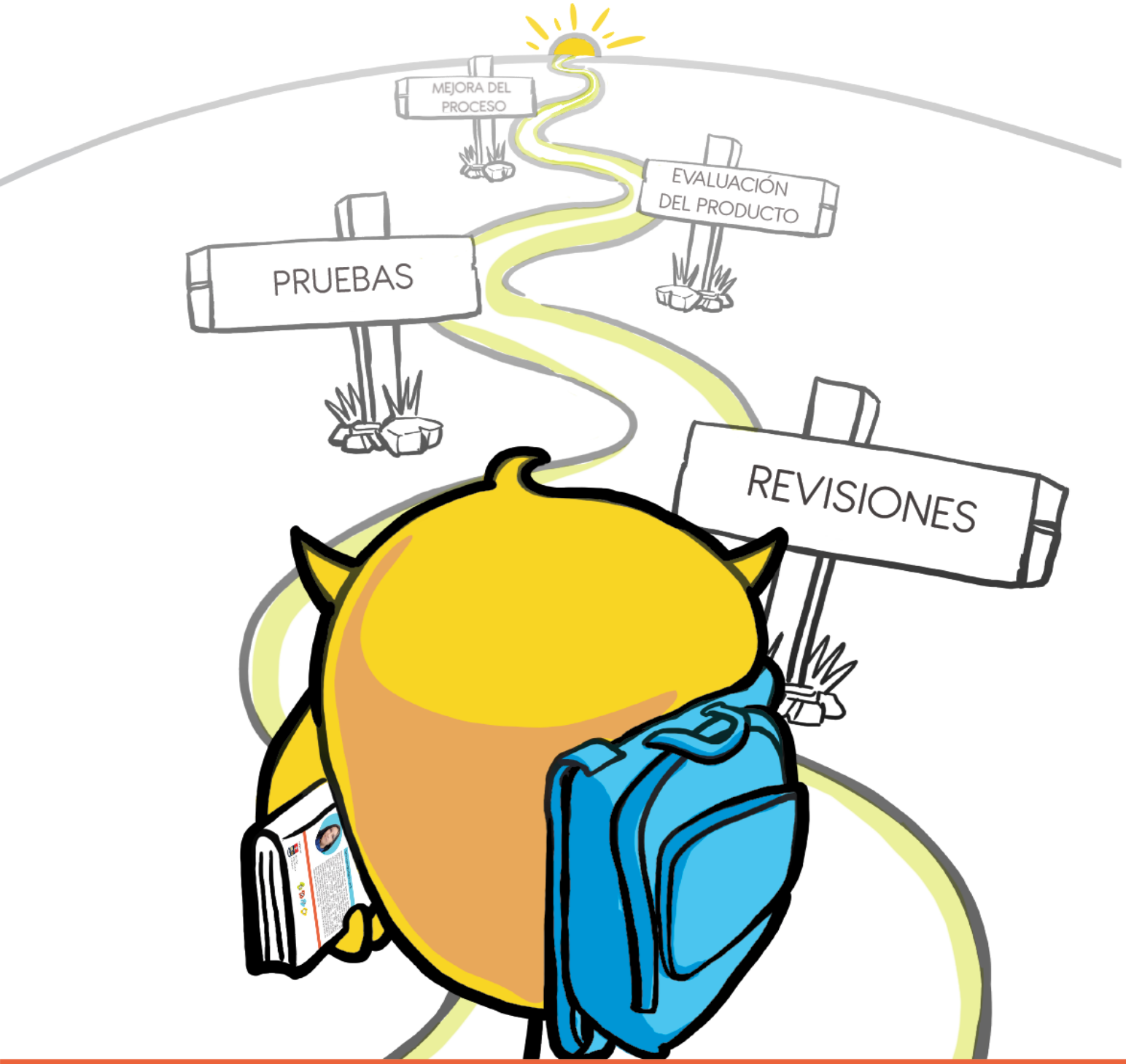
Primera Edición

Calidad en Software

UN LIBRO DE TEXTO



Sandra Sanchez-Gordon

Diseño gráfico por Gabriela Zaldumbide.
Gestión de sitio web por Daniela Zaldumbide.

Primera Edición
Publicado por Escuela Politécnica Nacional
Impreso en Quito-Ecuador

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**	34
1.4	**Definiciones**	42
1.5	**Ética y calidad de software**	51
1.6	**Autoevaluación, retos, y aprendizaje autónomo**	60

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

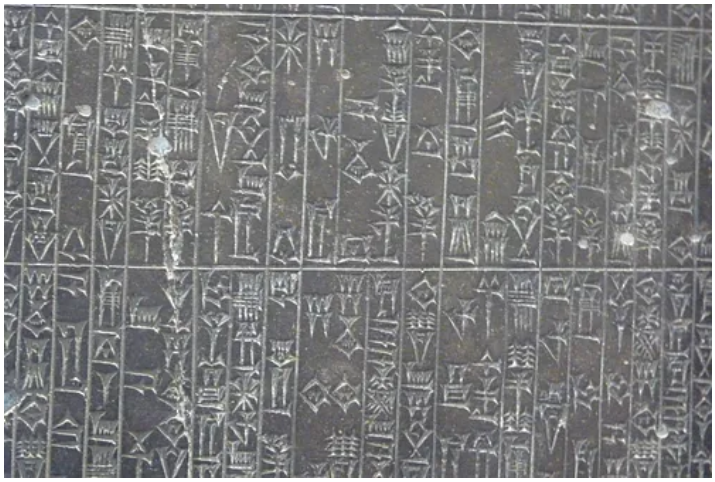


Figura 1.1: Fragmento del código de Hammurabi.
Fuente: Museo de Louvre [174].

> 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 haya 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 [19].

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 (hacer) todo tipo de productos." - Capilla Funeraria de Rekhmire [5].

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 especificaciones.

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.

Figura 1.2: Detalle de la tumba de Tekhmire.
Fuente: Arte historia Egipto [5].

1.1.2 Edad media

En la Europa medieval, desde finales del Siglo 13 hasta inicios de Siglo 17, aparecen los gremios de artesanos [148, 163]. 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 conocidos. 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 [147].

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 con-

Figura 1.3: Taller de zapatería.
Fuente: El libro de los oficios [3].

centren en las ciudades. Es así como, durante la época de la Primera Revolución Industrial, desde mediados de Siglo 18 hasta mediados del Siglo 19, los talleres artesanales desaparecieron 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 recurre 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 jerarquía de trabajo con roles y responsabilidades, y una estructura de informes de cada función. Los controles de los supervisores se volvieron necesarios para garantizar la productividad y la calidad [147].

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 el 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 X se tiene la escala de las muestra tomadas (en este caso 16 muestras) y en el eje vertical Y 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) - por sus siglas 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” [159] 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.

