Diseño de Casos de Pruebas a partir del Léxico Extendido del Lenguaje y la Especificación de Requerimientos

Julieta Screpnik[1][0009-0002-0787-1856], María Alejandra Paz Menvielle[2][0000-0002-8317-4078] y Leandro Antonelli[3][0000-0003-1388-0337]

[1]Fac. de Informática, UNLP, La Plata, Bs As, Argentina
[2]Fac. Regional Córdoba,UTN, Córdoba, Argentina
[3]Lifia, Fac. de Informática, UNLP, La Plata, Bs As, Argentina
[3]CAETI, Fac. de Tecnología Informática,UAI, Bs As, Argentina
julietascrepnik@info.unlp.edu.ar
mpaz@frc.utn.edu.ar
lanto@lifia.info.unlp.edu.ar

Resumen. La prueba de software es una fase crítica en el ciclo de desarrollo del software, ya que consiste en un conjunto de actividades diseñadas para evaluar la calidad de los productos que se construyen. Además, los datos obtenidos durante esta etapa fortalecen la confianza en el producto final y aportan información valiosa para respaldar la toma de decisiones por parte de todos los interesados. En la actualidad, el incremento de la complejidad de los productos que solicitan los clientes exige la implementación de estrategias innovadoras, orientadas a identificar la mayor cantidad de defectos y disminuir el riesgo de fallas. En este artículo, se propone un método para el proceso de diseño de casos de pruebas en formato Gherkin, tomando como entrada la Especificación de Requerimientos de Software y el Léxico Extendido del Lenguaje.

Palabras claves: Diseño de Casos de Pruebas, Léxico Extendido del Lenguaje, Especificación de Requerimientos de Software, Gherkin.

1 Introducción

En la actualidad, la industria del software enfrenta el desafío de construir productos de calidad en el plazo y presupuesto establecido. Asimismo, la demanda de software en diferentes áreas y la complejidad del producto que se pide crece rápidamente. El impacto del software en nuestra sociedad y en la cultura continúa siendo profundo. Al mismo tiempo que crece su importancia, la comunidad del software trata continuamente de desarrollar tecnologías que hagan más sencillo, rápido y menos costoso la construcción de programas de computadora de alta calidad [1].

La calidad no es un concepto nuevo, sino que ha sido estudiado a lo largo de la historia por diversos exponentes en el área. Según Jerry Weinberg “la calidad es valor para una persona”, lo cual fue extendido por Cem Kaner diciendo que “la calidad es valor para una persona a la que le interesa”[2]. La Real Academia Española (RAE)[3], por su parte, define a la calidad como una propiedad o conjunto de propiedades

Proceedings of the 28th Workshop on Requirements Engineering (WER2025),
August 20-22, 2025, Rio de Janeiro-RJ, Brazil.
https://doi.org/10.29327/1588952.28-7

inherentes a algo, que permiten juzgar su valor. A su vez, ISO 9000 [4] especifica que la calidad es el grado en que un conjunto de características inherentes a un objeto (producto, servicio, proceso, persona, organización, sistema o recurso) cumple con los requisitos. De acuerdo a la IEEE [5], la calidad de software es "el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario". La tabla 1 presenta los atributos de calidad de software definidos por Boehm [6].

Tabla 1. Atributos de la calidad de software.

Protección	Comprensibilidad	Portabilidad
Seguridad	Comprobabilidad	Usabilidad
Fiabilidad	Adaptabilidad	Reusabilidad
Flexibilidad	Modularidad	Eficiencia
Robustez	Complejidad	Facilidad para que el usuario aprenda a utilizarlo

Por consiguiente, la calidad del software no sólo se trata de si la funcionalidad de éste se implementó correctamente, sino también depende de los atributos no funcionales del sistema [6].

Deming [7] señala que la dificultad para definir la calidad reside en la traducción de las necesidades futuras del usuario a características conmensurables, de forma que el producto se pueda diseñar y fabricar proporcionando satisfacción por el precio que tenga que pagar el usuario. Estas necesidades son altamente volátiles. Siguiendo esta línea, se puede afirmar que la calidad es subjetiva y está influenciada por diferentes factores como la edad, el contexto en el que está inmerso y el tiempo. Además, el cliente a menudo no sabe exactamente lo que quiere o espera del producto o servicio que se está construyendo hasta que lo tiene en mano, por lo que hace que la calidad sea bastante difícil de alcanzar. Asimismo, la calidad es relativa incluso para una misma persona que puede cambiar su perspectiva respecto de lo que para él es calidad a lo largo del tiempo. Estas razones hacen que la calidad sea un concepto difícil de definir y mucho más difícil de medir para determinar si un producto o servicio tiene o no calidad.

Trabajar con requerimientos en lenguaje natural hace que esto sea mucho más complejo [8]. Los requerimientos en lenguaje natural, por otro lado, contienen problemas como ambigüedad, falta de coherencia, datos inexactos e incompletos [9]. Los requerimientos de cualquier proyecto de software sirven como su base; sin requisitos claros, no se pueden realizar pruebas, y un software de calidad debe cumplir con sus especificaciones requeridas [10]. Además, los errores en los requerimientos representan el 56% de todos los problemas en el software. La razón más común del fracaso de proyectos de programación es la falta de requerimientos bien definidos, los cuales suelen redactarse en inglés simple u otros lenguajes naturales. La presencia de ambigüedad es una de las principales características esenciales de los requerimientos en lenguaje natural [11]. En consecuencia, la fase de pruebas es crucial, ya que garantiza que se construye el producto de software correcto, de acuerdo con los procesos y estándares que posee la organización y las necesidades del cliente reflejadas en los requerimientos.

