UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SOFTWARE

Arquitectura cloud basada en la técnica fuzzing para la generación de escenarios de pruebas en el sector de desarrollo de software

TESIS

Para optar el título profesional de Ingeniero de Software

AUTOR(ES)

Gamarra Tafur, Alexander Giovanni (0000-0001-7858-9677)

Ramirez Espinoza, Gianina Andrea (0000-0003-1911-1838)

ASESOR

Barrientos Padilla, Alfredo (0000-0002-0029-4913)

Lima, 09 de febrero del 2023

DEDICATORIA

La presente investigación la dedicamos a nuestra familia, por todo el amor y cariño durante nuestra etapa universitaria.

AGRADECIMIENTOS

Primeramente, damos gracias a Dios por permitirnos culminar la primera etapa de nuestra vida universitaria y por encaminar nuestra etapa profesional. A la universidad, por la formación académica y su constante exigencia. Y finalmente a los docentes que estuvieron involucrados en el desarrollo de la presente investigación, muchas gracias por todo su apoyo emocional y académico.

RESUMEN

El desarrollo de software está en constante evolución, cada vez son más las nuevas tecnologías que se integran al ciclo de vida del desarrollo software y una de las fases más importantes es el aseguramiento de la calidad. A medida que el sistema va creciendo, son más las casuísticas que se generan y con ello la disminución del % de la cobertura de pruebas. Uno de los problemas en el desarrollo del software es que existen errores de programación no identificados con los métodos de pruebas convencionales durante la fase de desarrollo. Es por esta razón, que en el siguiente documento proponemos la implementación de una **arquitectura cloud** basada en la técnica fuzzing (**AcFGTC**) para la generación de escenarios de pruebas, con el fin de prevenir errores en el sistema.

Palabras clave**:** Aprendizaje automático, Computación en la nube, Pruebas fuzzing, Red neuronal

ABSTRACT

Software development is constantly evolving, new technologies are being integrated into the software development life cycle and one of the most important phases is the quality assurance. As the system grows, more cases are generated, and the percentage of test coverage decreases. One of the major issues in the software development is there are bugs not identified with conventional testing methods during the testing phase. For this reason, in the following document we propose the implementation of a cloud architecture based on the fuzzing technique (AcFGTC) for the generation of test scenarios. The objective here is prevent errors in the system.

Keywords: Cloud Computing, Fuzz Testing, Neuronal Networks, Machine Learning

N°5687_Arquitectura cloud basada en la técnica fuzzing para la generación de escenarios de pruebas en el sector de desarrollo de software

INFORME DE ORIGINALIDAD

12%	12%	4%	3%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	upc.aws.openrepository.com Fuente de Internet	4%
2	repositorioacademico.upc.edu.pe Fuente de Internet	3%
3	Submitted to Bocconi University Trabajo del estudiante	1%
4	export.arxiv.org Fuente de Internet	<1%
5	repositorio.ucv.edu.pe Fuente de Internet	<1%
6	www.techscience.com Fuente de Internet	<1%
7	Submitted to University of Maryland, Global Campus Trabajo del estudiante	<1%
8	codechina.csdn.net Fuente de Internet	<1%

9	ciberseguridad.com Fuente de Internet	<1 %
10	documentop.com Fuente de Internet	<1 %
11	doaj.org Fuente de Internet	<1 %
12	tesis.ucsm.edu.pe Fuente de Internet	<1 %
13	link.springer.com Fuente de Internet	<1 %
14	Submitted to Consorcio CIXUG Trabajo del estudiante	<1 %
15	Gerardo Quintana, Martin Solari. "A systematic mapping study on experiments with automatic structural test case generation", 2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI), 2012 Publicación	<1 %
16	cybersecurity.springeropen.com Fuente de Internet	<1 %
17	www.dropbox.com Fuente de Internet	<1 %
18	ouci.dntb.gov.ua Fuente de Internet	<1 %
	crestweb.cs.ucl.ac.uk	

19	Fuente de Internet	<1 %
20	aws.amazon.com Fuente de Internet	<1 %
21	dergipark.org.tr Fuente de Internet	<1 %
22	www.informatica.us.es Fuente de Internet	<1 %
23	Morteza Zakeri Nasrabadi, Saeed Parsa, Akram Kalaee. "Format-aware learn&fuzz: deep test data generation for efficient fuzzing", Neural Computing and Applications, 2020 Publicación	<1 %
24	aaai.org Fuente de Internet	<1 %
25	topsecretapiaccess.dovepress.com Fuente de Internet	<1 %
26	www.cacic2016.unsl.edu.ar Fuente de Internet	<1 %
27	Submitted to Universidad Peruana de Ciencias Aplicadas Trabajo del estudiante	<1 %
28	pure.tudelft.nl Fuente de Internet	<1 %

29	www.limnetica.net Fuente de Internet	<1 %

Excluir citas	Apagado	Excluir coincidencias	< 20 words
Excluir bibliografía	Activo		

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El presente trabajo de investigación tiene como objetivo principal el desarrollo de una arquitectura en capas basado en la técnica fuzzing para la generación de escenarios de pruebas en el sector de desarrollo de software.

La importancia de estudiar este tema en particular radica en que las consecuencias que se generan por la detección de un bug en producción pueden ser de alto impacto y generar pérdidas económicas en una organización.

En el capítulo 1, se abordan cuestiones teóricas que sustentan el planeamiento del proyecto, así como también, los antecedentes y la definición del problema. Por un lado, se específica el objetivo general y los objetivos específicos del proyecto. Por otro lado, se establece la planificación del proyecto como la gestión del alcance, tiempo, riesgos, etc.

En el capítulo 2, se describe el cumplimiento de cada student outcome.

En el capítulo 3, se centra en el desarrollo del marco teórico de la investigación. Se define un análisis de las tecnologías y métodos utilizados en el proyecto.

En el primer apartado del capítulo 4, se establece la metodología de trabajo para la identificación de artículos científicos, así como también, el desarrollo de las preguntas de investigación. En el segundo apartado del capítulo 4, se desarrolla el resumen de 20 artículos de investigación de acuerdo con las topologías identificadas en el paso anterior. Finalmente, se realizan las conclusiones por cada topología, y así como también, la conclusión general del estado de arte.

En el capítulo 5, se realiza el desarrollo del proyecto. Se detallan la arquitectura en capas planteada, así como también, el proceso del desarrollo de software. Asimismo, se especifican los procesos identificados y la especificación de los requisitos del sistema.

Finalmente, en el capítulo 6, se aborda la validación de factibilidad económica del proyecto, se analizan el costo por cada recurso humano y técnico que necesitará el proyecto; así como también, la encuesta de satisfacción a los expertos de la arquitectura propuesta.

CAPÍTULO 1. DESCRIPCIÓN DEL PROYECTO

En el siguiente capítulo se desarrollará el objetivo general del proyecto, objetivos específicos y los indicadores de éxito para cada uno de ellos.

1.1 Fundamentación

1.1.1 Antecedentes

Cada vez son más los equipos que utilizan la integración continua para acelerar los entregables de un producto de software, este enfoque de DevOps también incluye en su tabla matriz a los testers, quienes integran sus pruebas automatizadas con cada despliegue que se realiza en el sistema. A medida que el sistema va creciendo, son más las casuísticas que se generan y con ello la disminución del % de cobertura de pruebas. El problema es que las pruebas automatizadas no están enfocadas en la búsqueda de nuevos bugs, sino en probar la funcionalidad existente. Es por esta razón, que la presente investigación tiene como objetivo desarrollar la siguiente problemática: Errores de programación no identificados con los métodos de pruebas convencionales

1.1.2 Dominio del Problema

Errores de programación no identificados con los métodos de pruebas convencionales. Esto se debe, principalmente a tres motivos:

- Dificultad para cubrir escenarios de pruebas que no están dentro de los criterios de aceptación.
- La generación y gestión de los casos de prueba es de forma manual.
- El sistema es un monolito (Nido de bugs)

1.1.3 Propuesta de Solución del Problema

Se ha propuesto una arquitectura cloud para la generación de escenarios de prueba basada en la técnica fuzzing.

1.1.4 Oportunidad de Negocio

El proyecto, al culminado, permitirá realizar pruebas a diferentes métodos REST de forma automatizada con una mayor cobertura de escenarios de prueba. Los principales beneficiados del proyecto serán los Stakeholders (usuarios y desarrolladores). Adicionalmente, esta arquitectura podría ser implementada a gran escala en pruebas a nivel de Front End, de tal manera que, permitirá incrementar la cobertura de pruebas en conjunto con el backend y lograr un software con menor tolerancia a fallos.

1.1.5 Herramientas tecnológicas para utilizar

Azure Machine Learning: Azure Machine Learning permite a los científicos y desarrolladores de datos crear, implementar y administrar modelos de alta calidad más rápido y con confianza. Acelera el tiempo de generación de valor con operaciones de aprendizaje automático (Machine Learning)

Microsoft Neural Network Algorithm: Nuevas tecnologías se van implementado día a día y uno de los recientes lanzamientos de Microsoft sobre recursos Cloud, fue la implementación de una arquitectura de red neuronal para el aprendizaje automático. El algoritmo comprende cada estado de los atributos de entrada frente a cada posible estado del atributo de predicción, y realiza el cálculo de las probabilidades de cada combinación en función de los datos de aprendizaje. El objetivo del uso de este algoritmo es predecir resultados basados en parámetros de entrada. Asimismo, este recurso se puede integrar a los recursos existentes como redes neuronales y otros algoritmos desarrollados en la misma plataforma. (Microsoft, s.f)

Postman: Aplicación que nos permite testear APIs a través de una interfaz gráfica de usuario.

ArchiMate: ArchiMate es un software utilizado para el modelado ("lenguaje") con el fin describir arquitecturas empresariales.

1.2 Objetivos del Proyecto

1.2.1 Objetivo General

Proponer la implementación de una arquitectura cloud basado en la técnica fuzzing para la generación de escenarios de pruebas, con el fin de prevenir errores en una aplicación de software.

1.2.2 Objetivos Específicos

- OE1: Analizar técnicas, herramientas y enfoques basados en fuzzing, machine learning y modelos generativos para realizar una revisión sistemática.
- OE2: Diseñar una arquitectura en capas para la integración de algoritmos fuzzing y redes neuronales.
- OE3: Validar la propuesta con el líder técnico de desarrollo y QA a través de un focus group.

1.2.3 Indicadores de Logro de Objetivos

Tabla 1

Indicadores de Éxito de los Objetivos Específicos

Objetivo	Indicadores
OE1	IE01: Obtener acta de aprobación otorgado por el cliente, validando la revisión sistemática realizada y tecnologías a usar.
OE2	IE02: Diseño de la arquitectura en capas basado en algoritmos fuzzing y redes neuronales
OE2	IE03: Nivel de satisfacción alto obtenido en la encuesta a los usuarios sobre el análisis de la arquitectura propuesta
OE3	IE04: Obtener una aprobación del líder técnico y líder de QA sobre los tipos de escenarios que se obtendrían con la propuesta.

1.3 Planificación del proyecto

1.3.1 Gestión del alcance

Se han considerado las siguientes viñetas referente al alcance:

- Análisis de las aplicaciones y tipos de algoritmos basados en fuzz testing en el sector tecnológico
- Análisis comparativo de las herramientas tecnológicas de machine learning, modelos generativos y fuzz testing determinantes para la arquitectura de capas del proyecto.
- Diseño de la arquitectura cloud basado en fuzzing para la generación de escenarios de pruebas en el sector de desarrollo de software
- Diseño del dashboard propuesto para la visualización de los escenarios generados, y así como también, los resultados de su ejecución.
- Documento con la validación de la arquitectura de capas por parte de expertos y de los responsables del proceso de desarrollo de software.

1.3.2 Gestión del tiempo

En esta fase se presenta la siguiente tabla:

Tabla 2

Gestión del Tiempo

Proyecto	Hito del proyecto	Fecha estimada	Entregables incluidos	Prioridad
Inicio	Presentación de la propuesta (Definición del problema, Objetivo General, etc.)	30/10/23	Documento con la revisión sistemática del estado del arte	ALTA
Planificación	Presentación del plan del proyecto	12/12/23	Plan del proyecto	ALTA
Diseño	Diseño de la arquitectura Cloud basada en fuzzing	21/01/23	Diseño de la arquitectura Cloud	ALTA
Validación	Revisión de expertos y StakeHolders	27/01/23	Arquitectura de capas de la solución Documento del plan del presupuesto contable	ALTA

1.3.3 Gestión de recursos humanos

Figura 1

Organigrama de la Propuesta de Proyecto

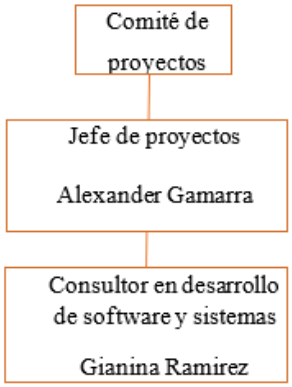


Tabla 3

Descripción de Roles y Responsabilidades

Rol	Responsabilidades
Comité de proyectos	Docentes encargados de la capacitación constante y revisión del proyecto de tesis
Jefe del proyecto	Aprobar las Historias de Usuario
	Absolver dudas sobre el detalle de los requerimientos de usuario
	Diseñar la propuesta del proyecto - arquitectura basado en cloud computing
	Investigar y aplicar de manera eficiente los recursos cloud en la arquitectura
	Establecer costos y presupuestos del uso de los recursos cloud
	Diseñar un proceso de desarrollo de software en el cual se incluya el proceso de pruebas automatizadas
Consultor en desarrollo de software y sistemas	En base a su experiencia, diseñar el uso de algoritmos fuzzing y de machine learning para su inclusión en la arquitectura de cloud computing
	Velar por la integridad del proceso de desarrollo de software

1.3.4 Gestión de las comunicaciones

La gestión de la comunicación tiene como objetivo organizar, coordinar y controlar la comunicación con todos los stakeholders del proyecto.

1.3.4.1 Comunicación del proyecto

La comunicación del proyecto se dará en reuniones a través de la aplicación teams, donde se revisará el avance del proyecto, así como también, las reuniones que involucran al proceso del desarrollo de software y la metodología ágil Sprint.

1.3.5 Gestión del riesgo

Tabla 4

Gestión del Riesgo

#	Riesgo	Probabilidad	Impacto	Estrategia de mitigación
1	La empresa no acepta el uso de la información solicitada por parte del gestor del proyecto	0.7	Alto	**Mitigar** Solicitar con anticipación la firma del documento con los permisos del uso de la información
2	La empresa no acepta la propuesta descrita debido a políticas de seguridad	0.4	Medio	**Mejorar** Realizar una investigación adecuada y utilizar buenas prácticas durante el desarrollo del diseño de la arquitectura
3	No abarcar todos los tipos de parámetros de entradas de los métodos a probar	0.6	Alto	**Mitigar** Solicitar la documentación del api a probar, a fin de mapear todos los parámetros que necesita el api
4	Cancelación de las reuniones con el cliente.	0.3	Media	**Aceptar Pasivamente** Se pactarán reuniones por internet o durante la semana para subsanar este riesgo.
5	Cambio del alcance del proyecto por necesidades del cliente	0.5	Alta	**Mitigar** Se realizará un ajuste del proyecto, considerando siempre el impacto en tiempo que este pueda ocasionar en el alcance actual.
6	El algoritmo por desarrollar genera escenarios de pruebas muy dispersos de la realidad	0.3	Media	**Mitigar** Alimentar la red neuronal con información relevante, que ayude al algoritmo principal a seleccionar escenarios de pruebas más cercanos a la realidad.

