UNIVERSIDAD DE SONSONATE

FACULTAD DE INGENIERÍA Y CIENCIAS NATURALES

ANÁLISIS DE SISTEMAS

CATEDRÁTICO:

ING. ANTONIO HUMBERTO MORÁN NAJARRO

TEMA:

ISTQB- Fundamentos de pruebas

ENTREGADO POR:

CARLHOS EDGARDO FIGUEROA TEJADA

GUILLERMO ALEXANDER AVILES CORTEZ

GERARDO JOSE VILLEDA ERAZO

Contenido

Introducción

¿Es posible construir software que no falle? Hoy en día estamos acostumbrados a que el software falle.

Al construir software habitualmente se cometen errores. En la industria, la técnica para solucionar los problemas derivados de dichos errores, serán las pruebas de software ("testing"), que consistirán en una serie de pasos realizados antes y después de la construcción de este software.

Históricamente, por la ausencia o por la incorrecta realización de las pruebas sobre el software, se han producido varios desastres que han llegado, no solo a tener consecuencias económicas nefastas, sino a producir la pérdida de vidas humanas.

Las pruebas son parte fundamental de cualquier proyecto, ya que nos ayudarán a tener mejores resultados, ofreceremos una calidad mayor de nuestro producto y en consecuencia los clientes estarán más satisfechos.

Según (Burnstein, 2003), el proceso de prueba de software tiene tres procesos principales: el desarrollo de los casos de prueba, la ejecución de estos casos de prueba y el análisis de los resultados de la ejecución.

ISTQB- Fundamentos básicos de las pruebas

¿Qué son las pruebas?

Antes de ver una definición de prueba, veremos la percepción que tienen los desarrolladores acerca de las pruebas. Según (Myers, 2004), los desarrolladores siguen las siguientes definiciones que llevan a una percepción falsa:

- "Las pruebas son el proceso de demostrar que no hay errores presentes".
- "El propósito de las pruebas es demostrar que un programa realiza las funciones indicadas correctamente"
- "Las pruebas son el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer".

Jhon Myers indica que estas definiciones están mal planteadas. Cuando probamos un programa se quiere aportar un valor añadido a lo que estamos probando, elevar la calidad y fiabilidad y esto nos lleva a tener que encontrar y eliminar los errores en el programa.

Esto quiere decir que no tenemos que probar un programa para demostrar que funciona, sino que tenemos que partir de la suposición de que el programa va a contener errores. La definición de prueba que aporta Myers es:

- "La prueba es el proceso de ejecución de un programa con la intención de encontrar errores".

Pero, ¿Por qué se toma esta definición como válida y las anteriores no?

Si tomamos la primera definición mencionada - "El testing es el proceso de demostrar que no hay errores presentes", psicológicamente estaremos dirigidos hacia esa meta y tenderemos a seleccionar los casos de prueba con una baja probabilidad de causar que el programa falle. Si por el contrario tomamos como objetivo demostrar que un programa tiene fallos, tendremos una mayor probabilidad de encontrar errores.

Si tomáramos la definición de "Testing es el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer" estaríamos dando por sentado que un programa que hace lo que se supone debe hacer, no tiene errores y eso es un error ya que, aunque el programa realice su función, puede contener errores.

El ISTQB (International Software Testing Qualifications Board), una organización sin ánimo de lucro creada en el año 2002 por empresas, instituciones, organizaciones y personas especializadas en el campo de las pruebas y la industria del software, define las pruebas como (ISTQB, 2021):

- "El proceso que consiste en todas las actividades del ciclo de vida, tanto estáticas como dinámicas relacionadas con la planificación, preparación y evaluación de productos de software y productos relacionados con el trabajo para determinar que cumplen los requisitos especificados, para demostrar que son aptos para el propósito y para detectar defectos".

Cem Kaner, profesor de Ingeniería de software en el instituto tecnológico de Florida, es uno de los defensores y principales gurús de las pruebas de software, las define como (Kaner, D, & D, 2021):

- "Las pruebas de software son la investigación empírica y técnica realizada para facilitar a los interesados información sobre la calidad del producto o servicio bajo pruebas".

Kaner introduce la figura del técnico que mediante la investigación aportará datos sobre la calidad del producto y no se centrará únicamente en la detección del fallo.

