Términos Relacionados y Definiciones

Pruebas de Sistemas

UNIVERSIDAD SIGLO 21

Términos relacionados y definiciones

Dentro de esta disciplina, comienza a manejarse una serie de términos de los cuales debemos conocer su significado. El International Software Testing Qualifications Board (ISTQB, 2011) ha definido un glosario de términos para establecer un vocabulario común entre los profesionales del *testing*, además de homologar las traducciones con sus regionalizaciones.

Cualquier profesional relacionado con las pruebas de sistemas debe conocer lo que se define como *caso de prueba*, tener claro la diferencia entre *validar* y *verificar*, y también, cuando hablamos de un *error*, una *falla* o un defecto, entender su acepción.

Error, falla, defecto

El ISTQB (2011) definió los conceptos de *error*, *falla* y *defecto* y realizó una distinción entre cada uno de ellos. Los profesionales del medio han adoptado internacionalmente estas definiciones.

***Error**: es la acción humana de equivocarse.*

***Falla**: es la diferencia entre el resultado esperado y el realmente obtenido, es decir, que es una desviación del componente o del sistema en cuanto al resultado esperado.*

***Defecto:** se llama de este modo al desperfecto, en un componente o sistema, que puede causar que el componente o sistema falle en desempeñar las funciones requeridas; por ejemplo, una sentencia o una definición de datos incorrectos.*

¿Es correcto decir que un error lleva a uno o más defectos que están presentes en un componente de software o que un defecto ocasiona cero, una o más fallas?

Claro que sí. El acto de equivocarse (error) lleva a un desperfecto, que podría ser un error en el código (defecto). Así, si durante la ejecución se localiza ese defecto, puede causar un fallo en el componente o sistema (falla). Entonces, un defecto no siempre causa una falla, sino que el software falla si se ejecuta esa parte del código donde se encuentra el defecto.

Si ya realizamos las pruebas, si un defecto no se detectó, ¿qué puede suceder?

Puede suceder que, una vez que el sistema se encuentre en funcionamiento en el entorno real, se produzca la falla. De esto se deduce que las pruebas tienen que estar enfocadas en provocar que el software falle para detectar el defecto vigente antes de que lo detecte el usuario o cliente.

Entonces, ¿qué detectamos durante las pruebas?, ¿fallas o defectos?

Con las pruebas propiamente dichas, estamos buscando provocar fallos, y todos esos fallos pueden deberse al mismo problema o al mismo error de código (defecto).

A partir de aquí, una vez comprendidos los conceptos, no los emplearemos indistintamente, sino que, cuando hablemos de fallas, sabremos que es cuando el sistema tiene un comportamiento no esperado o distinto al esperado y, cuando hablemos de defecto, estaremos indicando un desperfecto.

Al realizar pruebas, nos estamos refiriendo a ejecutar casos de pruebas que se construyen con el único fin de provocar fallos. Sin embargo, ¿qué es un caso de prueba o cómo se define?

Caso de prueba

La Institute of Electrical and Electronics (IEEE, 2008), define:

Caso de prueba *como la documentación que especifica las entradas, los resultados previstos o esperados, y un conjunto de condiciones de ejecución de un elemento de prueba.*

Su traducción en inglés es *test case* y, en la actividad diaria profesional, se adopta este término en inglés, sin su traducción al castellano.

Según la IEEE (1990), la descripción de un caso de prueba incluye, al menos, la siguiente información:

- Precondiciones.
- Conjunto de valores de entrada.
- Conjunto de resultados esperados.
- Pos-condiciones esperadas.
- Identificador único.
- Dependencia de otros casos de prueba.
- Referencia al requisito que será.
- Forma en la cual se debe ejecutar el caso de prueba y verificar los resultados (opcional).
- Prioridad (opcional).

De acuerdo con estas referencias y, si sostenemos un enfoque sistémico, el caso de prueba contiene entradas, proceso y salidas.

La **entrada** comprende lo siguiente:

- El **conjunto de datos** que ingresar para ejecutar el caso de prueba.
- Las **precondiciones** necesarias para poder ejecutar correctamente el caso de prueba. Estas pueden ser:
  - **De entorno**, es decir, en qué sistema operativo o con qué productos está corriendo, entre otras.
  - **De software**, es decir, en qué lugar se debe empezar o qué opción de menú tiene que estar disponible, entre otras.

