UD8. Distribución das aplicacións

RA8. Evalúa el funcionamiento de las aplicciones para lo que diseña y ejecuta pruebas.

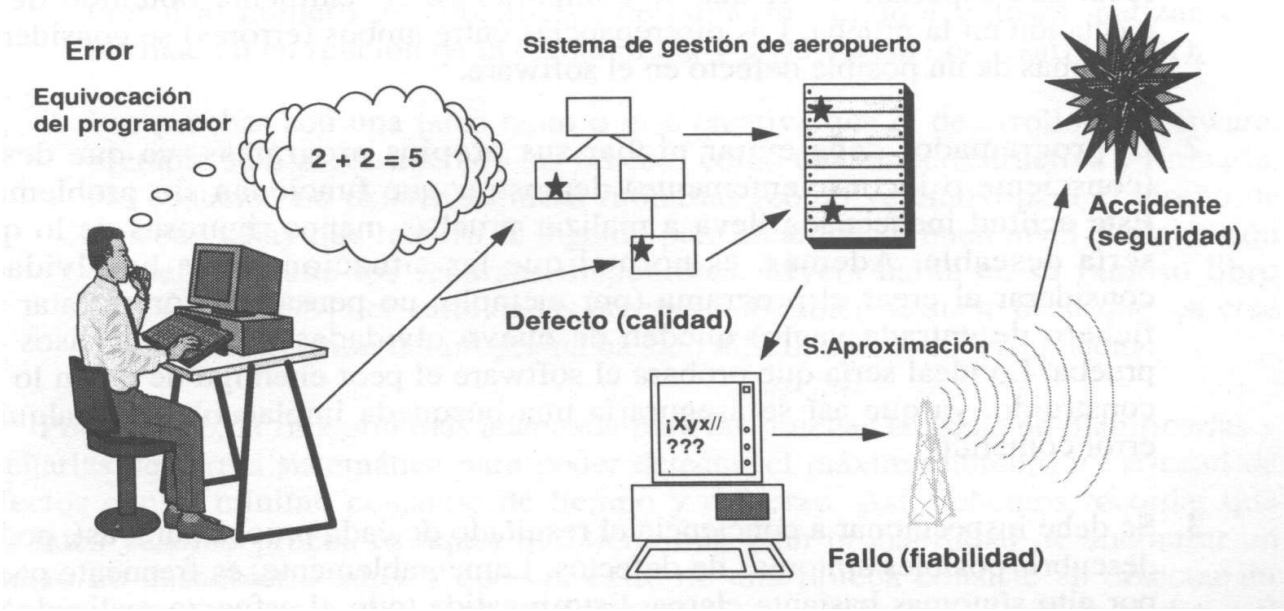
- CA3.1. Se estableció una estrategia de pruebas.
- CA3.2. Se realizaron pruebas de integración de los elementos.
- CA3.3. Se realizaron pruebas de regresión
- CA3.4. Se realizaron pruebas de volumen y estrés.
- CA3.5. Se realizaron pruebas de seguridad.
- CA3.6. Se realizaron pruebas de uso de recursos por parte de la aplicación.
- CA3.7. Se documentó una estrategia de pruebas y se analizaron los resultados obtenidos.

BC8. Realización de pruebas

- Objectivo, importancia y limitaciones del proceso de prueba. Estrategias.
- Pruebas de integración: ascendentes y descendentes.
- Pruebas de sistema: configuración, recuperación, etc.
- Pruebas de regresión.
- Pruebas de uso de recursos.
- Pruebas de seguridad.
- Pruebas manuales y automáticas. Herramientas de software para la realización de pruebas.
- Pruebas de aceptación. Versiones alfa e beta.

1. Objetivo, importancia y limitaciones del proceso de prueba. Estrategias.

Las **pruebas de software** son investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto desarrollado al futuro usuario y/o comprador. Son una actividad más en el proceso de **control de calidad.** Básicamente son un conjunto de actividades como una etapa más dentro del desarrollo de software.



Los errores en el software pueden causar víctimas

La **prueba exhaustiva del software es impracticable** (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos. Por ejemplo, en un programa sencillo de suma de dos números es imposible probar todos los casos (todos los números) a sumar. El objetivo de las pruebas **no es asegurar la ausencia de defectos** en un software, únicamente puede demostrar que existen defectos en el software.

El objetivo de las pruebas es la **detección de defectos** en el software (descubrir un error es el éxito de una prueba). El hallazgo de un error o defecto no implica que seamos malos profesionales ya que todo el mundo comete errores. Es más, **el descubrimiento de un defecto significa un éxito para la mejora de la calidad del producto.**

Por otro lado, el proceso de **verificación** es el proceso de evaluación de un sistema (o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase, es decir, responde a la pregunta: *"¿estamos construyendo el producto **correctamente**?"*

