# INTRODUCCIÓN

La preocupación por la calidad de software creció a medida que la imagen de las empresas pasó a estar cada vez más expuesta al público mediante el surgimiento de los sistemas web.

En torno a 1990, grandes empresas de ese ramo reconocían que miles de millones de dólares estaban siendo desperdiciados en softwares que no presentaban características y funcionalidades prometidas. Vivían aquel dilema de querer producir el software "perfecto", pero sin tener tiempo y esfuerzo necesario para tal hazaña. Esto las llevó a buscar nuevos medios para perfeccionar la calidad. Y uno de esos caminos fue el perfeccionamiento de las actividades relacionadas con lo test de software a través de la automatización.

A diferencia de la filosofía que muchas organizaciones siguen, tener un entorno de tests automatizadas no es algo tan costoso y complejo como parece ser. Con el conocimiento bien difundido, hoy tenemos a disposición diversas herramientas de automatización *open sources*, que con sólo unos clics, ya es posible crear *scripts* de tests eficientes que validan las funcionalidades del sistema tantas veces como sea necesario de forma automática. Un buen ejemplo de ello es el uso de la herramienta Selenium, la cual será la propuesta de ese libro.

# 1. CONCEPTOS Y FUNDAMENTOS DE TEST DE SOFTWARE

*"Nunca hay tiempo para hacer lo correcto, pero siempre hay tiempo para hacerla de nuevo."*

## 1.1. CALIDAD DE SOFTWARE

Definir calidad no es algo trivial como parece. Para Pressman (2011), calidad en software y algo que necesita ser definido en un nivel más pragmático. Para ello, se basa en el concepto de David Gravai (Gar84), que definió el tema en cinco puntos de vista diferentes:

**Visión trascendental:** Sostiene que la calidad es algo que se reconoce inmediatamente, pero no se puede definir explícitamente.

**Visión del usuario:** Viene la calidad en términos de los objetivos específicos de un usuario final. Si un producto cumple estos objetivos, presenta calidad.

**Visión del Fabricante:** Define calidad en términos de especificación original del producto. Si el producto cumple con las especificaciones, presenta calidad.

**Visión del producto:** Sugiere que la calidad pueda estar ligada a las características inherentes de un producto, como por ejemplo, funciones y recursos.

**Visión del valor:** Mide la calidad tomando como base cuánto un cliente estaría dispuesto a pagar por el producto.

### 1.1.1. FACTORES DE CALIDAD

La **ISO/IEC 25000**[1], conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), unió las normas **ISO/IEC 9126** (modelo) y **ISO/IEC 14598** (proceso) con el objetivo de perfeccionar la evaluación de calidad del software.

En su división **ISO / IEC 2501n**, encontramos el modelo **ISO / IEC 25010**, que define las características y subcaracterísticas que un software necesita tener para ser considerado un "software de calidad".

- **Adecuación Funcional**
    - Completde: Capacidad en la que las características cubren todos los requisitos del usuario.
    - Exactitud: Capacidad en que el sistema presenta resultados correctos.
    - Adecuación: Capacidad en que las funcionalidades poseen en adecuarse a los requisitos del usuario.
- **Eficiencia del Rendimiento**
    - Tiempo de Comportamiento: Capacidad de tiempo de respuesta, procesamiento o transferencia de las requisiciones efectuadas por el sistema.
    - Uso de recursos: Capacidad en la que los recursos (procesador, memoria, etc) son utilizados por el sistema.
    - Capacidad: Capacidad en la que el sistema admite transacciones.
- **Compatibilidad**
    - Coexistencia: Capacidad en que el software se desempeña al compartir el mismo ambiente con otros softwares.
    - Interoperabilidad: Capacidad en la que el software se comunica (intercambio de información) entre sí.
- **Usabilidad**
    - Reconocimiento: Capacidad en la que el usuario reconoce si el software atiende bien sus necesidades.

[1] ISO/IEC 25000:2014. System and software engineering. Disponible: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764>. Acceso en: 07/12/2015

- precibilidad: Capacidad en la que el software puede ser utilizado por usuarios específicos para atender objetivos específicos a nivel de aprendizaje.
- Operacionalidad: Capacidad que el usuario tiene en operar el software.
- Protección contra errores: Capacidad en la que el sistema tiene y evitar errores de manejo por parte del usuario.
- Estética de la interfaz: Capacidad en que la interfaz del sistema tiene en interactuar de forma satisfactoria y agradable con el usuario.
- Accesibilidad: Capacidad en que el software tiene que ser operado por diversos tipos de usuarios, de diferentes necesidades.