El **proceso** es el conjunto de acciones que realizar para ejecutar el caso de prueba que, si partimos de los datos de entrada, deberán producir los datos esperados como resultado.

La **salida** es el conjunto de resultados que deben estar preestablecidos, es decir, ser conocidos y previstos antes de la primera ejecución del caso de prueba o, lo que es lo mismo, se deben especificar en el momento de definir el caso de prueba.

¿Cuál es la diferencia entre un buen caso de prueba caso de prueba exitoso?

Un buen test case es aquel que tiene alta probabilidad de detectar un defecto aún no descubierto; en cambio, se llama *caso de prueba exitoso* a aquel que detecta un defecto aún no descubierto.

Como corolario de lo anterior, se dice que el desarrollo exitoso puede llevar a pruebas no exitosas, es decir, que no encuentra defectos.

Verificación y validación

Muchas veces empleamos de manera indistinta las palabras *validar* y *verificar*, pero en ingeniería de software tienen significados diferentes.

¿Por qué hablar de verificación y validación en pruebas de sistemas? Según muchos autores dedicados a la ingeniería de software, como Roger S. Pressman (2006), la verificación y validación contienen muchas de las actividades incluidas en el aseguramiento de la calidad de software, y las pruebas tienen un papel bien importante en este punto.

La norma ISO 9000 (International Organization for Standarization [ISO], 2005) define a estos términos en sus puntos 3.8.4 y 3.8.5 de la siguiente manera:

> Verificación: Confirmación mediante la aportación de evidencia objetiva (3.8.1) de que se han cumplido los requisitos (3.1.2) especificados…
> Validación: Confirmación mediante la aportación de evidencia objetiva (3.8.1) de que se han cumplido los requisitos (3.1.2) para una utilización o aplicación específica prevista. (https://goo.gl/L875XN)

Considerando estas definiciones, se puede decir que ambas son una comprobación, con la diferencia de que la validación es la comprobación con respecto a su usabilidad final, es decir, sirve para corroborar si funciona o no el producto para el cual se construyó. Para aclarar un poco más esto: implica observar si un producto, al final del proceso de desarrollo, cumple con sus funciones técnicas. Aplicado a la ingeniería de software, podemos decir:

Verificación y validación *implican un proceso que asegura que el software bajo desarrollo o cambio implementado cumple los siguientes requisitos:*

1. *Satisface los requerimientos funcionales y otros requerimientos (Verificación).*
2. *Cada paso en el proceso de desarrollo de software genera el producto correcto (Validación).*

A partir de aquí, Boehm (1981) plantea dos preguntas clave que ayudan a comprender ambos conceptos.

¿Estamos construyendo el sistema correctamente?, ¿a qué responde?

Responde a la verificación.

¿Estamos construyendo el sistema correcto?, ¿a qué responde?

Responde a la validación.

Actualmente, las pruebas son vistas como algo más que una validación. Como muestra de ello, tomamos la comparación que hace Boris Beizer (1983) entre las pruebas de sistemas y la inspección de un edificio, en la que muestra que, si se prueba únicamente considerando los requisitos del usuario final, sería como si se inspeccionara un edificio considerando solo el trabajo realizado por el diseñador de interiores, sin tener en cuenta los cimientos, las vigas ni las instalaciones.

Normas y estándares para pruebas de sistemas: norma ISO 9126, IEEE, CMM, TMM

Volviendo a la perspectiva histórica, allá por la década de los 80, junto con la tercera visión de las pruebas, consideradas como un proceso, comienzan los primeros estándares relacionados. Estos han crecido y evolucionado y hoy contamos con una serie de normas, modelos y estándares relacionados con las pruebas. ¿Qué relación tiene esto con las pruebas? Como profesionales de la informática, debemos estar de acuerdo con que el objetivo último de la ingeniería de software es producir software de alta calidad. En su desarrollo se trabaja con base en expectativas de los usuarios o clientes (decimos cliente o usuario sabiendo las diferencias entre uno y otro). Y es aquí donde está el problema, ya que normalmente no se expresa con claridad lo que realmente se quiere.

La calidad se basa en lo que alguien quiere y en cómo alguien lo quiere.

