

DI08.- Realización de pruebas.

1.- Pruebas de software.

En múltiples ocasiones, a lo largo de los distintos módulos que componen el ciclo, se comenta que la **calidad del software** que produzcamos es un requisito indispensable.

La calidad no se da a la aplicación una vez está construida, sino que es una forma de pensar y actuar del programador a lo largo de todas las fases del proyecto.

Calidad no es sólo que el software funcione bien y que vaya rápido, sino que además implica que satisfaga las expectativas de los clientes, **usabilidad**, **confiabilidad**, coste, **eficiencia** y cumplimiento de los estándares.

Para asegurar la calidad, debemos hacer uso de las **pruebas de software**.

Asegurar los requisitos de calidad de software lleva un coste y trabajo asociado.



[\(link: DI08 CONT R03 calidad.png.\)](#)

La realización de pruebas de software consume tiempo y recursos, pero es fundamental para que los fallos de la aplicación sean mínimos.

Serán los usuarios clientes de nuestro proyecto quienes impongan los requisitos y los que determinen si se ha conseguido o no nuestro objetivo de proporcionar un software fiable, correcto y de calidad.

Es imposible que los desarrolladores de software podamos saber de antemano, de forma total y absoluta, las necesidades que los clientes pueden tener en cada momento y puede ocurrir que a veces malinterpretemos ciertos requisitos. Es por ello que, como fase final en la estrategia de pruebas, el software termine siendo utilizado por los clientes durante un período de prueba, para que sean ellos quienes concluyan si nuestra aplicación es la que ellos demandaron en un principio o si es necesario aplicar alguna modificación.

La ausencia o la mala gestión de una estrategia de pruebas en un proyecto software pueden suponer:

- ✓ Falta de satisfacción de los clientes.
- ✓ Fallos en la **explotación** por defectos no detectados y eliminados a tiempo.
- ✓ Altos costes finales, por el tiempo empleado en corregir defectos.

1.1.- Objetivos.

Caso práctico

"No hace falta ser un experto para saber que tenemos que encontrar todos los defectos, "comenta **Antonio**". "Te equivocas", le corrige **María**. Por su propia naturaleza, el software siempre tendrá defectos. Nuestro objetivo es minimizarlos.



El objetivo de las pruebas al software es encontrar la mayor cantidad posible de errores.

Todas las aplicaciones que se construyen tienen defectos. El origen de los mismos puede ser de distinta naturaleza, pero tienen un denominador común: hay que detectarlos y eliminarlos.

En todo proyecto software se debe perseguir que los errores sean mínimos (exigir la no existencia de errores en un proyecto software es una utopía) y que el proceso de encontrarlos y corregirlos lleve asociado el mínimo coste posible.

Por ello, es imprescindible desarrollar la mejor estrategia de pruebas en nuestro proyecto.



[.\(link: DI08_CONT_R05_objetivos.png.\)](#)

Descubrir un error se considera el éxito de una prueba.

No sólo basta con "ejecutar código", sino que hay que hacerlo mediante requisitos, especificaciones funcionales, diseños, etc. Es decir, hay que verificar y validar el software.

El objetivo inmediato es evitar que los defectos se propaguen de unas fases a otras hasta llegar a los sistemas de producción.

En definitiva, buscamos **reducir** los riesgos por la aparición de defectos en los procesos de implantación y explotación de las aplicaciones. Todo ello mediante la detección temprana de problemas y errores. Con todo esto se pretende desarrollar un nivel de confianza sobre la **disponibilidad** (en tiempo) y la **fiabilidad** (a lo largo del tiempo) de los productos desarrollados, acorde a las expectativas del cliente.

Las pruebas deben entenderse como:

- ✓ **Verificación:** El producto que se está construyendo funciona correctamente; es decir, es capaz de realizar la tarea para la cual ha sido diseñado.
- ✓ **Validación:** El producto terminado, además de ser correcto, es conforme con lo que el cliente había esperado.

1.2.- Importancia.

El único recurso de que disponemos los desarrolladores de software para determinar y acreditar el nivel de calidad de nuestros productos software es el proceso de pruebas.

En este proceso se ejecutan pruebas dirigidas a los componentes software individualmente y al software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requisitos impuestos.

