UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Testing Basado en la Búsqueda en TESTAR

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Francisco Almenar Pedrós

Tutor: Tanja E. J. Vos

2015-2016

Agradecimientos

Quisiera dedicar unas palabras de agradecimiento a varias personas por la ayuda y el apoyo que me han prestado en la realización de este trabajo. Entre ellas, y en primer lugar, a mi tutora Tanja E.J. Vos por su ayuda y su guía durante los años que he colaborado con ella.

También quiero agradecer a Urko Rueda por su ayuda y por facilitar el desarrollo de mis capacidades. Como gran maestro de TESTAR, ha sido un gran guía y un buen compañero, gracias por todas esas dudas que has resuelto.

Me gustaría hacer una mención especial a Anna I. Esparcia por su comprensión y su ayuda a lo largo de todo el proyecto y más aún de cara al final del mismo. Echaré de menos esas críticas constantes que me han ayudado a crecer y que casi han acabado con mi paciencia.

También quiero agradecer de todo corazón a Mirella Oreto Martínez, sin su apoyo incondicional y su ayuda me hubiera resultado muy difícil acabar este trabajo. Has sido para mí todo un modelo de constancia y trabajo duro, lo que me ha inspirado para seguir hacia delante. Gracias por ser la persona que más alegra mi vida y por estar ahí cuando te necesito.

Me faltarían días en la vida para mostrarle todo mi agradecimiento a mi familia, gracias a su esfuerzo y su dedicación he podido dedicarme a mi pasión, la informática. Han sido un apoyo constante, y me lo han dado todo sin esperar nada a cambio.

Por último agradecer a Jose y a Luis por hacerme desconectar del trabajo pese a lo difícil que se lo he puesto.

Gracias a todos vosotros por colaborar en la realización de esta Trabajo Final de Máster. Unos, directamente conmigo, y otros, transformando los días en momentos inolvidables.

Resumen

Las interfaces gráficas de usuario (IGU) constituyen un punto vital para testear una aplicación. Para ello se han desarrollado diversas herramientas automáticas, que, en su mayoría, utilizan algoritmos en los que las acciones a ejecutar en cada paso se deciden aleatoriamente. Esto es eficaz en aquellas aplicaciones inmaduras que han sido poco testeadas y presentan muchos errores. Dotar de "inteligencia" a los mecanismos de selección de acciones constituiría un importante avance para conseguir una mayor implantación de las herramientas de testeo automatizado, lo que redundaría en un incremento de la calidad del software. Éste es precisamente el objetivo de este trabajo.

Para conseguirlo, se ha utilizado un enfoque *basado en búsqueda* (o *search-based*) que convierte el testeo en un problema de optimización. Nuestro punto de partida es la herramienta TESTAR, desarrollada en el ámbito del proyecto de investigación europeo FITTEST. Se han utilizado y evaluado dos métodos: Q-learning y programación genética. Otro resultado importante son la definición de las métricas apropiadas para optimizar; en este trabajo se han introducido cuatro nuevas métricas.

La combinación de los algoritmos *search-based* con estas métricas ha permitido obtener resultados muy prometedores, que redundarán en la mejora de TESTAR.

Palabras clave: técnicas de búsqueda, test automatizado a través de la IGU, TESTAR, IGU.

Abstract

Graphic User Interfaces (GUI) are a main entry point to test an application. Different automated tools to test at the GUI level exist. Those that automate the design of test cases usually use random algorithms to choose the action that should be executed next in the test sequence. This technique is quite useful in applications that are immature, have been poorly tested or present many errors. To give more "intelligence" to this action selection mechanism, in this work we suppose a great development in the implantation of the automated testing tools. This improvement will result in better testing.

To achieve this, we use search-based techniques to transform the problem into an optimization one. Our starting point is the tool called TESTAR, a tool developed during an EU research Project called FITTEST. Two different methods have been implemented and evaluated: Q-learning and genetic programming. Another results of our work is the definition of metrics that guide the optimization properly. Four new and different metrics have been introduced.

The combination between metrics and search-based algorithms has been assessed and promising results have been obtained that will increase TESTAR capabilities.

Keywords: search-based techniques, automated GUI testing, TESTAR, GUI.

Tabla de contenidos

Índice de tablas

Índice de Figuras

1. Introducción

El testeo de aplicaciones software a través de las Interfaces Gráficas de Usuario (IGU) es una tarea fundamental durante la etapa de pruebas. La IGU se considera un punto vital en muchas aplicaciones, ya que representa el único punto de acceso del usuario a las mismas. Al contrario que los test unitarios, donde los componentes son puestos a prueba aislados del resto, las pruebas sobre la IGU de un software se realizan sobre todo el conjunto a la vez, es decir, como un gran conjunto que integra otros elementos más pequeños. De esta forma, es posible descubrir problemas o ineficiencias entre la comunicación entre componentes. Sin embargo, debido a que las IGUs se diseñan para ser utilizadas por humanos, puede resultar muy complejo testear toda la aplicación a través de ellas. Además, las interfaces están sujetas a cambios frecuentes que hacen que sea muy difícil, incluso imposible (en aplicaciones muy grandes), desarrollar y mantener casos de prueba sin que les suponga un enorme consumo de tiempo a los testers.

Para solucionar el excesivo tiempo requerido por las pruebas manuales, han aparecido muchas herramientas de testeo automáticas que facilitan dicha labor. La automatización en el diseño y la ejecución de los casos de prueba a través de su interfaz gráfica no es una práctica que se realice en muchas empresas. Normalmente, todas estas pruebas se diseñan y ejecutan manualmente por testers o incluso desarrolladores.

TESTAR es una herramienta de testeo automatizado que realiza pruebas a través de las interfaces, para generar estas pruebas utiliza la API de accesibilidad proporcionada por el sistema operativo. Esta API permite reconocer los controles que conforman la interfaz y sus propiedades permitiendo la ejecución de casos de prueba y la programación de interacciones con los componentes de la interfaz. El objetivo de TESTAR es la generación automática de casos de prueba, estos casos deben facilitar la detección de fallos en la aplicación. Para conseguirlo, TESTAR genera un conjunto de acciones para cada estado en el que la IGU se encuentra, elige una de ellas y la ejecuta. Esta forma de actuar genera una secuencia de testeo “al vuelo”, es decir, que no se programa con anticipación y después se ejecuta, sino que se ejecuta y se diseña a la vez. En trabajos anteriores hemos mostrado como TESTAR ha sido aplicado con éxito en varias aplicaciones comerciales de escritorio [1.] [2.] [3.] .

Inicialmente, TESTAR seleccionaba las acciones que debía ejecutar de forma totalmente aleatoria. Se diseñó así porque se consideró que era una forma directa y efectiva de encontrar errores y por su sencillez de implementación, además cuando se puso a prueba generó buenos resultados. Este comportamiento aleatorio permitía realizar pruebas inesperadas por los diseñadores, simplemente generadas por el azar. Sin embargo, descubrimos que esto tiene sus limitaciones, por ejemplo es muy difícil de testear las partes más profundas de la aplicación y que debido a la facilidad de acceso a ellas, algunas acciones van a ser muchas más veces elegidas que otras. Por estas razones, hemos elegido implementar algoritmos más inteligentes que nos permitan realizar una mejor selección de acciones a ejecutar, con el objetivo de que TESTAR encuentre más errores y ponga a prueba un mayor porcentaje de la interfaz. En concreto vamos a probar dos técnicas: Q-learning como técnica de aprendizaje por

refuerzo y programación genética, que es un tipo de algoritmo evolutivo, este algoritmo permite generar una serie individuos que irán mejorando y adaptándose a la aplicación.

Además, para comprobar la efectividad, se necesita una forma de medir los resultados. Por esta razón se va a desarrollar métricas que nos permitan comparar los resultados obtenidos y elegir los métodos de selección que mejores resultados proporcionen.

Se ha descubierto que ambos algoritmos basados en la búsqueda han generado mejores resultados que la selección aleatoria, probando así la utilidad de utilizar algoritmos más "inteligentes". Incluso para el primer algoritmo de Q-learning han aceptado ya dos artículos que se presentarán el meses que viene:

[3.] Francisco Almenar, Anna I. Esparcia-Alcázar, Mirella Martínez, and Urko Rueda , "Automated testing of web applications with TESTAR. Lessons learned testing the Odoo tool". SSBSE challenge 2016. Proceedings of the annual symposium on to Search Based Software Engineering (SSBSE), Raleigh, North Carolina, USA, 2016.

[22-] Anna I. Esparcia-Alcázar, F. Almenar, M. Martínez, U.Rueda, and Tanja E.J. Vos, "Q-learning strategies for action selection in the TESTAR automated testing tool", Proceedings of 6th International Conference on Metaheuristics and Nature Inspired Computing META'16, Marrakech, Morocco, Oct 27-31, 2016.

Estos son los primeros avances para hacer de TESTAR una herramienta de testeo que no necesita saber cómo testear, pero es capaz de aprenderlo por sí mismo.

La estructura de este TFM es la siguiente. En la sección 2 se han descrito los objetivos del trabajo, mientras que en la sección 3 se ha descrito lo que es el testeo, testeo automatizado y el testeo de interfaces gráficas. En el apartado 4 se ha realizado una descripción de TESTAR, aclarando su funcionamiento. A continuación, en la sección 5, se puede ver una descripción detallada del problema que hemos encontrado en TESTAR y de las métricas desarrolladas para medir su calidad. Durante la siguiente sección se ha explicado los diferentes algoritmos de búsqueda implementados. En la sección 7 se ha hablado de qué herramientas hemos utilizado para probar la eficacia de los algoritmos, mientras la sección 8 recoge cuales son los pasos seguidos durante la ejecución de las pruebas. En la sección 9 se recogen los resultados y para terminar en las secciones 10 y 11 se presentan las conclusiones y el trabajo futuro.

2. Objetivos

Las herramientas como TESTAR tienen mucha utilidad a la hora de encontrar errores, sin embargo, estas herramientas suelen tener problemas a la hora de elegir qué acciones ejecutar en cada momento y la mayoría de ellas las toman de forma aleatoria o simplemente reproduciendo secuencias generadas por humanos.

El objetivo último de este trabajo es mejorar TESTAR para que sea capaz de encontrar mayor número de errores en el software.

Con la intención de mejorar el estado actual de TESTAR, vamos a emplear técnicas de optimización basadas en la búsqueda. Con esto pretendemos dotarle de mayor "inteligencia" a la aplicación con el fin de mejorar su toma de decisiones.

En concreto vamos a aplicar:

- Q-learning, técnica que aplica aprendizaje por refuerzo, y
- Un algoritmo evolutivo que a través de un proceso de probar y evaluar soluciones consigue evolucionar la elección de acciones para obtener mejores resultados.

Para ser capaz de comprobar si estos algoritmos mejoran la elección de acciones se necesita una forma de medirlo que nos permita guiar los algoritmos. Intuitivamente, la manera de medir la calidad de una herramienta de testeo sería a través del número de errores encontrados; sin embargo esto no siempre va a ser posible. Por ello se requieren métricas sustitutivas, que permitan evaluar y comparar los resultados obtenidos, siendo esto otro de los objetivos de este trabajo.

Por último, otro de los objetivos será evaluar las técnicas citadas testeando varias aplicaciones reales, analizando los resultados y comparándolos entre ellos.

3. Testing o pruebas software

Antes de entrar en el desarrollo de los objetivos, es necesario argumentar y presentar la necesidad de las pruebas del software, del coste que conllevan y la necesidad de automatizar este proceso. Además se hablará del testeo a través de interfaces gráficas como el medio para lanzar las pruebas.

Para tener éxito en el proceso de pruebas, Glenford J. Myers [32], define el testeo como el proceso de ejecución de un software con la intención de descubrir un error. De esta forma, un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error.

Un caso de prueba se define como un conjunto de condiciones bajo las cuales se puede indicar si la característica funciona como se espera o no.

Las pruebas se tienen que realizar sobre todos los elementos de una aplicación. Por ejemplo, poner a prueba la lógica del negocio, el funcionamiento de las interfaces gráficas o incluso el rendimiento de la aplicación (comprobar el consumo de recursos que tiene la herramienta).

A partir de estas pruebas se puede averiguar que fallos tiene el sistema testeado y gracias a ello mejorar la calidad del software, corregir errores e incrementar el rendimiento de la aplicación. Además, nos permiten conocer el estado de desarrollo de una aplicación, gracias a esto podemos tomar decisiones con mayor precisión. Por ejemplo, podemos averiguar si una aplicación está lista o no para salir al mercado y conocer las consecuencias que puede conllevar su salida, pudiendo predecir de cierta manera sus valoraciones y la satisfacción que podría tener el usuario final.

Las pruebas son realmente importantes en la industria del desarrollo de software, debido principalmente a que la confianza de los clientes es fundamental. Sin ella los clientes no se atreverán a comprar tus productos y por lo tanto, se hará difícil recuperar la inversión que supone desarrollar una herramienta software. Sacar al mercado una aplicación con demasiados fallos puede provocar una pérdida de reputación para la empresa e incluso crear resentimiento por parte de los usuarios, sobre todo si ha supuesto para ellos grandes problemas o pérdidas.

En lo que queda de sección vamos a hablar sobre el testeo automatizado como solución a los problemas, sobre todo temporales que conlleva el testeo. Además, describiremos en qué consiste el testeo de las interfaces gráficas de usuario como campo específico de los programas donde se pueden aplicar pruebas.

3.1 Testeo automatizado

Para todas las actividades que componen un buen proceso de testeo (planificación, diseño, ejecución y evaluación) existen herramientas para automatizar parte de las

actividades [5]. En este trabajo nos centramos en la automatización del diseño de los tests y su ejecución.

La automatización del testeo es un campo de la ingeniería del software, en concreto de la rama de la calidad del software, que está en auge debido a la utilidad que está demostrando tener en su aplicación en la industria [6]. Sin embargo, se trata de un campo aún en desarrollo y lejos de haber alcanzado su máximo. Se considera un campo difícil debido a que diseñar casos de prueba testeo es una tarea difícil que requiere de la experiencia del tester para sacar el mayor partido al tiempo empleado, no obstante, debido a los beneficios que puede aportar se considera un campo que merece la pena ser explorado.

A continuación vamos a hablar de algunas ventajas y desventajas del testeo automatizado.

Automatizar la ejecución del testeo tiene muchas ventajas, por ejemplo permite realizar pruebas de forma continua sin consumir grandes cantidades de tiempo a los testers. Además, la ejecución de los test manuales es una tarea repetitiva y tediosa, esto aumenta la probabilidad de introducir de errores humanos provocados por el cansancio, el aburrimiento o las prisas, entre muchas otras causas.

La automatización permite que este tipo de errores desaparezcan; no obstante, tiene como desventaja la necesidad de dedicar un tiempo inicial, antes de lanzar las pruebas, para preparar dicha automatización.

Una vez configuradas las pruebas automáticas podemos ejecutarlas todas las veces que queramos. Gracias a esta configuración, se puede comprobar cómo responde nuestra aplicación a la ejecución reiterada de una de sus funcionalidades.

Tenemos que tener en cuenta que por lo general, la ejecución de pruebas automatizadas es más rápida su versión manual. Sin embargo, tras lo que acabamos de ver queda patente que si se quiere hacer una cantidad de pruebas pequeña, el tiempo de preparación en la automatización puede ser mayor a lo que ganemos automatizando dichas pruebas.

Cabe destacar que otra de las ventajas que proporcionan es el ahorro temporal que obtienen los testers, pudiendo emplear ese tiempo en pruebas más complejas que no sean sencillas de automatizar. Gracias a esto se reduce la cantidad de errores y se mejora la calidad del software.

Uno de los problemas más importante de estas herramientas radica en el diseño y desarrollo de comportamientos más complejos. Una de las causas para esto es que es difícil diseñar un comportamiento que siempre obtenga buenos resultados porque cada aplicación es muy distinta a las demás y lo que funciona en una no tiene porque hacerlo en otra. Además, otro de los problemas consiste en conseguir unas métricas que nos permitan cuantificar la mejora que obtenemos con respecto a la versión original.

3.2 Testeo de Interfaces Gráficas de Usuario

Las Interfaces Gráficas de Usuario (IGU) componen de media entre el 45% y el 60% [7] del código de una aplicación. En las herramientas comerciales, la interfaz es la forma que tiene el usuario de comunicarse con la aplicación y por lo tanto, son los intermediarios entre los servicios y los usuarios. Por esta razón, se considera que realizar pruebas a través de las interfaces de usuario permite localizar errores en el funcionamiento de la aplicación. Así, se aprovecha esta entrada a la aplicación para generar pruebas que aseguren la calidad del software.

Hacer pruebas a través de las interfaces de usuario puede resultar tedioso y complejo, sobre todo por el tamaño que puede llegar a tener. A continuación ponemos algunos ejemplos que ilustran la complejidad de las pruebas:

- En los formularios, los valores que introduce el usuario son muy susceptibles, dado que normalmente se suelen almacenar en bases de datos o tienen restricciones de valores, por ejemplo solo se pueden introducir números. Si tuviéramos que comprobar al menos un elemento de todos los posibles en cada caso de prueba se podría hacer eterno, sin embargo, gracias a la automatización solo habría que crearlo una vez, y ya podríamos ejecutarlo todas las que quisiéramos.
- Elementos invisibles. A veces para facilitar las pruebas los desarrolladores crean elementos no visibles que están ahí facilitan las pruebas, pero que no deberían ser accesibles para los usuarios normales. Gracias a estas herramientas sería mucho más fácil localizarlas.
- Enlaces entre ventanas. Las interfaces gráficas normalmente suelen estar conectadas entre sí, por lo que una buena práctica radicaría en probar todos estos enlaces, no obstante, el número de ellos puede ser muy grande. Por esta razón, si se hace forma automática el tester se puede ahorrar mucho tiempo.

En la actualidad, las empresas están introduciendo herramientas de testeo automatizado en sus procesos [8]. Existen varias herramientas que automatizan el testeo de las aplicaciones a nivel del IGU [9] [10], por ejemplo TESTAR, Selenium[33], Murphy tools[34]. La agrupación de estas herramientas es muy variada y depende de la fuente consultada, en nuestro grupo las dividimos en tres grupos:

- “capture and replay”,
- “Visual testing” y
- “Traversal testing”.

Actualmente, la mayoría de las herramientas de testeo para las interfaces de usuario pertenecen al grupo de Capture and Replay (C&R). Estas herramientas permiten a los usuarios grabar las acciones que han ejecutado, así pueden repetir esas pruebas cada vez que quieran. Estas secuencias grabadas se almacenan en forma de scripts. Al tratarse del grupo más extendido, existen muchas herramientas tanto comerciales como de código libre, como por ejemplo Microsoft Coded UI para Windows, Selenium para navegadores o Appium para móviles. Las herramientas C&R son muy populares principalmente porque son muy intuitivas y sencillas para empezar. Sin embargo,

tienen un gran problema pues estos scripts tienden a romperse fácilmente si la IU sufre cambios aunque sean pequeños. Este problema provoca que muchos expertos tengan que invertir mucho tiempo manteniendo estos scripts en lugar de crear nuevos. Con la aparición de la nueva generación de aplicaciones basadas en Internet, este problema se ha acrecentado porque estas se estructuran dinámicamente a las necesidades de los usuarios.

