Una Herramienta Para Componer Escenarios A Través De Procesamiento Del Lenguaje Natural Y Derivar Casos De Prueba

Gregorio Maclen[1], Diego Torres[1], y Leandro Antonelli[1,2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata (UNLP), Buenos Aires, Argentina
[2] CAETI - Facultad de Tecnología Informática - Universidad Abierta Interamericana
{gmaclen, dtorres, lanto}@lifia.info.unlp.edu.ar

Resumen. El diseño de casos de prueba es una de las actividades más desafiantes en el contexto de la ingeniería de requerimientos, dado que implica la colaboración de diferentes individuos con variados conocimientos en el dominio y perspectivas, con el objetivo de desarrollar un producto que satisfaga las expectativas y necesidades de los clientes. El modelo de desarrollo en V propone abordar el diseño de casos de prueba a partir de los requerimientos. Aunque constituye un punto de partida valioso, no siempre resulta sencillo obtener una especificación de requerimientos ordenada y consistente, que describa de manera integral toda la funcionalidad del sistema de software. Los escenarios se presentan como un artefacto efectivo para la especificación de requerimientos. Consisten en descripciones en lenguaje natural que delinean una secuencia de pasos desde un contexto o punto de partida específico hasta un objetivo o meta determinados. Los escenarios son atómicos, lo que implica que diferentes expertos pueden describir distintos escenarios en función de su conocimiento y perspectiva. Sin embargo, para el diseño efectivo de los casos de prueba, resulta crucial organizar los escenarios en una estructura jerárquica que facilite la identificación sistemática de todos los casos que necesitan ser evaluados. Este artículo propone una herramienta que permite la edición de escenarios y utiliza diversas técnicas de procesamiento de lenguaje natural para organizarlos en un árbol. Este enfoque se fundamenta en la relación "un escenario se describe con otro escenario", lo que posibilita la generación de un árbol exhaustivo que abarque toda la funcionalidad del sistema. Posteriormente, la herramienta facilita la poda de ramas no deseadas para limitar el árbol a la funcionalidad que sea de interés evaluar. Finalmente, a partir del árbol resultante, se obtienen los casos de prueba. Es importante señalar que las bases teóricas y metodológicas para la generación de casos de prueba han sido presentadas en publicaciones previas; sin embargo, este trabajo se centra en el desarrollo y aplicación de una herramienta que automatiza dicho proceso, contribuyendo así a la eficiencia y precisión de la fase de pruebas en el ciclo de desarrollo de software.

Palabras clave: Escenarios · Casos de prueba · Procesamiento del lenguaje natural

1 Introducción

En el contexto de la ingeniería de requerimientos, el diseño de casos de prueba es una tarea desafiante que requiere la colaboración de diversos individuos con conocimientos variados en el dominio. Su objetivo es la satisfacción de las expectativas y necesidades de los clientes a través del desarrollo de productos de software funcionales y eficientes. En este sentido, el modelo de desarrollo en V [6] propone abordar el diseño de casos de prueba a partir de los requerimientos, reconociendo su valor como punto de partida fundamental. Sin embargo, obtener una especificación de requerimientos ordenada y consistente que describa de manera integral la funcionalidad del sistema de software no siempre resulta una tarea sencilla. Es en este contexto que los escenarios [4] emergen como un artefacto efectivo para la especificación de requerimientos, ya que proporcionan descripciones en lenguaje natural que delinean una secuencia de pasos desde un contexto específico hasta un objetivo determinado. Los escenarios, al ser atómicos, permiten que diferentes expertos los describan según su conocimiento y perspectiva, lo que enriquece el proceso de diseño de casos de prueba.

La organización jerárquica de los escenarios resulta un aspecto crucial para el diseño efectivo de casos de prueba, ya que facilita la identificación sistemática de todos los casos que requieren evaluación. La presente investigación propone una herramienta innovadora que permite implementar esta organización jerárquica, generando así un árbol exhaustivo que abarca toda la funcionalidad del sistema. Esta aproximación, basada en la relación "un escenario se describe con otro escenario", posibilita la generación de casos de prueba de manera eficiente y precisa [1].