Cuanto más tarde es encontrado un defecto más cuesta (en tiempo, recursos y dinero) su corrección.

Por tanto, hemos de intentar identificar y resolver los defectos tan pronto como son originados con la mejora de los procesos abiertos.

Como resultado, entregaremos aplicaciones de alta calidad en tiempo cobertura y costes, en línea con lo establecido.

Por tanto, la estrategia de pruebas contribuye al ahorro de coste del proyecto.

Una estrategia buena de pruebas contribuye a crear un software de mayor calidad. En concreto, contribuye a:

- ✓ Mejorar la satisfacción de la clientela.
- ✓ Incrementar la productividad.
- ✓ Disminuir los riesgos del negocio.
- ✓ Facilitar el crecimiento del negocio.



Una correcta gestión de pruebas implica una correcta gestión de la calidad.

En el caso de aplicaciones web, es a veces habitual sacar al mercado el software lo antes posible (para que no nos “pisen” nuestros competidores). Los errores pasan a un segundo plano en la primera versión y será en versiones posteriores cuando se ofrezca la aplicación mejorada. Este modelo tiene, lógicamente, el defecto de perder calidad de la marca de tu empresa y los clientes pueden desconfiar y recordarte por lo mala que fue tu primera versión.

Los defectos se introducen, en su mayoría, durante la fase de análisis de requisitos y diseño pero suelen ser detectadas durante las pruebas de aceptación y en la explotación (que son las fases donde resulta más costosa económicamente su reparación).

1.3.- Estrategias de pruebas.

Normalmente, un proyecto software será sometido a una estrategia de pruebas. La estrategia utilizada será la que mejor se adecúe a nuestro proyecto.

Todos los desarrollos de software tienen una característica en común: lo desarrollan personas. Las personas cometemos errores y cambiamos de ideas, por eso hay que probar y volver a probar.

El coste económico relativo a un error se multiplica por un factor de 1 si se comete en la fase de análisis de requisitos, por 3-6 si es en la fase de diseño, por 10 si es en la codificación y por 40-50 si es en la fase de pruebas.

Es muy importante contar con el apoyo de la dirección del proyecto y, en su caso, por la dirección de la empresa. Deben formar parte del presupuesto, la planificación y la asignación de recursos del proyecto.

Preguntas para determinar la estrategia de pruebas a seguir: ¿Cuántos niveles de pruebas serán necesarios planificar? ¿Cuándo será conveniente parar? ¿Será mejor automatizar las tareas? Etc. Por ello, **el objetivo de la estrategia de pruebas es determinar los tipos de pruebas a realizar, el calendario de las mismas, los responsables, los recursos asignados y el alcance de las pruebas.**

Además, será necesario conocer:

- ✓ La complejidad de la aplicación y de los módulos que la forman.
- ✓ Plataforma en la que va a funcionar la aplicación.
- ✓ Normativa legal pertinente.
- ✓ Conocimientos y experiencia de los responsables de realización de las pruebas.

Reflexiona

El coste medio de la fase de pruebas está en torno al 40-50 % del coste total del proyecto.

La estrategia de pruebas es un proceso complejo que consta de tres partes. En la siguiente animación encontrarás un breve resumen de las mismas.

[Resumen textual alternativo \(link: *DI08 Descripcion Presentacion PruebasSoftware.html* \)](#)

Hay que realizar una evaluación continua del producto software a lo largo del ciclo de vida completo de su desarrollo, lo cual requiere: planificación, control y seguimiento, análisis y mejora.

1.4.- Limitaciones del proceso.

Resulta que, por su propia naturaleza, no es posible probar completamente los productos software, tanto por su complejidad como por su coste.

Por tanto, el propósito más importante es el de reducir, mediante la detección temprana de problemas, los riesgos derivados de la aparición de los posibles defectos en los procesos de implantación y explotación.

En la práctica se considera que, en software, el conjunto completo de casos de prueba es teóricamente infinito. El número de pruebas a realizar implicará un equilibrio entre recursos y tiempo.

Según los datos y tal como demuestra la experiencia, resulta que por muy cuidadosos que hayamos sido en la estrategia de pruebas siempre habrá una serie de errores que sólo aparecen cuando el cliente comienza a usarlo.