CAPÍTULO 2. LOGROS DE LOS STUDENT OUTCOMES

El presente capítulo describe la capacidad de desarrollar y llevar a cabo la experimentación adecuada de analizar e interpretar datos, usando juicio de ingeniería de software.

2.1 Student Outcome 1

2.1.1 Definición

La capacidad de identificar, formular y resolver problemas complejos de ingeniería aplicando los principios de ingeniería, ciencia y matemática.

2.1.2 Aplicación

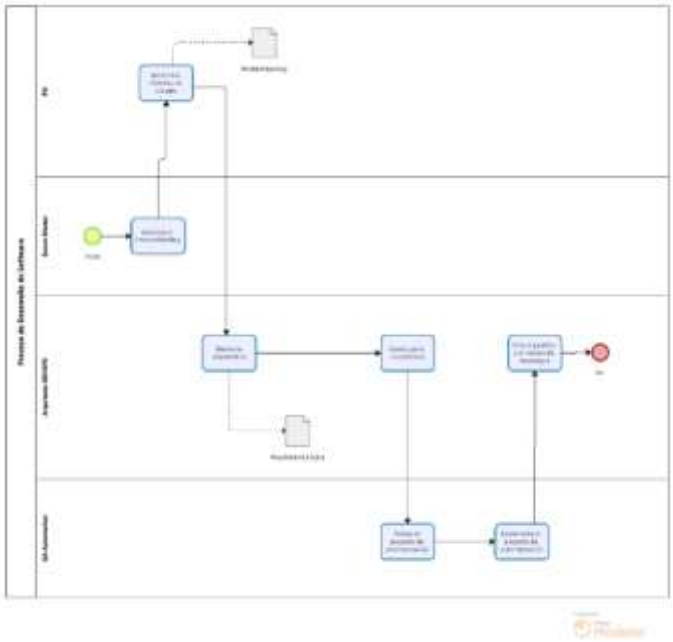
Nuestro proyecto contribuye con propuesta de una arquitectura, la cual busca Reducción de errores e incremento del % de la cobertura de pruebas del sistema. La complejidad de este proyecto radica en el entendimiento del funcionamiento del algoritmo de red neuronal para simplificar la data. El sustento científico de nuestro proyecto se basa en 20 aportes científicos relacionados a Machine Learning y Fuzzing, para entender estos conceptos revisar el capítulo 3, Marco Teórico, y el anexo E, Estado del Arte. El documento relacionado a este punto es:

- Proceso preliminar mediante BPM

Evidencia:

Figura 2

Subproceso: Proceso Preliminar mediante BPM



2.2 Student Outcome 2

2.2.1 Definición

Es la capacidad de aplicar el diseño de ingeniería para producir soluciones que satisfagan necesidades específicas con consideración de salud pública, seguridad y bienestar, así como factores globales, culturales, sociales, ambientales y económicos.

2.2.2 Aplicación

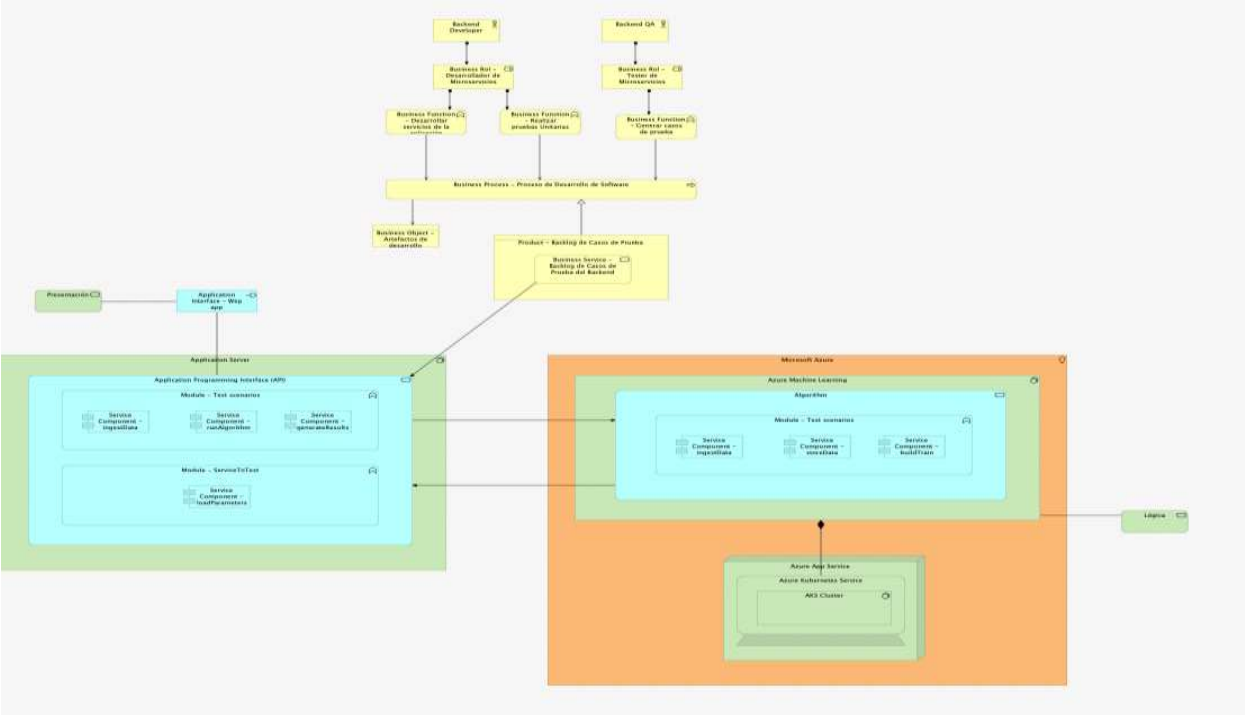
Para el presente proyecto se planteó una arquitectura a nivel de capas que busca satisfacer el soporte para una aplicación que tiene como objetivo generar escenarios de prueba. Esta propuesta tiene un impacto económico en el cliente, puesto que busca reducir las horas de soporte incurridas cuando un aplicativo presenta errores. El documento relacionado a este punto es:

- La arquitectura propuesta

Evidencia:

Figura 3

Arquitectura Propuesta



2.3 Student Outcome 3

2.3.1 Definición

Capacidad de comunicarse asertivamente con una audiencia.

2.3.2 Aplicación

Como parte de la investigación, se tuvo reuniones con el equipo de desarrollo y Testing del cliente. Estas sesiones brindaron al equipo un panorama completo sobre el alcance esperado y viable de la propuesta. Los documentos relacionados a este punto son:

- Actas de reunión
- Correo de autorización de la entidad

Figura 4

Acta de Reunión – Revisión del Proceso

Subject: RE: Review – Arquitectura Cloud – Generación Escenarios de Prueba

ACTA DE REUNIÓN

Se envía la siguiente acta con los puntos revisados en la reunión.

Participantes:
- Alexander Gamarra

Detalle:
- Se revisó el estado de la implementación del backlog para la arquitectura. Las dos actividades pendientes estarán finalizadas el **10/01/2022**

Acuerdos:

- Se realizará los diagramas de procesos
- Se va a validar el algoritmo planteado.
- Se revisará las mejoras encontradas para la memoria.
- Levantamiento de Observaciones.

En caso de haber algún error u omisión, favor de indicar.

Saludos cordiales,
Gianina A Ramirez

La validación se hizo con datos generados a partir de los datos reales. Ello con el fin de mantener la confidencialidad de las organizaciones y participantes que figuran en dichos formularios.

2.4 Student Outcome 4

2.4.1 Definición

El objetivo del presente Outcome es el desarrollo de un juicio informado después de reconocer las responsabilidades éticas y profesionales en diferentes situaciones del campo de la ingeniería.

2.4.2 Aplicación

Durante la etapa del levantamiento de información se respetó la ética por el trabajo de los colaboradores de la entidad. Asimismo, se tiene precaución con la confidencialidad de la información brindada por la entidad de acuerdo con la de protección de datos personales. Además, las entrevistas realizadas fueron con los permisos necesarios de los ejecutivos y sus respectivos jefes inmediatos. Finalmente, la ética profesional de la carrera se tuvo siempre presente con el objetivo de brindar la total transparencia de todo el proyecto. El documento relacionado a este punto es:

- Correo de autorización de la entidad

Figura 5

Acta de Reunión – Correo de Autorización

2.5 Student Outcome 5

2.5.1 Definición

Este outcome se define como la capacidad del trabajo en equipo, fusionando el liderazgo y creando un entorno de colaboración, asimismo, se establecen objetivos con el fin de ejecutar tareas para realizar su cumplimiento.

2.5.2 Aplicación

Durante el desarrollo del proyecto, se formaron equipos de trabajo conformados por los integrantes del grupo de investigación, los stakeholders, y un tester del proceso del área de desarrollo. Al utilizar la metodología ágil de Scrum se logró generar un ambiente colaborativo con foco en el trabajo eficiente. El documento relacionado a este punto es:

- Gestión de recursos humanos.

2.6 Student Outcome 6

2.6.1 Definición

Este outcome se define como la capacidad para analizar e interpretar información utilizando la experimentación y el desarrollo de la ingeniería para obtener conclusiones.

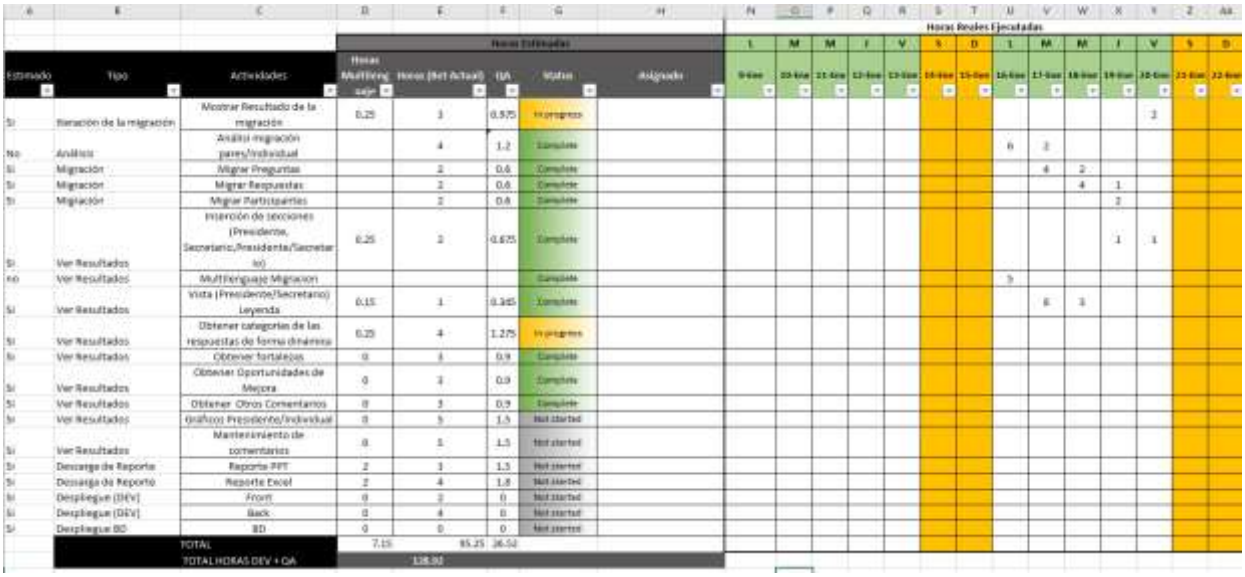
2.6.2 Aplicación

Los requisitos del proyecto se encuentran dentro de las historias de usuario las cuales abarcan las pruebas de concepto desarrolladas como el producto en su totalidad. Para ello, se diseñó un plan de pruebas el cual contiene los casos de prueba con sus diferentes flujos y criterios de aceptación. Por otro lado, se trabajó en base a los atributos de calidad que priman sobre el proyecto y la arquitectura en base a estos.

Evidencia:

Figura 6

Vista General de la Planificación de Horas Estimadas y Ejecutadas



Estimado	Tipo	Actividades	Horas Multiling uaje	Horas (Ref Actual)	QA	Status	Asignado	L 9-Ene	M 10-Ene	M 11-Ene	J 12-Ene	V 13-Ene	S 14-Ene	D 15-Ene	L 16-Ene	M 17-Ene	M 18-Ene	J 19-Ene	V 20-Ene	S 21-Ene	D 22-Ene
Si	Iteracion de la migracion	Mostrar Resultado de la migracion	0.25	1	0.375	In progress													2		
No	Analisis	Analisi migracion pares/Individual		4	1.2	Complete									6	2					
Si	Migracion	Migrar Preguntas		2	0.6	Complete										4	2				
Si	Migracion	Migrar Respuestas		1	0.6	Complete											4	1			
Si	Migracion	Migrar Participantes		2	0.6	Complete												2			
Si	Ver Resultados	Insercion de lecciones (Presidente, Secretario,Presidente/Secretario)	0.25	2	0.675	Complete												1	1		
no	Ver Resultados	Multilenguaje Migracion				Complete									3						
Si	Ver Resultados	Vista (Presidente/Secretario) Leyenda	0.15	1	0.345	Complete										6	3				
Si	Ver Resultados	Obtener categorias de las respuestas de forma dinamica	0.25	4	1.275	In progress															
Si	Ver Resultados	Obtener fortalezas	0	1	0.9	Complete															
Si	Ver Resultados	Obtener Oportunidades de Mejora	0	1	0.9	Complete															
Si	Ver Resultados	Obtener Otros Comentarios	0	1	0.9	Complete															
Si	Ver Resultados	Grafico Presidente/Individual	0	5	1.5	Not started															
Si	Ver Resultados	Mantenimiento de comentarios	0	1	1.5	Not started															
Si	Descarga de Reporte	Reporte PPT	2	1	1.5	Not started															
Si	Descarga de Reporte	Reporte Excel	2	4	1.8	Not started															
Si	Despliegue (DEV)	Front	0	2	0	Not started															
Si	Despliegue (DEV)	Back	0	4	0	Not started															
Si	Despliegue BD	BD	0	0	0	Not started															
	TOTAL		7.15	85.25	26.52																
	TOTAL HORAS DEV + QA		138.92																		

2.7 Student Outcome 7

2.7.1 Definición

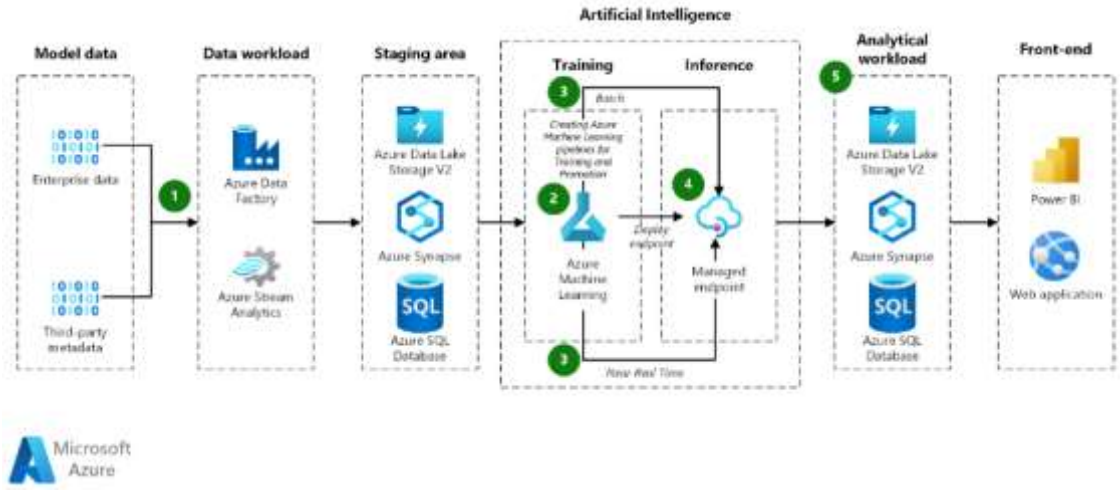
Este outcome se define como la capacidad profesional para adquirir e implementar nuevos conocimientos y/o tecnologías utilizando las mejores estrategias de aprendizaje.