- **Fiabilidad**

  - La madurez: Capacidad que el software posee en atender necesidades de Confiabilidad en operaciones normales.
  - Tolerancia a fallas: Capacidad que el software tiene que funcionar bien, incluso con fallas de hardware o demás softwares.
  - Recuperación: Capacidad que el software debe volver a su estado normal después de las interrupciones.

- **Seguridad**

  - Confidencialidad: Capacidad que el software tiene para proporcionar información sólo para usuarios autorizados.
  - Integridad: Capacidad en la que el software tiene que impedir accesos o cambios indebidos.
  - No repudio: Capacidad del software en reconocer la veracidad de ciertas acciones y eventos para no repudiar en otro momento.
  - Prestación de cuentas: Capacidad que el software tiene que asignar una acción a una entidad.
  - Autenticidad: Capacidad del software para reconocer la verdadera identidad de un recurso.

- ***Mantenibilidad***

  - *Modularidad: Capacidad que el software posee de tener sus componentes intercambiados sin tener impacto entre ellos.*
  - *Reutilización: Capacidad que el activo tiene que ser reutilizado en más de un sistema.*
  - Analisabilidad: Capacidad del software medir los impactos de los intercambios de componentes así como de identificar las debidas correcciones.

- Modificación: La capacidad del software tiene que no presentar fallas después de las modificaciones, ya sea por mejora o por corrección de errores.
- Comprobabilidad: Capacidad del software para ajustarse a los criterios de test.

- **Portabilidad**

  - Adaptabilidad: Capacidad del software para adaptarse a los más variados tipos de ambientes.
  - Inestabilidad: Capacidad de instalar y desinstalar software de un entorno.
  - Sustitución: La capacidad del software tiene que ser reemplazado por otro en menor impacto posible.

### 1.1.2. EL DILEMA DE LA CALIDAD DE SOFTWARE

En cuanto a calidad en las organizaciones, Pressman (2011) cita una entrevista (Ven03) publicada en internet, donde Bertrand Meyer discute lo que llamamos **dilema de la calidad:**

> Si producimos un sistema de software de pésima calidad, perdimos porque nadie querrá comprarlos. Si, por otro lado, pasamos un tiempo infinito, un esfuerzo extremadamente grande y grandes sumas de dinero para construir un software absolutamente perfecto, entonces eso llevará mucho tiempo para ser completado, y el costo de producción será tan alto que vamos a la quiebra. O hemos perdido la oportunidad de mercado o simplemente simplemente agotamos todos nuestros recursos. De esta manera, los profesionales de esta área tratan de encontrar ese término mágico donde el producto y lo suficientemente bueno para no ser rechazado de cara, como por ejemplo, durante una evaluación, pero tampoco es el objeto de tamaño proteccionismo y trabajo que llevaría mucho tiempo o que costaría demasiado para ser finalizado.

En otras palabras, ese dilema se refiere a la Ley de Meskimen, que dice:

*"Nunca hay tiempo para hacer lo correcto, pero siempre hay tiempo para hacerla de nuevo".*

Aunque hay puntos negativos para ambas decisiones, Pressman (2011), aconseja que tomar el tiempo necesario para hacer lo correcto la primera vez nunca es una decisión equivocada.

### 1.1.3. COSTO DE LA CALIDAD

En cuanto al dilema, es común encontrarnos con la siguiente objeción presentada por muchas empresas: "Sabemos que la calidad es importante, pero nos cuesta tiempo y dinero en exceso para obtener el nivel de calidad de software que realmente deseamos.".

Para Pressman (2011), no hay duda de que la calidad tiene un precio, pero la falta de ella también tiene un precio - no sólo para los usuarios finales, que tendrán que convivir con un software con errores, pero también para la organización de software que lo creó y, además de todo, tendrá que hacer mantenimiento. Y eso nos lleva a la siguiente pregunta: ¿Qué costo deberíamos preocuparnos? ¿El costo de la alta calidad o el costo generado por una baja calidad?

Los costos de calidad engloban todos los costos involucrados en la búsqueda, ejecución o falta de la misma. Se clasifican en:

**Costo de prevención**: Incluyen el costo de las actividades de gestión necesarias para planificar y coordinar todas las actividades de control y garantía de calidad. El costo de actividades técnicas adicionales para desarrollar modelos completos de requisitos y de diseño. Costos de planificación de tests y el costo de todo el entrenamiento asociado a esas actividades.

**Costo de evaluación**: Incluyen actividades para la comprensión en profundidad de la condición del producto "la primera vez a través de cada proceso. Entre los ejemplos de costes de evaluación tenemos:

- Costo para la realización de revisiones técnicas para productos resultantes de ingeniería de software.
- Costo para la recolección de datos y la evaluación de métricas.
- Coste de tests y depuración.

