - ¿Cuánto tiempo requiere la corrección de un error? ¿cual es el tiempo medio para la corrección de errores?
- Las herramientas de gestión de incidencias ofrecen una amplia variedad de informes sobre estadística de errores.

V – Gestión de pruebas

06- Gestión de incidencias

Resumen

- La gestión de incidencias es la **gestión de las desviaciones** / errores detectados durante las pruebas.
- La **gestión de incidencias** es un proceso por sí mismo con un flujo de trabajo (“workflow”) particular.
- Hay **potentes herramientas** disponibles para dar soporte a la gestión de incidencias, que cubren las tareas de la gestión del cambio.
- Las expresiones gestión de las evidencias (“**deviation management**”) o gestión de los errores (“**error management**”) son utilizados con frecuencia como sinónimos de la gestión de incidencias.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Observaciones generales

- Las herramientas de prueba pueden ser utilizadas para dar soporte a las actividades de pruebas.
 - El soporte en la **ejecución de pruebas** se refiere a la automatización de pruebas.
 - Las herramientas de prueba pueden dar soporte a otras actividades de pruebas.
- La denominación de las herramientas de pruebas se realiza según el tipo de soporte que prestan.
 - Hay herramientas disponibles para **cada nivel del proceso de pruebas**.
- En analogía a **CASE- Tools** (Computer Aided Software Engineering), en ocasiones se hace referencia a todas las herramientas de pruebas como CAST-Tools (Computer Aided Software Testing)

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Clasificación de las herramientas de pruebas / 1

- Las herramientas utilizadas para tareas específicas versus paquetes de herramientas de pruebas (“test tool suites”).
 - Las **herramientas unitarias** (“**single tools**”) dan soporte a una tarea particular, son diseñadas para una actividad de pruebas.
 - Los **paquetes de herramientas** cubren varias tareas e integran varias herramientas unitarias.
- Herramientas de pruebas intrusas versus herramientas que no alteran el objeto de prueba.
 - Herramientas **intrusas** (“**intrusive tools**”) puede interferir en la ejecución del objeto de prueba y puede provocar que difiera respecto al objeto en el entorno real.
 - El depurado introduce puntos de corte y alteran el tratamiento de interrupciones.
 - Los “drivers” de pruebas aportan al objeto de pruebas datos de entrada artificiales.
 - La cobertura se determina a través de contadores introducidos en el código
- Esto no siempre es deseable.
 - Durante las pruebas de rendimiento el objeto de prueba debe trabajar en un entorno tan cercano al real como sea posible.
 - Durante las pruebas de sistema los objetos de prueba deben ser embebidos en un entorno en tiempo real.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas de gestión de pruebas y planificación de pruebas / 2

- Las herramientas de gestión de incidencias (herramientas de gestión de defectos.)
 - Registro y seguimiento (“tracking”) de defectos.
 - Asignación de prioridades, categorización de defectos.
 - Evaluaciones, es decir, métricas que representen el grado de desarrollo / progreso de las pruebas.
 - Flujo de trabajo para el ciclo de vida de un defecto: cambios de estado, responsabilidad.
- Herramientas de gestión de la configuración.
 - Seguimiento de las diferentes versiones de componentes: requisitos cumplidos por una versión particular, entorno operativo, compilador en uso, etc.
 - Administración del código fuente y el código objeto.
 - Referencias a la gestión de pruebas / gestión de requisitos / gestión del cambio.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para pruebas estáticas: revisiones

- Herramientas para revisiones (“reviews”).
 - Apoyo al proceso de revisión {flujo de trabajo (“workflow”)}.
 - Documentación de los resultados de revisión.
 - Evaluación de los resultados de revisión.
 - Provisión de las listas de comprobación (“check lists”) para revisiones.
 - Apoyar la ejecución de revisiones en línea (“online reviews”).