2.7.2 Aplicación

Para la realización de nuestro proyecto fue fundamental aprender conceptos relacionados a Machine Learning y sobre algoritmos de redes neuronales. Para lograr nuestro objetivo, fue necesario diseñar un plan de aprendizaje que esté acorde al tiempo de duración del proyecto, por ejemplo, se diseñó un plan de aprendizaje para aprender sobre Azure Machine Learning estimado en las 2 primeras semanas del proyecto.

Figura 7

Recursos de Azure



Nota. Adaptado de "Inteligencia artificial en el perímetro con Azure Stack", por Microsoft, s.f. (https://learn.microsoft.com/es-es/azure/architecture/solution-ideas/articles/ai-at-the-edge).

CAPÍTULO 3. MARCO TEÓRICO

Con el paso del tiempo, nos vamos dando cuenta de la importancia de utilizar metodologías ágiles en el proceso de desarrollo de software. Debido a que, las organizaciones están expuestas a cambios y nuevos desarrollos tecnológicos. Es por esta razón, que el uso de entregables en cortos plazos permite una mejor capacidad de respuesta a estos cambios tecnológicos. Sin embargo, siempre se está en busca de la optimización del desarrollo de software, para este caso en específico la fase del aseguramiento de la calidad (Testing).

3.1 Técnica de pruebas de software (Fuzzing)

Durante la fase de pruebas del desarrollo del software se toman en cuenta diferentes técnicas y métodos para el aseguramiento de la calidad del sistema. Fuzzing es una técnica automatizada de pruebas para buscar errores y/o vulnerabilidades en una aplicación. El método utilizado por esta técnica es brindar datos de entradas erróneos al sistema con el objetivo de identificar bugs (Jitsunari & Arahori, 2019). Asimismo, utiliza la aleatoriedad para generar datos de prueba con el fin de detectar fallas. Las fallas indican errores y pueden representar una vulnerabilidad.

3.1.1 ¿Por qué usar **fuzzing/fuzz testing**?

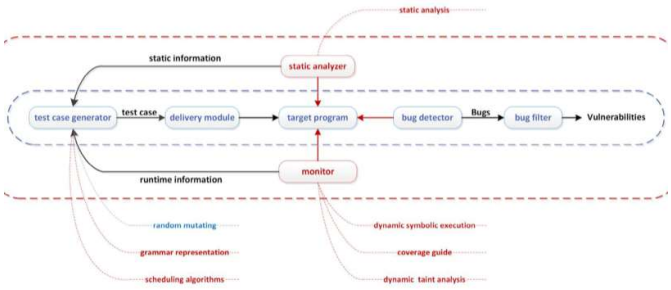
La eficiencia del fuzzing ha ido mejorando los programas durante los últimos 20 años. Asimismo, con la comprensión cada vez más profunda del proceso de fuzzing y el uso efectivo de las técnicas existentes, las técnicas de fuzzing brindarán un soporte técnico mejorado para las organizaciones (Chen et al., 2018).

3.1.2 Flujo general del fuzzing

Estructura inicial del fuzzing:

Figura 8

Anatomía del Fuzzing



Nota. Adaptado de "A systematic review of fuzzing techniques", por Chen et al., 2018 (https://doi.org/10.1016/j.cose.2018.02.002).

El primer módulo es para la generación de los casos de prueba; el siguiente módulo importante es el analizador estático, el cual tiene como función principal recopilar e informar los errores. Por último, la fase del filtrado de vulnerabilidades aclara si el error puede ser explotado. Generalmente, esto último se realiza manualmente y suele ser difícil de controlar (Chen et al., 2018)

3.1.3 Ejemplos

3.1.3.1 Fuzzing de caja blanca

El fuzzing de caja blanca tiene en cuenta la estructura del código para generar sistemáticamente nuevos datos de entradas para las pruebas. Asimismo, las nuevas entradas generadas están en su mayoría bien formadas y pueden guiar el código de destino hacia rutas de ejecución inexploradas, lo que da como resultado una mayor cobertura de código. (Jitsunari & Arahori, 2019)

3.1.3.2 Fuzzing de caja negra

Uso de algoritmos basados en fuzzing para probar directamente el código fuente. No se requiere ningún conocimiento experto sobre la estructura del código y, por lo tanto, es fácil de aplicar a varios tipos de aplicaciones. (Jitsunari y Arahori, 2019) Este método es el más utilizado por los piratas informáticos para detectar vulnerabilidades en una aplicación.

3.1.3.3 Fuzzing basado en mutaciones

Altera las muestras de datos actuales para crear nuevos datos de entrada. Es un método simple, que comienza con muestras precisas de reglas y sigue destruyendo cada byte o archivo. (Xu et al., 2020)

3.1.3.4 Fuzzing basado en generaciones

Define nuevos datos basados en la información del modelo. Existen algoritmos basados en fuzzing que generan resultados más eficientes, ya que, se alimentan de una estructura de datos normalizada y permiten tener casos de prueba de mayor precisión. (Xu et al., 2020)

3.1.4 Algunas herramientas basadas en fuzzing

3.1.4.1 Bfuzz

BFuzz es una herramienta fuzzer basada en .HTML, abre su navegador con una nueva instancia y pasa múltiples casos de prueba generados, además BFuzz realiza la misma tarea repetidamente sin destrozar ningún caso de prueba. (Gibson et al., 2020)

3.1.4.2 American fuzzy lop

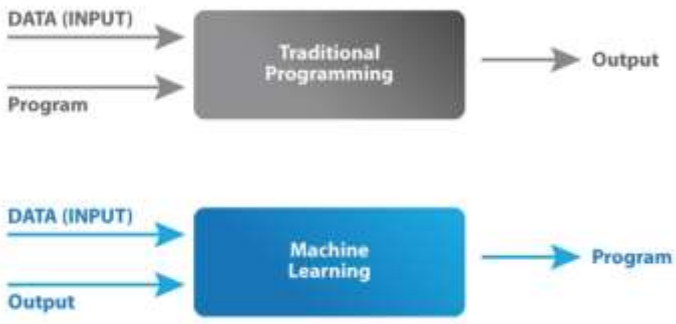
American fuzzy lop es un fuzzer orientado a la seguridad que emplea un tipo novedoso de instrumentación en tiempo de compilación y algoritmos genéticos, para descubrir automáticamente casos de prueba limpios e interesantes que desencadenan nuevos estados internos en el binario objetivo. (Gibson et al., 2020)

3.2 Aprendizaje automático (Machine Learning)

El aprendizaje automático posee la capacidad para que las máquinas puedan aprender sin ser expresamente programadas para ello. Es una rama de la inteligencia artificial y permite construir sistemas que reconozcan patrones o comportamientos según los diferentes parámetros de entrada. Sin embargo, el aprendizaje automático es un proceso complejo, ya que, necesita algoritmos y modelos para recolectar información e interpretar los datos para predecir resultados. Un modelo de machine learning se define como la salida de información seguido del entrenamiento.

Figura 9

Definición de Machine Learning



Nota. Adaptado de "What is Machine Learning?", por Great Learning Team, s.f. (https://www.mygreatlearning.com/blog/what-is-machine-learning)

3.2.1 Enfoques hacia el machine learning

Según los diferentes sectores y problemas sociales, existen diferentes categorías del aprendizaje automático.

3.2.1.1 Aprendizaje supervisado

El aprendizaje supervisado tiene como objetivo encontrar patrones en la información, para aplicar un proceso de analítica. Por ejemplo, se puede construir un software que interprete imágenes y prescripciones de lugares que se distingan entre diferentes bases de datos.

3.2.1.2 Aprendizaje no supervisado

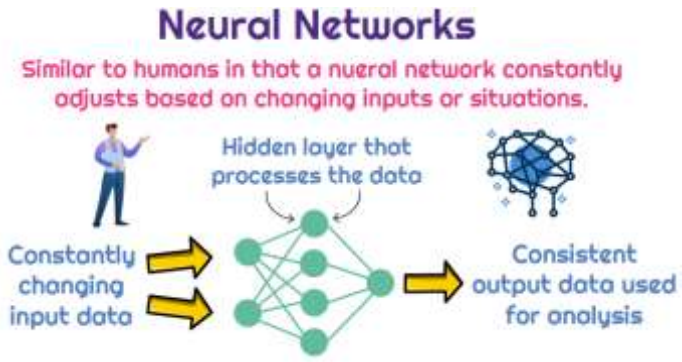
El aprendizaje no supervisado se utiliza cuando se tiene una enorme cantidad de información sin etiquetar, es decir, data que no tiene una relación en común. Por ejemplo, hoy en día las redes sociales manejan grandes volúmenes de información sin etiquetar. Los algoritmos tratan de identificar patrones que clasifiquen la información. Por ejemplo, una de las tecnologías implementadas utilizando este recurso es el uso de un algoritmo para detectar correos maliciosos y spam.

3.3 Red neuronal (Neuronal network)

Parte de la familia de los algoritmos de machine learning son las redes neuronales, este tipo de algoritmos pretende simular el comportamiento de un cerebro biológico a través de miles y miles de neuronas programadas artificialmente que se almacenan en capas formando millones de conexiones.

Figura 10

¿Qué es una Red Neuronal?



Nota. Adaptado de “What is a neural network in accounting and finance?”, por Universal CPA Review, s.f. (https://www.universalcpareview.com/ask-joey/what-is-a-neural-network-in-accounting-and-finance/)

3.3.1 Clasificación por el número de capas.

Podemos dividir esta clasificación en dos tipos:

- Mono-capas: Básicamente la capa de entrada se conecta de manera directa a la capa de neuronas de salida.
- Multi-capas: Entre las conexiones de entrada y de salida, existen diferentes capas ocultas que hacen de by-pass. Estas capas ocultas de neuronas pueden interconectarse con ellas de maneras automáticas con ellas o no.

3.3.2 Clasificación por los tipos de conexiones

Se pueden clasificar las redes neuronales en dos tipos de conexiones:

- Redes neuronales no recurrentes: Este tipo de redes neuronales, se define porque opera en una sola dirección o sentido; por ello, no tienen retroalimentación y además carecen de memoria. (Finance, 2019)
- Redes neuronales recurrentes: Las neuronas tienen la capacidad de realizar conexiones y realimentación operando entre neuronas de una misma o diferentes capas. Asimismo, la realimentación provoca que las redes neuronales recurrentes, tengan memoria. Esta tipificación posee mayores ventajas en general y cuentan con mayor potencia, que las redes neuronales no recurrentes. (Finance, 2019)

3.3.3 Redes neuronales convolucionales

La principal diferencia de las redes neuronales convolucionales con las otras es que se puede entrenar a nivel de multicapas las redes con el objetivo de conseguir mayor velocidad de procesamiento e identificar patrones de una forma más avanzada.

CAPÍTULO 4. ESTADO DEL ARTE

4.1 Prefacio

El desarrollo de software está en constante evolución, cada vez son más las nuevas tecnologías que se integran al ciclo de vida del desarrollo software y una de las fases más importantes es el aseguramiento de la calidad. El presente trabajo de investigación tiene como objetivo el diseño de un nuevo enfoque basado en fuzzing que haga uso de machine learning para la generación de casos de pruebas.

Para dar sustento científico a esta investigación, primero se establecieron las siguientes palabras claves: Fuzzing, Modelo Generativo y Machine learning; estas fueron los keywords que utilizamos para la búsqueda de artículos científicos en diferentes motores de búsqueda como “Scopus” y/o Web of Science.

Por un lado, definimos 4 preguntas de investigación con el objetivo de ir acotando los artículos a nuestro tema de interés: ¿Cómo realizar pruebas de fuzzing para mejorar la calidad del software de forma automatizada?, ¿Qué técnicas, modelos o frameworks generativos se han realizado respecto al fuzzing?, ¿Cuál es la relación entre machine learning y modelo generativo? ¿Qué tipos de técnicas se han utilizado? Y ¿Qué tipos de estudios se han realizado utilizando machine learning a través de un modelo generativo en el testing?

Por otro lado, definimos los siguientes criterios de búsqueda: Los artículos de investigación deben haberse publicado entre las fechas del 2018 y 2022; la revista donde es publicado el artículo debe tener el quartil Q1 o Q2 como mínimo. Si se da el caso de ser una conferencia, se debe verificar que el rank sea de A* o A.

Como resultado, obtuvimos un total de 20 artículos científicos, para ello tuvimos un constante trabajo en equipo con el coautor, quién se encargó de aprobar el contenido de los artículos en relación con la propuesta planteada. Asimismo, estos artículos se dividieron en tres tipologías:

La primera topología representa el conjunto de artículos relacionados con las últimas soluciones realizadas con fuzzing a nivel general. Asimismo, contiene información sobre la priorización de casos de prueba y cómo estás ayudan a mejorar el % en la cobertura de pruebas del sistema.

La segunda topología agrupa artículos relacionados con técnicas que utilizan modelos generativos para resolver problemas de seguridad basados en fuzzing.

La tercera y última topología contiene artículos relaciones con algoritmos de aprendizajes automáticos aplicados en diferentes soluciones de la vida real.

Para terminar en la última parte del documento, hemos realizado una sección con las conclusiones de cada topología, así como las conclusiones generales de la investigación.

4.2 Metodología

4.2.1 Desarrollo

La fase inicial de la investigación fue la definición de las preguntas de investigación en base a los keywords del proyecto. Se utilizaron diferentes bibliografías para la revisión sistemática de los documentos.

Tabla 5

Preguntas de Investigación

Término	Código	Pregunta
Fuzzing y Machine Learning	RQ1	¿Cómo realizar pruebas de fuzzing para mejorar la calidad del software de forma automatizada?
Fuzzing y Modelo Generativo	RQ2	¿Qué técnicas, modelos o frameworks generativos se han realizado respecto al fuzzing?
Machine Learning y Modelo generativo	RQ3	¿Cuál es la relación entre machine learning y modelo generativo? ¿Qué tipos de técnicas se han utilizado?
Machine Learning, Fuzzing y Modelo Generativo	RQ4	¿Qué tipos de estudios se han realizado utilizando machine learning a través de un modelo generativo en el testing?