Aplicado a la **calidad de software**, Pressman (2006) la define como:

"El cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se esperan de todo software desarrollado profesionalmente". (p. 463)

Aquí destacamos dos puntos relevantes. Primero, habla de requisitos de funcionalidad, que es todo aquello que se dice o expresa que se espera del software. Es lo que podemos llamar características explícitas. Y segundo, habla de las características implícitas. En software ocurre muy a menudo que el cliente no solicita algo, pero luego lo espera. Por ejemplo, el cliente solicita que su sistema emita determinado reporte, pero deja como implícito que esté en un tipo de fuente específico o que se genere en un tiempo acorde a las necesidades de su negocio.

Comprendimos lo que es calidad de software y vimos los conceptos de verificación y validación; por lo tanto, podemos afirmar que las pruebas son una actividad más que se realiza como parte del aseguramiento de calidad, y de ahí que las pruebas de sistemas deben ir de la mano con la calidad. Esto no significa que:

- las pruebas sean una barrera para todos los defectos existentes en el software;
- aumentar las pruebas incremente la calidad.

La calidad no se compra, es un esfuerzo entre todos. Al respecto, dice Pressman (2006): “Si no está ahí antes de que empiece la prueba, no estará cuando termine” (p. 384).

La calidad debe medirse y necesitamos una forma de medirla que sea comprensible para todos y esté expresada en un lenguaje accesible para indicar dónde están los principales problemas que atacar.

Según Pressman (2006), los factores que afectan la calidad de software se dividen en dos grupos. Por un lado, tenemos los factores directos, que son los que, por ejemplo, se miden directamente a través de la cantidad de defectos descubiertos sobre casos de pruebas corridos. Por su lado, los factores indirectos son los que miden indirectamente, como, por ejemplo, la facilidad de uso del producto.

A veces resulta difícil y hasta imposible desarrollar medidas directas de estos factores, pero, aunque se midan subjetivamente o sean derivados de una lista de comprobación, por lo menos se tiene una idea o se le asigna una graduación, y esto ayuda como base para evaluar la calidad del software.

Si listamos los estándares relacionados, tenemos, por un lado, la norma ISO 9126 y el modelo de McCall, orientados a los factores o atributos de calidad en relación con el producto[1]. Más adelante veremos la ISO 9001 y el modelo CMMI (*capability maturity model integration*), orientados al proceso de pruebas.[2]

Modelo de McCall

El modelo de Jim McCall es aquel que busca disminuir la brecha entre usuarios y desarrolladores. Fue creado inicialmente para la Fuerza Aérea de Estados Unidos en 1977 y es uno de los más famosos actualmente. Se orienta en un número de factores de calidad que reflejen las prioridades de ambos.

El modelo establece una jerarquía de perspectivas, factores, criterios de calidad y, como fin último, las métricas. Es muy rico porque logra definir métricas para los factores de calidad, además de dar un significado para cada uno de ellos. Cuando se dice que se requiere que el software sea

[1] Cuando decimos *producto*, nos referimos al producto que es generado por un proceso de desarrollo de software; puede ser un sistema, un programa, una aplicación, un componente, etcétera.
[2] Para cada uno de los modelos, estándares y normas, veremos lo que nos interesa a los efectos de esta materia. No se desarrollará el contenido de cada uno de ellos.

fácil de usar, lo que se pide es que cumpla con el factor de **facilidad de uso**. Pero ¿qué significa? La facilidad de uso puede tener definiciones dispares para diferentes personas, y lo que es fácil de usar para unos no es lo es para otros. Este modelo permite unificar. Explicamos, entonces, cómo se estructura.

El modelo de McCall se basa en 11 factores de calidad, organizados en torno a tres perspectivas:

- **Operación del producto**: corrección, confiabilidad, facilidad de uso, integridad y eficiencia.
- **Transición del producto**: portabilidad, facilidad de reutilización e interoperabilidad.
- **Revisión del producto**: facilidad de mantenimiento, flexibilidad y facilidad de prueba.

A partir de esos factores, toma a cada uno de ellos y los abre en criterios. Por ejemplo, el factor facilidad de uso se abre en facilidad de operación, facilidad de comunicación y facilidad de aprendizaje. Siguiendo con otro ejemplo, flexibilidad se abre en autodescripción, capacidad de expansión, generalidad y modularidad. Sobre cada uno de los factores, puede consultarse el significado en la bibliografía básica.

A continuación, plantearemos un ejemplo de aplicación de cierta lista de chequeo para la obtención de una de las métricas. Se trata de una simple lista de chequeo orientada al criterio **completitud**. Para evaluar la completitud en un producto de software, es necesario dar respuesta a esta lista de comprobación.