Debemos tener claro que "el cliente siempre lleva la razón" y, por esta razón, después de un cierto tiempo en el que el cliente haya estado usando nuestra aplicación y descubra defectos, o que el programa no realice lo que se esperaba (o en la forma en que se esperaba), debemos revisar la aplicación y corregir esas taras.



Es por ello que las pruebas de aceptación, como veremos más adelante, tienen gran importancia y pueden llevarse a cabo durante incluso semanas después de haber entregado el proyecto al cliente.

Las empresas se sorprenden de que no se pueda estimar de forma realista el coste de sus proyectos. El coste asociado al proceso de pruebas no es fácilmente estimable, puesto que no hay dos proyectos iguales.

El software no puede ser 100 % libre de errores. La cuestión más difícil en la realización de pruebas es estimar el momento en que debemos parar.

Por propia definición, un proyecto de software siempre tendrá defectos y nuestra principal finalidad es minimizarlos. Para conseguirlo, hay que tener en cuenta una serie de principios:

- a. Siempre hay falta de conocimiento del estado real de las aplicaciones durante el desarrollo de los proyectos. (Este es el motivo principal por el que nunca sabremos el coste asociado a la estrategia de pruebas).
- b. Tener buenos conocimientos técnicos no son una garantía de éxito de un proyecto.
- c. Hay que seguir procedimientos y actividades estandarizadas que nos guíen durante el desarrollo del proyecto y nos permitan realizar los mismos de forma repetitiva.

2.- Tipos de Pruebas.

Normalmente un sistema es sometido a una serie de pruebas para determinar que el software cumple la función para la cual ha sido construido y desarrollado. Según la bibliografía que consultes, encontrarás diferentes criterios a la hora de clasificar los distintos tipos de pruebas al software. La clasificación más habitual es la que comprende las siguientes de la siguiente figura.

A lo largo de la presente unidad, vas a ir viendo qué objetivos se persiguen en cada tipo de pruebas y cómo se debe planificar para que los resultados sean lo más eficientes posibles.



(link: DI08 CONT R11 tipos2.png.)

En la siguiente animación podrás ver un adelanto del resto de la unidad: Aspectos más destacados de las diferentes pruebas al software.

[Resumen textual alternativo \(link: D108_Descripcion_Presentacion_TiposPruebasSoftware.html\)](#).

Caso práctico

Antonio, que está estudiando el ciclo a distancia, sabe que hay que probar cada módulo del programa, de forma independiente, y comprobar que no tiene errores.

"¿Eso es todo?" Le pregunta **María** . ¿La relación entre varios módulos puede dar errores aunque los módulos, probados de uno en uno, no los hayan dado?



Las **pruebas de integración** son aquellas que se realizan una vez probado el funcionamiento de las partes o módulos que componen la aplicación por separado, es decir, una vez concluidas las pruebas unitarias.

Consiste en verificar que el software, en conjunto, cumple su misión y se realiza sobre la unión de todos los módulos a la vez, para probar que la interrelación entre ellos no da lugar a ningún error o defecto.

Son muy importantes por el hecho de que los módulos que componen una aplicación suelen fallar cuando trabajan de forma conjunta, aunque previamente se haya demostrado que trabajan bien de manera individual.

La integración recomendable es la llamada incremental, en la cual se van añadiendo nuevos módulos cada vez en la realización de las pruebas (y no todos de golpe). El nuevo módulo incorporado se debe probar con el conjunto de módulos que ya han sido probados.



La prueba de integración verifica la interacción entre componentes del sistema

Dependiendo de la filosofía de integración, se definen dos técnicas: **ascendentes y descendentes** .

2.1.1.- Ascendentes.

Es un tipo de estrategia de pruebas de integración en el cual se comienza por los módulos de más bajo nivel hasta llegar al programa principal. Es decir, se prueba la relación entre pocos módulos y, si es correcta, se va ascendiendo de manera que cada vez habrá más módulos involucrados, llegando finalmente a probar la aplicación en su totalidad. La integración es de abajo a arriba.

[Resumen textual alternativo \(link: *DI08_Descripcion_Presentacion_EstrategiaAscendente.html*\).](#)

El principal inconveniente es que tenemos una gran incertidumbre hasta el final de la prueba, ya que el programa en sí no existirá hasta que no se añada el último módulo. Como ventaja, no existe la necesidad de operar con **resguardos**.



