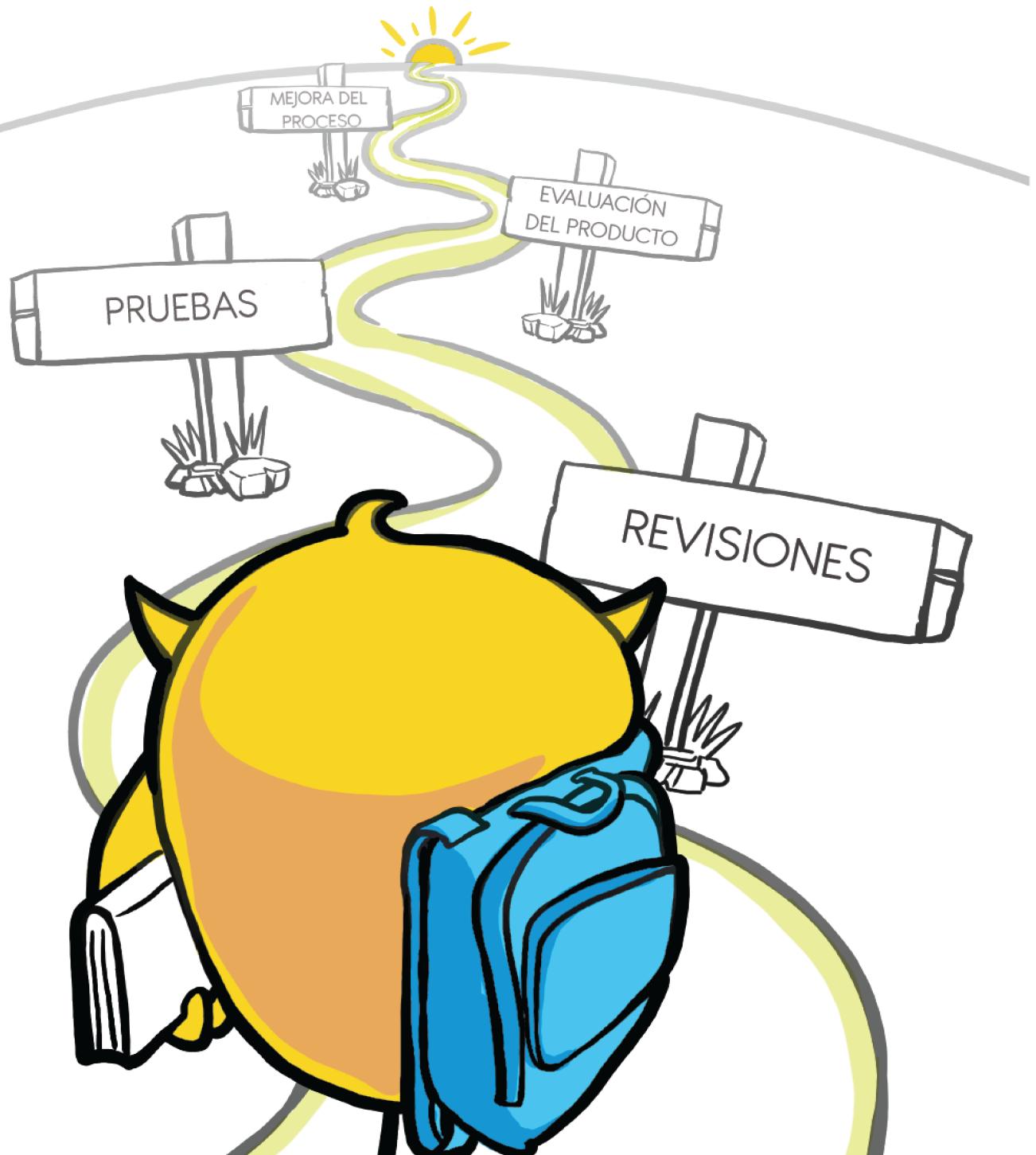


Primera Edición

Calidad en Software

UN LIBRO DE TEXTO



Sandra Sánchez Gordón, Ph.D.

Derechos Reservados © 2024 Todos los derechos reservados.

Esta publicación no puede ser reproducida o transmitida por ningún medio electrónico o mecánico, incluyendo copias, escaneos, fotografías, o por ningún sistema de almacenamiento digital.

Diseño gráfico por Gabriela Zaldumbide.

Gestión de sitio web por Daniela Zaldumbide.

Primera Edición

Publicado por Escuela Politécnica Nacional

Impreso en Quito-Ecuador

0.1 A los estudiantes

¡Bienvenidos al estudio de un tema importante y fascinante: la calidad y las pruebas de software!

Los objetivos de aprendizaje de este texto son:

- Comprender la evolución de la calidad en general y la calidad de software a lo largo de la historia.
- Utilizar normas y modelos de calidad de los productos de software.
- Utilizar normas y modelos de madurez de los procesos de software.
- Utilizar métricas de producto y proceso de software.
- Entender el proceso, los niveles, y los tipos de pruebas de software.
- Utilizar técnicas de diseño de pruebas de software.
- Comprender los enfoques de pruebas dirigidas por el contexto, ágiles, y DevOps.

0.2 A los instructores

Este texto está diseñado para ser utilizado en cursos de pregrado o nivel introductorio de posgrados en el área de Informática o afines.

Los temas generales que abarca este texto son:

- Fundamentos de calidad.
- Calidad del producto y proceso de software.
- Pruebas de software.
- Pruebas de contexto, ágiles y devops.

0.3 Recursos complementarios

Se puede acceder a los recursos complementarios de este libro de texto en el sitio web acompañante <https://calidadensoftware.com>.

Los estudiantes pueden encontrar:

- Diapositivas de cada capítulo.
- Soluciones a las preguntas, ejercicios y problemas de fin de capítulo.

Los instructores pueden encontrar:

- Diapositivas editables de cada capítulo.
- Librería de figuras del libro.
- Sílabo genérico de curso de calidad y pruebas de software.
- Notas del profesor.
- Tareas para proponer a los estudiantes.
- Banco de preguntas para exámenes.
- Enlaces a recursos de terceros.

0.4 Compromiso

Es mi interés interactuar con los estudiantes e instructores que utilicen este texto para recibir retroalimentación y sugerencias de mejora. Trataré de responder individualmente a quienes me escriban con sus comentarios tan pronto como sea posible. La dirección de correo es: info@calidadensoftware.com.

0.5 Agradecimientos

Ha sido mi aspiración escribir este texto de estudio por mucho tiempo. Su origen son las interacciones con mis estudiantes, las notas de clases, materiales recopilados, y recursos educativos generados para el curso de calidad de software que he enseñado por más de una década.

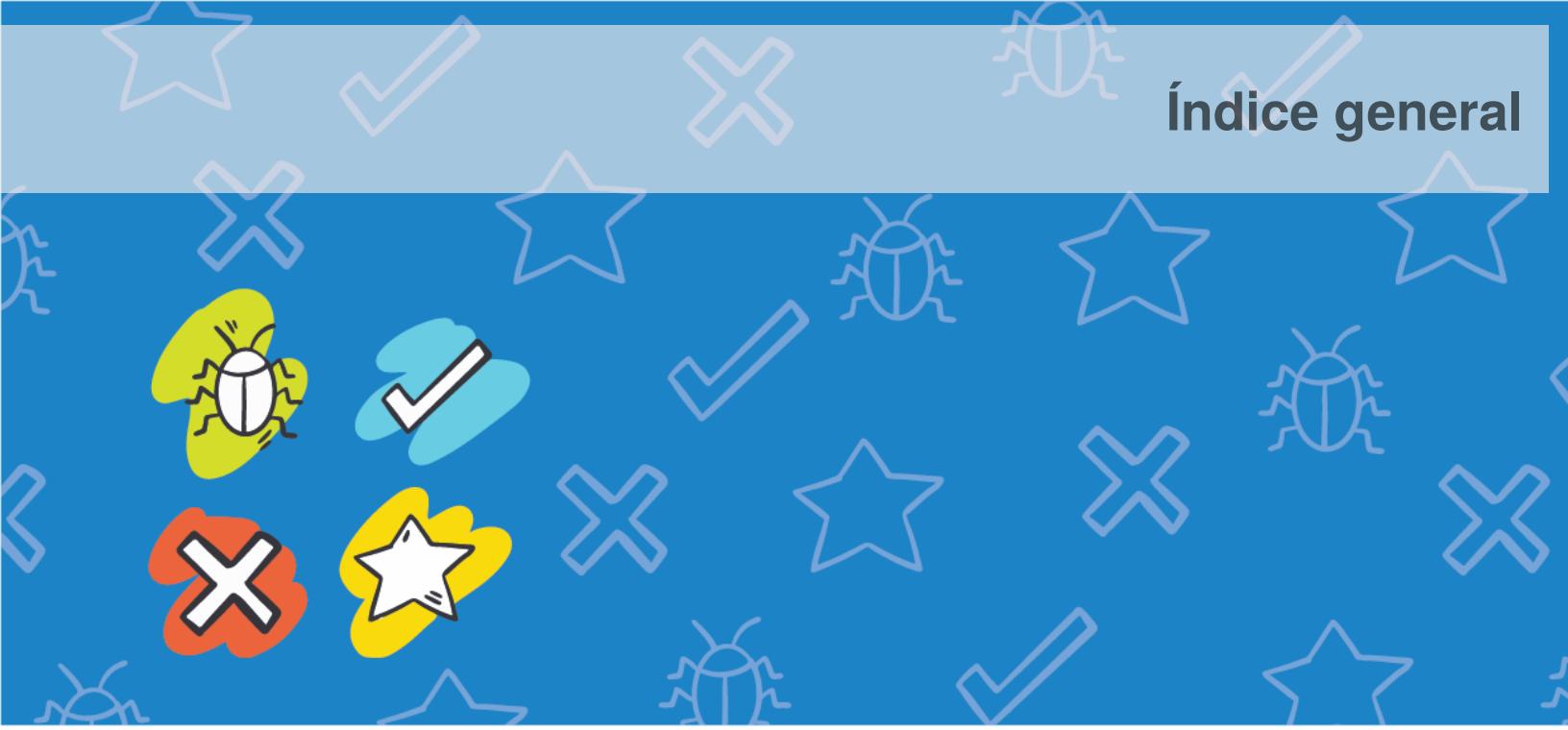
Agradezco a la Escuela Politécnica Nacional por haberme patrocinado para hacer realidad este libro.

Agradezco a los colegas y profesionales informáticos que han participado como pares revisores por sus valiosos comentarios y sugerencias de mejora.

Quiero agradecer sobretodo a mi familia y amigos por su apoyo incondicional.

Sandra Sanchez-Gordon, Ph.D.
Quito, Ecuador
2024

Esta página está intencionalmente vacía.



0.1	A los estudiantes	3
0.2	A los instructores	3
0.3	Recursos complementarios	4
0.4	Compromiso	4
0.5	Agradecimientos	5
1	Fundamentos de calidad de software	11
1.1	Historia de la calidad	11
1.1.1	Edad antigua	11
1.1.2	Edad media	13
1.1.3	Edad moderna	14
1.1.4	Edad contemporánea	15
1.1.5	Perspectivas de la calidad	20
1.2	Evolución de la calidad de software	21
1.2.1	Período 1840-1950	21
1.2.2	Período 1951-1970	22
1.2.3	Período 1971-1980	25
1.2.4	Período 1981-1990	28
1.2.5	Período 1991-2000	31
1.2.6	Período 2001-2024	33
1.3	Normalización	35
1.3.1	Importancia	35

1.3.2	Normas relativas a calidad de software	37
1.4	Definiciones	43
1.4.1	Definiciones relativas a software	43
1.4.2	Definiciones relativas a calidad de software	44
1.4.3	Definiciones de error, defecto, bug, y fallo	44
1.4.4	Ejemplos de fallos	47
1.5	Ética y calidad de software	53
1.5.1	Código de ética y de conducta profesional de la informática	54
1.5.2	Toma de decisiones éticas	56
1.6	Autoevaluación, retos, y aprendizaje autónomo	61
2	Calidad del producto y del proceso de software	67
2.1	Calidad del producto de software	67
2.1.1	Modelos iniciales de calidad del producto	67
2.1.2	Modelo de calidad del producto	71
2.1.3	Métricas de calidad del producto	79
2.1.4	Modelo de calidad en uso	90
2.1.5	Métricas de calidad en uso	92
2.2	Especificación de requisitos de calidad	97
2.2.1	Historias de usuario y criterios de aceptación	98
2.2.2	Proceso de especificación de requisitos	101
2.3	Evaluación de calidad del producto de software	103
2.3.1	Proceso de evaluación	103
2.3.2	Informe de evaluación	106
2.4	Calidad del proceso de software	106
2.4.1	Mejora de procesos de software	106
2.4.2	Ciclo PDCA	111
2.4.3	Sistema de gestión de calidad aplicado a software ISO 90003	112
2.4.4	Mejora de procesos ISO 33014	114
2.4.5	Modelo CMMI	116
2.4.6	Diretrices para la evaluación de procesos ISO 29110-3-1	126
2.4.7	Métricas de proceso de software	133
2.5	Autoevaluación, retos, y aprendizaje autónomo	142

3	Pruebas de software	151
3.1	Proceso de pruebas	151
3.1.1	Pruebas organizacionales	152
3.1.2	Planificación	158
3.1.3	Seguimiento y control	160
3.1.4	Diseño e implementación	163
3.1.5	Configuración y mantenimiento del entorno de pruebas	169
3.1.6	Ejecución	170
3.1.7	Notificación de incidentes	172
3.1.8	Finalización	173
3.1.9	Mejoramiento del proceso de pruebas	174
3.2	Niveles	177
3.2.1	Pruebas de componentes	178
3.2.2	Pruebas de integración	178
3.2.3	Pruebas de sistema	180
3.2.4	Pruebas de aceptación	180
3.3	Tipos	184
3.3.1	Pruebas funcionales	184
3.3.2	Pruebas no funcionales	185
3.4	Revisiones	190
3.4.1	Tipos de revisiones	191
3.4.2	Proceso genérico de revisión	192
3.4.3	Roles en las revisiones	194
3.4.4	Técnicas de revisión	194
3.5	Técnicas de diseño	199
3.5.1	Técnicas de diseño de pruebas basadas en la especificación	199
3.5.2	Técnicas de diseño de pruebas basadas en la estructura	209
3.5.3	Técnicas de pruebas basadas en la experiencia	215
3.6	Autoevaluación, retos, y aprendizaje autónomo	219
4	Enfoques en pruebas de software	231
4.1	Enfoques tradicionales	232
4.1.1	Analítico	232
4.1.2	Dirigido por normas	232
4.1.3	Orientado hacia la calidad	233

4.2	Pruebas dirigidas por el contexto	233
4.2.1	Principios básicos de las pruebas dirigidas por el contexto	233
4.2.2	Metodología de pruebas rápidas de software	234
4.3	Pruebas ágiles	244
4.3.1	Desarrollo ágil	244
4.3.2	Principios y prácticas de las pruebas ágiles	246
4.3.3	Habilidades de los probadores ágiles	246
4.3.4	Métodos de pruebas ágiles	247
4.4	Pruebas en DevOps	250
4.4.1	Metodología DevOps	250
4.4.2	Enfoque DevTestOps	253
4.4.3	Rol de los probadores en DevOps	257
4.5	Automatización de pruebas	258
4.5.1	Análisis estático	259
4.5.2	Pruebas unitarias	263
4.5.3	Pruebas de interfaz de usuario web	265
4.5.4	Pruebas de aplicaciones móviles	267
4.5.5	Pruebas de servicios web y API	268
4.5.6	Pruebas de CI/CD	269
4.5.7	Pruebas no funcionales	270
4.5.8	Generación de datos de pruebas	276
4.5.9	Gestión de pruebas	276
4.6	Autoevaluación, retos, y aprendizaje autónomo	278
	Bibliografía	285

1. Fundamentos de calidad de software



1.1	Historia de la calidad	11
1.2	Evolución de la calidad de software	21
1.3	Normalización	35
1.4	Definiciones	43
1.5	Ética y calidad de software	53
1.6	Autoevaluación, retos, y aprendizaje autónomo	61

La calidad no es un acto, es un hábito.

- Aristóteles

1.1 Historia de la calidad

La búsqueda de calidad no es una aspiración exclusiva de la modernidad. Los primeros indicios de actividades de control de calidad se remontan a las civilizaciones antiguas, pasando por los gremios artesanales en la edad media y las inspecciones de la era industrial, hasta llegar a la serie de normas ISO 9000 para la gestión de la calidad en las organizaciones contemporáneas. A continuación se presenta un recorrido de la evolución de la calidad a lo largo de las etapas históricas en el desarrollo de la humanidad.

1.1.1 Edad antigua

La primera mención escrita a la calidad data del año 1772 a. C. en el código de Hammurabi, que es un conjunto de 282 leyes talladas en piedra. En la Figura 1.1 se observa un fragmento de este código legal que fue elaborado por el Rey de Babilonia de nombre Hammurabi. El código de Hammurabi instaura el concepto de responsabilidad e instruye, entre otras cosas, cómo se debe proceder con el responsable de una edificación si ésta resulta de mala calidad:

“Si un constructor construye una casa para alguien, y no lo hace adecuadamente, y la casa que construyó se derrumba y mata a su propietario, entonces ese constructor será condenado a muerte. Si mata al hijo del propietario, el hijo del



Figura 1.1: Fragmento del código de Hammurabi.
Fuente: Museo de Louvre [198].

constructor será condenado a muerte. Si mata a un esclavo del dueño, éste deberá pagar, esclavo por esclavo (un esclavo equivalente) al dueño de la casa. Si arruina los bienes, deberá indemnizar por todo lo que se haya arruinado, y debido a que no ha hecho sólida esta casa que construyó y se derrumbó, deberá volver a levantar la casa con sus propios medios. Si un constructor construye una casa para alguien, aunque aún no la haya terminado; si las paredes parecen derrumbarse, el constructor deberá hacer sólidas las paredes con sus propios medios."

- Código de Hammurabi [21].

Los textos y figuras en la capilla funeraria del visir Rekhmire en la Necrópolis de Tebas, que datan del año 1450 a. C., prueban que en la civilización Egipcia se realizaban actividades de capacitación a los operarios e inspecciones de calidad. Las obligaciones de Rekhmire como visir incluían la supervisión de obras de construcción y el trabajo de los talleres de producción artesanal. En la Figura 1.2 se observa una imagen de Rekhmire de pie, sosteniendo su cetro y mirando hacia la izquierda para vigilar a un grupo de artesanos trabajando. Sobre la figura de Rekhmire se lee el siguiente texto:

"Inspeccionando a todos los artesanos del templo de Amón [...] y dando a cada hombre instrucciones para su tarea de hacer todo tipo de productos."
- Capilla Funeraria de Rekhmire [7].

Similarmente, en la civilización Fenicia, que floreció entre 1500 a. C. y 330 a. C, los inspectores cortaban las manos de los constructores cuando éstos no cumplían con las es-



Figura 1.2: Detalle de la tumba de Rekhmire.

Fuente: Arte e historia de Egipto [7].

pecificaciones. Así mismo, en la civilización de la Antigua Grecia, entre 776 a. C. y 146 a. C., se inspeccionaban minuciosamente los materiales de construcción, y se diseñaban y elaboraban moldes especiales para garantizar la exactitud de los objetos producidos.

1.1.2 Edad media

En la Europa medieval, desde finales del Siglo 13 hasta inicios del Siglo 17, aparecen los gremios de artesanos [161]. En esta época, el trabajo artesanal se realizaba en pequeños talleres cuyo propietario era un maestro artesano que cumplía los roles simultáneos de instructor del oficio e inspector de calidad. La capacitación obligatoria de los aprendices se formalizaba con un contrato de aprendizaje firmado entre el maestro que se comprometía a enseñar y el joven que deseaba aprender.

Los gremios se encargaban de organizar y regular el suministro de materias primas, los procesos de producción, la capacitación, y las características deseadas en los productos. De esta manera, los gremios establecieron los primeros estándares de calidad. Sin embargo, los controles de calidad eran inadecuados y las penalizaciones por incumplir los estándares eran insuficientes debido a que los gremios priorizaban la rentabilidad. Por ello, los comerciantes añadieron sus propios puntos de inspección de calidad al momento de comprar los productos. La Figura 1.3 muestra un grabado de un taller de zapatería de la época.

En esa época, cada maestro artesano era responsable del desarrollo completo de su producto, desde la concepción hasta la entrega al cliente. Esto daba lugar a que los artesanos tuvieran un fuerte sentimiento de orgullo por la calidad de sus productos, y los aprendices se unieran a los talleres de los artesanos para aprender las técnicas del oficio y convertirse a su vez en artesanos de éxito [186].



Figura 1.3: Taller de zapatería.
Fuente: El libro de los oficios [4].

1.1.3 Edad moderna

A finales del Siglo 17 se produjo una separación entre las ciudades y la ruralidad debido al desarrollo del comercio, lo que llevó a que paulatinamente los artesanos migren y se concentren en las ciudades. Es así como, durante la época de la Primera Revolución Industrial, desde mediados del Siglo 18 hasta mediados del Siglo 19, los talleres artesanales desaparecieron paulatinamente y dieron paso al sistema de fábricas de producción en serie, a la especialización del trabajo, y al incremento de la producción; a fin de satisfacer los altos niveles de demanda de las ciudades. Sin embargo, en las primeras fábricas se carecía de control de calidad de los productos y se produjo además un deterioro de la calidad de la mano de obra. Entonces, se recurrió al esquema de prueba y error, mediante inspecciones que se realizaban para validar si el producto cumplía o no con lo solicitado por el cliente. Los productos finales defectuosos se desechaban o se volvían a trabajar, con el consiguiente desperdicio y costo.

La revolución industrial supuso un cambio en este paradigma, y el rol de la mano de obra se organizó en gran medida, con trabajadores responsables para cada parte concreta del proceso de fabricación. El sentido de propiedad y el orgullo de la mano de obra en el producto se diluyeron, ya que los trabajadores sólo eran responsables de una parte del producto, y no del producto en su conjunto. Esto condujo a la necesidad de prácticas de gestión más estrictas, incluyendo la planificación, organización, ejecución, y control del trabajo en las fábricas. Para ello, se estableció jerarquías de trabajo con roles y responsabilidades, y una estructura de informes de cada función. Los controles por parte de los supervisores se volvieron necesarios para garantizar la productividad y la calidad.

1.1.4 Edad contemporánea

A partir del Siglo 20 se da un aceleramiento en la evolución de la gestión de calidad en las organizaciones gracias al aporte de varios pioneros, algunos de los cuales se presentan a continuación. Aunque sostienen diferentes puntos de vista en su concepción de la calidad, coinciden en dos aspectos muy importantes: el producto debe satisfacer los requisitos de los clientes y la elaboración del producto se debe realizar a través de un proceso.

Walter Shewhart

En la década de 1930, el doctor en física estadounidense Walter Shewhart trabajaba en los Laboratorios Bell, parte de la compañía Western Electric, posteriormente AT&T. Shewhart es considerado el creador del control estadístico de procesos debido a que desarrolló un gráfico de control que es utilizado hasta la actualidad. El gráfico de control de Shewhart es una herramienta para controlar procesos que incluye límites superiores e inferiores de rendimiento. Se considera que un proceso está bajo control estadístico si opera dentro de esos límites y por tanto no es necesario desarrollar acciones correctivas.

Ejemplo 1.1 Proceso contable de mayorización.

Todo software contable debe incluir la automatización del proceso de mayorización de cuentas. Normalmente, este proceso se ejecuta al cierre de cada mes pero puede también ser ejecutado en cualquier momento, a fin de actualizar los saldos contables. La Figura 1.4 muestra el gráfico de control para el proceso de mayorización de un software contable. En el eje horizontal se tiene la escala de las muestras tomadas, en este caso 16 muestras, y en el eje vertical se grafican los tiempos de respuesta. Los puntos de la gráfica muestran las mediciones concretas obtenidas para cada muestra. Los puntos rojos indican instancias de la ejecución del proceso de mayorización fuera de control (es decir fuera de los límites inferior y superior aceptables) que deben ser investigadas para mejorar el tiempo de respuesta del proceso. El rol de garantía de calidad de software (SQA) - bajo el acrónimo en inglés, Software Quality Assurance - es el encargado de analizar los gráficos de control de procesos.