Las letras *R*, *D* e *I* indican si la lista de comprobación es aplicable a los requisitos (R), el diseño (D) o la implementación (I). Se contesta a estas preguntas con **sí** o **no**.

1. No hay referencias ambiguas (R, D, I).
2. Todas las referencias a datos definidas son calculadas u obtenidas de una fuente externa (R, D, I).
3. Todas las funciones definidas son utilizadas (R, D, I).
4. Todas las referencias a funciones están definidas (R, D, I).
5. Se han definido todas las condiciones y procesamientos para cada punto de decisión (R, D, I).
6. Concuerdan todos los parámetros de llamada a funciones definidos y referenciados (D, I).
7. Todos los informes de problemas se han resuelto (R, D, I).
8. El diseño concuerda con los requisitos (D).
9. El código concuerda con el diseño (I).

Luego se aplica la siguiente fórmula matemática, que da como resultado el grado o nivel de calidad para dicho atributo:

$$\frac{(\text{número de SI para R/6}) + (\text{número de SI para D/8}) + (\text{número de SI para I/8})}{3}$$

De esta forma, la medida para la completitud es un número entre 0 y 1.

La **corrección** depende de la **completitud**, la **trazabilidad**, y la **consistencia**. Entonces, de la misma manera que existe la lista de comprobación para la completitud, existe otro recurso aplicable a la trazabilidad y así sucesivamente.

La medida para la corrección, por ejemplo, se calculará aplicando la fórmula: **(x + y + z)/3**

> Donde ***x*** es la medida para la *completitud*, ***y*** la medida para la *trazabilidad* y ***z*** la medida para la *consistencia*. Para que todas las métricas se puedan combinar sin problemas, lo habitual es que se normalicen sus valores dentro del intervalo [0,1]. (Herrán Suarez, s.f., https://bit.ly/2EbSEoU)

¿Qué indicaría el valor más cercano a 1?

¿Qué indicaría el valor más cercano a 0?

Llegamos hasta el desarrollo de aplicación de una métrica para que se pueda demostrar que, como señaláramos, aunque se midan subjetivamente o derivados de una lista de comprobación, por lo menos se tiene una idea o se le asigna una graduación, y esto ayuda para evaluar la calidad del software.

Norma ISO 9126

La norma ISO 9126 es un estándar internacional para la evaluación de la calidad de software que, si bien se deriva del anterior modelo de McCall, recoge las investigaciones de muchos modelos de calidad propuestos por profesionales del tema de los últimos treinta años. Ya a partir del 2005, fue reemplazada por la ISO 25000:2005.