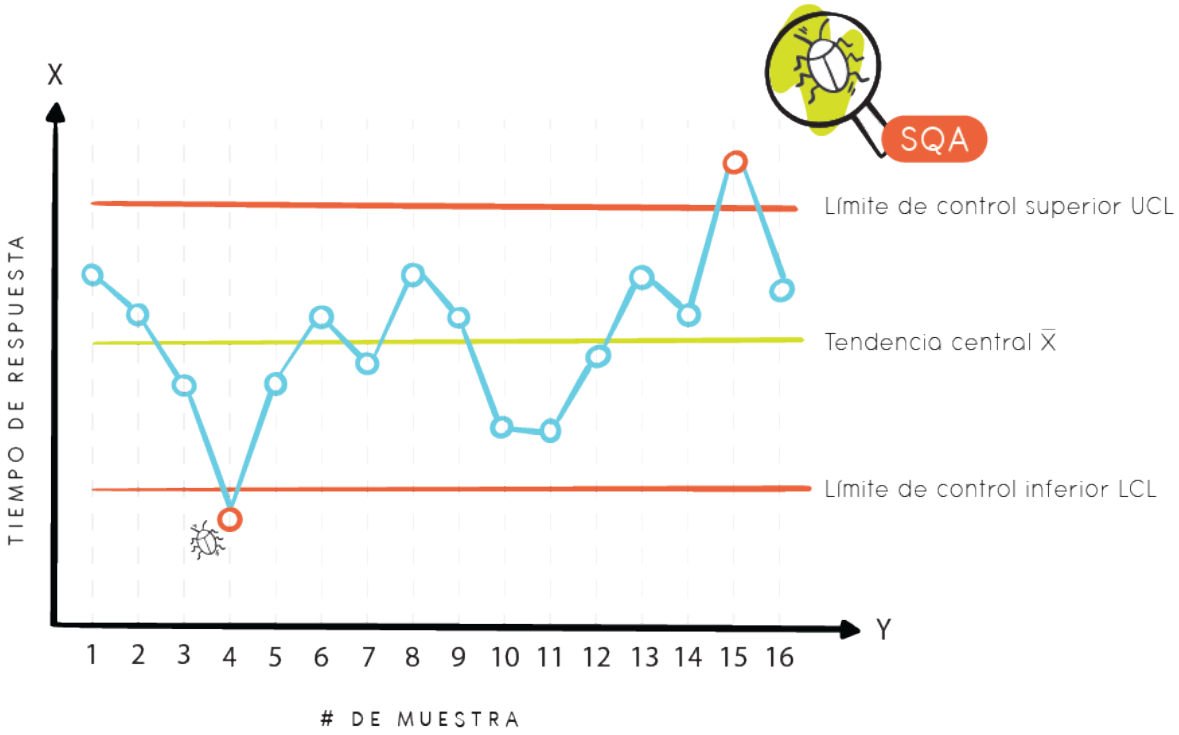


Figura 1.4: Ejemplo de Gráfico de Control de Proceso de Shewhart.
Elaboración: Autora.

Recomendación: *Investigue el concepto de gestión de la calidad total (TQM), por sus siglas en inglés, Total Quality Management.*

En 1939, en su libro ”Método estadístico desde la perspectiva del control de calidad” [160], 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 control 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, Edward 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), por sus siglas en inglés, Plan Do Check Act .

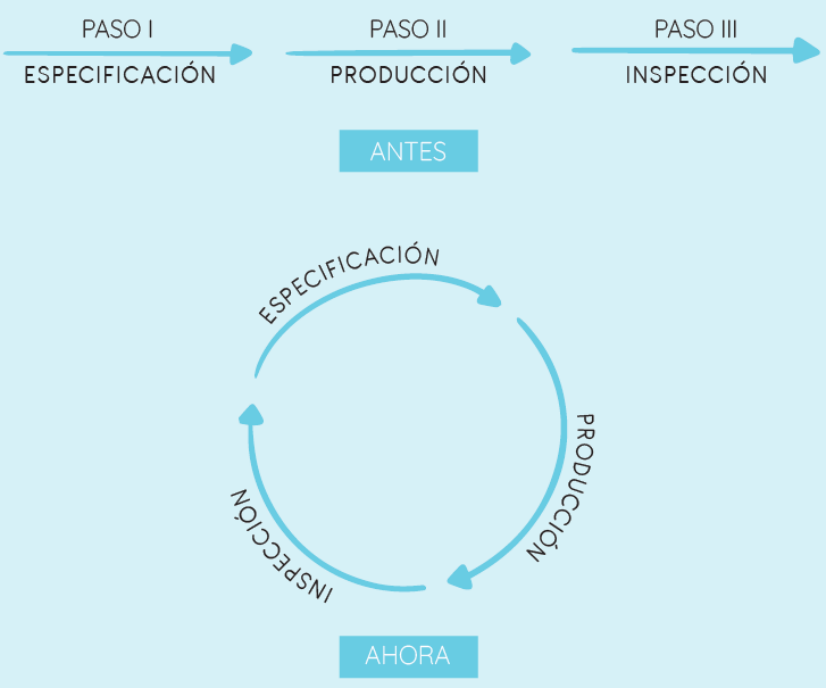


Figura 1.5: Ciclo Shewhart.
Fuente: Método estadístico desde la perspectiva del control de calidad. Adaptación: Autora. [160].

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" [70], 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.

> **Definición 1.1 Control de Calidad según Ishikawa [70].** Practicar el control de calidad es desarrollar, diseñar,producir y mantener un producto de calidad que sea el más económico, el más útil y siempre satisfactorio para el consumidor.

W. Edwards Deming

Terminada la Segunda Guerra Mundial en 1945, el doctor en física estadounidense W. 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 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” [34], 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 1.2 Calidad según Deming [34]. 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” [126] 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 mejoramiento. En su libro ”Juran sobre la planificación de la calidad” [125] 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 1.3 Calidad según Juran [125]. Calidad es adecuación para el uso.

Armand Feingenbaum

En 1961, el doctor en economía estadounidense Armand Feingenbaum acuñó el concepto de Control de Calidad Total (TQC) - por sus siglas 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, Feingenbaum 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” [44] 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 Feingenbaum, la calidad depende de la perspectiva del cliente y es responsabilidad de toda la empresa.

Definición 1.4 Calidad según Feingenbaum [44]. La calidad se basa en la experiencia real del cliente con el producto o servicio medido en función de sus requisitos - declarados o no, conscientes o solo percibidos, técnicamente operativos o totalmente subjetivos, y siempre representa un objetivo móvil en un mercado competitivo.

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” [33] donde propone el establecimiento de buenos principios de gestión de la calidad en las organizaciones.

Definición 1.5 Calidad según Crosby [33]. La calidad es la conformidad con un conjunto de especificaciones. Estas especificaciones se establecen en función de las necesidades y deseos del cliente.