4.2.2 Tabla de artículos

Tabla 6

Resumen del Estado de Arte

Tipología	Titulo	Fuente
Fuzzing	Embedded fuzzing: a review of challenges, tools, and solutions	Cybersecurity
	Acceptance testing based test case prioritization	Cogent Engineering
	Improved Test Case Selection Algorithm to Reduce Time in Regression Testing	Computers, Materials & Continua
	Optimizing seed inputs in fuzzing with machine learning	IEEE International Conference on Software Engineering
	Fuzzing: Cyberphysical System Testing for Security and Dependability	Computer
	Compiler Fuzzing through Deep Learning	International Symposium on Software Testing and Analysis
	Adaptive Grey-Box Fuzz-Testing with Thompson Sampling	ACM Conference on Computer and Communications Security
Modelo generativo	A systematic review of fuzzing techniques	Computers and Security
	Format-aware Learn&Fuzz: Deep Test Data Generation for Efficient Fuzzing	Neural Computing and Applications
	CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing of Deep Learning Systems	IEEE Transactions on Software Engineering
	Coverage-guided Learning-assisted Grammar-based Fuzzing	IEEE International Conference on Software Testing

	A Generative Model for Evasion Attacks in Smart Grid	IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)
	Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0	MDPI
Machine Learning	Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms	IEEE Access
	More Effective Test Case Generation with Multiple Tribes of AI	IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings
	DeepFuzz: Automatic Generation of Syntax Valid C Programs for Fuzz Testing	The Thirty-Third AAAI Conference on Artificial Intelligence
	A systematic review of fuzzing based on machine learning techniques	PLOS ONE
	Machine Learning Augmented Fuzzing	IEEE International Symposium on Software Reliability Engineering Workshops
	MTFuzz: Fuzzing with a Multi-task Neural Network	ACM Joint European Software Engineering Conference and Symposium

		on the Foundations of Software Engineering
	Fuzzing-base d hard-lab el black-box attacks against machine learning models	Computers & Security

Nota. La tabla mostrada especifica los artículos científicos analizados para el desarrollo de la investigación

4.3 Resumen de los artículos científicos

4.3.1 Artículos de la categoría machine learning

4.3.1.1 Embedded fuzzing: a review of challengues, tools and solutions

Aporte:

Mediante el siguiente artículo, los autores presentan una revisión sistemática sobre el área de enfoque del fuzzing, con el objetivo de conocer las contribuciones claves que se han realizado y sus respectivas limitaciones.

Proceso:

Los autores plantearon inicialmente que el fuzzing también puede ser usado en sistemas embebidos, debido a que los sistemas integrados están en auge en la sociedad moderna y hoy en día se utilizan en innumerables ejemplos: marcapasos, robots, internet de las cosas (IOT) y sistemas ciber físicos (Eisele et al., 2022). Como resultado, realizaron una investigación exhaustiva para ver la posibilidad de considerar el uso de fuzzing en sistemas integrados.

En primer lugar, los autores formularon la siguiente pregunta para iniciar la investigación: ¿Cuáles son las principales características y limitaciones de las herramientas actuales para el fuzzing?" (Eisele et al., 2022).

Después se utilizó los siguientes criterios de inclusión para la generación del estado del arte:

Tabla 7

Fuentes de Investigación – Artículo 1

Criterio	Fuente de investigación
N.º 1	Artículos de investigación que se publican en los cinco lugares principales en la categoría "Ingeniería y ciencias de la computación",

	subcategoría "Seguridad informática y criptografía" según Google Scholar (Scholar 2021).
N.º 2	Artículos de investigación que se publican durante los cinco años entre 2017 y 2021. (Max Eisele, 2022).
N.º 3	Artículos de investigación que mencionen "fuzzing" y "frmware" o, alternativamente, "fuzzing" e "incrustado". (Max Eisele, 2022).
N.º 4	Documentos o herramientas de investigación que consideramos que transmiten enfoques relevantes para el fuzzing integrado. (Max Eisele, 2022).

Nota. Información adaptada de “Embedded fuzzing: a review of challengues, tools and solutions”, por Eisele et al., 2022

Estos criterios ayudaron a contemplar diferentes enfoques del fuzzing y posteriormente su revisión sistemática

En tercer lugar, los autores crearon una matriz con los artículos encontrados con el fin de contrastar las contribuciones claves y las limitaciones en el fuzzing.

Tabla 8

Trabajos Realizados en el Fuzzing

Categoría del artículo	Contribuciones claves	Limitaciones
Instrumentación basada en hardware	Fuzzing al límite Módulo de hardware para recopilar datos de cobertura	Se requiere hardware específico Necesita ser compilado en el kernel
Interfaz de mensajes	Mecanismo IoTFuzzer mejorado Uso de la salida de registro para comentarios	Se necesitan registros detallados.
Emulación de modo de usuario basada en emulación	Fuzzing de páginas web de configuración de IoT Kernel personalizado para emulación	Solo aplicaciones basadas en Linux

Emulación de sistema completo	Fuzzing guiado por cobertura con QEMU Fuzzing guiado por cobertura en VP Controlador fuzzing para vehículos robóticos	El destino debe ser emulable por QEMU Se requiere prototipo virtual Se requiere una simulación física

Nota. Información adaptada de "Embedded fuzzing: a review of challengues, tools and solutions", por Eisele et al., 2022

Principal Resultado:

Tras realizarse la revisión sistemática en los entornos previamente descritos (Tabla 4), los autores llegaron a la conclusión de que aún existen restricciones y límites definidos para realizar el fuzzing en sistemas embebidos, ya que no se pueden hacer declaraciones generalizadas sobre las interfaces. Esta es la razón por la cual el campo de investigación del fuzzing integrado todavía carece de soluciones genéricas e incluso comparar diferentes herramientas aún sigue siendo un gran desafío. (Eisele et al., 2022)

4.3.1.2 Acceptance testing-based test case prioritization

Aporte: En el siguiente artículo, los autores plantean un modelo para la priorización de casos de prueba, utilizando un algoritmo de conjuntos (GA), con el objetivo de reducir los casos ejecutados durante las pruebas de regresión.

Proceso: El tiempo para ejecutar las pruebas de regresión aumenta considerablemente a medida que avanza el desarrollo del sistema. En la siguiente sección, se mencionará los puntos más importantes durante el proceso de desarrollo de la solución.

La solución propuesta por los autores es la incorporación de la técnica de optimización GA para la reducción y priorización de casos de prueba.

Algoritmo GA

- Paso 1: Selección de un conjunto de casos de prueba, {T1….Tn}
- Paso 2: Selección de un conjunto de sentencias {S1…..Sn}
- Paso 3: Se establece 1 nivel de reducción {R1}
- Paso 4: Se establece una variable {S1} para el conjunto de pruebas reducido.
- Paso 5: Se realiza la llamada genética {F}

- Paso 6: Se recopila los casos de pruebas refinados y reducidos por la llamada genética**.**
- Paso 7: Se realiza todas las iteraciones posibles del algoritmo hasta obtener una reducción de casos de prueba >=57% de aceptación (Geetha et al., 2021).

Asimismo, se realizó un análisis de rendimiento de la técnica (GA) para la priorización de casos de prueba. En la siguiente figura se muestra cómo se priorizaron los casos en función a las iteraciones del algoritmo realizadas con base en la matriz de cobertura inicial de casos de prueba (Geetha et al., 2021).

Figura 11

Priorización de Casos de Prueba

Técnica	PTS	Tasa de detección de fallas (%)
Aleatorio 1	T2,T4,T6,T5	68
Aleatorio 2	T6,T4.T2,T5	62
Aleatorio 3	T4, T6, T5, T2	55
Basado en la aceptación	T5,T2,T4,T6	89

Nota. Información de "Acceptance testing-based test case prioritization", por Geetha et al., 2021 (https://doi.org/10.1080/23311916.2021.1907013).

Como se observa en la **Figura 11**, la columna PTS (Tx….Tn) contiene a los conjuntos de casos de prueba priorizadas. En la primera iteración del algoritmo GA (aleatorio 1) se obtuvo la siguiente priorización {t2, t4, t6, t5} con una tasa de detección de fallas del 68%. Finalmente, la cuarta iteración del algoritmo obtuvo un 89% para la detección de fallas, siendo esta la mejor priorización del algoritmo.

Principal Resultado:

El algoritmo GA reduce la redundancia en un conjunto de casos de pruebas y genera casos priorizados en base a una predicción. Asimismo, La reducción de casos de prueba en su sistema fue de un 57% menor en porcentaje del número de casos iniciales, reduciendo el tiempo para realizar las pruebas de regresión y mejorando la cobertura de casos de prueba (Geetha et al., 2021).

4.3.1.3 Improved Test Case Selection Algorithm to Reduce Time in Regression Testing

Aporte:

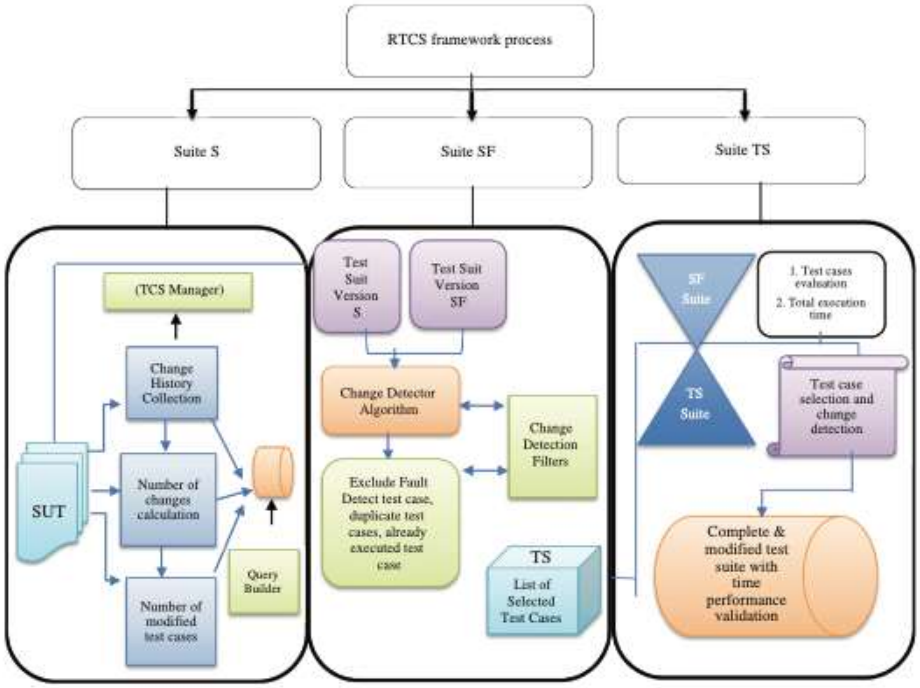
Mediante el siguiente artículo, los autores describen una nueva técnica que utiliza un algoritmo de selección de casos de prueba para reducir el tiempo de ejecución de las pruebas de regresión. Asimismo, los autores introducen un nuevo enfoque al mejorar la precisión del plan de pruebas y la detección de casos de pruebas modificados.

Proceso

Durante la fase inicial de la investigación se involucró el proceso de identificar las carencias y/o mejoras para asegurar la calidad del software. De esta manera, los autores plantearon un panorama general de las pruebas de regresión y su relación con un nuevo enfoque más eficiente.

Figura 12

Artículo 3 - Marco de Proceso de Pruebas de Regresión



Nota. Adaptado de "Improved Test Case Selection Algorithm to Reduce Time in Regression Testing", por Ghani et al., 2022 (https://doi.org/10.32604/cmc.2022.025027).

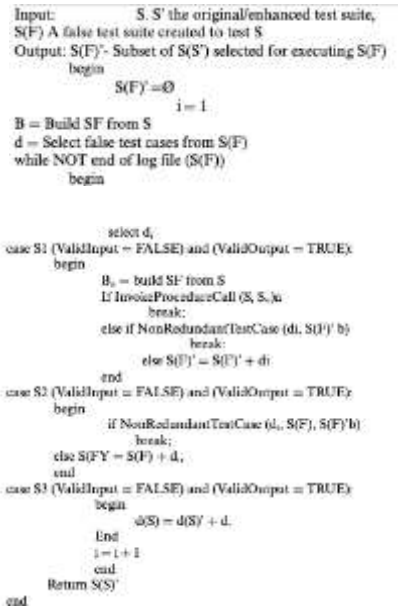
Como se observa en la **Figura 12**, los autores consideraron un flujo de detección de cambios con el objetivo de mejorar la selección de casos de prueba.

Algoritmo de selección de casos de prueba

Para asimilar mejor el siguiente algoritmo se debe tener en cuenta lo siguiente: conjunto de pruebas original/mejorado: **S'**, conjunto de pruebas falso creado para probar S: **S(F)'**. El algoritmo comprende los casos de prueba duplicados, falsos y ya ejecutados, los cuáles se probaron para obtener los resultados esperados en el tiempo dado (Ghani et al.,2021). Para asegurar los criterios mencionados, el algoritmo de selección de casos de prueba es presentado y mostrado de la siguiente manera:

Figura 13

Artículo 3, Algoritmo de Selección de Casos de Prueba



Nota. Información de "Improved Test Case Selection Algorithm to Reduce Time in Regression Testing", por Ghani et al., 2021 (https://doi.org/10.32604/cmc.2022.025027).

Paso a paso del algoritmo de selección:

Tabla 9

Pasos del Algoritmo de Selección

N.º	Descripción
1	Selección de casos de prueba usando reglas de mapeo de selección de casos de prueba

2	Dividir el conjunto de pruebas Conjuntos de pruebas de selección y detección
3	Insertar casos de prueba en la base de datos seleccionada para seleccionar casos de prueba modificados
4	Aplicar el algoritmo de detección de cambios contra los casos de prueba modificados seleccionados para excluir el detector de fallas/duplicados y los casos de prueba ya ejecutados de la prueba modificada suite
5	Enviar la solicitud para detectar los casos de prueba modificados.
6	Reciba una respuesta para los cambios seleccionados y modificados del conjunto de pruebas
7	Calcule el tiempo contra los casos de prueba únicos seleccionados y detectados a partir de modificados Banco de pruebas.

Nota. Información adaptada de "Improved Test Case Selection Algorithm to Reduce Time in Regression Testing", por Ghanie et al., 2021.

Este procedimiento se ha realizado en cuatro iteraciones: 50, 100, 150 y 200 casos de pruebas. En la siguiente imagen podemos verificar la eficacia del uso del algoritmo de selección:

Figura 14

Artículo 3, 200 Casos de Prueba

Nota. Imagen de "Improved Test Case Selection Algorithm to Reduce Time in Regression Testing", por Ghani et al., 2021(https://doi.org/10.32604/cmc.2022.025027).

El tiempo de ejecución se redujo en 5 segundos para 200 casos de prueba.