En 1931, Shewhart publica el libro “El control económico de los productos manufacturados” [181] donde describe métodos de control estadístico de procesos para reducir su variabilidad. Shewhart predijo que la productividad mejorará a medida que la variabilidad se reduzca, lo que fue comprobado en la década de 1950 por ingenieros japoneses. Al terminar la Segunda Guerra Mundial, las compañías japonesas dieron un salto de paradigma respecto a la calidad, logrando mejorar su productividad y liderar mercados a nivel mundial. Este fue el origen del movimiento de la calidad total.

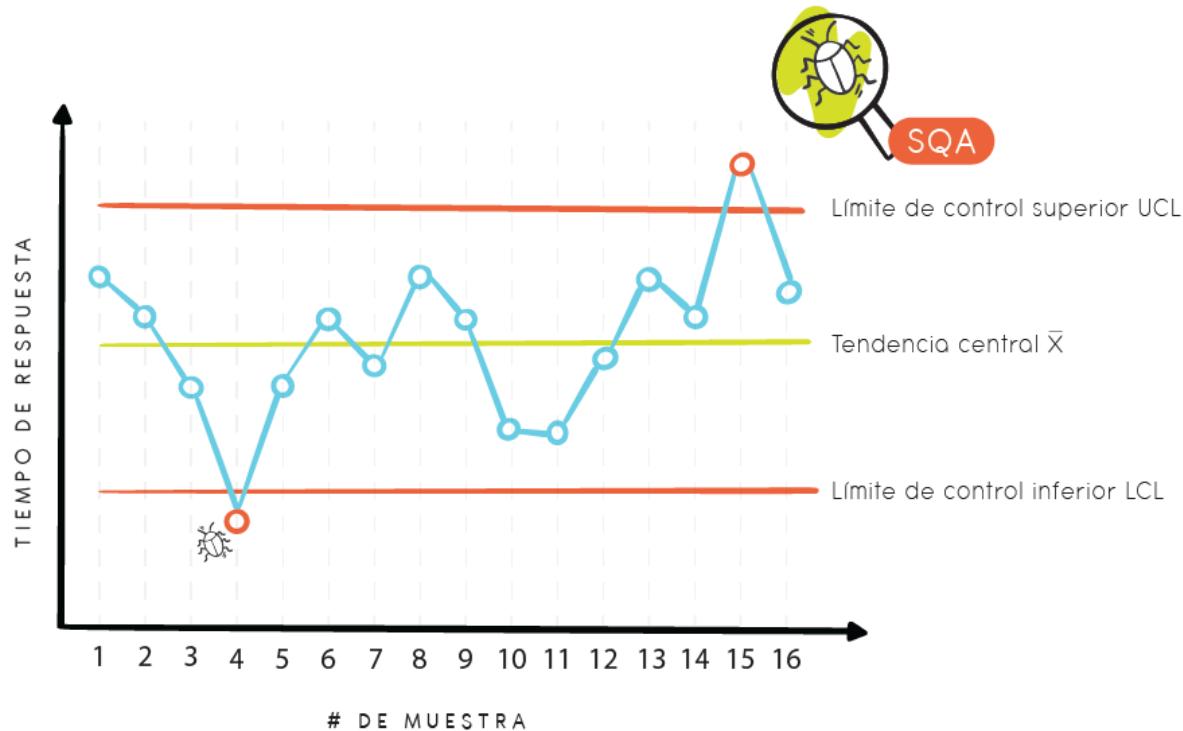


Figura 1.4: Ejemplo de gráfico de control de proceso de Shewhart.
Elaboración: Autora.



Recomendación: Investigue el concepto de gestión de calidad total (TQM), bajo el acrónimo en inglés, Total Quality Management.

En 1939, en su libro “Método estadístico desde la perspectiva del control de calidad” [182], Shewhart publica un ciclo de tres pasos al que describe como un proceso científico dinámico para adquisición de conocimientos. Parte de los tres pasos clásicos de especificación, producción, e inspección; pero señala que estos pasos no son efectivos aunque se logre mantener la producción dentro de los límites de tolerancia. Explica que por razones económicas y para garantizar la calidad, estos tres pasos no deben ser lineales sino que deben ser circulares, como se muestra en la Figura 1.5. Estos tres pasos además se corresponden con el método científico de formular hipótesis, ejecutar experimento, y probar la hipótesis. Por ello, explica que sería aún mejor representarlos como una espiral. Desde esta perspectiva, la producción se constituye en un método continuo y auto-corregido para hacer el uso más eficiente posible de la materia prima. Más tarde, Edwards Deming, quien trabajó con Shewhart en los Laboratorios Bell, se basaría en este método para popularizarlo con el nombre de ciclo para la mejora continua Planear Hacer Verificar Actuar (PDCA), bajo el acrónimo en inglés, Plan Do Check Act .



Figura 1.5: Ciclo Shewhart.

Fuente: Método estadístico desde la perspectiva del control de calidad [182].

Kaoru Ishikawa

En 1943, el químico japonés Kaoru Ishikawa desarrolla el diagrama Causa-Efecto como una herramienta sistemática para identificar, seleccionar, y documentar las causas de los problemas de calidad en procesos, productos, y servicios. Por este aporte, se le considera el padre del análisis de las causas de problemas en procesos industriales. En su libro “¿Qué es el control de calidad total? El estilo japonés” [82], publicado en 1981, Ishikawa afirma que la calidad es un concepto dinámico, ya que las necesidades, los requisitos, y las expectativas del cliente cambian continuamente.

W. Edwards Deming

Terminada la Segunda Guerra Mundial en 1945, el doctor en física estadounidense William Edwards Deming viaja a Japón para ayudar en la reconstrucción de la industria de ese país mediante la difusión del control estadístico de procesos y el ciclo PDCA de Shewhart. Por este motivo, Deming es considerado por los japoneses como el padre de la tercera revolución industrial. La filosofía post-guerra de Japón se disemina posteriormente a Estados Unidos y el resto del mundo. En 1982, en su libro “Fuera de la crisis” [38], Deming propone que la calidad se defina en términos de satisfacción del cliente. En 1950, propone catorce puntos y siete enfermedades mortales de la gerencia en donde afirma que todo proceso es variable y cuanto menor sea la variabilidad del mismo, mayor será la calidad del producto.

resultante.

Definición 1.1 Calidad según Deming [38]. “Un producto o servicio posee calidad si ayuda a alguien y disfruta de un mercado sostenible.”

Joseph M. Juran

En 1951, el ingeniero eléctrico rumano Joseph M. Juran, considerado el padre de la gestión de calidad, en su libro “Manual de control de calidad” [140] define tres procesos para la gestión de la calidad conocidos, como la trilogía de Juran: planificación de la calidad, control de la calidad, y mejora de la calidad. Juran trabajó con Shewhart y Deming en los Laboratorios Bell y fue consultor de calidad de Philips, Xerox, Toyota, entre otras empresas, hasta la década de 1990. Juran articuló el Principio de Pareto, también conocido como Regla 80/20, al plantear que el 80% de los defectos de calidad son producto del 20% de las causas. La utilidad práctica de este principio es priorizar los esfuerzos de mejora. En su libro “Juran sobre la planificación de la calidad” [139] plantea una frase corta para definir la calidad: “calidad es adecuación para el uso”. Esta definición implica la medición del grado en que el producto sirve satisfactoriamente a los fines del usuario durante su utilización.

Definición 1.2 Calidad según Juran [139]. “Calidad es adecuación para el uso.”

Philip Crosby

En 1957, el empresario estadounidense Philip Crosby desarrolla la teoría de cero defectos según la cual hay que eliminar todo lo que no aporte valor a un proyecto, lo que conduce a la eliminación de residuos, a la mejora del proceso y, en consecuencia, a la reducción de los costos. Crosby sintetiza su filosofía de cero defectos como un enfoque orientado a la prevención, cuyo principal camino es “hacerlo bien la primera vez” en lugar de pagar luego por reparaciones y retrabajo. En 1979, publica el libro “La calidad es gratuita” [37] donde propone el establecimiento de buenos principios de gestión de la calidad en las organizaciones.

Definición 1.3 Calidad según Crosby [37]. “La calidad es la conformidad con un conjunto de especificaciones. Estas especificaciones se establecen en función de las necesidades y deseos del cliente.”

Armand V. Feigenbaum

En 1961, el doctor en economía estadounidense Armand V. Feigenbaum acuñó el concepto de Control de Calidad Total (TQC) - bajo el acrónimo en inglés, Total Quality Control

- indicando que la calidad va más allá del control de defectos en planta de producción hacia un compromiso organizacional con la ética y la excelencia. Adicionalmente, Feigenbaum complementa el concepto de costo de la calidad de Shewhart añadiendo los gastos directos e indirectos causados por la insatisfacción del cliente en el proceso de compra. En su libro “Control de la calidad total” [50] define la calidad como una determinación del cliente, no una determinación de los ingenieros, ni del área de marketing, ni de la gerencia general. Acorde a Feigenbaum, la calidad depende de la perspectiva del cliente y es responsabilidad de toda la empresa.

Definición 1.4 Calidad según Feigenbaum [50]. “La calidad se basa en la experiencia real del cliente con el producto o servicio medida en función de sus requisitos - declarados o no, conscientes o percibidos, técnicamente operacionales o totalmente subjetivos, y siempre representa un objetivo móvil en un mercado competitivo.”

Genichi Taguchi

En 1986, el ingeniero y estadístico japonés Genichi Taguchi propone una filosofía de calidad basada en tres elementos: la función de pérdida, utilizada para medir la pérdida financiera de la sociedad resultante de la mala calidad; la filosofía de control de calidad fuera de línea; y el diseño de productos y procesos basado en parámetros de diseño que determinan el buen funcionamiento del equipo de trabajo.

Definición 1.5 Calidad según Taguchi [188]. “ La calidad de un producto está dada por la mínima pérdida que dicho producto ocasiona a la sociedad, consumidores, y productores, desde que sale de la fábrica.”

Serie de normas ISO 9000 - Sistemas de gestión de calidad

La Organización Internacional de Normalización (ISO) - bajo el acrónimo en inglés, International Organization for Standardization - publica en 1987 la primera edición de la serie de normas ISO 9000. La serie de normas ISO 9000 proporciona conceptos y principios fundamentales para el desarrollo de Sistemas de Gestión de Calidad (SGC) en toda entidad dedicada a la fabricación de bienes o prestación de servicios. La norma ISO 9000:2015 “Sistemas de gestión de calidad - Fundamentos y vocabulario” [84], revisión vigente a la fecha de publicación de la primera edición de este texto, contiene los conceptos fundamentales, principios, términos, y definiciones aplicables a la gestión de la calidad.

Definición 1.6 Calidad según norma ISO 9000:2015 [84]. “El grado en el que un conjunto de características inherentes de un objeto cumple con los requisitos. Un requisito es una necesidad o expectativa establecida, generalmente implícita u obligatoria.”

La norma ISO 9001:2015 “Sistemas de gestión de calidad - Requisitos” [85] , revisión vigente a la fecha de publicación de la primera edición de este texto, es la única norma de la serie ISO 9000 que otorga certificación a las organizaciones que la implementan. La norma ISO 9001:2015 es un conjunto de recomendaciones basadas en el ciclo PDCA, así como en la gestión de riesgos.

Ejemplo 1.2 Sistema ecuatoriano de calidad.

La calidad en Ecuador está regulada mediante el Sistema Ecuatoriano de la Calidad, cuya ley expresa, entre otras cosas:

“Art. 3. Declárase política de Estado la demostración y la promoción de la calidad, en los ámbitos público y privado, como un factor fundamental y prioritario de la productividad, competitividad, y del desarrollo nacional.

Art. 50. El Estado ecuatoriano propiciará el desarrollo y la promoción de la calidad, de la productividad, y la mejora continua en todas las organizaciones públicas y privadas, creando una conciencia y cultura de los principios y valores de la calidad a través de la educación y la capacitación.”

- Ley del Sistema Ecuatoriano de la Calidad [35].

Un estudio realizado en Ecuador en 2016 señala que a esa fecha existían 1,348 empresas certificadas con la norma ISO 9001 en el país. Para este estudio se tomó una muestra general de 163 empresas, de las cuales se determinó que el 26.38% tenían certificación ISO 9001. Posteriormente, se valoraron nueve factores de calidad y se encontró que las empresas certificadas con la norma ISO 9001 obtuvieron mayor valoración (promedio de 4.0) que las que no certificadas (promedio de 3.4) en todos los factores analizados [22]. Este resultado muestra los efectos positivos de la implementación del SGC de la norma ISO 9001 en las organizaciones.

1.1.5 Perspectivas de la calidad

De lo expuesto previamente, se puede concluir que no existe un consenso universal sobre la calidad. Esto se debe a que es un concepto relativo a quien lo expresa y de carácter dinámico según la época [28]. Es decir, las conceptualizaciones acerca de la calidad han ido evolucionando en la medida que han cambiado los contextos sociales y económicos de la sociedad. Sin embargo, la diversidad de definiciones de calidad se pueden clasificar desde dos perspectivas amplias [149]:

Por una parte, la calidad puede ser concebida en términos de superioridad de un producto

sobre otros, donde está implícita una evaluación positiva del producto o servicio. A esta categoría corresponden las definiciones propuestas por Jurán, Deming, y Feigenbaum.

Por otra parte, la calidad puede ser concebida como cualidad de un objeto, ésto es, un producto o servicio tendrá siempre, de manera implícita o incorporada, una determinada calidad que será de mayor o menor grado en la medida en que sus atributos estén en conformidad o se adecúen a un patrón o norma de referencia, sea según la perspectiva de los productores o según la perspectiva de los consumidores. A esta segunda categoría corresponden las definiciones propuestas por Crosby y la serie de normas ISO 9000.

1.2 Evolución de la calidad de software

La historia de la calidad de software va de la mano de la historia de la ingeniería de software, que a su vez va de la mano de la historia del desarrollo del hardware. La calidad de software es un campo muy amplio que abarca el control de calidad y la garantía de calidad. El control de calidad de software se enfoca en detectar defectos presentes en el producto mediante revisiones y pruebas. La garantía de calidad de software se enfoca en prevenir la inserción de defectos en el producto mediante la mejora continua de los procesos de desarrollo, evolución, y pruebas del producto. A continuación se presentan aportes de algunos de los actores en cada etapa de la evolución de la calidad de software. Estas etapas se han dado como ciclos evolutivos siguiendo la lógica de tesis, antítesis, y síntesis del método dialéctico propuesto por el filósofo alemán Friedrich Hegel [172].

Recomendación: Lea el artículo “Aristóteles, dialéctica hegeliana y evolución de la ingeniería de software”[172], autora Sandra Sánchez Gordón, 2012.



1.2.1 Período 1840-1950

Ada Lovelace

En la década de 1840, la matemática y escritora inglesa Augusta Ada Byron, condesa de Lovelace, tradujo del francés al inglés el artículo “Boceto del motor analítico inventado por Charles Babbage” [154]. En las notas de esta traducción, que ocupaban más páginas que el artículo original escrito por Luigi Menabrea, incluyó como anexo un algoritmo para utilizar la máquina analítica para el cálculo de números de Bernoulli. La máquina analítica diseñada por Babbage era un dispositivo mecánico que podía programarse mediante tarjetas perforadas. El algoritmo publicado por Lovelace es considerado el primer programa de computación. Adicionalmente, Lovelace fue la primera persona en envisionar el potencial futuro de las computadoras y el software, como se expresa en esta frase de su autoría (las letras itálicas acentuadas son de Lovelace):

“La máquina analítica no tiene pretensiones de crear **nada**. Puede hacer cualquier cosa que **sepamos como ordenarle** que realice.”

- Ada Lovelace [154].

Así mismo, Lovelace se dio cuenta de que un funcionamiento equivocado puede deberse no necesariamente al hardware sino también a defectos de programación, como explica en este párrafo extraído del mismo artículo:

“A esto puede responderse que también debe realizarse un proceso de análisis para proporcionar al motor analítico los datos operativos necesarios, y que en esto puede residir también una posible fuente de error. Asumiendo que el mecanismo real es infalible en sus procesos, las tarjetas pueden darle órdenes erróneas.”

- Ada Lovelace [154].



Recomendación: Mire la película “Conceiving Ada”, directora Lynn Hershman-Leeson, 1997.

Alan Turing

En 1949, en el artículo “Sobre la comprobación de una rutina grande”[191], el matemático, filósofo, y biólogo inglés Alan Turing, considerado uno de los padres de la ciencia de la computación y la informática, plantea la siguiente pregunta: ¿Cómo se puede comprobar una rutina en el sentido de asegurarse que es correcta? Para responder, Turing propone un método general de prueba que todavía es la base de la verificación de programas. Turing también señala que la persona que prueba debe ser distinta a la persona que programa. A continuación un párrafo extraído de este artículo:

“Para que la persona que prueba no tenga una tarea muy dificultosa, el programador debe hacer una serie de aserciones definidas que puedan ser comprobadas individualmente, y de las que se desprende fácilmente la corrección de todo el programa.”

- Alan Turing [191].



Recomendación: Mire la película “Código Enigma”, director Morten Tyldum, 2014.

1.2.2 Período 1951-1970

Daniel D. McCracken

En 1957, el científico computacional estadounidense Daniel D. McCracken publica el libro “Programación de computadores digitales” [153] considerado el primer texto sobre progra-

mación, en el cuál se hace la siguiente referencia a las pruebas:

“En estos casos es muy conveniente que el cliente prepare el caso de comprobación, sobretodo porque los errores lógicos y los malentendidos entre el programador y el cliente pueden ser señalados por este procedimiento. Si el cliente debe preparar la solución de la prueba, es mejor que lo haga con anticipación a la comprobación real, ya que para cualquier problema no trivial se tardará varios días o semanas en calcular la prueba.”

- Daniel McCracken [153].

Charles L. Baker

En 1957, el físico e ingeniero aeroespacial estadounidense Charles L. Baker publicó en la revista “Tablas matemáticas y otros medios de cálculo” una reseña sobre el libro “Programación de computadores digitales” de Daniel McCracken donde explica la diferencia entre probar programas y depurarlos. Al inicio de la década de los 50, no se distinguía entre pruebas y depuración. La atención se centraba en arreglar los defectos. Los desarrolladores solían escribir el código y, cuando se encontraban con un defecto, analizaban y depuraban los problemas. Aún no existía el concepto de pruebas ni de probadores.

Recomendación: Investigue la definición y pasos del proceso de depuración de defectos de software.



Gerald M. Weinberg

En 1958, en el marco del proyecto Mercurio de la NASA, se aplicó mini-incrementos con ventanas de tiempo y una técnica que consistía en planificar y escribir las pruebas antes de cada mini-incremento de desarrollo de software. El doctor en ciencias de la comunicación estadounidense Gerald M. Weinberg trabajó en el proyecto Mercurio. En 1961, en base a la experiencia en dicho proyecto, Weinberg y el ingeniero estadounidense Herbert D. Leeds publican el libro “Fundamentos de la programación informática” [146] que se convierte en el primer libro en tener un capítulo dedicado completamente a pruebas de software, donde plantean los siguientes principios de las pruebas:

1. Escribir el programa correctamente en primer lugar.
2. Pensar en la comprobación al codificar.
3. Conocer las herramientas de depuración disponibles.
4. Hacer que el programa demuestre que funciona.

Leeds y Weinberg sostienen que las pruebas deben demostrar la adaptabilidad del software en lugar de únicamente validar su capacidad para procesar información, y diferencian entre pruebas manuales y comprobaciones automáticas. En 1971, Weinberg publica su libro clásico "La psicología de la programación informática" [197] donde resalta el aspecto humano de la programación, cuyas ideas siguen vigentes.

"En septiembre de 1962, se publicó una noticia en la que se afirmaba que un cohete de 18 millones de dólares se había destruido en el primer vuelo porque **se omitió un guión en la cinta de instrucciones**... Siendo la naturaleza de la programación lo que es, no hay relación entre el **tamaño** del error y el problema que causa. Por lo tanto, es difícil formular cualquier objetivo para las pruebas de programas, salvo **la eliminación de todos los errores**, una tarea imposible."

- Gerald M. Weinberg [197].

En 2010, Weinberg publica el libro "Software perfecto y otras ilusiones sobre las pruebas" [196] donde sostiene que las pruebas son necesarias porque las personas no somos perfectas, pero el hecho de probar más, no garantiza una mayor calidad.



Recomendación: Lea el libro "La psicología de la programación informática" [197], autor Gerald M. Weinberg, 1971.

Bill Elmendorf

En 1967, el ingeniero eléctrico estadounidense Bill Elmendorf publica el artículo "Evaluación de las pruebas funcionales de programas de control" [47] donde se explica por primera vez la necesidad de un enfoque disciplinado para las pruebas funcionales del software. Posteriormente, en 1970, Elmendorf publica el artículo "Diseño automatizado de librerías de pruebas de programas" [46] donde propone la aplicación de las pruebas basadas en modelos para probar software.

Robert W. Bemer

En 1968, el matemático e ingeniero aeronáutico estadounidense Robert W. Bemer participa de la Conferencia de Ingeniería de Software patrocinada por el Comité Científico de la Organización del Tratado del Atlántico Norte (NATO) - bajo el acrónimo en inglés, North Atlantic Treaty Organization - donde uno de los temas tratados fue la garantía de calidad de software. El informe de la conferencia incluyó el documento "Lista de chequeo para planificar la producción de sistemas de software" [20] en el cual se dedica una sección a la garantía de la calidad de software. En este documento se plantean, entre otras, las siguientes preguntas:

- ¿Se ha probado el producto para garantizar que sea lo más útil para el cliente, además de ajustarse a las especificaciones funcionales?
- ¿Se someten los programas de prueba de garantía de la calidad del software al mismo ciclo y método de producción que el software que prueban?
- ¿Se dedica al menos una persona a la garantía de la calidad del software por cada diez que se dedican a su fabricación?

Edsger Dijkstra

En 1968, el físico, matemático y científico de computación de los Países Bajos Edsger Dijkstra escribió una carta al editor de la revista Communications de la Asociación de Maquinaria Computacional (ACM), bajo el acrónimo en inglés, Association for Computing Machinery. La carta se publicó con el título “Sentencia Go To considerada perjudicial” [41] y era una crítica al uso excesivo del Go To por parte de los programadores de la época y las dificultades que esta práctica implicaba para las pruebas. Se considera que esta publicación marcó el inicio de la programación estructurada.

Desde 1966, la ACM otorga anualmente el Premio Turing a personas que han contribuido excepcionalmente al avance de las Ciencias de la Computación. El Premio Turing es considerado el Premio Nobel de la informática. En 1972, Dijkstra recibió el premio Turing. Su discurso de aceptación titulado “El humilde programador” [42] plantea las siguientes ideas:

“Si quieres programadores más eficaces, descubrirás que no deben perder el tiempo depurando, no deben introducir los errores para empezar (...) Una técnica habitual es hacer un programa y luego probarlo. Pero, las pruebas de programas pueden ser una forma muy eficaz de mostrar la presencia de defectos, pero son irremediablemente inadecuadas para demostrar su ausencia. La única forma eficaz de aumentar el nivel de confianza de un programa de forma significativa es dar una prueba convincente de su corrección.”

- Edsger Dijkstra [42].

Recomendación: Investigue los nombres y países de los ganadores del Premio Turing de los últimos cinco años con una breve explicación del motivo de cada reconocimiento.



1.2.3 Período 1971-1980

William C. Hetzel y David Gelerin

En 1973, William C. Hetzel publica el libro “Métodos de prueba de programas” que contiene una compilación de los artículos presentados en un simposio del mismo nombre llevado a

cabo en Chapel Hill, Estados Unidos, donde se expusieron problemas relativos a la validación y pruebas de software. En 1984, Hetzel junto con David Gelperin organizan la Conferencia y Exposición Internacional sobre Pruebas de Software, que es la primera conferencia enfocada exclusivamente en pruebas de software de la que se tiene registro. En 1988, Gelperin y Hetzel publican el artículo “El crecimiento de las pruebas de software” [54] donde describen cuatro modelos para pruebas de software:

- Demostración. Demostrar que el software satisface su especificación.
- Destrucción. Detectar fallos de implementación.
- Evaluación. Detectar defectos en requisitos, diseño, e implementación.
- Prevención. Evitar defectos en requisitos, diseño e implementación.