En este trabajo, nos enfocaremos en el desarrollo y aplicación de esta herramienta que automatiza el proceso de generación de casos de prueba, contribuyendo así a la eficiencia y precisión de la fase de pruebas en el ciclo de desarrollo de software. A través de la integración de escenarios, procesamiento del lenguaje natural y derivación de casos de prueba, buscamos mejorar la calidad y confiabilidad de los productos de software resultantes.

2 Contexto

2.1 Escenarios

Los escenarios [4][5] son un tipo de artefacto utilizado en la ingeniería de requerimientos para describir, de forma organizada y concreta, la secuencia de actividades que realizan las personas (utilizando recursos) para alcanzar un objetivo. Estas actividades se denominan episodios, y un episodio puede ser, a su vez, otro escenario. Es decir, un escenario puede estar compuesto por otro/s escenario/s. La estructura completa de un escenario es la siguiente:

Tabla 1. Estructura de un escenario.

Campo	**Descripción**
Título	Acción a describir en infinitivo
Objetivo	Condiciones que se desean alcanzar luego de la ejecución del escenario
Contexto	Condiciones que deben estar cumplidas antes de la ejecución del escenario
Actores	Participantes que ejecutan acciones para alcanzar el objetivo
Recursos	Elementos que son utilizados por los actores para realizar acciones
Episodios	Listado de acciones que los actores llevan a cabo para alcanzar el objetivo

A lo largo de este artículo se utilizarán ejemplos dentro del dominio de la agricultura, similares a los descritos por Antonelli et al. (2023) [1]. Se podría describir entonces de la siguiente manera a la actividad de cultivar las plantas:

Tabla 2. Escenario "Cultivar las plantas".

Campo	**Descripción**
Título	Cultivar las plantas
Objetivo	Realizar las actividades necesarias para favorecer el crecimiento de la planta
Contexto	La planta ha llegado a su fase vegetativa
Actores	Agricultor
Recursos	Agua, fertilizante, tijeras de podar
Episodios	– El agricultor riega las plantas. – El agricultor realiza control de malezas. – El agricultor fertiliza utilizando una tubería de riego.

Además, se definirá otro escenario para describir a la actividad de fertilizar utilizando una tubería de riego:

Tabla 3. Escenario "Fertilizar utilizando una tubería de riego".

Campo	**Descripción**
Título	Fertilizar utilizando una tubería de riego
Objetivo	Agregar nutrientes a la planta
Contexto	La cisterna tiene suficiente agua para activar la tubería de riego
Actores	Agricultor
Recursos	Cisterna con agua, tubería de riego, minerales, tabla para calcular la cantidad de minerales
Episodios	– El agricultor calcula la cantidad de minerales. – El agricultor diluye los minerales en el agua. – El agricultor vierte la mezcla en la tubería de riego. – El agricultor activa la tubería de riego. – El agricultor vierte agua dulce en la tubería de riego.

Podrá notar el lector que uno de los episodios del escenario de la Tabla 3 es “El agricultor fertiliza utilizando una tubería de riego.”, y el escenario de la Tabla 4 se llama “Fertilizar utilizando una tubería de riego”. Se dice entonces que el primer escenario se compone o se describe con el segundo, ya que uno de sus episodios es a su vez un escenario.

2.2 Modelo Task/Method

En nuestro enfoque, se utiliza un modelo conceptual [3] para representar escenarios. Un modelo conceptual consta de dos partes: un modelo de dominio y un modelo de razonamiento. Los objetos del mundo (o más precisamente del dominio de aplicación) están representados en el modelo de dominio. El modelo de razonamiento describe cómo se puede realizar una acción (tarea). Por lo tanto, todos los objetos y relaciones relevantes en el mundo manejados por el modelo de razonamiento deben describirse en el modelo de dominio.

En este modelo, una tarea es una transición entre dos familias de estados del mundo (una acción), y un método describe una forma (en un único nivel de abstracción) de realizar una tarea. La forma en que se realiza el razonamiento se modela como una descomposición de tareas y subtareas. El modelo de razonamiento describe varias descomposiciones del desempeño de tareas utilizando métodos. En este sentido, el modelo de razonamiento es jerárquico, y las descomposiciones se detienen en las tareas terminales cuya ejecución no está descrita.