Edsger W. Dijstra, científico de la computación entre cuyas contribuciones a la ciencia esta "la solución del problema del camino más corto", también conocido como el algoritmo de Dijstra, define el proceso de pruebas como (Dijkstra, 2021):

- "Las pruebas de software pueden ser una manera muy eficaz de mostrar la presencia de errores, pero son totalmente inadecuadas para mostrar su ausencia."

Todas y cada una de estas definiciones tienen algo en común, todas se centran en mayor o menor manera en la detección de errores.

Para terminar de entender las pruebas debemos diferenciar los términos error, fallo y defecto. Estos conceptos están relacionados entre sí, pero tienen significados diferentes. Para comprender estas palabras y por ende las pruebas, vamos a ver como las define el ISTQB:

- "Una persona puede cometer un error que a su vez puede producir un defecto en el código de programa o en un documento. Si se ejecuta un defecto en el código, el sistema puede no hacer lo que debiera (o hacer algo que no debiera), lo que provocaría un fallo. Algunos defectos de software pueden dar lugar a fallos, pero no todos los defectos lo hacen."

Así pues tenemos:

- Error: está provocado por la acción humana, por ejemplo el error lo provocará el desarrollador que realizará una incorrecta interpretación de un método del programa que producirá un resultado no esperado.

- Defecto: provocado por un error de implementación, por ejemplo el defecto lo provocará el haber utilizado el operador "x+y>z" en vez de "x+y=>z"

- Fallo: al ejecutar el programa con un defecto obtendremos resultados no deseados, por ejemplo, cuando el resultado de la suma de los dos componentes fuese igual, no obtendríamos los mismos resultados al compararlos con las sentencias indicadas anteriormente. En sistemas muy complejos, como pueden ser una lanzadera espacial o una central eléctrica, pueden llegar a producir efectos catastróficos.

¿Por qué son importantes las pruebas?

Hoy en día, forman parte de nuestras vidas multitud de sistemas que contienen software, como por ejemplo los coches, smartphones, sistemas de producción de energía, programas bancarios, etc.

Para responder a la pregunta anterior vamos a ver una serie de sucesos ocurridos durante el siglo XX, que ejemplifican lo importante que puede llegar a ser probar el software antes de ponerlo en producción:

- El lanzamiento de la lanzadera Ariane-5 vuelo 501 (1996) fue considerado uno de los fallos de programación más caros de la historia hasta ese momento, sólo la carga que llevaba tenía un valor de 500 millones de dólares. Fue un cohete lanzado por la ESA (European Space Agency´s o agencia espacial europea) destruido aproximadamente a los 40 segundos de ser lanzado. Según el informe de la ESA (ESA, 2021), el fallo de la Ariane 501 fue causado por la completa pérdida de guía e información de orientación treinta y siete segundos después del comienzo de la secuencia de ignición del motor principal. Esta pérdida de información se debió a errores de especificación y diseño en el software del sistema de referencia inercial. Las extensas revisiones y test llevados a cabo durante el programa de desarrollo del Ariane-5 no incluyeron el adecuado análisis y prueba del sistema de referencia inercial o del sistema de control de vuelo completo, lo cual podría haber detectado los fallos potenciales.

- El lanzamiento de la sonda Mariner 1 de la NASA (1962), tuvo que ser abortado por un fallo de software que afectaba a la trayectoria del cohete. El cohete fue destruido antes de que soltara la sonda que transportaba, ya que corría peligro de estrellarse en las rutas marítimas del atlántico norte. El coste aproximado del proyecto de la sonda Mariner 1 fue de 554 millones de dólares (COMPUTERWORL, 2021).

- Therac 25, una máquina de radioterapia producida por la Atomic Energy of Canada Limited (AECL) en 1985, fue la causante de la muerte de tres personas directamente y otras tres sufrieron graves daños por la administración de sobredosis masivas de radiación. Una de las razones de que se produjera esta sobredosis fue un mal diseño del software, el código fuente no había sido revisado de forma independiente (WIKIPEDIA, 2021).

- Thomas Nicely, de la Universidad de Lynchburg, descubrió un error en la unidad de coma flotante del Pentium en 1994. El divisor en la unidad de coma flotante contaba con una tabla de división incorrecta, donde faltaban cinco entradas sobre mil, que provocaba errores de redondeo. Aunque el error afectaba a pocos usuarios, este hecho hizo mucho daño a la imagen de Pentium y el costo total fue de 475 millones de dólares. Finalmente reemplazó los chips de todos los usuarios que lo solicitaron (Patton, 2005).