Ese mismo año, Hetzel publica el libro “Guía completa de pruebas de software” [71] que describe metodologías, técnicas de prueba, y principios de las pruebas de software. En 1992, Gelperin y Hetzel inauguran en Estados Unidos la conferencia “Revisión, análisis y pruebas de software” (STAR) - bajo el acrónimo en inglés, “Software Testing, Analysis, and Review”. Al siguiente año, inauguran en Europa la conferencia paralela denominada EuroSTAR.

Frederick Brooks

En 1975, el científico computacional estadounidense Frederick Brooks publica su obra clásica “El mítico hombre-mes” [25]. Este libro contiene un conjunto de ensayos sobre ingeniería de software. Entre ellos, el más conocido se titula “No hay bala de plata”. Las ideas expuestas por Brooks en sus ensayos siguen teniendo vigencia, incluso para entornos ágiles y DevOps. A continuación, algunas de sus afirmaciones en relación a las pruebas de software:

“Como regla general, estimo que un programa de computación cuesta al menos tres veces más que un programa depurado con la misma función.”

“El cirujano [programador jefe] necesitará un banco de casos de prueba adecuados para probar partes de su trabajo a medida que lo escribe, y luego para probar el conjunto. El probador es, por lo tanto, un adversario que concibe casos de prueba del sistema a partir de las especificaciones funcionales, y al mismo tiempo un ayudante que concibe datos de pruebas para la depuración diaria”.

“Creo que la parte más difícil de la creación de software es la especificación, el diseño, y las pruebas de los constructos conceptuales, no el trabajo de representarlos y probar la fidelidad de dicha representación. Seguiremos cometiendo errores de sintaxis, sin duda, pero dichos errores son insignificantes comparados

con los errores conceptuales de la mayoría de los sistemas."

- Frederick Brooks [25].

Recomendación: Lea el ensayo "No hay bala de plata" en el libro "El mítico hombre-mes" [25], autor Frederick Brooks, 1975.



Tom Gilb

El ingeniero de sistemas estadounidense Tom Gilb publica en 1975 el artículo "Leyes de la no fiabilidad" [55]. Gilb es uno de los primeros informáticos en conceptualizar la fiabilidad del sistema y del software, y la relación entre error humano y error de sistema. Su libro "Métricas de software" [57], publicado en 1976, se considera un texto de referencia por la cantidad de métricas que presenta. En 1993, Gilb publica, junto con Dorothy Graham, el libro "Inspecciones de software" [58] donde se detalla este proceso de revisión formal de software.

Michael E. Fagan

En 1976, el físico e ingeniero eléctrico estadounidense Michael E. Fagan publica el artículo "Inspecciones de diseño y código para reducir errores en el desarrollo de programas" [49] donde propone un proceso sistemático de inspección tanto de diseños como de códigos con el objetivo de reducir el costo del retrabajo. IBM logró importantes mejoras en la calidad al aplicar las inspecciones de Fagan, llegando casi a duplicar el número de líneas de código producidas a la vez que el número de defectos por cada mil líneas de código se redujo en dos tercios.

Thomas J. McCabe

En 1976, Thomas McCabe publica el artículo "Una medida de la complejidad" [151] donde introduce la complejidad ciclomática como métrica de software para el control cuantitativo de la complejidad de un programa. La complejidad ciclomática se basa en la teoría de grafos y toma en cuenta la estructura del programa independientemente de su tamaño o del lenguaje de programación. McCabe también propuso la prueba de ruta básica como una técnica de prueba de caja blanca.

Glenford Myers

El ingeniero eléctrico y científico computacional Glenford Myers publica en 1976 el libro "Fiabilidad del software: Principios y prácticas" [158] en donde proclama:

“El objetivo de los probadores es hacer que el programa falle.”

- Glenford Myers [158].

Unos años más tarde, en 1979, Myers establece la terminología base de las pruebas de software en uno de los primeros libros que tratan exclusivamente sobre pruebas de software titulado “El arte de las pruebas de software” [159] donde introduce el concepto de pruebas de caja negra.

William C. Howden

En 1978, el matemático y doctor en ciencias de la computación William C. Howden publica el artículo “Estudios teóricos y empíricos sobre la comprobación de programas” donde acuña el término oráculo para referirse a un mecanismo para determinar si una prueba ha pasado o fallado.

1.2.4 Período 1981-1990

Barry W. Boehm

El matemático y científico computacional Barry Boehm publica en 1981 el libro “Economía de la ingeniería de software” [23] donde introduce la noción de que el costo de arreglar un defecto en el software, llamado costo de retrabajo, aumenta conforme pasa el tiempo. La Figura 1.6 muestra el gráfico propuesto por Boehm donde en el eje horizontal se tiene las etapas del ciclo de vida en la creación de software, y en el eje vertical se tiene el costo relativo de corregir un defecto detectado en una fase determinada comparado con arreglar el mismo defecto en una fase posterior. La línea con mayor pendiente corresponde a proyectos de software grandes, mientras que la segunda línea, que se muestra entrecortada y con menor pendiente, corresponde a proyectos de software pequeños. En este libro, Boehm también presenta el Modelo de Costos Constructivos (COCOMO) - bajo el acrónimo en inglés, Constructive Cost Model - para costeo de software.

James Martin

En 1984, el físico y consultor de tecnologías de información británico James Martin publica el libro “Manifiesto de los sistemas de información” [148] donde indica que la distribución de la inserción de defectos en un proyecto de software es la siguiente: 56% de los defectos se introducen durante la fase de requisitos, 27% durante el diseño, y 7% durante la codificación.

Paul E. Rook

En 1986, en el artículo “Control de proyectos de software” [170], el científico computacional inglés Paul E. Rook presenta el Modelo V para la creación de software, donde introduce

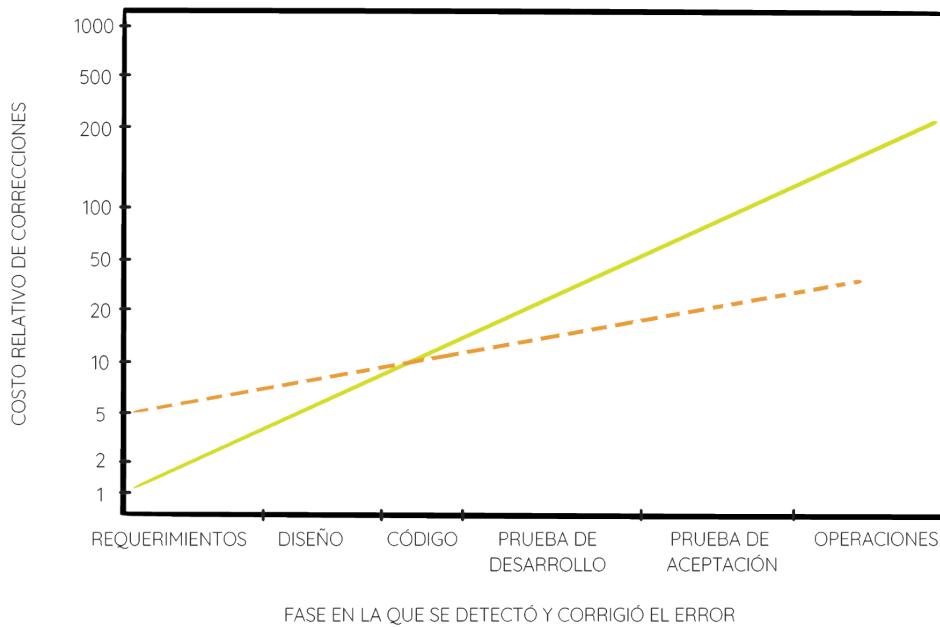


Figura 1.6: Costo relativo de correcciones durante el ciclo de vida en la creación de software.

Fuente: Economía de la ingeniería de software [23]. Adaptación: Autora.

un enfoque estructurado para las pruebas. El Modelo V se populariza en Europa como alternativa al tradicional Modelo Cascada. El Modelo V asocia a cada fase del ciclo de vida una correspondiente fase de pruebas: requisitos con pruebas de aceptación, diseño de sistema con pruebas de sistemas, diseño de software con pruebas de integración, y diseño de componentes con pruebas de unidad.

Robert B. Grady

En 1987, el ingeniero eléctrico estadounidense Robert B. Grady publica, en conjunto con Deborah L. Caswell, el libro “Métricas de software: Establecimiento de un programa para toda la empresa” [64] donde explican qué son las métricas y cuándo son útiles. Este libro presenta un estudio detallado de un programa de métricas. En 1992, Grady publica el libro “Métricas de software prácticas para la gestión de proyectos y la mejora de procesos” [62] donde presenta una taxonomía de defectos de software elaborada para la empresa Hewlett-Packard con el objetivo de identificar tendencias de defectos en productos ya terminados y utilizar esa información para la prevención de defectos en proyectos futuros. En 1996, Grady publica el libro “Mejora exitosa de los procesos de software” [63] donde explica como aplicar el ciclo PDCA a esfuerzos de mejora en el ámbito de software.

Cem Kaner

En 1988 se publica el libro “Pruebas de software informático” [144] escrito por el matemático, abogado, y sicólogo estadounidense Cem Kaner, en conjunto con Jack Falk y Hung Q. Nguyen, que se convirtió en un clásico por su enfoque pragmático. En este libro se utiliza por primera vez el término prueba exploratoria. En 1996, en la Conferencia y Exposición Internacional sobre Pruebas de Software Informático, Cem Kaner y James Bach introducen la noción de escuelas de pensamiento de las pruebas de software. En 1999 nace oficialmente la Escuela de Pruebas Dirigidas por el Contexto. En 2001, Cem Kaner, James Bach, y Bret Pettichord publican el libro “Lecciones aprendidas en pruebas de software: Un enfoque orientado al contexto” [143]. Kaner ha aportado además con leyes en Estados Unidos para el licenciamiento de software, regulación de la calidad de software, y comercio electrónico.

Watts Humphrey

El físico y administrador estadounidense Watts Humphrey es considerado el padre de la calidad de software por sus contribuciones a la mejora del proceso de software (SPI) - bajo el acrónimo en inglés, Software Process Improvement. Humphrey es el fundador del programa de procesos de software del Instituto de Ingeniería de Software (SEI) - bajo el acrónimo en inglés, Software Engineering Institute, asociado a la Universidad Carnegie Mellon. En 1989, Humphrey publica el libro “Gestión del proceso de software” [74] donde propone el modelo de madurez de las capacidades (CMM) - bajo el acrónimo en inglés, Capability Maturity Model - para mejorar la calidad y productividad del proceso de desarrollo de software. En 2005, Humphrey publica el libro “PSP, un proceso de auto-superación para ingenieros de software” [75] donde describe un proceso personal de software (PSP) - bajo el acrónimo en inglés, Personal Software Process - donde reduce las prácticas de software industrial para adaptarlas a las necesidades del desarrollo de programas de tamaño modular. En 2006, Humphrey publica el libro “TSP, Dirigiendo un equipo de desarrollo” [76] donde explica cómo liderar a equipos de ingenieros de software formados en PSP utilizando un proceso de software en equipo (TSP) - bajo el acrónimo en inglés, Team Software Process.

Boris Beizer

En 1990, el físico, ingeniero eléctrico, y científico computacional belga Boris Beizer propone una clasificación de defectos de software en el libro “Técnicas de pruebas de software” [18]. Adicionalmente, Beizer acuña el término “paradoja del pesticida” para describir el fenómeno de que cuanto más se prueba el software, más inmune se vuelve éste a las pruebas a las que se le somete.



Recomendación: Investigue cuáles son los siete principios de las pruebas de software, de los cuales el quinto es la paradoja del pesticida.



Figura 1.7: Dorothy Graham y Sandra Sánchez Gordón en el 7mo. Congreso Mundial de Calidad de Software, 2017, Lima-Perú.

Fuente: Archivo personal.

1.2.5 Período 1991-2000

Dorothy Graham

En 1991, Unicom publica el primer “Reporte sobre pruebas de software asistidas por computador (CAST)” [65] - bajo el acrónimo en inglés, Computer Aided Software Testing - escrito por la consultora en pruebas de software estadounidense Dorothy Graham. En 1999, Graham publica en conjunto con Mark Fewster el libro “Automatización de pruebas de software” [51], considerado una obra clásica en el ámbito de la automatización de pruebas. En 2006, Graham publica en conjunto con Erik Van Veenendaal, Isabel Evans, y Rex Black el libro “Fundamentos de las pruebas de software: Certificación ISTQB” [67] donde se describe el programa de estudios para la certificación de nivel básico de pruebas del Comité Internacional de Cualificaciones de Pruebas de Software (ISTQB) - bajo el acrónimo en inglés, International Software Testing Qualifications Board [133]. En 2017, tuve el gusto de conocer personalmente a Dorothy en el 7mo Congreso Mundial de Calidad de Software que se realizó en Lima, Perú (Figura 1.7).

Brian Marick

En 1994, el consultor de pruebas y agilismo Brian Marick publica el libro “El arte de las pruebas de software: Pruebas de subsistemas, incluidas las pruebas basadas en objetos y las orientadas a objetos” donde manifiesta que probar software es un oficio, como la carpintería, que se aprende mejor en persona, viendo cómo lo hace otra persona más experimentada e intentando hacerlo bajo su supervisión. El libro se enfoca en subsistemas de tamaño medio,

tales como controladores de dispositivos, bibliotecas de clases, módulos de optimización en compiladores, entre otros. En 2001, Marick participa como uno de los autores del Manifiesto Ágil. En 2003, Marick publica una serie de artículos sobre pruebas ágiles, entre ellos el artículo “Cuadrantes de pruebas ágiles” donde define dos dimensiones para categorizar los tipos de pruebas: pruebas de cara al negocio versus pruebas de cara a la tecnología; y pruebas que dan soporte a la programación versus pruebas que critican el producto.



Recomendación: Investigue en que consisten los cuadrantes de las pruebas ágiles.

Paul C. Jorgensen

En 1995, el matemático y doctor en informática Paul C. Jorgensen publica el libro “Pruebas de software: Un enfoque artesanal” [137]. En 2022, Jorgensen publica la quinta edición en conjunto con Byron DeVries. Las ediciones de este libro se han convertido en referencia de las tecnologías en evolución en el ámbito de las pruebas de software [138].

R. Geoff Dromey

En 1996, en su artículo “Acorralando a la quimera” [44], el doctor en física-química australiano R. Geoff Dromey propone un modelo de calidad para resolver la intangibilidad de las características de calidad propuestas en la norma ISO/IEC 9126:1991 “Ingeniería de Software - Calidad de Producto” [110].

James Bach

En 1996, James Bach propone el Modelo de Estrategia de Pruebas Heurísticas, que consiste en un conjunto de patrones para diseñar y elegir las pruebas que se van a realizar en un proyecto de pruebas de software. El propósito de este modelo es enfatizar que la selección de técnicas o heurísticas de prueba a utilizar debe tomar en cuenta el ambiente del proyecto, los elementos del producto, y los criterios de calidad. En 2001, Bach crea la metodología Pruebas Rápidas de Software (RST) - bajo el acrónimo en inglés, Rapid Software Testing - alineada a la Escuela de Pruebas Dirigidas por el Contexto.

Eric S. Raymond

En 1999, el desarrollador de software estadounidense Eric S. Raymond publica el libro “La catedral y el bazar” [168] donde describe el método de desarrollo de software que utilizó Linus Torvalds para crear el sistema operativo Linux. Raymond detalla 19 pautas para crear un buen software de código abierto y presenta la llamada Ley de Linus, que afirma:

“Si hay suficientes ojos, todos los errores son superficiales.”
- Linus Torvalds [168].

La Ley de Linus implica que cuanto más públicamente disponible esté el código fuente de un software para su comprobación, escrutinio, y experimentación por parte una base grande de probadores y desarrolladores, más rápidamente se descubrirán, caracterizarán, y solucionarán los defectos presentes en dicho software.

Jonathan Bach

En 2000, Jonathan Bach publica el artículo “Gestión de pruebas basada en la sesión” [12]. Una sesión es un bloque ininterrumpido de esfuerzo de prueba con una misión puntual donde se utiliza pruebas exploratorias y se reportan los resultados al término de la misma. En 2007, Bach propone la escala de libertad del probador. Esta escala modela la variación en el grado de libertad que tiene un probador cuando realiza pruebas, y va desde “Completamente guiado” hasta “Estilo libre exploratorio”. En palabras de Bach, la escala modela el grado en que se nos permite pensar.

1.2.6 Período 2001-2024

Kent Beck

En 2002, el ingeniero en ciencias de la computación estadounidense Kent Beck publica el libro “Desarrollo dirigido por pruebas: Mediante el ejemplo” [16] donde “re-descubre” la técnica de desarrollo de software que consiste en escribir las pruebas antes escribir el código, y la denomina Desarrollo Guiado por las Pruebas (TDD) - bajo el acrónimo en inglés, Test Driven Development. Otras contribuciones de Beck al desarrollo de software son los patrones de software, la familia de herramientas de pruebas unitarias xUnit , y la programación extrema (XP) - bajo el acrónimo en inglés, Extreme Programming.

Bret Pettichord

En 2003, el arquitecto de calidad de software estadounidense Bret Pettichord participa en un taller de capacitación en pruebas de software donde expone la conferencia titulada “Cuatro escuelas de pruebas de software” y propone la existencia de escuelas de pensamiento en las pruebas de software, a las que denomina: analítica, dirigida por normas, orientada hacia la calidad, y dirigida por el contexto. Posteriormente, se incorpora a la lista a la escuela ágil.

Michael Bolton

En 2004, el consultor de pruebas de software canadiense Michael Bolton se suma como co-autor de la metodología RST creada por James Bach. En 2009, en su artículo “Probando

vs. comprobando” [24], Bolton distingue entre estos dos conceptos. Para Bolton, comprobar es confirmar, verificar, y validar utilizando herramientas automáticas, mientras que probar es el proceso de exploración, descubrimiento, investigación, y aprendizaje realizado por los probadores.

Erik Van Veenendaal

En 2005, el consultor de pruebas de software neerlandés Erik Van Veenendaal, en conjunto con otros expertos, crea la Fundación TMMI con el objetivo de desarrollar el Modelo de Madurez de Pruebas Integrado TMMI, bajo el acrónimo en inglés, Test Maturity Model Integration [189]. El modelo TMMI sirve para evaluar y mejorar el proceso de pruebas de las organizaciones y se basa en su predecesor, el modelo TMM desarrollado en 1996 en el Instituto de Tecnología de Illinois.

Doron Reuveni

En 2007, el ingeniero en sistemas computacionales y sistemas de información israelí Doron Reuveni, en conjunto con Roy Solomon, publica el libro “Guía esencial de crowdtesting” [6]. El término crowdtesting surgió del término crowdsourcing que fue acuñado en 2006, por Jeff Howe y Mark Robinson, para describir la externalización de parte de las actividades de las organizaciones a grupos de voluntarios. Crowdtesting se basa en el enfoque de pruebas en el medio natural en lugar del laboratorio de calidad o la organización desarrolladora, buscando incluir la mayor cantidad de contextos de usos y dispositivos.

Mike Cohn

En 2009, Mike Cohn publica el libro “Triunfando con la agilidad” [33] donde propone la “pirámide de automatización de pruebas”. En este modelo, Cohn argumenta que una estrategia de automatización de pruebas eficaz requiere la automatización de pruebas en tres niveles: unidad, servicio, e interfaz de usuario.



Recomendación: Investigue en que consiste la “pirámide de automatización de pruebas” de Cohn.

Lisa Crispin

En 2009, la experta en pruebas de software y agilismo Lisa Crispin publica, en conjunto con Janeth Gregory, el libro “Pruebas ágiles: Una guía práctica para probadores y equipos ágiles” [36] que incluye un capítulo sobre pruebas exploratorias escrito con el apoyo de Michael Bolton. Este libro es considerado pionero en la disciplina de las pruebas ágiles. Adicionalmente, en 2014, Crispin y Gregory publicaron otro texto importante en el mismo ámbito,

titulado “Más pruebas ágiles: Viajes de aprendizaje para todo el equipo” [68]. Este libro abarca la adaptación de las pruebas ágiles a entornos y equipos, el aprendizaje a partir de la experiencia, y la mejora continua de los procesos de prueba.

Jonathan Kohl

En 2012, el consultor de pruebas de software canadiense Jonathan Kohl contribuye con el capítulo “La automatización es mucho más que pruebas de regresión: Pensando fuera de la caja” que forma parte del libro “Experiencias de automatización de pruebas: Casos prácticos de automatización de pruebas de software” [66] escrito por Dorothy Graham y Mark Fewster. En este capítulo, Kohl propone utilizar la automatización para llevar a cabo tareas tales como configuración de pruebas, generación de datos, y avance a lo largo de un flujo de trabajo. Adicionalmente, Kohl propone la utilización de pruebas exploratorias manuales para encontrar aquellos defectos insidiosos que de otro modo escaparían a la atención de los probadores.

1.3 Normalización

La normalización es el proceso de desarrollo y aplicación de normas técnicas. El proceso de normalización establece un acuerdo común para términos, principios, prácticas, procesos, y otros elementos necesarios dentro de un cuerpo de conocimiento. La normalización además facilita la difusión de dicho cuerpo de conocimiento.

Definición 1.7 Norma según ISO 24765:2017 [126]. “Conjunto de requisitos obligatorios establecidos por consenso y mantenidos por una organización reconocida para prescribir un enfoque uniforme y disciplinado, o para especificar un producto, con respecto a convenciones y prácticas obligatorias.”

1.3.1 Importancia

La adopción y adaptación de normas técnicas ayuda a tener un proceso de desarrollo de software más riguroso, lo que a su vez, permite obtener un producto de mejor calidad. El objetivo es reducir el número de defectos insertados en el software y, mediante el proceso de pruebas, depurar los defectos residuales antes de la entrega del producto. La industria de software gradualmente va asimilando los beneficios del uso de normas técnicas. Las normas internacionales, a diferencia de otras directrices como por ejemplo procesos o mejores prácticas, son documentos que pueden convertirse en requisitos de cumplimiento obligatorio cuando son adscritas por órganos reguladores de países.

Ejemplo 1.3 Norma ISO/IEC 40500:2012.

La norma ISO/IEC 40500:2012 “Tecnología de la información - Directrices de accesibilidad al contenido web (WCAG) 2.0” [108] es de cumplimiento obligatorio en muchos países.

AÑO	PAÍS	ÓRGANO REGULADOR	DESCRIPCIÓN
2010	Argentina	Instituto Argentino de Normalización y Certificación (IRAM)	Ley 26.653. Establece que el estado nacional, los entes públicos no estatales, las empresas del Estado y las empresas privadas concesionarias de servicios públicos, deberán respetar en los diseños de sus páginas Web las normas y requisitos sobre accesibilidad de la información que faciliten el acceso a sus contenidos a todas las personas con discapacidad [34].
2011	Colombia	Instituto Colombiano de Normas Técnicas y Certificación (ICONTEC)	Norma Técnica Colombiana NTC 5854. Los sitios web del Estado deben cumplir con el nivel de accesibilidad AA establecido en la NTC 5854 [53].
2012	España	Asociación Española de Normalización (AENOR)	Norma UNE-EN 301 549. En el Real Decreto 1494 se aprueba el reglamento que obliga a las webs de las administraciones públicas a cumplir los requisitos de prioridad 1 y 2 especificados en la norma UNE-EN 301 549 [1].
2014	Ecuador	Servicio Ecuatoriano de Normalización (INEN)	Normativa técnica INEN-ISO/IEC 40500. Aplica a sitios web del sector público y privado que presten servicios públicos. Los propietarios de sitios web tuvieron plazo hasta 2018 para adecuar sus sitios web al nivel de conformidad AA. Acorde al reglamento RTE INEN 288, el incumplimiento implica sanciones previstas en la Ley No. 2007-76 del Sistema Ecuatoriano de la Calidad [180].