**Costo de fallas**: Son aquellos que desaparecerían si ningún error hubiera surgido antes o después de la entrega de un producto a clientes. Estos costos se pueden subdividir en costos de fallas internas y costos de fallas externas. Los costos de fallas internas ocurren cuando se detecta un error en un producto antes de que se entregue y cubra:

- Costo necesario para realizar trabajos (reparaciones) para corregir un error.
- Costo que ocurre cuando los trabajos generan, inadvertidamente, efectos colaterales que deben ser reducidos.
- Costes asociados a la reunión de métricas de calidad que permiten a una organización evaluar los modos de fallo.

Por otro lado, los costos de fallas externas están asociados a defectos encontrados después de que el producto haya sido entregado al cliente. Costos que provienen de reclamaciones, devolución / sustitución de productos, soporte por teléfono / mal y hasta costos por mano de obra asociada a la garantía del producto.

En cuanto al valor aproximado de los costos, esto queda relativo a la medida un error / defecto va progresando de la etapa de prevención a la detección de fallas internas y externas. A través de los datos recogidos por Boehm y Basili (Boe01b), Cigital Inc (Cig07) ejemplifica este escenario a través de la siguiente ilustración:

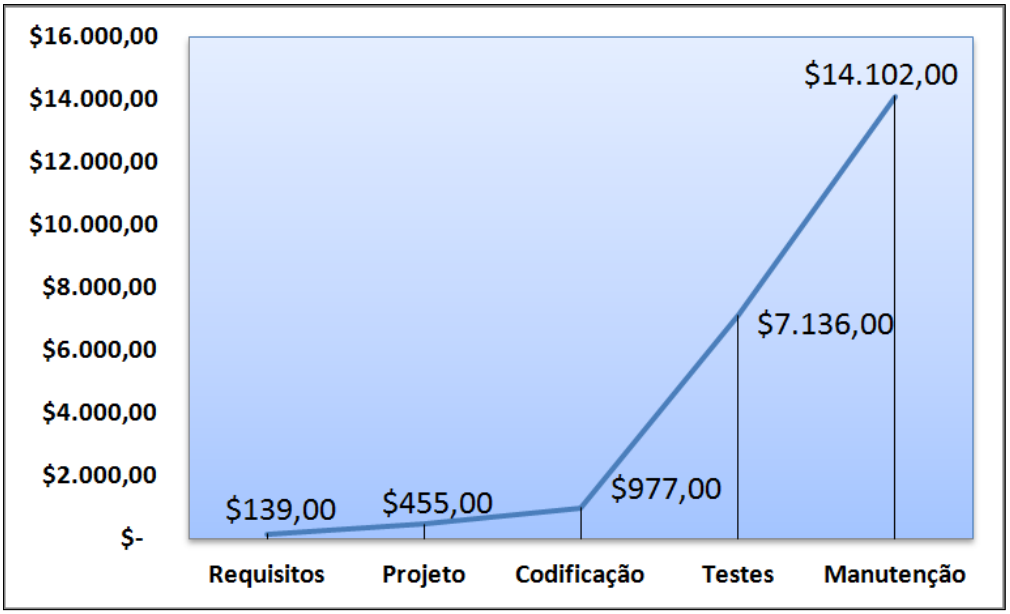


**Figura 1 -** Costo relativo a la corrección de errores y defectos

Fuente: Pressman (2011, p.367)

Como vemos en la Figura 2.1, el costo promedio de la industria de software para corregir un defecto durante la generación de código es aproximadamente US $ 977 por error. El costo promedio para corregir el mismo error si se descubrió durante los tests (fallo interno) del sistema, pasa a ser de US $ 7.136 por error. Y ese costo puede doblarse, si el error es detectado por el usuario final (fallo externo), llegando a un valor de aproximadamente de US $ 14.102 por error.

Este análisis llevó a Pressman (2011) a la conclusión de que incluso la organización de software con costos equivalentes a la mitad del promedio del sector, el ahorro de costos asociados a actividades iniciales de control con garantía de calidad es considerable.

### 1.1.4. CÓMO ALCANZAR LA CALIDAD?

La calidad de software no aparece simplemente de la nada. Es el resultado de una buena gestión de proyectos y una práctica consistente de ingeniería de software, afirma Pressman (2011). Esta administración y práctica se aplican en el contexto de cuatro grandes actividades que ayudan a un equipo a alcanzar el alto nivel de calidad en el software. Son ellas:

**Métodos de Ingeniería de Software:** Si la expectativa es construir software de calidad, es importante entender el problema a ser resuelto. Tener la capacidad de crear un proyecto que sea adecuado al problema y, al mismo tiempo, presentar características que lleven a un software con las dimensiones y factores de calidad.