Principal Resultado: La implementación del algoritmo de selección de casos de prueba de regresión y la detección del enfoque de cambio redujo los tiempos de ejecución de la suite

de pruebas, asimismo, el enfoque propuesto reduce la redundancia de los casos de pruebas duplicados y detecta solo los cambios modificados. Por lo tanto, las pruebas de regresión son más precisas y eficientes (Ghani et al.,2021).

4.3.1.4 Optimizing seed inputs in fuzzing with machine learning

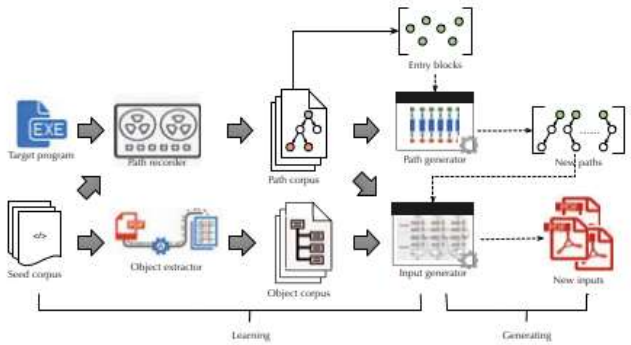
Aporte: Los autores presentan un marco basado en el aprendizaje automático que descubre y aprovecha la correlación entre las entradas iniciales y la ejecución del programa de destino para generar nuevas entradas iniciales que activan una mayor cobertura de código del programa de destino en tres pasos: Preparación de datos, Generación de la ruta y Generación de semillas.

Proceso: El marco planteado en el artículo utiliza un modelo generativo que se basa en redes neuronales recurrentes (RNN) para generar nuevas rutas de ejecución del programa de destino no cubiertas por el corpus semilla original. Luego, las nuevas rutas de ejecución se reenvían a un modelo de transición basado en secuencia a secuencia (Seq2seq) para traducirlas en archivos PDF válidos (es decir, nuevas entradas de inicialización) y activarlas. En estas tareas, ambos modelos se entrenan con las entradas semilla originales y la ejecución correspondiente del programa de destino.

Se utilizó RNN para aprender tanto las entradas iniciales como las rutas de ejecución porque supera a otros modelos al aprender secuencias largas de tokens discretos con ciertas estructuras sintácticas, la propuesta se desarrolla en tres pasos:

Figura 15

Artículo 4, Marco para Mejorar las Entradas de Semillas en Fuzzing



Nota. Adaptado de "Optimizing seed inputs in fuzzing with machine learning", por Cheng et al., 2019 (https://doi.org/10.1109/ICSE-Companion.2019.00096).

- Paso 1: Preparación de datos. El Path Recorder de la Figura 15, construido sobre la herramienta de instrumentación Pin de Intel, primero alimenta el corpus semilla original al programa de destino y registra las secuencias de ejecución resultantes. Estas rutas de ejecución se codifican como las direcciones iniciales de los bloques básicos a lo largo de las rutas y se almacenan en el corpus de rutas. Dado que las rutas de ejecución a menudo son demasiado largas para ser manejadas por los modelos RNN, se introduce un algoritmo de compresión de rutas para comprimir rutas largas a una longitud inferior a 300 mediante la sustitución de secuencias cortas de bloques básicos compartidos por múltiples rutas de ejecución con superbloques.
- Paso 2: Generación de ruta. Las rutas de ejecución en el corpus de rutas se utilizan para entrenar el Generador de rutas, un modelo de lenguaje basado en RNN construido sobre la implementación Char-RNN de Andrej Karpathy1, para aprender la distribución condicional de bloques básicos en estas rutas. Este modelo de lenguaje hereda su estructura de dos capas de char-RNN estándar, una para aprender cómo los bloques básicos forman funciones y la otra para aprender cómo las funciones forman rutas de ejecución completas, donde el número de estados ocultos en cada capa se establece en 256.
- Paso 3: Generación de semillas. Las rutas de ejecución producidas por el Generador de rutas son 'traducidas' por el Generador de entradas en archivos PDF (es decir, nuevas entradas iniciales) que activan estas rutas de ejecución. El generador de entrada incluye: 1) un extractor de objetos que recupera todas las secuencias de objetos de los archivos PDF en el corpus inicial original; y 2) un modelo Seq2Seq que, después de entrenarse con el corpus de ruta y las secuencias de objetos correspondientes recuperadas por el Extractor de objetos, aprende y aprovecha la correlación entre el corpus semilla original y el corpus de ruta para lograr una traducción precisa de nuevas rutas de ejecución a nuevos insumos de semillas.

Principal Resultado:

Utilizando el marco planteado generó que el corpus semilla aprendiendo la gramática de los archivos PDF y los nuevos corpus cubrieron un 0,11 % más de instrucciones. Seguidamente, evaluaron su framework probando MuPDF y otros tres visores de PDF (pdfium, podofo y poppler) con el corpus original y generado durante 24 horas. Esto produjo resultados

similares: las entradas semilla mejoradas generadas por nuestro marco exploraron en promedio un 23,21 % más de bloques básicos y un 31,69 % más de rutas de ejecución. Asimismo, se el marco a libpng (una biblioteca de referencia PNG) y freetype (una biblioteca TTF de código abierto). Entrenado con archivos PNG y TTF extraídos de los archivos PDF descargados, nuestro marco generó nuevas entradas iniciales que llevaron a un aumento significativo de la cobertura de código en ambos programas de destino (es decir, 53,90 % más de rutas y 22,38 % más de bordes cubiertos en tipo libre después de 24 horas de fuzzing)). Finalmente, los autores sugieren que su marco propuesto es aplicable a otros formatos de entrada complejos.

4.3.1.5 Fuzzing: Cyberphysical System Testing for Security and Dependability

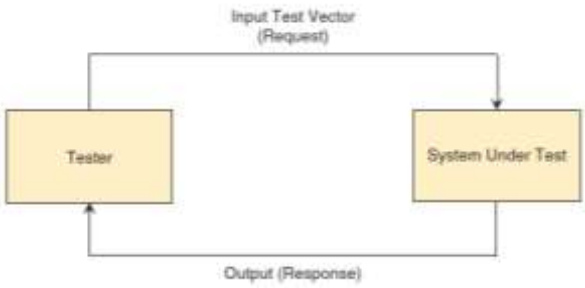
Aporte: Los autores de esta publicación buscan explorar un nuevo ámbito considerando las pruebas continuas y automatizadas (fuzzing) como un método prometedor para confirmar el funcionamiento correcto, seguro y protegido de los CPS (sistemas ciber físicos) y los sistemas interconectados de IoT en el campo.

Proceso:

Considerando que el enfoque en los CPS y los sistemas IoT, a continuación, los autores plantean que el SUT (Sistema bajo prueba) es un sistema IoT conectado a la red, que se prueba (fuzzed) en una red, ya sea en la fase de diseño o en tiempo de ejecución en el campo. Por lo tanto, los vectores de entrada en son paquetes enviados a través de una red (solicitudes en el sistema distribuido) y las salidas son respuestas.

Figura 16

Artículo 5, Configuración de Prueba



Nota. Adaptado de "Fuzzing: Cyberphysical System Testing for Security and Dependability", por D. Serpanos y K. Katsigiannis, 2021 (https://doi.org/10.1109/10.1109/MC.2021.3092479).

Además, los autores plantean un enfoque en los CPS y los sistemas IoT incluyendo nuevos protocolos de comunicación y aplicación, los cuales proporcionará una herramienta importante para proporcionar un IoT seguro y protegido.

Esta publicación considera entornos de caja blanca usando American Fuzzy Lop como estrategia basándose en resultados anteriores, asimismo, utilizando técnicas que incluyen el cambio de bits a pie y la aritmética simple. Otros, como Sulley, se basan en bloques e incluyen bibliotecas de valores precalculados y cadenas especiales, que se aplican bajo la dirección del usuario (probador). MTF Storm es un Modbus fuzzer automatizado que dirige su proceso de selección a través de errores de codificación de software, como el manejo de valores y condiciones límite, así como las condiciones de procesamiento del formato límite.

Principal Resultado: Teniendo en cuenta las fases de funcionamiento de un fuzzer, queda claro que se puede proporcionar soporte de IA/ML en la mayoría de ellas, si no en todas. El reconocimiento puede beneficiarse de las técnicas que capturan y analizan la sintaxis y la gramática de los protocolos, incluidos los patentados, impulsando la síntesis de vectores de prueba.

Aunque ML/AI aún no ha proporcionado un progreso significativo para las pruebas y fuzzing, algunos resultados preliminares indican un crecimiento significativo en el campo y un alto potencial de mejora en el desarrollo de fuzzers efectivos y eficientes

4.3.1.6 Fuzzing del compilador a través del aprendizaje profundo

Aporte: Los autores presentan DeepSmith, un nuevo enfoque de aprendizaje automático para acelerar la validación del compilador a través de la inferencia de modelos generativos para las entradas del compilador. El enfoque que proponen infiere en un modelo aprendido de la estructura del código del mundo real basado en un gran corpus de código fuente abierto. Luego, utiliza el modelo para generar automáticamente decenas de miles de programas realistas. Finalmente, aplican metodologías de pruebas diferenciales establecidas sobre ellos para exponer errores en los compiladores.

Proceso:

Se plantea un enfoque rápido, efectivo y de bajo esfuerzo para la generación de programas aleatorios para compilar fuzzing a través de una metodología que utiliza avances recientes en aprendizaje profundo para automáticamente construir modelos probabilísticos de cómo los humanos escriben código, en lugar de definir minuciosamente una gramática con el

mismo fin. Al entrenar una red neuronal profunda en un corpus de código escrito a mano, puede inferir tanto la sintaxis como la semántica del lenguaje de programación y las construcciones y patrones comunes. El enfoque de los propietarios del artículo es esencialmente enmarcar la generación de programas aleatorios como un problema de modelado de lenguaje. Esto simplifica y acelera enormemente el proceso. La expresividad de los programas generados está limitada únicamente por lo que está contenido en el corpus, no por la experiencia del desarrollador o el tiempo disponible. Dicho corpus se puede ensamblar fácilmente a partir de repositorios de código abierto.

El paper expone el objetivo principal, el OpenCL, un estándar abierto para programar sistemas heterogéneos. Básicamente, por las siguientes razones: es un estándar emergente con la desafiante promesa de portabilidad funcional en una amplia gama de hardware heterogéneo; OpenCL se compila "en línea", lo que significa que es posible que incluso los bloqueos y bloqueos del compilador no se descubran hasta que se implemente un producto para los clientes; y ya existe un generador de programas aleatorios escrito a mano para comparar el lenguaje. Brindamos resultados preliminares que respaldan el potencial de DeepSmith para el fuzzing de compiladores multilingües.

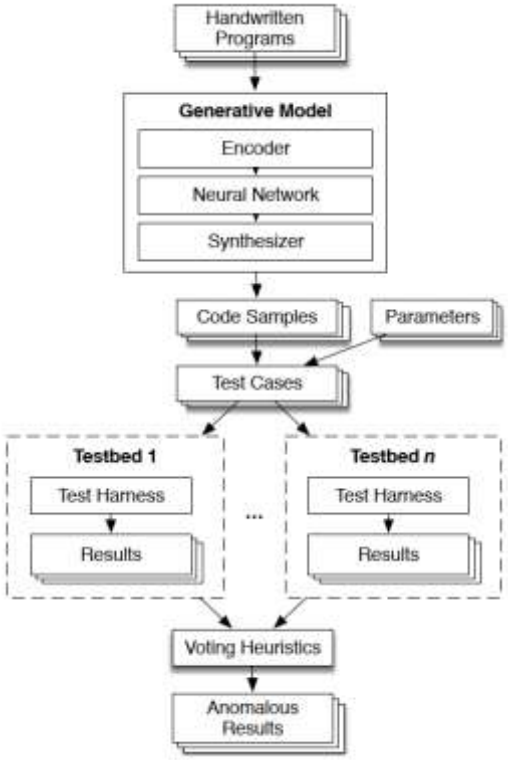
Como parte del proceso es generar casos de prueba para compiladores; sin embargo, es difícil porque sus entradas están muy estructuradas. Producir un texto con la estructura adecuada requiere un conocimiento experto y un importante esfuerzo de ingeniería, que debe repetirse desde cero para cada nuevo idioma. En cambio, tratamos el problema como una tarea de aprendizaje automático no supervisada, empleando técnicas de aprendizaje profundo de última generación para construir modelos sobre cómo los humanos escriben programas. La línea de la solución está inspirada en resultados innovadores en el modelado de conjuntos de datos desafiantes y de alta dimensión a través del aprendizaje no supervisado.

Principal Resultado:

Se presenta un framework para compilar fuzzing a través de la generación de programas aleatorios obteniendo como resultado docenas de errores en las implementaciones de OpenCL. Se exponen errores en partes del compilador donde los enfoques actuales no lo han hecho.

Figura 17

Artículo 6, Descripción General del Sistema DeepSmith



Nota. Adaptado de "Compiler Fuzzing through Deep Learning", por Cummins et al., 2018 (https://doi.org/10.1145/3213846.3213848).

4.3.1.7 Adaptive Grey-Box Fuzz-Testing with Thompson Sampling

Aporte: Los autores, presentan un enfoque de aprendizaje automático que se basa en AFL, el preeminente fuzzer de caja gris, mediante el aprendizaje adaptativo de una distribución de probabilidad sobre sus operadores de mutación sobre una base específica del programa. Estos operadores, que se seleccionan uniformemente al azar en AFL y fuzzers mutacionales en general, dictan cómo se generan nuevas entradas, una parte central de la eficacia del fuzzer. El artículo presenta dos aportes: primero, mostramos que una distribución de muestreo sobre operadores de mutación estimados a partir de programas de capacitación puede mejorar significativamente el rendimiento de AFL. En segundo lugar, presentan un enfoque de optimización basado en bandidos de Thompson Sampling que ajusta la distribución del mutador de forma adaptativa, durante el curso de fuzzing de un programa individual.

Proceso:

En el artículo se brinda detalles del enfoque para estimar distribuciones sobre mutadores fuzzing de caja gris. Primero, se presenta un par de ejemplos motivadores y último se especifica un método para aprender una mejor distribución estacionaria sobre mutadores (para verificar que existen mejores distribuciones de mutadores que la uniforme)

Para comprender el efecto que tiene la distribución de mutadores en el fuzzing se presenta el siguiente ejemplo

Servidor de contenidos ASCII. Un extracto simplificado del programa en C ASCII_Content_Server, que demuestra la funcionalidad de alto nivel para un simple

servidor web HTTP basado en texto. El programa soporta la funcionalidad para iniciar una sesión web, en la que un usuario (a través de STDIN) puede emitir

una serie de comandos para crear páginas web ("SEND"), buscar, interactuar y visualizar páginas ("QUERY", "INTERACT", "VISUALIZE"), y emitir llamadas a la API ("REQUEST"). Las sesiones pueden consistir en un número arbitrario de estos comandos, ya que todos se especifican a través de STDIN. Sin embargo, tenga en cuenta que cada comando comienza con la palabra clave palabra clave (como una cadena) del tipo dado, seguido por los datos para el comando dado (por ejemplo, "REQUEST <DATA>") (Cummins et al., 2018).