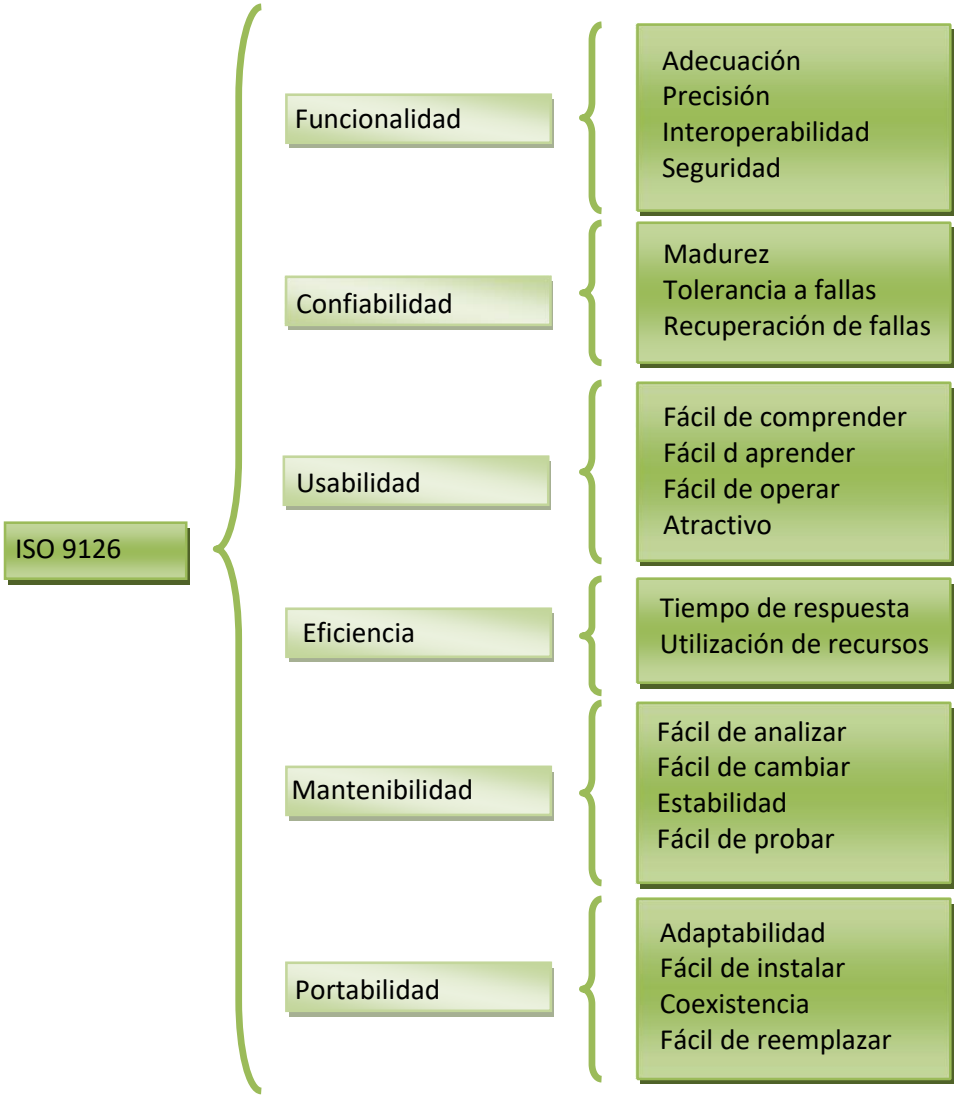
La ISO 9126 (2001) propone un modelo de calidad que se divide en tres vistas: interior, exterior y en uso. Estas vistas están compuestas por 6 características que se dividen en sub-características, y que éstas a su vez se componen de atributos, es decir una entidad que puede ser verificada o medida en el producto *software*.

De la misma manera que se definen tres vistas, las mediciones realizadas dan como resultado una serie de métricas que se clasifican en tres categorías según sea su naturaleza:

- Métricas internas: son aquellas mediciones que no dependen de la ejecución del *software*, se llaman métricas estáticas.
- Métricas externas: son aquellas que son aplicables al *software* en su ejecución, se llaman métricas dinámicas.
- Métricas de uso: son aquellas que sólo están disponibles cuando el producto final es utilizado en condiciones reales.

En el siguiente gráfico, se pueden ver las seis características y las subcaracterísticas asociadas.

Figura 1: Características y subcaracterísticas



Fuente: elaboración propia.

Las visiones de calidad permiten definir modelos de calidad (conjunto de características). Estos modelos, a su vez, permiten determinar propiedades del producto y del proceso. Con base en estas propiedades, definimos métricas y, finalmente, a partir de estas, se realiza la **gestión de calidad** para asegurarla.

En ninguno de los dos casos vistos, se tiene el tema resuelto respecto de las mediciones de la calidad de software, pero ambos casos proveen un

entorno para que las organizaciones definan un modelo de calidad para su producto software.

Cada organización tiene la tarea de especificar precisamente su propio modelo, con base en estos u otros, por medio de la incorporación de otros atributos o factores que resulten necesarios alcanzar. Esto se puede hacer especificando los objetivos para las métricas de calidad (evalúan el grado de presencia de los atributos de calidad).

Hasta aquí vimos lo que está relacionado con el producto, de modo que ahora continuaremos con la norma ISO 9001 y el modelo CMMI, orientados al proceso de pruebas.

Norma ISO 9000:2005

Se puede decir que son reglas básicas de calidad, independientes del producto o servicio de que se trate. Se definen como un conjunto de buenas prácticas de fabricación de un producto u ofrecimiento de un servicio. Es un modelo que brinda la seguridad de que el proveedor tiene la capacidad de producir los bienes o servicios requeridos.

En forma complementaria a la ISO 9000, existe el documento ISO 9000-3, el cual es una guía específica para la aplicación de la norma ISO 9001 al desarrollo y mantención de software. La ISO 9000-3 está dividida en 3 partes:

- Marco de trabajo.
- Actividades en el ciclo de vida.
- Actividades de apoyo.