Los esfuerzos de normalización han sido liderados por las siguientes organizaciones:

- Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) - bajo el acrónimo en inglés, Institute of Electrical and Electronics Engineers. El IEEE tiene su sede central en Estados Unidos. En 1963 adopta su nombre actual pero fue creado inicialmente en 1884 por el científico e inventor estadounidense Thomas Alva Edison, el científico e inventor británico-estadounidense Alexander Graham Bell, y el ingeniero electrotécnico e inventor Franklin Pope.
- Comisión Electrotécnica Internacional (IEC) - bajo el acrónimo en inglés, International Electrotechnical Commission. La IEC tiene su sede central en Suiza y fue fundada en 1906. Su primer presidente fue el físico y matemático británico William Thomson, también conocido como Lord Kelvin.
- Organización Internacional de Normalización (ISO) - bajo el acrónimo en inglés, International Organization for Standardization. La ISO tiene su sede central en Suiza. La ISO fue creada inicialmente en 1926 y suspendida temporalmente durante la Segunda Guerra Mundial. En 1947 adopta su nombre actual.
- Asociación de Maquinaria Computacional (ACM) - bajo el acrónimo en inglés, Association for Computing Machinery. La ACM tiene su sede central en Estados Unidos y fue fundada en 1947 como la primera organización en el campo de la computación. ACM fue fundada por el matemático estadounidense Richard Hamming.

1.3.2 Normas relativas a calidad de software

A continuación se describen las normas relativas a calidad de software más utilizadas en orden cronológico de publicación de su primera versión. La primera norma que promovió una visión integral de la calidad de software fue publicada en 1980 por el IEEE. Esta norma trataba sobre planes de garantía de calidad del software e impulsó el desarrollo de normas subsecuentes sobre requisitos, diseño, pruebas, y verificación y validación de software.

Norma IEEE Std. 730 - Procesos de garantía de la calidad del software y Norma ISO/IEC 12207 - Procesos del ciclo de vida del software

En el año 1976, el IEEE conforma el sub-comité para el desarrollo de normas de ingeniería de software. Este equipo, liderado por el ingeniero eléctrico estadounidense Fletcher Buckley, desarrolla la norma IEEE Std. 730-1980 “Planes de garantía de la calidad del software” [77]. El objetivo inicial de la norma IEEE Std. 730-1980 era proporcionar requisitos uniformes para la preparación, estructura, y contenido de los planes de garantía de la calidad del software. La revisión vigente a la fecha de publicación de la primera edición de este texto es la norma IEEE Std. 730-2014 “Procesos de garantía de la calidad del software” [78] que amplía el

alcance original para abordar los procesos definidos en el marco del ciclo de vida del software establecido en la norma ISO/IEC 12207:2008 “Ingeniería de sistemas y software - Procesos del ciclo de vida del software” [86], cuya primera revisión fue en 1995. La revisión de la norma ISO/IEC 12207 vigente a la fecha de publicación de la primera edición de este texto corresponde al año 2017 [125].

Norma ISO/IEC/IEEE 90003 - Directrices para la aplicación de la norma ISO 9001:2015 al software computacional

En 1991 se publica la primera versión de la norma ISO/IEC 9000-3:1997 “Normas de gestión de la calidad y de garantía de la calidad - Parte 3: Directrices para la aplicación de la norma ISO 9001 al desarrollo, suministro, y mantenimiento de software” [83]. Esta norma fue concebida como una lista de comprobación para el desarrollo, suministro, y mantenimiento de software en alineación con la norma ISO 9001. La revisión de esta norma vigente a la fecha de publicación de la primera edición de este texto se denomina ISO/IEC/IEEE 90003:2018 “Ingeniería de software - Directrices para la aplicación de la norma ISO 9001:2015 a software computacional” [132].

Serie de normas ISO/IEC 25000 - Evaluación y requisitos de calidad de sistemas y software

En 1991 se publica la primera revisión de la norma ISO/IEC 9126:1991 “Ingeniería de software - Calidad del producto” [110] que propone un modelo de calidad para productos de software con seis características. En 1998, se publica la primera revisión de la norma ISO/IEC 14598-5:1998 “Tecnología de la información - Evaluación de productos de software - Parte 5: Proceso para evaluadores” [87] que describe las etapas para evaluar la calidad de un producto de software en base a las características de calidad propuestas en la norma ISO/IEC 9126. En 2011, las normas ISO/IEC 9126 e ISO/IEC 14598 fueron absorbidas por la serie de normas ISO/IEC 25000:2014 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Guía de SQuaRE” [88]. Las normas que forman parte de la serie ISO/IEC 25000 son:

- Norma ISO/IEC 25000:2014 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Guía de SQuaRE” [88].
- Norma ISO/IEC 25001:2014 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Planificación y gestión” [89].
- Norma ISO/IEC 25010:2023 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Modelo de calidad de producto” [90].

- Norma ISO/IEC TS 25011:2017 “Tecnología de la información - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Modelos de calidad del servicio” [114].
- Norma ISO/IEC 25012:2008 “Ingeniería de software - Evaluación y requisitos de calidad de productos de software (SQuaRE) - Modelo de calidad de datos” [91].
- Norma ISO/IEC 25019:2023 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Modelo de calidad en uso” [92].
- Norma ISO/IEC 25020:2019 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Marco de medición de la calidad” [93].
- Norma ISO/IEC TR 25021:2012 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Elementos de medida de la calidad” [119].
- Norma ISO/IEC 25022:2016 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Medición de la calidad en uso” [94].
- Norma ISO/IEC 25023:2016 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Medición de la calidad de los productos de sistemas y software” [95].
- Norma ISO/IEC 25030:2019 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Marco de requisitos de calidad” [96].
- Norma ISO/IEC 25040:2011 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Proceso de evaluación” [97].
- Norma ISO/IEC 25041:2012 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Guía de evaluación para desarrolladores, adquirentes y evaluadores independientes” [98].
- Norma ISO 25065:2019 “Ingeniería de sistemas y software - Evaluación y requisitos de calidad de productos de software (SQuaRE) - Formato común de la industria (CIF) para la usabilidad: Especificación de requisitos de usuario” [99].

Serie de normas ISO/IEC 33000 - Evaluación de procesos

En 1998 se publica la serie de normas ISO/IEC 15504 “Mejora de los procesos de software y determinación de la capacidad” (SPICE) - bajo el acrónimo en inglés, Software Process

Improvement and Capability dEtermination, que es un marco de trabajo para evaluar y mejorar procesos de software. En 2015, esta serie fue reemplazada por la serie de normas ISO/IEC 33000 “Evaluación de procesos”. Esta serie proporciona un marco para la evaluación de las características de calidad de los procesos y la madurez de la organización. El marco de evaluación propuesto abarca los procesos empleados en el desarrollo, evolución, y uso de sistemas en el ámbito de las tecnologías de la información; así como también los procesos empleados en el diseño, transición, prestación, y mejora de servicios. Los resultados de la evaluación pueden aplicarse para mejorar el rendimiento de los procesos, realizar una evaluación comparativa, o identificar los riesgos asociados a la aplicación de los procesos. Algunas de las normas que forman parte de la serie ISO/IEC 33000 son:

- Norma ISO/IEC 33001:2015 “Tecnología de la información - Evaluación de procesos - Conceptos y terminología” [102].
- Norma ISO/IEC 33002:2015 “Tecnología de la información - Evaluación de procesos - Requisitos para realizar evaluación de procesos” [103].
- Norma ISO/IEC 33003:2015 “Tecnología de la información - Evaluación de procesos - Requisitos para marcos de medición de procesos” [104].
- Norma ISO/IEC 33004:2015 “Tecnología de la información - Evaluación de procesos - Requisitos para la referencia de procesos, evaluación de procesos, y modelos de madurez” [105].
- Norma ISO/IEC DTS 33010 “Tecnología de la información - Evaluación de procesos - Guía para la realización de evaluaciones de procesos” [111].
- Norma ISO/IEC TR 33014:2013 “Tecnología de la información - Evaluación de procesos - Guía para la mejora de procesos” [113].
- Norma ISO/IEC 33020:2019 “Tecnología de la información - Evaluación de procesos - Marco de medición de procesos para la evaluación de la capacidad de los procesos” [106].
- Norma ISO/IEC TS 33030:2017 “Tecnología de la información - Evaluación de procesos - Un proceso de evaluación documentado ejemplar” [115].
- Norma ISO/IEC 33063:2015 “Tecnología de la información - Evaluación de procesos - Modelo de evaluación de procesos para pruebas de software” [107].

Norma ISO/IEC TR 19759 - Cuerpo de conocimientos de ingeniería de software

En 2005, la ISO publica la primera revisión de la norma ISO/IEC TR 19759. Esta norma define el alcance de la disciplina de la ingeniería del software y proporciona acceso por temas

a la literatura publicada. La revisión vigente a la fecha de publicación de la primera edición de este texto es la norma ISO/IEC TR 19759:2015 “Ingeniería de software - Guía del cuerpo de conocimientos de ingeniería de software (SWEBOK)” [118] - por sus siglas en inglés, Software Engineering Book of Knowledge).

Serie de normas ISO/IEC 29110 - Normas y directrices de ingeniería de sistemas y software para entidades muy pequeñas (VSE)

Definición 1.8 VSE según norma ISO/IEC TR 29110-1:2016 [120]. “Empresa, organización, departamento, o proyecto que cuenta como máximo con 25 personas.”

En 2005, la ISO forma un grupo de trabajo coordinado por el físico y doctor en informática canadiense Claude Laporte para el desarrollo de la serie de normas y reportes técnicos ISO/IEC 29110. Esta serie proporciona un mapa de ruta para entidades muy pequeñas (VSE) - bajo el acrónimo en inglés, Very Small Entities - que desarrollan sistemas o software. La serie ISO/IEC 29110 define un grupo de cuatro perfiles genéricos: perfil de entrada, perfil básico, perfil intermedio, y perfil avanzado, para guiar a las VSE en su evolución desde el inicio hasta la madurez. Las directrices proporcionan a las VSE un conjunto de roles, procesos, actividades, y tareas, así como el contenido de los productos de trabajo, tanto para la gestión del proyecto como para la implementación de productos software. La ISO/IEC 29110 puede ser utilizada con cualquier ciclo de vida: cascada, iterativo, incremental, evolutivo, o ágil. Algunas de las normas que forman parte de la serie ISO/IEC 29110 son:

- Norma ISO/IEC TR 29110-1:2016 “Ingeniería de sistemas y software - Perfiles del ciclo de vida para entidades muy pequeñas (VSEs) - Parte 1: Visión general” [120].
- Norma ISO/IEC 29110-2-1:2015 “Ingeniería del software - Perfiles del ciclo de vida para entidades muy pequeñas (VSE) - Parte 2-1: Marco y taxonomía” [100].
- Norma ISO/IEC TR 29110-3-1:2020 “Ingeniería de sistemas y software - Perfiles del ciclo de vida para entidades muy pequeñas (VSE) - Parte 3-1: Directrices para la evaluación de procesos” [121].
- Norma ISO/IEC 29110-3-3:2016 “Ingeniería de sistemas y software - Perfiles del ciclo de vida para empresas muy pequeñas (VSE) - Parte 3-3: Requisitos de certificación para las evaluaciones de conformidad de los perfiles VSE utilizando la evaluación de procesos y los modelos de madurez” [101].
- Norma ISO/IEC 29110-4-1:2018 “Ingeniería de sistemas y software - Perfiles del ciclo de vida para entidades muy pequeñas (VSE) - Parte 4-1: Ingeniería del software - Especificaciones de los perfiles: Grupo de perfiles genéricos” [112].

- Norma ISO/IEC TR 29110-5-1-2:2011 “Ingeniería del software - Perfiles del ciclo de vida para entidades muy pequeñas (VSE) - Parte 5-1-2: Guía de gestión e ingeniería: Grupo de perfiles genéricos: Perfil básico” [122].

Serie de normas ISO/IEC/IEEE 29119 - Pruebas de software

En 2007, la ISO forma un grupo de trabajo liderado por el doctor en pruebas de software británico Stuart Reid para desarrollar la serie de normas ISO/IEC/IEEE 29119 específicas para pruebas de software. Los conceptos generales y terminología común se definen en la Parte 1. Los procesos de prueba están definidos en la Parte 2, y abarcan las pruebas a nivel de organización, la gestión de las pruebas, y los niveles dinámicos de las pruebas. La documentación de las pruebas, definida en la Parte 3, se elabora durante la ejecución de los procesos de pruebas. Las diferentes técnicas de diseño de pruebas se definen en la Parte 4. A la fecha de publicación de la primera edición de este texto, esta serie abarca las siguientes normas:

- Norma ISO/IEC/IEEE 29119-1:2022 “Ingeniería de software y sistemas - Pruebas de software - Parte 1: Conceptos generales” [127].
- Norma ISO/IEC/IEEE 29119-2:2021 “Ingeniería de software y sistemas - Pruebas de software - Parte 2: Procesos de prueba” [128].
- Norma ISO/IEC/IEEE 29119-3:2021 “Ingeniería de software y sistemas - Pruebas de software - Parte 3: Documentación de las pruebas” [129].
- Norma ISO/IEC/IEEE 29119-4:2021 “Ingeniería de software y sistemas - Pruebas de software - Parte 4: Técnicas de prueba” [130].
- Norma ISO/IEC/IEEE 29119-5:2016 “Ingeniería de software y sistemas - Pruebas de software - Parte 5: Pruebas basadas en palabras clave” [131].
- Norma ISO/IEC TR 29119-6:2021 “Ingeniería de software y sistemas - Pruebas de software - Parte 6: Directrices para el uso de la norma ISO/IEC/IEEE 29119 (todas las partes) en proyectos ágiles” [124].
- Norma ISO/IEC TR 29119-11:2020 “Ingeniería de software y sistemas - Pruebas de software - Parte 11: Directrices para pruebas de sistemas basados en IA” [123].

Norma ISO/IEC 20246 - Revisiones de productos de trabajo

En 2017 se publica la norma ISO/IEC 20246:2017 “Ingeniería de software y sistemas - Revisiones de productos de trabajo” [117] que describe un proceso genérico, actividades, tareas, técnicas de revisión, y plantillas de documentación a aplicarse durante la revisión

de un producto de trabajo. Esta norma reemplaza a IEEE Std. 1028-2008 “Revisiones y auditorías de software” [79].

1.4 Definiciones

La calidad de un producto de software puede entenderse, analizarse, y evaluarse de diferentes formas. Cada punto de vista es importante, y centrarse en un solo aspecto con el riesgo de descuidar alguno de los demás, puede llevar a una percepción sesgada de la calidad del software. Por ello, se necesita una estrategia integral de calidad de software que se fundamente en un marco conceptual aceptado por todas las partes interesadas. A continuación se presentan definiciones relativas a software, calidad de software, error, defecto, bug, y fallo utilizadas en este libro de texto.

1.4.1 Definiciones relativas a software

En este apartado se exponen las definiciones de: sistema, software, elemento de software, proceso de software, y producto de trabajo.

Definición 1.9 Sistema según norma ISO/IEC 25010:2023 [90]. “Combinación de elementos que interactúan entre sí y que se organizan para lograr uno o más propósitos declarados. Por ejemplo, un sistema informático se compone de hardware, software, y humanware.”

Definición 1.10 Software según norma ISO/IEC/IEEE 90003:2018 [132]. “Conjunto de programas informáticos, procedimientos y, posiblemente, documentación y datos asociados. Un producto de software puede ser destinado para la entrega, ser una parte integral de otro producto, o ser utilizado en el desarrollo. El software incluye al firmware.”

Definición 1.11 Elemento de software según norma ISO/IEC/IEEE 90003:2018 [132]. “Parte identificable de un producto de software.”

Definición 1.12 Proceso de software según Sommerville [184]. “Conjunto de actividades relacionadas que conducen a la producción del software. Estas actividades pueden implicar el desarrollo del software desde cero o la modificación de un producto existente.”

Definición 1.13 Producto de trabajo según norma ISO/IEC 20246:2017 [117]. “Un producto de trabajo es cualquier artefacto producido por un proceso. Son ejemplos de productos de trabajo de un proceso de software: plan del proyecto, especificación de requisitos, documentación de diseño, código fuente, plan de pruebas, entre otros.”

1.4.2 Definiciones relativas a calidad de software

En este apartado se exponen las definiciones de: calidad de software, control de calidad de software, garantía de calidad de software, y gestión de calidad de software.

Definición 1.14 Calidad de software según norma IEEE Std. 730-2014 [78]. “El grado en que un producto de software cumple los requisitos establecidos; sin embargo, la calidad depende del grado en que esos requisitos establecidos representan con precisión las necesidades, deseos, y expectativas de las partes interesadas.”

Definición 1.15 Control de calidad de software (SQC) según Pressman y Maxim [167]. Se abrevia SQC bajo el acrónimo en inglés, Software Quality Control. “Conjunto de actividades que ayudan a eliminar problemas de calidad en los productos de trabajo. Para determinar si las actividades de control de calidad están siendo efectivas se utiliza una combinación de mediciones y retroalimentación.”

Definición 1.16 Garantía de calidad de software (SQA) según norma IEEE Std. 730-2014 [78]. Se abrevia SQA bajo el acrónimo en inglés, Software Quality Assurance. “Conjunto de actividades independientes que definen y evalúan la adecuación de los procesos de software para proporcionar evidencias que establezcan confianza en que los procesos son apropiados y producen productos de software de calidad acorde a los fines previstos.”

Definición 1.17 Gestión de calidad de software según norma IEEE Std. 730-2014 [78]. “Actividades coordinadas para dirigir y controlar una organización con respecto a la calidad del software.”

1.4.3 Definiciones de error, defecto, bug, y fallo

“Cuiusvis hominis est errare: nullius nisi insipientis, in errore perseverare.”

“Errar es propio de todo humano, pero sólo un insensato persevera en el error.”

- Marco Tulio Cicerón [32].

Todo miembro del equipo de desarrollo de un producto de software está sujeto a cometer un error o equivocación. Los errores pueden deberse a una infinidad de razones como

descuido, cansancio, apuro, falta de comunicación, falta de conocimiento, entre otros. Los errores también pueden ser originados por actores no humanos que intervienen en el proceso, por ejemplo, herramientas automatizadas de generación de código.

Definición 1.18 Error según ISTQB [135]. “Acción humana que produce un resultado incorrecto.”

Un error cometido implica la inserción de uno o varios defectos en un producto de trabajo específico durante una fase puntual del ciclo de vida en la creación de software, lo que puede desencadenar la inserción de más defectos en otros productos de trabajo o en elementos del software durante fases posteriores.

Definición 1.19 Defecto de software según ISTQB [135]. “Una imperfección o deficiencia en un producto de trabajo que hace que no cumpla con sus requisitos o especificaciones.”

El término bug (en español, bicho) es utilizado como sinónimo de defecto y forma parte de la jerga de la ingeniería desde el Siglo XIX. Originalmente, se utilizaba en la ingeniería de hardware para referirse a fallos mecánicos. En 1878, el inventor estadounidense Thomas Alva Edison - quien creó el primer laboratorio de investigación industrial del mundo - estaba trabajando para Western Union en mejorar el teléfono. Edison escribió una carta al presidente de Western Union indicando problemas en el nuevo diseño de teléfono donde, a modo de broma, indicaba que encontró un 'bug' de tipo 'callbellum', haciendo referencia a su competidor, el inventor británico Alexander Graham Bell. La Figura 1.8 muestra un extracto de la carta en mención, donde se lee:

“Tenías razón en parte, encontré un bug en mi aparato, pero no estaba en el teléfono propiamente dicho. Era del género 'callbellum'. El insecto parece encontrar condiciones para su existencia en todos los aparatos de llamada de teléfonos.”
- Thomas Alva Edison [45].

En 1947, la pionera de la informática Grace Hopper utiliza el término bug en un relato acerca del computador electromecánico Mark II. Los operadores descubrieron una polilla atrapada en un relé del Mark II. El insecto fue extraído cuidadosamente del relé y pegado en el cuaderno de bitácora, como se muestra en la Figura 1.9. Los operadores que encontraron la polilla estaban familiarizados con el uso del término bug en ingeniería, por lo que escribieron divertidamente la siguiente anotación:

“Primer caso real de bug encontrado.”

- Bitácora Mark II [192].

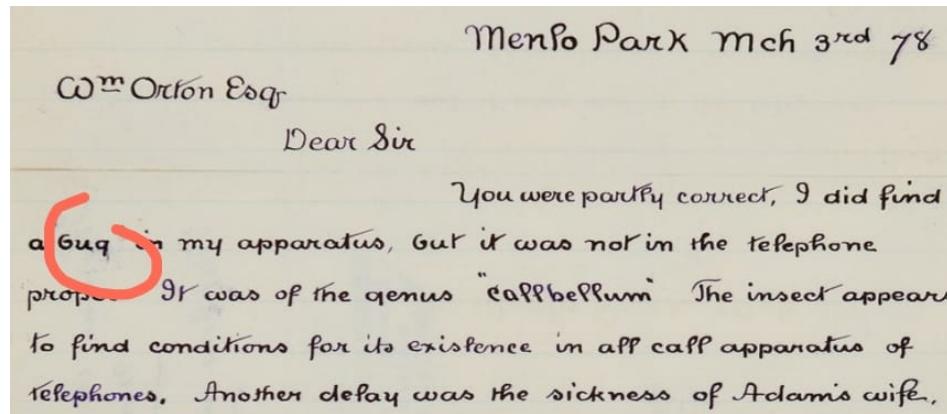


Figura 1.8: Carta de Thomas Alva Edison de 1878.

Fuente: Galerías de Subastas Swann [45].

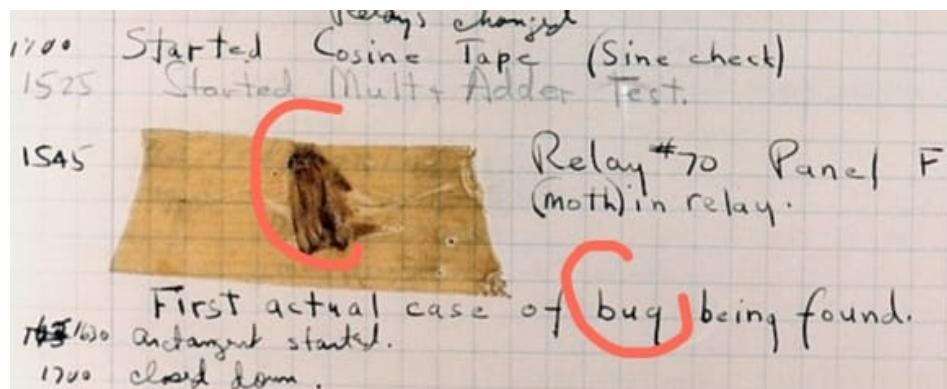


Figura 1.9: Captura de la bitácora del computador Mark II.

Fuente: Librería del Centro Naval Histórico de Estados Unidos [192].

Definición 1.20 Fallo de software según ISTQB [135]. “Evento en el que un componente o sistema no ejecuta una función requerida dentro de los límites especificados.”

Si se dan las circunstancias necesarias, los defectos o bugs introducidos en el software pueden provocar fallos, los mismos que pueden repercutir negativamente en los clientes y usuarios con afectaciones económicas, a la salud, al entorno, e incluso pérdida de vidas. Las condiciones del entorno también pueden provocar fallos durante la ejecución del software. Por ejemplo, si el correcto funcionamiento de un dispositivo de entrada de datos se ve afectado por clima extremo.

1.4.4 Ejemplos de fallos

Las consecuencias de la mala calidad del software pueden ser molestias e inconvenientes, pero también generan costos en daños, costos para corregir el software, pérdida de credibilidad de la empresa, o incluso pérdida de vidas humanas. Según un estudio realizado en 2020 por el Instituto Nacional de Estándares y Tecnología (NIST) - bajo el acrónimo en inglés, National Institute of Standards and Technology - del Departamento de Comercio de los Estados Unidos, los fallos de software cuestan a la economía estadounidense 59.500 millones de dólares al año. Este estudio afirma que el impacto de los fallos de software es enorme porque prácticamente todas las empresas de Estados Unidos dependen de software para el desarrollo, producción, distribución, y soporte posventa de productos y servicios. El estudio también reveló que la mejora del proceso de pruebas podría sacar a la luz defectos y que la eliminación de los mismos en fases iniciales del desarrollo podría reducir el costo en unos 22.200 millones de dólares. A continuación se presentan algunos ejemplos de fallos de software y sus consecuencias que ilustran porque la calidad debe tenerse muy en cuenta al desarrollar productos de software.