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 linea; y el diseño de productos y procesos basado en parámetros de diseño que determinan el buen funcionamiento del equipo de trabajo.

Definición 1.6 Calidad según Taguchi [165]. La calidad de un producto esta 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) - por sus siglas 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 cualquier tipo de organización orientada a la producción de bienes o servicios. La norma ISO 9000:2015 ”Sistemas de gestión de calidad - Fundamentos y vocabulario” [73] ,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 1.7 Calidad según norma ISO 9000:2015 [72]. 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” [73] , 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 el mejoramiento continuo 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 [31].

Un estudio realizado en Ecuador en el año 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 [20]. 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 caracter dinámico según la época [25]. 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 [135]. 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 Feingenbaum. Por otra parte, se puede entender a la calidad como cualidad de un objeto, esto 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 desde el punto de vista de los productores o desde el punto de vista de los consumidores. A esta segunda categoría corresponden las definiciones propuestas por Crosby y la serie de normas ISO 9000:2015.

1.2 Evolución de la calidad de software

La historia de la calidad de software va de la mano de la historia del software, que a su vez va de la mano de la historia 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, mantenimiento, 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.

1.2.1 Período 1840-1950

Ada Lovelace

En la década de 1840, la matemática y escritora inglesa Ada Lovelace tradujo del francés al inglés el artículo ”Boceto del motor analítico inventado por Charles Babbage” [140]. 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 itálicas son de Lovelace):

> ”La máquina analítica no tiene pretensiones de crear ***nada***. Puede hacer cualquier cosa que ***sepamos como ordenarle*** que realice.” - Ada Lovelace [140].

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 [140].

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” [167], 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 base para 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 [167].

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

1.2.2 Período 1951-1970

Daniel McCracken

En 1957, el científico computacional estadounidense Daniel McCracken publica el libro ”Programación de computadores digitales” [139] 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 [139].

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.

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

Gerald Weinberg

En 1958, en el marco del proyecto espacial 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 Gerard M. Weinberg trabajó en Mercurio. En 1961, en base a la experiencia en dicho poryecto, Weinberg y el ingeniero estadounidense Herbert D. Leeds publican el libro ”Fundamentos de la programación informática” [132] 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” [173] donde resalta el aspecto humano

de la programación, cuyas ideas se mantienen vigentes. En 2010, Weinberg publica el libro ”Software perfecto y otras ilusiones sobre las pruebas” [172] donde sostiene que las pruebas son necesarias porque las personas no somos perfectas, pero el hecho de probar más, no garantiza una mayor calidad.

> ”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 Weinberg [173].

Recomendación: *Lea el libro ”La psicología de la programación informática”, autor Gerard 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” [41], 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” [40] 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 Organizacion del Tratado del Atlantico Norte (NATO) - por sus siglas 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” [18] en el cual se dedica una sección a la garantía de la calidad de software. En este documento se proponen, 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), por sus siglas en inglés, Association for Computing Machinery. La carta se publicó con el título ”Sentencia Go To considerada perjudicial” [35] y era una crítica el 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. 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 de premio se tituló ”El humilde programador” [36], en donde manifestó 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 [36].

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

1.2.3 Período 1971-2000

William C. Hetzel y David Gelperin

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, 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 registrada enfocada exclusivamente en pruebas de software. En 1988, Gelperin y Hetzel publican el artículo ”El crecimiento de las pruebas de software” [49] donde describen cuatro modelos para pruebas

de software: demostración - que el software satisface su especificación, destrucción - detectar fallos de implementación, evaluación - detectar defectos en los requisitos, el diseño, y la implementación, y prevención - evitar defectos de requisitos, diseño e implementación. Ese mismo año, Hetzel publica el libro ”Guía completa de pruebas de software” [63], 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) - por sus siglas 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” [23]. 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 tendiendo vigencia incluso para entornos ágiles y DevOps. A continuación, algunas de sus ideas 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 prueba 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 [23].

Recomendación: *Lea el ensayo ”No hay bala de plata” del libro ”El mítico hombre-mes”, autor Frederick Brooks, 1975.*

Tom Gilb

El ingeniero de sistemas estadounidense Tom Gilb publica en 1975 el artículo ”Leyes de la no fiabilidad” [50]. Gilb es uno de los primeros informáticos en definir la fiabilidad del sistema y del software, y la relación entre el error humano y el error del sistema. Su libro

”Métricas de software” [52], publicado en 1976, se considera texto de referencia por el gran número de métricas que presenta. En 1993, Gilb publica junto con Dorothy Graham, el libro ”Inspecciones de software” [53] donde detallan el proceso de revisión formal de software denominado inspección.

Michael E. Fagan

En 1976, el físco e ingeniero eléctrico estadounidense Michael E. Fagan publica el artículo ”Inspecciones de diseño y código para reducir errores en el desarrollo de programas” [43] 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” [137] 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” [144] en donde proclama:

> ”El objetivo de los probadores es hacer que el programa falle.” - Glenford Myers [144].

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” [145] 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