A modo de resumen, las características más importantes de la estrategia incremental ascendente son:

- ✓ Se combinan los módulos de bajo nivel en grupos con una función similar.
- ✓ Se controlan las entradas en esos módulos de bajo nivel.
- ✓ Se prueba todo el grupo.
- ✓ Se reemplazan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura de módulos hasta llegar al programa principal.

2.1.2.- Descendentes.

Es la otra estrategia de las pruebas de integración.

([link: DI08_CONT_R15_desc.png.](#))

Consiste en integrar los módulos moviéndose de arriba abajo por la jerarquía, comenzando desde el esqueleto del programa principal, hasta ir incorporando los módulos subordinados de forma incremental. Tras la incorporación de cada módulo, habrá que realizar la prueba de integración. El programa principal actuará como coordinador de la prueba.



La integración es, por tanto, de arriba abajo.

Ventaja: Se prueban antes los módulos más importantes. Como desventaja: es necesario trabajar con resguardos (con las complicaciones que eso conlleva).

A modo de resumen, la estrategia incremental descendente comprende los siguientes pasos:

- ✓ Se usa el módulo principal como controlador de la prueba, disponiendo de resguardos para todos los módulos subordinados.
- ✓ Se van sustituyendo los resguardos subordinados, uno a uno, por los módulos reales.
- ✓ Cada vez que se integra un módulo se realizan las pruebas.
- ✓ Tras terminar con éxito las pruebas, se reemplaza otro resguardo por otro módulo real.

En la siguiente animación podrás ver, de forma gráfica, la realización de la estrategia descendente:

[Resumen textual alternativo \(link: DI08_Descripcion_Presentacion_EstrategiaDescendente.html\).](#)

Debes conocer

El orden de integración elegido (ascendente o descendente) afecta a diversos factores, como son:

- ✓ Las herramientas necesarias.
- ✓ El coste de preparación de los casos de prueba.
- ✓ El orden de codificar y probar los módulos.
- ✓ El coste de la depuración.
- ✓ La forma de preparar los casos de prueba.

La selección de una estrategia de integración depende de las características del software y del plan de proyecto. Es bastante habitual combinar las dos técnicas: usar la integración descendente para los módulos superiores de la estructura del programa y la ascendente para los niveles subordinados de bajo nivel.

Autoevaluación

Tras probar varios módulos, de forma independiente, terminamos por librarlos de errores. ¿Es necesario realizar la prueba de integración?

☐ ([link:](#))

No.

☐ (link:)

Sí.

2.2.- Sistema.

Las pruebas de sistema verifican el comportamiento del sistema en su conjunto. En concreto, se comprueban los requisitos no funcionales de la aplicación:

- ✓ Seguridad.
- ✓ Velocidad.
- ✓ Exactitud.
- ✓ Fiabilidad.



También se prueban los interfaces externos con otros sistemas, utilidades, unidades físicas y el entorno operativo.

Entre las pruebas de sistema más relevantes, encontramos las que vemos a continuación.

2.2.1.- Configuración.

El objetivo es verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas y determinar la configuración óptima del sistema.

([link: DI08_CONT_R18_config.jpg.](#))

Estas pruebas verifican la instalación del software en el entorno de destino y comprueban el comportamiento del sistema frente a los requisitos de configuración. Además, analizan el software bajo configuraciones diferentes para diferentes usuarios.



La meta de la prueba es hacer que la aplicación falle en el cumplimiento de los requerimientos de configuración, de manera que los defectos escondidos sean identificados, analizados, arreglados y prevenidos en el futuro.

Durante estas pruebas, el **téster** valida que nuestro proyecto actual es capaz de soportar diferentes tipos de tecnologías de hardware, como diferentes tipos de impresoras, interfaces, topología, etc.

Estas pruebas también son llamadas pruebas de hardware o pruebas de portabilidad.

Actualmente se simulan las pruebas de configuración en equipos con máquinas virtuales, siendo su realización totalmente equivalente a la que se haría en varios equipos físicos.

En general, lo que se suele hacer en esta prueba es configurar una sala o laboratorio con varios equipos físicos que ejecuten diferentes combinaciones de sistemas operativos, exploradores web y otro software , para probar la aplicación con distintas combinaciones de hardware y software . Esta realización (manual) puede resultar bastante costosa, tanto en tiempo como en recursos, por lo que en la actualidad está automatizada con herramientas software que simulan todas estas situaciones sin tener que llevarlas a cabo físicamente.