Los escenarios se pueden representar en forma de modelo de Task/Method [2] para generar casos de prueba. Por ejemplo, para el escenario “Cultivar las plantas”, la tarea es “Cultivar las plantas” y el método que la realiza es el conjunto de episodios del escenario. A su vez, la subtarea de ese método, “El agricultor fertiliza utilizando una tubería de riego.”, es realizada por otro método compuesto por los episodios del escenario “Fertilizar utilizando una tubería de riego”, mientras que las demás subtareas son terminales ya que no hay un método que las realice.

2.3 Derivación de casos de prueba

El enfoque propuesto obtiene todas las combinaciones de situaciones descritas en episodios de los escenarios. Además, se asumirá que la ejecución de una tarea sólo puede tener éxito o fracasar. Las fallas en las tareas están relacionadas con errores en la realización de la tarea (manualmente) o en el módulo que la implementa. Cada episodio está representado por una tarea o subtarea y se debe analizar el resultado del episodio [2]. Si el resultado es exitoso, la ejecución del escenario continúa, pero en el otro caso, si el resultado falla, el escenario finaliza. En los escenarios, los casos de falla de logro del escenario se describen en los casos de prueba y en el modelo Task/Method corresponden a una falla en la ejecución de subtareas.

3 Herramienta

3.1 Arquitectura y diseño

La herramienta descrita en este artículo es en realidad una funcionalidad agregada a una aplicación web preexistente. La función de esta aplicación es gestionar proyectos de requerimientos, y provee una vasta cantidad de funcionalidades para gestionar artefactos. Está implementada a través del entorno de desarrollo Django, que funciona sobre el lenguaje de programación Python. Para identificar la equivalencia semántica entre oraciones se utiliza spaCy [7], una potente librería de código abierto que provee funcionalidades para procesar texto en lenguaje natural.

Cuando un usuario selecciona un escenario y selecciona la opción de ver el árbol a partir de ese escenario, la petición se envía al servidor, el cual se encarga de procesar los datos y enviar una respuesta al cliente, es decir, el navegador web. Cuando el cliente recibe la respuesta, el gráfico se crea utilizando el lenguaje de programación JavaScript junto a la librería D3.js [8]. Esta librería, cuya versión inicial fue lanzada en 2011, es gratuita y de código abierto. Permite la creación de múltiples tipos de gráficos, como árboles, gráficos de torta, de barras, de burbujas, etcétera, mediante la posibilidad de seleccionar determinados elementos del DOM (modelo de objeto del documento) y aplicarles distintas transformaciones como estilos, atributos, o transiciones.

3.2 Funcionamiento

El punto de partida se dará en un contexto con múltiples escenarios que pertenecen a un mismo problema o dominio. Las tareas de esta herramienta serán, a partir de un escenario que elija el usuario, identificar, para cada uno de sus episodios, si se trata a su vez de un escenario y repetir el proceso recursivamente para armar un árbol y poder graficarlo visualmente.

Con respecto al ejemplo descrito en la sección 2.1, si un usuario selecciona la opción de ver el árbol de escenarios partiendo del escenario “Cultivar las plantas”, el desafío de la herramienta será reconocer que, a pesar de que estas dos oraciones estén escritas en distinta forma, ambas se refieren a lo mismo, relacionar los escenarios para poder crear un árbol, y, por último, graficarlo.

Procesar los datos Es en esta etapa que sucede toda la lógica para construir una estructura de datos adecuada a partir de los escenarios almacenados en la base de datos. El principal desafío es identificar la equivalencia entre dos oraciones escritas en distintas formas, pero que significan lo mismo, como en el caso del ejemplo provisto.

Mediante el uso de spaCy, se debe determinar la equivalencia semántica entre cada par de oraciones (es decir, nombres de escenarios y nombres de episodios). Para ello, dado un par de oraciones, la herramienta elimina el sujeto y los caracteres de puntuación de cada una de ellas, luego las convierte a su forma en infinitivo y, por último, establece el grado de similaridad entre una y otra.