Para solucionar estos problemas de mantenimiento, han surgido las otras dos propuestas. El "Visual-based UI test" o testeo visual utiliza técnicas de reconocimiento de imágenes para poder interpretarlas y traducirlas a la estructura interna del programa. EggPlant y JAutomate son dos de los principales productos de estos grupos. Al utilizar reconocimiento visual, estas herramientas son más resistentes a los cambios en la interfaz. Sin embargo, este tipo de herramientas tienen también otros tipos de dificultades, por ejemplo al extraer información solo a través de una imagen, se complica bastante la obtención de la información, sobretodo información precisa de la estructura interna de la herramienta.

Las herramientas de testeo de recorrido (Traversal-based testing), como por ejemplo TESTAR y Murphy, inspeccionan la interfaz con el objetivo de comprobar las propiedades generales, entre otras para comprobar que no se queda bloqueada la interfaz. La estructura de la aplicación se obtiene como reflejo a partir del estado de la interfaz, esto provoca que el problema de mantenimiento no le afecte, dado que la estructura no es fija sino que se calcula cada vez. Este método permite examinar toda la jerarquía de los elementos que existen en la pantalla, lo que nos permite interactuar con ellos con precisión. La forma más fácil de aplicar estrategias de recorrido es seleccionar las acciones de forma aleatoria, ya que la aplicación de estrategias más inteligentes puede complicarse en exceso. Además, se crea una dependencia con la tecnología que produce la reflexión y sus limitaciones pueden afectar a la herramienta. En otras palabras si la herramienta bajo pruebas no es reconocida por la tecnología de reflexión, la herramienta de testeo no sirve para nada. TESTAR es una herramienta que se clasifica en este grupo, utiliza la API de accesibilidad de Microsoft como tecnología para obtener la reflexión (obtener la estructura a partir de la interfaz) de la herramienta que va a ser testeada.

4. TESTAR

TESTAR es una herramienta de código libre, está desarrollada en Java. La herramienta ha sido desarrollada en el grupo PROS en el ámbito del proyecto europeo FITTEST (Future Internet Testing) [11]. Este proyecto duró cuatro años, desde el 2010 hasta el 2013 y contó con la participación del centro de investigación de métodos de producción Software -PROS- de la Universitat Politècnica de València, la Universidad de Utrecht, University College London (UCL) y de las empresas Berner&Mattner, IBM, Bruno Kessler, y el grupo SOFTEAM.

TESTAR es una herramienta que permite la automatización del testeo de aplicaciones web, de escritorio y móviles a nivel de IGU. Se trata de una herramienta de código libre disponible bajo la licencia BSD3.

Actualmente la herramienta se encuentra en un estado de prueba de concepto, durante el cual se ha desplegado en entornos reales, es decir, se ha utilizado para testear herramientas que se están desarrollando actualmente y que necesitan medios para ponerlas a prueba.

Se ha elegido TESTAR como herramienta con la que trabajar debido a que su estructura se encuentra separada en capas, lo que permite que la modificación de una de ellas sea más sencilla, al estar desarrollada en Java, se puede desplegar en casi cualquier plataforma. Además, debido a que interactúa con un API desarrollada por Microsoft y que forma una parte fundamental del kit de desarrollo de Windows, se corre poco riesgo de quedarse obsoleta. Otro argumento que motivó a su elección ha sido ha sido la participación del autor de este trabajo en el desarrollo de la aplicación como becario. Por último, esta herramienta ya ha sido implantada en diversas empresas y algunas de ellas han mostrado su interés en la mejora de selección de acciones.

Las implantaciones en empresas han mostrado el potencial de la herramienta y los beneficios que podría aportar a estas empresas. Esto fue una motivación inicial para el presente proyecto. TESTAR en su versión actual, selecciona las acciones de forma aleatoria. Aunque es rápido y fácil de usar, en ocasiones esta selección de acciones a ejecutar y el orden de las mismas no son óptimos, lo que provoca que queden secciones de la interfaz a través de las cuales no realizamos pruebas.

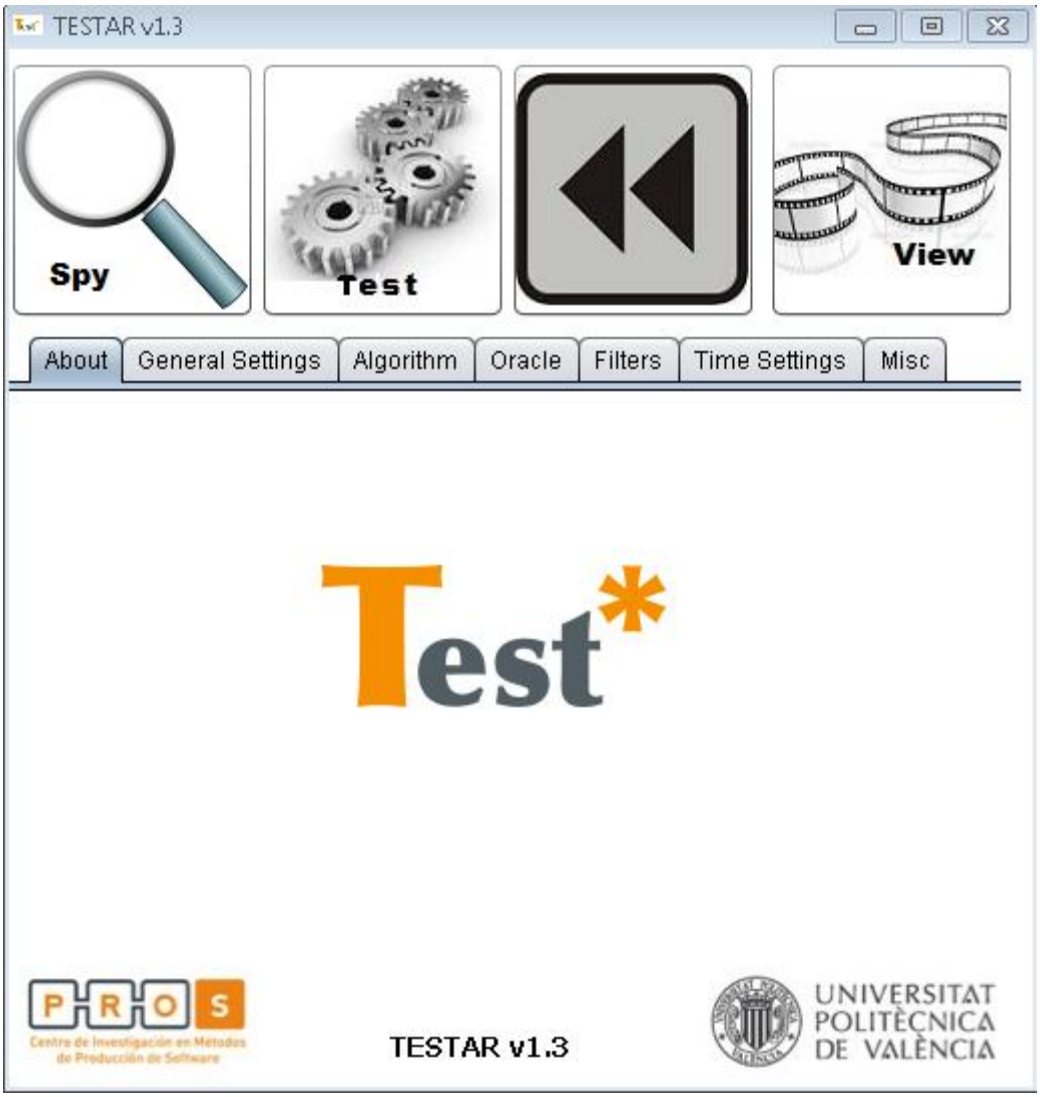

Figura 1: Ventana principal de TESTAR

Una de las ventajas de TESTAR es que no necesita acceder al código de la aplicación, por lo que solo necesita una forma de ejecutar la aplicación para funcionar. Esto lo consigue gracias a que lo que detecta los elementos gráficos que componen la pantalla, es el propio sistema operativo. Sin embargo, sin más información TESTAR no es capaz de predecir qué es lo que van a provocar las acciones que aún no ha ejecutado y no puede averiguar cuanto queda por ejecutar, puesto que no conoce la totalidad de la aplicación, solo la parte de la interfaz que ha explorado.

TESTAR funciona a través de la API de accesibilidad proporcionada por Microsoft [12]. Esta API reconoce los elementos gráficos que se están mostrando actualmente en pantalla y proporciona información sobre ellos, indicando el tipo que elementos de que se trate (si son botones, imágenes, cuadros de texto…), su posición en la pantalla, su nombre, los eventos que se les pueden aplicar… Esta información será utilizada por TESTAR para generar su representación interna, que recoge de forma jerarquizada todos los elementos que componen el **estado** actual de la IGU. Definimos **estado** como la situación de los elementos que componen la IGU en un momento dado, es decir, es la composición de todos los elementos y todos sus atributos que nos permitirían reconstruir la interfaz. El estado de la interfaz se deriva de forma

automática cada vez que TESTAR interactúa con la herramienta que está siendo testeada, esto permite que se pueda ejecutar la herramienta sin necesidad de indicarle nada más que la aplicación sobre la que se van a ejecutar las pruebas.

Para sacarle el máximo partido a la herramienta, es necesario un proceso de adaptación. Sin embargo, TESTAR permite su aplicación a herramientas desde el principio mediante el uso de configuraciones por defecto. TESTAR cuenta con un protocolo, que no es más que una clase Java que permite modificar el comportamiento por defecto de la herramienta. Esto nos permite adaptarla mejor a la aplicación sobre la que se van a lanzar las pruebas, reduciendo los problemas de detección y mejorando su rendimiento.

El protocolo permite controlar qué acciones se pueden ejecutar y sobre qué elementos. En el contexto de la herramienta se define como **acción** a la interacción con un elemento de la interfaz que puede provocar un cambio en la misma, por ejemplo hacer click izquierdo en un botón o escribir en un cuadro de texto.

La ejecución de TESTAR se divide conjuntos de acciones denominados secuencias. Una ejecución puede tener una o varias secuencias y cada secuencia tiene un número fijo de acciones que se han de ejecutar. Estos dos parámetros (número de secuencias y número de acciones por secuencia) deben ser especificados en la herramienta antes de iniciar las pruebas.

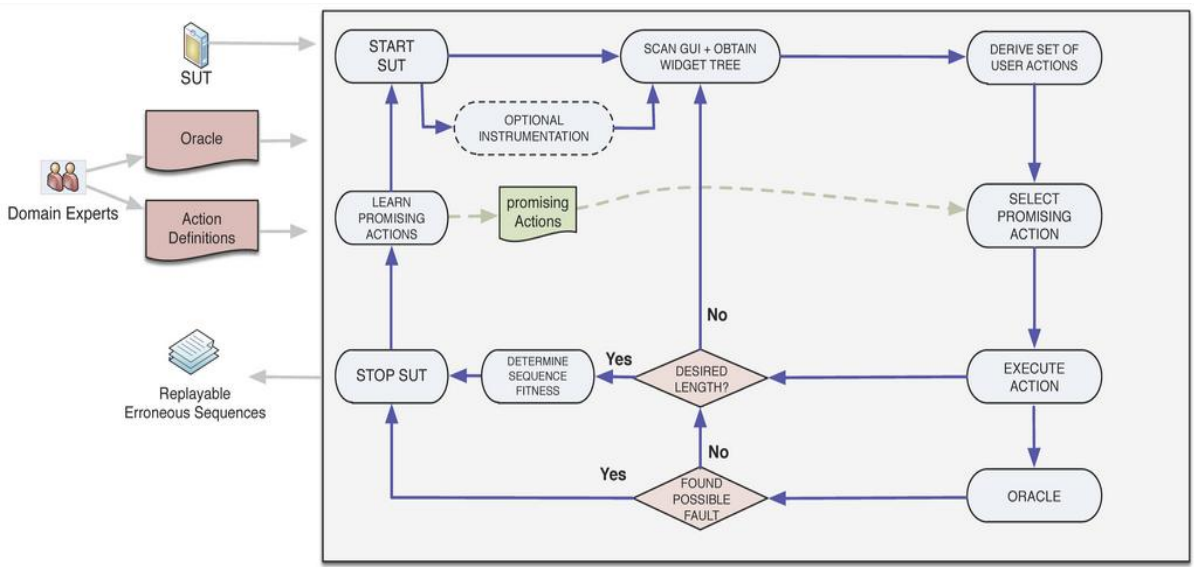

Figura 2: Ciclo de TESTAR

El funcionamiento de TESTAR se puede ver reflejado en la Figura 2: Ciclo de TESTAR. En ella podemos ver que el ciclo de ejecución es un bucle que termina sólo si hemos encontrado un error o si hemos alcanzado el criterio de parada, que viene dado por número de acciones o por tiempo transcurrido.

El ciclo de funcionamiento está divido en dos etapas, la una primera que es manual y se centra en la adaptación de TESTAR a la aplicación que va a ser testeada y una segunda etapa que se centra en la ejecución de las pruebas.

La primera etapa comienza con la definición de los oráculos y de las acciones que va a poder realizar la herramienta. Un oráculo se define como una serie de instrucciones que permiten comprobar si la ejecución de una acción ha causado un problema en el sistema.

Una vez definidos estos dos puntos, comienza la segunda etapa y se inicia la herramienta. TESTAR escaneará la IGU identificando todos los elementos que la componen, tanto aquellos con los que se puede interactuar, como los que actúan como contenedores y no se puede realizar ninguna acción sobre ellos. Con esta información construye un árbol en el que se representan los distintos componentes relacionados jerárquicamente. Es decir, se muestran las relaciones padre-hijo (relación jerárquica) las cuales representa que un elemento de la interfaz contiene a otro. Este árbol se genera una vez por cada iteración del bucle actualizándose con la nueva información que muestra la IGU.

A partir del árbol y de las reglas especificadas en el protocolo, TESTAR crea una lista con todas las posibles acciones que puede ejecutar en dicho estado. A continuación, se utiliza un algoritmo de selección para elegir cuál de todas estas acciones se ha de ejecutar, inicialmente el método de elección era aleatorio, en este proyecto se han implementado dos algoritmos: Q-learning y programación genética, con el objetivo de mejorar la elección de acciones. La acción seleccionada pasa a ser ejecutada, tras lo cual el oráculo comprueba si se ha producido algún error.

El siguiente paso es comprobar las dos condiciones de salida:

- Error. Comprueba si ha habido un error durante la ejecución. Existen diversas fuentes para detectar estos errores, por ejemplo que la aplicación haya dejado de funcionar, que haya saltado un error o una excepción o bien que se haya encontrado una palabra “peligrosa”. Este último tipo de error se activa cuando alguno de los elementos gráficos contiene en su nombre una palabra que puede indicar un fallo, por ejemplo, que contenga “null pointer” o “out of bounds”.
- Por tamaño o duración. Al ejecutar TESTAR indicamos cuál va a ser el número máximo de acciones que va a ejecutar y el tiempo máximo que puede dedicar a hacer si pruebas, si se alcanza cualquiera de los dos límites y no ha ocurrido ningún error, la secuencia de testeo también acaba.

Hay un caso específico en el que TESTAR añade un paso adicional, si estamos entrenando un algoritmo evolutivo, tras la finalización de las pruebas se calcula la utilidad que ha tenido el individuo. El funcionamiento del algoritmo evolutivo se detalla en la sección 6.2.

Si ha habido algún error, la ejecución se terminará reportando el error; en caso contrario se volverá a realizar la etapa de ejecución.

La arquitectura de TESTAR está divida en tres capas, el núcleo que recoge todas las funcionalidades de la aplicación, la capa *plugin* que es dependiente del Sistema Operativo donde se ejecuta y adapta la herramienta a dicho sistema. La última capa recoge la interfaz gráfica de la aplicación y la gestión del protocolo. Esta arquitectura en capas permite que el desarrollo en cualquiera de ellas no afecte al resto, por lo que resulta más cómodo modificar la aplicación.

5. Optimizar el proceso de selección

El objetivo principal de este trabajo es mejorar la toma de decisiones a la hora de seleccionar que acciones debe ejecutar la herramienta de testeo automatizado. Por tanto, hay que optimizar dicho proceso de selección.

Como primera aproximación, se podría pensar que la mejor forma de encontrar todos los posibles errores a través de la IGU pasa por probar todos los elementos gráficos de todas las formas posibles. Sin embargo, debido al tamaño que tiene una aplicación relativamente compleja esto es inviable en la práctica. La principal causa de ello es el tiempo necesario para conseguirlo, un tiempo que la mayoría de las empresas no están dispuestas a esperar. Por esta razón, el testeo se debe centrar en probar la mayor parte de la interfaz en el menor tiempo posible, es decir, intentar asegurar la calidad del software dentro de unos márgenes temporales asumibles por las empresas.

El proceso de generación de casos de prueba, en TESTAR, se traduce como el proceso de derivación de las acciones posibles y la selección de la más adecuada. Inicialmente, dicha selección se realizaba de forma aleatoria [2], es decir, que se determinaban todas las acciones que era posible ejecutar en cada estado y se elegía al azar cual ejecutar.

Como se describe en [13], este tipo de selección tiene ciertas ventajas aunque también desventajas. Con respecto a las ventajas, cabe destacar que el coste temporal es muy pequeño, puesto que se requiere hacer grandes cálculos para decidir qué hacer. Además, ha demostrado tener mucha utilidad para realizar pruebas de lo inesperado: dado que las acciones elegidas no siguen una lógica, es relativamente fácil que la herramienta acabe realizando una serie de acciones en las que nadie había pensado y por lo tanto tendentes a causar errores. Al tratarse de acciones inesperadas, es fácil que oculten errores que pueden suponer un problema en manos de los usuarios finales.

Otra ventaja de la selección aleatoria es que es muy fácil de mantener, dado que los cambios en la interfaz no afectan a su ejecución. Además, se trata de un modelo que no requiere un proceso de aprendizaje y por lo tanto, se puede ejecutar desde el principio sin emplear tiempo en nada más.

De entre las desventajas derivadas de este modo de selección de acciones, destaca la baja probabilidad de que acceda a las secciones más profundas de la interfaz o a aquellas que necesitan que se realice un proceso complejo para alcanzarlas. Además, se puede acabar repitiendo muchas pruebas porque no se tiene en cuenta las veces que se han ejecutado ni la utilidad de las acciones. Esto provoca que se consuman recursos ejecutando acciones que pueden no aportan nada.

Con el objetivo de mejorar el procedimiento de selección de acciones y que por lo tanto, se mejore la calidad de los casos de prueba, en este trabajo se han planteado alternativas a la selección aleatoria, que se describen en la Sección 6.

5.1 Métricas para guiar la optimización

El objetivo principal de TESTAR es encontrar el mayor número posible de fallos. Sin embargo, debido a que no sabemos el numero de errores en un aplicación con antemano (si lo supiéramos, no haría falta testar más), es necesario buscar alternativas al simple conteo de errores para medir la calidad de las pruebas, de forma que se puedan comparar distintas soluciones. La hipótesis es que, gracias a estas métricas sustitutivas podamos guiar a los algoritmos para encontrar un mayor número de errores. En este apartado nos centraremos en describir las alternativas que hemos definido.