2.2.2.- Recuperación.

La prueba de recuperación es una prueba de sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo de forma satisfactoria.

Es un parámetro muy importante de la aplicación, ya que si un sistema no es capaz de recuperarse ante una entrada no válida o de un fallo súbito, la calidad de nuestro software quedará en entredicho.

La recuperación de nuestra aplicación puede llevarse a cabo de forma automática o manual. Si la recuperación es automática hay que evaluar la corrección de:

- ✓ La inicialización.
- ✓ Los mecanismos de recuperación del estado del sistema.
- ✓ Los datos.
- ✓ El proceso de arranque.

Si la recuperación requiere la intervención humana, hay que evaluar los tiempos medios de reparación (TMR) para determinar si están dentro de unos límites aceptables.



[.\(link: DI08_CONT_R19_recuperacion.png.\).](#)

Un sistema tolerante a fallos evita que cese el funcionamiento de todo el sistema cuando se produce un fallo del proceso.

2.3.- Regresión.

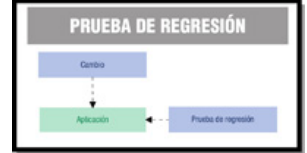
Resulta que cada vez que se añade un nuevo módulo a la aplicación, o cuando se modifica, el software cambia.

Como resultado de esa modificación (o adición) de módulos, podemos introducir errores en el programa que antes no teníamos. Por ello, se hace necesario volver a probar la aplicación. La prueba de regresión es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados.

Así, el objetivo de las pruebas de regresión es comprobar que los cambios sobre un componente de una aplicación no introducen errores adicionales en otros componentes no modificados.

([link: DI08_CONT_R21_regresion.png.](#))

La prueba de regresión se puede hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automáticas (es lo más frecuente).



El conjunto de pruebas de regresión contiene, a su vez, tres clases de prueba diferentes:

- ✓ Planificar una muestra de pruebas sobre todas las funciones del software.
- ✓ Plantear pruebas adicionales que se centren en las funciones del software que se van a ver afectadas por el cambio.
- ✓ Planificar pruebas que se centren únicamente en los componentes que han cambiado.

Es totalmente desaconsejable, por razones obvias de pérdida de tiempo, volver a realizar todas y cada una de las pruebas sobre cada uno de los componentes.

Actualmente, las pruebas de regresión son una parte integral del método de desarrollo de software conocido como Programación Extrema (Extreme Programming). Se utilizan herramientas que permiten detectar este tipo de errores de manera parcial o totalmente automatizada en cada una de las fases del desarrollo del software.

Debes conocer

Entre las herramientas libres para automatizar pruebas de regresión para Java encontramos JUnit , JWebUnit , NUnit , etc. En los siguientes apartados de la presente unidad profundizaremos más sobre la automatización de las pruebas, tanto de regresión como el resto.

2.4.- Funcionales.

Las pruebas funcionales (o de conformidad) validan si el comportamiento observado del software es conforme con sus especificaciones (requisitos funcionales).

También son conocidas como Functional Testing y su objetivo fundamental es probar que las aplicaciones desarrolladas cumplen la función específica para la cual ha sido creada.

Normalmente, los responsables de realizar estas pruebas son los **analistas** de la aplicación (las personas responsables de la especificación de los requerimientos) con apoyo de los usuarios finales. Su aprobación dará paso a la fase de producción propiamente dicha.

Son **pruebas de caja negra** (a los testers de la prueba no les importa cómo se generan las respuestas, sólo analizan las salidas de la aplicación frente a sus entradas).

Las pruebas funcionales intentan responder a las preguntas: "¿funciona esta utilidad de la aplicación?", "¿el usuario podrá hacer esto?".

Los sistemas que ya han sido sometidos a pruebas unitarias requieren menor tiempo para las pruebas funcionales. Esto es así porque se considera que el sistema debe ser ya bastante estable, con pocos errores críticos. Si esto no se cumpliera, deberíamos volver hacia atrás, a la etapa de pruebas unitarias (un sistema inestable sometido a pruebas funcionales requiere altos tiempos asociados con un avance lento, además al ser pruebas de caja negra, al tester no le interesa el funcionamiento intermedio del sistema).