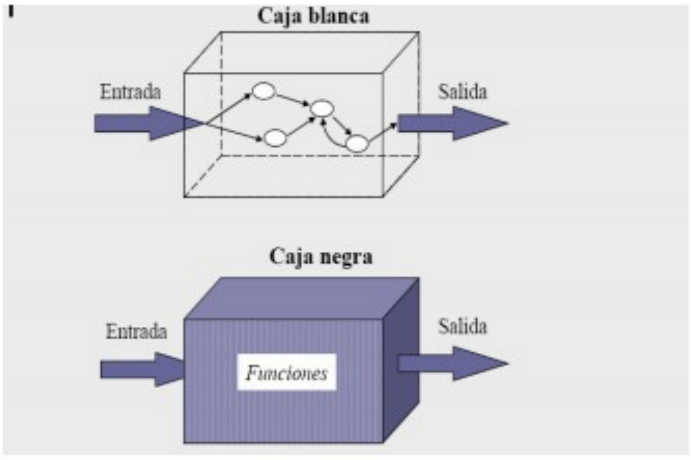
Finalmente esta el proceso de **validación** o proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los

requisitos marcados por el usuario, es decir, responde a la pregunta: "*¿estamos construyendo el producto **correcto**?"*

Veamos a continuación una serie de definiciones importantes en el campo de las pruebas de software algunas de las cuales mencionaremos en apartados posteriores:

- **Pruebas (test):** actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de uno o varios aspecto concretos.
- **Caso de prueba (test case):** conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.
- **Defecto (defect, fault, «bug»):** un defecto en el software como, por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa. Por ejemplo: *"el software es incapaz de cargar los sumandos"*
- **Fallo (failure):** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados. Por ejemplo: *"carga los datos pero no los suma".*
- **Error (error):** La diferencia entre un valor calculado, observado o medio y el valo verdadero, especificado o teóricamente correcto. Por ejemplo: *"la suma es incorrecta".*

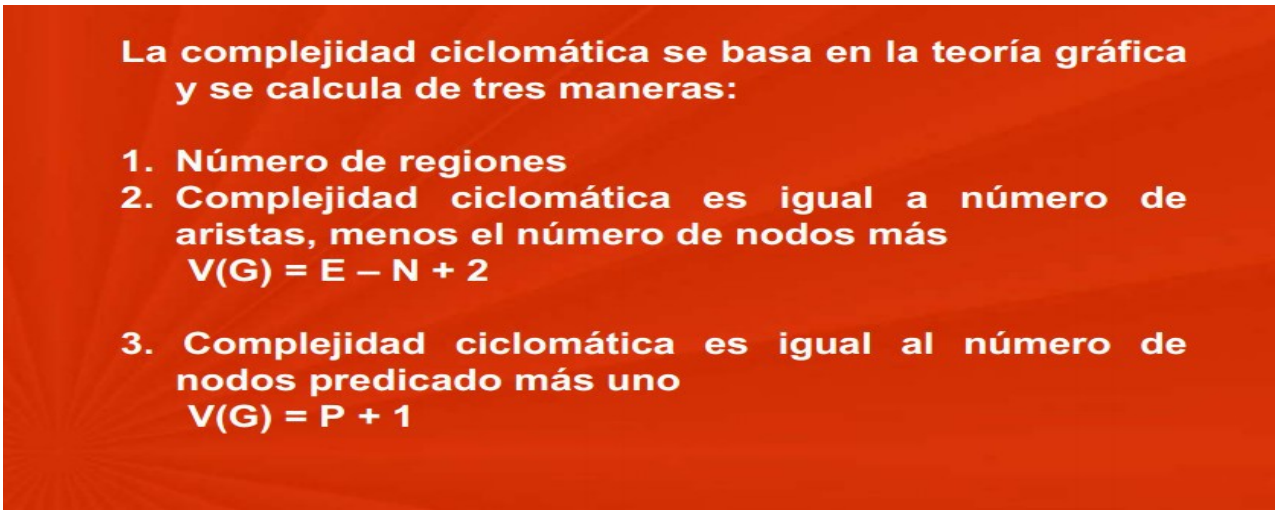
Las **pruebas de caja negra o integración** también conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba independientemente de su diseño interno o *"o de como lo haga"*.