- En otoño de 1994, la compañía Disney lanzó su primer juego multimedia en formato CD-ROM. Las ventas fueron enormes ya que fue uno de los juegos más comprados la navidad de ese año. El 26 de diciembre, el departamento de atención al cliente de Disney se vio desbordado por las llamadas de un gran número de usuarios descontentos que habían comprado el juego. Resulta que Disney no realizó pruebas en

los diferentes modelos de PC disponibles en el mercado. Solo se realizaron pruebas sobre los PC que utilizaban los desarrolladores (Patton, 2005).

Como se puede apreciar en los ejemplos anteriores, no probar adecuadamente el software, antes de ponerlo en producción, puede producir no sólo pérdidas económicas, sino también daños personales, llegando incluso a producir la muerte en algunos casos.

A día de hoy el funcionamiento de casi todas las empresas depende en gran medida del software, ya sea por el sistema de finanzas de dicha empresa o por la maquinaria que lleva a cabo la fabricación de los productos, por lo que las empresas dependen del funcionamiento del software y de que éste pueda llegar a causar grandes fallos como los mencionados anteriormente que llevan a la pérdida de miles de millones de euros. No a todas las empresas les afectan de la misma manera los fallos producidos en el software, por lo que tenemos que evaluar los riesgos de éste, ya que pueden llegar a producir pérdidas irreparables.

¿Cuál es el objetivo de las pruebas?

El objetivo principal de las pruebas es aportar calidad al producto que se está desarrollando.

Pero, ¿qué es calidad?

Este término es definido por muchas organizaciones e ilustres en el mundo de la calidad de formas diferentes:

- ISO 9000 es un conjunto de normas sobre calidad y gestión de calidad, la define como "la calidad es el grado en el que un conjunto de características inherentes cumple con las requisitos" (ISO 2500 - WIKIPEDIA, 2021).
- ISO 25000 es un conjunto de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software, dice: "la calidad es el grado en que el producto de software satisface las necesidades expresadas o implícitas, cuando es usado bajo condiciones determinadas" (ISO 2500 - WIKIPEDIA, 2021).
- Philip Bayard Crosby, que contribuyó a las prácticas de la gestión de la calidad, la define como "Conformidad con los requisitos" (TEGSOLTIONS, 2021).
- Armand V. Feigenbaum, que diseñó el concepto del control total de la calidad, la define como "satisfacción de las expectativas del cliente" (TEGSOLTIONS, 2021).
- Genichi Taguchi, que desarrolló una metodología para la aplicación de estadísticas para mejorar la calidad de los productos manufacturados, dice: "calidad es la pérdida (monetaria) que el producto o servicio ocasiona a la sociedad desde que es expedido" (WIKIPEDIA, 2021).

Podríamos seguir indicando definiciones de calidad de numerosos ilustres en el tema, en donde cada uno nos diría su definición y su manera de llegar a esta calidad. Hay dos puntos principales que tienen casi todas las definiciones de calidad: la satisfacción del cliente y el cumplimiento de los requisitos del producto.

La ISO 25010 (ISO, 2021), norma donde se trata la calidad del producto software, nos indica qué características tiene que tener un software para tener la calidad deseada:

Ilustración 1. Características CALIDAD ISO 25010 (ISO, 2021)

Habiendo definido que las pruebas buscan la calidad en sus productos y también el concepto de este, podemos intuir la importancia del control de la calidad de un producto o de su QA ya que obtener la calidad de un producto es:

- Obtener la confianza del consumidor o cliente
- Obtener la mejor rentabilidad del producto
- Mejorar los estándares generales del producto (Desarrollo)
- Motivación de los trabajadores de la empresa
- Mejora en la imagen y prestigio de la empresa

“El coste de la calidad se obtiene de la suma de todos los costes que desaparecerían si no hubiese problemas de calidad.” -CEUPE, magazín.

¿Cómo llevamos a cabo las pruebas?

Para llevar a cabo las pruebas verificaremos el comportamiento del programa sobre un conjunto de casos de prueba. Estos casos de prueba se generarán mediante técnicas y estrategias específicas de pruebas que nos ayudarán a conseguir la búsqueda de los errores de un programa.