Ejemplo 1.4 Sonda espacial Mariner 1.

El 22 de julio de 1962, la Administración Nacional de Aeronáutica y el Espacio (NASA) - bajo el acrónimo en inglés, National Aeronautics and Space Administration - de Estados Unidos lanzó un cohete que transportaba a la sonda espacial Mariner 1. La Mariner 1 tenía por objetivo orbitar Venus. Apenas despegó, el cohete se desvió de su curso con el riesgo de un aterrizaje forzoso, por lo que, 290 segundos después del lanzamiento, la NASA emitió una orden de autodestrucción. La investigación posterior determinó que la causa fue la omisión del símbolo “-” en el código escrito en lenguaje Fortran. La presencia o no del superguion determinaba la diferencia entre leer la velocidad promedio del cohete, o entregar datos no ponderados al sistema de navegación. El único motivo por el que el código defectuoso se expuso fue debido

a un fallo de hardware. Una antena que dejó de funcionar ocasionó que el software defectuoso tomara el control para guiar la nave a posición. El costo de este fallo superó los 18 millones de dólares de la época, equivalentes a 169 millones de dólares en la actualidad.

Ejemplo 1.5 Dispositivo médico Therac-25.

Entre 1985 y 1987, el equipo médico Therac-25 fue utilizado en Canadá y Estados Unidos para administrar radioterapias para tratamiento de cáncer. El Therac-25 causó seis accidentes en los que los pacientes recibieron sobredosis de radiación, incluyendo tres fallecimientos. El fallo del Therac-25 fue un agregado de situaciones: eliminación de hardware de seguridad para reemplazarlo por control por software, defectos de diseño en la interfaz de usuario, y manipulación errónea por parte de los técnicos operadores. Si el operador de turno pulsaba cierta secuencia de teclas en forma rápida para rectificar un ingreso erróneo de parámetros, se producía un “fallo de cursor”. A pesar de que la interfaz indicaba que los parámetros se habían rectificado, en realidad los pacientes recibieron hasta 125 veces más radiación que lo normal.

Ejemplo 1.6 Sistema antimisil Patriot.

El 25 de febrero de 1991, durante la Guerra del Golfo, un campamento militar de Estados Unidos localizado en Dharan, Arabia Saudí, fue alcanzado por un misil Scud iraquí causando el fallecimiento de 28 soldados y más de cien heridos. El sistema antimisil Patriot que protegía al campamento falló en detectar al misil. La causa fue un error de redondeo en el cálculo de la ruta del misil. Específicamente, el tiempo era medido en décimas de segundo utilizando el reloj interno del sistema. Para transformarlo a segundos, se debía multiplicar el tiempo del reloj interno por $1/10$. Como se utilizaba un registro de punto fijo de 24 bits para almacenar el resultado, se daba un redondeo de 0,000000095 segundos en la transformación. La batería del Patriot llevaba 100 horas seguidas funcionando por lo que el redondeo acumulado llegó a 0,34 segundos. Dado que los misiles Scud alcanzan una velocidad de 1,7 km/s, el desplazamiento del misil en 0,34 segundos fue de 573 metros, suficiente para que el Patriot no lo detectara. El problema de redondeo ya había sido detectado previamente, pero la solución temporal de reiniciar el sistema periódicamente fue ignorada por el personal del campamento. Además, el software actualizado con la corrección llegó el día anterior pero no fue instalado. La Guerra del Golfo terminó tres

días después de este incidente.

Ejemplo 1.7 Cohete Ariane 5.

El 4 de junio de 1996, la Agencia Espacial Europea realizó el lanzamiento del cohete Ariane 5. El cohete se autodestruyó 37 segundos después del despegue por un fallo del software de control. El problema fue un desbordamiento porque se intentó convertir un número decimal de 64 bits (punto flotante) que representaba la velocidad horizontal a un entero con signo de 16 bits, en una parte del código heredado del Ariane 4. El costo de este fallo fue 500 millones de euros.

Ejemplo 1.8 Mars Climate Orbiter.

En 1999, el módulo espacial Mars de la NASA se quemó tras acercarse demasiado a la superficie de Marte. La investigación posterior halló que la causa fue un error de conversión de unidades. El software de control en Tierra desarrollado por la empresa Lockheed Martin calculaba distancias en unidades inglesas (pulgadas), mientras que el software de a bordo, desarrollado por la NASA, utilizaba unidades métricas del Sistema Internacional (centímetros). El costo fue 320 millones de dólares.

Ejemplo 1.9 Cambio de milenio (Y2K).

El problema del cambio de milenio - también conocido como Y2K bajo el acrónimo en inglés, Year 2000 - fue causado por la representación incompleta de los años en las fechas utilizando solamente dos dígitos en lugar de cuatro dígitos. Potencialmente, todo software con este defecto de diseño, funcionaría incorrectamente al iniciar el nuevo milenio. Las empresas gastaron grandes sumas de dinero para rectificar el problema. La solución consistió en analizar todo el código que tenía un impacto por el cambio de milenio; planificar y realizar los cambios necesarios; y verificar la corrección de dichos cambios. El coste mundial de la corrección se estima en 300 mil millones de dólares. En el cambio de milenio si se dieron fallos en sistemas telefónicos, cajeros automáticos, parquímetros, y otros incidentes de poca gravedad.

Ejemplo 1.10 Apagón.

En 2003, el noreste y el medio oeste de Estados Unidos y la provincia cana-

diense de Ontario sufrieron un apagón generalizado. La causa principal fue un defecto de software en el sistema de alarma de la sala de control de una compañía eléctrica de Ohio, que provocó que los operadores no recibieran alertas de la sobrecarga del sistema eléctrico. Las alertas en cola averiaron el servidor primario, por lo que todas las aplicaciones, incluyendo el sistema de alarma, se transfirieron automáticamente al servidor de respaldo, que también falló. La falta de alarmas provocó que lo que debió ser un apagón local manejable se convirtiera en un colapso de toda la red por el cierre forzado de más de cien estaciones eléctricas. La sala de control también se quedó sin luz.

Ejemplo 1.11 Sistema de apoyo a la infancia.

En 2004, el gobierno británico migró a un nuevo sistema de gestión de la Agencia de Apoyo a la Infancia (CSA) - bajo el acrónimo en inglés, Child Support Agency. El contrato se adjudicó a la empresa de servicios informáticos Sistemas de Datos Electrónicos (EDS), bajo el acrónimo en inglés, Electronic Data Systems. El sistema defectuoso pagó en exceso a 1,9 millones de personas, pagó de menos a 700.000 personas, acumuló 7.000 millones de dólares en pagos de manutención no cobrados, retrasó 239.000 casos, y dejó sin procesar 36.000 casos. Un informe reveló que el sistema estaba “mal diseñado, mal probado, y mal implementado” y “tenía más de mil problemas notificados, de los cuales 400 no tenían solución conocida”, lo que provocaba “unos 3000 incidentes informáticos a la semana”. El sistema estaba presupuestado en 450 millones de libras, pero terminó costando 768 millones de libras, con el correspondiente perjuicio a los contribuyentes británicos. EDS también anunció una pérdida de 153 millones de dólares en sus resultados financieros por motivo de este proyecto.

Ejemplo 1.12 Sistema de gestión sanitaria.

En 2006 se inició la implantación de un nuevo sistema de gestión sanitaria en 670 centros de salud y hospitales de la red pública de Andalucía, España. La migración de datos de historias clínicas, citas, y tratamientos fue incompleta. El nuevo sistema presentó problemas de tiempos de respuesta. En palabras de un usuario: “Se tarda más tiempo en hacer una receta por ordenador que cinco ó seis a mano”. Por problemas técnicos, el sistema no pudo implantarse en varios municipios pequeños. El costo del sistema fue 60 millones de euros.

Ejemplo 1.13 Máquina tragamonedas.

En 2009, la Comisión de Juego de Ontario, Canadá se negó a pagar 42,9 millones de dólares canadienses a un usuario de una máquina tragamonedas. La señal de ganancia que recibió el usuario de la máquina tragamonedas de dos centavos en la que jugaba era errónea y no debería haber mostrado ninguna ganancia superior a 9,000 dólares canadienses. La causa de esta señal defectuosa durante el juego fue un error de software.

Ejemplo 1.14 Vehículos híbridos.

En 2010, Toyota retiró más de 400.000 vehículos híbridos del mercado por un problema de software que provocaba un retraso en el sistema anti-bloqueo de frenos. Se estima que, entre sustituciones y demandas, el defecto de software le costó a Toyota tres billones de dólares.

Ejemplo 1.15 Sistema de alerta de emergencias.

El 13 de enero de 2018, en el estado estadounidense de Hawái, se emitió erróneamente una alerta por televisión, radio, y teléfonos móviles a través del Sistema de Alerta de Emergencia. La alerta indicaba que había una amenaza de misil, pedía a los residentes buscar refugio, y afirmaba que no se trataba de un simulacro. Como consecuencia, la población entró en pánico pues a la fecha había tensiones entre Estados Unidos y Corea del Norte. La investigación respectiva concluyó que el incidente se debió a “insuficientes controles de gestión, mal diseño del software, y factores humanos”. Posterior al incidente, se implementó un comando de cancelación que puede ser activado segundos después del envío de una alerta errónea. Adicionalmente, se actualizó la interfaz de alertas de emergencia con una selección de falsa alarma, solucionando así otra deficiencia que dificultaba la anulación de una alerta enviada por error.

Ejemplo 1.16 Carrito de compras.

En la madrugada del 3 de mayo del 2021, varios usuarios ingresaron a la tienda en línea de la cadena de supermercados PlazaVea de Perú al enterarse que todos los productos estaban a un precio de 35 soles peruanos (9 dólares) y compraron tele-



Figura 1.10: Tienda en línea.
Fuente: Diario El Popular de Perú [39].

visores, celulares, lavadoras, entre otros artefactos. PlazaVea emitió un comunicado informando: “el día de hoy entre las 12am y 6.30am se presentó un error involuntario que impactó en los precios consignados en nuestro sistema” y que no entregaría los productos. Se trató de un fallo en la programación automática de precios. La Figura 1.10 muestra la interfaz de la tienda en línea.

Ejemplo 1.17 Sistema de turnos de revisión técnica vehicular.

En julio de 2022, el sistema de agendamiento de turnos para la revisión vehicular anual de la Agencia Metropolitana de Tránsito de Quito, Ecuador presentaba intermitencias en la disponibilidad debido a la gran cantidad de usuarios intentando acceder para obtener un turno. Los usuarios denunciaron que, una vez que lograban acceder, el sistema les ofrecía turnos para diciembre del año 2210. La Figura 1.11 muestra la interfaz del sistema.

Ejemplo 1.18 Notificación de Airbnb.

En la madrugada del 17 de agosto del 2022, la aplicación de Airbnb envió una notificación a los usuarios que tenían la aplicación instalada en sus teléfonos con sistema operativo Android que contenía el mensaje “test dev” y al hacer clic únicamente abría la página de inicio. Este tipo de notificaciones están destinadas para el entorno de pruebas donde los desarrolladores pueden probar los cambios realizados sin alterar el comportamiento de la aplicación para los usuarios. Utilizar accidentalmente el entorno



Figura 1.11: Fallo en asignación de turnos.
Fuente: Diario El Universo de Ecuador [40].

equivocado es un error aparentemente leve pero que puede tener graves consecuencias como modificación errónea de datos reales. La Figura 1.12 muestra la notificación.

1.5 Ética y calidad de software

Esta sección explora la relación entre la calidad de software y la ética en el contexto de la conducta profesional. Debido a la diversidad de aplicaciones de los productos software, la calidad - o falta de la misma - de los productos software puede tener un profundo impacto en el bienestar individual y en la armonía de la sociedad.

Definición 1.21 Ética Informática según ACM [60] . “La ética informática abarca todo comportamiento de los profesionales de la informática durante el diseño, desarrollo, construcción, y mantenimiento de artefactos informáticos que afecte a otras personas.”

Los códigos de ética y conducta profesional comprenden los valores y el comportamiento que se debe presentar durante la práctica profesional y en la toma de decisiones del profesional. Las infracciones éticas pueden ser actos de comisión, como por ejemplo: ocultar un trabajo inadecuado, revelar información confidencial, falsificación de información, tergiversación de las capacidades profesionales. Las infracciones éticas también pueden producirse por omisión, como por ejemplo: no revelar riesgos, no proporcionar información importante,

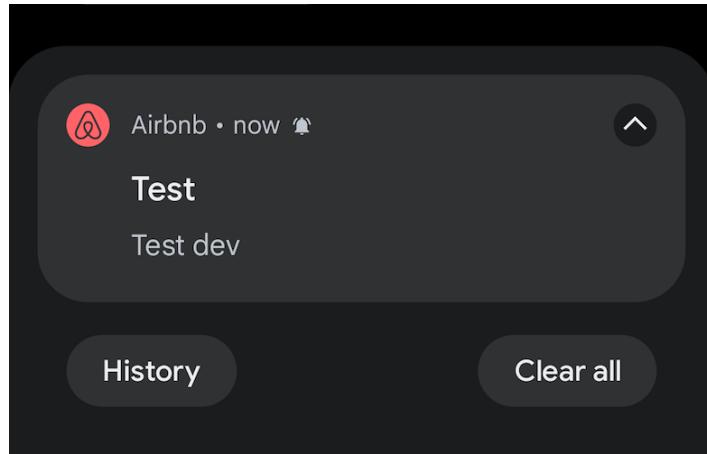


Figura 1.12: Fallo en la aplicación de Airbnb.

Fuente: Portal Airbnbbase [3].

no dar el crédito adecuado, no reconocer referencias, no representar los intereses del cliente. En este contexto, los códigos de ética y conducta profesional proporcionan una orientación frente a escenarios conflictivos. Una sólida ética supone que los profesionales de la informática comuniquen con precisión la información, las condiciones, y los resultados relacionados con la calidad de los productos software que desarrollan.

1.5.1 Código de ética y de conducta profesional de la informática

El código de ética y de conducta profesional de la informática se publicó por primera vez en 1999 y fue aprobado tanto por el Consejo de la ACM como por la Junta de Gobernadores de la Sociedad Informática del IEEE. Desde entonces, el código de ética ha sido adoptado por numerosas organizaciones a nivel mundial. La última versión a la fecha de publicación de la primera edición de este libro corresponde al año 2018 [60]. El código contiene principios éticos generales, principios sobre la responsabilidad profesional, principios de liderazgo profesional, y principios de cumplimiento del código que se listan a continuación.

1. Principios éticos generales.
 - (a) Contribuir a la sociedad y al bienestar humano, reconociendo que todas las personas son partes interesadas en la informática.
 - (b) Evitar causar daño.
 - (c) Ser honesto y digno de confianza.
 - (d) Ser justo y tomar acciones para no discriminar.
 - (e) Respetar el trabajo necesario para producir nuevas ideas, inventos, obras creativas, y artefactos informáticos.
 - (f) Respetar la privacidad.

(g) Respetar la confidencialidad.

2. Responsabilidades profesionales.

- (a) Esforzarse para lograr una alta calidad tanto en los procesos como en los productos del trabajo profesional.
- (b) Mantener altos niveles de competencia, conducta, y ética en la práctica profesional.
- (c) Conocer y respetar las reglas vigentes relativas al trabajo profesional.
- (d) Aceptar y proporcionar revisiones de pares constructivas, adecuadas, y profesionales.
- (e) Realizar evaluaciones completas y exhaustivas de los sistemas informáticos y sus impactos, incluyendo análisis de riesgos.
- (f) Realizar trabajos sólo en las áreas de su competencia profesional.
- (g) Fomentar la conciencia pública y la comprensión de la informática, las tecnologías relacionadas, y sus consecuencias.
- (h) Acceder a los recursos informáticos y de comunicación sólo cuando se cuente con autorización o cuando así lo obligue el bien público.
- (i) Diseñar e implementar sistemas que sean robustos y seguros para su uso.

3. Principios de liderazgo profesional.

- (a) Asegurarse que el bien público sea la preocupación principal durante el trabajo profesional de la informática.
- (b) Articular, fomentar la aceptación, y evaluar el cumplimiento de las responsabilidades sociales por parte de los miembros de la organización o equipo.
- (c) Gestionar el personal y los recursos para mejorar la calidad de la vida laboral.
- (d) Articular, aplicar, y apoyar políticas y procesos que reflejen los principios de este código.
- (e) Crear oportunidades para que los miembros de la organización o equipo crezcan como profesionales.
- (f) Tener cuidado al modificar o retirar sistemas.
- (g) Reconocer y tener especial cuidado de los sistemas que se integran en la infraestructura de la sociedad.

4. Cumplimiento del código.

- (a) Defender, promover, y respetar los principios de este código.
- (b) Considerar las violaciones a este código como incompatibles con la pertenencia a la profesión informática.

1.5.2 Toma de decisiones éticas

La competencia del profesional informático comienza con los conocimientos técnicos y con el reconocimiento del contexto en el que se despliega su trabajo. La competencia del profesional informático requiere también habilidad en la comunicación, en el análisis reflexivo, y en reconocer y afrontar retos éticos. Los sistemas informáticos son creaciones socio técnicas. Para tomar decisiones éticas es recomendable involucrar a varias personas con conocimientos técnicos y éticos. Así como es necesario utilizar pruebas técnicas para identificar y eliminar defectos en el producto de forma proactiva; así mismo, se deben utilizar pruebas éticas para identificar y eliminar defectos éticos. El código de ética y conducta profesional de la informática no define un algoritmo para resolver problemas éticos, sino que sirve de base para la toma de decisiones éticas por parte de los profesionales acorde a cada situación y contexto. La toma de decisiones éticas requiere la capacidad de anticipar los efectos, positivos o negativos, de un comportamiento. La toma de decisiones éticas debe ser pausada, consciente, utilizando atención, y energía cognitiva. Las cuestiones éticas pueden responderse mejor mediante una consideración reflexiva de los principios éticos fundamentales, entendiendo que el bien público es la consideración primordial. A manera de guía, se puede utilizar la técnica denominada CARE que propone los siguientes pasos para la toma de decisiones éticas[61]:

- Considerar las partes interesadas y los elementos éticos.
- Analizar el impacto.
- Revisar las responsabilidades y los enfoques alternativos.
- Evaluar los pros y los contras.

Considerar

Se debe considerar ampliamente quién será afectado colectivamente, en qué forma, en qué contexto, y las posibles alternativas, tomando en cuenta la complejidad del sistema. Para ello, se recomienda reflexionar dando respuestas a las siguientes preguntas:

- ¿Quiénes se verán afectados en su comportamiento o en sus procesos?
- ¿Quiénes se verán afectados en sus circunstancias o en su trabajo?
- ¿Quiénes se verán afectados en sus experiencias?
- ¿Qué alternativas plausibles pueden abordar las necesidades e impactos de las diferentes partes interesadas?
- ¿A quiénes se debe tomar en cuenta para plasmar estas alternativas?

Analizar

En el segundo paso se debe analizar las obligaciones y los derechos de todas las partes interesadas, dando respuesta a las siguientes preguntas:

- ¿Cómo las soluciones alternativas satisfacen obligaciones funcionales y obligaciones éticas?
- ¿Qué elementos del código de ética señalan los derechos de las partes interesadas?
- ¿Qué factores técnicos son los más relevantes?
- ¿Qué principios del código son los más relevantes?
- ¿Qué valores personales, institucionales, y legales deben tenerse en cuenta?

Revisar

El tercer paso es revisar las acciones potenciales que podrían marcar la diferencia, respondiendo a las siguientes preguntas:

- ¿Qué responsabilidades, prácticas o políticas son las más importantes para el análisis?
- ¿Existen alternativas creativas adicionales a las opciones que se han considerado hasta ahora?
- ¿Qué valores profesionales señalados por el código de ética se deben aplicar?

Reconsiderar una vez más los pasos previos de considerar y analizar, dando respuesta a las siguientes preguntas:

- ¿Qué suposiciones se han hecho acerca de las partes interesadas?
- ¿Cómo podrían utilizar el sistema los usuarios con discapacidades?
- ¿Cómo apoyan las alternativas planteadas a cumplir los valores profesionales del código?

Evaluar

El último paso es evaluar el trabajo realizado hasta el momento para la toma de la decisión ética, mediante las respuestas a las siguientes preguntas:

- ¿Cuál de las opciones consideradas es la mejor?
- ¿Cuáles son los pros y los contras de cada opción?

- ¿Existen alternativas creativas adicionales a las opciones que se han considerado hasta ahora?
- ¿Existen principios del código adicionales que sean relevantes para las deliberaciones sobre esta acción?

Finalmente, se debe seleccionar una alternativa que sea técnica y éticamente viable, manteniendo la supremacía del bien público, articulando claramente las compensaciones éticas de la alternativa seleccionada. Una decisión tomada en equipo reduce la necesidad de héroes morales. Un héroe moral es aquella persona que denuncia un hecho no ético una vez que ha sido cometido y lidera la indignación pública. Cada persona aporta diversas experiencias y enfoques a los dilemas éticos. Una vez tomada la decisión ética, se debe monitorear su cumplimiento.

Ejemplo 1.19 Accesibilidad en el desarrollo de software.

Este es un caso ficticio presentado en [73]:

La herramienta web AllTogether es utilizada por grupos comunitarios, organizaciones sin fines de lucro, y empresas de todo el mundo para gestionar proyectos. Con el aumento del trabajo remoto, ALLTogether ha ganado popularidad como plataforma para que los equipos controlen plazos, compartan documentos, realicen seguimiento de tareas y se comuniquen.

En la nueva versión de AllTogether , el equipo del producto incluyó un nuevo patrón de diseño para proporcionar acceso en línea a la función de edición mediante controles ocultos. Con la nueva función de edición en línea, al pasar el puntero por encima de un elemento, por ejemplo un evento en el calendario, aparece un ícono de un lápiz en el que el usuario puede hacer clic para que el elemento sea editable. Al alejar el puntero del elemento, el ícono desaparece. Este patrón de diseño de control oculto sustituyó a la anterior funcionalidad de edición proporcionada mediante un botón de texto de edición visible y un diálogo modal. El objetivo es proporcionar mayor comodidad y menos desorden.

El equipo de producto estaba en el proceso de implementar la nueva función cuando se dio cuenta de que había pasado por alto requisitos de accesibilidad. Dado que el ícono del lápiz sólo se muestra al pasar el puntero por encima, la función era inexistente para personas que no pueden apuntar y hacer clic, incluidas personas ciegas y con discapacidad motora. También las personas con discapacidades cognitivas pueden tener dificultades para encontrar controles ocultos, así como personas

con visión limitada que utilizan la lupa de pantalla y pueden tener dificultades para localizar y manejar controles dentro de una ventana gráfica ampliada, y personas con destreza motora limitada que pueden tener dificultades para interactuar con controles dinámicos de tipo mostrar/ocultar utilizando un dispositivo de puntero.

AllTogether cuenta con un Consejo Asesor sobre Diversidad, creado para ayudar a la empresa a evitar prácticas discriminatorias en el desarrollo tecnológico, incluida la discriminación por discapacidades. La política de accesibilidad de la empresa especifica como requisito tener conformidad con las directrices de accesibilidad para contenido Web (WCAG) - bajo el acrónimo en inglés, Web Content Accessibility Guidelines, para todos los productos web y móviles producidos por la empresa.

Sin embargo, el propietario del producto estaba siendo presionado para lanzar la nueva versión a pesar de los defectos de accesibilidad, con los directivos cuestionando: ¿Cuántas personas ciegas en verdad utilizan nuestro producto? La dirección de la empresa decidió que los clientes que utilizan AllTogether podrían proporcionar adaptaciones a sus empleados que se vieran afectados por los problemas de accesibilidad de la nueva función. Por tanto, el propietario del producto siguió adelante con el lanzamiento según lo previsto. También se pidió al área de atención al cliente crear una página web de accesibilidad y un correo electrónico de asistencia exclusivo para ayudar a los usuarios que encontraran barreras de accesibilidad en la nueva versión.