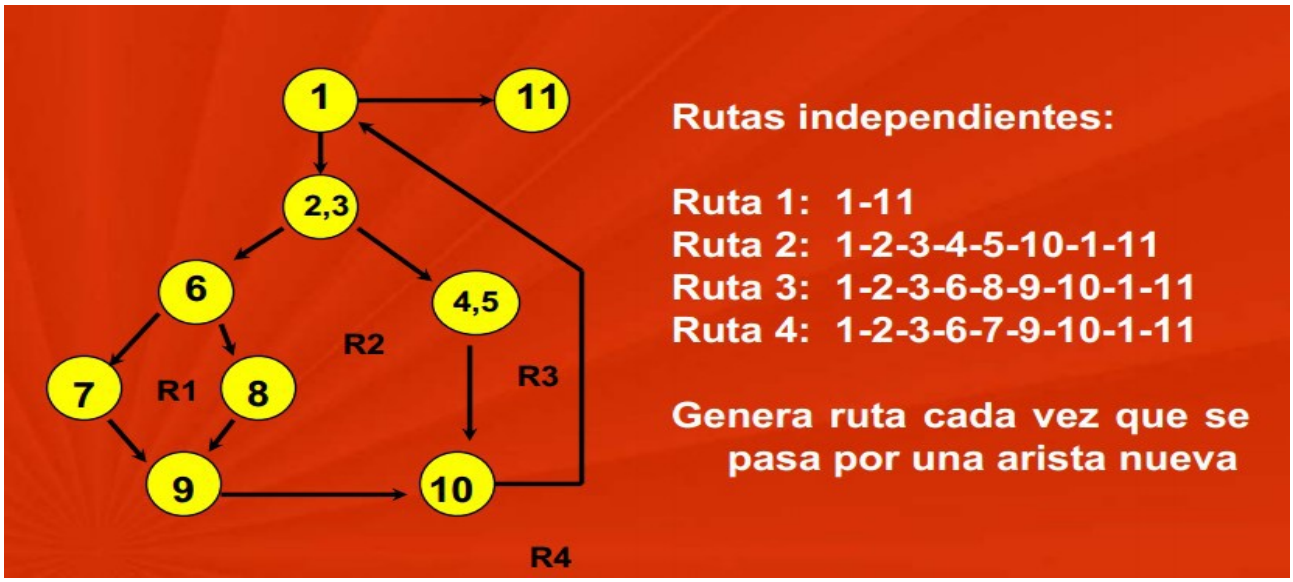
El componente se ve como una "Caja Negra" cuyo comportamiento **sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas.**

Las **pruebas de Caja Blanca o Estructurales** a este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al **comportamiento interno y la estructura del programa**, es decir, interesa **.** Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento.

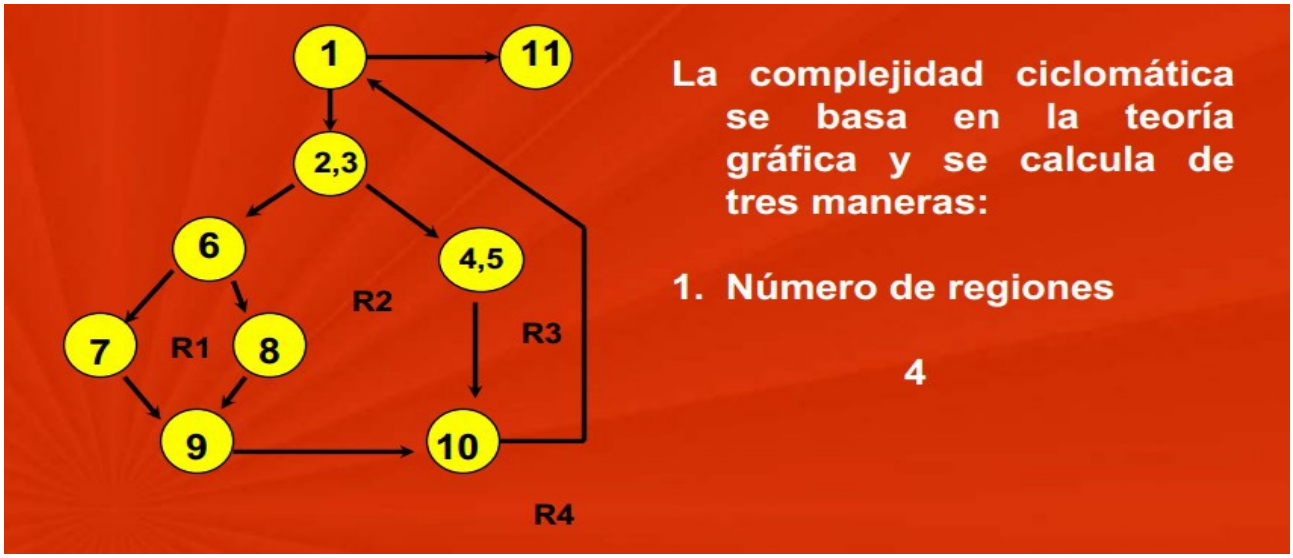
El objetivo de la técnica es **diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa**, y todas las condiciones tanto en su vertiente verdadera como falsa. De estas pruebas se obtiene la **complejidad ciclómatica de los algoritmos**. Veremos un ejemplo sencillo:



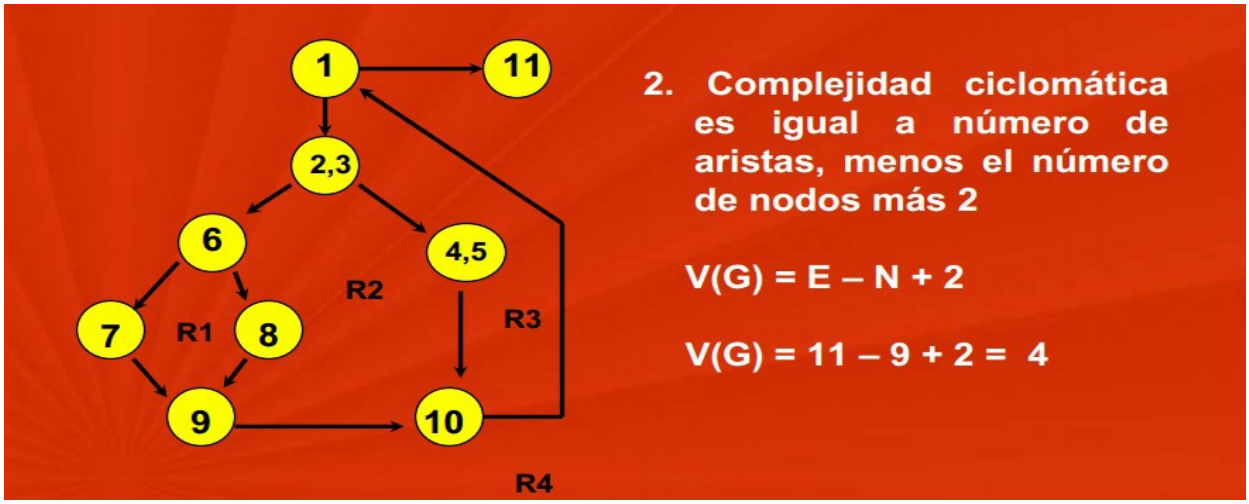
Formas de Calcular la complejidad ciclomática



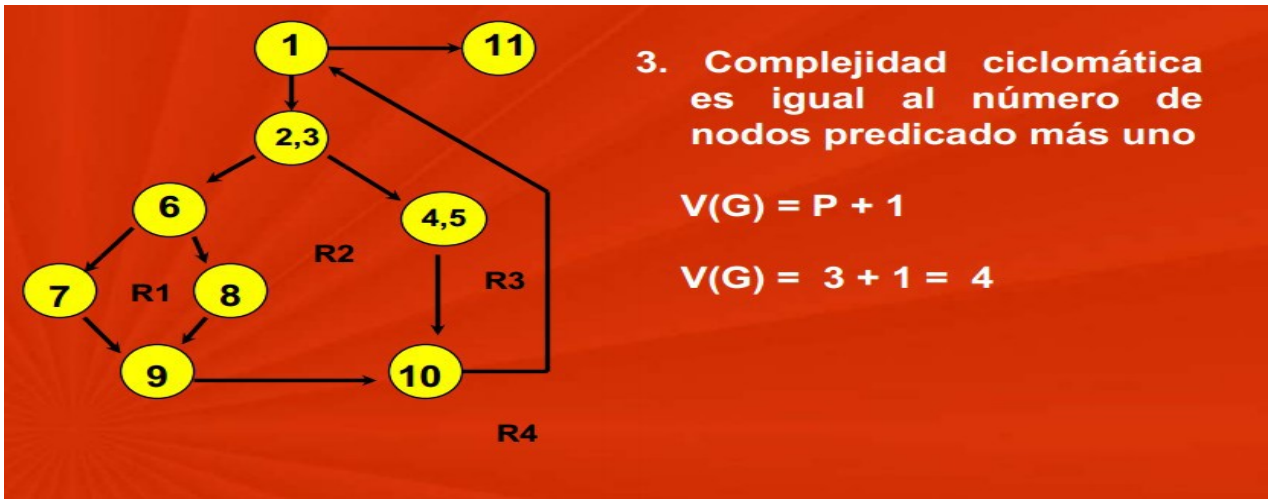
Calculamos las diferentes rutas.



Calculamos el número de regiones



Forma de Calcular la V(G) (I)



Forma de Calcular la V(G) (II)

Para la realización con éxito de unas pruebas sobre el software he aquí una serie de consejos o **recomendaciones**:

- Cada caso de prueba debe definir el **resultado de salida esperado** que se comparará con el realmente obtenido.
- El programador **debe evitar probar sus propios programas,** ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas. Además, es normal que las situaciones que olvidó considerar al crear el programa queden de nuevo olvidados al crear los casos de prueba.
- Se debe **inspeccionar a conciencia** el resultado de cada prueba, así, poder descubrir posibles síntomas de defectos.
- Al generar casos de prueba, se deben incluir tanto **datos de entrada válidos y esperados como no válidos e inesperados**
- Las pruebas deben centrarse en dos objetivos:
  - Se deben **evitar los casos desechables**, es decir, los no documentados ni diseñados con cuidado.
  - No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas siempre hay defectos
- Probar **si el software no hace lo que debe hacer**
- Probar **si el software hace lo que debe hacer**, es decir, si provoca efectos secundarios adversos.
- La experiencia parece indicar que **donde hay un defecto hay otros,** es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.
- Las pruebas **son una tarea tanto o más creativa** que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.
- Es interesante planificar y diseñar las pruebas para poder detectar el máximo número y variedad de defectos con el **mínimo consumo de tiempo y esfuerzo**.

Las tareas a realizar para probar un software según el orden establecido son:

- **Diseño de las pruebas.** Esto es, identificación de la técnica o técnicas de pruebas que se utilizarán para probar el software. Distintas técnicas de prueba ejercitan diferentes criterios como guía para realizar las pruebas. Seguidamente veremos algunas de estas técnicas.

