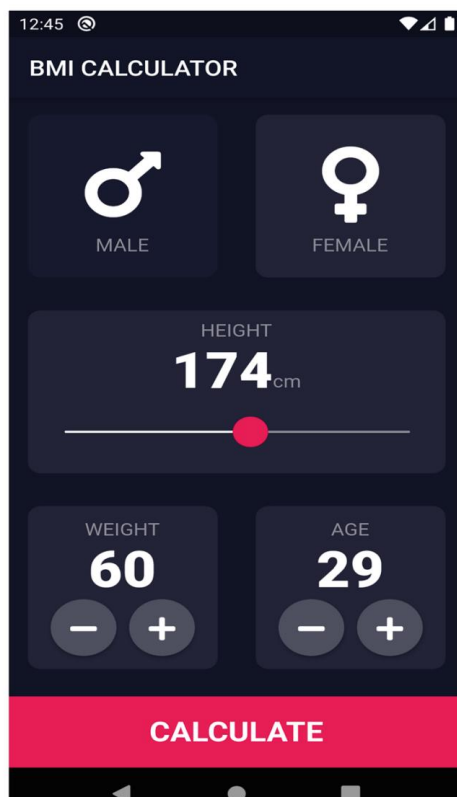


Fig. 4.16 Calculadora de IMC con una interfaz diferente

(fuente: <https://www.amazon.com/Meet-shingala-Bmi-calculator/dp/B08FJ4 8KYQ>)



Para intentar forzar una falla desde el elemento 1 de la lista, el evaluador puede intentar ingresar una cadena muy larga de dígitos en el campo de entrada "peso".⁴

Para intentar forzar un fallo en el punto 2 de la lista, el evaluador puede dar una altura cero, ya que el IMC se calcula como el cociente del peso y el cuadrado de la altura.

Para intentar forzar una falla del punto 3, el evaluador puede dar, por ejemplo, un valor de peso negativo para forzar un valor de IMC negativo (incorrecto).

Tenga en cuenta que estos ataques se pueden realizar para la aplicación anterior porque su interfaz permite ingresar valores de peso y altura directamente desde el teclado. Es una buena idea diseñar interfaces de manera que se evite la introducción de valores incorrectos. Por ejemplo, al usar campos con controles de rango de valores incorporados, permitimos al usuario ingresar la entrada solo a través de botones que permiten aumentar o disminuir los valores en un rango controlado. Otra idea es utilizar mecanismos como listas desplegables o controles deslizantes. La Figura 4.16 muestra una interfaz de este tipo para una aplicación de calculadora de IMC análoga.

⁴ La fórmula del IMC divide el peso por el cuadrado de la altura. Para obtener un desbordamiento de los valores del IMC, el numerador de la fracción debe ser lo más grande posible. Por lo tanto, no tiene sentido dar valores grandes para la altura, porque cuanto mayor es el denominador, menor es el IMC.

4.4.2 Pruebas exploratorias

Descripción de la técnica Las pruebas

exploratorias son el enfoque de diseñar, ejecutar y registrar pruebas no escritas (es decir, pruebas que no han sido diseñadas de antemano) y evaluarlas dinámicamente durante la ejecución. Esto significa que el evaluador no ejecuta pruebas preparadas en una fase separada de diseño o análisis, sino que cada uno de sus próximos pasos en el escenario de prueba actual es dinámico y depende de:

- Conocimiento e intuición del evaluador •

- Experiencia con esta o aplicaciones similares • La forma

- en que se comportó el sistema en el paso anterior del escenario

De hecho, la estricta división entre pruebas exploratorias y programadas no refleja del todo la verdad. Esto se debe a que cada actividad de prueba incluye algún elemento de planificación y algún elemento de dinámica y exploración. Por ejemplo, incluso en pruebas puramente exploratorias, es habitual planificar una sesión exploratoria con antelación, asignando, por ejemplo, un tiempo específico para que el evaluador la ejecute. Un evaluador exploratorio también puede preparar varios datos de prueba necesarios para la prueba antes de que comience la sesión.

Pruebas exploratorias basadas en sesiones

Las pruebas exploratorias basadas en sesiones son un enfoque que permite a los evaluadores y administradores de pruebas obtener una mayor estructuración de la actividad del evaluador, así como la posibilidad de gestionar mejor las actividades de pruebas exploratorias. Consta de tres pasos básicos:

1. El evaluador se reúne con el gerente para determinar el alcance de la prueba y asignar tiempo. El gerente entrega la carta de prueba (ver Tabla 4.13) al evaluador o la escribe en colaboración con el evaluador. La carta de prueba debe crearse durante el análisis de la prueba (ver Sección 1.4.3).
2. Realizar una sesión de prueba exploratoria por parte del evaluador, durante la cual toma notas de cualquier observación relevante (problemas observados, recomendaciones para el futuro, información sobre lo que falló durante la sesión, etc.).
3. Reunirse nuevamente al final de la sesión, donde se presentan los resultados de la misma. Se discuten y se toman decisiones sobre posibles próximos pasos.