En este tipo de pruebas no interesa el código, ni el algoritmo, ni la eficiencia ni la construcción de elementos innecesarios. Sólo interesa verificar que las salidas que devuelve la aplicación son las esperadas, en función de las entradas que tenga.

Dentro de las pruebas funcionales, encontramos tres tipos:

- ✓ **Análisis de valores límite:** Como entradas, seleccionamos aquellos valores que están en el límite donde la aplicación es válida. (Si estos valores no dan problemas, el resto tampoco los dará).
- ✓ **Particiones equivalentes:** Como entradas, se seleccionará una muestra representativa de todas las posibles. El resultado será extrapolable al resto.
- ✓ **Pruebas aleatorias:** Como entradas, se selecciona una muestra de casos de prueba al azar. Se utiliza sólo en aplicaciones no interactivas.

2.5.- De capacidad y rendimiento.

✓ De capacidad:

La prueba de capacidad (también conocida como prueba de resistencia o stress) ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Es decir, determina hasta dónde el sistema es capaz de soportar determinadas condiciones extremas.

También tiene como finalidad determinar la capacidad del programa para soportar entradas incorrectas.

Para realizar la prueba de capacidad se recurre a los siguientes métodos: Aumentar la frecuencia de entradas para ver cómo responde la aplicación. Se ejecutan casos que requieran el máximo de memoria. Se buscan una cantidad máxima de datos que residan en disco. Se realizan pruebas de sensibilidad, intentando descubrir combinaciones de datos que puedan producir inestabilidad en el sistema.



[\(link: DI08_CONT_R24_capacidad.png.\)](#)

✓ De rendimiento:

Determinan los tiempos de respuesta (lo rápido que realiza una tarea un sistema en condiciones de trabajo), el espacio que ocupan los módulos en disco y en memoria, el flujo de datos que genera a través de un canal de comunicaciones, etc. La prueba de rendimiento está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Se realiza durante todos los pasos del proceso de la prueba. Incluso a nivel de unidad, se debe asegurar el rendimiento de los módulos individuales a medida que se llevan a cabo las **pruebas de caja blanca**. Sin embargo, hasta que no están completamente integrados todos los elementos del sistema no se puede asegurar realmente el rendimiento del sistema.

Prueban el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

- ✓ Requiere de instrumentación tanto software como hardware para los procesos de **monitorización** y medición.
- ✓ Se lleva a cabo durante todos los pasos de prueba.

Cuanto más se tarde en detectar un defecto de rendimiento, mayor será el coste de la solución.

Los sistemas de tiempo real y los sistemas empujados se tienen que ajustar a los requisitos de rendimiento.

2.6.- De uso de recursos.

La evolución de los sistemas informáticos lleva incorporada una mayor complejidad en su diseño y en los recursos hardware y software que requiere. Las aplicaciones requieren cada vez mayor memoria RAM y espacio en disco para funcionar correctamente y nuestros equipos habitualmente tienen limitaciones en este sentido. Es por ello importante el uso eficaz que hace una aplicación de los recursos de que dispone.

La prueba de uso de recursos es también conocida como prueba de eficiencia.

Se desarrolla un conjunto de técnicas que garantice un uso eficiente de los recursos.

Este software estará diseñado, implementado y optimizado sobre sistemas concretos. Igualmente, existe la necesidad de disponer de técnicas que aseguren el uso en sistemas que no son los originales en los que la aplicación fue diseñada. Es decir, asegurar la adaptación a otros sistemas.

La optimización es necesaria para lograr un uso eficiente de recursos. La optimización se debe hacer tanto en el código como en el uso del código. Todo ello para lograr la adaptación del software a la arquitectura de destino y así reducir los tiempos de ejecución de forma automática.



2.7.- De seguridad.

La prueba de seguridad (también conocida como prueba de Integridad) intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos no autorizados. Mide la habilidad de un sistema para soportar ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede ejecutar mediante los programas, los datos o los documentos de la aplicación.

Es decir, determinan los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a los datos.