- **Generación de los casos de prueba.** Los casos de prueba representan los datos que se utilizarán como entrada para ejecutar el software a probar. Más concretamente los casos de prueba determinan un conjunto de entradas, condiciones de ejecución y resultados esperados para un objetivo particular. Como veremos posteriormente, cada técnica de pruebas proporciona unos criterios distintos para generar estos casos o datos de prueba.
- **Definición de los procedimientos de la prueba.** Esto es, especificación de cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, …
- **Ejecución de la prueba**, aplicando los casos de prueba generados previamente e identificando los posibles fallos producidos al comparar los resultados esperados con los resultados obtenidos.
- **Realización de un informe de la prueba,** con el resultado de la ejecución de las pruebas, qué casos de prueba pasaron satisfactoriamente, cuáles no, y qué fallos se detectaron.

2. Probas de integración: ascendentes e descendentes.

Aún cuando **los módulos de un programa funcionen correctamente** por separado es necesario probarlos conjuntamente. Un módulo puede tener un efecto adverso o inadvertido sobre otro módulo; las subfunciones, cuando se combinan, pueden producir la función principal o un resultado indeseado; la imprecisión aceptada individuamente puede crecer hasta niveles inaceptables al combinar los modulos.

Por lo tanto, es necesario probar el software ensamblando todos los módulos probados previamente. Y este **es el objetivo de la pruebas de integración.**

A menudo hay una tendencia a intentar una **integración no incremental**; es decir, a combinar todos los módulos y probar todo el programa en su conjunto. El resultado puede ser un poco caótico con un gran conjunto de fallos y la consiguiente dificultad para identificar el módulo (o módulos) que los provocó.

Por ello y, a veces, se puede aplicar la **integración incremental** en la que el programa se prueba en pequeñas porciones en las que los fallos son más fáciles de detectar. Existen dos tipos de integraciones incrementales, la denominada **ascendente y descendente**.

- **Integración incremental ascendente:**

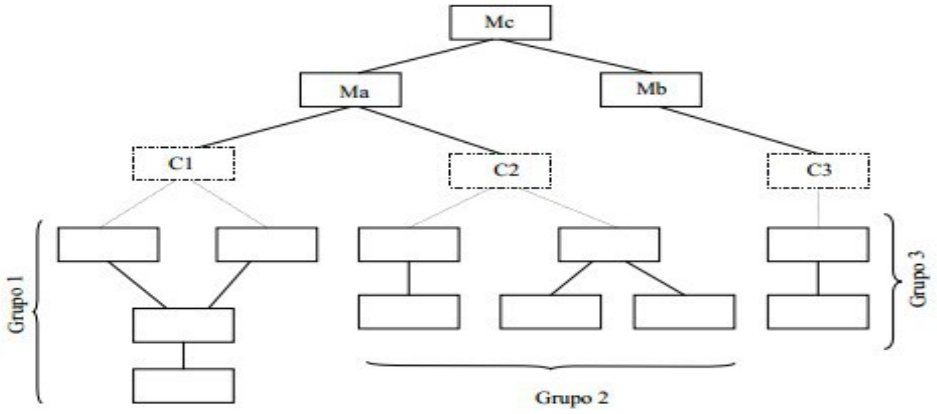
  a. Se combinan los módulos de bajo nivel en grupos que realicen una subfunción específica.

  b. Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y salida de los casos de prueba.

  c. Se prueba el grupo

d. Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

En la figura de ejemplo, se forman los grupos 1, 2 y 3 de módulos relacionados, y cada uno de estos grupos se prueba con el controlador C1, C2 y C3 respectivamente. Seguidamente, los grupos 1 y 2 son subordinados de Ma, luego se eliminan los controladores correspondientes y se prueban los grupos directamente con Ma. Análogamente se procede con el grupo 3 eliminando el controlador C3 y probando el grupo directamente con Mb. Tanto Ma y Mb se integran finalmente con el módulo Mc y así sucesivamente.



- **Integración incremental descendente**

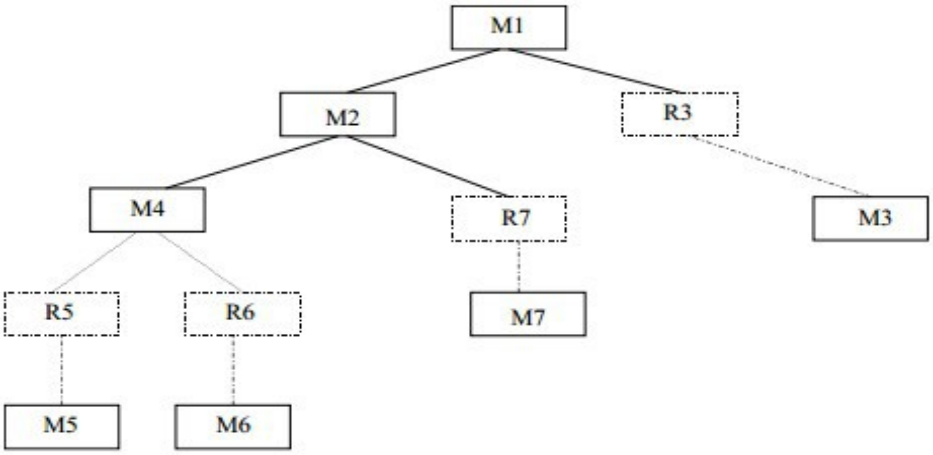
  a. Se usa el **módulo de control principal** como controlador de la prueba, creando resguardos (módulos que simulan el funcionamiento de los módulos que utiliza el que está probando) para todos los módulos directamente subordinados al módulo de control principal.

  b. Dependiendo del enfoque e integración elegido (es decir, primero-en-profundidad, o primero-en-anchura) se van sustituyendo uno a uno los resguardos subordinados por los módulos reales.

  c. Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.

  d. Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.