Asimismo, los casos de prueba que se generan suelen ser incompletos e inconsistentes, ya que no verifican exhaustivamente el comportamiento del sistema. En muchas empresas, los probadores están mal preparados para ejecutar la difícil tarea de ensayar los productos software, cada vez más complejos [12]. Esto se debe a que el equipo de testers, en muchos casos, no comprenden en profundidad los requerimientos especificados ni poseen el conocimiento suficiente del dominio de la aplicación. Cabe destacar que, aunque se han realizado numerosas investigaciones en el área del testing a lo largo de los años, los autores no conocen métodos o técnicas centradas en los requerimientos que aborden el problema que se intenta resolver, así como las necesidades y expectativas del cliente sobre el producto.

Este artículo presenta un método innovador para el diseño de casos de pruebas en formato Gherkin, tomando como entrada la Especificación de Requerimientos de Software y el Léxico Extendido del Lenguaje. La Especificación de Requerimientos de Software es un documento que detalla los aspectos fundamentales del software que se va a construir, con el objetivo de ayudar a que todas las partes interesadas tengan un entendimiento común del sistema a desarrollar y evitar posibles malentendidos [13]. En cambio, el Léxico Extendido del Lenguaje [14] es un modelo que permite representar y documentar, con tecnología hipertextual un conjunto de símbolos que representan el lenguaje del dominio, sin necesidad de entender el problema. Consiste en una descripción de los términos significativos del macrosistema, acotando el lenguaje externo con el uso de símbolos definidos en el mismo LEL y a su vez, minimiza el uso de símbolos externos al lenguaje del dominio.

Es relevante señalar que este enfoque otorga una participación significativa al cliente a lo largo de todo el proceso, ya que su conocimiento del sistema que se desea construir resulta indispensable para la creación de casos de prueba de alta calidad. La colaboración del cliente permite definir los escenarios principales y críticos que deben ser abordados en la etapa de prueba, contribuyendo así a la calidad final del producto.

El resto del artículo está organizado de la siguiente manera. La sección 2 describe algunos conceptos básicos que sirven para entender el método propuesto. La sección 3 describe en detalle nuestra propuesta. En la sección 4 se presentan los trabajos relacionados. Finalmente, en la sección 5 se presentan las conclusiones y trabajos futuros.

2 Marco conceptual

En esta sección, se presentan algunos conceptos básicos que permiten entender el método propuesto.

2.1 Especificación de Requerimientos de Software

La Especificación de Requerimientos de Software (ERS) es un documento que proporciona una descripción detallada de todos los aspectos del software que se va a elaborar, antes de que el proyecto comience [1]. Este documento es elaborado por el ingeniero de requerimientos durante la fase de análisis, en el cual el cliente

desempeña un rol fundamental al comunicar sus expectativas y necesidades, definiendo así el alcance del producto de software a desarrollar.

2.2 Template de Gherkin para especificar casos de prueba

El Desarrollo Guiado por Comportamiento (BDD, Behavior-Driven Development) es un enfoque ágil para el desarrollo de software cuyo objetivo principal es promover la comunicación y la colaboración entre todos los interesados en el producto, incluidos desarrolladores, analistas, testers y clientes, mediante el uso de un lenguaje común. Este enfoque cuenta con diversos lenguajes, entre los cuales el más utilizado es Gherkin. Gherkin es un lenguaje cuasi-natural con muy poca estructura que se utiliza en la industria del desarrollo de software. Esto permite que el cliente describa el comportamiento del sistema en mente de una manera que los desarrolladores puedan entender [15].

Una especificación en Gherkin consiste en una serie de características que, a su vez, están compuestas por escenarios escritos en declaraciones (conocidas como pasos) que comienzan con las palabras clave Given (Dado), When (Cuando), Then (Entonces) y And (Y). La palabra clave Given se utiliza para describir una precondición, es decir, el contexto en el cual el escenario es relevante. La declaración When describe la acción que puede realizarse cuando se cumple la precondición, mientras que la declaración Then describe la postcondición que debería cumplirse tras la acción. Finalmente, la palabra clave And se usa como un mecanismo de enlace cuando el autor de un escenario necesita dividir una precondición, acción o postcondición en varios pasos [15]. En la Fig. 1 se presenta un ejemplo de escenario de prueba para el alta de un paquete en el sector bancario.

```
Feature: Generar alta de paquete
Scenario: Generar alta de paquete exitosa
 Given el cliente existe en la base de datos del banco
 And el cliente posee al menos una cuenta bancaria de tipo caja de ahorro habilitada con estado activa
 And el cliente tiene una oferta disponible para el alta de un paquete y una tarjeta de crédito asociada
 When el analista de sucursal accede a la opción de menú "Alta Solicitud"
 And selecciona un paquete disponible de la lista según la oferta vigente
 Then el sistema muestra un mensaje de confirmación "solicitud de paquete creada con éxito"
```

Fig. 1. Ejemplo de escenario de prueba siguiendo el template de Gherkin.