Una revisión del estado del arte [14.] [15.] [16.] reveló que la mayoría de métricas se centraban en aplicaciones de escritorio y no cubrían todos los puntos que considerábamos necesarios, como por ejemplo la profundidad de la exploración. Otro problema residía en el hecho de que la mayoría se utilizaban para herramientas de testeo que tenían acceso al código fuente, lo que las volvía inútiles ya que TESTAR no accede al código fuente de la aplicación durante sus pruebas.

Para hacer frente a estas limitaciones se han definido unas nuevas métricas en las que basar la optimización:

- **Estados visitados**. Esta métrica hace referencia al *número de pantallas distintas* que se ha visitado durante la ejecución. Se consideran pantallas distintas a aquellas situaciones de la interfaz que tengan al menos un nuevo elemento gráfico o bien, tengan uno menos. Cabe destacar que no se considerarán pantallas distintas a aquellas en las que sólo varíe el contenido de un campo, por ejemplo que una tabla tenga una fila o una columna más, o que el contenido de un campo de texto sea distintas (p.ej. “Francisco” en lugar de “Juan”).
- El **camino más largo**. Todo algoritmo de testeo automatizado debería asegurar que incluso las partes más profundas de la interfaz se ponen a prueba. Para medir si la herramienta ha llegado a las zonas más profundas y no se ha quedado en la superficie, definimos el camino más largo como la *secuencia de pantallas no repetidas* (por ejemplo excluimos los bucles) consecutivas.
- **Cobertura mínima y máxima por estado**. El porcentaje de cobertura por estado se define como el ratio del número de acciones ejecutadas y del total de las acciones disponibles por cada uno de los estados. Las métricas miden el menor y el mayor valor del porcentaje de cobertura por cada secuencia completa de acciones.

Estas métricas juntos con primeras evaluaciones nos lo han aceptado como artículo para presentarlo en el International Symposium on Search-Based Software Engineering (SSBSE) en octubre 2016 en North Carolina, USA

6. Técnicas de búsqueda para testear

En el campo de la ingeniería del software existe una rama que busca aplicar técnicas de búsqueda, esta rama recibe el nombre de **Ingeniería del Software Basada en Búsqueda** (ISBB) o en inglés **Search-Based Software Engineering** (SBSE). Este concepto fue acuñado hace 15 años por Mark Harman [17.] y desde entonces este campo ha crecido considerablemente.

Las técnicas de búsqueda se basan en la utilización de técnicas de optimización o metaheurísticas, como algoritmos genéticos, la búsqueda tabú[18] o el recocido simulado (simulated annealing) [19]. Zanakins y Evans, definen una metaheurística como "procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido". (1981) [20]. Dicho con otras palabras, las técnicas de búsqueda utilizan el conocimiento de expertos sobre el problema para generar un algoritmo que, en función de ese conocimiento, permita generar soluciones al problema, aunque no tenga porqué ser la mejor.

Estas técnicas también se puede aplicar en el testeo de las aplicaciones, sobre todo en el campo del testeo automatizado. Como hemos comentado anteriormente, nuestra herramienta de automatización realiza un proceso de selección de acciones, que se hacía inicialmente de forma aleatoria; el objetivo de este trabajo consiste en aplicar metaheurísticas a este proceso y comprobar si mejoran la elección aleatoria.

Hemos decidido aplicar estas técnicas porque la naturaleza del problema de selección cumple razonablemente con los requisitos para los que este tipo de soluciones son aceptables, tales como la gran cantidad de posibles acciones donde elegir, con unos requisitos vagos, pues lo que queremos es encontrar errores, pero es difícil averiguar que es un error exactamente y definírselo al programa y sin embargo, buscamos la solución óptima o al menos una cercana a ella.

Para comprobar la utilidad de estas técnicas vamos a emplear dos algoritmos distintos, Q-learning y Programación genética. El primero consiste en una técnica de aprendizaje por refuerzo, donde la exploración de estados y acciones nuevas se ve recompensada. Por otro lado, el segundo se basa en la evolución de "programas" (en nuestro caso, reglas de decisión) que se evalúan poniéndolos a prueba en el problema dado. El proceso evolutivo cosiste en la generación de nuevas reglas a partir de las anteriores, basándose en la calidad, o fitness, de éstas. Eventualmente, se obtendrán reglas se adaptarán mejor al problema, proporcionando mejores resultados. El proceso se repite hasta que se generen reglas que proporcionen unos resultados por encima de la calidad deseada, o bien cuando se supere un umbral de tiempo determinado. En los próximos subapartados vamos a describir con mayor detalle el funcionamiento de ambos algoritmos.

6.1 Q-learning

Q-learning es [21] una técnica que aplica recompensas y penalizaciones cada vez que ejecuta una acción: si ésta puede dar buenos resultados recibirá una recompensa, y en caso contrario una penalización. Por esta razón, el objetivo del algoritmo será maximizar la recompensa, mientras evita ser penalizado. Debido a que se considera de gran importancia la exploración, aquellas acciones no ejecutadas generarán mayores recompensas, mientras que aquellas acciones que ya han sido ejecutadas varias veces no darán mucha recompensa, por lo que acabarán penalizando los resultados del mismo.

Esta técnica de búsqueda incluye la capacidad de "mirar al futuro", esto se debe a que a la hora de seleccionar que acción se va a ejecutar, se tiene en cuenta las posibilidades actuales y las que podríamos tener en ese estado futuro al que vamos a caer. Es decir, que a la hora de elegir la siguiente acción se mira la recompensa que se puede obtener en este estado y se predice la que podría sacar como máximo en el futuro estado en el caería, siendo máxima la recompensa si la acción seleccionada no ha sido ejecutada y si en el estado que se alcanza existe al menos una acción que no haya sido ejecutada.

Este algoritmo no descarta las acciones que ya han sido ejecutadas, por lo que éstas pueden volver a serlo. La razón por la que no las eliminamos es que algunas aplicaciones necesitan que una acción se ejecute varias veces poder acceder a otras partes más profundas de la interfaz, como por ejemplo pulsar varias veces en el botón siguiente durante la visualización de los términos de uso. Debido a que algunas acciones han de ser ejecutadas más que otras, es necesario que el algoritmo pueda mirar más allá de la acción actual.

Para solventar dichas necesidades, Q-learning incluye dos parámetros adaptables, la recompensa y el descuento. De esta forma, se puede influir en el comportamiento por defecto del algoritmo, dando mayor prioridad a las acciones actuales o a las futuras.

La recompensa ($\boldsymbol{r_m}$) se utiliza para medir cuán favorable resulta ejecutar una acción, ya sea porque nunca ha sido ejecutada o porque nos lleva a partes de la interfaz que no han sido muy exploradas, aumentando la probabilidad de que dichas acciones sean elegidas. Por otro lado, el descuento (**γ**) se utilizar para medir cuanta importancia tiene mirar al futuro para elegir la siguiente acción, cuanto más menor sea este valor, menor importancia tendrá la predicción de lo que podemos obtener en el futuro estado.

A continuación pasamos a describir con mayor precisión el funcionamiento del algoritmo. El primer paso es iniciar el SUT, y asignar un valor por defecto de la recompensa ($\boldsymbol{V(s,a)}$) a cada acción (**a**) en el estado (**s**) de -1. Una vez iniciado el SUT, obtenemos todos los elementos gráficos que componen el estado actual y derivamos el conjunto de acciones disponibles sobre ese estado, y calculamos la recompensa para cada una de las acciones. Si el valor era -1, significa que es la primera vez que esa acción ha estado disponible dentro del conjunto de acciones ejecutables y por lo tanto, al valor de la recompensa total ($\boldsymbol{Vt(s,a)}$) se le asigna el valor de la recompensa máxima multiplicada por un parámetro corrector($2^{x_{a \in A_w}}$), por el contrario si el valor es distinto de -1, entonces el valor de la recompensa total de la acción será igual a la recompensa por ejecutar esa acción ($\boldsymbol{V(s,a)}$) sumado al valor de descuento multiplicado por la máxima recompensa ($V_{(s',a')}$) que se puede conseguir de entre todas las acciones

disponibles en el siguiente estado (**s'**). Si el estado que se va a alcanzar es desconocido, es decir, la aplicación no lo ha alcanzado antes, entonces ese valor es igual a $\boldsymbol{r_m}$ multiplicada por el parámetro corrector.

$$Vt(s,a) \leftarrow \begin{cases} \dfrac{r_m}{x_{a \in A_t} * \ 2^{x_{a \in A_w}}} & \forall a \in A_s \ \wedge \ V(s,a) = \ -1 \\ V(s,a) + \gamma \cdot \max\limits_{a'}\{V(s',a') | a' \ \in \ A_{s'} \ \wedge \ S(a) = s'\} & \forall a \in A_s \ \wedge \ V(s,a) \neq \ -1 \end{cases}$$

Una vez calculada la recompensa total (***Vt***) para cada acción, seleccionaremos aquella cuya recompensa sea la más grande, ejecutaremos esa acción y recalcularemos su valor de recompensa. Tras esto recalcularemos los valores de las recompensas en función del número de veces que ha sido ejecutada esa acción (x_a). Esta secuencia se repetirá hasta que se cumpla el criterio de parada especificado en TESTAR, es decir que se haya detectado un error, que se haya ejecutado el número de acciones especificado o bien que haya transcurrido el tiempo máximo especificado. Una vez se ha cumplido el criterio de parada y siguiendo el ciclo de ejecución de TESTAR, se pasa a cerrar el SUT.

A continuación podemos ver cómo funciona el algoritmo en el siguiente pseudocódigo:

Entrada: 0 < γ < 1 /* γ = descuento */

Entrada: 0 < r_m /* r_m = máxima recompensa para las acciones sin ejecutar*/

Inicio

  Iniciar SUT

  $V(s,a) \leftarrow -1 \quad \forall (s,a) \in S \ x \ A$

  repetir

    obtener el estado actual s y todas acciones disponibles A_s

    $$Vt(s,a) \leftarrow \begin{cases} \dfrac{r_m}{x_{a \in A_t} * \ 2^{x_{a \in A_w}}} & \forall a \in A_s \ \wedge \ V(s,a) = \ -1 \\ V(s,a) + \gamma \cdot \max_{a'}\{V(s',a') | a' \ \in \ A_{s'} \ \wedge \ S(a) = s'\} & \forall a \in A_s \ \wedge \ V(s,a) \neq \ -1 \end{cases}$$

    $a^* \leftarrow \max_a\{Vt(s,a) | a \ \in A_s\}$

    ejecutar a^*

    $V(s,a^*) \leftarrow 1/x_a \cdot 1/\log x_{a \in A_t}$

  hasta que se cumpla el criterio de parada

  detener el SUT

Fin

$A_t \subset \ A_s \ = \{a | \ acciones \ de \ tipo \ t\}$ /* t = botón izquierdo, escribir … */

$A_w \subset A_s = \{a | \ acciones\ sobre\ la\ componente\ w\}$ S = {estados}, A = {acciones}

El parámetro recompensa máxima puede adoptar casi valor positivo, mientras que el descuento solo puede adoptar valores comprendidos en 0 y 1. Según la elección de estos parámetros, el rendimiento del algoritmo puede variar severamente, además que no existe una combinación de parámetros definitiva, por lo que dependiendo del problemas una configuración puede ser mejor que otras que en otros problemas la superaban.

En este trabajo hemos elegido la misma combinación de parámetros (r_m =1.000.000 y $\gamma = 0.95$) para todos los problemas, de esta forma podremos comparar de una manera más justa los resultados obtenidos.

Para decidir que combinación de parámetros vamos a utilizar hemos realizado un estudio [3][22], donde se probaron distintas combinaciones de parámetros y se analizó cuál de ellas generaba mejores resultados.

Las combinaciones seleccionadas fueron Q1, Q20, Q99 y Q10M, los valores de los parámetros han sido recogidos en la Tabla 1. Q1 tiene una recompensa 1 y un descuento de 0,20, se compensa muy poco la exploración de nuevas acciones y no se tienen casi en cuenta las acciones en el nuevo estado. Q20 cuya recompensa de 20 y su descuento es de 0,20, le da mayor importancia a ejecutar acciones nuevas ahora que en el futuro. Q99 tiene un valor de 99 en su recompensa y de 0,50 en el descuento, esto hace que la importancia de ejecutar acciones nuevas esté a la par que tener en cuenta el siguiente estado. Por último, Q10M que tiene 9999999 de recompensa máxima y 0.95 de descuento, le da muchísima importancia tanto a ejecutar nuevas acciones como a tener en cuenta el futuro.

Tabla 1: Parámetros evaluados

	Parámetros	
Nombre	Recompensa Máxima	Descuento
Q1	1	0,20
Q20	20	0,20
Q99	99	0,50
Q10M	9.999.999	0,95

Para poner a prueba Q-learning ejecutamos las cuatro combinaciones de parámetros en dos herramientas distintas, Odoo y Power-Point. Las pruebas consistieron en lanzar 30 ejecuciones de 1000 acciones cada una y analizar estadísticamente los resultados; este análisis es idéntico al descrito en la sección 9.

Como resultado de estas pruebas obtuvimos que la combinación ganadora en ambos casos fue Q10M y por lo tanto, se ha decidido utilizar esta combinación para todas las pruebas que utilicen este algoritmo.

Los resultados obtenidos para cada una de las métricas se han recogido en las tablas Tabla 2: Resultados para Odoo y Tabla 3: Resultados para Power-Point. En estas tablas podemos ver ordenadas de izquierda a derecha las combinaciones de parámetros que mejores resultados dieron para cada una de las métricas, por ejemplo en los resultados

de Odoo, en estados explorados, la que mejor resultados ha dado es Q10M, mientras que Q20 es la que peores resultados ha generado, por otro lado en cobertura máxima, Q99 es el que mejores resultados ha obtenido, mientras que Q1 el que peores.

Tabla 2: Resultados para Odoo

Métrica	Combinaciones			
Estados explorados	Q10M	Q1	Q99	Q20
Camino más largo	Q10M	Q99	Q1	Q20
Cobertura mínima	Q10M	Q20	Q1	Q99
Cobertura máxima	Q99	Q20	Q10M	Q1

Tabla 3: Resultados para Power-Point

Métrica	Combinaciones			
Estados explorados	Q10M	Q20	Q99	Q1
Camino más largo	Q99	Q20	Q10M	Q1
Cobertura mínima	Q10M	Q20	Q99	Q1
Cobertura máxima	Q1	Q20	Q10M	Q99

Como podemos ver, no existe una combinación que sea superior en todo. Esto es normal debido a que, como hemos comentado anteriormente, debido a la especificidad de los algoritmos de búsqueda, no existe una combinación que sea perfecta para todos las aplicaciones ni para todos los casos dentro de una misma aplicación. Por esta razón, lo que hay que elegir es que métricas tiene más prioridad, es decir, cuáles son las métricas que se ha considerado que nos permitirán acercarnos mejor a nuestro objetivo final, encontrar más errores. Para este proyecto, se ha considerado que la exploración de estados es más prioritaria, esto se debe a que se prevé una relación entre la exploración y los errores encontrados, es decir, se espera que la combinación que más explore sea la que más errores encuentra.

Teniendo en cuenta lo anterior, se pude ver que en ambos casos ha ganado la misma combinación hemos decidido aplicarla en lo sucesivo. Esta distribución prioriza sobre todo la exploración de nuevas acciones y estados, teniendo muy en cuenta la recompensa máxima que se ha predicho que se puede obtener en el estado de llegada.

6.2 Algoritmo evolutivo (programación genética)

Podemos definir un algoritmo evolutivo como un método de optimización y búsqueda que está basado en la evolución de los seres vivos [23]. Al igual que los seres vivos, estos algoritmos se adaptan al problema evolucionando poco a poco. El proceso que sigue un algoritmo evolutivo es el siguiente:

- Partimos de una población inicial, esta población consiste en una serie de algoritmos que podrían resolver el problema. No todos los individuos que conforman esta población inicial tienen por qué ser útiles, de hecho pueden crearse individuos que no supongan una solución factible.

- Evaluación de los individuos. Para cada posible solución se calcula la calidad de la misma. Para esto se define una función denominada *fitness* que nos permite asignar una "puntuación" a cada uno de los individuos.
- Comprobar si hemos encontrado una solución lo suficientemente buena. Se comprueba si al menos una de las puntuaciones supera un valor umbral especificado.
- Si no hay ninguno, comienza el proceso de generación de nuevos individuos. Elegimos que soluciones existentes se van a reproducir, normalmente se escoge las dos mejores soluciones. Estas soluciones se recombinan, es decir, se mezclan generando individuos que se parecen a sus antecesores. Además, como ocurre en la realidad, los individuos creados recientemente pueden sufrir un proceso de mutación, en el que el algoritmo de solución sufre un cambio de forma espontánea.
- Selección natural. Una vez se ha creado la siguiente generación eliminamos aquellos individuos que peor fitness tienen, simulando la selección natural. Esta eliminación representa que el individuo no se adapta igual de bien que el resto de la población al problema.
- Si hemos superado el umbral, se da por finalizado el proceso evolutivo y se usa el mejor individuo como solución al problema.

En la Figura 3: Algoritmo evolutivo podemos ver una representación gráfica del proceso descrito.

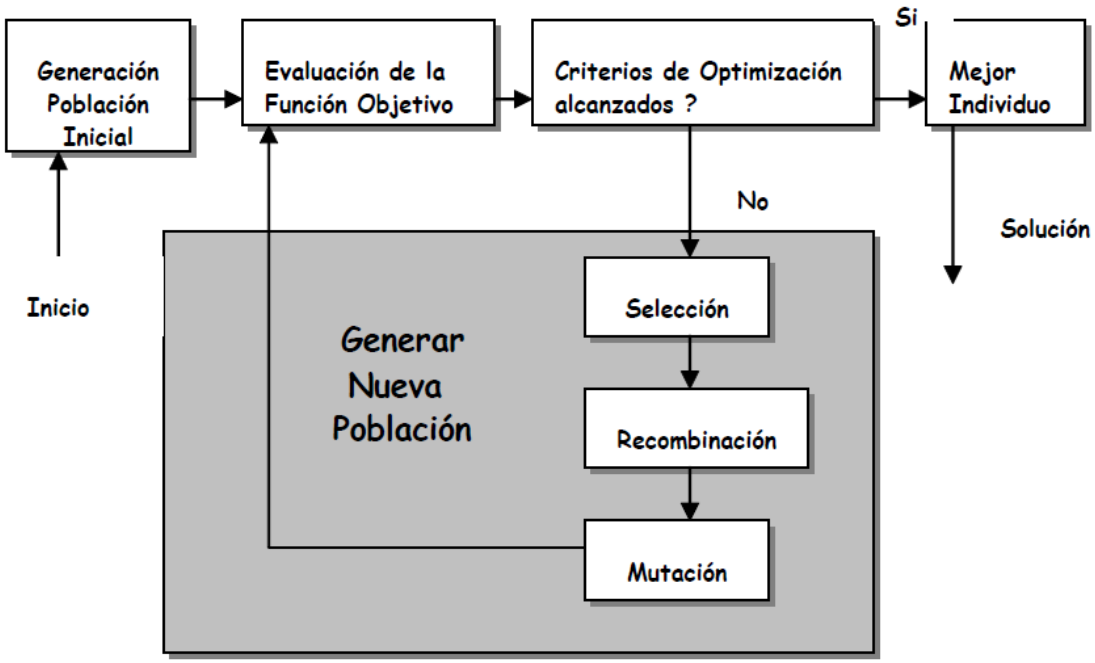

Figura 3: Algoritmo evolutivo

Existen una gran variedad de técnicas que implementan este tipo de evolución, por ejemplo los algoritmos genéticos, programación genética y las estrategias evolutivas. En este trabajo se ha decidido implantar una técnica de programación genética.