En la figura siguiente vemos una prueba de integración descendente. Supongamos que se selecciona una integración descendiente por profundidad, y que por ejemplo se prueba M1, M2 y M4. Sería entonces necesario preparar resguardos para M5 y M6, y para M7 y M3. Estos resguardos se ha representado en la figura como R5, R6, R7 y R4 respectivamente. Una vez realizada esta primera prueba se sustituiría R5 por M5, seguidamente R6 por M6, y así sucesivamente hasta probar todos los módulos.



3. Pruebas de sistema

Este tipo de pruebas tiene como propósito probar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, etc.) y que realizan las funciones adecuadas. Concretamente se debe comprobar que:

- Se cumplen los requisitos funcionales establecidos.
- El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- La adecuación de la documentación de usuario.
- Rendimiento y respuesta en condiciones límite y de sobrecarga.

Para la generación de casos de prueba de sistema se utilizan técnicas de **caja negra**. Este tipo de pruebas se suelen hacer inicialmente en el **entorno del desarrollador**, denominadas **Pruebas Alfa,** y seguidamente en el **entorno del cliente** denominadas **Pruebas Beta.**

Se distinguen los siguientes tipos de pruebas (algunas solo se mencionaran porque su nombre es autoexplicativo):

- **Pruebas de comunicaciones.** Comprueba que las interfaces tanto locales como remotas funcionan adecudamente.
- **Pruebas de rendimiento.** Comprueba los tiempos de respuesta de la aplicación.
- **Pruebas de recuperación.** Se fuerza el fallo del software para comprobar su capacidad de recuperación.
- **Pruebas de volumen.** Se comprueba su funcionamiento con cantidades próximas a su capacidad total de procesamiento
- Pruebas de sobrecarga. Similar al anterior pero en el límite de su capacidad.
- **Pruebas de tensión.** Es similar a la prueba de volumen pero se restringe el tiempo disponible (algo como determinar la velocidad de trabajo)

- **Pruebas de disponibilidad de datos.** Comprueba si tras la recuperación el sistema mantuvo la integridad de los datos.
- **Pruebas de facilidad de uso.** Refiriéndose al usuario final.
- **Pruebas de operación.** Comprueba la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y rearranque del sistema, etc...
- **Pruebas de entorno.** Comprueba la interacción con otros sistemas.
- **Pruebas de seguridad.** Comprobación de los mecanismos de control de acceso al sistema.
- **Pruebas de usabilidad.** Ya visto en la UD 4.
- **Pruebas de almacenamiento.** Comprobación de la cantidad de memoria principal y secundaria que el programa usa y el tamaño de los archivos temporales.
- **Pruebas de configuración.** Aunque a veces resulta imposible se debe comprobar el progrma con cada tipo de hardware, sistema operativo, antivirus....
- **Pruebas de instalación.** En especial la automatización para facilidad del usuario final.
- **Pruebas de la documentación.** Hace referencia tanto a la documentación técnica para desarrolladores futuros como a la documentación para el usuario.

4. Pruebas de regresión

Se denominan **pruebas de regresión** a cualquier tipo de pruebas que intentan descubrir las causas de nuevos errores o bugs, carencias de funcionalidad, o divergencias funcionales con **respecto al comportamiento esperado del software**, inducidos **por cambios recientemente realizados en partes de la aplicación** que anteriormente al citado cambio no eran propensas a este tipo de error. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio o modificación realizada en el programa para su mejor o adaptación a nuevas necesidades del cliente.

Este tipo de cambio puede ser debido a prácticas no adecuadas del **control de las versiones,** falta de consideración acerca del ámbito o contexto de producción final y extensibilidad del error que fue corregido (fragilidad de la corrección), o simplemente una consecuencia del rediseño de la aplicación. El **objetivo de las pruebas de regresión es eliminar el efecto onda**, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un

comportamiento no deseado o errores adicionales en otros componentes no modificados

Por lo tanto, en la mayoría de las situaciones del desarrollo de software se considera una buena práctica que cuando se localiza y corrige un bug, se grabe una prueba que exponga el bug y se vuelvan a probar regularmente después de los cambios subsiguientes que experimente el programa.