2.3 Léxico Extendido del Lenguaje

El Léxico Extendido del Lenguaje (LEL, Language Extended Lexicon) [14] es una representación de los símbolos del lenguaje del dominio de la aplicación que intenta capturar el vocabulario de una aplicación. Su objetivo principal es que el ingeniero de software entienda el lenguaje que habla el usuario, entendiendo los términos que utiliza. Los símbolos son, en general, las palabras o frases utilizadas por el usuario y que repite con más frecuencia. También se incluyen aquellas palabras o frases que son relevantes para el dominio del problema más allá de su frecuencia de repetición. Se genera una lista con todos los símbolos reconocidos. La semántica de cada símbolo se representa con una o más nociones y uno o más impactos. La noción indica qué es el

símbolo y el impacto cómo repercute en el sistema. Por lo tanto, cada símbolo tiene un "nombre" que lo identifica, una "noción" y un "impacto" que lo describen [16]. Cada símbolo del LEL pertenece a una de las siguientes categorías que se detallan en la tabla 2:

Tabla 2. Categorías del LEL [17].

Categoría	Características	Noción	Impacto
Sujeto	Elementos activos que realizan acciones	Características o condición que satisface el sujeto	Acciones que el sujeto realiza
Objeto	Elementos pasivos sobre los cuales los sujetos realizan acciones	Características o atributos que tiene el objeto	Acciones que se realizan sobre el objeto
Verbo	Acciones que los sujetos realizan sobre los objetos	Objetivo que persigue el verbo	Pasos necesarios para completar la acción
Estado	Situaciones en las que pueden encontrarse sujetos y objetos	Situación representada	Acciones que deben realizarse para cambiar a otro estado

3 Propuesta

En esta sección, se presenta de manera general el método propuesto, seguido de una descripción detallada de cada uno de los pasos involucrados en el mismo.

3.1 Resumen del método

Este artículo propone un método secuencial que permite la generación de casos de pruebas en formato Gherkin, utilizando como entrada requerimientos funcionales del sistema en lenguaje natural descritos en la Especificación de Requerimientos de Software y símbolos definidos en el Léxico Extendido del Lenguaje. Como salida se obtiene un conjunto de casos de prueba expresados en lenguaje natural y en formato Gherkin. El método se estructura en los siguientes pasos: aplicación de sugerencias para mejorar el LEL, generación de casos de prueba y aplicación de sugerencias para mejorar los casos de pruebas. La Fig. 2 resume el método propuesto.

Este proceso permite generar casos de prueba mediante una correspondencia sintáctica y semántica entre los requerimientos descritos en la Especificación de Requerimientos y los símbolos detallados en el Léxico Extendido del Lenguaje. Además, los símbolos seleccionados del Léxico Extendido del Lenguaje ayudan a identificar los elementos principales que deben ser testeados y que formarán parte de los escenarios de pruebas. Por último, la información proporcionada en los campos

"noción" e "impacto" de los símbolos extraídos del Léxico Extendido del Lenguaje aporta mayor nivel de detalle y precisión en el diseño de los casos de prueba para lograr una cobertura más completa.

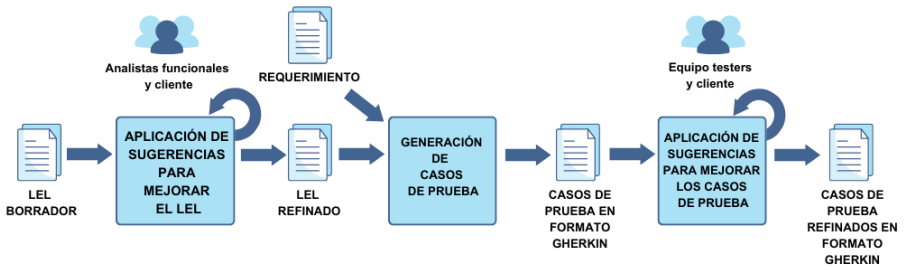

Fig. 2. Método propuesto.

3.2 Aplicación de sugerencias para mejorar el LEL

El primer paso consiste en refinar los símbolos capturados en el Léxico Extendido del Lenguaje con el objetivo de tener un dominio del lenguaje más específico y con el mayor detalle posible que ayude luego al siguiente paso de generación de casos de prueba. En este punto, intervienen los analistas funcionales y el cliente, quienes poseen el conocimiento del dominio. El Léxico Extendido del Lenguaje refinado obtenido como salida en este paso, tiene como propósito captar conocimiento del dominio del negocio y conocimiento de la aplicación que se va a construir.

Este paso sigue un enfoque iterativo, permitiendo a los analistas funcionales y al cliente refinar progresivamente el LEL a través de múltiples ciclos hasta alcanzar el nivel de precisión deseado. A continuación, se presenta en la tabla 3 sugerencias para mejorar el LEL:

Tabla 3. Sugerencias para mejorar el LEL.