[.\(link: DI08_CONT_R27_seguridad.png.\).](#)

En concreto, se intenta verificar que el sistema es robusto frente a problemas de seguridad, tales como:

- ✓ Intentar conseguir las claves de acceso de cualquier forma.
- ✓ Atacar con software a medida.
- ✓ Bloquear el sistema.
- ✓ Provocar errores del sistema, entrando durante su recuperación.

Durante la prueba, lo usual es que el encargado desempeña el papel de intruso y trata de violar los mecanismos de seguridad de acceso al sistema. Intenta obtener la clave de acceso de cualquier forma.

Además, también debe intentar bloquear el sistema, curiosear los datos públicos en busca de claves e introducir errores de forma consciente al sistema para ver cómo responde éste.

2.8.- Manuales y automáticas.

Ya sabes que la prueba consiste en la ejecución de un programa con el único objetivo de encontrar defectos. Las condiciones bajo las cuales se realizará la prueba tienen que estar prefijadas para poder evaluar los resultados que obtengamos.

Es decir, las pruebas no se hacen al azar, sino que seguirán una estrategia. En ella, será necesario definir una serie de casos de prueba, que son un conjunto de entradas, de condiciones de ejecución y resultados esperados, desarrollados para un objetivo particular.

Todos los tipos de pruebas que estamos viendo se ejecutan de manera regular con el objetivo de probar que la evolución del software desarrollado funciona perfectamente, tanto para las funcionalidades nuevas como para aquellas que ya existían. Una de las claves es que las pruebas sean automatizadas, porque si se ejecutaran manualmente, el coste de hacerlo "continuamente" sería inasumible.



¿Por qué automatizar?

La razón obvia es ganar tiempo y facilitarnos la vida.

Aparte de esta, que es evidente, los otros motivos son:

- ✓ Mejorar la calidad del producto.
- ✓ Detectar errores en etapas tempranas del desarrollo del proyecto.
- ✓ Disminuir el tiempo de pruebas y, por tanto, el tiempo de salida al mercado.
- ✓ Reducir costes.
- ✓ Ejecutar pruebas de manera continua.

Es importante, en caso de utilizar una herramienta que automatice el proceso de pruebas, realizar una correcta selección de la misma.

En el mercado disponemos de una amplia gama de herramientas en este sentido. Existen infinidad de herramientas gratuitas que, según las características de nuestro proyecto (número de requisitos, grado de estabilidad, organización del equipo, etc.), podemos valorar muy positivamente.

Según Meudec, las herramientas de pruebas de software persiguen automatizar tres tareas fundamentales:

1. Tareas administrativas y generación de documentación.
2. Tareas mecánicas (ejecución).
3. Tareas de generación de casos de prueba.

En los últimos años se han desarrollado una serie de **frameworks** (como el conocido entorno JUnit) para automatizar pruebas unitarias y hay una multitud de entornos de desarrollo (como NetBeans) que incorporan **plugins** y componentes que permiten su integración.

La elección de la herramienta de automatización es muy importante y depende del tipo de pruebas a automatizar, del entorno de desarrollo utilizado y del lenguaje de programación con el que hayamos codificado la aplicación.

2.8.1.- Herramientas software para la realización de pruebas.

A continuación se presenta una tabla con varias herramientas que permiten la automatización de ciertos tipos de pruebas de software.
Herramientas de automatización de pruebas software

Tipo de prueba	Herramienta	Licencia	Lenguaje
Unitari	JUnit	CPL	Java
	Simple Test	LGPL	PHP
	PHP Unit	PHP	PHP
	Jtiger	GPL	Java
Funcional y de Regresión	Selenium	Apache	Java, PHP, Python, Ruby, entre otros.
	Watir	BSD	Ruby
	Watij	GPL v2	Java
	JWebUnit	GPL	Java
	NUnit	GPL	C#, J#, VB y C++
	Http Unit	GPL	Java
Integración	TestNG	GPL	Java
	Hudson	Subversion	Java
	Continuum	Subversion	Java
Carga y Rendimiento	JMeter	Apache	Java
	Jcrawler	CPL	Java
Aceptación	FitNesse	GPL	Java, PHP, Ruby, .NET
	Concordion	GPL	Java, Python, Ruby, .NET
	Nessus	GPL	NASL,XML,HTML

En la automatización de las pruebas al software, debes tener en cuenta:

- ✓ Definir los objetivos de la automatización en base a los objetivos de calidad.
- ✓ Elegir las pruebas que se van a automatizar.
- ✓ Seleccionar correctamente las herramientas para la automatización.
- ✓ Se trata de una inversión cuyos beneficios se observan a medio plazo.
- ✓ Debe contar con personal especializado.