Específicamente en lo relativo a las pruebas en general, establece los siguientes elementos principales:

a) Las pruebas deben ser realizadas en varios niveles, desde el ítem de software individual hasta el producto completo.

b) En algunas ocasiones la validación, la prueba de campo y la prueba de aceptación pueden ser una sola actividad.

c) El documento que describe el plan de pruebas puede ser un documento independiente o estar compuesto de varios documentos.

Relativo a la planificación de las pruebas

Dentro de la planificación de las pruebas, se debe contemplar lo siguiente:

a) Planes para realizar las pruebas de unitarias, integración, sistema y aceptación.
b) Definición de casos de pruebas, datos de prueba y resultados esperados.
c) Tipos de pruebas que realizar, por ejemplo, prueba funcional, prueba de límites, pruebas de rendimiento o prueba de usabilidad.
d) Especificar ambiente de pruebas, herramientas y técnicas de pruebas de software.
e) Criterios de completitud en las pruebas.
f) Documentación disponible (requerimientos, diseño, otros).
g) Personal y entrenamiento requerido.

Relativo a la validación del proveedor

Antes de que el software sea entregado y luego realizada la prueba de aceptación con el usuario; el proveedor que realizó el producto debe validar si la operación del mismo es completa, y en la medida que sea posible, bajo las mismas condiciones de cómo va a ser el ambiente en producción de acuerdo a lo especificado con el cliente.

Relativo a la aceptación del comprador

Cuando el proveedor está listo para entregar el producto validado, el comprador debe juzgar si el producto es aceptable de acuerdo con los criterios estipulados en el contrato. El método de manejar los problemas detectados durante la prueba de aceptación debe ser acordado entre el comprador y el proveedor y dejarse debidamente documentado. (Guerrero, 2008, https://bit.ly/2QFFxTF)

Recursos y personal para verificación

El proveedor deberá identificar internamente los requerimientos, proveer los recursos adecuados y asignar personal entrenado para las actividades de verificación. Las actividades de verificación deben incluir:

- inspección;
- pruebas;
- monitoreo del diseño, producción, instalación y procesos de servicio o productos;
- revisiones del diseño;
- auditorías del sistema de calidad, procesos o productos.

Todo debe ser realizado por personal independiente de aquel que tiene directa responsabilidad en el trabajo que se desarrolla.

Modelo CMMI

Tiene su origen en el Departamento de Defensa de Estados Unidos. Es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software. Fue uno de los mayores aportes para superar la llamada *crisis de software*.

¿Cuál es la idea de este modelo?

Evaluar el proceso:

- ¿Cuán "bueno" es nuestro actual proceso?
- ¿Cuáles son nuestras fortalezas y debilidades?
- ¿Qué debemos hacer para mejorar?

Y, mejorarlo:

- Pasos requeridos para iniciar el cambio.
- Acciones que resultan más rentables.
- Métodos para un control cuantitativo.
- Cómo ir de un enfoque de corrección de problemas a un enfoque de prevención.

Se estructura en 5 niveles, que explicaremos brevemente:

Inicial (nivel 1): se opera sin procedimientos formales, sin estimaciones ni planes y las herramientas no están bien integradas. No hay buena información acerca de los problemas y asuntos internos. No existe un control de cambio riguroso. Por todo esto, la instalación y mantenimiento de software a menudo presenta problemas.

Repetible (nivel 2): en este estado, superior al anterior, la organización ha logrado un rendimiento estable a través de una administración de compromiso, costos, planificación y control de cambios. Esta estabilidad se traduce en un conjunto de acciones diseñadas para lograr controlar los costos y plazos. Hay organización, administración de proyectos, administración del proceso de desarrollo y tecnología.

Definido (nivel 3): ya en este nivel, hay un proceso estándar definido, se puede decir que hay metodología. Si bien hay escasos datos cuantitativos, se manejan las contingencias, se puede mejorar ordenadamente y facilitar la introducción de tecnología de apoyo.

Administrado cuantitativamente (nivel 4): completo proceso de análisis y medidas, tales como calidad, costo o productividad, entre otros. Existe un preciso conocimiento de los procesos como base para una continua calidad de sus productos y un mejoramiento de la productividad. Existe la asignación de recursos para la recopilación y mantención de los datos.

Optimizado (nivel 5): en este nivel la completa información acerca del proceso de software es usada para evaluar la efectividad de este proceso y hacer ajustes de forma regular. Esto provee las bases para una continua mejora del desarrollo y un ordenado y planificado aumento de la productividad.