Categoría	Noción	Impacto
Sujeto	¿Cuál es el objetivo del sujeto? ¿porque lo hace? ¿Qué características esenciales debe tener? ¿Qué características no debe tener? ¿Puede asumir distintos roles o jerarquías dentro del sistema? ¿Se clasifican por prioridad?	¿Qué acciones generan efectos críticos en el sistema? ¿Cómo se espera que el sujeto interactúe con el sistema? ¿Existen acciones del sujeto que cambien el estado de otros elementos en el sistema? ¿Qué cambios deberían producir sus acciones sobre el sistema?
Objeto	¿Qué características esenciales debe tener? ¿Qué características no debe tener? ¿Tiene alguna clasificación o categorización?	¿Qué consecuencias tienen las acciones del sujeto en el objeto? ¿Cómo afecta la presencia o ausencia del objeto al flujo del sistema?

	¿Existen dependencias entre los objetos? ¿Se clasifican por prioridad?	¿Qué cambios debe sufrir el objeto después de cada interacción con el sujeto? ¿Cómo debería reaccionar el sistema si el objeto está en un estado incorrecto o incompleto?
Verbo	¿Cómo queremos hacer esa acción? Pensando en Requerimientos no funcionales ¿Existen acciones similares o relacionadas? como si fueran una variante de una misma acción ¿Existen excepciones o limitaciones? ¿Existen operaciones que sean críticas o principales en el sistema? ¿Qué precondiciones deben cumplirse para realizar la acción?	¿Qué criterios se deben cumplir para considerar que la acción ha sido realizada con éxito? ¿Cuándo podríamos considerar que la acción falló? ¿Qué impacto tiene la ejecución de una acción determinada en los estados de los elementos afectados? ¿Qué debería suceder si la acción se interrumpe inesperadamente?
Estado	¿Puede haber subestados en el sistema? ¿Puede haber estados que dependen de otros estados? ¿Hay situaciones que provocan un cambio de estado a varios componentes del sistema? ¿Existen estados intermedios?	¿Cómo se valida que el sistema ha alcanzado un estado específico? ¿Qué efecto tiene en el sistema el cambio a un estado específico? ¿Qué condiciones son necesarias para que se produzca un cambio de estado? ¿Qué consecuencias tiene que el sistema permanezca en un estado determinado durante un tiempo prolongado? ¿Qué consecuencias hay si el sistema no logra cambiar a un estado específico?

Como sugerencia principal, se recomienda redactar oraciones completas para cada símbolo del LEL, incluyendo sujetos y objetos con el mayor nivel de detalle posible. Esto permitirá obtener información más completa y facilitará la generación de casos de prueba en el paso siguiente.

3.3 Generación de casos de prueba

Este paso podría ser automatizado mediante el uso de una herramienta con soporte de la inteligencia artificial, que toma como entrada la Especificación de Requerimientos de Software y el Léxico Extendido del Lenguaje refinado en el paso anterior. El proceso se estructura en una serie de subpasos que permiten identificar los elementos del LEL que requieren especial atención y que, posteriormente, se usarán para crear los casos de prueba. Los subpasos son los siguientes:

Selección del requerimiento: la norma IEEE 830-1998 [18], establece que los requerimientos deben describir claramente lo que el sistema de software debe hacer. Por consiguiente, se selecciona un requerimiento funcional de la Especificación de Requerimientos de Software para servir como base en los siguientes subpasos.

Identificación de símbolos LEL: es fundamental comprender la relación existente entre los símbolos del LEL y los requerimientos. Los verbos principales en los requerimientos suelen estar asociados con el nombre del símbolo verbo en el LEL, ya que estos expresan acciones realizadas por un sujeto que interactúa con el sistema. De forma similar, los sustantivos que aparecen en los requerimientos suelen vincularse con los nombres de los símbolos objeto y sujeto en el LEL, reflejando elementos que interactúan o son utilizados en el sistema. En este contexto, los sujetos representan actores que usan el sistema, mientras que los objetos son componentes internos del sistema. Además, las condiciones descritas en los requerimientos pueden incorporarse en la noción de los símbolos objeto en el LEL, ya que estas condiciones frecuentemente se refieren a atributos específicos de componentes o entidades del sistema. Por otro lado, algunos sustantivos presentes en los requerimientos pueden enriquecer la noción de los símbolos sujeto, aludiendo a características inherentes a los actores en cuestión. De este modo, se realiza un análisis y selección de los símbolos LEL para identificar aquellos que contienen palabras coincidentes con los términos presentes en el requerimiento elegido previamente, buscando una correspondencia sintáctica y semántica entre ambos artefactos.

Extracción de atributos clave: los términos (símbolos) se definen a través de dos atributos: noción e impacto. La noción describe las características intrínsecas y sustanciales del símbolo (denotación), mientras que el impacto (connotación) describe el vínculo entre el término que se describe y otros [19]. En consecuencia, se extraen las columnas noción e impacto de los símbolos seleccionados en el paso anterior para lograr una cobertura más completa y detallada en la generación de casos de prueba.