**Técnicas de Gestión de Software:** La calidad de software se ve afectada positivamente cuando un gerente de proyecto utiliza estimaciones para comprobar que las fechas de entrega son plausibles. Cuando se entienden las dependencias de cronograma y el equipo resiste la tentación de usar atajos. Cuando la planificación de riesgos se conduce de moda que los problemas no generan caos.

**Control de calidad:** Se aplica una serie de pasos de test para descubrir errores en la lógica de procesamiento, manipulación de datos y comunicación de interfaz. Uma combinação de medições e realimentação (*feedback*) permite a un equipo de software ajustar el proceso cuando cualquiera de estos productos resultantes deje de cumplir las metas establecidas para la calidad.

**Garantía de Calidad:** Consiste en un conjunto de funciones de auditoría e informes que posibilita una evaluación de efectividad y de la completitud de las acciones de control de calidad. Se pretende proporcionar al personal técnico administrativo los datos necesarios para ser informados sobre la calidad del producto, ganando, por lo tanto, entendimiento y confianza de que las acciones para alcanzar la calidad deseada del producto están funcionando.

## 1.2. TEST DE SOFTWARE

Test de Software es una buena práctica realizada en un desarrollo de software, donde su principal objetivo, según Bastos (2012), es reducir los riesgos que las aplicaciones pueden traer para los negocios. Y para Pressman (2011), un bueno test es aquelle que posee una serie de características que le permiten encontrar el mayor número de errores posibles con el mínimo esfuerzo.

En el concepto de test, Bastos (2012), relata que grandes modelos de mejora de procesos de desarrollo de software tales como CMMI y MPS. BR contemplaron las actividades de tests, identificándolas en las áreas de verificación y validación que, por cierto, son términos con sentidos completamente diferentes.

**Verificación**: tests que permiten verificar si el software está siendo construido correctamente.
**Validación**: tests que permiten validar si el software está haciendo lo que se ha definido en los requisitos.

En otras palabras:

**Verificación**: ¿La forma en que construimos el software es correcta?
**Validación**: ¿El software que construimos, es correcto?

En aplicaciones convencionales, Pressman (2011) dice que el software se prueba en dos perspectivas diferentes: caja blanca y caja negra.
**Test de Caja Blanca**: Se fundamenta en un examen riguroso del detalle procedimental. Los caminos lógicos del software y las colaboraciones entre componentes se prueban, ejercitando conjuntos específicos de condiciones y / o ciclos.

**Test de Caja Negra**: Se hace referencia a las pruebas realizadas en la interfaz del software. Examina algunos aspectos fundamentales de un sistema, con poca preocupación en relación con la estructura lógica interna del software.

### 1.2.1. PROCEDIMIENTO DE TEST

De acuerdo con Bastos (2012), presupone que el proceso de prueba esté en paralelo al proceso de desarrollo. Para ello, menciona la utilización del concepto "V".