La mejor manera de optimizar el fuzzing es aprender las distribuciones de los mutadores de forma adaptativa. Sin embargo, es necesario evaluar si existen mejores distribuciones fijas de mutadores que la distribución uniforme utilizada por AFL, para tener una idea exacta de lo ineficiente que es la heurística actual de AFL. Para ello, esbozamos un enfoque que encuentra una distribución fija mediante el análisis y la extrapolación de un conjunto de ejecuciones completas de fuzzing en un conjunto de programas que no están relacionados con los que probamos. Intuitivamente, el objetivo aquí es utilizar los datos recogidos en programas no relacionados para aprender una nueva distribución de mutadores que funcione mejor que AFL. Aunque esta distribución estacionaria estimada empíricamente no es óptima para un programa individual (ya que no se ajusta de forma adaptativa), nuestro enfoque para aprenderla tiene dos ventajas clave: 1) nos permite hacernos una idea de la ineficacia actual del AFL, al tiempo que proporciona un punto de comparación sólido para nuestra estrategia de muestreo adaptativo de Thompson, y 2) nos permite intuir cómo utilizar los datos para

informar de la elección de las de las distribuciones de los mutadores, antes de profundizar en el enfoque de muestreo adaptativo de Thompson (Cummins et al., 2018).

Principal Resultado:

Los autores ofrecen detalles de cada lote de experimentos que realizan. Muchos de los siguientes experimentos consisten en comparar muchas diferentes estrategias (por ejemplo, Thompson Sampling vs. FidgetyAFL, vs. AFL). En muchos de nuestros experimentos, evaluamos tanto AFL y FidgetyAFL como líneas de base - en estos casos, tratamos FidgetyAFL como nuestra mejor línea de base, debido a su rendimiento significativamente mejor. En Además de informar de los recuentos agregados sobre qué estrategia mejor por programa, también informamos de la Cobertura Relativa, una métrica que proporciona una medida aproximada de lo mejor que es una estrategia una estrategia en relación con las demás (ya que AFL no instruye el programa para darnos las estadísticas básicas de cobertura de bloques adecuadas): Sea s corresponde a una estrategia determinada, t al intervalo de tiempo dado, y código-trayectorias (s) es el número de rutas de código únicas que la estrategia se ha descubierto en el tiempo t (Cummins et al., 2018).

4.3.2 Artículos de la categoría modelo generativo

4.3.2.1 A systematic review of fuzzing techniques

Aporte

Mediante el siguiente artículo, los autores realizaron una investigación profunda sobre los sistemas fuzzing, donde se tomaron en cuenta principios, ventajas y desventajas de las técnicas utilizadas, y cuyo objetivo fue establecer una referencia para futuras investigaciones.

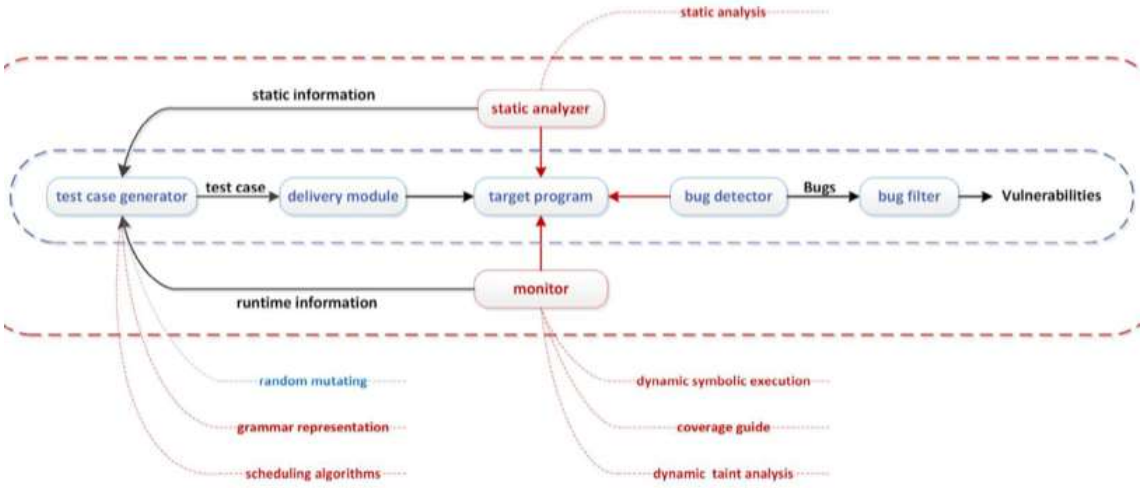
Proceso

La investigación realizada por los autores se ha dividido en varias secciones, con el propósito de analizar y tener diferentes puntos de vista del fuzzing, sus sistemas y las técnicas utilizadas.

Asimismo, antes de iniciar con el proceso de investigación los autores realizaron una estructura inicial del fuzzing, con el objetivo de tener un panorama general de la técnica.

Figura 18

Artículo 8, Anatomía del Fuzzing



Nota. Adaptado de "A systematic review of fuzzing techniques", por Chen et al., 2018 (https://doi.org/10.1016/j.cose.2018.02.002).

Como se observa en la imagen, el primer módulo es para la generación de los casos de prueba; el siguiente módulo importante es el analizador estático, el cual tiene como función principal recopilar e informar los errores. Por último, la fase del filtrado de vulnerabilidades aclara si el error puede ser explotado. Generalmente, esto último se realiza manualmente y suele ser difícil de controlar (Chen et al., 2018).

En ese sentido, los autores realizaron un diagrama de evolución de las técnicas del fuzzing en las pruebas de seguridad desde el año1990 al 2018.

Ahora bien, La siguiente tabla muestra las ventajas y desventajas de cada técnica identificada en el paso interior:

Figura 19

Artículo 8, Comparación de Técnicas del Fuzzing

System	search strategy	If vulnerabilities specific	Need source code	categories of symbolic executors
MoWF	searching the space where it prunes paths that are exercised by invalid, malformed inputs	YES	NO	online
KLEE	heuristic and random	NO	YES	online
Mayhem	heuristic	NO	NO	hybrid
SZE	heuristic	NO	NO	online
BORG	targeting buffer over-read bugs.	YES	NO	online
Taintscope	targeting checksum parsing code	NO	NO	offline
Dowser	targeting buffer overflow and underflow vulnerabilities	YES	YES	online
SAGE	heuristic	NO	NO	offline
GWF	heuristic	NO	NO	offline
Driller	targeting the point where AFL is unable to identify inputs to search new paths	NO	NO	hybrid

Nota. Adaptado de "A systematic review of fuzzing techniques", por Chen et al., 2018 (https://doi.org/10.1016/j.cose.2018.02.002).

Para comprender las diferencias técnicas descritas por los autores, se utilizaron cuatro aspectos: ejecución simbólica dinámica, la retroalimentación de cobertura, la representación gramatical y el análisis de corrupción. Asimismo, se comparó los sistemas fuzzing en términos de su estrategia de búsqueda, si se enfocan en vulnerabilidades específicas y si se necesitan código fuente o no (Chen et al., 2018).

Principal Resultado:

La eficiencia del fuzzing ha ido mejorando los programas de seguridad durante los últimos 20 años, el presente artículo mostró diferentes técnicas utilizadas, los pros y contras de cada una, si bien es cierto algunas de estas estrategias de fuzzing no son las más eficientes, pero pueden proporcionar un marco para futuros objetivos. Asimismo, con la comprensión cada vez más profunda del proceso de fuzzing y el uso efectivo de las técnicas existentes, las técnicas de fuzzing brindarán un soporte técnico mejorado para posteriores investigaciones (Chen et al., 2018).

4.3.2.2 Format-aware Learn&Fuzz: Deep Test Data Generation for Efficient Fuzzing

Aporte:

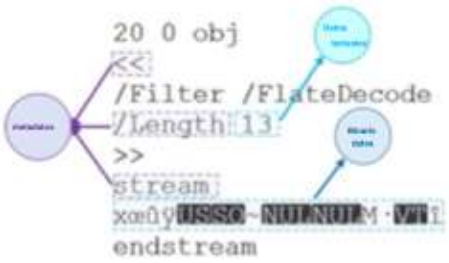
Los autores presentan un modelo de lenguaje neuronal (NLM) basado en redes neuronales recurrentes profundas (RNN) para generar nuevos datos de prueba para fuzzing. Asimismo, finalizan su investigación analizando modelos de aprendizajes más simples.

Proceso

Los autores se refieren a datos de pruebas de estructura compleja a datos textuales y binarios descrito por algunos metadatos. Un gran ejemplo es el formato de tipo PDF. (Nasrabadi et al., 2021) En la siguiente imagen se observa un objeto de estructura compleja y sus diferentes partes:

Figura 20

Artículo 9, Estructura de Entrada Compleja

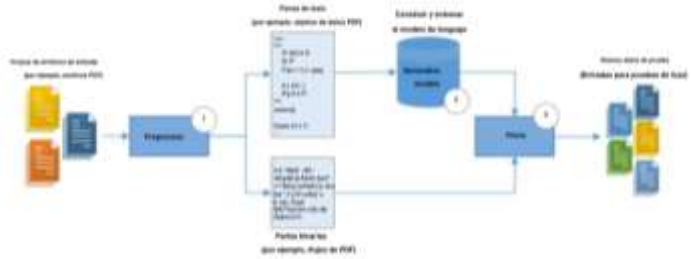


Nota. Información de "Format-aware Learn&Fuzz: Deep Test Data Generation for Efficient Fuzzing", por Nasrabadi et al., 2021 (https://doi.org/10.1007/s00521-020-05039-7).

Para la prueba de datos con estructura compleja, los autores plantean que es necesario un método de generación para comprender a fondo la estructura de entrada y así distinguir los datos de los metadatos, mientras se generan nuevos datos de prueba dirigidos a etapa de análisis o renderizado. Los autores plantearon el siguiente diagrama de flujo para su modelo de lenguaje neuronal:

Figura 21

Artículo 9, Diagrama de Flujo del Modelo de Generación de Datos de Prueba



Nota. Información de "Format-aware Learn&Fuzz: Deep Test Data Generation for Efficient Fuzzing", por Nasrabadi et al., 2021 (https://doi.org/10.1007/s00521-020-05039-7).

Para explicar mejor la Figura 15, los autores recopilan una gran cantidad de archivos de muestra de formatos estructurados (HTML o PDF). Después, dentro de cada archivo se reemplaza con un único token binario (BT) con el objetivo de entrenar a un LM como un conjunto de secuencias ASCII (Nasrabadi et al., 2021).

En la fase de preprocesamiento, también se agrega un token final (ET) al final de cada archivo. Luego, se concatena todos de los archivos juntos para construir una gran secuencia de archivos. Esta secuencia se utiliza para entrenar LM, pero antes de entrenar, se divide en tres conjuntos separados como el conjunto de tren, el conjunto de prueba y el conjunto de validación. Tal división es necesaria para medir la precisión del modelo y la perplejidad en los datos invisibles. Asimismo, ayuda a generar nuevos datos

Finalmente, nuestros dos algoritmos fuzzing recién introducidos se utilizan para generar y fuzz los nuevos datos de prueba, llamados DataNeuralFuzz y MetadataNeuralFuzz. El primero se usa para fuzzear los datos en datos de prueba, y el segundo es se utiliza para confundir el formato o los metadatos del archivo (Nasrabadi et al., 2021).

Principal Resultado:

El principal resultado es la generación de datos de prueba para fuzzing de partes textuales y binarias para estructuras complejas. Asimismo, este resultado aún cuenta con limitaciones, una de ellas fue que aún existen dificultades para eliminar secciones binarias, lo que ocasiona una cobertura de código deficiente.

4.3.2.3 CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing of Deep Learning Systems

Aporte:

Los autores proponen un nuevo enfoque de prueba para sistemas DL (Deep Learning), llamado CAGFuzz (Fuzzing generativo adversario guiado por cobertura) El objetivo del CAGFuzz es maximizar la cobertura de neuronas y generar ejemplos de prueba contradictorios tanto como sea posible con pequeñas perturbaciones para los DNN (Redes Neuronales Profundas) de destino.

Proceso:

CAGFuzz selecciona iterativamente la prueba ejemplos en el grupo de procesamiento y genera los ejemplos contradictorios a través del ejemplo contradictorio pre-entrenado generador (consulte la Sección 3 para obtener detalles) para guiar los sistemas DL a exponer conductas incorrectas. Durante el proceso de generación ejemplos contradictorios, CAGFuzz mantiene ejemplos contradictorios válidos, que pueden proporcionar una cierta mejora en las neuronas cobertura para el procesamiento difuso posterior, y limitar las pequeñas perturbaciones invisibles a los ojos humanos, asegurando el mismo significado entre el ejemplo original y el ejemplo contradictorio. Las contribuciones de este trabajo incluyen los siguientes tres aspectos:

Se diseñó un generador de ejemplos adversos, AEG, que puede generar ejemplos adversos con pequeñas perturbaciones a partir de conjuntos de datos generales. El objetivo de Cycle GAN es transformar la imagen A en la imagen B con diferentes estilos. Basado en CycleGAN, nuestro objetivo es transformar la imagen B en la imagen A, y obtener la imagen A' similar a la imagen original A. En consecuencia, combinamos dos generadores con funciones opuestas de CycleGAN como nuestro generador de ejemplos adversos. Los ejemplos adversos generados por AEG pueden añadir pequeñas perturbaciones invisibles para los ojos humanos a los a los ejemplos originales. AEG se entrena a partir de conjuntos de datos generales de datos generales y no se basa en ningún modelo DNN específico, por lo que su capacidad de generalización es mayor que la de los que los enfoques más avanzados. Además, debido a la lógica de restricción inherente de CycleGAN, el AEG entrenado AEG no sólo tiene una alta eficiencia en la generación de ejemplos de ejemplos publicitarios, sino que también puede mejorar eficazmente la robustez de los sistemas DL. La robustez de los sistemas DL

Extraemos las características profundas del ejemplo original y del ejemplo adversario, y los hacemos tan similares como sea posible mediante la medición de la similitud. Utilizamos la red VGG-19 para extraer la información semántica profunda del ejemplo original y del ejemplo adversario, y utilizamos el método de medición de la similitud del coseno para garantizar que la información semántica profunda del ejemplo adversario es coherente con el original.

Principal Resultado:

En el artículo se presentan los experimentos con varios modelos DNN populares con un rendimiento competitivo para cada conjunto de datos. Estos modelos DNN han demostrado ser estándar en los experimentos de investigadores anteriores. Por el lado de los autores, siguen de cerca las prácticas y directrices comunes de entrenamiento de aprendizaje de máquinas, y establecemos la tasa de aprendizaje para el entrenamiento del modelo DNN. Durante el proceso de inicialización de la tasa de aprendizaje del modelo DNN, si la tasa de aprendizaje es demasiado alta, el peso del modelo aumentará rápidamente, lo que tendrá un impacto negativo en el entrenamiento de todo el modelo. Para los tres modelos LeNet del conjunto de datos MNIST, fijamos la tasa de aprendizaje en 0,05.