Las pruebas exploratorias basadas en sesiones se llevan a cabo en un periodo de tiempo bien definido (generalmente de 1 a 4 h). El evaluador se concentra en la tarea especificada en la carta de prueba, pero, por supuesto, si es necesario, puede desviarse en cierta medida de la tarea esencial, en una situación en la que se observe algún defecto grave en otra área de la aplicación.

El evaluador documenta los resultados de la sesión en la carta de prueba.

Se recomienda realizar pruebas exploratorias basadas en sesiones de forma colaborativa, por ejemplo, mediante emparejamiento. Un evaluador puede asociarse con un representante comercial, un usuario final o un propietario del producto para explorar la aplicación mientras la prueba. Un evaluador también podría asociarse con un desarrollador para evitar probar funciones que son inestables, para abordar cierto problema técnico o para demostrar un comportamiento no deseado de inmediato, sin tener que proporcionar evidencia extensa en el informe de defectos.

Tabla 4.13 Carta de prueba

Carta de prueba—TE 02-001-01	
Meta	Pruebe la funcionalidad de inicio de sesión
Áreas	Inicie sesión como usuario existente con un nombre de usuario y contraseña correctos Inicie sesión como usuario existente a través de una cuenta de Google Inicie sesión como usuario existente a través de una cuenta de Facebook Inicio de sesión incorrecto: ningún usuario Inicio de sesión incorrecto: contraseña incorrecta Acciones que resultan con el bloqueo de la cuenta Uso de la función de recordatorio de contraseña Ataque de inyección SQL y otros ataques a la seguridad
Ambiente	Sitio accesible a través de varios navegadores (Chrome, FF, IE) 2019-06-12, 11:00–
Tiempo	13:15
Ensayador	Bob cazador de insectos
Notas del probador	
Archivos	(Capturas de pantalla, datos de pruebas, etc.).
Defectos encontrados	
división del tiempo	20% preparación para la sesión 70% conducción de la sesión 10% análisis de problemas

^a La inyección SQL es un tipo de ataque que implica la llamada inyección de un código malicioso. Es un ataque a la seguridad al insertar una sentencia SQL maliciosa en un campo de entrada con la intención de ejecutarla.

¿Cuándo utilizar pruebas exploratorias?

Las pruebas exploratorias serán una solución buena, efectiva y eficiente si se cumplen una o más de las siguientes premisas:

- Las especificaciones del producto bajo prueba están incompletas, son de mala calidad o inexistentes.
- Hay presión de tiempo; Los evaluadores tienen poco tiempo para realizar las pruebas.
- Los evaluadores conocen bien el producto y tienen experiencia en pruebas exploratorias.

Las pruebas exploratorias están fuertemente relacionadas con la estrategia de prueba reactiva (ver Sección 5.1.1). Las pruebas exploratorias pueden utilizar otras técnicas de caja negra, de caja blanca y basadas en la experiencia. Nadie puede dictarle al evaluador cómo llevar a cabo la sesión. Si, por ejemplo, al evaluador le resulta útil crear un diagrama de transición de estado que describa cómo funciona el sistema y luego, en una sesión exploratoria, diseña casos de prueba y los ejecuta, tiene derecho a hacerlo. Este es un ejemplo del uso de pruebas programadas como parte de pruebas exploratorias.

Ejemplo La organización planea probar la funcionalidad básica de un sitio web. Se tomó la decisión de realizar pruebas exploratorias y se preparó para el evaluador una carta de prueba que se muestra en la Tabla 4.13. El administrador de pruebas, basándose en los resultados recopilados de múltiples sesiones de pruebas exploratorias, deriva las siguientes métricas de muestra para su posterior análisis:

• Número de sesiones realizadas y completadas • Número de defectos reportados • Tiempo dedicado a prepararse para la sesión • Tiempo de la sesión de prueba real • Tiempo dedicado a analizar problemas • Número de funcionalidades cubiertas

Tours de pruebas exploratorias

Andrew Whittaker [55] propone un conjunto de enfoques de pruebas exploratorias inspirados en las visitas turísticas y el turismo. Esta metáfora permite a los evaluadores aumentar su creatividad al realizar sesiones exploratorias. Los objetivos de estos enfoques son:

- Comprender cómo funciona la aplicación, cómo es su interfaz, qué funcionalidad ofrece al usuario • Obligar al software a demostrar sus capacidades • Encontrar defectos

La metáfora del turista permite dividir las áreas de software de manera análoga a las partes de la ciudad visitadas por el turista:

- Distrito de negocios—corresponde a aquellas partes del software que están relacionadas con su parte "negocio", es decir, las funcionalidades y características que el software ofrece a los usuarios.
- Distrito histórico (centro histórico): corresponde al código heredado y al historial de funcionalidad defectuosa. • Distrito turístico: corresponde a aquellas partes del software que atraen a nuevos usuarios (turistas), funciones que es poco probable que un usuario avanzado (residente de la ciudad) vuelva a utilizar.
- Distrito de entretenimiento: corresponde a funciones y características de soporte relacionados con la usabilidad y la interfaz de usuario.
- Distrito hotelero: el lugar donde descansa el turista; Corresponde a los momentos en los que el usuario no utiliza activamente el software pero el software aún hace su trabajo.
- Barrios sospechosos: lugares donde es mejor no aventurarse; en el software, corresponden a lugares donde se pueden lanzar varios tipos de ataques contra la aplicación.

Whittaker describe una variedad de tipos de exploración (turismo) para cada distrito.

Para obtener más información sobre las pruebas exploratorias, consulte [56, 57].

4.4.3 Pruebas basadas en listas de verificación

Descripción de la técnica Las pruebas

basadas en listas de verificación, al igual que las dos técnicas anteriores descritas en esta sección, utilizan el conocimiento y la experiencia del evaluador, pero la base para la ejecución de la prueba son los elementos contenidos en la llamada lista de verificación. La lista de verificación contiene las condiciones de prueba que se deben verificar. La lista de verificación no debe contener elementos que puedan verificarse automáticamente, elementos que funcionen mejor como criterios de entrada/salida o elementos que sean demasiado generales [58].

Las pruebas basadas en listas de verificación pueden parecer una técnica similar a los ataques de fallas. La diferencia entre estas técnicas es que un ataque de falla comienza a partir de defectos y fallas, y la acción del evaluador es revelar problemas, dada una falla o defecto en particular.

En las pruebas basadas en listas de verificación, el evaluador también actúa de forma sistemática, pero verifica las características "positivas" del software. La base de prueba en esta técnica es la propia lista de verificación.

Los elementos de la lista de verificación a menudo se formulan en forma de pregunta. La lista de verificación debería permitirle verificar cada uno de sus elementos por separado y directamente. Los elementos de la lista de verificación pueden hacer referencia a requisitos, características de calidad u otras formas de condiciones de prueba. Se pueden crear listas de verificación para admitir varios tipos de pruebas, incluidas pruebas funcionales y no funcionales (por ejemplo, 10 heurísticas para pruebas de usabilidad [59]).

Algunos elementos de la lista de verificación pueden volverse gradualmente menos efectivos con el tiempo a medida que quienes la desarrollan aprenden a evitar cometer los mismos errores. También es posible que sea necesario agregar nuevos elementos para reflejar defectos de alta gravedad que se hayan descubierto recientemente. Por lo tanto, las listas de verificación deben actualizarse periódicamente en función del análisis de defectos. Sin embargo, se debe tener cuidado de que la lista de verificación no sea demasiado larga [60].

En ausencia de casos de prueba detallados, las pruebas basadas en listas de verificación pueden proporcionar cierto grado de coherencia a las pruebas. Dos evaluadores que trabajen con la misma lista de verificación probablemente realizarán su tarea de manera ligeramente diferente, pero en general probarán las mismas cosas (las mismas condiciones de prueba), por lo que necesariamente sus pruebas serán similares. Si las listas de verificación son de alto nivel, es probable que haya cierta variabilidad en las pruebas reales, lo que dará como resultado una cobertura potencialmente mayor pero una menor repetibilidad de las pruebas.

Tipos de listas de verificación

Existen muchos tipos diferentes de listas de verificación para diferentes aspectos del software. Además, las listas de verificación pueden tener distintos grados de generalidad y un campo de aplicación más amplio o más limitado. Por ejemplo, las listas de verificación para el uso de pruebas de código (por ejemplo, en pruebas de componentes) tenderán a ser muy detalladas y normalmente incluirán muchos detalles técnicos sobre aspectos del desarrollo de código en un lenguaje de programación particular. Por el contrario, una lista de verificación para pruebas de usabilidad puede ser de muy alto nivel y general. Por supuesto, esto no es una regla: los evaluadores siempre deben ajustar el nivel de detalle de la lista de verificación para adaptarla a sus propias necesidades.