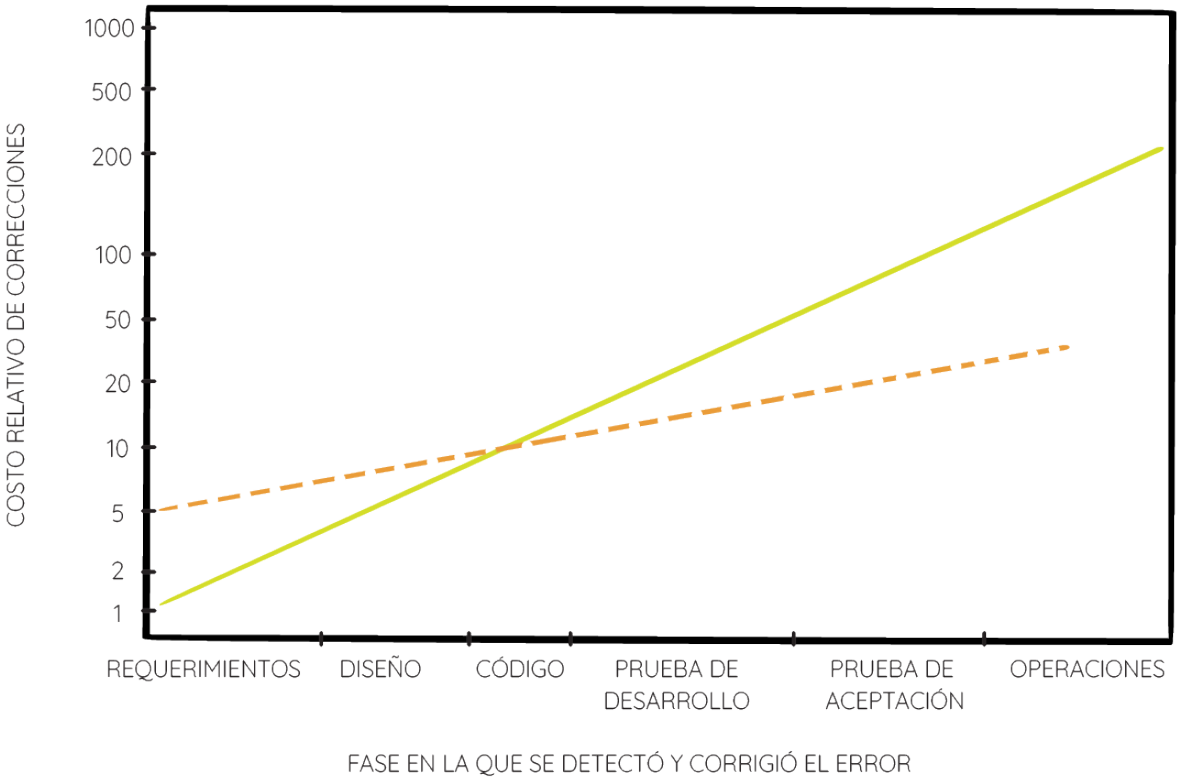


Figura 1.6: Costo relativo de correcciones durante el ciclo de vida del software.
Fuente: Economía de la ingeniería de software [21]. Adaptación: Autora.

el término oráculo para refererirse a un mecanismo para determinar si una prueba ha pasado o fallado.

Barry W. Boehm

El matemático y científico computacional Barry Boehm publica en 1981 el libro ”Economía de la ingeniería de software” [21] 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 X se tiene las fases del ciclo de vida del software, y en el eje vertical Y el costo relativo de corregir un defecto detectado en una fase determinada comparado con arreglar el mismo defecto en una fase diferente. La línea con mayor pendiente corresponde a proyectos grandes, mientras que la segunda línea, que se muestra entrecortada y con menor pendiente, corresponde a proyectos de software pequeños. En este libro también presenta el Modelo de costos Constructivos (COCOMO) - por sus siglas 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” [134] 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” [155], el científico de la computación inglés Paul E. Rook presenta el Modelo V para desarrollo de software, donde introduce 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 del desarrollo una correspondiente fase de pruebas: requerimientos y pruebas de aceptación, diseño de sistema y pruebas de sistemas, diseño de software y pruebas de integración, diseño de componentes y pruebas de unidad.

Cem Kaner

En 1988 se publica el libro ”Pruebas de software informático” [130] escrito por el matemático, abogado, y sicólogo estadounidense Cem Kaner, junto con 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” [129]. Kaner ha aportado además con la redacción de leyes para el licenciamiento de software, la regulación de la calidad de software, y el comercio electrónico en los Estados Unidos.

Watts Humphrey

El físico y administrador estadounidense Watts Humprey es considerado el padre de la calidad de software por sus contribuciones al mejoramiento del proceso de software (SPI) - por sus siglas en inglés, Software Process Improvement. Humphrey es el fundador del programa de procesos de software del Instituto de Ingeniería de Software (SEI) - por sus siglas en inglés, Software Engineering Institute. En 1989, Humphrey publica el libro ”Gestión del proceso de software” donde propone el modelo de madurez de las capacidades (CMM) - por sus siglas en inglés, Capability Maturity Model - para mejorar la calidad y productividad del proceso de desarrollo de software.

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” [16]. 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.*

Dorothy Graham

En 1991, Unicom publica el primer ”Reporte sobre pruebas de software asistidas por computador (CAST)” [57] - por sus siglas en inglés, Computer Aided Software Testing - escrito por la consultora en pruebas de software estadounidense Dorothy Graham. En 1999, Graham publica junto con Mark Fewster el libro ”Automatización de pruebas de software” [45] , considerado una obra clásica en el ámbito de la automatización de pruebas. En 2006, Graham publica junto con Erik Van Veenendaal, Isabel Evans, y Rex Black el libro ”Fundamentos de las pruebas de software: Certificación ISTQB” [59] donde se describe el programa de estudios para la certificación de nivel básico de pruebas del Cómite Internacional de Cualificaciones de Pruebas de Software (ISTQB) - por sus siglas en inglés, International Software Testing Qualifications Board [120]. 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 intentado hacerlo bajo su supervisión. El libro se enfoca en subsistemas de tamaño medio, tales 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.

Figura 1.7: Dorothy Graham y Sandra Sanchez-Gordon en el 7mo. Congreso Mundial de Calidad de Software, 2017, Lima-Perú.
Fuente: Archivo personal.

Recomendación: *Investigue en que consisten los cuatro 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” [123]. Un cuarto de siglo después, en 2021, Jorgensen publica la quinta edición que tiene como co-autor a Byron DeVries. Este libro se ha convertido en referencia de las tecnologías en evolución en el ámbito de las pruebas de software [124].

R. Geoff Dromey

En 1996, en su artículo ”Acorralando a la quimera” [38], 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 [96].

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) - por sus siglas 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” 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 [153].

La Ley de Linus implica que cuanto más publicamente 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” [10]. 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”.

1.2.4 Período 2001-2020

Kent Beck

En 2002, el ingeniero en ciencias de la computación estadounidense Kent Beck publica el libro ”Desarrollo dirigido por pruebas: Mediante el ejemplo”[14] 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) - por sus siglas 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) - por sus siglas 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 pronuncia la charla 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, diriga por normas, orientada hacia la calidad, y dirigida por el contexto. Posteriormente, se incorpora 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” [22], Bolton distingue entre los conceptos de probar y comprobar. 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, por sus siglas en inglés, Test Maturity Model Integration [46]. El modelo TMMI sirve para evaluar y mejorar el proceso de pruebas de las organizaciones y se basa en su predecesor, el modelo de TMM desarrollado en 1996 en el Instituto de Tecnología de Illinois.

Mike Cohn

En 2009, Mike Cohn publica el libro ”Triunfando con la agilidad” [29] 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.*

Doron Reuveni

En 2007, el ingeniero en sistemas computacionales y sistemas de información israelí Doron Reuveni, junto con Roy Solomon, publica el libro ”Guía esencial de crowdtesting” [4]. El término crowdtesting surgió del término crowdsourcing que fue acuñado en 2006, por Jeff

Howe y Mark Robinson, para describir la externalización de actividades parciales de las organizaciones a un grupo de voluntarios. Crowtesting 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.