Si el grado de similaridad es muy alto, significa que las oraciones iniciales son semánticamente equivalentes. Este proceso se realiza para cada par de oraciones y, en base a los resultados de equivalencia, se agregan los nodos del árbol.

Graficar el árbol Una vez construida la estructura de datos, es necesario crear el gráfico para que el usuario pueda verlo en su pantalla. El primer paso es definir, según las opciones que brinda D3.js, qué tipo de gráfico se desea dibujar. En este caso, será un gráfico de tipo árbol. Luego, se inicializan algunos datos como las dimensiones del gráfico, se le da al gráfico la estructura de datos de entrada, y se definen los atributos para los nodos y los enlaces del árbol. Se define que se dibujará un nodo por cada episodio que esté incluido en el árbol, y se dibujarán enlaces desde cada nodo a sus respectivos episodios hijos. Si se toma el ejemplo definido previamente, se formará el siguiente gráfico:

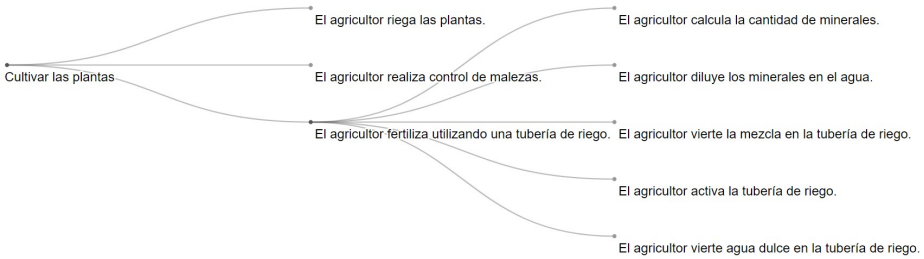

Figura 1. Árbol inicial de episodios.

Se observa que el árbol parte del escenario "Cultivar las plantas". Sus nodos hijos son sus episodios y, como su episodio "El agricultor fertiliza utilizando una tubería de riego." es otro escenario, se agregan los episodios de ese escenario como nodos hijos. Como resultado se observa que el gráfico se compone de todas las acciones que involucra el escenario inicial.

Podar el árbol Una funcionalidad importante que debe tener esta herramienta es la de podar el árbol. Esta actividad no es automática, sino que debe ser realizada manualmente por una persona que tenga los conocimientos necesarios y un profundo entendimiento del dominio del negocio para poder decidir qué ramas del árbol no son relevantes para el mismo.

Para implementar esta funcionalidad, se agrega con D3.js a cada nodo intermedio (es decir, que no es la raíz ni es una hoja) un botón para podar esa rama (Figura 2). De esa forma, el usuario tiene la posibilidad de podar ramas irrelevantes y visualizar el árbol con las ramas que sí quiera ver. En el desarrollo de la herramienta se decidió que, cuando una rama es podada, la misma no desaparece del gráfico sino que su color cambia a rojo (Figura 3) para que el usuario note la diferencia, aunque también existe la posibilidad de modificar la configuración de la librería para que las ramas desaparezcan al ser podadas.

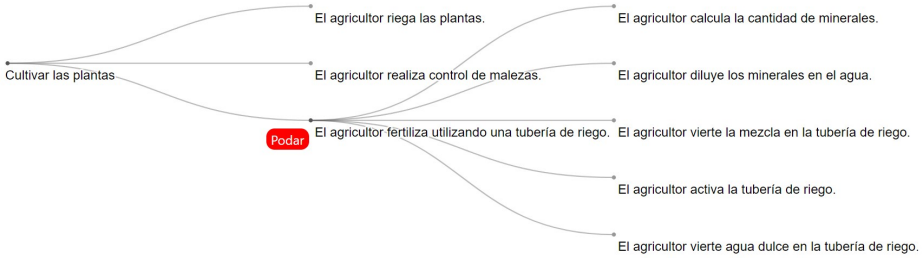

Figura 2. Árbol sin ninguna rama podada.