Hemos decidido emplear programación genética como técnica de implementación del algoritmo evolutivo debido a que presenta la mutación como su elemento de evolución principal, al permitir generar nuevas reglas mediante métodos evolutivos a partir de las reglas que se han generado originalmente. Además, se necesita que la estructura utilizada para conformar los individuos se mantenga y la programación genética sí que respeta la estructura, lo que no es el caso en otros métodos.

Antes de entrar en la implementación es necesario que hablemos de los puntos vitales en un algoritmo evolutivo, la representación del individuo y la función que calcula de fitness del mismo. Sin estos dos elementos bien definidos no se puede diseñar el resto del algoritmo, además son muy difíciles de diseñar porque requieren tiempo de maduración y mucho conocimiento sobre el problema.

En nuestra implementación representamos a los individuos mediante la estructura **if-then-else**, es decir, los individuos que conforman la población tendrán una condición si se cumple se realizará una acción (then), pero en caso de no hacerlo se hará otra (else). El método de almacenamiento de los individuos será JSON [24]. A continuación describimos cada uno de los tres componentes:

If. Se utiliza para indicar que lo que viene a continuación es la condición a valorar. La condición está formada de forma muy similar a las condiciones de las instrucciones condicionales de los lenguajes de programación, terminal – operador lógico – terminal, al igual que en la instrucción se puede anidar condiciones. Los terminales que podemos usar puede ser valores numéricos (int) o bien indicar el tipo de acción (tipoDeAccion). Los operadores lógicos solo se pueden aplicar a elementos de mismo tipo, es decir, int con int y tipoDeAccion con tipoDeAccion.

Los terminales que podemos usar son:

- nActions: Número de acciones disponibles para cada un estado (int).
- nTypeInto: Número de escrituras que se pueden realizar en un estado (int).
- nLeftClick: Número de clicks izquierdos del ratón que se pueden realizar en un estado (int).
- RND: Número aleatorio que se especifica cuando se crea el individuo, pero que no varía durante la ejecución (int).
- typeLeftClick: Indica que la acción es de tipo botón izquierdo (tipoDeAccion).
- typeTypeInto: Indica que la acción es de tipo escritura (tipoDeAccion).
- Any: Recoge absolutamente todas las acciones (tipoDeAccion).
- PreviousAction: Recoge el tipoDeAccion de la acción ejecutada anteriormente (tipoDeAccion).

Los operadores lógicos aceptados son:

- LT: Menor que. (2 inputs de tipo int).
- LE: Menor o igual que. (2 inputs de tipo int).
- EQ: Igual que. (2 inputs de tipo int o type).
- AND: Une dos condiciones y comprueba si ambas son verdad (2 inputs de tipo boolean).

- OR: Une dos condiciones y comprueba si al menos una de ellas es verdad (2 inputs de tipo boolean).
- NOT: Niega el valor de una condición, si era verdadera pasa a ser falsa y si era falsa para a ser verdadera (1 input de tipo boolean).

Por lo que un ejemplo de condición podría ser:

IF: nLeftClick LT nTypeInto AND previousAction EQ TypeInto.

Esta condición comprueba si hay menor número de acciones de tipo click izquierdo que de escrituras y además comprueba si la anterior acción fue una escritura. Si ambas condiciones se cumple se ejecutará la acción del Then sino la del Else.

Then. Representa la acción que va a ser ejecutada si se cumple la condición, está formada por una instrucción que selecciona la acción que se va a ejecutar.

El conjunto de acciones que puede ejecutar son:

- Pick: Recibe dos parámetros, el primero es un terminal de tipo tipoDeAccion, que indica cuál debe de ser el tipo de la acción elegida, mientras que el segundo (es opcional) hace referencia a cuál de todas las coincidencias vamos a elegir, si no se proporciona un segundo parámetro se considera que será TESTAR el que decida cuál escoger. Si el número proporcionado es mayor al número de acciones de ese tipo, se pasar a ejecutar una instrucción de tipo pickAny Ej: pick (LeftClick, 4), escogería la cuarta acción de tipo botón izquierdo disponible en la lista de acciones.
- PickAny: Recibe un parámetro de tipo tipoDeAccion, que indica sobre el tipo de acción vamos a escoger. Cuando empleamos esta función, la acción escogida es al azar.
- PickAnyUnexecuted: Al igual que la anterior instrucción, solo que en lugar de escoger aleatoriamente la acción, escogerá la primera acción no ejecutada.

Un ejemplo de Then podría ser:

THEN: pickAny(TypeInto)

Esta instrucción seleccionaría una acción al azar de tipo TypeInto, es decir, que ejecutaría una escritura.

Else. Al igual que en Then, Else representa una acción que va a ser ejecutada. Sin embargo, esta solo se ejecutará si no se cumple la condición de If. El conjunto de acciones que puede ejecutar son las misma que Then.

Un ejemplo de Else podría ser:

ELSE: pickAny(LeftClick)

Esta instrucción indica que en caso de no cumplirse la condición ejecutaríamos una acción al azar de tipo botón izquierdo.

Si juntamos estos tres campos podríamos formar un individuo, si recogemos los tres ejemplos tendríamos

IF: nLeftClick LT nTypeInto AND previousAction EQ TypeInto.

THEN: pickAny(TypeInto)

ELSE: pickAny(LeftClick)

Este individuo comprobaría si menor botones izquierdos que escrituras y si la acción anterior ha sido una escritura, si es así ejecutará otra escritura pero si no lo es ejecutará un botón izquierdo.

Como hemos comentado antes, para que TESTAR entienda el individuo, este ha de estar representado en formato JSON. El individuo anterior en este formato sería el siguiente:

[

"IF": "nLeftClick LT nTypeInto AND previousAction EQ TypeInto."

"THEN": "pickAny(TypeInto)"

"ELSE": "pickAny(LeftClick)"

]

Como uno de los campos más importantes que queremos medir es la exploración, representada por el número de estados distintos alcanzados, definiremos la función fitness como el número de estados explorados.

El resto de métricas definidas, aunque no se contemplan en la función de fitness, sí que se evaluarán en el análisis final de resultados.

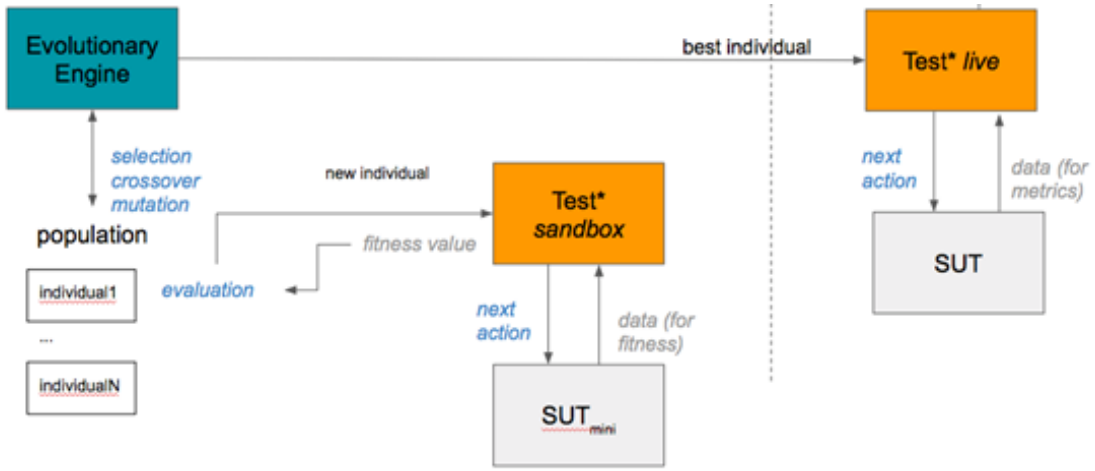

Figura 4: Proceso de aplicación del algoritmo evolutivo.

En la Figura 4: Proceso de aplicación del algoritmo evolutivo. se recoge los pasos que se han seguido una vez definidas las principales componentes del algoritmo. A continuación se procede a explicar con detalle cada uno de los pasos.

El proceso comienza con la evolución de los individuos. Para la evolución de los individuos vamos a seguir el esquema proporcionado anteriormente, sin embargo vamos a emplear una herramienta desarrollada para implementar la evolución.

La herramienta se llama Pony GP [25], ha sido desarrollada por grupo de investigación CSAIL del MIT. Al gestionar la propia herramienta el proceso evolutivo nuestra labor se ha centrado en especificar la estructura de los individuos, como realizar el cruzamiento y la mutación, realizar la invocación a TESTAR para que ejecute las pruebas con el individuo que ha elegido el algoritmo y por último leer el resultado de la fitness. Por lo tanto, Pony GP, realizará la población inicial, ejecutará el cruzamiento y las mutaciones y elegirá que individuos serán los padres de las nuevas generaciones.

Para la declaración de la estructura hemos indicado el número de parámetros que reciben, dejando a los terminales con un cero, dado que no reciben parámetros. Para el cruzamiento, hemos especificado que se pueden cruzar los valores tras las palabras clave if-then-else, sin embargo, los contenidos solo se podrán cambiar por otros del mismo tipo, por ejemplo podremos intercambiar las condiciones del if entre ellas pero no podremos cambiar una condición de un if, por una acción especificada en el else. Las mutaciones solo se pueden aplicar sobre terminales y permiten cambiar un terminal por otro del mismo tipo, es decir, que podremos cambiar un nLeftClick por un nTypeInto, pero no por un AND.

Dado que tras la compilación de TESTAR se genera un archivo tipo .jar[26], podemos ejecutar este fichero desde la Consola de Comandos de Windows [27] para iniciar las ejecuciones. Y para recoger los resultados de la fitness hemos modificado TESTAR para que cuando finalice la ejecución escriba en un fichero los valores finales; una vez hecho esto, basta con leer desde Pony GP los valores.

Como se trata de una primera aproximación al problema con el objetivo de probar la utilidad de estas técnicas, por esta razón los valores elegido están limitados en tamaño.

Al inicio de la ejecución Pony GP generará 20 individuos distintos que serán considerados la primera generación. , Pony GP empleará un método de selección denominado torneo, que consiste en seleccionar de forma aleatoria un numero de individuos de la población, cinco en este caso, y elegir de entre ellos a los dos individuos con mejor fitness. Los dos individuos elegidos serán los “padres” que se cruzarán generando un “hijo”. TESTAR utilizará a este hijo como mecanismo de selección para ejecutar las pruebas, devolviendo los valores de las métricas obtenidas, que se utilizarán para calcular la fitness. Las pruebas consisten en ejecutar Power-Point y lanzar una secuencia de 100 acciones[1] .TESTAR finalizará la ejecución cuando la métrica de estados generados para el individuo evaluado supere el valor de 20. Este individuo se incorporará a la población sustituyendo a aquél que tenga peor fitness; en caso de empate se elegirá a uno de ellos al azar.

Una vez finalizado el proceso evolutivo, el mejor individuo obtenido se utilizará como algoritmo de selección de acciones para ejecutar el ciclo de vida de TESTAR.

Para este proyecto se ha decidido utilizar el método evolutivo con sólo una herramienta, en lugar de ejecutarlo en las tres, sin embargo, el individuo creado sí que se utilizará como motor de selección para el todos las aplicaciones que van a ser testeadas. La razón detrás de esta decisión es que se espera de este algoritmo que tenga capacidad de generalización y aunque no se obtengan los mejores resultados que se

[1] Se ha elegido 100 acciones porque es un número que permite probar la utilidad del individuo sin consumir excesivo tiempo en las pruebas.

podrían obtener, sea capaz de generar buenos resultados aunque haya evolucionado para dicha aplicación.

El individuo que mejores resultados ha generado es el siguiente:

```
[
{
"IF": "nLeftClick LT nTypeInto",
“THEN": "pickAny(LeftClick)",
"ELSE": "pickAnyUnexecuted"
}
]
```

La herramienta elegida para evolucionar el algoritmo ha sido Power-Point. La razón principal es que se trata de una aplicación completa, que posee muchas funcionalidades y que además es compleja. Esto permitirá generar individuos robustos que sean capaces de dar buenos resultados en otras aplicaciones.

La Tabla 4: Resumen algoritmo evolutivo, muestra un resumen de las características del algoritmo evolutivo empleado.

Tabla 4: Resumen algoritmo evolutivo

Característica	**Valor**
Tamaño de la población	20
Tamaño máximo del árbol	El número máximo de individuos que se podía generar en total es de 200.
Funciones	Pick, PickAny, PickAnyUnexecuted, AND, OR, LE,EQ, NOT
Terminales	nActions, nTypeInto, nLeftClick, previousAction, RND,typeLeftClick, typeTypeInto, Any, Previous Action
Operadores evolutivos	Mutación y cruzamiento
Método evolutivo	Steady state
Método de selección	Selección por torneo, los grupos son de cinco individuos y de ellos se escogen los dos mejores
Criterio de finalización	Generar más de 30 estados distintos

7. Aplicaciones a testear

Para probar la eficacia de los algoritmos empleados vamos a poner a prueba tres aplicaciones distintas o SUTs, del inglés System Under Test. Las tres aplicaciones se pueden agrupar en dos tipos, aplicaciones de escritorio y aplicaciones web.

Los SUTs de escritorio son PowerPoint, un conocido software para la creación de presentaciones, y Testona, una aplicación de testeo y de validación técnica. Por otro lado, se va a testear Odoo, un programa de gestión integral de código abierto.

A continuación se detallan las características y funcionamiento de las distintas aplicaciones estudiadas.

7.1 Escritorio

Estas aplicaciones se caracterizan porque se instalan en un ordenador y se ejecutan en él, aunque pueden necesitar acceso a Internet para obtener datos. Una de las grandes ventajas que aportan es que son rápidas, dado que todos los eventos y los controles se encuentran instalados en el ordenador y no han de ser solicitados a un servidor.

Sin embargo poseen una serie de desventajas; por ejemplo necesitaremos descargarnos actualizaciones cada vez que el desarrollador cambia el programa o corrija algunos fallos. Además de la dependencia del sistema en el que se instala.

Recientemente han aparecido versiones portables de las aplicaciones de escritorio que no requieren instalación previa a su uso, sin embargo, suelen depender del sistema igualmente.

7.1.1 Power-Point

Power-Point es un programa de presentación en forma de diapositivas desarrollado por Microsoft, forma parte del paquete ofimático Microsoft Office y ha sido desarrollado para los sistemas operativos Windows y Mac OS.

Actualmente, Power-Point es uno de los programas de presentación más utilizados y destaca sobre todo en sectores como la enseñanza o los negocios. Las tres funciones principales de esta herramienta son: la inserción y formateo de texto, la inserción o modificación de imágenes y/o gráficos, la animación de los componentes de la presentación y la reproducción de las diapositivas generadas.

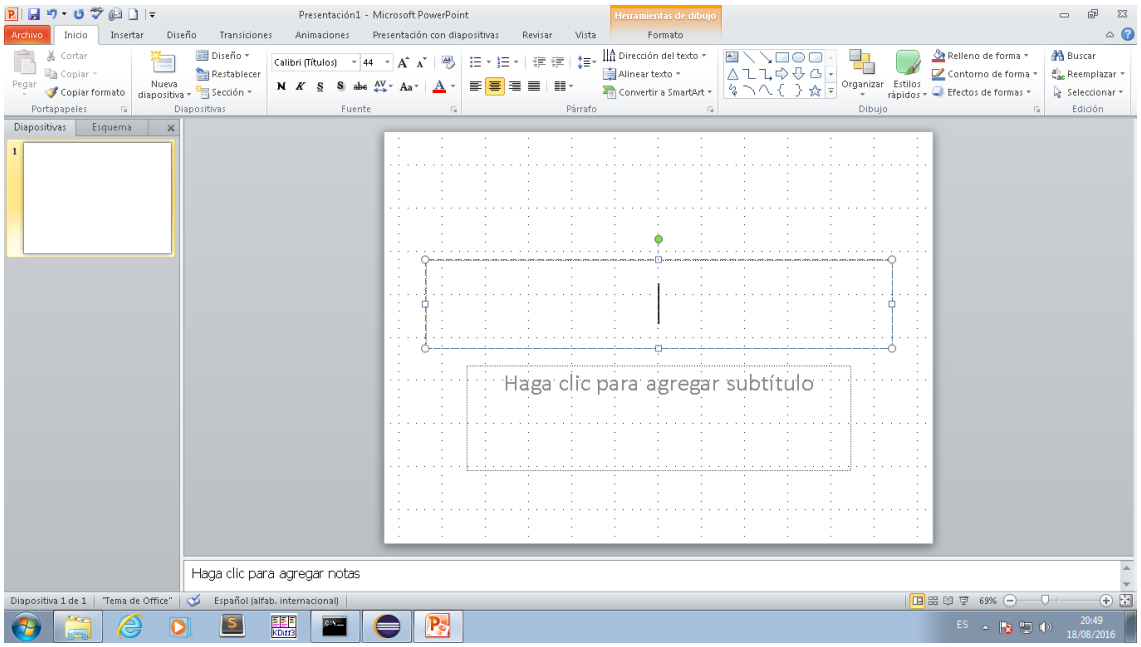

Figura 5: Interfaz Power-Point

Se trata de una aplicación de escritorio que requiere ser instalada antes de ejecutada, por lo que para poder realizar las pruebas hemos instalado esta aplicación en el sistema operativo Windows 7.