Lisa Crispin

En 2009, la experta en pruebas de software y agilismo Lisa Crispin publica junto con Janeth Gregory el libro ”Pruebas ágiles: Una guía práctica para probadores y equipos ágiles” [32] 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, Crispin y Gregory publicaron, en 2014, otro texto importante en el mismo ámbito, titulado: ”Más pruebas ágiles: Viajes de aprendizaje para todo el equipo” [60]. Este libro explica 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” [58]. En este capítulo, Kohl propone utilizar la automatización para llevar a cabo tareas como: la configuración de pruebas, la generación de datos, y para progresar a lo largo de un flujo de trabajo. Kohl propone además 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 1.8 Norma según ISO 24765:2017 [112]. 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

En el contexto de desarrollo de software, 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 la construcción del software y, mediante el proceso de pruebas, depurar los defectos residuales antes de la entrega del producto. La identificación y tratamiento de estos defectos en las primeras fases del proyecto de software es crucial para disminuir los costos de su desarrollo y posterior mantenimiento. La industria de software gradualmente va asimilando la importancia y beneficios del uso de normas técnicas. Los esfuerzos de normalización en el ámbito de software han sido históricamente liderados por las siguientes organizaciones:

- Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) - por sus siglas 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 siguiente equipo: 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) - por sus siglas 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.
- Asociación de Maquinaria Computacional (ACM) - por sus siglas 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.
- Organización Internacional de Normalización (ISO) - por sus siglas 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.

Las normas internacionales, a diferencia de otras directrices escritas, como por ejemplo procesos o méjores prácticas de la industria, son documentos que pueden convertirse en requisitos de cumplimiento obligatorio cuando son adscritas por los órganos reguladores de los países signatarios.

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” [94] es de cumplimiento obligatorio en

muchos países a nivel mundial, incluyendo España y Latinoamérica.

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 [30].
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 [48].
2012	España	Asociación Española de Normalización (AENOR)	Norma UNE-EN 301 549. En el Real Decreto 1494/2007 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 propietario de un sitio web que incumpla recibirá sanciones previstas en la Ley No. 2007-76 del Sistema Ecuatoriano de la Calidad [158].

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 1979 por la IEEE. Esta norma trataba sobre planes de garantía de calidad del software e impulsó el desarrollo de normas subsecuentes en requisitos, diseño, pruebas, y verificación y validación del 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, la 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” [65]. 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” [66] 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” [74], 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 es la del año 2017 [111].

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” [71]. 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” [118].

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” [96] 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” [75] 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 2015, las normas ISO/IEC 9126 e ISO/IEC 14598 fueron reemplazadas 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” [76]. 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” [76].
- Norma ISO/IEC 25001:2014 ”Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Planificación y gestión” [77].
- Norma ISO/IEC 25010:2011 ”Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Modelos de calidad de sistemas y software” [78].
- 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” [100].
- Norma ISO/IEC 25012:2008 ”Ingeniería de software - Evaluación y requisitos de calidad de productos de software (SQuaRE) - Modelo de calidad de datos” [79].
- 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” [80].
- 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” [105].
- 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” [81].
- 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” [82].
- 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” [83].

- Norma ISO/IEC 25040:2011 ”Ingeniería de sistemas y software - Evaluación y requisitos de calidad de sistemas y software (SQuaRE) - Proceso de evaluación” [84].
- 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” [85].
- 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” [86].

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) - por sus siglas en inglés, Software Process Improvement and Capability dEtermination, que es un marco para la evaluación y mejoramiento de 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, mantenimiento, 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” [88].
- Norma ISO/IEC 33002:2015 ”Tecnología de la información - Evaluación de procesos - Requisitos para realizar evaluación de procesos” [89].
- Norma ISO/IEC 33003:2015 ”Tecnología de la información - Evaluación de procesos - Requisitos para marcos de medición de procesos” [90].
- 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” [91].
- Norma ISO/IEC DTS 33010 ”Tecnología de la información - Evaluación de procesos - Guía para la realización de evaluaciones de procesos” [97].
- Norma ISO/IEC TR 33014:2013 ”Tecnología de la información - Evaluación de procesos - Guía para la mejora de procesos.” [99].

- 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.” [92].
- Norma ISO/IEC TS 33030:2017 ”Tecnología de la información - Evaluación de procesos - Un proceso de evaluación documentado ejemplar” [101].
- Norma ISO/IEC 33063:2015 ”Tecnología de la información - Evaluación de procesos - Modelo de evaluación de procesos para pruebas de software” [93].

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)” [104] - por sus siglás 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 1.9 VSE según norma ISO/IEC TR 29110-1:2016 [106]. 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) - por sus siglas en inglés, Very Small Entities - que desarrollan sistemas o software. La serie ISO/IEC 29110 define un grupo de cuatro perfiles genéricos: entrada, básico, intermedio, y avanzado, para guiar a las VSE en su evolución desde el inicio hasta la madurez. Las directrices proporcionan a las VSE un conjunto de procesos, actividades, tareas y roles, así como el contenido de los productos de trabajo de entrada y salida, tanto para la gestión del proyecto como para la implementación del 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” [106].
- 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” [87].

- Norma ISO/IEC TR 29110-3-1:2020 ”Ingeniería de sistemas y software - Perfiles del ciclo de vida para entidades muy pequeñas (VSE) - Parte 3-1:2020 Directrices para la evaluación de procesos” [107].
- 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. [102].
- 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” [98].
- 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” [108].

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

En 2007, la ISO forma un grupo de trabajo coordinado 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 para describir los resultados. 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, la serie abarca las siguientes normas:

- Norma ISO/IEC/IEEE 29119-1:2022 ”Ingeniería de software y sistemas - Pruebas de software - Parte 1: Conceptos generales” [113].
- Norma ISO/IEC/IEEE 29119-2:2021 ”Ingeniería de software y sistemas - Pruebas de software - Parte 2: Procesos de prueba” [114].
- Norma ISO/IEC/IEEE 29119-3:2021 ”Ingeniería de software y sistemas - Pruebas de software - Parte 3: Documentación de las pruebas” [115].
- Norma ISO/IEC/IEEE 29119-4:2021 ”Ingeniería de software y sistemas - Pruebas de software - Parte 4: Técnicas de prueba” [116].
- Norma ISO/IEC/IEEE 29119-5:2016 ”Ingeniería de software y sistemas - Pruebas de software - Parte 5: Pruebas basadas en palabras clave” [117].

- 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 [110].
- 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” [109].

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” [103] 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” [67].

1.4 Definiciones

La calidad de un producto 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 todos los interesados. 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 esta sección se presentan las definiciones de: sistema, software o producto de software, elemento de software, componente de software, y producto de trabajo.

Definición 1.10 **Sistema según norma ISO/IEC 25010:2011 [78].** 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 1.11 **Producto de Software según norma ISO/IEC/IEEE 90003:2018 [118].** 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 1.12 Elemento de Software según norma ISO/IEC/IEEE 90003:2018 [118]. Parte identificable de un producto de software.