Existen herramientas de software que permiten detectar este tipo de errores de manera parcial o totalmente automatizada. La práctica habitual en programación es que este tipo de pruebas se ejecuten en cada uno de los pasos del **ciclo de vida del desarrollo del software.**

- **Tipos de regresión**
  - **Clasificación de ámbito**
    - **Local:** los cambios introducen nuevos errores.
    - **Desenmascarada:** los cambios revelan errores previos.
    - **Remota:** Los cambios vinculan algunas partes del programa (módulo) e introducen errores en ella.
  - **Clasificación temporal**
    - **Nueva característica:** los cambios realizados con respecto a nuevas funcionalidades en la versión introducen errores en otras novedades en la misma versión del software.
    - **Característica preexistente:** los cambios realizados con respecto a nuevas funcionalidades introducen errores en funcionalidad existente de previas versiones.

Para mitigar los riesgos en las modificaciones realizadas en los módulos del programa ya desarrollados:

- **Repetición completa** y habitual de la batería de pruebas, manual o mediante automatización.
- **Repetición parcial** basada en trazabilidad y análisis de los posibles riesgos.
- **Pruebas de cliente** o usuario:
  - **Beta** - distribución a clientes potenciales y actuales de las **versiones Beta.**
  - **Pilot** - distribución a un subconjunto bien definido y localizado.
  - **Paralela** - simultaneando uso de ambos sistemas. Usar releases **(software candidato**

a definitivo) mayores. Probar nuevas funciones a menudo cubre las funciones existentes. Cuantas más nuevas características haya en un release, habrá mayor nivel de pruebas de regresión "accidental".

- **Parches de emergencia -** estos parches se publican inmediatamente, y serán incluidos en releases de mantenimiento futuras.

Las Pruebas de Regresión pueden usarse no solo para probar la **corrección** de un programa, sino a menudo usarse para rastrear la calidad de su salida. Por ejemplo, en el diseño de un compilador, las pruebas de regresión deben rastrear el tamaño del código, tiempo de simulación, y el tiempo de compilación de las suites de prueba. Cuando quiera que aparece un nuevo build, el proceso de regresión aparece.

5. Pruebas de uso de recursos

Esta relacionado con las **pruebas de rendimiento**. Estas son las pruebas que se realizan para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. Dentro de ellas están las pruebas del uso de los recursos preferentemente hardware, red, sistema operativo...

Por ejemplo, puede ser el número de sesiones que un servidor de aplicaciones puede soportar o el número de acceso que un servidor de datos es capaz de transferir llegándose al máximo o **pruebas de stress** con el objetivo de que la aplicación falle en esas condiciones.

6. Pruebas de seguridad

La prueba de seguridad intenta verificar que los mecanismos de protección incorporados en el sistema. Estas pruebas van desde la introducción de datos incorrectos o inapropiados, intentos de acceso no autorizados, pruebas de control de la integridad de los datos, etc...

7. Pruebas manuales y automáticas. Herramientas de software para la realización de pruebas.

La **prueba manual** se realiza ejecutando el sofware, introduciendo datos de entrada e inspeccionado los de salida. El proceso es sencillo y no hace falta aprendizaje previo, ademas ejecutamos las pruebas exactamente como el usuario final pero tiene algunos inconvenientes.

- **Repetitiva**. Cada vez que se cambia o añade una nueva funcionalidad, o se corrige un error, necesitamos “reprobar” las aplicaciones para asegurarnos de que no se ha producido un error por causa de este cambio.

- **Susceptible de error.** Precisamente por ser repetitiva, es posible que pasemos por alto detalles a la hora de probar, o que no se prueben funcionalidades aparentemente no relacionadas con el cambio, pero realmente afectadas. Es imposible conocer todas las interrelaciones existentes entre las funcionalidades de nuestro software.
- **Solo pruebas lo que ves.** Esto significa que solo pruebas indirectamente lo que no ves, y es especialmente el caso de las aplicaciones de la parte servidor. Necesitamos que funcionen correctamente la capa de presentación y la capa de negocio, pero especialmente la capa de negocio residente en el servidor.
- **Con la prueba manual,** otros no pueden verificar el correcto funcionamiento de nuestra aplicación. Otras personas deben aceptar que nuestra prueba manual ha sido correcta y satisfactoria, o realizar ellos mismos una nueva prueba manual, que quizá no estan preparados para realizar por no conocer la lógica de nuestro software.