Pero antes de continuar, y teniendo en cuenta las definiciones anteriores, vamos a plantearnos una pregunta muy importante que tendremos que tener en cuenta cuando veamos las estrategias y técnicas de prueba.

¿Es posible encontrar todos los errores de un programa?

En casi toda la documentación estudiada para este proyecto, (ISTQB, 2021) , (Myers, 2004) , (Pressman, 2005), etc., nos encontraremos con un apartado donde vemos que la respuesta a esta pregunta es negativa, no suele ser práctico, en lo que a tiempo y coste empleado en un proyecto se refiere, y la gran mayoría de las veces es imposible encontrar todos los errores de un programa.

Para encontrar los errores, dos de las técnicas más utilizadas en las pruebas son las técnicas de "caja blanca" y "caja negra".

La técnica de pruebas de caja negra, consiste en ver el programa que queremos probar como una caja negra despreocupándonos del comportamiento interno y concentrando el esfuerzo en encontrar el comportamiento incorrecto, de acuerdo a las especificaciones de dicho programa, teniendo sólo en cuenta las entradas y salidas de dicho programa.

La técnica de pruebas de caja blanca, al contrario de las pruebas de caja negra, consiste en verificar la estructura interna de un programa.

Si utilizáramos el método de pruebas de caja negra para encontrar todos los errores del programa, como respuesta a la pregunta anterior, generando un caso de prueba para cada entrada, el número de casos de prueba que tenemos que utilizar sería muy elevado y no sería productivo. Para ver esta afirmación se va a proponer un ejemplo. Un programa que clasifica tres entradas numéricas de carácter entero representado cada entrada la longitud de un lado de un triángulo. Dicho programa procesará los datos y mostrará al usuario de qué tipo de triángulo se trata, escaleno, isósceles o equilátero. Ahora bien, tomando como estrategia las pruebas de caja negra, para probar el programa a fondo, tendríamos que crear, como dijimos anteriormente, un caso de prueba para cada entrada del programa. Generando un caso de prueba para encontrar todos los tipos de triángulo, tendríamos un número de casos de prueba muy elevado, es decir, si tenemos la entrada en el programa 3-3-3, tendríamos un triángulo equilátero, pero también tendríamos un triángulo equilátero para las entradas 5-5-5 y 300-300-300. Para probar todos los casos y encontrar todos los errores no sólo probaríamos los casos válidos sino que también tendríamos que probar todos los casos incorrectos, es decir, todo tipo de entradas válidas y no válidas. Esto nos llevaría a un número de casos de prueba infinito, que sería imposible de probar. Si nos parece complicado encontrar todos los errores de un programa con tres entradas, entonces, un programa mucho más complejo, como puede ser un programa que realice la gestión de las cuentas de un empresa con miles de entradas y salidas, sería más complicado todavía.

Si, por lo contrario, eligiéramos las pruebas de caja blanca para contestar a la pregunta, no solo tendríamos que probar todas las líneas de un programa sino que tendríamos que realizar todos los posibles caminos lógicos que se pueden producir, ya que este tipo de método, como veremos más adelante, se fundamenta en las sentencias de tipo "if","while","case","for", etc. Por lo tanto, al igual que en las pruebas de caja negra, tendríamos un número de casos de prueba astronómicamente grande, que sería inviable para cualquier tipo de proyecto.

Como veremos, no podemos probar todas las combinaciones y no podemos realizar un test para demostrar que el programa está libre de errores por ello tenemos optimizar las pruebas. Para ello vamos a establecer prioridades que nos marcarán las limitaciones y los objetivos de un proyecto de software.

En el libro (Everett & MacLeod Jr, 2007), establecen cuatro prioridades:

- Identificar la magnitud y las fuentes de riesgo del desarrollo reducible por las pruebas.
- Realizar pruebas para reducir los riesgos de un negocio identificados.
- Saber cuándo se ha completado la prueba.
- Administrar las pruebas como un proyecto más dentro del desarrollo del proyecto.

Everett y McLeod explican con estos puntos que la principal motivación para probar es reducir los riesgos de un proyecto, para reducir el riesgo de gastos de desarrollo no planificados, o peor aún, el riesgo de fracaso del proyecto por varios motivos. Lo que nos lleva a realizar un análisis exhaustivo de los riesgos de negocio, antes de probar, conociendo con éste, el tamaño del riesgo y la probabilidad de que se produzca.