Definición 1.13 Componente de Software según norma ISO/IEC 25010:2011 [78]. Entidad con estructura discreta, como un conjunto o un módulo de software, dentro de un sistema, considerado a un nivel de análisis específico.

Definición 1.14 Producto de Trabajo según según norma ISO/IEC 20246:2017 [103]. Un producto de trabajo es cualquier artefacto producido por un proceso. Ejemplos: plan de proyecto, especificación de requisitos, documentación de diseño, código fuente, plan de pruebas, entre otros.

1.4.2 Definiciones relativas a calidad de software

En esta sección se presentan las definiciones de: calidad de software, control de calidad de software, garantía de calidad de software, y gestión de la calidad de software.

Definición 1.15 Calidad de Software según norma IEEE Std. 730-2014 [66]. 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 1.16 Control de Calidad de Software (SQC) según Pressman y Maxim [152]. 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. Se abrevia SQC por sus siglas en inglés, Software Quality Control.

Definición 1.17 Garantía de Calidad de Software (SQA) según norma IEEE Std. 730-2014 [66]. 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. Se abrevia SQA por sus siglas en inglés, Software Quality Assurance.

Definición 1.18 Gestión de Calidad de Software según norma IEEE Std. 730-2014 [66]. 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 [28].

Todo miembro del equipo de desarrollo de un producto de software puede 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 1.19 **Error según ISTQB [121].** Acción humana que produce un resultado incorrecto.

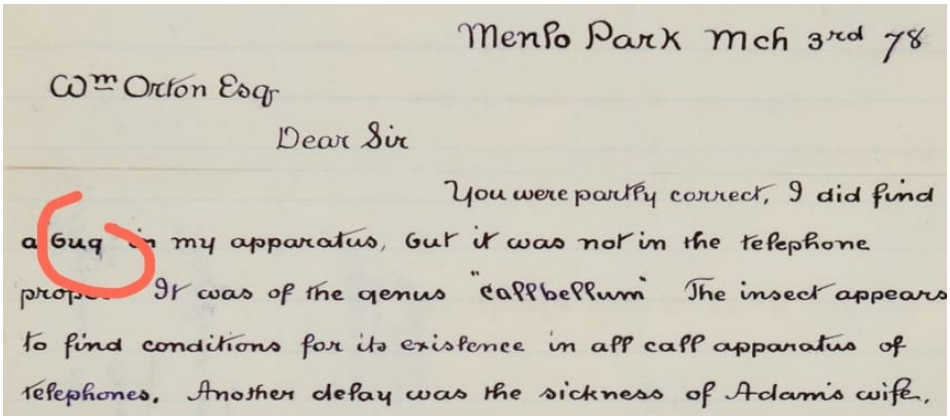
Un error cometido implica la inserción de uno o varios defectos en un producto de trabajo específico durante una fase puntual del ciclo de vida de desarrollo, lo que puede desencadenar la inserción de más defectos en otros productos de trabajo o en elementos del software durante fases posteriores.

Definición 1.20 **Defecto de software según ISTQB [121].** 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 sinómino 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 creo 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, indica 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 [39].

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


Menlo Park Mch 3rd 78

Wm Orton Esqr

Dear Sir

You were partly correct, I did find a bug in my apparatus, but it was not in the telephone proper. It was of the genus "cappbellum" The insect appears to find conditions for its existence in all call apparatus of telephones. Another delay was the sickness of Adams wife.

Figura 1.8: Carta de Thomas Alva Edison de 1878.
Fuente: Galerías de Subastas Swann [39].

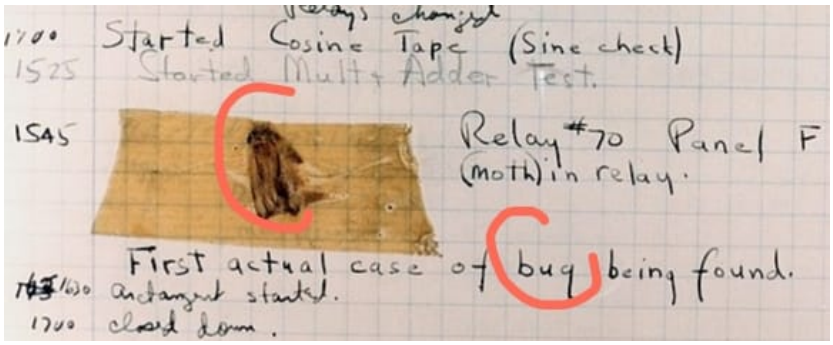

1700 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Figura 1.9: Captura de la bitácora del computador Mark II.
Fuente: Librería del Centro Naval Histórico de Estados Unidos [168].

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 [168].

Definición 1.21 Fallo de sofware según ISTQB [121]. 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 un 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) - por sus siglas 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 unos 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 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) - por sus siglas en inglés, National Aeronautics and Space Administration - de los Estados Unidos lanzó un cohete que transportaba la sonda espacial Mariner 1 con el objetivo de 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 superguión 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 previo. 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 en la actualidad a 169 millones de dólares.

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 antimisíl Patriot.

El 25 de febrero de 1991, durante la Guerra del Golfo, un campamento militar de Estados Unidos localizado en Dharan, Arabia Saudí, fue alcancado por un misíl Scud iraquí causando el fallecimiento de 28 soldados y más de cien heridos. El sistema antimísil Patriot que protegía al campamento falló en detectar al misíl. La causa fue un error de redondeo en el cálculo de la ruta del mísil. Específicamente, el tiempo era medido en décimas de segundo utilizando el reloj interno del sistema. Para transformalo 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 periodicamente 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 del incidente.

Ejemplo 1.7 Cohete Ariane 5.

El 4 se junio de 1996, la Agencia Espacial Europea realizó el lanzamiento del cohete Ariane 5. El cohete se auto-destruyó 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 de la misión fallida fue 320 millones de dólares.

Ejemplo 1.9 Cambio de milenio (Y2K).

El problema del cambio de milenio - también conocido como Y2K por sus siglas 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 la realidad, 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 canadiense 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 quedo 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) - por sus siglas en inglés, Child Support Agency. El contrato se adjudicó a la empresa de servicios informáticos Sistemas de Datos Electrónicos (EDS), por sus siglas 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.