Los meses siguientes revelaron el impacto de la nueva función. Además de responder a las consultas de los usuarios de teclado que de repente no podían utilizar la función de edición, el área de atención al cliente se ocupó de ayudar a usuarios a superar otras barreras de accesibilidad causadas por los controles ocultos. Muchos usuarios tuvieron problemas para aprender y recordar el nuevo patrón. Al principio, pensaron que la función de edición se había eliminado por completo. Cuando se les indicaba la función, tenían problemas para recordar qué elementos eran editables y cuáles no. Varios usuarios reportaron dificultades para mostrar y activar la función de edición con un ratón debido a la falta de destreza manual. Y en el caso de los usuarios con lupa de pantalla, el ícono de edición aparecía a veces fuera de pantalla, y tenían dificultades para activar el ícono y acceder a la funcionalidad. Dados los costos de productividad y el riesgo de discriminación por discapacidad que implica utilizar la nueva versión con defectos de accesibilidad, algunos clientes decidieron volver a la versión anterior y esperar a que AllTogether corrigiera los defectos de accesibilidad en una versión futura.

En este caso, la dirección de la empresa decidió lanzar la aplicación web con la función inaccesible, a pesar de los costos y riesgos para la empresa, sus clientes y

los usuarios discapacitados. En el lado positivo, el propietario del producto respondió trabajando con el equipo de atención al cliente para documentar los defectos de accesibilidad y aumentar el apoyo a los usuarios afectados. Además, el equipo de atención al cliente respondió a las solicitudes de asistencia y trabajó con los clientes que optaron por volver a la versión anterior. Sin embargo, los usuarios discapacitados se encontraron con barreras de accesibilidad que les impedían realizar tareas con la función de edición. Los clientes que actualizaron a la nueva versión se arriesgaron a discriminar a sus empleados, clientes, y comunidad de discapacitados al utilizar una herramienta con defectos de accesibilidad. AllTogether también se arriesgó a causar daños a su reputación por producir un software defectuoso. Los costos de funcionamiento de la empresa aumentaron como consecuencia del incremento de las solicitudes de asistencia de los clientes y de la necesidad de mantener dos versiones del producto de software, costos que terminaron siendo superiores a los de abordar los requisitos de accesibilidad en el desarrollo de la nueva función.

Este caso demuestra cómo el código de ética y conducta profesional podría haber guiado las acciones y decisiones hacia el desarrollo de un mejor software a un menor coste, minimizando el daño a la empresa y a sus clientes y usuarios. Aunque su Consejo Asesor sobre Diversidad y su política de accesibilidad demuestran la intención de AllTogether de beneficiar a todas las personas, incluidas las personas con discapacidad, algunas de sus acciones y decisiones no fueron coherentes con esa intención. Si la dirección de la empresa se hubiera guiado por el principio 1.a (contribuir a la sociedad y al bienestar humano, reconociendo que todas las personas son partes interesadas en la informática), habría dado prioridad a la resolución de los defectos de accesibilidad sobre el cumplimiento del calendario de lanzamiento. El principio 1.b (evitar causar daño) nos lleva a seguir las mejores prácticas como forma de minimizar las consecuencias negativas. En el lado positivo, la empresa estableció una vía dedicada a informar de los problemas de accesibilidad, apoyando el principio 1.c (ser honesto y digno de confianza) al revelar los posibles problemas a los clientes y ofrecerles ayuda. Varios roles de la empresa contribuyeron a limitar innecesariamente la capacidad de trabajo de las personas con discapacidad al lanzar al mercado un producto de software defectuoso. Si la práctica profesional se hubiera guiado por el principio 1.d (ser justo y tomar acciones para no discriminar), todos los implicados habrían reorientado sus esfuerzos hacia una aplicación accesible. Si se hubiera diseñado y construido adecuadamente, una interfaz simplificada y organizada habría beneficiado a las personas con discapacidad y mejorado la usabilidad general.

1.6 Autoevaluación, retos, y aprendizaje autónomo

Preguntas

Pregunta 1.1 Seleccione la lección aprendida más importante de los casos Therac-25, Patriot, y Ariane 5.

- (a) Reusar software funcional y verificado no garantiza su confiabilidad.
- (b) Las pruebas de sistema por sí solas no son suficientes.
- (c) Se debe procurar que los diseños de la interfaz de usuario seán lo más sencillos posible.
- (d) Todas las anteriores.

Pregunta 1.2 ¿Cuál de las siguientes oraciones corresponde a la definición de defecto de software?

- (a) Acción humana que produce un resultado incorrecto.
- (b) Una deficiencia que hace que el software no cumpla con sus requisitos o especificaciones.
- (c) Evento en el que un software no ejecuta una función requerida.
- (d) Ninguna de las anteriores.

Pregunta 1.3 Suponga que Usted acaba de empezar a trabajar en una empresa tecnológica de tamaño medio y que ahora es el desarrollador jefe, es decir, Usted tiene responsabilidades tanto de gestión como de desarrollo. La empresa tiene una buena situación financiera y mantiene una posición competitiva moderada. En estos momentos, su empresa está entrando en un lucrativo mercado dominado por un único competidor. Cuando Usted intenta averiguar cómo importar datos desde el sitio web de este competidor, descubre una grave vulnerabilidad que permitiría acceder fácilmente a toda la información de los clientes del competidor. ¿Qué decisión ética toma Usted?

- (a) Comparte su descubrimiento con los desarrolladores a su cargo.
- (b) Reporta lo que ha descubierto al competidor.
- (c) Utiliza la vulnerabilidad para acceder a la información de los clientes del competidor.
- (d) No hace nada.

Ejercicios

Ejercicio 1.1 Analice el siguiente código escrito en Phyton que recibe un número entero que representa el número de día dentro del año, de tal manera que 1 es 1 de enero, 32 es 1 de febrero, y 365 es 31 de diciembre de un año no bisiesto. El método showday() devuelve el nombre del mes y el número de día dentro del mes. El método recibe un segundo parámetro que indica si el año es bisiesto, en cuyo caso febrero tiene 29 días y el año 366 días. El método debe lanzar la excepción ValueError si el número de día ingresado no es válido, es decir, menor que 1 o mayor que 366. Haga un recorrido manual del código utilizando los siguientes pares de datos de pruebas: 1 y Falso, 32 y Falso, 60 y Verdadero, 366 y Verdadero. ¿En qué líneas de código encuentra defectos? ¿Cuáles son las correcciones necesarias? Muestre las salidas del código defectuoso, corrija los defectos, muestre el código refactorizado, y las nuevas salidas obtenidas.

```
1
2 class Month:
3     pass
4
5 def showday( daynumber, isleapyear ):
6
7 # Muestra el mes y día dentro del mes del número de día del año.
8 # daynumber: número de día del año.
9 # isleapyear: Verdadero si el año es bisiesto.
10
11     months = [ 'Enero', 'Febrero', 'Marzo',
12                 'Abril', 'Mayo', 'Junio',
13                 'Julio', 'Agosto', 'Septiembre',
14                 'Octubre', 'Noviembre', 'Diciembre' ]
15
16     days = [ 31 for x in months ]
17
18     thirtylist = ( 'Abril', 'Junio',
19                     'Septiembre', 'Noviembre' )
20
21     for j in [ months.index(k) for k in thirtylist]:
22         days[j] = 30
23
24     # Considerar año bisiesto
25     days[months.index('Febrero')] = 28 + ((isleapyear and 1) or 0)
26
27     # daymap contiene 12 objetos Month, cada uno de los cuales tiene un par
```

```
28     nombre/días.  
29  
29     daymap = [ ]  
30  
31     for i in range(len(months)):  
32         newMonth = Month()  
33         newMonth.name = months[i]  
34         newMonth.days = days[i]  
35         daymap.append(newMonth)  
36  
37     if daynumber > 0:  
38         for el in daymap:  
39             if daynumber < el.days:  
40                 print (el.name, daynumber)  
41                 return  
42             daynumber = daynumber - el.days  
43     raise ValueError
```

Listing 1.1: Código Phyton para cálculo de mes y día.

Fuente: Encuentra el error. Un libro de programas incorrectos [13].

Ejercicio 1.2 Escriba un código en el lenguaje de programación de sus elección que reciba un número entero y devuelva el correspondiente número romano. Hay siete letras que se utilizan para representar números romanos: I=1, V=5, X=10, L=50, C=100, D=500, y M=1000. Los números romanos se escriben de mayor a menor de izquierda a derecha, a excepción de cuando el carácter de la izquierda es de menor valor que el de la derecha, en cuyo caso se resta. Compruebe su código con los siguientes pares de datos de prueba y salidas esperadas: 4 y IV, 90 y XC, 521 y DXI.

Problemas

Problema 1.1 El departamento de fabricación de la empresa de autos AcmeCars está desarrollando una nueva tecnología para habilitar una función específica que los usuarios llevan pidiendo desde hace tiempo en los vehículos eléctricos. Lamentablemente, al momento la tecnología solo puede funcionar provocando que los campos electromagnéticos de las celdas de energía aumenten más allá de los límites permitidos legalmente. Si se espera a lograr la solución dentro de los límites legales es muy probable que los competidores se adelanten en el mercado. Un alto directivo sugiere hacer cambios en el software del vehículo para que detecte cuándo se estén realizando pruebas reglamentarias y en ese caso se modifique el comportamiento del software para evitar el aumento del campo electromagnético y pasar así con éxito las pruebas. ¿Qué hace Usted en su calidad de dueño del producto? Para fundamentar

su decisión ética, explique cuáles valores del código de ética son relevantes para este caso.

Problema 1.2 Al revisar una especificación de software para cuya implementación se está considerando contratar a su empresa, su equipo descubre un defecto importante en dicha especificación que podría afectar a los usuarios del producto. Su empresa ha pasado el último año intentando negociar este lucrativo contrato y sus colegas directivos no quieren informar al cliente de este problema porque podría alargar aún más las negociaciones o incluso fracasarlas. ¿Qué hace Usted como directivo de la empresa? Para fundamentar su decisión ética, explique cuáles valores del código de ética son relevantes para este caso.

Proyectos

1. Proyecto “ERP para la Cámara Nacional de Comercio”

La Cámara Nacional de Comercio es una entidad gremial privada, sin fines de lucro, encargada de fomentar el desarrollo empresarial de su país. La Cámara Nacional de Comercio, que agrupa cientos de empresas, ha decidido financiar la implementación de un sistema de Planificación de Recursos Empresariales (ERP) - bajo el acrónimo en inglés, Enterprise Resource Planing - para uso de los socios de la Cámara. El Sistema ERP de la Cámara Nacional de Comercio debe cubrir las necesidades básicas de las empresas socias para la gestión automatizada de inventario, compras, ventas, clientes, contabilidad, y recursos humanos. Investigue las necesidades de una empresa pequeña que comercializa productos en un almacén físico con una sola bodega. En base a lo investigado, redacte en lenguaje natural un listado de cinco o más requisitos funcionales para cada uno de los siguientes módulos: inventario, compras, ventas, clientes, contabilidad, y recursos humanos.

2. Proyecto “Aplicación móvil para Monitoreo Prenatal y Primer Año de Vida del Recién Nacido”

Su emprendimiento de desarrollo de software quiere lanzar al mercado una aplicación móvil dirigida a madres primerizas. Investigue las necesidades de información de las madres primerizas respecto de los cuidados requeridos durante el embarazo, parto, y primer año de vida del bebé. La aplicación móvil debe permitir el control y seguimiento

del desarrollo del bebé mes a mes durante la gestación y durante el primer año de vida. En base a lo investigado, redacte en lenguaje natural un listado de cinco o más requisitos funcionales para cada uno de los siguientes módulos: embarazo, parto, y primer año de vida.

3. Proyecto “Curso de Inglés Online para Personas Ciegas o con Discapacidad Visual”

La Federación Nacional de Ciegos de su país le ha contratado para desarrollar un curso de inglés online para adultos con ceguera y diversos tipos de discapacidad visual. Investigue las necesidades de personas mayores a 18 años con ceguera o discapacidades visuales en relación al aprendizaje del idioma inglés por medio de un curso online. El curso online debe permitir al estudiante registrarse indicando sus requisitos de accesibilidad, autentificarse, dar una prueba de ubicación, iniciar el curso desde el nivel que le corresponda o niveles inferiores, dar una prueba de fin de cada nivel, descargar certificados de aprobación por nivel, y desbloquear el siguiente nivel. En base a lo investigado, redacte en lenguaje natural un listado de cinco requisitos funcionales para cada uno de los siguientes módulos: registro y autenticación, prueba de ubicación, niveles de estudio.

Soluciones seleccionadas

- Pregunta 1.1. Solución: (d). La opción (a) es correcta porque en el caso de Ariane 5, el problema fue un desbordamiento en una parte del código heredado del Ariane 4. La opción (b) es correcta porque en el caso del Patriot, el problema de redondeo había sido detectado en las pruebas, pero la solución temporal de reiniciar el sistema periódicamente fue ignorada por el personal y el software actualizado con la corrección permanente llegó el día anterior al incidente pero no fue instalado. La opción (c) es correcta porque en el caso del Therac-25, la falta de claridad en la interfaz hacía pensar al operador que la corrección de parámetros realizada había sido asimilada por el equipo.
- Pregunta 1.2. Solución: (b). La opción (b) es correcta porque corresponde a la definición de defecto según ISTQB. La opción (a) es incorrecta porque corresponde a la definición de error. La opción (c) es incorrecta porque corresponde a la definición de fallo.
- Pregunta 1.3. Solución: (b). La opción (b) es la decisión ética más apropiada. Las opciones (a) y (c) no garantizan el cumplimiento de los Principios 1.b (evitar causar daño), 1.c (ser honesto y digno de confianza), 1.f (respetar la privacidad), 1.g (respetar la confidencialidad), 2.b (mantener altos niveles de competencia, conducta, y ética en la práctica profesional.), y 2.h (acceder a los recursos informáticos y de comunicación sólo cuando se cuente con autorización o cuando así lo obligue el bien público).

- Ejercicio 1.1. Solución: Existen varios defectos. Entre ellos uno en la línea 41 y otro en la línea 43. El código en la línea 41 tiene una validación incompleta. Este defecto se manifiesta al ingresar un número mayor que 366. Tal y como está escrito, el código no lanza como debería la excepción ValueError. En su lugar, el código debería ser:

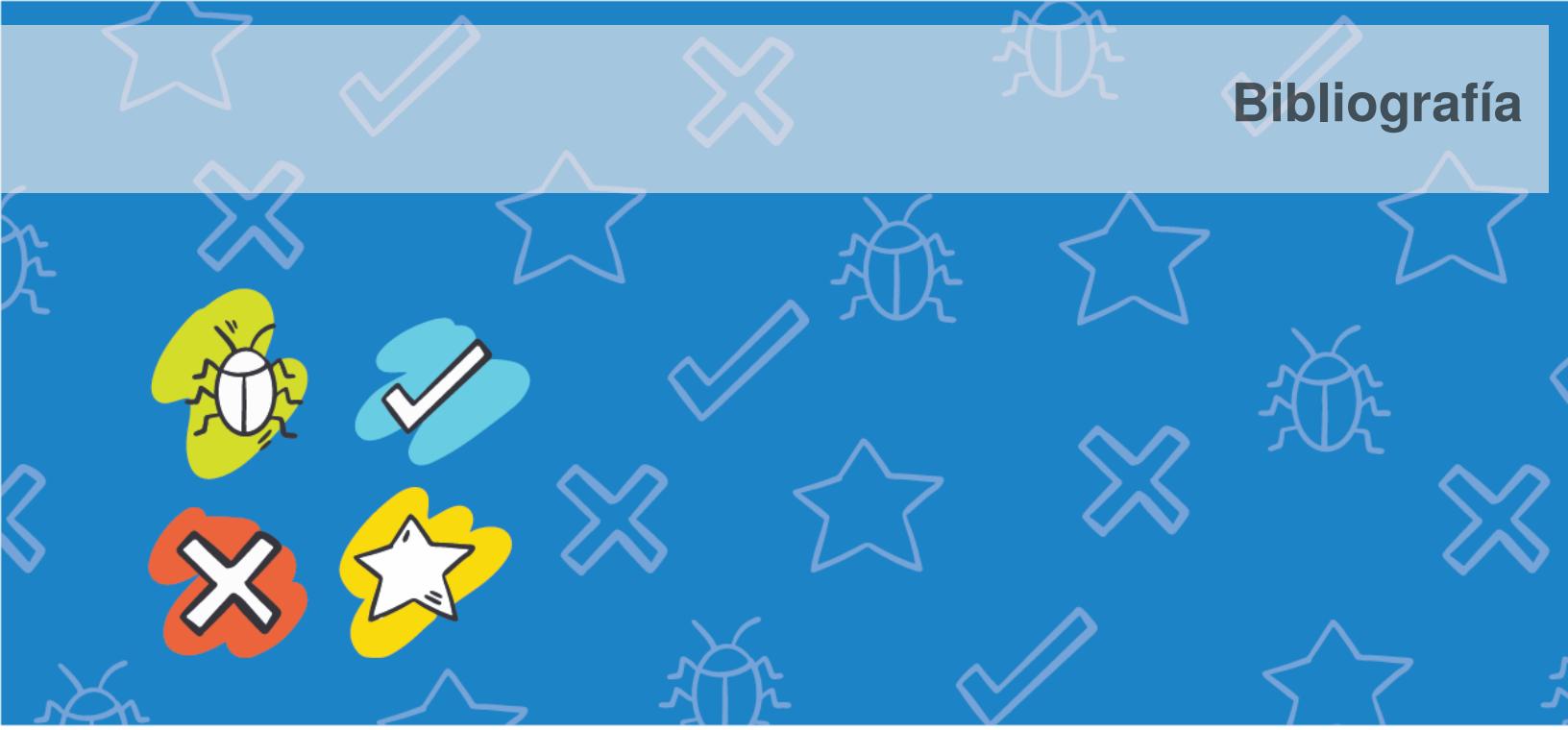
41 if (daynumber > 0 and daynumber < 367):

El código en la línea 43 utiliza el operador de comparación incorrecto. Este defecto se manifiesta al ingresar un número correspondiente al último día de un mes. Por ejemplo, tal y como está escrito, el código calcula que el día 31 es “0 de febrero” en lugar de “31 de enero”. En su lugar, el código debería ser:

43 if daynumber <= el.days:

Una vez corregido el código, las salidas esperadas para los pares de datos de pruebas son: Enero 1, Febrero 1, Febrero 29, y Diciembre 31.

- Problema 1.1. Solución: la decisión ética más apropiada como dueño del producto es oponerse a que se modifique el software para pasar con éxito las pruebas reglamentarias de los límites permitidos legalmente. Los principios del código relevantes para este caso son: 1.a (contribuir a la sociedad y al bienestar humano, reconociendo que todas las personas son partes interesadas en la informática), 1.b (evitar causar daño), 1.c (ser honesto y digno de confianza), 1.e (respetar el trabajo necesario para producir nuevas ideas, inventos, obras creativas, y artefactos informáticos), 2.a (esforzarse para lograr una alta calidad tanto en los procesos como en los productos del trabajo profesional), 2.b (mantener altos niveles de competencia, conducta, y ética en la práctica profesional), 2.i (diseñar e implementar sistemas que sean robustos y seguros para su uso), 3.a (asegurarse que el bien público sea la preocupación principal durante el trabajo profesional de la informática), 3.b (articular, fomentar la aceptación, y evaluar el cumplimiento de las responsabilidades sociales por parte de los miembros de la organización o equipo), 3.g (reconocer y tener especial cuidado de los sistemas que se integran en la infraestructura de la sociedad).

- 
- [1] Accesibilidad-Web. *La accesibilidad web en distintos países. Legislación sobre accesibilidad web en Estados Unidos, Europa, y España.* 2024. URL: <https://www.tuwebaccesible.es/accesibilidad-web-paises-2024/>. (último acceso: 08.06.2024).
 - [2] Patricia Acosta-Vargas y col. "Accessibility assessment in mobile applications for Android". En: *International Conference on Applied Human Factors and Ergonomics*. 2019, págs. 279-288.
 - [3] Airbnb. *Test Dev Notification*. 2022. URL: <https://airbnbbase.com/test-dev/>. (último acceso: 08.06.2024).
 - [4] Jost Amman. *The Book of Trades*. 1568.
 - [5] Apache. *Apache JMeter*. 2024. URL: <https://jmeter.apache.org/>. (último acceso: 08.06.2024).
 - [6] Applause. *The Essential Guide of Crowdtesting*. 2022.
 - [7] Arte-Historia-Egipto. *TT100 La Tumba de Rekhmire*. URL: https://artehistoriaeegipto.blogspot.com/2019/08/capitulo-126-tt100-la-tumba-de-rekhmire_20.html. (último acceso: 08.06.2024).
 - [8] James Bach. *Collecting and Interpreting Bug Metrics*. 2004. URL: <https://www.satisfice.com/download/bug-metrics-tutorial>. (último acceso: 08.06.2024).
 - [9] James Bach. *Rapid Software Testing Methodology*. 2022. URL: <https://www.satisfice.com/rapid-testing-methodology>. (último acceso: 08.06.2024).

- [10] James Bach y Michael Bolton. *A Rapid Introduction to Rapid Software Testing*. 2019. URL: <https://spring2019.stpcon.com/wp-content/uploads/2019/03/Bolton-Rapid-Intro-to-Rapid-SoftwareTesting.pdf>. (último acceso: 08.06.2024).
- [11] James Bach y Michael Bolton. *RST Appendices, v5.2.0*. 2021.
- [12] Jonathan Bach. "Session-Based Test Management". En: *Software Testing and Quality Engineering Magazine* 6 (2000), págs. 32-37.
- [13] Adam Barr. *Find the Bug. A Book of Incorrect Programs*. Addison-Wesley Professional, 2004. ISBN: 0321223918.
- [14] Onur Baskirt. *Whole Team Approach in Agile Testing*. 2018. URL: <https://www.testeryou.com/whole-team-approach-in-agile-testing>. (último acceso: 08.06.2024).
- [15] Graham Bath y Erik Van Veenendaal. *Improving the Test Process: Implementing Improvement and Change - A Study Guide for the ISTQB Expert Level Module*. 1^a ed. Addison-Wesley Professional, 2013.
- [16] Kent Beck. *Test Driven Development: By Example*. 1^a ed. Addison-Wesley Professional, 2002. ISBN: 9780321146533.
- [17] Ken Beck et al. *Agile Manifesto*. 2001. URL: <https://agilemanifesto.org/>. (último acceso: 08.06.2024).
- [18] Boris Beizer. *Software Testing Techniques*. Van Nost Reinhold, 1990.
- [19] Richard Bellairs. *What Is Static Analysis? Static Analysis Tools + Static Code Analyzers Overview*. 2023. URL: <https://www.perforce.com/blog/sca/what-static-analysis>. (último acceso: 08.06.2023).
- [20] Robert W. Bemer. "Checklist for Planning Software System Production". En: *Software Engineering Conference - NATO Science Committee*. 2015, págs. 165-180.
- [21] James Bennett Pritchard. *The Ancient Near East: An Anthology of Texts and Pictures*. Princeton University Press, 2010. ISBN: 9780691147260.
- [22] Jorge Benzaquen-De las Casas y Maximiliano Pérez-Cepeda. "El ISO 9001 y TQM en las Empresas de Ecuador". En: *Revista de Globalización, Competitividad y Gobernabilidad* 10.3 (2016), págs. 153-176.
- [23] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.
- [24] Michael Bolton. *Testing vs. Checking*. 2009. URL: <https://www.developsense.com/blog/2009/08/testing-vs-checking>. (último acceso: 08.06.2024).
- [25] Frederick Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.
- [26] Ilene Burnstein y Taratip Suwannasart. "Developing a Testing Maturity Model, Part II". En: *CrossTalk*. 1996.