Clasificación en elementos de prueba: existe una relación directa entre los casos de prueba y el LEL. La noción de los sujetos y objetos que participan en el impacto del símbolo verbo en el LEL pueden incorporarse en las precondiciones de un caso de prueba. Asimismo, los pasos descritos en el impacto del símbolo verbo en el LEL, junto con los sujetos y objetos relacionados, se reflejan en los pasos de ejecución de un caso de prueba. La noción del símbolo verbo en el LEL puede asociarse al resultado esperado de un caso de prueba, mientras que el nombre del símbolo verbo, junto con el sujeto y el objeto implicados, puede utilizarse en el título de un caso de prueba. Adicionalmente, el requerimiento también mantiene una relación significativa con el caso de prueba: los sustantivos y condiciones especificados en el requerimiento pueden formar parte de las precondiciones, y el verbo principal puede vincularse tanto al título como al resultado esperado de un caso de prueba. Por estos motivos, los símbolos del LEL seleccionados en subpasos anteriores se clasifican en categorías de precondiciones, pasos y resultados esperados, considerando escenarios tanto positivos como negativos.

Creación de casos de prueba en formato Gherkin: los casos de prueba se redactan en formato Gherkin utilizando las palabras clave Given(Dado), When(Cuando) y Then(Entonces) para definir de manera clara las precondiciones, los pasos de ejecución y el resultado esperado de los escenarios positivos y negativos.

Para ello, se utilizan los símbolos del LEL clasificados previamente y el requerimiento seleccionado.

3.4 Aplicación de sugerencias para mejorar los casos de pruebas

Como último paso, se recomienda un refinamiento de los casos de prueba generados en la etapa anterior, con el objetivo de que estos reflejen los escenarios principales o críticos definidos por el cliente con el mayor detalle. En este paso, participan tanto el equipo de testers como el cliente. Los testers, debido a su experiencia técnica en el área de pruebas y el cliente para aportar el conocimiento sobre los escenarios críticos y prioritarios que requieren validación. Estos casos de prueba refinados se redactarán en formato Gherkin para una estructura clara y accesible. Este paso sigue un enfoque iterativo, permitiendo al equipo de testers y al cliente realizar tantos refinamientos como consideren necesarios para garantizar la exhaustividad de los casos de prueba. Para guiar este proceso de mejora, se plantean las siguientes preguntas:

- ¿Es una funcionalidad nueva o un cambio solicitado sobre una funcionalidad existente?
- En caso de ser un cambio solicitado sobre una funcionalidad existente, ¿implica un cambio a nivel funcional o técnico?
- En caso de ser una funcionalidad nueva, ¿Cómo impacta esta nueva funcionalidad al sistema actual?
- ¿Cuál es el alcance y las limitaciones de la funcionalidad que se quiere probar?
- ¿Existen flujos de usuario principales o críticos que deban ser considerados?
- ¿Existen flujos de usuario secundarios que deban ser considerados?
- ¿Cuál es la entrada de datos? ¿Tiene algún archivo o datos específicos?
- ¿Cuál es la salida esperada? ¿Genera alguna salida en base de datos, archivos o un log?
- ¿Qué servicios web y APIs consume la funcionalidad?
- ¿Qué base de datos y tablas consulta?
- ¿Se relaciona con otros productos o sistemas de la organización? A nivel de dependencias o integraciones.

Es importante señalar que, si bien estas preguntas no especifican de forma directa qué cambios deben realizarse en la estructura de los casos de prueba, su propósito es actuar como disparadores analíticos para detectar qué ajustes podrían ser necesarios. Por ejemplo, una nueva funcionalidad podría requerir la incorporación de escenarios adicionales, mientras que una modificación sobre una funcionalidad existente podría implicar cambios en las precondiciones, los pasos de ejecución o los resultados esperados de los casos de prueba generados.

4 Evaluación

El enfoque propuesto fue evaluado a través de un caso de aplicación. Específicamente, se evaluó el segundo paso del enfoque, que implica el diseño de los casos de prueba en formato Gherkin usando como entrada el Léxico Extendido del

Lenguaje mejorado en el primer paso del método y la Especificación de Requerimientos de Software.

Los participantes en la evaluación fueron 10 miembros de un proyecto de desarrollo de software del sector bancario en Argentina. Dicho proyecto se enfocó en el desarrollo y mantenimiento de un sistema para la gestión de transacciones financieras, administración de cuentas de clientes y operación de productos, garantizando el cumplimiento de normativas regulatorias y altos estándares de seguridad. Todos los participantes tenían experiencia en el área del desarrollo de software y desempeñaron el rol de analista de aseguramiento de calidad por más de 3 años. Es importante mencionar que los participantes no tenían experiencia previa en el uso del Léxico Extendido del Lenguaje. Por lo tanto, antes del experimento recibieron capacitación sobre el enfoque propuesto. Todos los participantes colaboraron en el diseño de casos de prueba. Su tarea consistió en seleccionar un requerimiento de la Especificación de Requerimientos de Software. A partir de dicho requerimiento, se identificaron los símbolos del Léxico Extendido del Lenguaje que presentaban correspondencia tanto sintáctica como semántica con el mismo. De los símbolos seleccionados, se extrajeron las columnas "Noción" e "Impacto". Posteriormente, los símbolos del LEL fueron clasificados en las categorías de precondiciones, pasos y resultados esperados, considerando solo el escenario positivo. Finalmente, los casos de prueba fueron redactados en formato Gherkin, utilizando las palabras clave Given (Dado), When (Cuando) y Then (Entonces).