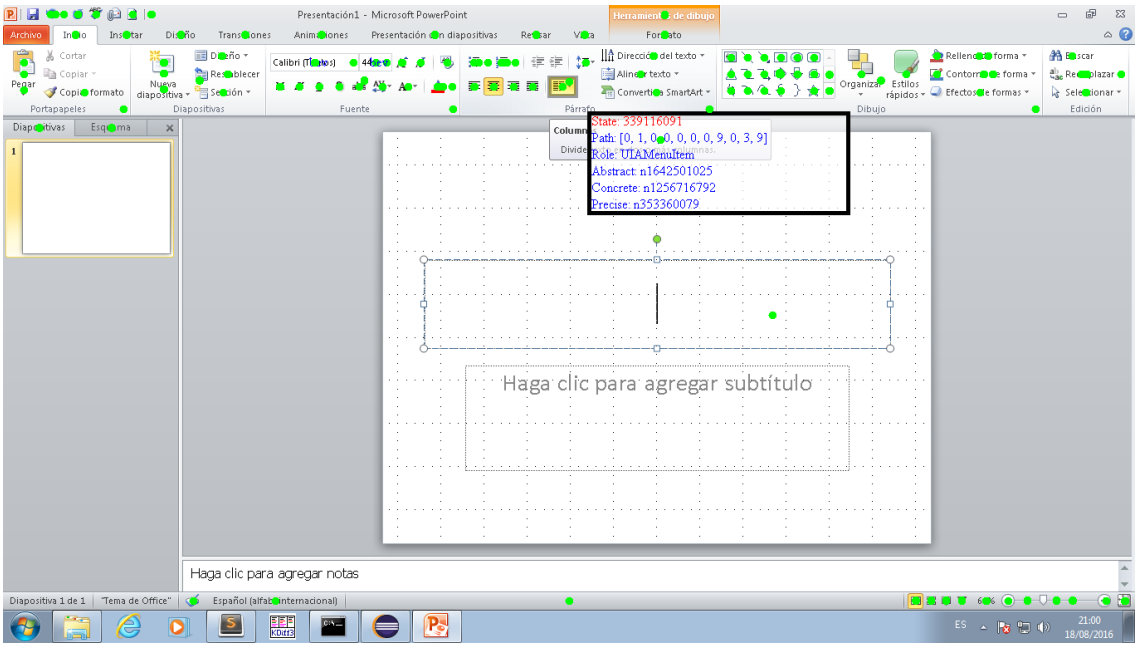

Figura 6: Interfaz Power-Point con TESTAR

7.1.2 Testona

Testona es una herramienta de testeo anteriormente conocida como *Classification Tree Editor*. Este software implementa el método de clasificación en forma de árbol, que consiste en clasificar el dominio de la aplicación sobre la que se van a ejecutar las pruebas en un árbol y asignar pruebas a cada una de sus hojas.

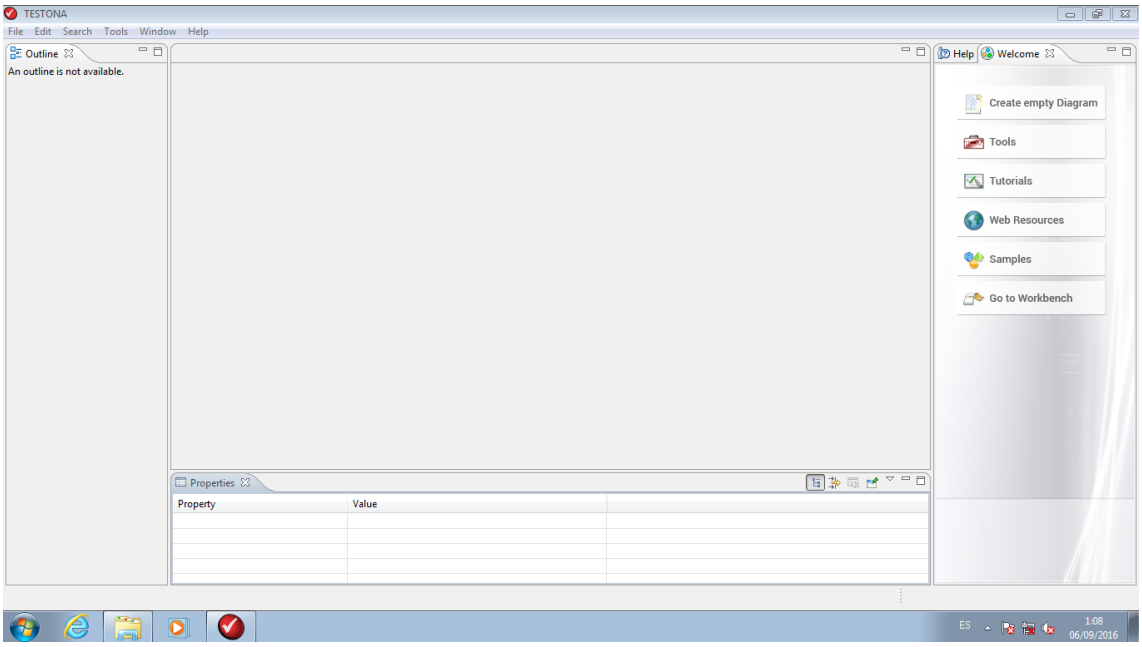

Figura 7: Pantalla inicial Testona

Con esta herramienta podemos derivar casos de prueba de una forma más sencilla y automática. Además, permite la integración con otras herramientas de testeo y de desarrollo automáticas.

Este software alemán tiene una versión gratuita pero las versiones completas con todas las capacidades hay que pagar. Sin embargo, hemos hablado con la empresa y nos ha proporcionado durante unos meses la versión completa, por lo que las pruebas se han realizado sobre esta versión. Hemos instalado la aplicación en un sistema Windows 7.

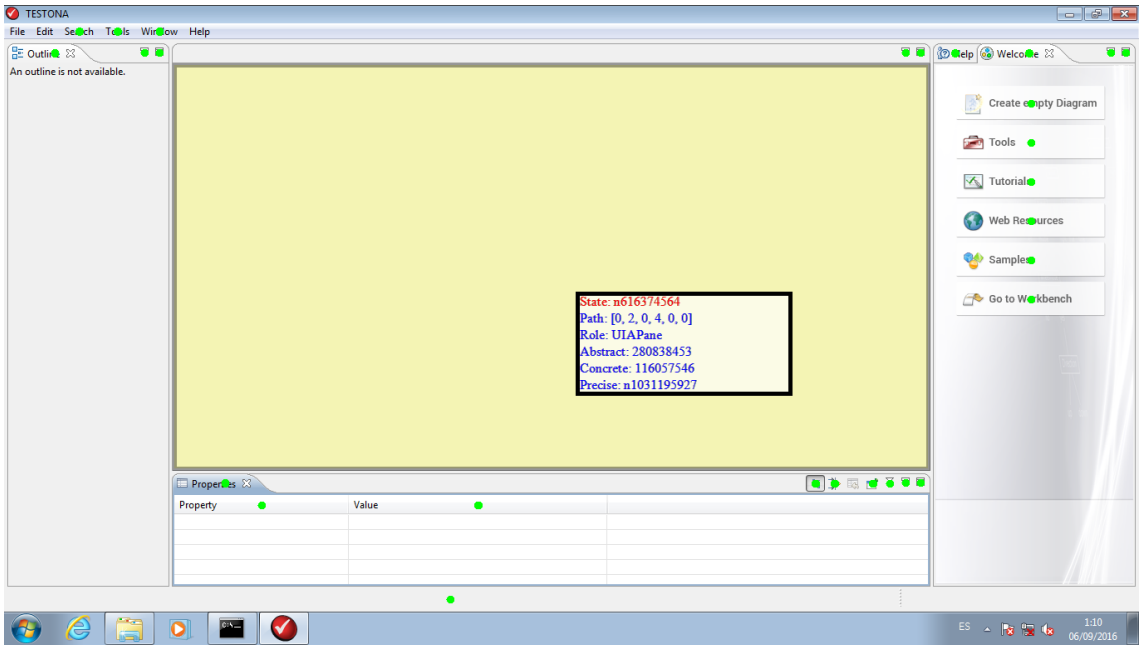

Figura 8: TESTAR sobre Testona

7.2 Aplicaciones web

Una aplicación web es una página web especial, que tiene información sobre la que se puede actuar o modificar. Al contrario que las aplicaciones de escritorio, éstas no necesitan ser instaladas en tu ordenador, sino que se instalan en uno o varios servidores y se ejecutan a través de tu navegador web.

Una de las principales ventajas de los navegadores web es su independencia de los sistemas operativos, pues basta solo con tener instalado un navegador web. No obstante, a veces sí que suelen haber limitaciones con respecto a qué navegadores y qué versiones de ellos pueden ejecutar la aplicación.

Otra ventaja es que las actualizaciones no suponen ningún problema para los usuarios, pues no suelen requerir ningún tipo de acción adicional.

Por otro lado una de sus mayores desventajas reside en su necesidad de estar conectados a Internet para funcionar, incluso para poder acceder a los servicios.

7.2.1 Odoo

Odoo [28] es un software para ERP, Enterprise Resource Planning, es decir un sistema informático destinado a la administración de recursos en una organización. Esta herramienta está formada por un conjunto de aplicaciones que pueden ser instaladas o no en función de las necesidades de la organización.

Podemos utilizar esta herramienta para crear sitios web, gestionar recursos humanos, el cálculo de las finanzas, gestión de las ventas y de proyectos. Odoo está basado en la arquitectura cliente servidor y utiliza una base de datos PostgreSQL para almacenar los datos de la empresa.

En la Figura 9: Interfaz de Odoo se puede ver como es parte de la interfaz del cliente odoo,. A la izquierda podemos el conjunto de opciones aplicables, mientras que a la derecha se muestran las principales acciones realizadas sobre el inventario, ya sean entregas, recepciones o transferencias internas.

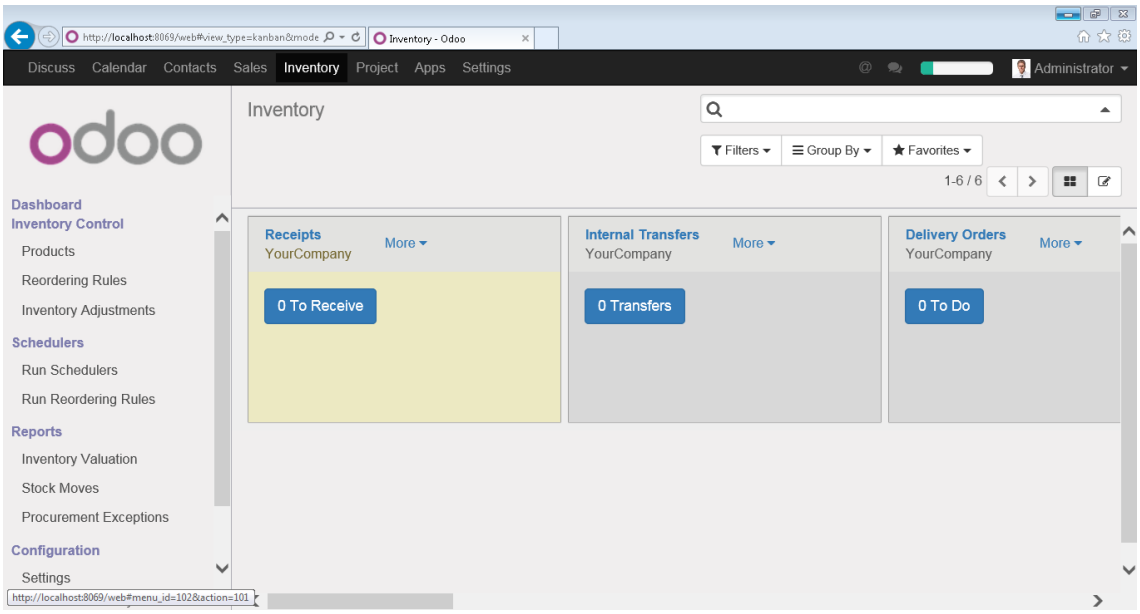

Figura 9: Interfaz de Odoo

Los módulos de Odoo están cubiertos, en su mayoría, por una licencia AGPL o una derivada Mozilla Public License. Al tratarse de licencias de abiertas, no se requiere realizar ningún pago para poder utilizarlo y se puede modificar el código fuente de la aplicación sin problemas.

El servidor consiste en una aplicación de código abierto escrita en JavaScript que atiende a las peticiones del cliente, mientras que para actuar de cliente vale cualquier navegador, pues solo es necesario saber realizar peticiones http y saber interpretar las respuestas y mostrarlas.

Para realizar las pruebas hemos instalado un servidor Odoo [29]. Como hemos comentado, Odoo está formado por una serie de aplicaciones más pequeñas, en este caso hemos instalado los siguientes módulos: correos, el calendario de tareas, contactos, ventas, inventario y gestión de proyectos. Por otro lado, como cliente hemos utilizado el navegador web Internet Explorer.

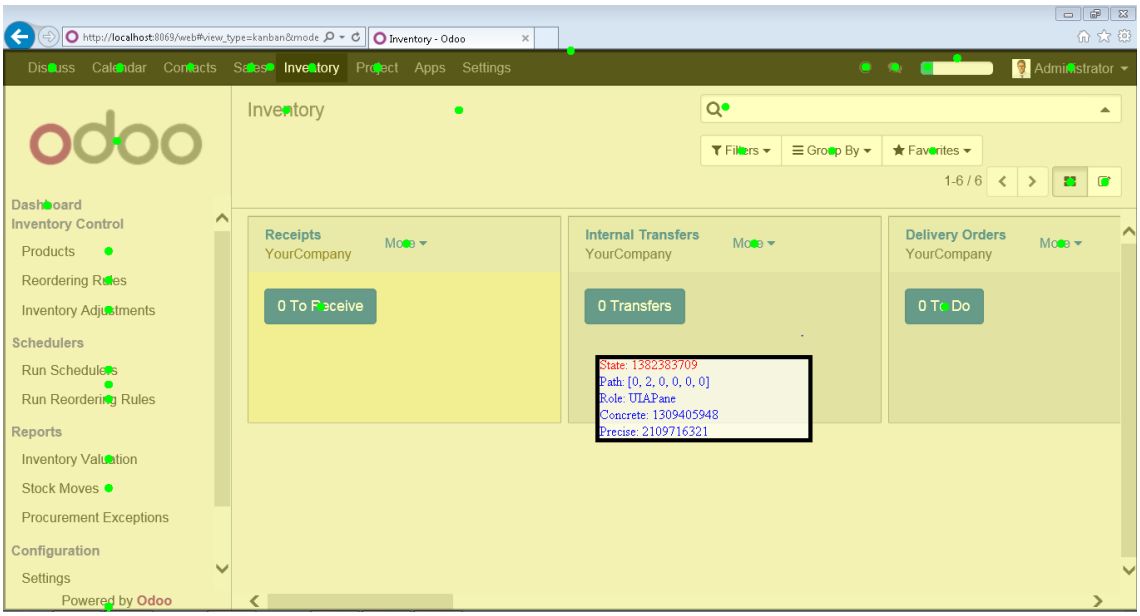

Figura 10: TESTAR en Odoo

8. Configuración experimental

En esta sección se describen los pasos que se han seguido a la hora de lanzar las pruebas sobre las cuatro aplicaciones. El proceso seguido ha sido igual para todas ellas, mientras que la etapa de mejora del protocolo ha sido específica de cada aplicación.

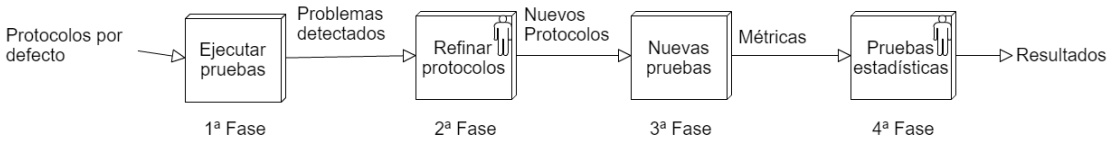

Figura 11: Proceso seguido

Como podemos ver en la Figura 11: Proceso seguido la primera fase consiste en utilizar los protocolos básicos de TESTAR según sean aplicaciones de escritorio o web (Ambos protocolos, así como todos los que se han realizado durante este proyecto, se encuentran adjuntos en los anexos de esta memoria), tras la ejecución de TESTAR con estos protocolos examinamos los resultados obtenidos y evaluamos los problemas que han tenido las ejecuciones y las corregimos; por ejemplo, observamos los falsos positivos obtenidos y a través de una mejora del protocolo indicamos al oráculo que esas acciones no se pueden considerar errores realmente. Por lo que la segunda fase consiste en mejorar los protocolos por defecto, cabe destacar que tras esto, cada protocolo será distinto al del resto de aplicaciones pues se hacen específicos de la herramienta. Una vez mejorados los protocolos comenzamos la fase tres, que consiste en volver a lanzar las pruebas sobre cada aplicación, sobre estas pruebas recopilaremos las métricas que se especificaron en la sección 5.1. La siguiente fase consiste en la ejecución de pruebas estadísticas para evaluar los resultados. El fin de estas pruebas es comprobar si hay mejoras en los distintos algoritmos y para demostrar que las ejecuciones realmente son diferentes. Una vez terminado todo este proceso, realizaremos una fase de análisis sobre los resultados por cada una de las herramientas, compararemos los algoritmos empleados y valoraremos la utilidad de los mismos a la hora de elegir que acciones ejecutar.

El número de ejecuciones a realizar en cada etapa de pruebas será el mismo para todos los SUTs; en concreto vamos a realizar 30 secuencias de 1.000 acciones cada una para cada herramienta por cada una de las dos fases de pruebas y por cada uno de los algoritmos, es decir, que en conjunto vamos a ejecutar 60.000 acciones por aplicación y algoritmo, por lo que al final probaremos 180.000 acciones sobre cada herramienta.

Como TESTAR no tiene forma de acceder al código fuente de las aplicaciones, no podemos saber qué porcentaje total de la aplicación hemos probado, pero si el porcentaje de exploración relativo a lo que hemos explorado, pues para cada estado sabemos el número de acciones que se pueden ejecutar (se deriva a partir de la API de Microsoft y de la configuración del protocolo de TESTAR), los que ya hemos ejecutado (se guarda un histórico) y los que quedan por ejecutar(sabiendo el total de acciones y las que se han ejecutado, podemos calcular las que quedan por hacerlo), sin embargo

como no podemos afirmar que es esta métrica sea suficiente, hemos decidido complementarlo con las métricas proporcionadas en la sección o .

Para finalizar esta sección adjuntamos a continuación una descripción somera de la mejoras que hemos aplicado a cada protocolo (los detalles se han incluido en los anexos). Las mejoras son:

- Power-Point. En esta herramienta podemos encontrar una gran cantidad de acciones que nos permiten acceder al sistema de archivos de Windows para guardar o abrir documentos. Estas acciones son peligrosas dado que TESTAR podría acabar borrando archivos importantes, por lo que se han filtrado usando el protocolo. Además, para evitar que se salga de forma prematura de la aplicación hemos filtrado todos los botones de cerrar y minimizar. Cabe destacar que TESTAR no detectaba algunas acciones como por ejemplo cambiar el tamaño de la letra. Por esta razón hemos introducido estas acciones como posibles.
- Testona. Al igual que en la herramienta anterior hemos filtrado todos las posibilidades de guardar o abrir documentos y de minimizar o cerrar la aplicación. Además, se ha revisado toda la interfaz en busca de frases que contengan la palabra error y que no lo sean, por ejemplo “This helps to detect the **error** rate of single classes within a test suite and gives a hint of possible defects of the system under test.” Durante las ejecuciones se descubrió que Testona tiene una función para ejecutar un servidor, este servidor necesitaba un tiempo de arranque largo, tan largo que TESTAR acaba pensando que la aplicación se había bloqueado y la cerraba, provocando un error inesperado en Testona.
- Odoo. Al tratarse de una aplicación web, hemos tenido que tener cuidado de testear el navegador sin darnos cuenta (por ejemplo pulsar los marcadores o utilizar la barra de navegación), por lo que hemos filtrado la toolbar completa del navegador. Esta herramienta necesita iniciar sesión antes de poder utilizarse, por lo que se ha automatizado TESTAR para que cada vez que se sitúe en la pantalla de inicio, introduzca el usuario y la contraseña e inicie sesión. Cuando estábamos realizando la primera fase, nos dimos de que la herramienta cada vez que intentábamos ejecutar una acción inesperada (intentar escribir en una imagen), lanzaba una excepción que TESTAR reconocía como error, aunque la aplicación podía seguir ejecutándose normalmente impedía a TESTAR terminar sus ejecuciones, pues al detectar el error terminaba la ejecución. Para poder seguir haciendo pruebas, se indicó al oráculo que esto no se cerrara cuando saltara esa excepción. Nos hemos visto obligados a filtrar todo aquellos enlaces que llevasen a páginas externas como por ejemplo Twitter o Facebook, dado que al pulsar sobre ellos, TESTAR se ponía a testear esas webs. Para finalizar, comentar uno de los refinamientos más difíciles aplicados ha sido la detección de ciertas componentes como las ventanas emergentes, las aplicaciones en general pueden mostrar ventanas que se superponen a la aplicación y te impiden interactuar con ella hasta que no se cierre, el problema que hemos encontrado es que la API de accesibilidad que usa TESTAR no puede detectar estas ventanas si se crean en las aplicaciones web, por lo tanto nunca cierra esas ventanas y nunca interacciona con la aplicación una vez aparecen.