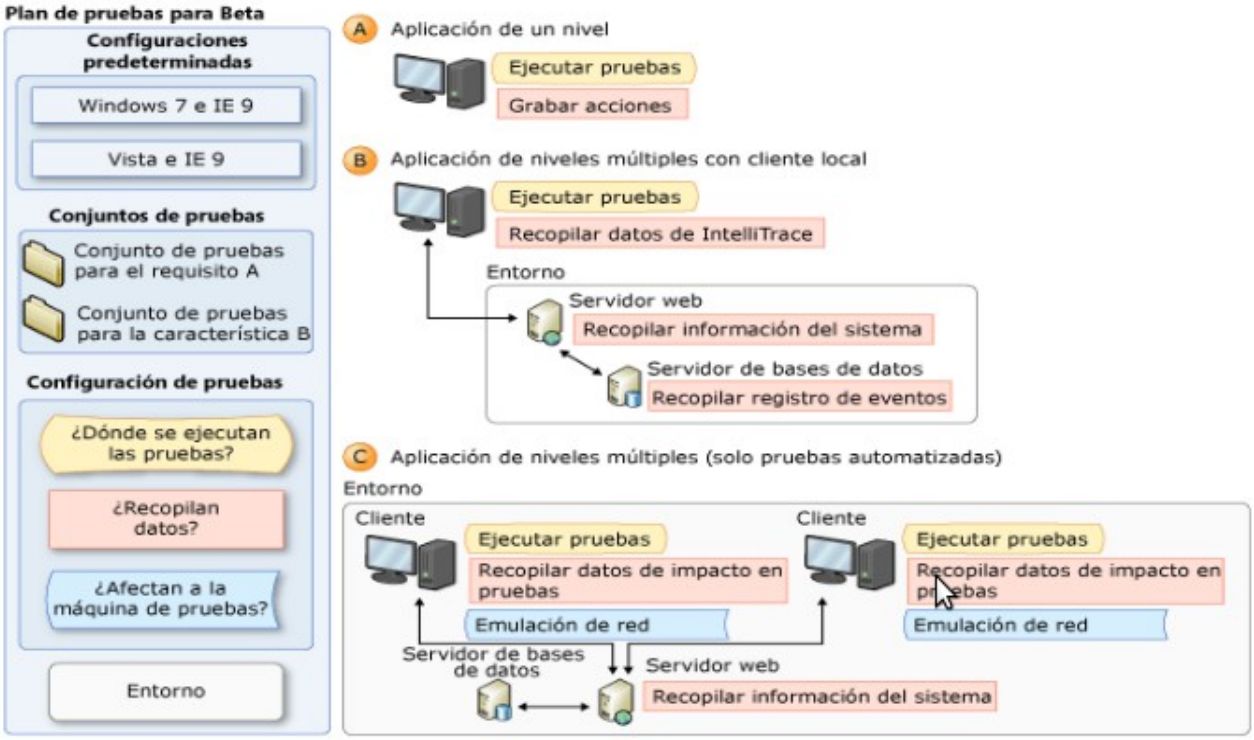
La **prueba automatizada** se integra en las metodologias modernas de una manera integral en el desarrollo del software. La prueba automatizada es repetible y portable. Repetible significa que la prueba se puede repetir indefinidamente produciendo los mismos resultados siempre, y portable se refiere a que otras personas pueden ejecutar la prueba y verificar que el software pasa todas las pruebas. Por otro lado, la prueba evoluciona con el software, de forma que si los requisitos y funcionalidad del software cambian, la prueba debe cambiar también. No es muy adecuada una prueba que no verifica el correcto funcionamiento del software en su estado actual.

Realizando pruebas automáticas, podemos tener dos tipos: **prueba funcional y prueba unitaria**.

La **prueba funcional es de tipo “caja negra”** realizada sin conocimiento interno de la aplicación, a alto nivel, simulando la actuación del usuario.

La **prueba unitaria**, por el contrario, implica un conocimiento del software a bajo nivel, no solo las entradas y las salidas. Se deben probar todos los metodos y clases importantes, e implica simular las entradas de información. Proporciona también soluciones a los problemas antes planteados:

- **Elimina el componente repetitivo.** Se transmite el trabajo repetitivo al ordenador (para eso está), que seguira exactamente la secuencia de pruebas, cada vez que lo solicitemos.
- **Reduce los errores.** Cada repetición será exactamente igual cada vez que se ejecute, de forma que se puede probar facilmente todas las funcionalidades ante cualquier cambio sin temor a olvidar alguna parte.
- **Permite probar las partes no visibles de codigo,** el corazón de la aplicación realmente. El nivel de detalle depende de la programación de la prueba.
- **Permite a los usuarios finales** verificar que el software funciona como debe.



Ejemplo de esquema de prueba para aplicación en Microsot Visual Studio

Entre las herramientas que tenemos para las pruebas automatizadas de software (solo freeware):

- **Para el seguimiento de defectos:** *Bugzilla, BugRat.* Realmente son bases de datos que van almacenando los defectos encontrados en las diferentes versiones
- **Para evaluación de pruebas de carga y rendimiento:** *Jmeter* (especifíca para aplicaciones web creada por Apache Foundation)
- **Para la gestión y manejo de pruebas:** *rth, QaTraq*. Ambas almacenan los resultados de las diferentes tipos de pruebas.
- **Para pruebas de unidad:** *JUnit.* Muy enfocadasa las pruebas de regresión).

IBM tiene el **Rational Software de IBM que contiene diferentes módulos según el tipo de pruebas** que puede considerarse como la herramienta más avanzada para este campo.

8. Pruebas de aceptación. Versiones alfa y beta.

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y **permitir al usuario de dicho sistema que determine su aceptación**, desde el punto de vista de su funcionalidad y rendimiento.

Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de

desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

Cuando se construye software a medida para un cliente, se lleva a cabo una serie de **pruebas de aceptación** para permitir que el cliente valide todos los requisitos. La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

- **Prueba alfa***:* **(versión alfa)** se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.
- **Prueba beta:** (**versión beta)** se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

Bibliografía