Si bien el término *pruebas de sistema* no es un término usado en CMMI, debido a sus múltiples interpretaciones entre diferentes disciplinas no implica que las pruebas no sean contempladas por el modelo. Por razones de consistencia y claridad, en vez de *sistema* se usan los términos *producto* y *componente de producto*, y, en vez de *pruebas*, se usaron los términos *verificación* y *validación*, porque las pruebas pueden ser parte de la verificación o la validación.

Las pruebas en el modelo CMMI están tratadas principalmente en las áreas de proceso **verificación** y **validación**. Si bien ya vimos el significado de estos conceptos, estos se pueden ampliar mediante la consulta directamente en el modelo.

IEEE

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) es la sociedad técnica profesional más grande del mundo dedicada, entre otras cuestiones, a la estandarización. Su trabajo es promover la creatividad, el desarrollo y la integración, compartir y aplicar los avances en las tecnologías de la información, electrónica y ciencias en general para beneficio de la humanidad y de los mismos profesionales. (Ecured, s.f.)

Se listarán algunos de sus estándares relacionados con las pruebas, que son los más consultados dentro de la comunidad de probadores, lo que demuestra la importancia internacional que adquieren las pruebas. Entre ellos, encontramos:

IEEE 610 (IEEE, 1990): estandariza una terminología. Si bien no es una terminología específica de pruebas, ya que es de ingeniería de software, incluye todos los términos relacionados con las pruebas, por lo cual es utilizada en común por los profesionales de pruebas.

IEEE 829 (IEEE, 2008a): desarrolla una estructura de plan de pruebas maestro acreditada que incluye la descripción de un caso de prueba, el

contenido de un informe de estado de pruebas o la estructura de un informe de incidencias, entre otros puntos.

IEEE 730 (IEEE, 2002): estructura y da los contenidos de un plan de calidad, asociación indispensable con las pruebas de sistemas.

IEEE 1028 (IEEE, 2008b): dedicado exclusivamente a las revisiones, cómo deben hacerse y qué deben incluir.

Referencias

Beizer, B. (1983). *Software Testing Techniques*. New York, US: Van Nostrand-Reinhold.

Boehm, B. (1981). *Software Engineering Economics*. New York, US: Prentice Hall.

Ecured. (s.f.). IEEE. Recuperado de: https://www.ecured.cu/IEEE

Guerrero, J. (2008). Pruebas de Software. Recuperado de: https://www.u-cursos.cl/diplomados/2008/0/DGCSTPS/1/material_docente/previsualizar?id_material=198490

Herrán Suarez, J. (s.f.). *Evaluación de Software. Trabajo colaborativo N°3.* Recuperado de: https://es.scribd.com/document/248232317/Ttrabajo-colaborativo-1-evaluacion-de-software

Institute of Electrical and Electronics Engineers (IEEE). (1990). *Standard 610: Glossary of Software Engineering Terminology*. Recuperado de https://ieeexplore.ieee.org/document/159342/

Institute of Electrical and Electronics Engineers (IEEE). (2002). *Standard 730: Standard for Software Quality Assurance Plans*. Recuperado de https://ieeexplore.ieee.org/servlet/opac?punumber=8063

Institute of Electrical and Electronics Engineers (IEEE). (2008a). *Standard 829: Standard Software and System Documentation.* Recuperado de https://ieeexplore.ieee.org/servlet/opac?punumber=4578271

Institute of Electrical and Electronics Engineers (IEEE). (2008b). *Standard 1028: Standard for Software Reviews and Audits*. Recuperado de https://ieeexplore.ieee.org/servlet/opac?punumber=4601582

International Organization for Standarization (ISO). (2001). *Software engineering — Product quality — Part 1: Quality model* (ISO/IEC estándar N.° 9126-1). Recuperado de https://www.iso.org/standard/22749.html

International Organization for Standarization (ISO). (2005). *Sistemas de gestión de la calidad — Fundamentos y vocabulario* (Norma ISO N.° 9000). Recuperado de https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-3:v1:es

International Software Testing Qualifications Board (ISTQB). (2011). *Fundamentals of Testing*. *Certified Tester. Foundation Level Syllabus*. Recuperado de https://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html

Pressman, R. (2006). *Ingeniería de software. Un enfoque práctico* (6.ª ed.). México D. F., MX: McGraw-Hill.