Por lo tanto, a la hora de probar tenemos que establecer prioridades. Una de las prioridades más importantes que hay que tener en cuenta son los recursos de los que se va a disponer en el proyecto. Al realizar un análisis de los riesgos del negocio queremos conocer cuáles son los principales riesgos para asegurar que se dispone de recursos suficientes para poder llevar a

cabo las pruebas. Estos recursos irán desde el personal, hasta las herramientas que se vayan a utilizar.

Uno de los principios que se suelen aplicar a la hora de realizar las pruebas es el principio de Pareto, "Regla del 80/20". Este principio dice (WIKIPEDIA, 2021):

"El 80% de los fallos de un software es generado por un 20% del código de dicho software, mientras que el otro 80% genera tan solo un 20% de los fallos".

Teniendo en cuenta estas prioridades conseguiremos el objetivo de optimizar las pruebas.

(Sánchez Peño, 2015)

Las pruebas se llevan cabo a partir de Casos de prueba (test-case), un caso de prueba es un conjunto de documentos que definen los requisitos de un componente o sistema. Utilizado como fundamento para el desarrollo de casos de prueba.

Caso de prueba (test case):

La definición de un caso de prueba incluye la siguiente información (según IEEE Std 610):

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

Para el desarrollo de pruebas además de los test-case se debe tener en cuenta los siete principios de generales del proceso de prueba de software.

Estos principios fundamentales de prueba ayudan a los equipos de prueba a utilizar su tiempo y esfuerzo para hacer que el proceso de prueba sea efectivo. En este artículo, aprenderemos en detalle sobre dos principios, es decir, **Agrupación de defectos, principio de Pareto y paradoja de plaguicidas**

Los siete principios generales del proceso de pruebas de software.

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

- El proceso de pruebas puede probar la presencia de defectos.
- Las desviaciones identificadas a lo largo del proceso de pruebas demuestran la presencia de un fallo.
- La causa de un fallo puede no ser obvia.
- El proceso de pruebas no puede demostrar la ausencia de defectos.
- Las pruebas reducen la probabilidad de la presencia de defectos que permanecen sin ser detectados. La ausencia de fallos no demuestra la corrección de un producto software.
- El mismo proceso de pruebas puede contener errores.
- Las condiciones de las pruebas pueden ser inapropiadas para detectar errores.

Principio 2: No es posible realizar pruebas exhaustivas

- Pruebas exhaustivas (exhaustive testing).
- Enfoque del proceso de pruebas en el cual el juego de pruebas (suite de pruebas) abarca todas las combinaciones de valores de entrada y precondiciones.
- Explosión de casos de prueba (test case explosión).
- Define el incremento exponencial de esfuerzo y costo en el caso de pruebas exhaustivas.
- Prueba de muestra (sample test).
- La prueba incluyo solamente aun subconjunto (generado de forma sistemática o aleatoria) de todos los posibles valores de entrada.
- En condiciones normales, se utilizan generalmente pruebas de muestra. Probar todas las combinaciones posibles de entradas y precondiciones sólo es económicamente viable en casos triviales.

Principio 3: Pruebas tempranas (early testing)

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

Principio 4: Agrupamiento de defectos (defect clustering)

- Al encontrar un defecto se encontrarán más defectos "cerca"
- Los defectos aparecen agrupados como hongos o cucarachas.
- Cuando se detecta un defecto es conveniente investigar el mismo módulo en el que ha sido detectado.
- Los probadores (testers) deben ser flexibles
- Habiendo sido detectado un defecto es conveniente volver a considerar el rumbo de las pruebas siguientes.
- La identificación de un defecto puede ser investigada con un mayor grado de detalle, por ejemplo, realizando pruebas adicionales o modificando pruebas existentes.

Principio 5: Paradoja del pesticida

- Repetir pruebas en las mismas condiciones no es efectivo.
- Cada caso de prueba debe contar con una combinación única de parámetros de entrada para un objeto de pruebas particular, de lo contrario no se podrá obtener información adicional.

- Si se ejecutan las mismas pruebas de forma reiterada no se podrán encontrar nuevos defectos.