Hemos tenido que enfrentarnos a varias de estas ventanas y a partir de referencias automatizar su cierre una vez aparecen.

9. Resultados y evaluación.

En esta sección se presentan los resultados obtenidos tras las ejecuciones, se compararán los resultados de los tres métodos de selección (aleatoria, Q-learning, Programación genética). Asimismo se mostrarán los resultados del análisis estadístico con el fin de comprobar si las diferencias son significativas.

Se va a analizar los resultados de cada una métricas para cada una de las aplicaciones, por lo que tendremos 12 comparativas diferentes (recogidas en la Tabla 5: Resumen de las pruebas) donde veremos el rendimiento de cada uno de los métodos de elección (aleatorio, Q-learning, programación genética).

Tabla 5: Resumen de las pruebas

Herramienta	Métrica
Power-Point	Estados explorados
	Camino más largo
	Cobertura mínima
	Cobertura máxima
Testona	Estados explorados
	Camino más largo
	Cobertura mínima
	Cobertura máxima
Odoo	Estados explorados
	Camino más largo
	Cobertura mínima
	Cobertura máxima

Para verificar si hay diferencias estadísticamente significativas entre los tres métodos, se han utilizado test estadísticos no paramétricos, dado que no se puede asegurar que los datos obtenidos siguen una distribución normal. Las pruebas devuelven valores cercanos a 0 cuando hay diferencias significativas y valores altos cuando no las hay.

En concreto hemos utilizado dos pruebas estadísticos:

Mann-Whitney-Wilcoxon [31.] . Se utiliza para comparar dos conjuntos de pruebas distintos y comprobar la heterogeneidad de las dos muestras. En método está ligado a dos hipótesis: la hipótesis nula, según la cual la distribución de partida de ambas muestras es la misma, y la hipótesis alternativa, según la cual los valores de una de las muestras *tienden a exceder* a los de la otra. Si el resultado (o p-valor) de aplicar Mann-Witney-Wilcoxon es distinto de 0 se rechaza la hipótesis nula con una significatividad del 5%.

Kruskal – Wallis[31]. Es una extensión del método Mann-Whitney para comparar tres o más grupos.

Las pruebas estadísticas se realizarán en dos etapas; en la primera aplicaremos Kruskal-Wallis (K-W) para comparar los tres algoritmos y en la segunda, aplicaremos Mann-Witney-Wilcoxon (MWW) para comparar cada pareja de métodos de selección.

Una vez establecido si existen diferencias, se utilizará diagramas de caja para visualizar qué distribuciones quedan por encima o por debajo del resto, para determinar qué algoritmo proporciona mejores resultados.

El test de K-W se ha aplicado cuatro veces por SUT, una vez por cada una de las cuatro métricas. Con respecto a los resultados, la mayoría de las pruebas permiten concluir que al menos uno de los algoritmos explorados obtiene resultados significativamente diferentes de los otros dos. Sin embargo, para la métrica de cobertura mínima tanto en Power-Point como en Testona el test no encuentra diferencias significativas entre los resultados. Lo mismo ha ocurrido para la métrica de cobertura máxima en la herramienta Testona.

A continuación se ha aplicado el test de MWW para comparar los conjuntos de datos por pares; de esta forma esperamos obtener información más específica sobre qué conjuntos de datos son los que se parecen.

Este test se ha utilizado tres veces por cada combinación aplicación-métrica. La comparación se ha realizado por pares, todos los valores del Q-learning se han comparado con los del aleatorio primer y después con el evolutivo, para finalizar se han comparado el aleatorio y el evolutivo entre sí. En total se han realizado 36 pruebas distintas.

La tabla 6 recoge los resultados de las pruebas; en ella podemos ver para cada aplicación y métrica si cada una de las combinaciones entre algoritmos son o no significativamente diferentes.

Tabla 6 Resultados de las pruebas estadísticas del método Mann-Witney-Wilcoxon

Aplicación	Métrica	Algoritmos	¿Son iguales?
Power-Point	Estados explorados	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Camino más largo	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Mínima cobertura	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	Si
		Aleatorio-Evolutivo	Si
	Máxima cobertura	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	Si
		Aleatorio-Evolutivo	No
Testona	Estados explorados	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Camino más largo	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Mínima cobertura	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	Si
		Aleatorio-Evolutivo	Si
	Máxima cobertura	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	Si
		Aleatorio-Evolutivo	No

Odoo	Estados explorados	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Camino más largo	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Mínima cobertura	Q-learning-Aleatorio	No
		Q-learning-Evolutivo	No
		Aleatorio-Evolutivo	No
	Máxima cobertura	Q-learning-Aleatorio	Si
		Q-learning-Evolutivo	Si
		Aleatorio-Evolutivo	No

Los resultados de la Tabla 6 Resultados de las pruebas estadísticas del método Mann-Witney-Wilcoxon muestran que aunque en la mayor parte de los casos los datos son distintos, existen casos en los que no hay diferencias significativas, sobre todo en las métricas de cobertura, esto nos hace pensar que quizás estas métricas no nos estén proporcionando información útil y que por lo tanto, hay que replanteárselas.

Podemos ver que el test MWW contradice los resultados de Kruskal-Wallis en el caso de Testona y máxima cobertura. Esto se debe a que en ambas pruebas el p-valor obtenido está próximo a 0.05 y por tanto los resultados están en una zona "gris" y no es posible determinar si existen diferencias con este nivel de significatividad.

Una vez determinado qué distribuciones son diferentes, se ha utilizado diagramas de caja para establecer cuáles son mejores con respecto a la métrica utilizada; estos diagramas están basados en cuartiles y por lo tanto, nos permiten visualizar la distribución de los datos. Están compuestos por una caja (que contiene el segundo y el tercer cuartil) y dos bigotes (el situado en la parte inferior de la caja representa el primer cuartil, mientras que el que está situado en la parte superior muestra el cuarto cuartil). Cada cuartil recoge el 25 % de los datos, por lo que la caja muestra el 50% de los datos recogidos, además dentro de esta caja localizamos una línea que señaliza el valor de la mediana.

En los resultados obtenidos en Power-Point se puede ver (Figura 12: Power-Point - estados explorados) que el método de selección que mayor número de estados ha visitado es el evolutivo, seguido de Q-learning. Lo mismo sucede en el caso del camino más largo generado: el que mejores resultados ha obtenido ha sido el algoritmo evolutivo, seguido de Q-learning y el que peor resultado ha obtenido ha sido el aleatorio. En el caso de la cobertura mínima, esta vez es Q-learning es que genera mejores resultados mientras que la selección aleatoria genera los peores. Sin embargo, en la cobertura máxima es el aleatorio el que mejores resultados proporciona y el evolutivo el que peores tiene.

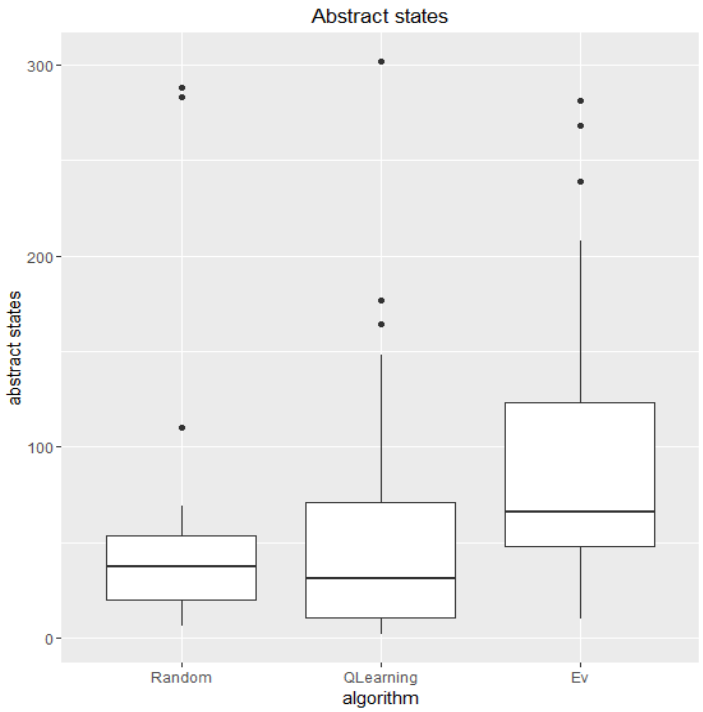

Figura 12: Power-Point - estados explorados

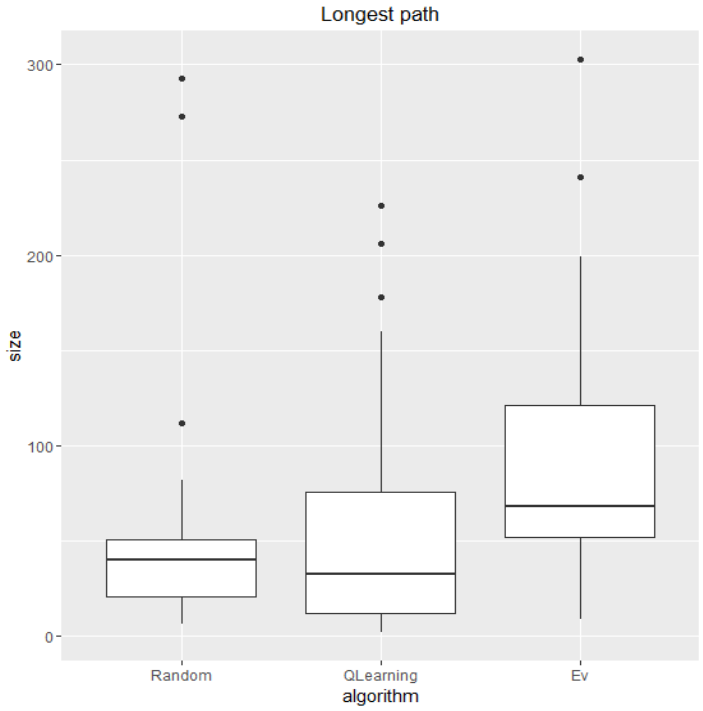

Figura 13: Power-Point - camino más largo

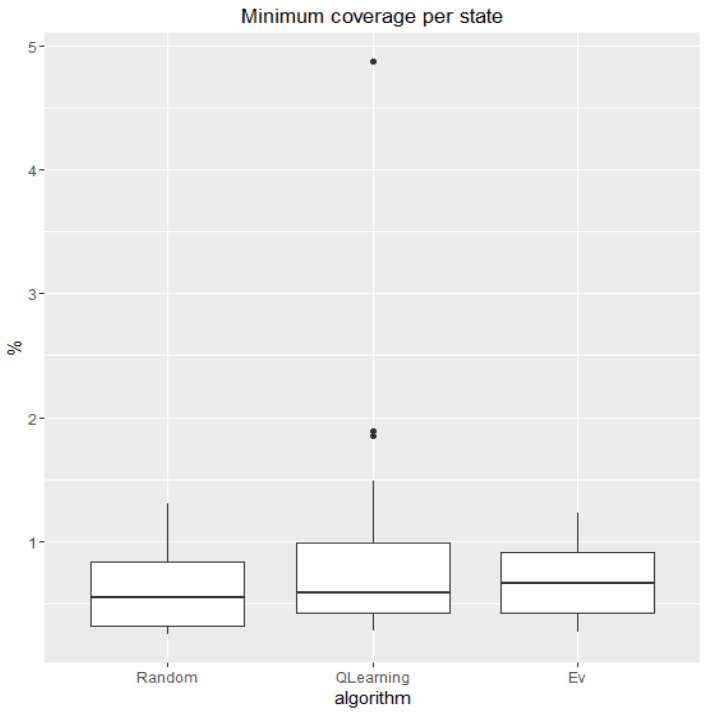

Figura 14: Power-Point - cobertura minima

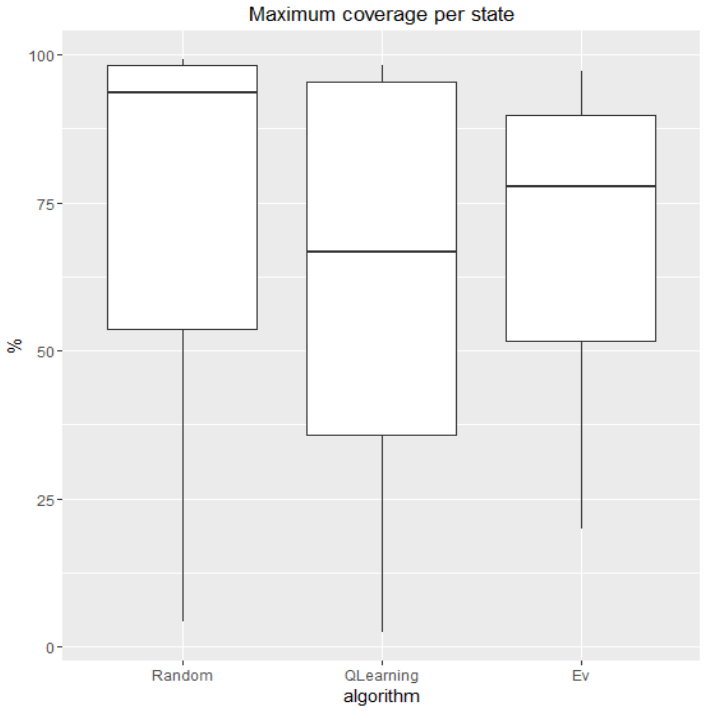

Figura 15: Power-Point - cobertura máxima

Con respecto a los resultados obtenidos en la herramienta Testona, se puede apreciar que la selección aleatoria tiene mejores resultados en la generación de estado distintos y en el camino más largo, mientras que en las dos coberturas ha obtenido mejores resultados el algoritmo evolutivo.

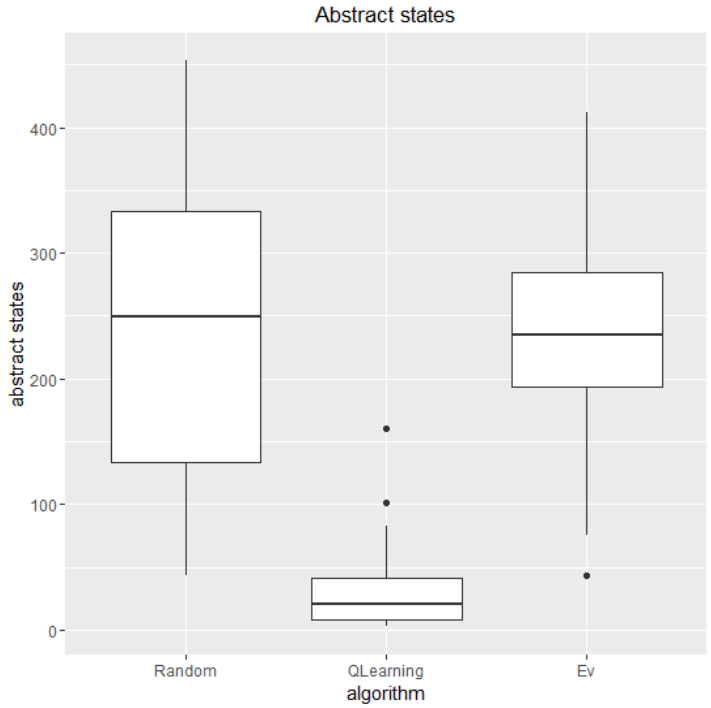

Figura 16: Testona- estados explorados

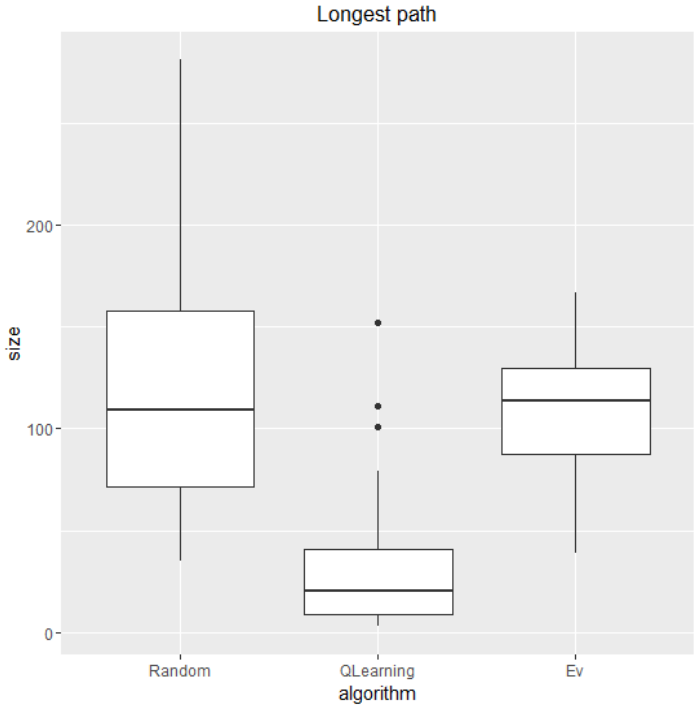

Figura 17: Testona - Camino más largo

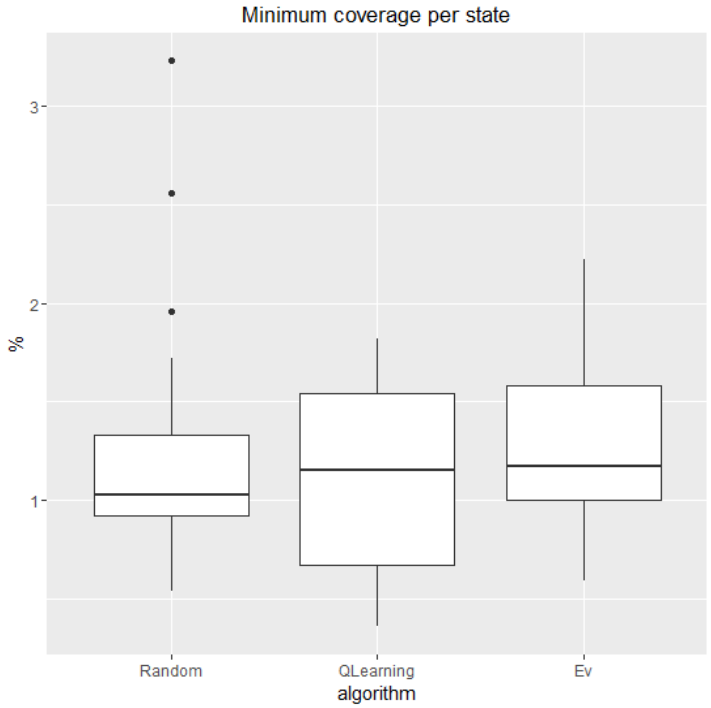

Figura 18: Testona - cobertura mínima

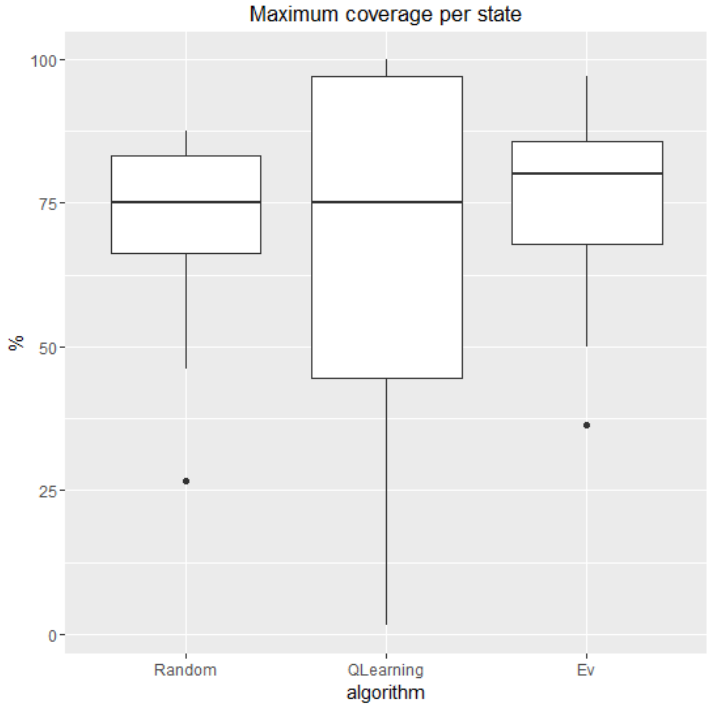

Figura 19: Testona - cobertura máxima

Con respecto a los resultados obtenidos en Odoo, se puede ver que el algoritmo de programación genética ha obtenido mejores resultados en todas las métricas, con la excepción de la cobertura mínima, donde ha generado mejores resultados la selección aleatoria. En el caso del algoritmo Q-learning ha obtenido en todos los casos una posición intermedia entre ambos.