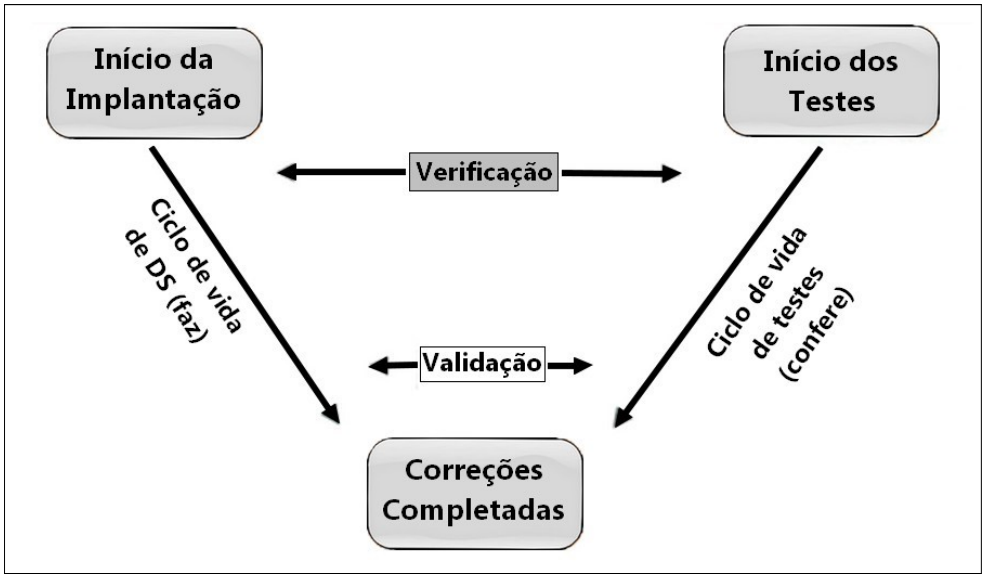


**Figura 2 -** Concepto "V" de test de software
Fuente: Bastos (2012, p.41)

Como podemos ver en la figura 2.2, ambos equipos empiezan desde el mismo punto. En la etapa de especificación, mientras el equipo de desarrollo se encarga de capturar y documentar los requisitos con el propósito de desarrollar el sistema, el equipo de prueba realiza el mismo procedimiento, pero con el fin de probar el sistema. Al entrar en la etapa de ejecución, las pruebas de verificación y validación se aplican respectivamente a la construcción e instalación / mantenimiento del sistema.

Es importante resaltar que el concepto "V" se centra en las etapas de especificación y ejecución del proceso de prueba. La forma más completa del proceso se describe en lo que llamamos **Ciclo de vida de Tests.**

Al referirse al libro Test de Software, Bastos (2012) dice que los autores Rios (2003) y Moreira (2003), utilizan la metodología Test Management (TMAP) para ilustrar el ciclo de vida de las pruebas de forma didáctica y de fácil visualización.

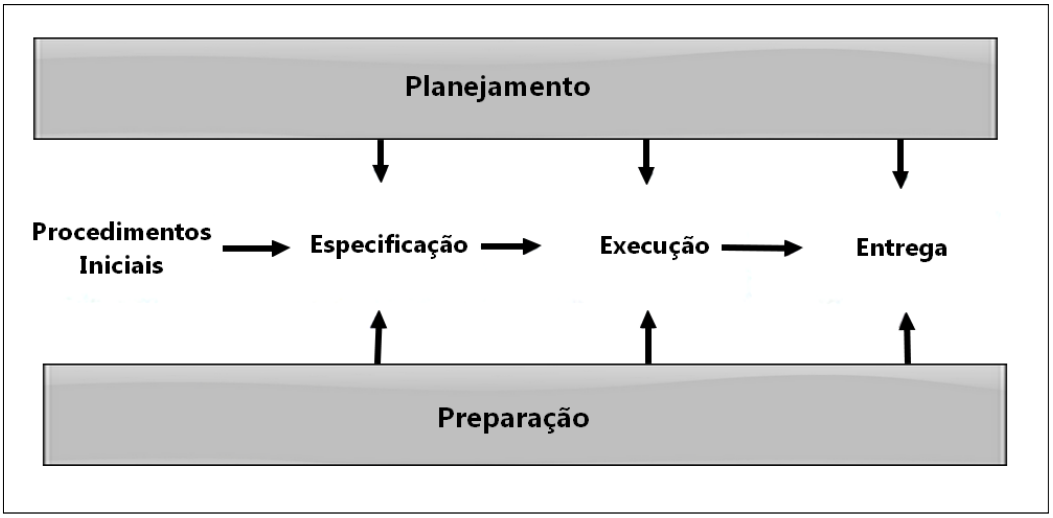


**Figura 3 -** Modelo 3P x 3E del ciclo de vida de los tests.
Fuente: Bastos (2012, p.45)

El ciclo se compone de seis etapas, cuatro de ellas secuenciales y dos paralelas. A continuación, sigue la descripción y función de cada paso:

**Procedimientos iniciales:** Etapa donde se realiza un estudio en profundidad de los requisitos de negocio del sistema a ser desarrollado. Garantizando que el mismo esté completo de sin redundancia. Después de eso, se deben definir las principales actividades de pruebas que se ejecutarán.

**Planificación:** Etapa que consiste en la elaboración de una estrategia de prueba y un plan de prueba, con el objetivo de minimizar los principales riesgos al negocio y proporcionar caminos para las próximas etapas.

**Preparación:** En esta etapa, el objetivo es preparar el ambiente para que las pruebas se ejecuten correctamente. Esta preparación implica la definición de configuración de equipos, entrenamientos de equipo, instalación de herramientas de tests, etc.

**Especificación:** Etapa que consiste en elaborar y revisar casos de pruebas de acuerdo con los requisitos definidos en los procedimientos iniciales.

**Ejecución:** Paso donde se ejecutan los casos de prueba es de forma manual o automática (usando *scripts*). En casos de corrección de errores los ajustes de mejoras, una nueva ejecución es efectuada, garantizando la integridad del sistema.