Se utilizó la Escala de Usabilidad del Sistema (SUS, System Usability Scale) [20] [21] para evaluar la usabilidad y aplicabilidad del enfoque propuesto. Aunque la SUS se emplea principalmente para evaluar la usabilidad de sistemas de software, ha demostrado ser efectiva en la evaluación de procesos y productos [22]. Consiste en un cuestionario de 10 ítems, donde cada pregunta debe responderse en una escala de cinco puntos, que va desde "1" ("Totalmente en desacuerdo") hasta "5" ("Totalmente de acuerdo"). A pesar de que el cuestionario tiene 10 preguntas, estas están emparejadas, formulando la misma pregunta desde una perspectiva complementaria para obtener un resultado de alta confianza. El puntaje SUS se calcula de la siguiente manera: primero, los ítems 1, 3, 5, 7 y 9 se puntúan considerando el valor asignado menos 1. Luego, los ítems 2, 4, 6, 8 y 10 se puntúan considerando 5 menos el valor asignado. Posteriormente, se suman los puntajes de cada participante y se multiplican por 2.5, obteniendo un valor que varía entre 0 y 100. Finalmente, se calcula el promedio. El enfoque puede clasificarse en una de las siguientes categorías: "No aceptable" (0-64), "Aceptable" (65-84) y "Excelente" (85-100) [23].

El puntaje obtenido fue de 70.01, lo que posiciona al enfoque dentro de la categoría "aceptable". Este resultado evidencia que el método propuesto puede ser comprendido y aplicado eficazmente por profesionales en contextos reales, permitiendo que diseñen casos de prueba incluso sin experiencia previa con el Léxico Extendido del Lenguaje.

5 Trabajos relacionados

A lo largo de los años, se han publicado numerosos artículos científicos que definen un proceso para la generación de casos de prueba a partir de los

requerimientos. Sin embargo, los autores no han identificado enfoques que contemplen el uso del Léxico Extendido del Lenguaje y la Especificación de Requerimientos como entrada. En este contexto, se llevó a cabo un análisis sistemático de la literatura que aborda métodos basados en el análisis semántico y sintáctico de los requerimientos, con el objetivo de contextualizar y comparar los enfoques existentes con el método propuesto, especialmente en lo relativo al paso de "Generación de casos de prueba".

Gutiérrez Rodríguez et al [24] proponen un proceso que convierte la sintaxis concreta de un requerimiento funcional a una sintaxis abstracta mediante el desarrollo de metamodelos y la definición de transformaciones QVT (Query-View Transformation), con el fin de obtener escenarios de prueba y valores de prueba que luego permitan generar casos de prueba funcionales. Por su parte, Dwarakanath y Sengupta [25] presentan una herramienta que analiza las oraciones de un documento de requerimientos funcionales mediante un analizador sintáctico para determinar la testeabilidad. Si la oración es compleja, se divide en oraciones simples, de las cuales se generan intenciones de prueba que representan los aspectos a evaluar. Estas intenciones se agrupan y secuencian para generar uno o más casos de prueba positivos y negativos. En una línea similar, Alcaraz [26] desarrolla una gramática libre de contexto que permite realizar un análisis sintáctico y semántico de los requerimientos, a partir del cual se generan automáticamente casos de prueba. Los autores señalan que, para aplicar este enfoque a otros tipos de requerimientos, es necesario ampliar la gramática, ya que fue diseñada específicamente para el análisis de requerimientos funcionales de hardware.

Sneed [27], en cambio, presenta un enfoque para la generación de casos de prueba a partir de requerimientos escritos en lenguaje natural. Se utiliza un analizador de texto que examina tanto requerimientos funcionales como no funcionales, identificando objetos, estados y acciones relevantes mediante el análisis de sustantivos y su contexto, con el fin de generar casos de prueba a partir de los eventos especificados. Análogamente, Verma y Beg [28] proponen un enfoque para generar casos de prueba a partir de requerimientos expresados en lenguaje natural utilizando técnicas de procesamiento de lenguaje natural. Para ello, se realiza un preprocesamiento para normalizar el texto. Luego, se realiza un etiquetado de partes del discurso donde se asigna una categoría gramatical a cada palabra en una oración. A continuación, se utiliza un analizador sintáctico que genera árboles por cada requerimiento y se utilizan grafos para representar el conocimiento de los requerimientos. Por último, se recorre el grafo utilizando la técnica de valores límites para generar los casos de prueba. En esta misma línea, Ansari et al. [29] proponen un método que genera casos de prueba automáticamente a partir de requerimientos funcionales expresados en lenguaje natural, utilizando técnicas de procesamiento de lenguaje natural. El sistema identifica estructuras condicionales del tipo "if...then" dentro de los requerimientos, y extrae de ellas los pasos de prueba, los datos de entrada y los resultados esperados, organizándolos en una tabla estructurada.