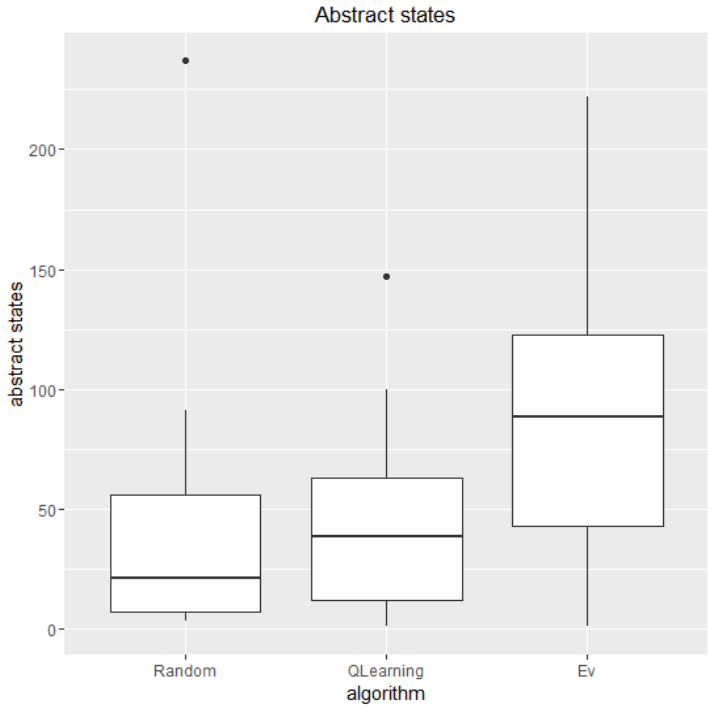

Figura 20: Odoo - estados explorados

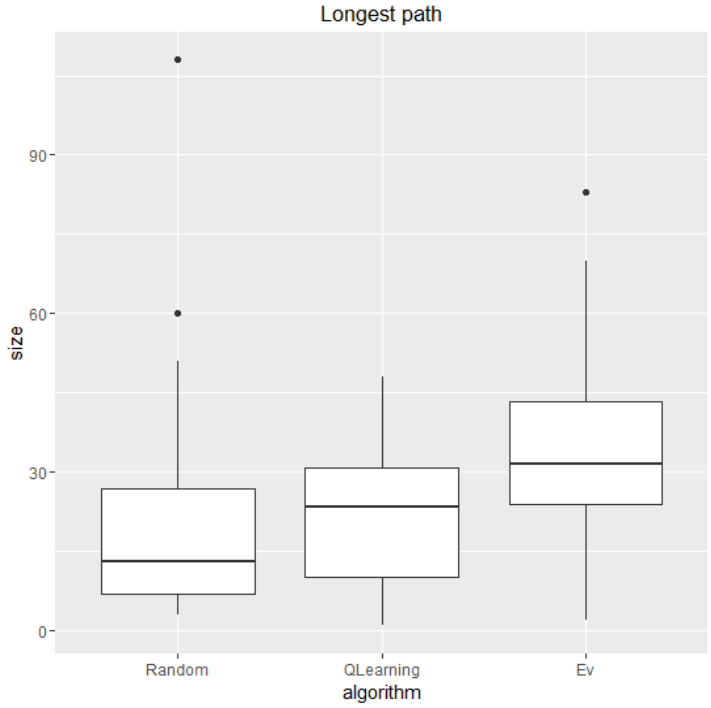

Figura 21: Odoo – camino más largo

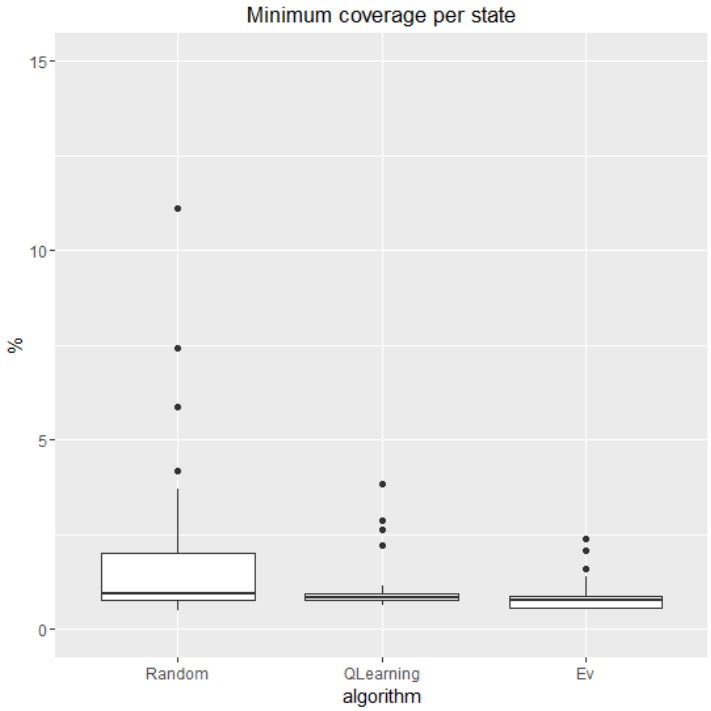

Figura 22: Odoo - Cobertura minima

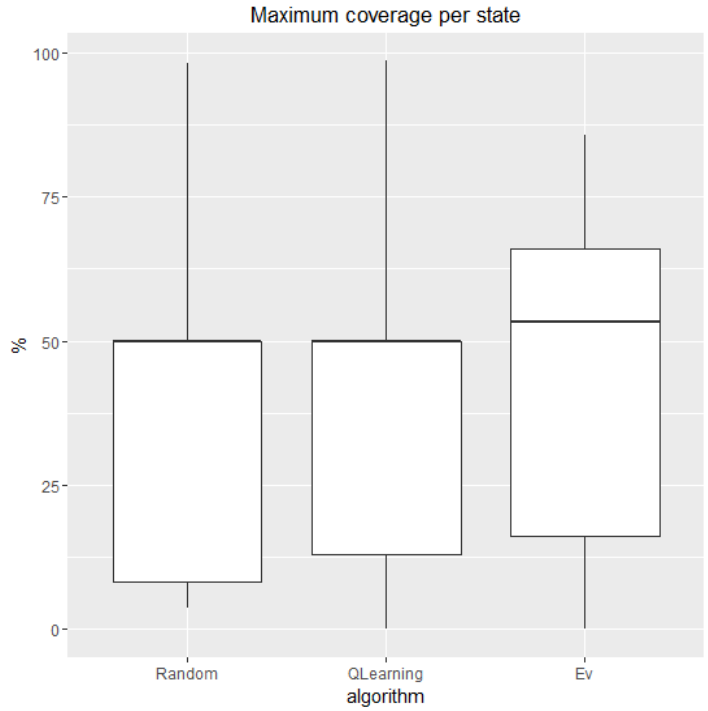

Figura 23: Odoo - cobertura máxima

Como se ha presentado en las métricas podemos ver como, por norma general, el algoritmo evolutivo ha generado mejores resultados que el resto de opciones, mientras que Q-learning por norma general ha mejorado los resultados de la selección aleatoria, con la excepción de Testona, donde la selección aleatoria ha generado buenos resultados.

Con estos resultados queda patente que no hay una solución perfecta que funcione en todas las aplicaciones por igual. Por esta razón los dos algoritmos empleados destinan una parte de su tiempo de ejecución al aprendizaje y a la mejora continua.

Cabe destacar que el coste temporal del algoritmo evolutivo ha sido superior al resto de métodos de selección, causando que se necesitase tiempo adicional para terminar las pruebas.

Además de las métricas propuestas, se han analizado los errores encontrados, que se recogen en la Tabla 7: Errores encontrados, clasificados en tres tipos. En primer lugar los falsos positivos, que tienen lugar cuando TESTAR identifica como error algo que realmente no lo es; por ejemplo, encontrar la palabra "error" en un cuadro de texto no significa que necesariamente haya ocurrido uno, simplemente puede ser parte de la aplicación. El segundo tipo de error es por congelamiento y se produce cuando una acción ha bloqueado la aplicación y ésta tarda demasiado en responder. Por último, el error real, es un error que afecta al funcionamiento de la aplicación.

Tabla 7: Errores encontrados

Herramienta	Tipo de selección	Errores
Power-Point	Aleatorio	1 x Congelamiento 2 x Falso positivo
	Q-learning	1 x Congelamiento 5 x Falso positivo
	Evolutivo	5 x Falso positivo 1 x Error real
Testona	Aleatorio	3 x Congelamiento 6 x Falso positivo
	Q-learning	3 x Falso positivo 1 x Congelamiento 1 x Error real
	Evolutivo	3 x Falso positivo 2 x Congelamiento 2 x Error real
Odoo	Aleatorio	4 x Falso positivo
	Q-learning	6 x Falso positivo 1 x Congelamiento 1 x Error real
	Evolutivo	4 x Error real 2 x Falso positivo

En la aplicación Power-Point el fallo más común ha sido confundir las respuestas del corrector ortográfico por errores reales. Sin embargo, tanto Q-learning como la selección aleatoria han conseguido bloquear la aplicación durante más de 10 segundos al intentar pasar a pdf una presentación grande con imágenes. Por otro lado, durante una de las secuencias del algoritmo evolutivo, la aplicación dejo de funcionar al intentar pasar al modo presentación repetidas veces con una presentación grande. Se ha intentado reproducir este error pero no se ha conseguido que vuelva a hacerlo.

En Testona se han encontrado errores derivados del acceso a base de datos, aplicación tiene una opción de acceder a través de un servidor a pruebas, sin embargo, al no disponer de red las máquinas esta operación fallaba. Además, se ha encontrado un

error en el ha fallado al recuperar el banco de trabajo, por un fallo de comunicación entre las interfaces (solo ha ocurrido en el evolutivo). Otro error localizado es que si no se dispone de internet y se accede a la ayuda online, la aplicación falla y se cierra inesperadamente (este error ha pasado en Q-learning y en el algoritmo evolutivo).

Con Odoo el número de falsos positivos ha sido aún mayor; esto se ha debido a que por velocidad de procesamiento del servidor, algunas operaciones han tardado más en terminar y TESTAR ha identificado esto como errores. Sin embargo, durante las ejecuciones del algoritmo Q-learning, tras la expiración de la sesión y su posterior reanudación, una de las funciones ha dejado de estar disponible. Este error no ocurre siempre que se realizan los mismos pasos.

10. Conclusiones

Como se ha visto en este trabajo las herramientas de testeo automatizado tienen cabida en la actualidad, sobre todo porque el testeo de aplicaciones es costoso pero necesario. Se han presentado alternativas al testeo manual y se ha visto el impacto que tienen en ellas las técnicas de búsqueda.

Unos buenos procesos de testeo pueden ayudar a las empresas a mejorar la calidad de su software y para conseguirlo, se puede emplear herramientas de testeo automatizado.

Las contribuciones principales de este trabajo son dos:

Por una parte, se ha introducido cuatro métricas para evaluación de la calidad de una herramienta de testeo automatizado. Estas métricas han sido utilizadas para guiar la selección de acciones, en sustitución del simple recuento de errores, que no siempre es viable o relevante. De estas cuatro métricas, dos han mostrado no ser relevantes, dado que los resultados generados no presentan, por lo general, diferencias significativas entre los distintos métodos de selección.

Por otra parte, se ha introducido un nuevo método de selección de acciones en la herramienta de testeo automatizado TESTAR, basado en programación genética. Este método se ha comparado con otro también basado en búsqueda, Q-learning, y con la selección aleatoria. El análisis estadístico llevado a cabo muestra que ambos métodos basados en búsqueda obtienen mejores resultados.

Las dos victorias de la selección aleatoria han sido debidas principalmente a que Testona posee dos pantallas con una gran cantidad de acciones distintas, lo que ha provocado que, con el fin de explorar, Q-learning y el algoritmo evolutivo se hayan centrado en ejecutar muchas de las acciones que hay en esas pantallas.

El algoritmo que más estados ha explorado es el más errores ha encontrado, aunque sería necesario realizar más pruebas para establecer una relación causal entre ambos fenómenos.

El hecho de que el mejor individuo generado sea simple nos una buena idea del potencial del enfoque evolutivo, pues incluso con una solución sencilla se han obtenido mejores resultados que con los otros métodos de selección. Esto se debe a la capacidad de generalización de este tipo de algoritmos.

Por lo que respecta a los errores encontrados, éstos se han dado en todas las aplicaciones, aunque en su mayoría no son reproducibles, dado que no se producen sistemáticamente. No obstante, en el caso concreto de en la herramienta Testona sí se ha podido encontrar errores que se pueden reproducir. Adicionalmente se ha obtenido información útil por lo que respecta a la aplicación práctica de TESTAR. Por ejemplo, se ha observado que TESTAR es bastante sencilla de utilizar en aplicaciones de escritorio, sin embargo, se vuelve más compleja a la hora de testear las aplicaciones web, porque necesita tener mucha precaución con los enlaces externos, el retraso que pueda tener la red o incluso las dificultades que tiene el API de accesibilidad para detectar elementos.

Así pues, podemos concluir que los objetivos de este trabajo se han cumplido y se ha probado que aplicar técnicas más “inteligentes” que la selección aleatoria, contribuye a mejorar los resultados de las herramientas de testeo automatizado. Además, aunque no haya quedado totalmente probado, se han encontrado métricas sustitutivas al conteo de errores para evaluar los resultados.

11. Trabajo futuro

Cómo hemos mencionado en la introducción, los resultados de este trabajo son los primeros avances para hacer de TESTAR una herramienta de testeo que no necesita saber cómo testear, pero es capaz de aprenderlo por sí mismo. Por consiguiente, a raíz de este trabajo se han planteado una serie de líneas de investigación futuras.

Por una parte, como ya se ha mencionado anteriormente, sería necesario evolucionar soluciones adaptadas a cada aplicación, es decir, ejecutar la etapa evolutiva para las otras dos herramientas y evaluar la mejora obtenida

Asimismo, sería interesante integrar todas las métricas definidas en una búsqueda multiobjetivo, por varias métricas y comprobar si se producen mejoras. Así como ejecutar más pruebas para poder confirmar si realmente son representativas y pueden sustituir el conteo de errores.

En este trabajo se han probado una parametrización del algoritmo evolutivo, por lo que queda pendiente comprobar si otras combinaciones de valores generan mejores resultados o no, por ejemplo en lugar de generar un individuo que sustituye al peor existente, implementar relevos generacionales y que se cambien todos los individuos actuales por nuevos.

Un paso más allá, y siguiendo desarrollos llevados a cabo en robótica evolutiva, consistiría en desarrollar mecanismos de selección de acciones que no estén guiados por una métrica, sino que exploren basándose en la novedad o la curiosidad (*novelty and curiosity search*).

Como trabajo futuro también se podría expandir el proceso evolutivo para que esté activo a la vez que se están ejecutando las pruebas. De esta forma, el proceso evolutivo sigue teniendo lugar permanente, pero cada vez que se encuentra un individuo mejor que el que está siendo utilizado por TESTAR se envía a la herramienta de testeo tal y como se puede ver en la Figura 24: Proceso evolutivo completo en TESTAR.

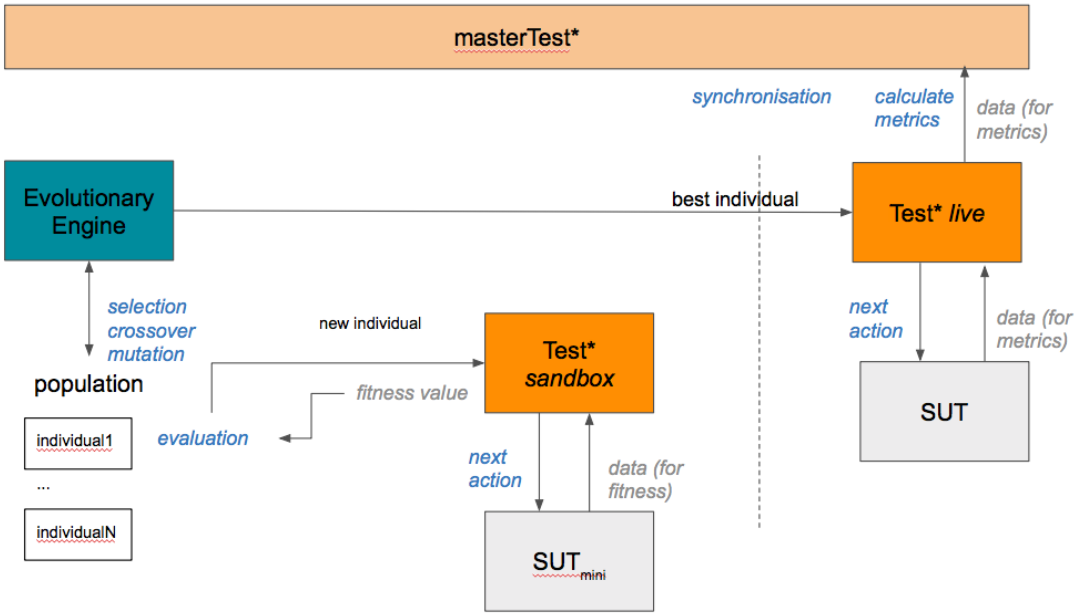

Figura 24: Proceso evolutivo completo en TESTAR

Por último, sería conveniente ejecutar pruebas sobre otras herramientas no incluidas en este trabajo.