2.9.- De usuario.

Los usuarios finales van a ser los que utilicen nuestra aplicación y, por tanto, ésta debe satisfacer sus necesidades, objetivos y expectativas.

La prueba de usuario (también conocida como de usabilidad) consiste en asegurar que la interfaz de usuario (GUI) de la aplicación sea amigable, intuitiva y que funcione correctamente. Se determina la calidad del software en la forma en la que el usuario interactúa con el sistema, considerando la facilidad de uso y satisfacción del cliente.

Se define la usabilidad de una aplicación como un parámetro que mide de qué forma los usuarios la utilizan: cómo navegan a través de sus pestañas, ventanas, buscadores, etc. Los análisis de usabilidad mostrarán lo que los usuarios realmente van a ver.

Podemos evaluar la usabilidad de una aplicación midiendo la manera en que el usuario realiza una tarea concreta, el tiempo y número de clics que supone acabarla y los errores que comete durante el proceso.

Es obvio que el estudio de la reacción de los individuos ante nuestra aplicación es un proceso complicado, pero existen varias **métricas** que implican un grado de fiabilidad que queda cubierto en un entorno de pruebas automatizadas.

Para verificar la facilidad de uso de una aplicación se pueden incluir en los ensayos de pruebas:

- ✓ Encuestas y entrevistas.
- ✓ Grupos de enfoque.
- ✓ Pruebas con herramientas de software avanzado.
- ✓ Evaluaciones realizadas por expertos.



Como en otras muchas ocasiones, cuanto antes se realicen las pruebas de usabilidad, mucho mejor: no sólo resultará más barato a largo plazo, sino que se adaptará también mejor al programa (si se corrigen los problemas a tiempo no habrá que construir sobre errores).

2.10.- De aceptación.

La prueba de aceptación (o de validación) sirve para comprobar que el software cumple con los requerimientos iniciales y que funciona de acuerdo con lo que el cliente esperaba de él, es decir, satisfaciendo sus expectativas. Por eso, y para que la prueba sea objetiva, es el mismo cliente quien la realiza. Podemos escuchar comentarios como: "esto no era lo que yo quería" o "esto no es lo que yo necesito", e incluso "esto no es lo que yo pedí".

Para su ejecución se suelen emplear dos técnicas de aceptación: alfa y beta.

✓ Pruebas alfa

Las lleva a cabo el cliente en el lugar donde se ha desarrollado la aplicación y en presencia del desarrollador. El cliente utilizará la aplicación como lo haría en su lugar de trabajo. Se utiliza un entorno controlado con las mismas condiciones que hay en el lugar de trabajo habitual. Una vez realizadas, se documentan los resultados.

✓ Pruebas beta

Las lleva a cabo el cliente pero ya en su lugar de trabajo. A diferencia de las pruebas alfa, el desarrollador ya no está presente durante su realización, por lo que estas pruebas se denominan "en vivo", puesto que no se ejecutan en un entorno controlado. El cliente registrará todos los problemas que encuentre durante la prueba beta e irá informando al desarrollador a intervalos regulares de tiempo. Éste realizará las correcciones y modificaciones pertinentes y preparará una nueva versión del producto.

En esta prueba se evalúa el grado de calidad del software en relación a su uso. Es definida por el usuario cliente y preparada por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

La validación del sistema se consigue mediante la realización de **pruebas de caja negra** que demuestran la conformidad con los requisitos.





En la prueba de aceptación se evalúa:

- ✓ Que se satisfacen los requisitos funcionales.
- ✓ Que se alcanzan los requisitos de rendimiento.
- ✓ Que la documentación es correcta e inteligible.

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	<p>Autoría: Silveira Neto.</p> <p>Licencia: CC by-sa.</p> <p>Procedencia: http://www.flickr.com/photos/silveiraneto/2579658422/</p>		<p>Autoría: Cirofono.</p> <p>Licencia: CC-by.</p> <p>Procedencia: http://www.flickr.com/photos/ciroduran/225283/</p>