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 en 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”. 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 era errónea y no debía haber superado los 9 025 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 Hawai, se emitió erroneamente 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 envió 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 televisores, 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 Metropolitada de Tránsito de Quito-Ecuador presentaba intermitencias de dispobilidad debido a la gran cantidad de usuarios intentando

Figura 1.10: Tienda en línea.
uente: Diario El Popular de Perú [151].

acceder para obtener un turno. Los usuarios denunciaron que, una vez que lograban acceder, el sistma 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 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 de software, la calidad - o falta de calidad - de los productos de software puede tener un profundo impacto

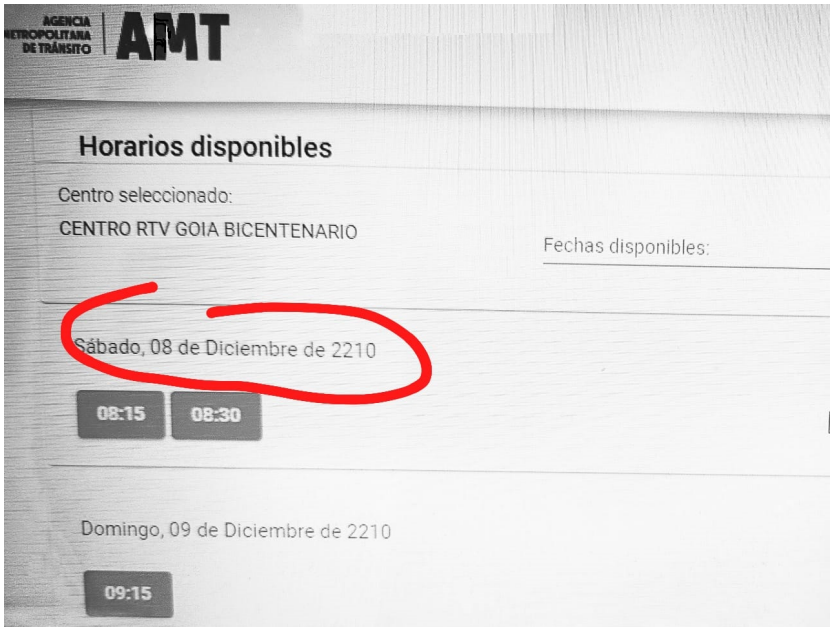


Figura 1.11: Fallo en asignación de turnos.
Fuente: Diario El Universo de Ecuador [170].

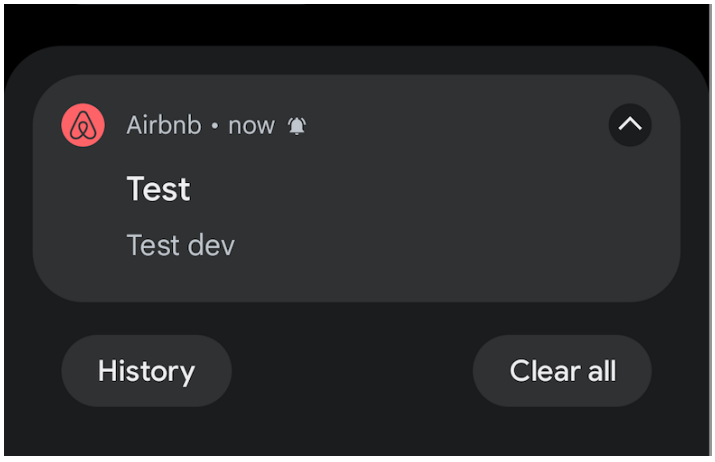


Figura 1.12: Fallo en la aplicación de Airbnb.
Fuente: Portal Airbnbase [2].

en el bienestar individual y en la armonía de la sociedad.

Definición 1.22 Etica Informática según ACM [54] . 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, 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 los imperativos 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 de software que desarrollan.

1.5.1 Código de ética y conducta profesional de la informática

El código de ética y 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 de la 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 [54]. El código contiene principios éticos generales, principios sobre la responsabilidad profesional, principios de liderazgo profesional, y principios de cumplimento del código.

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) Honrar 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 el análisis de los riesgos.

   (f) Realizar trabajos sólo en las áreas de 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 sociotécnicas. 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. La técnicadenominada CARE propone los siguientes pasos [55]:

- **C**onsiderar las partes interesadas y los elementos éticos.
- **A**nalizar el impacto.
- **R**evisar las responsabilidades y los enfoques alternativos.
- **E**valuar los pros y los contras.

Considerar

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

- ¿Quienes se verán afectados en su comportamiento y sus procesos de trabajo?
- ¿Quienes se verán afectados en sus circunstancias o trabajo?
- ¿Quiénes se verán afectados en sus experiencias?
- ¿Qué alternativas plausibles pueden abordar las necesidades e impactos de los diferentes interesados?
- ¿A quiénes se necesita para plasmar estas alternativas?

Analizar

A continuación se debe analizar las obligaciones y los derechos de todos los interesados, mediante 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 más relevantes para el sistema?
- ¿Qué principios del código son 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, mediante las siguientes preguntas:

- ¿Qué responsabilidades, autoridad, prácticas, o políticas parecen 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 analizar y considerar, a la luz de las siguientes cuestiones:
  - ¿Qué suposiciones se han hecho acreca de las partes interesadas?
  - ¿Cómo podrían utilizar el sistema los usuarios con discapacidades?
  - ¿Cómo apoyan las alternativas planteadas a cumblir los valores profesionales del código?

Evaluar

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

- ¿Cuál de las opciones consideradas parece ser la mejor?
- ¿Cuáles son los pros y los contras?

- ¿Existen alternativas creativas adicionales a las opciones que se ha considerado hasta ahora?

- ¿Hay en este punto otros principios del código que sean más 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. 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 a los dilemas éticos. Una vez tomada la decisión, ésta debe ser monitoreada.

Ejemplo 1.19 Accesibilidad en el desarrollo de software.

Este es un caso de estudio ficticio presentado en [64]. La herramienta web All-Together es utilizadas por grupos comunitarios, organizaciones sin ánimo 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 remotos controlen plazos, compartan documentos, hagan un seguimiento de tareas y se comuniquen.

En la última versión, 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 función de edición en línea, al pasar el puntero por encima de un elemento, por ejemplo un evento en el calendario o una tareas, aparece un icono 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, con el objetivo de lograr 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 clave de accesibilidad. Dado que el ícono del lápiz sólo se muestra al pasar el puntero por encima, la función era inexistente para las personas que no pueden apuntar y hacer clic, incluidas las personas ciegas y con discapacidad motora. También se ha observado que las personas con discapacidades cognitivas pueden tener dificultades para encontrar controles ocultos, así como personas con pérdida de visión que utilizan la lupa de pantalla pueden tener dificultades para localizar y manejar los controles dentro de una

ventana gráfica ampliada, y las personas con movilidad o destreza motora limitadas pueden tener dificultades para interactuar con los controles dinámicos de 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 el requerimiento de conformidad con las Directrices de Accesibilidad al Contenido en la Web (WCAG) -por sus siglas en inglés, Web Content Accesibility Guidelines, para todos los contenidos y 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 preguntando: ¿Cuántas personas ciegas en verdad utilizan nuestro producto? La dirección de la empresa decidió que los clientes que utilizaban 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 con un correo electrónico de asistencia dedicado para ayudar a los usuarios que encontraran barreras de accesibilidad con 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 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 supone utilizar la versión actualizada con defectos de accesibilidad, algunos clientes decidieron volver a la versión anterior y esperar a que AllTogether corrigiera los defectos de accesibilidad y lanzara una nueva versión.