Bibliography

[1.] S. Bauersfeld, A. de Rojas, and T. E. J. Vos, “Evaluating rogue user testing in industry: an experience report”, Proceedings of 8th International Conference RCIS. IEEE, 2014.

[2.] S. Bauersfeld, T.E.J. Vos, N. Condori-Fernández, A. Bagnato and E. Brosse,“Evaluating the TESTAR tool in an Industrial Case Study”. Proceedings of the 8th ACM EEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2014, Industrial Track, Torino, 2014.

[3.] Francisco Almenar, Anna I. Esparcia-Alcázar, Mirella Martínez, and Urko Rueda , “Automated testing of web applications with TESTAR. Lessons learned testing the Odoo tool”. SSBSE challenge 2016. Proceedings of the annual symposium on to Search Based Software Engineering (SSBSE), Raleigh, North Carolina, USA, 2016.

[4.] C. Kaner, “Exploratory Testing”, Florida Institute of Technology, *Quality Assurance Institute Worldwile Annual Software Testing Conference,* 2006.

[5.] E. Dustin, J. Rashka, J. Paul, “Automated software testing: introduction, management, and performance”, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.

[6.] E. Figueiredo Collins and V. Ferreira de Lucena, "Software test automation practices in agile development environment: an industry experience report", *Proceeding AST '12 Proceedings of the 7th International Workshop on Automation of Software Test*, pp. 57-63, 2016.

[7.] A. Memon, "A comprehensive framework for testing graphical user interfaces", Doctoral Dissertation, University of Pittsburgh, 2001.

[8.] E. Kit and S. Finzi, *Software testing in the real world*. New York, N.Y.: ACM Press, 1995.

[9.] Jolt Awards 2014: The Best Testing Tools". Dr.Dobbs.com.

[10.] "List of GUI testing tools", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/List_of_GUI_testing_tools. [Accessed: 09- Ago- 2016]..

[11.] EU project no: 257574 FP7 Call 8 ICT-Objective 1.2 Service Architectures and Infrastructures

[12.] "Windows Automation API (Windows)", *Msdn.microsoft.com*, 2016. [Online]. Available:https://msdn.microsoft.com/en-us/library/windows/desktop/ff486375(v=vs.85).aspx. [Accessed: Feb- 2016].

[13.] S. Bauersfeld and T.E.J. Vos, "A reinforcement learning approach to automated gui robustness testing", *4th Symposium on Search Based-Software Engineering,* vol. 7, 2012.

[14.] W. Lewis, D. Dobbs and G. Veerapillai, *Software testing and continuous quality improvement*. Boca Raton, Fla.: CRC Press, 2009.

[15.] H. Choi and H. Chae, "[5.] A Comparison of the Search Based Testing Algorithm with Metrics", *Journal of KIISE*, vol. 43, no. 4, pp. 480-488, 2016.

[16.] T. Grossman, G. Fitzmaurice, and R. Attar. “A survey of software learnability: Metrics, methodologies and guidelines”. SIGCHI Conference on Human Factors in Computing Systems, pages 649–658. ACM, 2009.

[17.] M. Harman and B. F. Jones, “Search-based software engineering, Information & Software Technology”, Vol. 43, No. 14, pp. 833-839 (2001)

[18.] F. Glover and M. Laguna, *Tabu search*. Boston: Kluwer Academic Publishers, 1997.
[19.] P.J.M. van Laarhoven and E. H. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht: Springer Netherlands, 1987.
[20.] A. García, "Técnicas metaheurísticas", [Online]. Available: http://www.iol.etsii.upm.es/arch/metaheuristicas.pdf. [Accessed: Mar- 2016].
[21.] S. Bauersfeld and T. Vos, "A reinforcement learning approach to automated gui robustness testing," in In Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE 2012). IEEE, 2012, pp. 7–12.
[22.] Anna I. Esparcia-Alcázar, F. Almenar, M. Martínez, U.Rueda, and Tanja E.J. Vos, "Q-learning strategies for action selection in the TESTAR automated testing tool", Proceedings of 6th International Conference on Metaheuristics and Nature Inspired Computing META'16, Marrakech, Morocco, Oct 27-31, 2016.
[23.] Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press. ISBN 0-262-11170-5.
[24.] "JSON", Json.org, 2016. [Online]. Available: http://www.json.org/. [Accessed: Ago- 2016
[25.] "Pony GP", csail.mit.edu, 2016. [Online]. Available:http://groups.csail.mit.edu/EVO-DesignOpt/PonyGP/out/index.html [Accessed: Ago- 2016]
[26.] "jar". Oracle.org, 2016. [Online]. Available: http://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html [Accessed: Feb - 2016]
[27.] "CMD", Microsoft.com 2016. [Online]. Available:https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/cmd.mspx?mfr=true. [Accessed: Feb-2016]
[28.] "Homepage", Odoo S.A., 2016. [Online]. Available: https://www.odoo.com/es_ES/. [Accessed: May- 2016].
[29.] "Installing Odoo — odoo 8.0 documentation", Odoo.com, 2016. [Online]. Available: https://www.odoo.com/documentation/8.0/setup/install.html. [Accessed: May- 2016]
[30.] Fay, Michael P.; Proschan, Michael A. (2010). "Wilcoxon–Mann–Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules". Statistics Surveys. 4: 1–39. doi:10.1214/09-SS051. MR 2595125. PMC 2857732free to read. PMID 20414472.
[31.] William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association 47 (260): 583–621, December 1952.
[32.] Glenford J. Myers, The Art of Software Testing, Wiley, 1979.
[33.] "Homepage" Selenium. [Online]. Available: http://www.seleniumhq.org/ [Accessed: Jul- 2016].
[34.] P. Aho, M. Suarez, T. Kanstren, and A.M. Memon, "Industrial adoption of automatically extracted GUI models for testing", Proc. Int. Workshop on Experiences and Empirical Studies in Software Modelling (EESSMod), 1 Oct 2013, Miami, Florida, USA, pp. 49-54.

Anexo I: Esqueleto del protocolo

En este y en los anexos II, III, IV, V y VI se va a presentar los protocolos empleados para configurar TESTAR, no obstante, debido a su tamaño se ha decidido mostrar solo las partes fundamentales que se diferencian entre versiones.

En este anexo se presenta un esqueleto con las componente que van a diferenciarse del resto, sin embargo, no incluirá líneas de código.

```
/*Variables*/

/*Browser special methods*/

/*
        * This method is called when the Rogue User requests the state of the SUT.
        * Here you can add additional information to the SUT's state or write your
        * own state fetching routine. The state should have attached an oracle
        * (TagName: <code>Tags.OracleVerdict</code>) which describes whether the
        * state is erroneous and if so why.
        * @return  the current state of the SUT with attached oracle.
 */
protected State getState(SUT system) throws StateBuildException{}

/*
        * This method is used by the Rogue User to determine the set of currently available actions.
        * You can use the SUT's current state, analyze the widgets and their properties to create
        * a set of sensible actions, such as: "Click every Button which is enabled" etc.
        * The return value is supposed to be non-null. If the returned set is empty, the Rogue User
        * will stop generation of the current action and continue with the next one.
        * @param system the SUT
        * @param state the SUT's current state
        * @return  a set of actions
        */
```

```
protected Set<Action> deriveActions(SUT system, State state) throws
ActionBuildException{}

/*
        *Derives left click actions
*/
private boolean isClickable(Widget w){}

/*
        *Derives Type Into actions
*/
private boolean isTypeable(Widget w){}
```

Anexo II: Protocolo escritorio por defecto

```
/*Variables*/

Empty

/*Browser special methods*/

Empty

protected Set<Action> deriveActions(SUT system, State state) throws
ActionBuildException{
		Set<Action> actions = super.deriveActions(system,state);
		StdActionCompiler ac = new AnnotatingActionCompiler();
		//---------------------
		// BUILD CUSTOM ACTIONS
		//---------------------

		// iterate through all widgets
		for(Widget w : state){
			if(w.get(Enabled, true) && !w.get(Blocked, false)){
				if (!blackListed(w)){
					// left clicks
					if(whiteListed(w) || isClickable(w))
						actions.add(ac.leftClickAt(w));
					// type into text boxes
					if(isTypeable(w))
						actions.add(ac.clickTypeInto(w,
this.getRandomText(w)));
				}
			}
		}
```

```
            if(actions.isEmpty())
                actions.add(ac.hitKey(KBKeys.VK_ESCAPE));

            return actions;
        }
    private boolean isClickable(Widget w){
            Role role = w.get(Tags.Role, Roles.Widget);
            if(!Role.isOneOf(role, NativeLinker.getNativeUnclickable())){
                String title = w.get(Title, "");
                if(!title.matches(settings().get(ClickFilter))){
                    if(Util.hitTest(w, 0.5, 0.5))    return true;
                }
            }
            return false;
        }
        private boolean isTypeable(Widget w){
            Role role = w.get(Tags.Role, Roles.Widget);
            if(Role.isOneOf(role, NativeLinker.getNativeRoles("UIAEdit"))){
                if(Util.hitTest(w, 0.5, 0.5))    return true;
            }
            return false;
        }
```

Anexo III: Protocolo web por defecto

```
/*variables*/

static double browser_toolbar_filter;

static Role webController;

static Role webText;

private static boolean firefox; // we expect Mozilla Firefox or Microsoft Internet Explorer (for more browsers code must be changed!)

/*Browser special methods*/

// check whether we use Internet Explorer or Firefox (other browser support must be coded)

private void initBrowser(){
	String sutPath = settings().get(ConfigTags.Executable);
	firefox = !sutPath.contains("iexplore.exe");
	if(firefox)
		setFilterToFirefox();
	else
		setFilterToExplorer();
}

private void setFilterToExplorer(){
	webController = NativeLinker.getNativeRole("UIADataItem");
	webText = NativeLinker.getNativeRole("UIAText");
}

private void setFilterToFirefox(){
	webController = NativeLinker.getNativeRole("UIACustomControl");
	webText = NativeLinker.getNativeRole("UIAEdit");
}
```

```
protected Set<Action> deriveActions(SUT system, State state) throws
ActionBuildException{
        Set<Action> actions = super.deriveActions(system,state);
        StdActionCompiler ac = new AnnotatingActionCompiler();
                //----------------------
                // BUILD CUSTOM ACTIONS
                //----------------------
through all widgets
for(Widget w : state){
        if(w.get(Enabled, true) && !w.get(Blocked, false)){
                if (!blackListed(w)){
                        if(whiteListed(w) || isClickable(w))
                                actions.add(ac.leftClickAt(w));
                if(isTypeable(w)){
                                actions.add(ac.clickTypeInto(w,
this.getRandomText(w)));
                }
        }
}
}
        if(actions.isEmpty())
                actions.add(ac.hitKey(KBKeys.VK_ESCAPE));
        return actions;
        }

        private boolean isClickable(Widget w){
      Role role = w.get(Tags.Role, Roles.Widget);
      if(!Role.isOneOf(role, NativeLinker.getNativeUnclickable())){
        String title = w.get(Title, "");
        String clickFilter = settings().get(ClickFilter);
```

```
        if(!title.matches(clickFilter) || clickFilter.equals("")){
          Shape shape = w.get(Tags.Shape,null);
          double hit = 0.5;
          if(Util.hitTest(w, hit, hit)){
            if (shape != null && shape.y() > browser_toolbar_filter)
                return true;
          }
        }
      return false;
        }

private boolean isTypeable(Widget w){
        Role r = w.get(Tags.Role, null);
        Boolean b = NativeLinker.isNativeTypeable(w);
        if (b != null && b.booleanValue() &&
          r != null && Role.isOneOf(r, NativeLinker.getNativeRole("UIAEdit"))){
                if(Util.hitTest(w, 0.5, 0.5)){
                        Shape shape = w.get(Tags.Shape,null);
                        if (shape != null && shape.y() > browser_toolbar_filter)
                                return true;
                        }
                }
                return false;
        }
```

Anexo IV: Protocolo Power-Point

Las acciones peligrosas y las frases conocidas que generan falsos errores han sido introducidas en la lista negra.

Solo se van a mostrar los campos que difieren del protocolo básico de escritorio.

```
protected Set<Action> deriveActions(SUT system, State state) throws
ActionBuildException{
        Set<Action> actions = super.deriveActions(system,state);
        StdActionCompiler ac = new AnnotatingActionCompiler();
        String titleRegEx = settings().get(SuspiciousTitles);
        // search all widgets for suspicious titles
        for(Widget w : state){
                String title = w.get(Title, "");
                if(title.matches(titleRegEx) && (!title.contains("Sin errores") &&
!title.contains("Erorres"))){
                Visualizer visualizer = Util.NullVisualizer;
                // visualize the problematic widget, by marking it with a red box
                if(w.get(Tags.Shape, null) != null){
                        Pen redPen =
Pen.newPen().setColor(Color.Red).setFillPattern(FillPattern.None).setStrokePattern(S
trokePattern.Solid).build();
                        visualizer = new ShapeVisualizer(redPen, w.get(Tags.Shape),
"Suspicious Title", 0.5, 0.5);
                }
        for(Widget w : state){
                if(w.get(Enabled, true) && !w.get(Blocked, false)){
                        if (!blackListed(w)){
                                // left clicks
                                if(whiteListed(w) || isClickable(w))
                                        actions.add(ac.leftClickAt(w));
                                // type into text boxes
```

```
                        if(isTypeable(w))
                            actions.add(ac.clickTypeInto(w,
this.getRandomText(w)));
                    }
                }
            }
            if(actions.isEmpty())
                actions.add(ac.hitKey(KBKeys.VK_ESCAPE));
            return actions;
            }

            // should the widget be clickable?
            private boolean isClickable(Widget w){
                Role role = w.get(Tags.Role, Roles.Widget);
                if(!Role.isOneOf(role, NativeLinker.getNativeUnclickable())){
                    String title = w.get(Title, "");
                    if(!title.matches(settings().get(ClickFilter))){

                        if(Util.hitTest(w, 0.5, 0.5))
                            return true;
                    }
                }
                return false;
            }
}
```

Anexo V: Protocolo Testona

Las acciones peligrosas y las frases conocidas que generan falsos errores han sido introducidas en la lista negra.

Solo se van a mostrar los campos que difieren del protocolo básico de escritorio.

```
// should the widget be clickable?

private boolean isClickable(Widget w){

        Role role = w.get(Tags.Role, Roles.Widget);

        if(!Role.isOneOf(role, NativeLinker.getNativeUnclickable())){

                String title = w.get(Title, "");

                if(!title.matches(settings().get(ClickFilter))){
                        if(Util.hitTest(w, 0.5, 0.5))

                                return true;

                }

        }

        return false;

}
```

Anexo VI: Protocolo Odoo

Las acciones peligrosas y las frases conocidas que generan falsos errores han sido introducidas en la lista negra.

Solo se van a mostrar los campos que difieren del protocolo básico web.

```
protected Set<Action> deriveActions(SUT system, State state) throws
ActionBuildException{
        Set<Action> actions = super.deriveActions(system,state);
        StdActionCompiler ac = new AnnotatingActionCompiler();
        String titleRegEx = settings().get(SuspiciousTitles);
        // search all widgets for suspicious titles
        for(Widget w : state){
                String title = w.get(Title, "");
                if(title.matches(titleRegEx) && (!title.contains("Procurement") &&
!title.contains("Odoo Client Error"))){
                        System.out.println(title);
                        Visualizer visualizer = Util.NullVisualizer;
                        if(w.get(Tags.Shape, null) != null){
                                Pen redPen =
Pen.newPen().setColor(Color.Red).setFillPattern(FillPattern.None).setStrokePattern(S
trokePattern.Solid).build();
                                visualizer = new ShapeVisualizer(redPen,
w.get(Tags.Shape), "Suspicious Title", 0.5, 0.5);
                        }
                }
        }
        for(Widget w:state){
                Role role = w.get(Tags.Role, Roles.Widget);
                if(Role.isOneOf(role, NativeLinker.getNativeRoles("UIAEdit"))){
                        String title = w.get(Title, "");
                        //check if we are in the login state, if so we log in the application
```

```
                    if (title.contains("Email") && w.parent().get(Title,
"").contains("Log in")){
                        new CompoundAction.Builder().add(new
                        Type("admin"),0.1)
                        .add(new KeyDown(KBKeys.VK_TAB),0.5)
                        .add(new KeyDown(KBKeys.VK_ENTER),0.5).build()
                        .run(system, null, 0.1);
                        Util.pause(settings().get(ConfigTags.StartupTime));
                    }
                }
            }
            //Close emergent windows
            boolean stop = false;
            for(Widget w : state){
                Role role = w.get(Tags.Role, Roles.Widget);
                if(Role.isOneOf(role, NativeLinker.getNativeRoles("UIAButton"))){
                    String title = w.get(Title, "");
                    if (title.contains("ancel") || title.contains("Ok")){
                        stop = true;
                        actions.add(ac.leftClickAt(w));
                    }
                }
            }
        if(!stop){
            for(Widget w : state){
            if(w.get(Enabled, true) && !w.get(Blocked, false)){
                if (!blackListed(w)){
                    if(whiteListed(w) || isClickable(w))
                        actions.add(ac.leftClickAt(w));
                    if(isTypeable(w)){
```

```
                                actions.add(ac.clickTypeInto(w,
this.getRandomText(w)));
                            }
                        }
                }
                if(actions.isEmpty())
                        actions.add(ac.hitKey(KBKeys.VK_ESCAPE));
                        return actions;
                }
}
private boolean isClickable(Widget w){
        Role role = w.get(Tags.Role, Roles.Widget);
        if(!Role.isOneOf(role, NativeLinker.getNativeUnclickable())){
                if(Role.isOneOf(role,NativeLinker.getNativeRole("UIAImage"))          &&
w.parent().get(Title, "").contains("dmin")){
                        return false;
                }
                String title = w.get(Title, "");
                if(w.childCount()>0 && w.child(0).get(Title, "").contains("dmin")){
                        return false;
                }
                String clickFilter = settings().get(ClickFilter);
                if(!title.matches(clickFilter) || clickFilter.equals("")){
                   Shape shape = w.get(Tags.Shape,null);
                   double hit = 0.5;
                   if(Util.hitTest(w, hit, hit)){
                      if (shape != null && shape.y() > browser_toolbar_filter)
                        return true;
                   }
                }
```

```
        }
        return false;
}
```