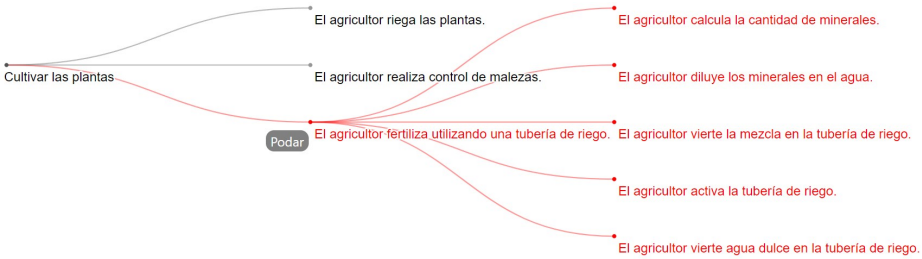

Figura 3. Árbol con la rama de su único nodo intermedio podada.

Task/Method y derivar casos de prueba Una vez que fueron podadas las ramas irrelevantes del árbol, el proceso [2] puede continuar su flujo normal. El siguiente paso para la herramienta es convertir el árbol resultante al formato de Task/Method, es decir, traducir los nodos y ramas del árbol de escenarios en tareas y métodos que representen de manera clara y concisa las acciones y secuencias necesarias para alcanzar los objetivos establecidos, siguiendo lo descrito en la sección 2.2. Por último, se procede a la generación de los casos de prueba correspondientes, donde cada método del modelo Task/Method se convierte en un caso de prueba individual. Esto permite identificar posibles fallos en la implementación del software. Para el ejemplo de la Figura 3, en el que sólo hay dos métodos ("El agricultor riega las plantas." y "El agricultor realiza control de malezas."), se generan dos casos de prueba que corresponden al caso de fallo de cada método. Los casos de prueba generados son "El agricultor falla al regar las plantas." y "El agricultor falla al realizar control de malezas.".

4 Conclusiones

La herramienta desarrollada representa un avance significativo en el campo de la ingeniería de requerimientos y el diseño de software, y esperamos que facilite la aplicación del método de diseño de casos de prueba basado en escenarios, permitiendo a los equipos de desarrollo generar casos de prueba de manera más eficiente y precisa. Se espera que la implementación de esta herramienta en los

procesos de desarrollo de software conduzca a una mejora sustancial en la calidad y confiabilidad del software resultante, al garantizar una cobertura exhaustiva de los escenarios y una mayor precisión en la definición de casos de prueba.

Como siguiente paso, planeamos realizar pruebas de usabilidad de la herramienta para evaluar su facilidad de uso, eficacia y eficiencia en la composición de escenarios y la derivación de casos de prueba. Estas pruebas nos permitirán identificar posibles áreas de mejora y ajustes necesarios para optimizar la experiencia del usuario y maximizar la utilidad de la herramienta en entornos de desarrollo de software. Además, tenemos previsto realizar una extensión de la herramienta para incorporar funcionalidades adicionales que puedan enriquecer su capacidad de generación de casos de prueba y su integración con otros procesos de desarrollo de software.

Para concluir, confiamos en que esta herramienta no solo facilitará la generación de casos de prueba, sino que también contribuirá a la mejora general de la calidad del software a través de un proceso de pruebas más riguroso y efectivo.

Referencias

1. Antonelli, L., Torres D., Camilleri G., Zarate P.: User acceptance test for software development in the agricultural domain using natural language processing (2023)
2. Antonelli, L., Torres D., Camilleri G., Zarate P.: AGUTER a platform for automated generation of user acceptance tests from requirements specifications (2021)
3. Trichet, F., Tchounikine, P.: Dstm: a framework to operationalise and refine a problem solving method modeled in terms of tasks and methods (1999)
4. Carrol, J. M.: Five reasons for scenario-based design, in Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences (1999)
5. Ridao, M., Doorn, J. H., Sampaio do Prado Leite, J. C.: Uso de Patrones en la Construcción de Escenarios. WER 2000: 140-157
6. Forsberg, K., Mooz, H.: The relationship of system engineering to the project cycle (1991)
7. Vasiliev, Y.: Natural Language Processing with Python and SpaCy: A Practical Introduction (2020)
8. Página de D3.js, https://d3js.org/. Accedido por última vez el 3 de marzo de 2024