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para la gestión de pruebas / 2

- Generadores de datos de prueba asociados a **bases de datos**.
 - Generan datos a partir de bases de datos o a partir de ficheros planos.
 - Obtienen datos a partir del reconocimiento de **estructuras** y contenidos.
- Generadores de datos de prueba basados en el **código**.
 - Generan datos de prueba a partir del **código fuente**.
 - No son capaces de aportar los **valores de resultados esperados**.
 - De forma similar a los métodos de caja blanca, **sólo** pueden generar datos de prueba en base al **código aportado**.
 - **No pueden identificar** una funcionalidad **ausente**.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para la especificación de pruebas / 3

- Generadores de datos de prueba **asociados a la interfaz**.
 - Generan datos de acuerdo a los parámetros de la interfaz.
 - Obtienen clases de evidencia y valores límite directamente para los rangos de los parámetros definidos.
 - No pueden aportar valores de resultados esperados pero pueden ser utilizados para pruebas de robustez.
- Generadores de datos de prueba **basados en la especificación**.
 - Generan datos de prueba directamente a partir de documentos de especificación.
 - Los documentos de especificación requieren del uso de una notación **formal estricta**.
 - Los documentos generados con la ayuda de una **herramienta CASE** pueden aportar una buena base para estas herramientas.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para la ejecución de pruebas – “Drivers” y “ stubs”

- **Drivers de prueba (“test drivers”)**
 - Permiten acceder al objeto de pruebas cuando las interfaces aún no han sido implementadas.
 - Regulan la entrada de datos, salida de datos y registran (“log”) el desarrollo de la prueba.
 - Registran los resultados reales (obtenidos).
 - Pueden ser **productos estándar** o programas desarrollados para un objeto de prueba **específico**.
 - Normalmente aportan su entorno de sistemas propio.
- **Stubs**
 - Simulan la funcionalidad de un **componente** invocado.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para la ejecución de pruebas – Simuladores

- Son una replica del **entorno de producción** (o parte del mismo) y son necesarios cuando **consideraciones de seguridad** impiden el uso del entorno de producción objetivo (real).
- Principalmente son utilizados en el nivel de **pruebas de sistema y pruebas de integración** – posiblemente son utilizados también en pruebas de componente.
- La emulación del entorno de producción debe **ser tan próxima al mismo como sea posible**.

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para el análisis de pruebas y análisis del objeto de prueba

- **Herramientas de comparación (“comparison tools”)**
 - Comparan los resultados esperados y los reales (obtenidos) en base a ficheros o bases de datos de diferentes formatos.
 - Aquellos datos relevantes a comparar son seleccionados haciendo uso de funcionalidades filtro (“filter functionalities”).
- **Herramientas de análisis y cobertura (“analysis of coverage”)**
 - Se implementan contadores que registran cada acceso.
 - Tras completar las pruebas, los contadores serán utilizados para evaluar la cobertura (por ejemplo, cobertura de sentencia, cobertura de rama).
- **Herramientas de análisis dinámico (“ dynamic analysis ”)**
 - Pueden soportar las **pruebas dinámicas**.
 - Controlan y registran el estado interno del objeto de prueba, por ejemplo uso de memoria

VI – Herramientas de pruebas

01- Tipos de herramientas de pruebas

Herramientas para pruebas no funcionales

- **Herramientas para pruebas de carga (“load”) y rendimiento (“performance”).**
 - Seguimiento de comportamiento en tiempo real del objeto de prueba en distintas situaciones.
 - Las herramientas generan y ejecutan una repetición de casos de prueba dirigidos por parámetros.
 - El despliegue en entornos complejos requiere un conocimiento experto con el objeto de asegurar que los resultados son similares a las condiciones reales, por ejemplo interacción de red.
- **Supervisores de pruebas (“test monitors”)**
 - Los supervisores de pruebas analizan, verifican y documentan de forma continua el uso o recursos de sistema.
 - “observan” el comportamiento del objeto de prueba en el entorno del sistema y detecta situaciones que pudieran ser susceptibles de problemas.