Las listas de

verificación de aplicaciones se pueden utilizar básicamente para cualquier tipo de prueba. En particular, pueden aplicarse a pruebas funcionales y no funcionales.

En las pruebas basadas en listas de verificación, el evaluador diseña, implementa y ejecuta pruebas para cubrir las condiciones de prueba que se encuentran en la lista de verificación. Los evaluadores pueden utilizar listas de verificación existentes (por ejemplo, disponibles en Internet) o pueden modificarlas y adaptarlas a sus necesidades. También pueden crear dichas listas ellos mismos, basándose en su propia experiencia y la de su organización con defectos y fallas, su conocimiento de las expectativas de los usuarios del producto desarrollado o su conocimiento de las causas y síntomas de las fallas del software. Hacer referencia a la propia experiencia puede hacer que esta técnica sea más eficaz, porque normalmente, las mismas personas, en la misma organización, trabajando en productos similares, cometerán errores similares. Normalmente, los evaluadores menos experimentados trabajan con listas de verificación ya existentes.

Cobertura

Para una prueba basada en una lista de verificación, no se definen medidas específicas de cobertura. Por supuesto, como mínimo, se debe cubrir cada elemento de la lista de verificación. Sin embargo, dado que es difícil saber hasta qué punto se ha cubierto dicho elemento (debido al hecho de que la técnica apela al conocimiento, la experiencia y la intuición del evaluador individual), esto tiene ventajas y desventajas.

La desventaja, por supuesto, es la falta de información detallada sobre la cobertura y una menor repetibilidad que con las técnicas formales. La ventaja, por otro lado, es que se puede lograr una mayor cobertura si dos o más evaluadores utilizan la misma lista de pruebas.

Esto se debe a que lo más probable es que cada uno de ellos realice un conjunto de pasos ligeramente diferente para cubrir el mismo elemento de la lista de verificación. Así, habrá cierta variabilidad en las pruebas, lo que se traducirá en una mayor cobertura pero a costa de una menor repetibilidad.

Ejemplo A continuación presentamos dos listas de verificación de muestra. El primero contiene las denominadas heurísticas de usabilidad de Nielsen. Junto a cada elemento de la lista de verificación, se proporciona además un comentario entre paréntesis, que puede ayudar al evaluador con las pruebas.

Heurísticas de Nielsen para la usabilidad del sistema

1. Visibilidad del estado del sistema: ¿se muestra el estado del sistema en cada punto de su funcionamiento? (El usuario siempre debe saber dónde se encuentra; el sistema debe utilizar las llamadas rutas de navegación, que muestran el camino que ha seguido el usuario, así como títulos claros para cada pantalla)
2. Coincidencia entre el sistema y el mundo real: ¿Existe compatibilidad entre el sistema y la realidad? (La aplicación no debe utilizar un lenguaje técnico sino un lenguaje sencillo utilizado todos los días por los usuarios del sistema, relacionado con el dominio empresarial en el que opera el programa)
3. Control y libertad del usuario: ¿tiene el usuario control sobre el sistema? (por ejemplo, debería ser posible deshacer una acción que el usuario realizó por error, como eliminar del carrito de compras un producto colocado allí por error)
4. Coherencia y estándares: ¿mantiene el sistema coherencia y estándares? (Las soluciones de apariencia deben ser consistentes en toda la aplicación, por ejemplo, el mismo formato de enlaces, uso de la misma fuente; también se deben utilizar enfoques familiares, por ejemplo, colocar el logotipo de la empresa en la esquina superior izquierda de la pantalla)