Por otro lado, existen enfoques que utilizan diferentes artefactos que por lo general suelen ser creados en la etapa de análisis como entrada al método para la generación de casos de prueba. Ibrahim et al [30] desarrollan una herramienta que genera diagramas de casos de uso, flujos de eventos y diagramas de secuencia a partir de los

requerimientos funcionales, y emplea estos elementos tanto para la creación como para la verificación y validez de los casos de prueba generados. De manera similar, Freudenstein et al [31] también utilizan una herramienta para modelar requerimientos funcionales utilizando gráficos de causa-efecto. A partir de estos gráficos, se diseñan de manera automática los casos de prueba. Lafi et al [32], por el contrario, propone un método para generar casos de prueba a partir de descripciones de casos de uso. Para ello, se utiliza la descripción textual del caso de uso y su diagrama UML para extraer la información necesaria. Luego, se crea un grafo de flujo de control y una tabla de procesamiento de lenguaje natural a partir de la información extraída. Con estos elementos, se generan rutas de prueba que se utilizan para crear los casos de prueba. Al mismo tiempo, Chatterjee y Johari [33] desarrollan la herramienta STATEST 1.0.0 que formaliza los requerimientos transformando los casos de uso y sus escenarios en diagramas de transición de estado (STD), los cuales se utilizan para generar los casos de prueba.

A pesar de los avances en estos enfoques, ninguno de ellos aborda de manera efectiva los problemas relacionados con la ambigüedad, incompletitud e inconsistencia de los requerimientos. En este contexto, nuestra propuesta introduce el uso del Léxico Extendido del Lenguaje junto con la Especificación de Requerimientos de Software como entrada del método para la generación de casos de prueba como posible solución. El objetivo es mejorar la completitud y precisión de los casos de prueba generados, logrando una mayor cobertura y reduciendo las ambigüedades inherentes a los requerimientos expresados en lenguaje natural que puedan comprometer la calidad del software resultante.

6 Conclusiones y trabajos futuros

El artículo propone un método secuencial que permite la generación de casos de pruebas en formato Gherkin, utilizando como entrada la Especificación de Requerimientos de Software y el Léxico Extendido del Lenguaje. Si bien en el presente artículo el método se presenta usando una ERS como entrada, también es posible utilizar historias de usuario o casos de uso para la generación de casos de prueba. El principal desafío del método radica en que la efectividad de los casos de prueba depende de la claridad y precisión de los requerimientos especificados en la ERS, así como de la correcta definición de los símbolos en el LEL. Requerimientos ambiguos o inconsistentes, o un LEL mal estructurado pueden resultar en la generación de casos de prueba poco adecuados o imprecisos, comprometiendo la efectividad del proceso de prueba.

Como línea de investigación futura, se propone que el paso “Generación de casos de prueba” adopte un enfoque iterativo, permitiendo ajustes progresivos en los casos de pruebas, en respuesta al crecimiento en tamaño y complejidad del software en desarrollo. Asimismo, se plantea la posibilidad de que el método sea iterativo en su totalidad, de manera que el LEL y los casos de prueba refinados en formato Gherkin obtenidos en iteraciones previas, puedan ser reutilizados como entrada en ciclos sucesivos. Una vía adicional consiste en incorporar inteligencia artificial en el paso “Generación de casos de prueba”, aplicando técnicas de procesamiento de lenguaje

natural para realizar un análisis sintáctico y semántico de los requerimientos y los símbolos del LEL, con el objetivo de generar escenarios más precisos y representativos. Finalmente, se plantea una evaluación más profunda en donde profesionales con diferentes roles y años de experiencia, generen escenarios positivos y negativos con el fin de evaluar la robustez y factibilidad del método.

Referencias

1. Pressman, R.S.: Ingeniería del software un enfoque práctico. McGraw-Hill (2010).
2. Toledo, F.: Introducción a las pruebas de sistemas de información. Abstracta (2020).
3. Real Academia Española: Calidad, en Diccionario de la lengua española, 23.ª ed., https://dle.rae.es/calidad (s.f.).
4. ISO: Calidad, en ISO 9000:2015 - Sistemas de gestión de la calidad - Fundamentos y vocabulario, Organización Internacional de Normalización, https://www.iso.org/obp/ui/es/#iso:std:iso:9000:ed-4:v1:es (2015).
5. IEEE: IEEE Standard Glossary of Software Engineering Terminology, in IEEE Std 610.12-1990, pp.1-84. doi: 10.1109/IEEESTD.1990.101064 (1990).
6. Sommerville, I.: Ingeniería del software. Addison-Wesley (2011).
7. Deming, W.E.: Calidad, productividad y competitividad: La salida de la crisis. Díaz de Santos (1989).
8. Denney, E., Pai, G.: Tool support for assurance case development. Automated Software Engineering, vol. 25, no. 3, pp. 435–499 (2018).
9. Carvalho, G., Falcao, D., Barros, F., Sampaio, A., Mota, A., Motta, L., Blackburn, M.R.: NAT2TEST(SCR): Test case generation from natural language requirements based on SCR specifications, in Proceedings of the 28th Annual ACM Symposium on Applied Computing, doi: 10.1145/2480362.2480591 (2013).
10. Medeiros, J., Vasconcelos, A., Silva, C., Goulão, M.: Quality of software requirements specification in agile projects: a cross-case analysis of six companies. Journal of Systems and Software, vol.142, pp 171-194, doi: 10.1016/j.jss.2018.04.064 (2018).
11. Farooq, M.S., Tahreem, T.: Requirement-Based Automated Test Case Generation: Systematic Literature Review. VFAST Transactions on Software Engineering, doi:10.21015/vtse.v10i2.940 (2022).
12. Serna, M.E., Arango, F.: Software testing: More than a stage in the life cycle. Revista de Ingeniería, Universidad de los Andes, Bogotá D.C., Colombia, Vol. 35, pp. 34-40, ISSN 0121-4993 (2011).
13. Laplante, P.A.: Requirements Engineering for Software and Systems. Taylor & Francis Group (2018).
14. Leite, J.C.S.P., Franco, A.P.M.: O uso de Hipertexto na Elicitaçao de Linguagens da Aplicaçao", in Brazilian Symposium on Software Engineering, doi:10.5753/sbes.1990.24172 (1990).
15. Cauchi, A., Colombo, C., Francalanza, A., Micallef, M., Pace, G.: Using Gherkin to extract tests and monitors for safer medical device interaction design, in EICS '16: Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 275-280 (2016).
16. Hadad, G., Kaplan, G., Oliveros, A., Leite, J.C.S.P.: Integración de Escenarios con el Léxico Extendido del Lenguaje en la elicitación de requerimientos: Aplicación a un caso real. Departamento de Investigación, Universidad de Belgrano y Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (1996).