VI – Herramientas de pruebas

02- Uso efectivo de herramientas de pruebas

Ventajas de las herramientas de pruebas en general

- Reducción de la necesidad de la totalidad de recursos **humanos**.
 - Asignación de actividades a las herramientas de pruebas.
 - Apoyando y acelerando tareas manuales.
- Incremento de la **calidad** de la **ejecución de pruebas**.
 - **Iteración** de actividades idénticas.
 - Las **evaluaciones automáticas** aportan medidas objetivas.
- Mayor potencial para el control de pruebas.
 - La gestión de datos con herramientas de pruebas hace posible una **mayor variedad de evaluaciones**.
 - Aporta al proceso de gestión mejor información sobre la cual basar la toma de decisiones ("**decision marking**")

VI – Herramientas de pruebas

02- Uso efectivo de herramientas de pruebas

Riesgos de las herramientas en general

- Desviaciones en la calidad de la herramienta desplegada.
 - Funcionalidad de la herramienta de pruebas no cumple con las expectativas.
 - La usabilidad de la herramienta de pruebas no cumple con las expectativas.
 - Otros requisitos de calidad no son alcanzados.
- Estimación errónea de los beneficios y costes
 - **Beneficio** ha sido sobreestimado.
 - **Costes** de adquisición, introducción u operación ha sido subestimado.
- Despliegue erróneo de la herramienta
 - Por ejemplo dejar atrás el conocimiento de los métodos de prueba.
 - Por ejemplo omitir la tarea de considerar los diferentes procedimientos / procesos de prueba y adecuación a un proyecto específico.
 - “un instrumento predictivo no es buena opción”

VI – Herramientas de pruebas

02- Uso efectivo de herramientas de pruebas

Beneficios y riesgos de las herramientas de pruebas de rendimiento

- Las herramientas de pruebas de rendimiento generalmente son utilizadas en aplicaciones (sistemas) **distribuidas** y cuya comunicación se realiza a través de **redes**.
- En la mayoría de los casos, el **entorno de pruebas no puede** estar completamente **aislado** y es objeto de influencia de factores que no son conocidos en detalle a la hora de preparar y ejecutar las pruebas,
- La complejidad del entorno puede hacer que sea **imposible repetir pruebas idénticas** (los resultados son difícilmente comparables)
- En muchos casos es necesario un **conocimiento experto** en detalle con el objeto a analizar las salidas de la herramienta de forma correcta y extraer las conclusiones correctas

VI – Herramientas de pruebas

02- Uso efectivo de herramientas de pruebas

Beneficios y riesgos de otras herramientas de pruebas

- **Herramientas de análisis estático**
 - Examinan en código fuente con el objeto de comprobar la conformidad con convenciones.
 - Con frecuencia es necesario preparar el código para el análisis estático.
 - Un problema detectado con frecuencia: una cantidad relativamente grande de indicaciones (mensajes), es difícil identificar su relevancia.
- **Herramientas de gestión de pruebas.**
 - La información debe ser mantenida abiertamente accesible.
 - Una hoja de cálculo es la herramienta más utilizada por los jefes de pruebas para evaluaciones e informes.
 - ¡Los informes y evaluaciones se deben adaptar a la organización, y no al revés!

VI – Herramientas de pruebas

03 – Introducción de herramientas de pruebas

Ventajas de un proyecto piloto para la introducción de una herramienta

- Llegar a conocer la herramienta en detalle con sus **puntos fuertes y débiles**.
- **Establecimiento de interfaces (“interfacing”)** con **otras herramientas** en uso, adaptación de procesos y flujos de trabajo.
- Definir **informes** de acuerdo con los estándares de la organización.
- Evaluar si la herramienta cumple con los **beneficios esperados**.
- Estimar si el **coste** del despliegue se encuentra dentro del alcance
- No introducir la herramienta sin el **desarrollo de un piloto** : de lo contrario esperar/contar con problemas de aceptación.

VI – Herramientas de pruebas

03 – Introducción de herramientas de pruebas

Factores de éxito en el despliegue de software

- Introducción y lanzamiento **paso a paso** en la totalidad de la **organización**, no solamente en un proyecto.
- Hacer **obligatorio** el uso de la herramienta para los flujos de trabajo/ procesos respectivos.
- Son necesarias **guías de usuario** para el despliegue de la herramienta.
- Los usuarios deben tener acceso a la **información** adecuada, debe estar disponible un soporte rápido para los usuarios.
- La **experiencia** adquirida a partir del despliegue de la herramienta debe estar disponible para **todos los usuarios**.
- El **uso en curso** de la herramienta debe ser objeto de seguimiento, de tal manera que pueda ser posible cualquier intervención para **mejorar su aceptación**.