**Entrega:** Etapa de cierre de las pruebas, donde se elaboró una documentación con todas las ocurrencias de tests, buscando la mejora del proceso.

### 1.2.2. AMBIENTE DE TEST

El entorno de prueba es la estructura donde se ejecutan las pruebas, que va mucho más allá de una simple configuración de software y hardware.

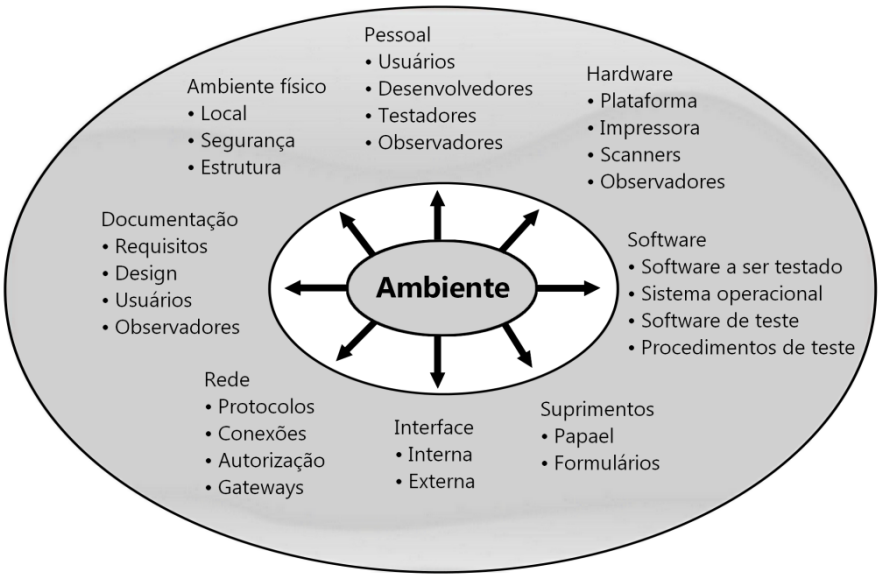


**Figura 4 -** Estructura del Ambiente de Test

Fuente: Bastos (2012, p.77)

Es importante resaltar que el equipo de test no sólo está compuesto por probadores como vemos por ahí. En grandes instituciones (principalmente cuando se tiene un proyecto de prueba aparte), podemos encontrarnos como otros profesionales, cuya responsabilidad va mucho más allá de ejecutar tests:

| | |
|---|---|
| **Líder de Proyecto de Test (LT)** | Técnico responsable del liderazgo de un proyecto de test específico, generalmente relacionado con un sistema de desarrollo, sea un proyecto nuevo o un mantenimiento. |

| | |
|---|---|
| **Arquitecto de Test (AT)** | Es el técnico responsable del montaje de la infraestructura de test; monta el ambiente de test, escoge las herramientas de test y capacita al equipo para ejecutar su trabajo en ese ambiente de trabajo test. |
| **Analista de Test (AN)** | Es el técnico responsable por el modelado y elaboración de los casos de test y por los *scripts* de test |
| **Testador (TE)** | Técnico responsable de la ejecución de los casos de test y *scripts* de test |

**Cuadro 1 -** Descripción de los profesionales que componen el equipo de test.

Fuente: Bastos (2012, p.80)

Cada profesional de éste puede actuar en una o más actividades que componen el ciclo de test. Sin embargo, cada uno desempeña su debida responsabilidad:

| **Pasos** | **Actividades** | **LT** | **AT** | **AN** | **TE** |
|---|---|---|---|---|---|
| **Planificación** | Elaborar estrategias de tests | E | P | | |
| | Elaborar planes de test | E | P | | |
| **Preparación** | Proporcionar infraestructura y herramientas | A | E | P | |
| | Ofrecer personal | E | P | | |
| **Especificación/ Ejecución** | Elaborar casos de test | A | P | E | |
| | Ejecutar los tests | | | | E |
| | Seguimiento del progreso de los tests | A | | E | |
| **Entrega** | Evaluar y archivar la documentación de test | A | E | P | |
| **A – Acompaña; P - Participa; E - Ejecuta.** | | | | | |

**Cuadro 2** - Matriz de responsabilidades.

Fuente: Bastos (2012, p.81)

Otro elemento importante del entorno de prueba son las herramientas de prueba. Ellas son el apoyo de los profesionales de esa área, pues cubren gran parte de las actividades de prueba y son aplicables en todas las fases del ciclo de vida del software.

La elección adecuada de la herramienta influye positivamente en la eficiencia y eficacia de las pruebas. Para ello, es importante analizar factores como objetivos de la organización, costos de la herramienta y el nivel de conocimiento para manejarla.