- [27] Tania Calle-Jimenez y col. "Improving Usability with Think Aloud and Focus Group Methods. A Case Study: An Intelligent Police Patrolling System (I-Pat)". En: *Advances in Human Factors and Systems Interaction (AHFE)*. 2019, págs. 361-373.
- [28] Schmidt Calvo de Mora y col. *Gestión de la Calidad*. Ediciones Pirámide, 2021. ISBN: 8436845463.
- [29] Rubén Cano Montero. *Implementando BDD con Cucumber*. 2017. URL: <https://blog.softtek.com/es/implementando-bdd-con-cucumber>. (último acceso: 08.06.2024).
- [30] Luis Castillo-Salinas y col. "Evaluation of the implementation of a subset of ISO/IEC 29110 software implementation process in four teams of undergraduate students of Ecuador: An empirical software engineering experiment". En: *Computer Standards and Interfaces* 70 (2022).
- [31] Tyler Charboneau. *What is DevOps?* 2022. URL: <https://www.orangematter.solarwinds.com/2022/03/21/what-is-devops>. (último acceso: 08.06.2024).
- [32] Marco Tulio Cicerón. *Duodécima Filípica*. 43.
- [33] Mike Cohn. *Succeeding with Agile: Software Development using Scrum*. 1^a ed. Addison-Wesley Professional, 2009. ISBN: 0321579364.
- [34] Congreso-Argentino. *Ley 26.653 Accesibilidad de la Información en las Páginas Web*. 2010. URL: <https://observatorioplanificacion.cepal.org/sites/default/files/instrument/files/26653.pdf>. (último acceso: 08.06.2024).
- [35] Congreso-Nacional-del-Ecuador. *Ley del Sistema Ecuatoriano de la Calidad*. 2014.
- [36] Lisa Crispin y Janet Gregory. *Agile testing: A practical guide for testers and agile teams*. 1^a ed. Addison-Wesley Professional, 2009. ISBN: 9780321534460.
- [37] Philip Crosby. *Quality is Free*. McGraw-Hill, 1979.
- [38] W. Edwards Deming. *Out of the Crisis*. MIT Press, 1982.
- [39] Diario-El-Popular. *Plaza Vea se pronuncia sobre la caída de precios en su web y ciudadanos reclaman*. 2021. URL: <https://elpopular.pe/actualidad/2021/05/07/plaza-vea-compraron-televisores-35-soles-ahora-quieren-entregarles-productos-video-62455>. (último acceso: 08.06.2024).
- [40] Diario-El-Universo. *Página web para revisión técnica vehicular en Quito sigue con problemas*. 2022. URL: <https://www.eluniverso.com/noticias/ecuador/pagina-web-para-revision-tecnica-vehicular-en-quito-sigue-con-problemas-nota/>. (último acceso: 08.06.2024).
- [41] Edsger Dijkstra. "Go To Statement Considered Harmful". En: *Communications of the ACM* 11.3 (1968), págs. 147-148.

- [42] Edsger Dijkstra. "The Humble Programmer". En: *Communications of the ACM* 15.10 (1972), págs. 859-866.
- [43] R. Geoff Dromey. "A Model for Software Product Quality". En: *IEEE Transactions on Software Engineering* 2 (1995), págs. 146-162.
- [44] R. Geoff Dromey. "Cornering the Chimera". En: *IEEE Software* 1 (1996), págs. 33-43.
- [45] Thomas A. Edison. *Carta de Thomas Alva Edison a William Orton*. URL: <https://altenwald.org/2018/10/19/catalogacion-de-bichos/>. (último acceso: 08.06.2024).
- [46] Bill Elmendorf. *Automated Design of Program Test Libraries*. 1970.
- [47] Bill Elmendorf. *Evaluation of the Functional Testing of Control Programs*. 1967.
- [48] EuroSPI. *SPI Manifesto*. 2022. URL: <https://conference.eurospi.net/index.php/en/manifesto>. (último acceso: 08.06.2024).
- [49] Michael E. Fagan. "Design and Code Inspections to Reduce Errors in Program Development". En: *IBM Systems Journal* 15.3 (1976), págs. 182-211.
- [50] Armand Feigenbaum. *Total Quality Control: Engineering and Management*. McGraw-Hill, 1961.
- [51] Mark Fewster y Dorothy Graham. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley, 1999.
- [52] Martin Fowler. *Technical Debt*. 2019. URL: <https://martinfowler.com/bliki/TechnicalDebt.html>. (último acceso: 08.06.2024).
- [53] Fundación-Saldarriaga-Concha. *Norma técnica colombiana 5854*. 2011. URL: <http://ntc5854.accesibilidadweb.org/>. (último acceso: 08.06.2024).
- [54] David Gelperin y Bill Hetzel. "The Growth of Software Testing". En: *Communications of the ACM* 31.6 (1988), págs. 687-695.
- [55] Tom Gilb. "Laws of Unreliability". En: *Datamation* 21.3 (1975), págs. 81-81.
- [56] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley, 1988.
- [57] Tom Gilb. *Software Metrics*. Winthrop Publishers, 1977.
- [58] Tom Gilb y Dorothy Graham. *Software Inspection*. 1^a ed. Addison Wesley, 1993.
- [59] GoogleDevelopers. *PageSpeed Insights*. 2024. URL: <https://pagespeed.web.dev/>. (último acceso: 08.06.2024).
- [60] Don Gotterbarn y Marty J. Wolf. *ACM Code of Ethics and Professional Conduct*. 2018.
- [61] Don Gotterbarn y Marty J. Wolf. *Leveraging the ACM Code Of Ethics against Ethical Snake Oil and Dodgy Development*. 2020.
- [62] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. 2^a ed. Prentice Hall, 1992. ISBN: 9780137203840.

- [63] Robert B. Grady. *Successful Software Process Improvement*. Prentice Hall, 1997. ISBN: 9780136266235.
- [64] Robert B. Grady y Deborah L. Caswell. *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall, 1987. ISBN: 9780138218447.
- [65] Dorothy Graham. *Computer-aided Software Testing: The CAST Report*. 1991.
- [66] Dorothy Graham y Mark Fewster. *Experiences of Test Automation: Case Studies of Software Test Automation*. 1^a ed. Addison-Wesley Professional, 2012. ISBN: 0321754069.
- [67] Dorothy Graham, Erik Van Veenendaal y Rex Black. *Foundations of Software Testing: ISTQB Certification*. 1^a ed. Cengage Learning Business Press, 2012. ISBN: 1844803554.
- [68] Janeth Gregory y Lisa Crispin. *More Agile Testing: Learning Journeys for the Whole Team*. Addison-Wesley, 2014. ISBN: 0321967054.
- [69] George Guimarães. *What is a Linter and why your Team should use it?* 2020. URL: <https://sourcelevel.io/blog/what-is-a-linter-and-why-your-team-should-use-it>. (último acceso: 08.06.2024).
- [70] Elisabeth Hendrickson, Lyndsay James y Emery Dale. *Test Heuristics Cheat Sheet: Data Type Attacks & Web Tests*. 2006.
- [71] Bill Hetzel. *The Complete Guide to Software Testing*. QED Information Sciences, 1988.
- [72] William Hetzel. *Program Test Methods*. Englewood Cliffs, 1973. ISBN: 013729624X.
- [73] Sarah Horton. *Case: Accessibility in Software Development*. 2022. URL: <https://ethics.acm.org/code-of-ethics/using-the-code/case-accessibility-in-software-development>. (último acceso: 08.06.2024).
- [74] Watts Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.
- [75] Watts Humphrey. *PSP: A Self-Improvement Process for Software Engineers*. 1^a ed. Addison-Wesley, 2005.
- [76] Watts Humphrey. *TSP Leading a Development Team*. 1^a ed. Addison-Wesley, 2006.
- [77] IEEE. *Norma Internacional IEEE Std. 730-1980 IEEE Standard for Software Quality Assurance Plans*. 1980.
- [78] IEEE. *Norma Internacional IEEE Std. 730:2014 Standard for Software Quality Assurance Processes*. 2014.
- [79] IEEE. *Norma Internacional IEEE Std. 829-2008 Software Reviews and Audits*. 2008.
- [80] ISACA. *CMMI Adoption Guidance*. 2023.
- [81] ISACA. *CMMI V3.0 - Overview*. 2023.
- [82] Kaoru Ishikawa. *What is Total Quality Control? The Japanese Way*. 1981.

- [83] ISO. *Norma Internacional ISO 9000-3:1991 Quality Management and Quality Assurance Standards — Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*. 1991.
- [84] ISO. *Norma Internacional ISO 9000:2015 Quality Management systems — Fundamentals and Vocabulary*. 2015.
- [85] ISO. *Norma Internacional ISO 9001:2015 Quality Management Systems — Requirements*. 2015.
- [86] ISO/IEC. *Norma Internacional ISO/IEC 12207:2008 Systems and Software Engineering — Software Life Cycle Processes*. 2008.
- [87] ISO/IEC. *Norma Internacional ISO/IEC 14598-5:1998 Information Technology — Software Product Evaluation — Part 5: Process for Evaluators*. 1998.
- [88] ISO/IEC. *Norma Internacional ISO/IEC 25000:2014 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*. 2014.
- [89] ISO/IEC. *Norma Internacional ISO/IEC 25001:2014 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Planning and Management*. 2014.
- [90] ISO/IEC. *Norma Internacional ISO/IEC 25010:2023 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Product Quality Model*. 2023.
- [91] ISO/IEC. *Norma Internacional ISO/IEC 25012:2008 Software Engineering — Software Product Quality Requirements and Evaluation (SQuaRE) — Data Quality Model*. 2008.
- [92] ISO/IEC. *Norma Internacional ISO/IEC 25019:2023 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model*. 2023.
- [93] ISO/IEC. *Norma Internacional ISO/IEC 25020:2019 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality Measurement Framework*. 2019.
- [94] ISO/IEC. *Norma Internacional ISO/IEC 25022:2016 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Measurement of Quality in Use*. 2016.
- [95] ISO/IEC. *Norma Internacional ISO/IEC 25023:2016 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Measurement of System and Software Product Quality*. 2016.
- [96] ISO/IEC. *Norma Internacional ISO/IEC 25030:2019 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality requirements framework*. 2019.

- [97] ISO/IEC. *Norma Internacional ISO/IEC 25040:2011 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Evaluation process.* 2011.
- [98] ISO/IEC. *Norma Internacional ISO/IEC 25041:2012 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Evaluation Guide for Developers, Acquirers and Independent Evaluators.* 2012.
- [99] ISO/IEC. *Norma Internacional ISO/IEC 25065:2019 Systems and Software Engineering — Software Product Quality Requirements and Evaluation (SQuaRE) — Common Industry Format (CIF) for Usability: User Requirements Specification.* 2019.
- [100] ISO/IEC. *Norma Internacional ISO/IEC 29110-2-1:2015 Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 2-1: Framework and Taxonomy.* 2015.
- [101] ISO/IEC. *Norma Internacional ISO/IEC 29110-3-3:2016 Systems and software engineering — Lifecycle profiles for Very Small Enterprises (VSEs) — Part 3-3: Certification Requirements for Conformity Assessments of VSE Profiles using Process Assessment and Maturity Models.* 2016.
- [102] ISO/IEC. *Norma Internacional ISO/IEC 33001:2015 Information Technology — Process Assessment — Concepts and Terminology.* 2015.
- [103] ISO/IEC. *Norma Internacional ISO/IEC 33002:2015 Information Technology — Process Assessment — Requirements for Performing Process Assessment.* 2015.
- [104] ISO/IEC. *Norma Internacional ISO/IEC 33003:2015 Information Technology — Process Assessment — Requirements for Process Measurement Frameworks.* 2015.
- [105] ISO/IEC. *Norma Internacional ISO/IEC 33004:2015 Information Technology — Process Assessment — Requirements for Process Reference, Process Assessment and Maturity Models.* 2015.
- [106] ISO/IEC. *Norma Internacional ISO/IEC 33020:2019 Information technology — Process Assessment — Process Measurement Framework for Assessment of Process Capability.* 2019.
- [107] ISO/IEC. *Norma Internacional ISO/IEC 33063:2015 Information Technology — Process Assessment — Process Assessment Model for Software Testing.* 2015.
- [108] ISO/IEC. *Norma Internacional ISO/IEC 40500:2012 Information Technology — W3C Web Content Accessibility Guidelines (WCAG) 2.0.* 2012.
- [109] ISO/IEC. *Norma Internacional ISO/IEC 9126-1:2001 Software Engineering — Product Quality — Part 1: Quality model.* 2001.
- [110] ISO/IEC. *Norma Internacional ISO/IEC 9126:1991 Software Engineering— Product Quality.* 1991.

- [111] ISO/IEC. *Norma Internacional ISO/IEC DTS 33010 Information Technology — Process Assessment — Guidance for Performing Process Assessments*. 2022.
- [112] ISO/IEC. *Norma Internacional ISO/IEC TR 29110-4:2018 Systems and Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 4-1: Software Engineering - Profile Specifications: Generic Profile Group*. 2018.
- [113] ISO/IEC. *Norma Internacional ISO/IEC TR 33014:2013 Information Technology — Process Assessment — Guide for Process Improvement*. 2013.
- [114] ISO/IEC. *Norma Internacional ISO/IEC TS 25011:2017 Information Technology — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Service Quality Models*. 2017.
- [115] ISO/IEC. *Norma Internacional ISO/IEC TS 33030:2017 Information Technology — Process Assessment — An Exemplar Documented Assessment Process*. 2017.
- [116] ISO/IEC. *Norma Internacional Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality Model Overview and Usage*. 2024.
- [117] ISO/IEC. *Norma ISO/IEC 20246:2017 Software and Systems Engineering — Work Product Reviews*. 2017.
- [118] ISO/IEC. *Reporte Técnico ISO/IEC TR 19759:2015 Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)*. 2015.
- [119] ISO/IEC. *Reporte Técnico ISO/IEC TR 25021:2012 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality Measure Elements*. 2012.
- [120] ISO/IEC. *Reporte Técnico ISO/IEC TR 29110-1:2016 Systems and Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 1: Overview*. 2016.
- [121] ISO/IEC. *Reporte Técnico ISO/IEC TR 29110-3-1:2020 Systems and Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 3-1: Process Assessment Guidelines*. 2020.
- [122] ISO/IEC. *Reporte Técnico ISO/IEC TR 29110-5-1-2:2024 Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile*. 2024.
- [123] ISO/IEC. *Reporte Técnico ISO/IEC TR 29119-11:2020 Software and Systems Engineering — Software Testing — Part 11: Guidelines on the Testing of AI-based Systems*. 2020.
- [124] ISO/IEC. *Reporte Técnico ISO/IEC TR 29119-6:2021 Software and Systems Engineering — Software Testing — Part 6: Guidelines for the Use of ISO/IEC/IEEE 29119 (all parts) in Agile Projects*. 2021.

- [125] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 12207:2017 Systems and Software engineering - Software Life Cycle Processes*. 2017.
- [126] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 24765:2017 Systems and Software engineering — Vocabulary*. 2017.
- [127] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 29119-1:2022 Software and Systems Engineering — Software Testing — Part 1: General Concepts*. 2022.
- [128] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 29119-2:2021 Software and Systems Engineering — Software Testing — Part 2: Test Processes*. 2021.
- [129] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 29119-3:2021 Software and Systems Engineering — Software Testing — Part 3: Test Documentation*. 2021.
- [130] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 29119-4:2021 Software and Systems Engineering — Software Testing — Part 4: Test Techniques*. 2021.
- [131] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 29119-5:2016 Software and Systems Engineering — Software Testing — Part 5: Keyword-Driven Testing*. 2016.
- [132] ISO/IEC/IEEE. *Norma Internacional ISO/IEC/IEEE 90003:2018 Software Engineering — Guidelines for the Application of ISO 9001:2015 to Computer Software*. 2018.
- [133] ISTQB. *Certified Tester Foundation Level Syllabus V4.0*. 2023.
- [134] ISTQB. *Foundation Level Specialist Syllabus Performance Testing*. 2018.
- [135] ISTQB. *Standard Glossary of Terms used in Software Testing Version 3.2*. 2018.
- [136] Hugh Jack. *Automating Manufacturing Systems with PLCs*. 7^a ed. Lulu.com, 2009. ISBN: 0557344255.
- [137] Paul Jorgensen. *Software Testing: A Craftsman's Approach*. 1^a ed. CRC Press, 1995. ISBN: 084937345X.
- [138] Paul Jorgensen. *Software Testing: A Craftsman's Approach*. 5^a ed. Auerbach Publications, 2022.
- [139] Joseph Juran. *Juran on Planning for Quality*. The Free Press, 1988.
- [140] Joseph Juran. *Quality Control Handbook*. McGraw-Hill, 1951.
- [141] Cem Kaner. *Defining Exploratory Testing*. 2008. URL: <https://kaner.com/?p=46>. (último acceso: 08.06.2024).
- [142] Cem Kaner. *Schools of Software Testing*. 2006. URL: <https://kaner.com/?p=15>. (último acceso: 08.06.2024).
- [143] Cem Kaner, James Bach y Bret Pettichord. *Lessons Learned in Software Testing: A Context-Driven Approach*. 1^a ed. Wiley Computer Publishing, 2001. ISBN: 9780471081128.
- [144] Cem Kaner, Jack Falk y Hung Q. Nguyen. *Testing Computer Software*. Wiley, 1999.

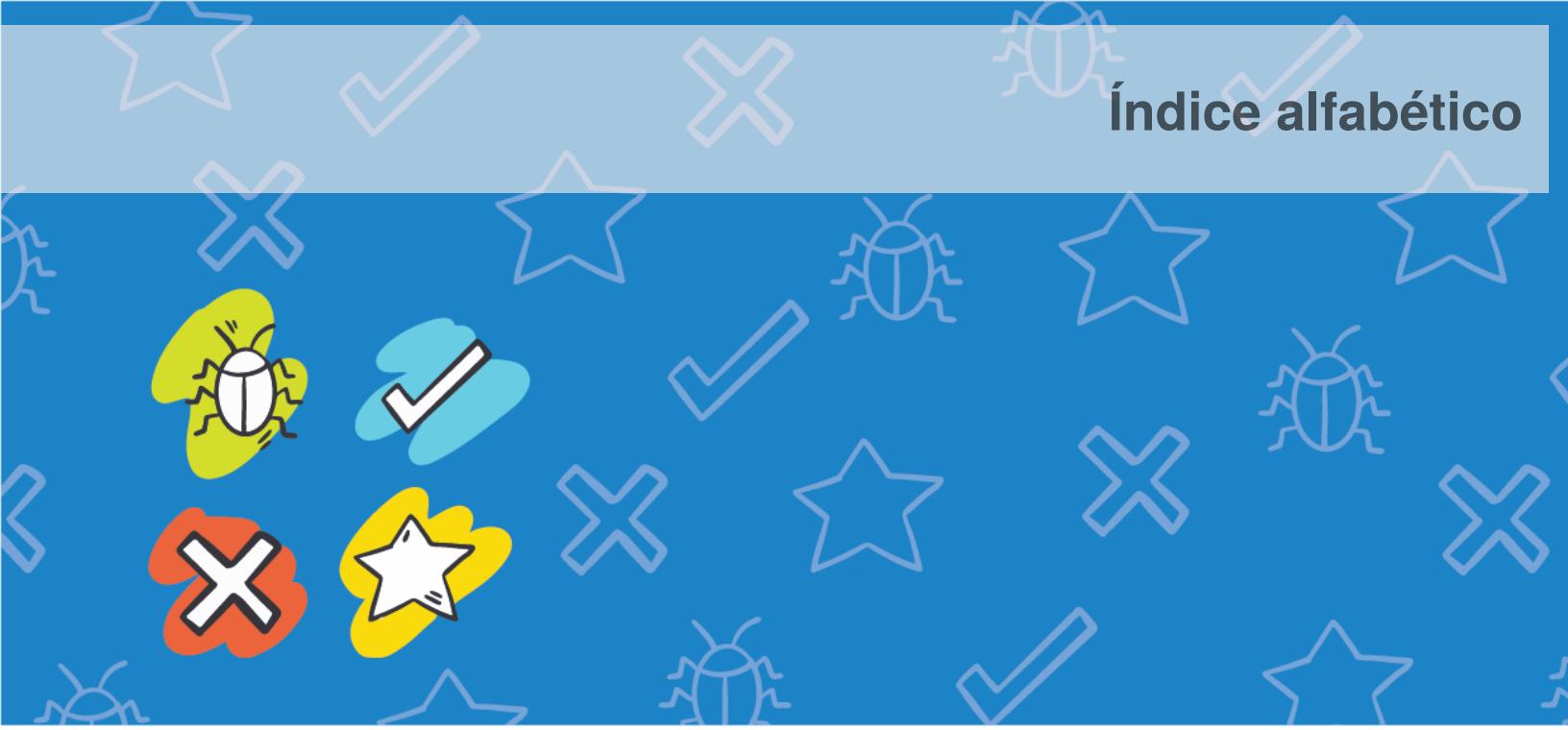
- [145] Richard Lawrence y Paul Rayner. *Behavior-Driven Development with Cucumber: Better Collaboration for Better Software*. Addison-Wesley Professional, 2019.
- [146] Herbert Leeds y Gerald M. Weinberg. *Computer Programming Fundamentals*. McGraw-Hill, 1961.
- [147] Gärtner Markus. *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley, 2012.
- [148] James Martin. *An Information Systems Manifesto*. Prentice-Hall, 1984.
- [149] Juan Carlos Mayo Alegre, Néstor Alberto Loredo Carballo y Saadia Noemí Reyes Benites. “En torno al concepto de calidad. Reflexiones para su definición”. En: *Retos de la Dirección* 9.2 (2015), págs. 49-67.
- [150] Ann McArthur. *Continuous Testing Best Practices*. 2023. URL: <https://devcycle.com/blog/continuous-testing-best-practices>. (último acceso: 08.06.2024).
- [151] Thomas J. McCabe. “A Complexity Measure”. En: *Second International Conference on Software Engineering (ICSE)*. Vol. 2. 4. 1976, págs. 308-320.
- [152] Jim A. McCall, Paul K. Richards y Gene F. Walters. *Factors in Software Quality*. 1977.
- [153] Daniel D. McCracken. *Digital Computer Programming*. John Wiley y Sons, Inc., 1957.
- [154] Luigi F. Menabrea. “Sketch of the Analytical Engine invented by Charles Babbage”. En: *Scientific Memoirs* (1843), págs. 666-731.
- [155] Microsoft. *Lint Python code in Visual Studio*. 2024. URL: <https://docs.microsoft.com/en-us/visualstudio/python/linting-python-code>. (último acceso: 08.06.2024).
- [156] Jamie L Mitchell y Rex Black. *Advanced Software Testing - Vol. 3. 2^a ed.* Rocky Nook, 2015. ISBN: 9781933952390.
- [157] Mockaroo. *Mockaroo - New Schema*. 2024. URL: <https://www.mockaroo.com/schemas/new>. (último acceso: 08.06.2024).
- [158] Glenford Myers. *Software Reliability: Principles and Practices*. Wiley, 1976.
- [159] Glenford Myers. *The Art of Software Testing*. Wiley, 1979.
- [160] Gerard O'Regard. *Concise Guide to Software Engineering. From Fundamentals to Application Methods*. Springer, 2017. ISBN: 9783319577494.
- [161] Sheilagh Ogilvie. “The Economics of Guilds”. En: *Journal of Economic Perspectives* 28.4 (2014), págs. 169-192.
- [162] Jorge Pérez-Medina y col. “Usability Study of a Web-Based Platform for Home Motor Rehabilitation”. En: *IEEE Access* 7 (2019), págs. 7943-7947.