17. Antonelli, L., Rossi, G., Leite, J.C.S.P., Oliveros, A.: Language extended lexicon points: Estimating the size of an application using its language, in Proceedings IEEE 22nd International Requirements Engineering Conference (RE), pp. 263-272, doi: 10.1109/RE.2014.6912268 (2014).
18. IEEE: IEEE Recommended Practice for Software Requirements Specifications, in IEEE Std 830-1998, Revision of IEEE Std 830-1993 (1998).
19. Antonelli, L., Rossi, G., Leite, J.C.S.P., Oliveros, A.: Deriving requirements specifications from the application domain language captured by Language Extended Lexicon, Workshop in Requirements Engineering (WER), Buenos Aires, Argentina, April 24 – 27 (2012).
20. Brooke, J.: "SUS-A quick and dirty usability scale." Usability evaluation in industry. CRC Press (June 1996), https://www.crcpress.com/product/isbn/9780748404605, iSBN: 9780748404605.
21. Brooke, J.: SUS: a retrospective. Journal of usability studies 8(2), 29–40 (2013).
22. Bangor, A., Kortum, P.T., Miller, J.T.: An Empirical Evaluation of the System Usability Scale. Intl. Journal of Human-Computer Interaction 24(6), 1–44, https://doi.org/10.1080/10447310802205776 (2008).
23. McLellan, S., Muddimer, A., Peres, S. C.:The effect of experience on System Usability Scale ratings. Journal of usability studies 7.2, pp. 56-67 (2012).
24. Gutiérrez Rodríguez, J.J., Escalona Cuaresma, M.J., Mejías Risoto, M.: Generación de pruebas del sistema a partir de la especificación funcional, Tesis doctoral, Universidad de Sevilla, España (2011).
25. Dwarakanath, A., Sengupta, S.: Litmus: Generation of Test Cases from Functional Requirements in Natural Language. Lecture Notes in Computer Science, pp 58–69, doi:10.1007/978-3-642-31178-9_6 (2012).
26. Alcaraz, A.: Generación automática de casos de prueba a partir de requerimientos funcionales utilizando gramáticas libres de contexto. Dirección de Posgrado, CIATEQ A. C. Centro de Tecnología Avanzada (2021).
27. Sneed, H.M.: Testing against Natural Language Requirements, in Seventh International Conference on Quality Software (QSIC 2007), Portland, OR, USA, 2007, pp. 380-387, doi: 10.1109/QSIC.2007.4385524 (2007).
28. Verma, R.P., Beg, M.R.: Generation of Test Cases from Software Requirements Using Natural Language Processing, in 6th International Conference on Emerging Trends in Engineering and Technology, doi:10.1109/icetet.2013.45 (2013).
29. Ansari, A., Baig, M.S., Fatima, A.S., Shaikh, T.: Constructing Test Cases Using Natural Language Processing, 3rd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB17), IEEE, India (2017).
30. Ibrahim, R., Saringat, M. Z., Ibrahim, N., Ismail, N.: An Automatic Tool for Generating Test Cases from the System's Requirements, in 7th IEEE International Conference on Computer and Information Technology (CIT 2007), doi:10.1109/cit.2007.116 (2007).
31. Freudenstein, D., Radduenz, J., Junker, M., Eder, S., Hauptmann, B.: Automated test-design from requirements: the Specmate tool, in proceedings of the 5th International Workshop on Requirements Engineering and Testing (RET '18), https://doi.org/10.1145/3195538.3195543 (2018).
32. Lafi, M., Alrawashed T., Hammad, A.M.: Automated Test Cases Generation From Requirements Specification, in 2021 International Conference on Information Technology (ICIT), doi: 10.1109/ICIT52682.2021.9491761 (2021).
33. Chatterjee, R., Johari, K.: A prolific approach for automated generation of test cases from informal requirements. SIGSOFT Software Engineering Notes, vol. 35, pp. 1-11, doi:10.1145/1838687.1838702 (2010).