Actualmente existen varias herramientas de tests *open source* disponibles para *downloads*. Em el sitio[2] *Open Source Testing* encontramos la relación de herramientas más descargadas por categoría:

- **Herramientas de Test Unitaria**

***JUnit***

*Framework* de teste de regresión escrita por Erich Gamma y Kent Beck. Usado por el desarrollador que programa pruebas unitarias en Java.

***NUnit***

*Framework* de test de unidad para todos los linguajes .NET. En primer lugar, portado de JUnit, la versión actual, 2.0 es el segundo gran lanzamiento de esta herramienta de prueba de unidad con base xUnit para Microsoft .NET. Está totalmente escrito en C# y fue completamente rediseñado para aprovechar los muchos recursos de lenguaje .NET, por ejemplo, atributos personalizados y otros recursos relacionados con la reflexión.

- **Herramientas de prueba funcional**

***Canoo WebTest***

Usado para tests funcionales de páginas web, WebTest es um *framework* de tests de código abierto construido en la parte superior del HttpUnit. Permite que los tests que se definen en XML como objetivos Ant.

***Selenium***

Herramienta de prueba para navegadores web. Se puede utilizar tanto para funcionalidad, compatibilidad (que tiene amplio soporte cross-browser) y pruebas de regresión.

- **Herramientas de test de rendimiento**

***JMeter***

[2] Dirección del sitio: http://www.opensourcetesting.org/

JMeter es una aplicación de escritorio Java 100% puro diseñado para cargar el comportamiento y la prueba de rendimiento medida. Fue originalmente diseñado para aplicaciones web de prueba, pero desde entonces se ha ampliado a otras funciones de test. JMeter se puede utilizar para pruebas de rendimiento tanto en recursos estáticos y dinámicos (archivos, Servlets, scripts Perl, objetos Java, bases de datos y consultas, servidores FTP y mucho más). Se puede utilizar para simular una carga pesada en un servidor, una red o un objeto para probar su resistencia o para analizar el rendimiento general bajo diferentes tipos de carga. Puede utilizarlo para realizar un análisis de rendimiento gráfico o para probar su comportamiento / objeto de servidor / secuencia de comandos bajo carga pesada concurrente.

***WebLoad***

WebLoad Open Source es una herramienta (de nivel comercial) para prueba de desempeño totalmente funcional desarrollada por la empresa RadView y que ya está implantado en 1.600 sitios. Disponible gratuitamente para download, WebLoad es un proyecto de código abierto de nivel comercial con más de 250 años de ingeniería de desarrollo de productos. Las empresas que necesitan un soporte comercial, recursos adicionales de productividad y compatibilidad con los protocolos de terceros tienen la opción de comprar WebLoad Professional directamente de RadView.

- **Herramientas de gestión de tests**

***Fitnesse***

Fitnesse es una herramienta de prueba y documentación colaborativa. Proporciona una manera muy sencilla para los equipos para crear documentos de colaboración, especificar los tests, y realizar estes tests.

***TestLink***

TestLink es un basado en la web y sistema de gestión de pruebas de ejecución. La herramienta incluye la especificación de prueba, planificación, elaboración de informes, requisitos de seguimiento y colaborar con los seguidores de *bugs* conocidos.

- **Herramienta de seguimiento de bugs**

***Bugzilla***

Bugzilla ha madurado inmenso, y ahora tiene muchas características avanzadas. Estos incluyen: integrados, granular esquema de seguridad basado en el producto, dependencias inter-bugs y dependencia de gráficos, características avanzadas de informe, un robusto, RDBMS estable *back-end*, gran capacidad de configuración, una resolución de error natural muy bien comprendida y bien pensada en el protocolo, e-mail, XML, APIs console, e HTTP,

integración disponible con sistemas de gestión de configuración de software automatizados, incluidos Perforce y CVS a través de la interfaz del usuario e-mail Bugzilla y *scripts* de check-in / check-out), muchas más características para la lista.

***Eventum***

Eventum es un sistema de rastreo cuestión flexible y de fácil manejo que puede ser utilizado por un departamento de soporte para controlar las solicitudes de soporte técnico de entrada o por un equipo de desarrollo de software para organizar rápidamente tareas y *bugs*.

### 1.2.3. TÉCNICAS DE TEST

La técnica de prueba es un proceso que asegura el funcionamiento adecuado de algunos aspectos del sistema o de la unidad. El probador necesita conocer primero las

técnicas de prueba para luego entender qué herramientas deben ser usadas para aplicar cada técnica, afirma Bastos (2012).