- [163] Bret Pettichord. "The Four Schools of Software Testing". En: *Software Testing Conference (STARWEST)*. 2004.
- [164] Henrry Pilco y col. "An agile approach to improve the usability of a physical telerehabilitation platform". En: *Journal of Applied Sciences* 9.3 (2019).
- [165] Javier Pino Herrera. *Introducción a las pruebas unitarias con JUnit*. 2020.
- [166] PowerMapper. *SortSite Desktop One-Click Web Testing*. 2024. URL: <https://www.powermapper.com/products/sortsite/>. (último acceso: 08.06.2024).
- [167] Roger Pressman y Bruce Maxim. *Software Engineering: A Practitioner's Approach*. 9^a ed. McGraw-Hill Education, 2019. ISBN: 1260548007.
- [168] Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.
- [169] Stuart Reid. *ISO/IEC/IEEE 29119 Software Testing Standards*. STA, 2017. ISBN: 9788994711041.
- [170] Paul Rook. "Controlling Software Projects". En: *Software Engineering Journal* 1.1 (1986), págs. 7-16.
- [171] Raymond J. Rubey y R. Dean Hartwick. "Quantitative Measurement of Program Quality". En: *23rd ACM National Conference*. 1968, págs. 671-677.
- [172] Sandra Sanchez-Gordon. "Aristóteles, Dialéctica Hegeliana y Evolución de la Ingeniería de Software". En: *International Conference in Software Engineering and New Technologies in Systems Engineering (CISOFT)*. 2012.
- [173] Sandra Sanchez-Gordon y Viera-Bautista Daniel. "Mapping between CMMI-DEV v1.3 and ISO/IEC 90003:2014". En: *International Conference on Information Systems and Software Technologies (ICI2ST)*. 2019, págs. 134-140.
- [174] Sandra Sanchez-Gordon y Sergio Luján-Mora. "A Method for Accessibility Testing of Web Applications in Agile Environments". En: *7th World Congress for Software Quality (WCSQ)*. 2017, págs. 1-8.
- [175] Mary Sánchez-Gordón, Sandra Sanchez-Gordon y Ricardo Colomo-Palacios. "Vote Item: Is "Compassionate Software Development" a Topic Worth Researching?" En: *IEEE/ACM 15th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 2022, págs. 107-108.
- [176] Mary Sánchez-Gordón y col. "Integration of Accessibility Design Patterns with the Software Implementation Process of ISO/IEC 29110". En: *Journal of Software: Evolution and Process* 31.1 (2019).
- [177] Mary Sánchez-Gordón y col. "Learning Tool for the ISO/IEC 29110 Standard: Understanding the Project Management of Basic Profile". En: *International Conference on Software Process Improvement and Capability Determination (SPICE)*. 2016, págs. 270-283.

- [178] Selenium. *WebDriver*. 2024. URL: <https://www.selenium.dev/documentation/webdriver/>. (último acceso: 08.06.2024).
- [179] Selenium-Easy. *Execute Appium Tests on Real Device - Android [Native App]*. 2022. URL: <https://www.seleniumeasy.com/appium-tutorials/execute-appium-tests-native-app-real-device-android>. (último acceso: 08.06.2024).
- [180] Servicio-Ecuatoriano-de-Normalización. *Reglamento técnico INEN RTE 288*. 2011. URL: <https://www.normalizacion.gob.ec/buzon/reglamentos/RTE-288.pdf>. (último acceso: 08.06.2024).
- [181] Walter Shewhart. *Economic Control of Quality of Manufactured Products*. Macmillan y Co. Ltd., 1931.
- [182] Walter Shewhart. *Statistical Method from the Viewpoint of Quality Control*. Lancaster Press, 1937.
- [183] SoapUI. *Sample SOAP Project in SoapUI*. 2022. URL: <https://www.soapui.org/resources/tutorials/soap-sample-project/>. (último acceso: 08.06.2024).
- [184] Ian Sommerville. *Software Engineering*. 10^a ed. Pearson Education, 2016.
- [185] Sonarcloud. *Covid Project*. 2020. URL: https://sonarcloud.io/summary/new_code?id=Covid. (último acceso: 08.06.2024).
- [186] Peter Stabel. "Guilds in Late Medieval Flanders: Myths and Realities of Guild Life in an Export-oriented Environment". En: *Journal of Medieval History* 30.2 (2004), págs. 187-212.
- [187] StudySection. *Statement and Decision Coverage in White Box Testing*. 2022. URL: <https://studysection.com/blog/statement-and-decision-coverage-in-white-box-testing/>. (último acceso: 08.06.2024).
- [188] Genichi Taguchi. *Introduction to Quality Engineering: Designing Quality into Products and Processes*. Asian Productivity Organisation, 1986.
- [189] TMMI-Foundation. *TMMI in the Agile world Version 1.4*. 2020.
- [190] Tricentis. *Types of Software Testing Tools*. 2022. URL: <https://www.tricentis.com/learn/software-testing-tools>. (último acceso: 08.06.2024).
- [191] Alan Turing. *Checking a Large Routine*. 1949.
- [192] U.S.-Naval-Historical-Center-Online-Library. *Bitácora del Computador Mark II*. URL: <https://commons.wikimedia.org/w/index.php?curid=165211>. (último acceso: 08.06.2024).
- [193] University-of-Alberta. *Reviews and Metrics for Software Improvements*. 2020.

- [194] Tom Verhoeff. *Review Checklist for Software Requirements Document*. 2001. URL: <https://www.win.tue.nl/~wstomv/edu/sep/checklists/review/srd.html>. (Último acceso: 08.06.2024).
- [195] WebAIM. *WAVE Web Accessibility Evaluation Tools*. 2024. URL: <https://wave.webaim.org/>. (Último acceso: 08.06.2024).
- [196] Gerald M. Weinberg. *Perfect Software And Other Illusions About Testing*. Weinberg & Weinberg, 2010.
- [197] Gerald M. Weinberg. *The Psychology of Computer Programming*. Van Nostrand Reinhold Company, 1971.
- [198] World-History-Encyclopedia. *Code of Hammurabi – Detail*. URL: <https://www.worldhistory.org/image/14340/code-of-hammurabi---detail>. (Último acceso: 08.06.2024).

Esta página está intencionalmente vacía.



ACM, 25, 37
ACM, código de ética de, 54
agilismo, principios del, 244
agilismo, valores del, 244
análisis de valores frontera, 202
análisis de valores límites, 202
análisis estático, 259
API, 268
arnés, 178
ATDD, 249
auditoría de seguridad, 189
author check, 192

Babbage, máquina analítica de, 21
Bach, James, 30, 32, 33, 231, 234
Bach, Jonathan, 33
back-end, 255
Baker, Charles L., 23
banderas de características, 255
BDD, 248
Beck, Kent, 33
Beizer, Boris, 30, 217
Bell Labs, 15
Bell, Alexander Graham, 37, 45
Bemer, Robert, 24
bicho, 45

Black, Rex, 31
Boehm, Barry, 28
Bolton, Michael, 33, 34, 234
Brooks, Frederick, 26
Buckley, Fletcher, 37
buddy check, 192
bug, 45, 261
burndown chart, 138

calidad, 18, 19
calidad en uso, 90
calidad total, 15
calidad total, control de la, 18
calidad total, gestión de la, 16
calidad, asistencia a la, 237
calidad, criterio de, 241
calidad, garantía de, 237
calidad, modelo de, 67
Caswell, Deborah, 29, 69
CD, 252
chequeo de escritorio por pares, 192
chequeo de amigo, 192
chequeo de autor, 192
CI, 179
CI/CD, 253, 269
CI/CD/CD, 253

- ciclo Deming, 112
ciclo PDCA, 16, 17, 20, 112
ciclo Shewhart, 112
clase de equivalencia, 200
CMM, 30
CMMI, 109, 116, 233
CMMI, área de práctica, 118
cobertura de rama, 214
cobertura de sentencia, 213
COCOMO, 28
code smell, 260
Cohn, Mike, 34
complejidad ciclomática, 27
controlador, 178
Crispin, Lisa, 34
criterio de finalización, 163
Crosby, Philip, 18
crowdtesting, 34, 183
CST, 183
CSV, 276
cuestionario CSUQ, 187
cuestionario NASA TLX, 187
cuestionario SUS, 187
Cunningham, Ward, 246
código de Hammurabi, 11
código de ética, 54
datos de pruebas, 160
DDT, 269
decisión, regla de, 203
decisión, tabla de, 204
decisión, técnica de tabla de, 203
defecto, 45
defecto, estado de resolución de, 140
defecto, estado de triaje de, 139
defecto, gravedad de, 139
defecto, prioridad de, 139
defectos, lista de, 217
defectos, taxonomía de, 217
Deming, Edwards, 17
desplazamiento a la derecha de pruebas, 254
desplazamiento a la izquierda de pruebas, 253, 255
despliegue continuo, 252
deuda técnica, 246, 254
DevOps, 250
DeVries, Byron, 32
DevTestOps, 253, 263
diagrama causa-efecto, 17
diagrama de estados, 206
diagrama de quemado, 138
diagrama de transición de estados, 206
Dijkstra, Edsger, 25
driver, 178
Dromey, modelo de calidad de, 70
Dromey, R. Geoff, 32, 70
E2E testing, 256
Edison, Thomas Alva, 37, 45
Elmendorf, Bill, 24
entrega continua, 252
equivocación, 45
ERP, 64, 148, 223, 280
error, 45
error guessing, 216
error, predicción de, 216
estados, diagrama de, 206
estados, tabla de, 206
estados, técnica de transición de, 205
EULA, 240
evaluación de calidad de software, 103
Evans, Isabel, 31
experiencia de usuario, 254
exploración de vulnerabilidades, 189
Fagan, inspección de, 27, 191
Fagan, Michael, 27, 191
Falk, Jack, 30, 217
fallo, 47
feature flags, 255

- Feigenbaum, Armand V., 18
Fewster, Mark, 31, 35
flujo de control, 211
front-end, 255
FURPS, modelo de calidad, 69
- Gelperin, David, 26
Gherkin, 248, 265
Gilb, modelo de calidad de, 70
Gilb, Tom, 27, 70, 79
Grady, Robert, 29, 69, 217
Graham, Dorothy, 27, 31, 35
Gregory, Janeth, 34
gráfico de convergencia de defectos críticos, 141
- Hamming, Richard, 37
HCI, 255
Hegel, Friedrich, 21
herramienta Accessibility Scanner, 275
herramienta Appium, 267
herramienta Cucumber, 249, 265
herramienta JUnit, 263
herramienta Mockaroo, 276
herramienta Pylint, 260
herramienta Rapid Reporter, 218
herramienta Robot Framework, 250
herramienta Selenium, 266
herramienta SonarQube, 261
herramienta SortSite, 272
herramienta WAVE, 274
herramienta Zephyr, 277
Hetzl, William C., 25
historia de usuario, 245
Hopper, Grace, 45
Howden, William, 28
HTTP, 268
Humphrey, Watts, 30, 116
- IDE, 263
IEC, 37
- IEEE, 37
incidente, 173
informal group review, 192
ingeniería social, 242
inspección, 191
inspection, 191
integración continua, 179
interacción humano-computador, 255
interfaz de programación de aplicación, 268
ISACA, 116
Ishikawa, Kaoru, 17
ISO, 19, 37
ISTQB, 31, 152, 176, 233
- Jason, 276
JDBC, 269
Jorgensen, Paul C., 32
Juran, Joseph, 18
- Kanban, 245
Kaner, Cem, 30, 217, 231
Kelvin, Lord, 37
Kohl, Jonathan, 35
- Laboratorios Bell, 15
Laporte, Claude, 41
Leeds, Herbert, 23
linter, 259
linting, 259
Linus, Ley de, 32
Linux, 32
lista de chequeo, 218
lista de verificación, 218
Lovelace, Ada, 21
- manifiesto ágil, 32, 244
Marick, Brian, 31
Mark II, computador, 45
Martin, James, 28
McCabe, Thomas, 27
McCall, Jim, 68

- McCall, modelo de calidad de, 68
McCracken, Daniel D., 22, 23
medición, 79
medida, 80
mejora del proceso de software, 106
Menabrea, Luigi, 21
milestone review, 191
modelo de calidad datos, 71
modelo de calidad de producto, 71
modelo de calidad en uso, 71, 90, 101
modelo V, 29
Myers, Glenford, 27
método de desarrollo dirigido por el comportamiento, 248
método de desarrollo dirigido por pruebas, 247
método de desarrollo dirigido por pruebas de aceptación, 249
métrica Acoplamiento de componentes, 88
métrica Adaptabilidad al entorno de hardware, 89
métrica Adecuación de idiomas soportados, 86
métrica Capacidad de importación y reutilización de datos, 89
métrica Claridad de mensajes, 85
métrica Cobertura de pruebas, 87
métrica Cobertura funcional, 81
métrica Coexistencia con otros productos, 83
métrica Consecuencias de la fatiga, 94
métrica Corrección de errores de entrada de usuario, 85
métrica Correctitud funcional, 81
métrica CSAT, 135
métrica Dentro de presupuesto, 134
métrica Eficiencia de la modificación, 89
métrica Entrega a tiempo, 133
métrica Flexibilidad del producto, 97
métrica Frecuencia de reporte de salud de usuario, 97
métrica Integridad de datos, 87
métrica Intensidad de errores en tareas, 93
métrica Intercambio de formatos de datos, 83
métrica Interfaz de usuario autoexplicativa, 84
métrica MTBF, 70, 86
métrica NPS, 136
métrica Prioridad/gravedad de defectos, 140
métrica Proporción de tiempo productivo, 94
métrica Proporción de usuarios que se quedan, 95
métrica Resolución/gravedad de defectos, 141
métrica Retorno de la inversión, 96
métrica ROI, 96
métrica Satisfacción con características, 95
métrica Satisfacción del cliente, 135
métrica Suficiencia del mecanismo de autenticación, 88
métrica Tareas realizadas, 93
métrica Tiempo medio de respuesta, 82
métrica Tiempo medio entre fallos, 86
métrica Utilización media del procesador, 82
métrica Velocidad del equipo, 137
métrica Índice de promotores neto, 136

NASA, 47
Nguyen, Hung Q., 30, 217
NIST, 47
norma, 35
Norma IEEE Std. 1028-2008, 43, 191
Norma IEEE Std. 730-1980, 37
Norma IEEE Std. 730-2014, 37, 44
Norma ISO 25065:2019, 39
Norma ISO 90001:2015, 112
Norma ISO 9000:2015, 19
Norma ISO 9001:2015, 20, 112
Norma ISO/IEC 12207:2008, 38
Norma ISO/IEC 12207:2017, 130

- Norma ISO/IEC 14598-5:1998, 38
Norma ISO/IEC 20246:2017, 42, 44, 190
Norma ISO/IEC 25000:2014, 38
Norma ISO/IEC 25001:2014, 38
Norma ISO/IEC 25010:2023, 38, 71
Norma ISO/IEC 25012:2008, 39, 71
Norma ISO/IEC 25019:2023, 39, 71, 90, 101
Norma ISO/IEC 25020:2019, 39
Norma ISO/IEC 25022:2016, 39, 92–97, 101
Norma ISO/IEC 25023:2016, 39, 80–89
Norma ISO/IEC 25030:2019, 39, 101
Norma ISO/IEC 25040:2011, 39, 103
Norma ISO/IEC 25041:2012, 39
Norma ISO/IEC 29110-2-1:2015, 41
Norma ISO/IEC 29110-3-3:2012, 41
Norma ISO/IEC 29110-4-1:2018, 41
Norma ISO/IEC 33001:2015, 40
Norma ISO/IEC 33002:2015, 40
Norma ISO/IEC 33003:2015, 40
Norma ISO/IEC 33004:2015, 40
Norma ISO/IEC 33014:2013, 40, 114
Norma ISO/IEC 33020:2019, 40
Norma ISO/IEC 33030:2017, 40
Norma ISO/IEC 33063:2015, 40, 175
Norma ISO/IEC 40500:2012, 36
Norma ISO/IEC 9000-3:1997, 38
Norma ISO/IEC 9126:1991, 32, 38
Norma ISO/IEC DIS 29110-5-1-2:2024, 130
Norma ISO/IEC DTS 33010:2022, 40
Norma ISO/IEC TR 19759:2015, 41
Norma ISO/IEC TR 25021:2012, 39
Norma ISO/IEC TR 29110-1:2016, 41
Norma ISO/IEC TR 29110-3-1:2020, 41, 126
Norma ISO/IEC TR 29110-5-1-2:2011, 42
Norma ISO/IEC TR 29119-11:2020, 42
Norma ISO/IEC TR 29119-6:2021, 42
Norma ISO/IEC TS 25011:2017, 39
Norma ISO/IEC/IEEE 12207:2017, 38, 113
Norma ISO/IEC/IEEE 25010:2011, 185
Norma ISO/IEC/IEEE 25010:2023, 43
Norma ISO/IEC/IEEE 29119-1:2022, 42
Norma ISO/IEC/IEEE 29119-2:2021, 42, 152
Norma ISO/IEC/IEEE 29119-3:2021, 42, 152
Norma ISO/IEC/IEEE 29119-4:2021, 42
Norma ISO/IEC/IEEE 29119-5:2016, 42
Norma ISO/IEC/IEEE 90003:2018, 38, 43, 112
OAT, 182
olor de código, 262
oráculo, 28
pair review, 192
paradoja del pesticida, 30
Pareto, principio de, 18
partición de equivalencia, 200
PDCA, 16, 17, 20, 112
peer desk check, 192
Pettichord, Bret, 30, 33, 231
Pope, Franklin, 37
predicción de errores, 216
probar software, 233
proceso de software, 106
proceso iterativo, 245
proceso, rendimiento de, 127
producto de trabajo, 44
programación extrema, 33, 245
prueba, 234
prueba automatizada, 170
prueba basada en la especificación, 184, 199
prueba basada en la experiencia, 215
prueba basada en listas de comprobación, 218
prueba continua, 256
prueba de accesibilidad, 188
prueba de aceptación, 180, 181
prueba de aceptación contractual, 182
prueba de aceptación de usuario, 181
prueba de aceptación operacional, 182

- prueba de aceptación regulatoria, 182
prueba de caja blanca, 209
prueba de caja de cristal, 209
prueba de caja negra, 184, 199
prueba de capacidad, 186
prueba de carga, 186
prueba de compatibilidad, 187
prueba de componente, 178
prueba de confirmación, 190
prueba de código, 213
prueba de desempeño, 185
prueba de escalabilidad, 186
prueba de estrés, 186
prueba de fiabilidad, 188
prueba de gestión de memoria, 186
prueba de instrucciones, 213
prueba de integración, 178
prueba de integración de componentes, 179
prueba de integración de sistemas, 180
prueba de mantenibilidad, 189
prueba de módulo, 178
prueba de penetración, 189
prueba de portabilidad, 190
prueba de privacidad, 189
prueba de ramas, 214
prueba de regresión, 190
prueba de repetición, 190
prueba de resistencia, 186
prueba de seguridad, 188
prueba de sentencias, 213
prueba de servicios web, 268
prueba de sistema, 180
prueba de software, 234
prueba de unidad, 178
prueba de usabilidad, 187
prueba de volumen, 186
prueba exploratoria, 30, 217
prueba funcional, 184
prueba manual, 170
prueba repetida, 190
prueba, caso de, 164
prueba, elemento de, 164
prueba, etapa de, 177
prueba, fase de, 177
prueba, nivel de, 177
prueba, procedimiento de, 164
prueba, sub-proceso de, 177
prueba, tipo de, 184
pruebas A/B, 255
pruebas alfa, 182
pruebas basada en la especificación, técnica de diseño de, 199
pruebas basadas en la estructura, 209
pruebas basadas en la estructura, técnica de diseño de, 209
pruebas Beta, 182, 255
pruebas canario, 255
pruebas de caja blanca, técnica de diseño de, 209
pruebas de caja de cristal, técnica de diseño de, 209
pruebas de caja negra, técnica de diseño de, 199
pruebas de calidad, 184
pruebas de campo, 182
pruebas de extremo a extremo, 267
pruebas de integración, 178
pruebas de sistemas, 180
pruebas dirigidas por el contexto, 30
pruebas E2E, 256
pruebas en producción, 254
pruebas no funcionales, 184
pruebas, automatización de, 31
pruebas, base de, 160
pruebas, enfoques de, 231
pruebas, escuelas de pensamiento de, 231
pruebas, herramienta de, 258
pruebas, monitoreo y control de, 160

- pruebas, seguimiento y control de, 160
pruebas, técnica de diseño de, 163
PSP, 30

QA, 237

Raymond, Eric S., 32
recorrido, 192
refactorización, 247
Regla 80/20, 18
regla de decisión, 203
Reid, Stuart, 42
rendimiento de proceso, 127
requisito de calidad, 98
requisito funcional, 98
requisito no funcional, 98
REST, 268
resultado de la prueba, 171
resultado esperado, 171
resultado obtenido, 171
resultado real, 171
retesting, 190
retrabajo, 18, 27, 28, 106, 120, 123, 191, 257
Reuveni, Doron, 34
revisión de hitos, 191
revisión informal en grupo, 192
revisión por parejas, 192
revisión técnica, 191
rework, 18, 27, 28, 106, 120, 123, 191, 257
Richards, Paul, 68
riesgo de producto, 159
riesgo de proyecto, 159
ROI, 96
RST, 32, 33, 234
ruta de flujo de control, 211

Scrum, 245
SEI, 30, 116
sentencia, cobertura de, 213
SEO, 270
Serie de normas ISO/IEC 15504, 40

Serie de normas ISO/IEC 25000, 38
Serie de normas ISO/IEC 29110, 41, 130
Serie de normas ISO/IEC 33000, 40
Serie de normas ISO/IEC/IEEE 29119, 42
servicio web, 268
SGC, 19
Shewhart, Walter, 15
shift-left testing, 253, 255
shift-right testing, 254
sistema, 43
SLA, 240
SOAP, 268
software, 43
software, aseguramiento de calidad de, 44
software, calidad de, 44
software, característica de, 98
software, control de calidad de, 44
software, elemento de, 43
software, garantía de calidad de, 44
software, gestión de calidad de, 44
software, mejora del proceso de, 30, 106
software, proceso de, 43
software, producto de, 43
software, requisito de, 98
Solomon, Roy, 34
SPI, 30, 106
SPI, manifiesto, 108
SPICE, 39
SQA, 15, 44
SQC, 44
SQL, 276
stub, 178

tabla de decisión, 204
tabla de estados, 206
Taguchi, Genichi, 19
TDD, 33, 247, 253, 263
technical review, 191
teoría de cero defectos, 18
Thomson, William, 37

TMMI, 34, 155, 176, 233
Torvalds, Linus, 32
TQC, 18
TQM, 16
tres amigos, 248, 249
trilogía de Jurán, 18
TSP, 30
Turin, Alan, 22
Turing, premio, 25
técnica de lanzamiento oscuro, 255
técnica de tabla de decisión, 203
técnica de transición de estados, 205
técnicas basadas en la especificación, 199
técnicas basadas en la estructura, 209
técnicas de caja blanca, 209
técnicas de caja negra, 199

UAT, 181
UX, 254, 255

Van Veenendaal, Erik, 31, 34
velocity, 137
VSE, 41
vulnerabilidad, 261

walkthrough, 192
Walters, Gene, 68
WCAG, 59, 272, 274
Weinberg, Gerald M., 23

XML, 276
XP, 33, 245
xUnit, 33

“Los temas incluidos son los necesarios para el conocimiento de la materia. El orden de presentación se corresponde con la evolución de la calidad de software. Muy completo y con información rigurosa con un importante nivel de investigación. Bibliografía completa y actualizada incluyendo autores de reciente publicación.”

Alfonsina Morgavi

Directora, QActions Quality Group, Argentina.



“Este libro abarca de manera clara el proceso de pruebas, la calidad, y métricas del producto y proceso de software, siendo de mucha utilidad en la enseñanza de calidad de software teniendo en consideración lo escaso de textos en español enfocados en este tema.”

Javier Pino Herrera

Profesor, Universidad Veracruzana, México.

“Sandra es una de las investigadoras hispanoamericanas más productivas en Ingeniería del Software. Destacan sus innovadores trabajos relacionados con accesibilidad y usabilidad, atributos de calidad del software de creciente importancia durante este siglo.”

Ignacio Trejos Zelaya

Profesor Catedrático, Escuela de Ingeniería en Computación, Tecnológico de Costa Rica.



Sandra Sánchez Gordón, Ph.D.

Investigadora acreditada por la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador. Miembro de la Red Ecuatoriana de Mujeres Científicas y la Organización para Mujeres en Ciencia del Mundo en Desarrollo de Unesco. Doctora en Aplicaciones Informáticas, Universidad de Alicante, España y Master en Ingeniería de Software, Universidad Drexel, Estados Unidos. Docente del Departamento de Informática y Ciencias de la Computación de la Escuela Politécnica Nacional del Ecuador. En 2017 obtuvo reconocimiento a la mejor producción científica de su universidad. Representante de Ecuador en el Comité de Calificaciones de Pruebas de Software, Hispanoamérica desde 2013. Tiene 30 años de experiencia en desarrollo de software, soluciones informáticas y gestión de calidad en Ecuador, Panamá y Estados Unidos. Co-fundadora de la empresa de desarrollo de software InSoft Cia. Ltda.

Impreso por:



ESCUELA
POLITÉCNICA
NACIONAL



ISBN: 978-9942-42-867-7



9 789942 428677