- Las pruebas deben ser revisadas/modificadas regularmente para los distintos módulos (código).

- Es necesario repetir una prueba tras una modificación del código (corrección de defectos, nueva funcionalidad)

- La automatización de pruebas puede resultar conveniente si un conjunto de casos de prueba se debe ejecutar frecuentemente.

Principie 6: Las pruebas dependen del contexto

- Las pruebas se desarrollan de forma diferente en diferentes contextos.

- Objetos de prueba diferentes son probados de forma diferente.

- El controlador del motor de un coche requiere pruebas diferentes respecto de aquellas para una aplicación de "e-Commerce".

- Entorno de pruebas (cama de pruebas - test bed) vs. entorno de producción.

- Las pruebas tienen lugar en un entorno distinto del entorno de producción. El entorno de pruebas debe ser similar al entorno de producción.

- Siempre habrá diferencias entre el entorno de pruebas y el entorno de producción. Estas diferencias introducen incertidumbre con respecto a las conclusiones que se pudieran obtener tras las pruebas.

Principio 7: La falacia de la ausencia de errores

- Un proceso de pruebas adecuado detectará los fallos más importantes.

- En la mayoría de los casos el proceso de pruebas no encontrará todos los defectos del sistema (ver Principio 2), pero los defectos más importantes deberían ser detectados.

- Esto en sí no prueba la calidad del sistema software.

- La funcionalidad del software puede no satisfacer las necesidades y expectativas de los usuarios.

- No se puede introducir la calidad a través de las pruebas, ella tiene que construirse desde el principio.

Bibliografía

Burnstein, I. (2003). *Practical Software Testing. Primera edición.* Springer.

COMPUTERWORL. (15 de septiembre de 2021). *Fallo sonda Mariner-1*. Obtenido de http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html?page=2

Dijkstra, E. (15 de septiembre de 2021). *Definición de pruebas Dijkstra.* Obtenido de http://es.wikipedia.org/wiki/Edsger_Dijkstra

ESA. (15 de septiembre de 2021). *Informe fallo Ariane 5*. Obtenido de http://www.esa.int/esapub/bulletin/bullet89/dalma89.htm

Everett, G. D., & MacLeod Jr, R. (2007). *Software testing: Testing across the entire software development life cycle. Primera edición.* John Wiley & Sons.

ISO. (15 de septiembre de 2021). *Norma ISO 25010*. Obtenido de http://iso25000.com/index.php/normas-iso-25000/iso-25010

ISO 2500 - WIKIPEDIA. (15 de septiembre de 2021). *Definición de calidad ISO 9000*. Obtenido de http://es.wikipedia.org/wiki/Calidad

ISTQB. (15 de septiembre de 2021). *Manual - ISTQB Foundation.* Obtenido de http://www.sstqb.es/ficheros/sstqb_file95-a69acf.pdf

Kaner, C., D, J., & D, P. (15 de septiembre de 2021). *Software Testing as a Social Science.* Obtenido de http://www.kaner.com/pds/KanerSocialSciencesSTEP.pdf

Myers, G. J. (2004). The art of software testing. Segunda edición. John Wiley & Sons.

Patton, R. (2005). *Software Testing. Segunda Edición.* Sams Publishing.

Pressman, R. S. (2005). Ingeniería del Software: Un enfoque practico. Sexta edición. Mc Graw Hill.

Sánchez Peño, J. M. (2015). Pruebas de Software. Fundamentos y Técnicas. En J. M. Sánchez Peño, *Pruebas de Software. Fundamentos y Técnicas.* (págs. 11-24). MADRID: UNIVERSIDAD POLITÉCNICA DE MADRID.

TEGSOLTIONS. (15 de septiembre de 2021). *Definición de calidad*. Obtenido de http://www.tegsolutions.com/Que%20es%20la%20Calidad.htm

WIKIPEDIA. (15 de septiembre de 2021). *Definición de calidad*. Obtenido de http://es.wikipedia.org/wiki/Genichi_Taguchi

WIKIPEDIA. (15 de septiembre de 2021). *Principio de Pareto (Regla 80/20)*. Obtenido de http://es.wikipedia.org/wiki/Principio_de_Pareto

WIKIPEDIA. (15 de septiembre de 2021). *Therac 25*. Obtenido de http://es.wikipedia.org/wiki/Therac-25