En este caso de estudio, la dirección de la empresa decidió lanzar la aplicación 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 équipo de atención al cliente para documentar los defectos de accesibilidad y aumentar el apoyo a los usuarios afectados. 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 discriminación por discapacidad, así como 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 software, costos que terminaron siendo superiores a los de abordar los requisitos de accesibilidad en el desarrollo de las funciones.

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 discapacitadas, 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 críticos de accesibilidad sobre el cumplimiento del calendario de lanzamiento. La inversión de AllTogether en servicio al cliente muestra el compromiso de salvaguardar la empresa y a sus clientes y usuarios. El Principio 1.b (Evitar causar daño) nos dirige 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 h. Si la práctica profesional se hubiera q por el Principio 1.d (Ser justo y tomar acciones para no discriminar), todos los implicados habrían reorientado sus esfuerzos hacia un enfoque y una aplicación accesibles. Si se hubiera hecho bien, una interfaz simplificada y organizadas habría beneficiado a las personas con discapacidad y mejorado la usabilidad en 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 el departamento de desarrollo de software de una empresa tecnológica de tamaño medio y que ahora es el programador jefe, es decir, alguien que tiene responsabilidades tanto de gestión como de programación. 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 toma Usted?

(a) Reporta lo que ha descubierto a su jefe inmediato superior.
(b) Reporta lo que ha descubierto directamente al competidor.
(c) Utiliza la vulnerabilidad para acceder a la información de clientes del competidor.
(d) No hace nada.

Ejercicios

Ejercicio 1.1 Analice el siguiente código Python [11] que recibe un número entero que representa el número de día dentro del año (1 es 1 de enero, 32 es 1 de febrero, 365 es 31 de diciembre de un año no bisiesto). El código devuelve el nombre del mes y el día dentro del mes que corresponde. El código recibe un segundo parámetro que indica si el año es bisiesto (en cuyo caso febrero tiene 29 días). El código debe lanzar la excepción ValueError si el número del día no es válido (menor que 1 o mayor que 366). Recorra el código con los siguientes pares de datos de prueba: 1 y Falso, 32 y Falso, 60 y Verdadero, 366 y Verdadero. ¿En que líneas de código encuentra defectos y cuáles son las correcciones necesarias?

```
1  class Month:
2        pass
3
4  def showday( daynumber, isleapyear ):
5
6  """ Muestra el mes y día dentro del mes del número de día del año.
7
8     daynumber: número de día del año.
9     isleapyear: Verdadero si el año es bisiesto.
10 """
11
12    months = [ "Enero", "Febrero", "Marzo",
13               "Abril", "Mayo", "Junio",
14               "Julio", "Agosto", "Septiembre",
15               "Octubre", "Noviembre", "Diciembre" ]
16
17    days = [ 31 for x in months ]
18
19    thirtylist = ( "Abril", "Junio",
20                   "Septiembre", "Noviembre" )
21
22    for j in [ months.index(k) for k in thirtylist]:
23          days[j] = 30
24
25    # Considerar año bisiesto  days[months.index("Febrero")] = \
26    28 + ((isleapyear and 1) or 0)
27
28    """ daymap contiene 12 objetios Month, cada uno     de los cuales tiene un
       part nombre/días.
29    """
30
```

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

Ejercicio 1.2 Escriba un código 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 menor que el de la derecha, en cuyo caso se resta. Ejercite su código con los siguientes pares de datos de prueba y salidas esperadas: 4 y IV, 90 y XC, 521 y DXXI.

Problemas

Problema 1.1 El departamento de fabricación de la empresa de autos Volkswagen está desarrollando una nueva tecnología para habilitar una función específica que los usuarios llevan pidiendo desde hace tiempo. 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 directivo sugiere hacer cambios en el software del vehículo para que detecte cuándo se están realizando pruebas reglamentarias y en ese caso se modifique el comportamiento del software para evitar el aumento del campo electromagnético y pasar con éxito las pruebas. ¿Qué hace usted en su calidad de dueño del producto? Para fundamentar su decisión, 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 considera 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 fracasarlas. ¿Qué hace usted como directivo de su empresa? Para fundamentar su decisión, explique cuáles valores del código de ética son relevantes para este caso.

Proyectos

1. Proyecto Open ERP para Cámara Nacional de Comercio

   La Cámara Nacional de Comercio es una entidad gremial privada, sin ánimo de lucro, encargada de fomentar el desarrollo empresarial del país. La Cámara Nacional de Comercio agrupa cientos de empresas medianas, pequeñas y muy pequeñas. La Cámara Nacional de Comercio ha decido financiar la implementación de un sistema de Planificación de Recursos Empresariales (ERP) - por sus siglas en inglés, Enterprise Resource Planing - para uso de sus socios. El Sistema ERP de la Cámara Nacional de Comercio debe cubrir las necesidades de las empresas socias para la gestión automatizada de inventario, compras, ventas, clientes, contabilidad, y recursos humanos. Investigue las necesidades básicas de inventario, ventas, clientes, y contabilidad 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 requisitos funcionales para cada uno de los siguientes módulos: inventario, ventas, contabilidad, y recursos humanos.

2. Proyecto App 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. El app 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 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 contrata para desarrollar un curso de inglés online para personas 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 periodicamente 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 asimilidad 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 (Honrar 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 dos defectos, uno en la línea 40 y otro en la línea 42. El código en la línea 40 tiene una validación incompleta. Este defecto se manifiesta al ingresar un número mayor que 366. Tal y como está escrito, el código no lanza la excepción ValueError. En su lugar, el código debería ser:

  if (daynumber > 0 and daynumber < 367):

  El código en la línea 42 utiliza el operador de comparación incorrecto. En su lugar, 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:

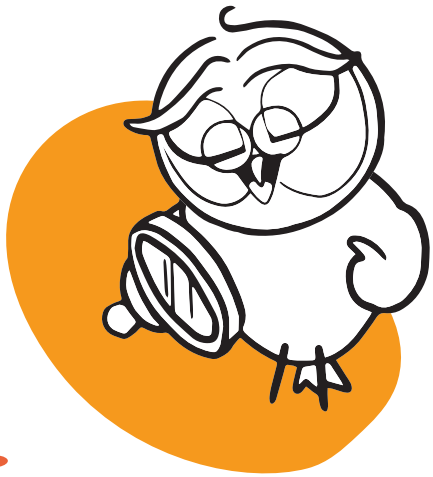
  if daynumber <= el.days:

- 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).

"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 Sanchez-Gordon, 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 Cía. Ltda.

Impreso por

ESCUELA POLITÉCNICA NACIONAL