5. Prevención de errores: ¿el sistema previene adecuadamente los errores? (El usuario no debe tener la impresión de que algo ha salido mal; por ejemplo, no debemos darle la opción al usuario de seleccionar una versión de un producto que no está disponible en la tienda, sólo para ver más tarde un error de "no stock").
6. Reconocimiento en lugar de recuerdo: ¿el sistema te permite seleccionar en lugar de obligarte a recordar? (por ejemplo, las descripciones de los campos en un formulario no deben desaparecer cuando comienza a completarlos; esto sucede cuando la descripción se coloca inicialmente dentro de ese campo)
7. Flexibilidad y eficiencia de uso: ¿el sistema proporciona flexibilidad y eficiencia? (por ejemplo, las opciones de búsqueda avanzada deberían estar ocultas de forma predeterminada si la mayoría de los usuarios no las utilizan)
8. Diseño estético y minimalista: ¿el sistema es estéticamente agradable y no está sobrecargado de contenido? (La aplicación debe atraer a los usuarios; debe tener un buen diseño, combinación de colores adecuada, disposición de los elementos de la página, etc.).
9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores: ¿se proporciona un manejo eficaz de errores? (una vez que ocurre un error, el usuario no debe recibir un mensaje técnico al respecto o que es culpa del usuario; también debe haber información sobre lo que debe hacer el usuario en esta situación)
10. Ayuda y documentación: ¿hay ayuda disponible en el sistema? ¿Existe documentación de usuario? (Algunos usuarios necesitarán una función de ayuda; dicha opción debería estar disponible en la aplicación; también debería incluir información de contacto para soporte técnico)

El segundo ejemplo de lista de verificación se refiere a la revisión del código. Esta lista, a modo de ejemplo, incluye sólo aspectos técnicos seleccionados de buenas prácticas de escritura de código y está lejos de ser completa.

Lista de verificación para inspecciones de códigos


1. ¿El código está escrito según los estándares actuales?
2. ¿El código tiene un formato coherente?
3. ¿Hay funciones que nunca se llaman?
4. ¿Son eficientes los algoritmos implementados y cuentan con recursos computacionales adecuados?
¿complejidad?
5. ¿Se utiliza la memoria de forma eficaz?
6. ¿Existen variables que se utilizan sin haber sido declaradas primero?
7. ¿Está el código debidamente documentado?
8. ¿Es coherente la forma de comentar?
9. ¿Toda operación de división está protegida contra la división por cero?
10. En las declaraciones IF-THEN, ¿son los bloques de declaraciones que se ejecutan con mayor frecuencia?
comprobado primero?
11. ¿Cada declaración CASE tiene un bloque predeterminado?
12. ¿Se libera alguna memoria asignada cuando ya no está en uso?

4.5 Enfoques de prueba basados en la colaboración

FL-4.5.1 (K2) Explicar cómo escribir historias de usuarios en colaboración con desarrolladores y representantes empresariales.

FL-4.5.2 (K2) Clasificar las diferentes opciones para redactar criterios de aceptación.

FL-4.5.3 (K2) Utilice el desarrollo basado en pruebas de aceptación (ATDD) para derivar pruebas casos.

La popularidad de las metodologías ágiles ha llevado al desarrollo de métodos de prueba específicos para este enfoque, teniendo en cuenta los artefactos utilizados por estas metodologías y enfatizando la colaboración entre clientes (negocios), desarrolladores y evaluadores. Este capítulo analiza las siguientes cuestiones relacionadas con las pruebas en el contexto de metodologías ágiles, utilizando un enfoque de prueba basado en la colaboración: 

- Historias de usuarios como contraparte ágil de los requisitos del usuario (Sección [4.5.1](#))
- Criterios de aceptación como contraparte ágil de las condiciones de prueba, proporcionando la base para el diseño de pruebas (Sección [4.5.2](#))
- Desarrollo basado en pruebas de aceptación (ATDD) como una forma de prueba de alto nivel (pruebas de sistema y pruebas de aceptación) que se utiliza a menudo en metodologías ágiles, basadas en un enfoque de "prueba primero" (Sección [4.5.3](#)).

Cada una de las técnicas descritas en las Secciones [4.2](#), [4.3](#) y [4.4](#) tienen un objetivo específico con respecto a la detección de defectos de un tipo específico. Los enfoques colaborativos, por otro lado, también se centran en evitar defectos mediante la cooperación y la comunicación.

4.5.1 Escritura colaborativa de historias de usuario

En el desarrollo ágil de software, una historia de usuario representa un incremento funcional que será de valor para el usuario, el comprador del sistema o software, o cualquier otra parte interesada. Las historias de usuarios se escriben para capturar los requisitos desde la perspectiva de desarrolladores, evaluadores y representantes comerciales. En los modelos SDLC secuenciales, esta visión compartida de una característica o función de software específica se logra mediante revisiones formales después de que se hayan escrito los requisitos. En los enfoques ágiles, por otro lado, la visión compartida se logra a través de revisiones informales frecuentes durante la redacción de requisitos o escribiendo requisitos juntos, de manera colaborativa, por parte de evaluadores, analistas, usuarios, desarrolladores y cualquier otra parte interesada. Las historias de usuario constan de tres aspectos, conocidos como las "3C":

- Tarjeta
- Conversación •
- Confirmación