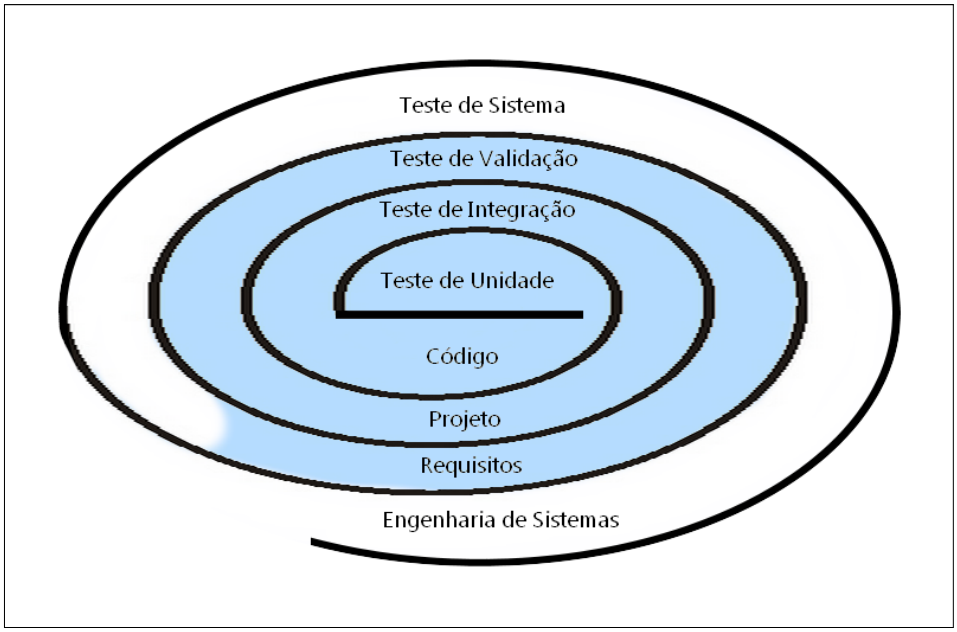


**Figura 5** - Estrategias de Test
Fuente: Pressman (2011, p. 404)

- ***Test de Unidad***

Test que enfoca el esfuerzo de verificacin en la menor unidad del proyecto de software, pudiendo ser éste un componente o módulo.

- ***Test de Integración***

Test que verifica la integridad del sistema después de la unión de los componentes probados individualmente. Como ejemplos de tests de integración:

  - **Test Top-Down:** Comprueba la integración de cada módulo, empezando por el módulo de mayor nivel (módulo de control principal).

  - **Test Bottom-up:** Comprueba la integración de cada módulo, empezando por los módulos de bajo nivel (módulos subordinados).

- **Test Regrésion:** Re-ejecución de los mismos subconjuntos de pruebas para asegurar que la inclusión de nuevos módulos no haya propagado efectos colaterales no deseados en el sistema.

- **Test Humo:**
- Rutina de tests que ocurre en el sistema entero (punta a punta). El propósito es detectar errores a cada uno *build* de los módulos que se han integrado.

- ***Test de Validación***

Test que valida las acciones visibles al usuario, de modo que el software funcione de una manera razonablemente esperada por él. Como ejemplo tenemos los tests:

- **Test Alpha:** Se produce en un ambiente controlado, donde el desarrollador monitorea el uso del sistema por los usuarios finales, registrando posibles errores funcionales.

- **Test Beta:** Ocurre en el propio ambiente de producción, sin la presencia del desarrollador. Los errores encontrados se registran a intervalos regulares por los propios usuarios finales.

- ***Test del Sistema***

Test que ejercita totalmente el sistema, verificando si los elementos del sistema se integraron adecuadamente y se ejecutan las funciones asignadas a ellos. Los ejemplos son:

- **Test de Recuperación:** Evalúa la capacidad del sistema de recuperarse mediante las fallas, retornando el procesamiento en poco o ningún tiempo de parada.

- **Test de Seguridad:** Comprueba si los mecanismos de protección incorporados al sistema están aptos para protegerlo de accesos indebidos.

- **Test por Esfuerzo:** Conocido como prueba de estrés, pretende colocar el sistema en condiciones anormales para saber hasta dónde va su tolerancia a fallas. Para que este límite sea superado, se utilizan los recursos al máximo, sea en cuanto a cantidad, frecuencia o volumen.

- **Test de Rendimiento:** Evalúa el rendimiento del sistema como: tiempo de ejecución, uso de recursos (procesador, memoria, red).

**Test de Disponibilidad:** También conocido como prueba de configuración, valida la flexibilidad del sistema en ser operado en los más diversos tipos de ambientes (sistemas operativos y navegadores).