4.3.2.4 Coverage-guided Learning-assisted Grammar-based Fuzzing

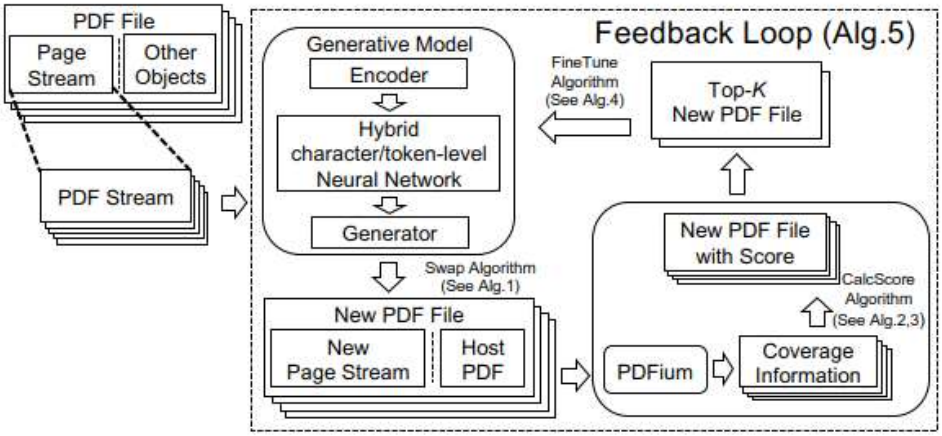
Aporte: Los autores proponen un conjunto de técnicas para mejorar el fuzzing basado en gramáticas de aprendizaje a través de un modelo generativo para las secuencias de instrucciones mediante el entrenamiento de una red neuronal recursiva híbrida a nivel de caracteres y fichas. Asimismo, muestran las métricas de cobertura recogidas durante ejecuciones anteriores de fuzzing para refine (o fine-tune) eficazmente el modelo aprendido de modo que pueda realizar nuevas entradas inductoras de alta cobertura.

Proceso:

El artículo propone un fuzzing basado en gramáticas y guiado por la cobertura bajo el contexto del fuzzing de un analizador PDF. Los autores indican que el mecanismo clave de nuestro enfoque es el bucle de retroalimentación, que explota la información de cobertura recopilada durante las ejecuciones anteriores del fuzzing para afinar el modelo ya aprendido de forma que pueda generar nuevas entradas interesantes (es decir, nuevas entradas). (es decir, nuevas secuencias de instrucciones de flujos de páginas) que puedan inducir una mejor cobertura del código.

Figura 22

Artículo 11, Descripción General del Fuzzing Basado en Gramática Asistido por Aprendizaje Guiado por Cobertura



Nota. Información de "Coverage-guided Learning-assisted Grammar-based Fuzzing", por Jitsunari et al., 2019 (https://doi.org/10.1109/ICSTW.2019.00065).

En primer lugar, se genera un modelo generativo para nuevas entradas (archivos PDF) mediante el entrenamiento de una red neuronal recurrente (RNN, por sus siglas en inglés) utilizando un gran corpus de archivos PDF recogidos de Internet. Los autores indican que extrajeron alrededor de 1.988 secuencias de páginas de los 261 archivos PDF recopilados. Con estos flujos de páginas, se entrenó una RNN para aprender un modelo generativo de nuevas páginas que se incorporan a los nuevos archivos PDF.

El modelo generativo se compone de tres partes: codificador, red neuronal híbrida a nivel de char/token y generador.

- Codificador: Para aprender un modelo generativo para secuencias de instrucciones bien formadas de flujos de páginas, utilizamos un modelo híbrido char/token-RNN. El modelo char/token-RNN nos permite evitar la generación de instrucciones mal formadas con opcode inválidos, al tiempo que permite la generación de operandos de instrucción interesantes. En nuestro modelo, tratamos los opcodes de instrucciones de más de dos caracteres como tokens individuales, mientras que el resto (incluidos los operandos) se codifica a nivel de caracteres.
- Red neuronal híbrida a nivel de carácter/token: Nuestra implementación del modelo generativo es una LSTM con dos capas ocultas, cada una de las cuales consta de 128 nodos ocultos (o estados). Entrenamos el modelo con Adam durante 10 épocas, con una tasa de aprendizaje de 0,001.
- Generador. Se genera nuevas secuencias de páginas que se incrustarán en nuevos archivos PDF (que posteriormente se introducirán en el analizador de PDF). El

modelo aprendido representa una distribución de probabilidad, que toma como entrada una secuencia de caracteres y predice y produce el siguiente carácter con su probabilidad. Para la generación de nuevas secuencias de páginas, utilizamos el algoritmo SampleFuzz de Learn&Fuzz.

Principal Resultado:

Los autores para confirmar la eficacia de nuestro fuzzing basado en gramáticas asistidas por el aprendizaje y la cobertura, realizaron experimentos comparativos con PDFium, un analizador sintáctico de PDF real. El objetivo de los experimentos es medir hasta qué punto el enfoque planteado es capaz de lograr una mejor cobertura de código que los fuzzers basados en gramática asistida Learn&Fuzz y DeepSmith. Los experimentos se realizaron en un entorno de hardware/software.

Los resultados muestran que tanto la cobertura de líneas como la de funciones logradas por el modelo tienden a aumentar a medida que lo hace el número de iteraciones de ajuste fino. Esto significa que, en nuestros experimentos, nuestro ajuste final guiado por la cobertura del modelo ya aprendido es eficaz para mejorar la capacidad del modelo de generar aquellas nuevas secuencias de instrucciones que probablemente induzcan una mejor cobertura de código del analizador sintáctico objetivo.

4.3.2.5 A Generative Model for Evasion Attacks in Smart Grid

Aporte: El enfoque de este documento se limita al estudio de los ataques de evasión, donde el objetivo de un adversario suele ser evitar la identificación de la verdadera clase a la que pertenece una entrada. Esto hace que el método de aprendizaje sea poco efectivo al obligarlo a funcionar aleatoriamente con resultados inexactos. Por ejemplo, si el objetivo de un método de ML es detectar un ataque, y un ejemplo adversario para la evasión sería elaborar una estrategia de ataque, de modo que cuando el modelo de ML confronta este ejemplo, la entrada del ataque se infiere como benigna con una alta probabilidad.

Proceso:

En esta sección los autores proponen la estrategia generativa para crear ejemplos contradictorios. Los modelos generativos se ocupan de cómo se generan los datos dado un modelo de clasificación que produce un determinado resultado. Observamos que utiliza un enfoque discriminativo para la clasificación. Por lo tanto, nuestro esfuerzo es generar las

distribuciones de datos de evasión utilizando el enfoque generativo que intenta escapar del clasificador discriminativo sin sacrificar el impacto operativo del ataque original

Descripción general de la solución: A partir del clasificador gaussiano plegado, los autores saben que la puntuación de un dispositivo de medidor inteligente será más alta si más y más lecturas de medidores inteligentes se acercan más a la media de consumo de energía de la población. Con este conocimiento, la estrategia de un adversario sería generar una estrategia de ataque que mantenga las lecturas cambiadas más cerca de la media de la población temporal mientras sigue cambiando los datos reales. Tal proximidad de los datos perturbados hacia la media de la población aumentaría el puntaje de confianza y potencialmente aumentaría el error de clasificación mientras se preserva la restricción ÿavg requerida para el impacto operativo.

Adicionalmente, existe una similitud de diseño subyacente de este enfoque con DBSCAN y confianza a distancia KL, aunque las matemáticas reales de cada enfoque son diferentes. La similitud está en la discretización de los datos en niveles discretos y luego en el uso de la densidad de probabilidad de cada nivel para la estimación de puntajes que los combinan en un enfoque de agrupación. Esto es lo que el método contradictorio busca aprovechar y otorga el poder de la transferibilidad al clasificador de confianza basado en la distancia DBSCAN y KL. Si bien es posible que no eluda por completo el clasificador, el aumento en el puntaje de confianza degradará la precisión de la clasificación del método gaussiano plegado, lo que tendrá un efecto significativo en los sistemas IoT de vida inteligente a gran escala como AMI

Principal Resultado:

En este artículo, se ha presentado los impactos de los ataques de evasión en redes inteligentes. Utilizamos el modelo generativo para crear las muestras de evasión óptimas. Finalmente, se demuestra el rendimiento de la solución propuesta utilizando datos reales de medición inteligente de Texas. Los resultados muestran que la estrategia de evasión de datos falsificados creada con nuestro generador utilizando el conocimiento del clasificador Folded Gaussian Trust tiene un impacto negativo en la precisión de clasificación de la detección de medidores comprometidos.

Además, se descubrió que la estrategia de evasión demostró ser transferible frente a otros enfoques como DB-SCAN y clasificador de distancia KL para la detección de medidores comprometidos.

4.3.2.6 Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0

Aporte:

Los autores nos indican que los algoritmos de ML pueden utilizarse para resolver varios problemas con los enormes datos disponibles generados por las industrias, por lo que el ML se ha utilizado ampliamente en la informática y otras áreas, como la PdM del sistema de fabricación, la herramienta o la máquina, y es una de las posibles áreas de uso de los métodos basados en datos (Red Neural Artificial (ANN), Aprendizaje por Refuerzo (RF), Máquina de Vectores de Apoyo (SVM), Regresión Logística (LR) y Árbol de Decisión (DT)). Recientemente, las técnicas de ML se han aplicado ampliamente en varios campos de estudio. La selección de la más apropiada, sencilla y efficiente podría ser una gran preocupación. Los algoritmos de ML suelen requerir la recopilación de enormes cantidades de datos de los escenarios del estado de fallo y de las condiciones de salud para el entrenamiento del modelo. Estos algoritmos que requieren principalmente grandes cantidades de datos incluyen el Modelo de Espacio Vectorial (VSM), LR, DT y RF. El desarrollo de algoritmos de ML abarca la selección de datos históricos, el preprocesamiento de datos, la selección de modelos, el entrenamiento de modelos, la validación de modelos y el mantenimiento. Los pasos implicados en el desarrollo de algoritmos de ML pueden especificarse como entrada, extracción y selección de características, características, técnicas tradicionales de ML y salida.

Proceso

En la publicación, detalla algunas técnicas de Machine learning, tales como:

- Red Neural Artificial (RNA) De hecho, la RNA se desarrolla a partir del tema de la biología, donde la Red Neural (NN) juega un papel significativo en el cerebro humano. La RNA es una técnica computacional inteligente que se ha inspirado en las neuronas biológicas. Es un sistema de computación masivamente paralelo que consiste en un número extremadamente grande de procesadores simples con muchas interconexiones. En lugar de seguir las leyes especificadas por los expertos humanos, las RNA aprenden las leyes básicas a partir del conjunto de situaciones simbólicas dadas en los ejemplos. situaciones simbólicas dadas en los ejemplos. Se organizan en tres capas o más (es decir, capa de entrada, varias capas ocultas y una capa de

salida). y una capa de salida). Además, la actividad analítica de estas RNA se deriva de las relaciones entre las unidades de procesamiento de la red.

- Máquina de vectores de apoyo (SVM)

  La SVM es una conocida técnica de ML que se utiliza ampliamente tanto para la clasificación como para el análisis de regresión debido a su alta precisión. La SVM se define como un concepto de aprendizaje estadístico con un método de aprendizaje computacional adaptativo. SVM emplea vectores de entrada para mapear de forma no lineal en un espacio de características cuya dimensión es alta.

- Árbol de decisión (DT)

  El árbol de decisión es un sistema de red compuesto principalmente por nodos y ramas, y los nodos que comprenden nodos raíz y nodos intermedios. Los nodos intermedios se utilizan para representar una característica, y los nodos hoja se utilizan para representar una etiqueta de clase. DT puede utilizarse para la selección de características. Los clasificadores DT han ganado una considerable popularidad en varias áreas, como la identificación de caracteres, el diagnóstico médico y el reconocimiento de voz. identificación de caracteres, el diagnóstico médico y el reconocimiento de voz. Más concretamente, el modelo DT tiene el potencial para descomponer un complicado mecanismo de toma de decisiones

- Bosque aleatorio (RF)

  RF fue desarrollado por Breiman, L. Se trata de un algoritmo de aprendizaje de conjunto formado por varios clasificadores DT, y la categoría de salida se determina colectivamente por estos árboles individuales. Cuando el número de árboles en el bosque aumenta, el error de generalización del bosque converge

Principal Resultado:

Según el artículo el SVM, RF, y ANN, son los algoritmos de ML más utilizados en la literatura revisada. Han sido aplicados con éxito en varios campos de aplicaciones de PdM. Algunos autores se centraron en los algoritmos ML de RNA. Otros autores estudiaron la técnica RF. En los últimos años, el uso de la técnica SVM ha recibido la atención de los autores. Algunos autores han prestado atención a la técnica GBM. Algunos autores han realizado estudios utilizando la técnica DT. Sin embargo, se observa que se ha prestado

menos atención a la técnica XGBoost y sólo se han encontrado unos pocos estudios en la literatura.

4.3.3 Artículos de la categoría Machine Learning

4.3.3.1 Detecting Spam Email with Machine Learning Optimized with Bio-Inspired Metaheuristic Algorithms

Aporte

En el siguiente artículo, los autores presentan la implementación de un modelo de machine learning para identificar el spam en correos electrónicos, inspirándose en diferentes técnicas y realizando una comparación de sus resultados con otros modelos actuales de machine learning.

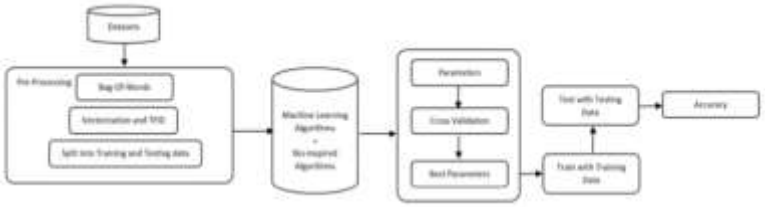
Proceso

Hoy en día, la mayoría de las personas acostumbra a usar correos electrónicos para comunicarse y socializarse. Sin embargo, la existencia de malwares, como el fishing, vulneran la privacidad e información de los usuarios. Esta es la principal motivación de los autores para introducir una herramienta para identificar el spam entre los correos electrónicos (Gibson et al., 2020).

En primer lugar, los autores realizaron una investigación exhaustiva para obtener conocimiento sobre diferentes técnicas de machine learning: Naive Bayes, Support Vector Machine, Random Forest, Decision Tree y MultiLayer Perceptron, y algunas herramientas como Weka, Scikit Learn, Keras y Tensor Flow con el objetivo de utilizar dicha información para construir un nuevo modelo de aprendizaje automático. A continuación, se presenta un esquema diseñado por los autores:

Figura 23

Artículo 14, Diagrama de Bloques de Detección de Spam



Nota. Adaptado de "Detecting Spam Email with Machine Learning Optimized with Bio-Inspired Metaheuristic Algorithms", por Gibson et al., 2020 (https://doi.org/10.1109/ACCESS.2020.3030751).

La imagen tiene como propositó mostrar el camino recorrido por todas las técnicas de machine learning y el modelo generado, con el objetivo de obtener la mayor precisión posible de cada una y comparar el resultado con diferentes trabajos de investigación para determinar la mejor técnica de identificación de spam.

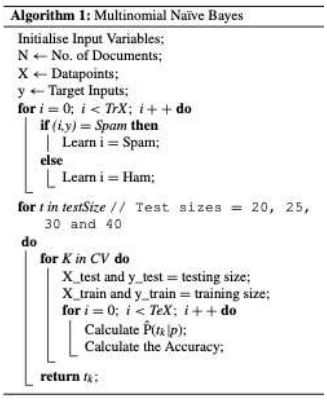
En segundo lugar, el artículo describe a detalle cada algoritmo utilizado por cada técnica, pero a continuación se procederá a mostrar el proceso del algoritmo que tuvo una mayor precisión para detectar correos electrónicos fraudulentos.

Naive Bayes - Multinomial

Este modelo se utiliza para resolver problemas de clasificación utilizando técnicas de probabilidad. Se utilizó el algoritmo multinomial de la técnica Naive Bayes, debido a que, está relacionado con el texto y es la mejor opción para realizar la identificación de correo no deseado (Gibson et al., 2020).

Figura 24

Artículo 14, Algoritmo Multinomial – Naive Bayes



Algorithm 1: Multinomial Naïve Bayes

Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
for $i = 0;\ i < TrX;\ i++$ **do**
  if $(i,y) = Spam$ **then**
    Learn i = Spam;
  else
    Learn i = Ham;
for *t in testSize* // Test sizes = 20, 25, 30 and 40
do
  for *K in CV* **do**
    X_test and y_test = testing size;
    X_train and y_train = training size;
    for $i = 0;\ i < TeX;\ i++$ **do**
      Calculate $\hat{P}(t_k|p)$;
      Calculate the Accuracy;
  return t_k;

Nota. Adaptado de "Detecting Spam Email with Machine Learning Optimized with Bio-Inspired Metaheuristic Algorithms", por Gibson et al., 2020 (https://doi.org/10.1109/ACCESS.2020.3030751).

Para explicar mejor el contenido del algoritmo, primero los autores plantearon el uso de datasets para la identificación de los correos electrónicos no deseados. Luego utilizaron un

clasificador de puntos de datos y entradas. Finalmente, el algoritmo realiza el cálculo de la **precisión** de las pruebas realizadas (Gibson et al., 2020).

Finalmente, los autores utilizaron el siguiente algoritmo de optimización bio-inspiradores para mejorar los resultados de los experimentos.

Uno de ellos es el algoritmo de optimización de enjambre de partículas (PSO), este método en particular utiliza una función objetiva y un número de iteraciones para ejecutar el PSO y proporcionar el mejor costo global y la mejor posición global (Gibson et al., 2020).

A continuación, el pseudocódigo de algoritmo:

Figura 25

Artículo 14, Algoritmo PSO

Nota. Adaptado de "Detecting Spam Email with Machine Learning Optimized with Bio-Inspired Metaheuristic Algorithms", por Gibson et al., 2020 (https://doi.org/10.1109/ACCESS.2020.3030751).

Principal Resultado

La propuesta inicial de los autores se implementó con éxito, se utilizaron diferentes modelos combinados con algoritmos bio-inspirados en técnicas de machine learning. El modelo de aprendizaje se probó en aproximadamente 50 000 correos electrónicos y se demostró que el algoritmo de Naive Bayes fue el mejor algoritmo para la detección de spam. Asimismo, la

investigación brindó un análisis profundo sobre la exploración de las redes neuronales de machine learning (PSO y GA) (Gibson et al., 2020).

4.3.3.2 More Effective Test Case Generation with Multiple Tribes of AI

Aporte

En el siguiente artículo, el autor introduce una nueva técnica para la generación de casos de pruebas de forma automatizada, con el objetivo de reducir el tiempo de construcción y ejecución de los casos de una suite de pruebas.

Proceso

Inicialmente el autor menciona que ya ha sido demostrado que generar datos de entrada automáticamente, no suele ser del todo estructurada y puede ser difícil de leer e interpretar. Por otro lado, el fuzzing basado en gramática si es efectivo para generar datos de entrada estructurados basados en una gramática especificada por el usuario (Olsthoorn, 2022).

Para la generación de los datos de entrada, el usuario propuso inyectar datos en formatos de tipo **JSON** con cierta probabilidad. Asimismo, para la creación y el desarrollo de las estructuras de los casos de prueba se usó la técnica EA (Fuzzing basado en gramática) (Olsthoorn, 2022).

Por otro lado, el usuario realizó un estudio empírico para evaluar la eficacia y viabilidad del enfoque mencionado. Para ello, se implementó el software EvoSuite (Herramienta de TCD de última generación para java) y se evaluó 20 clases de los tres canalizadores de JSON más populares (Gson, fastjson y org.json); con el objetivo de evaluar si el enfoque tiene un impacto negativo en el rendimiento de las clases.

Principal Resultado

Teniendo en cuenta que el artículo descrito no mostró a detalle el desarrollo de la solución, el autor menciona que el enfoque logra una mejor cobertura de pruebas del 15% más a la línea base (Este estudio se validó con el software EvoSuite sin fuzzing). Asimismo, el estudio recalca que no es suficiente el proceso de inyección de cadenas **Json** como datos de entrada en la población inicial, para incrementar la cobertura es necesario tener mayor precisión (Olsthoorn, 2022).

4.3.3.3 Automatic Generation of Syntax Valid C Programs for Fuzz Testing

Aporte

En este artículo, se propone una herramienta de fuzzing utilizando el algoritmo DeepFuzz basado en gramática para compiladores en lenguaje C. Asimismo, los autores realizan un análisis del rendimiento de DeepFuzz, así como también la verificación de la eficacia de las pruebas en compiladores y su mejora en la cobertura de casos de prueba (Liu et al., 2019).

Proceso

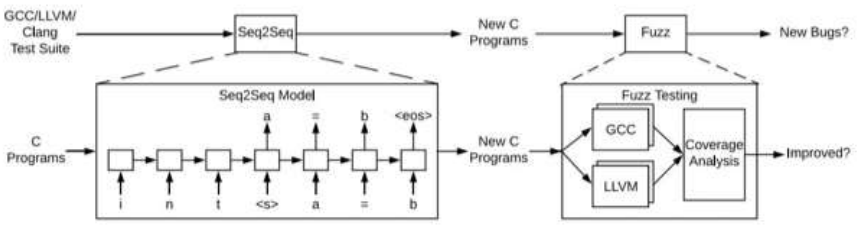
Por un lado, el compilador analizado por los autores es llamado **GCC**, es un software creado en 1987 y ha tenido errores desde su fecha de lanzamiento. Por otro lado, los autores seleccionaron la técnica de fuzzing utilizando una red neuronal recurrente regenerativa como "gramática", con el objetivo de usarlas como reglas para la generación de casos de prueba para detectar fallas (Liu et al., 2019).

DeepFuzz

Está técnica o herramienta tiene como objetivo la implementación de dos redes neuronales recurrentes (RNN) para la predicción de secuencias a nivel de caracteres (Liu et al., 2019).

Figura 26

Artículo 16, Flujo de Trabajo de DeepFuzz



Nota. Adaptado de "Automatic Generation of Syntax Valid C Programs for Fuzz Testing", por Liu et al., 2019 (https://doi.org/10.1609/aaai.v33i01.33011044).

Para la generación de la herramienta se planteó el esquema mostrado, el cual maneja una serie de iteraciones (secuencias) para obtener los casos de prueba. En la primera etapa, se entrenó un modelo generativo con datos recopilados en la suite de pruebas de la aplicación GCC. En la segunda etapa se evaluó un previo procesamiento para evitar información no relevante en las secuencias. Luego, los autores alimentaron los programas C generados

sintácticamente al compilador de GCC, para finalmente verificar los errores encontrados, así como también la cobertura total de las pruebas realizadas (Liu et al., 2019).

Para la generación de las pruebas, se registró inicialmente la información de la cobertura del sistema (Cantidad de líneas, funciones y sucursales) más el conjunto de pruebas generado por la herramienta DeepFuzz, con el objetivo de contrastar métricas y generar métodos de muestreos futuros (Liu et al., 2019).

Finalmente, uno de los errores encontrados por la herramienta DeepFuzz es el siguiente:

Bug - Argumento mal definido

El siguiente fragmento de código contiene una función que debería haber sido un puntero y no de tipo indefinido:

Figura 27

Artículo 16, Bug Argumento Mal Definido

```
double f () {
  double r;
  asm ("mov %S1,%S0; mov %R1,%R0" : "=r" (r) : "i" (20));
  asm ("mov %S1,%S0; mov %R1,%R0" : "+r" (r) : "i" (20.));
  __atomic_load_n ((enum E *) 0, 0);
  ;
  return r;
}
```

Nota. Adaptado de "Automatic Generation of Syntax Valid C Programs for Fuzz Testing", por Liu et al., 2019 (https://doi.org/10.1609/aaai.v33i01.33011044).

Principal Resultado

El desarrollo de la herramienta DeepFuzz tuvo contribuciones significantes con la aplicación GCC, debido al marco de trabajo utilizado, ya que, el proceso de crear un modelo sequence-to-sequence utilizando una red neuronal generativa proporcionó una mejora en la cobertura de las pruebas y del mismo modo se detectaron e informaron 8 errores en GCC. Cabe señalar que la herramienta también cuenta con algunas limitaciones, una de ellas es la insuficiencia de patrones para entrenar a las redes neuronales (Liu et al., 2019).

4.3.3.4 A systematic review of fuzzing based on machine learning techniques

Aporte:

El presente artículo brinda una investigación detallada sobre las diferentes técnicas de aprendizaje automático (Machine Learning) que se han utilizado para el fuzzing durante los últimos años. Asimismo, los autores analizan como estás técnicas mejoran el proceso y los resultados de la detección de vulnerabilidades, con el fin de generar referencias para futuras investigaciones, a través del análisis y resumen de múltiples dimensiones.

Proceso

Para iniciar con la revisión sistemática los autores plantearon las siguientes preguntas de investigación:

Tabla 10

Artículo 17 – Preguntas de Investigación

N.º	Pregunta de investigación
1	¿Por qué se pueden utilizar técnicas de aprendizaje automático para fuzzing?
2	¿Qué pasos del fuzzing han utilizado técnicas de aprendizaje automático?
3	¿Qué algoritmos de aprendizaje automático se han utilizado para fuzzing?
4	¿Qué técnicas se utilizan para el preprocesamiento de datos de fuzzing basado en aprendizaje automático?
5	¿Qué conjuntos de datos se usan para entrenar y evaluar?
6	¿Qué medidas de desempeño se utilizan para evaluar los resultados?
7	¿Cómo configurar los hiperparámetros de la aritmética de aprendizaje automático?
8	¿Cuál es la capacidad de los fuzzers basados en técnicas de aprendizaje automático para descubrir vulnerabilidades?
9	¿Cuál es la capacidad de los fuzzers basados en técnicas de aprendizaje automático para descubrir vulnerabilidades?

Nota. Información adaptada de "A systematic review of fuzzing based on machine learning techniques", por Wang et al., 2020 (https://doi.org/10.1371/journal.pone.0237749).

Están preguntas ayudaron a generar los criterios de investigación del estudio, y posteriormente a la recolección de artículos que involucran el aprendizaje automático en el fuzzing. Asimismo, se presenta el siguiente flujo de trabajo del fuzzing: generación de casos

de prueba, ejecución del programa, monitoreo del estado del tiempo de ejecución y el análisis de fallas (Wang et al., 2020).

Tras identificar los criterios de búsqueda, los autores recopilaron 44 artículos para su revisión sistemática e identificaron el siguiente enfoque:

Tabla 11

Preguntas de Investigación

Categoría	Objetivo / Técnicas
Generación de archivos	PCFG (gramática probabilística sensible al contexto, que contiene reglas semánticas y características gramaticales), GAN (Generative Antagonistic Network), RLS (algoritmo de mínimos cuadrados recursivos)
Generación de casos de prueba	Se realizan a través de mutaciones en archivos semillas, o al construirse por una función con data inputs conocidos. Augmented-AFL, Modelos NN
Filtrado de casos de prueba	Se selecciona la entrada de prueba que es más probable que active nuevas rutas o vulnerabilidades.
Selección del operador de mutación	Todos los cambios que puedan ocurrir en los casos de prueba generados tras los resultados del algoritmo genético. Se evita el desperdicio de recursos de prueba
Análisis de explotabilidad	Se analiza la posibilidad de que un atacante explote alguna vulnerabilidad

Nota. Información adaptada de "A systematic review of fuzzing based on machine learning techniques", por Wang et al., 2020 (https://doi.org/10.1371/journal.pone.0237749).

Finalmente, los autores realizan una evaluación del rendimiento en la detección de vulnerabilidades. Realizando una comparación de las técnicas recopiladas para posteriormente llevarlos a un cuadro de comparación y analizar el % de la cobertura realizada. Se mencionan las siguientes categorías de cobertura: cobertura de bloque básico, cobertura de línea, cobertura relativa, cobertura de instrucción y cobertura de sucursales.

Principal Resultado:

La revisión sistemática de las diferentes técnicas de aprendizaje automático, muestran que actualmente existen herramientas para detectar vulnerabilidades y mejorar la cobertura en el fuzzing. Asimismo, los puntos de vista mostrados pueden usarse como punto de referencia con el campo actual.

4.3.3.5 Machine Learning Augmented Fuzzing

Aporte: El siguiente artículo científico realiza tres aportes para el uso del machine learning en el SBST (Search-Based Software Testing). En primer lugar, el autor muestra un análisis con machine learning basado en las tazas de ejecución para definir una estrategia de prueba dirigida. En segundo lugar, se muestra cómo el análisis de trazas basado en el aprendizaje automático puede utilizarse para construir un paisaje universal de la fitness. Finalmente, se muestra la validez de este paisaje como función de fitness; asimismo, implementa un aumento generativo basado en ML para que un fuzzer para realizar una búsqueda dirigida.

Proceso:

El artículo científico aborda dos problemáticas en SBST. El primero es el problema de los paisajes de búsqueda no lineales con mesetas. El segundo problema, es el hecho de que SBST generalmente se basa en criterios de cobertura, en lugar de centrarse en una propiedad o comportamiento específico. Para ello, se harán uso de técnicas modernas de Machine Learning (ML), las cuales constan de redes neuronales convolucionales (CNN), redes neuronales recurrentes (RNN), autocodificadores (AE) y redes neuronales generativas. El proceso para el desarrollo de la propuesta se lleva a cabo en tres partes:

1. Prueba de fuzz de selección de propiedades:

   La primera parte de la tesis muestra un enfoque en el que la aptitud de una herramienta SBST impulsada por la diversidad se reemplaza por una aptitud específica generada por ML. El método se basa en un modelo de red neuronal entrenado en ejecuciones de programas etiquetadas con fallas y no fallas. Los resultados sugieren que guiar el proceso de prueba utilizando la predicción del modelo de probabilidad de choque mejora la eficiencia del descubrimiento de choques. El enfoque es un ejemplo novedoso del uso de una herramienta ML para interpretar el comportamiento del programa y guiar un método SBST hacia un comportamiento específico.

2. Espacios latentes como paisajes de búsqueda:

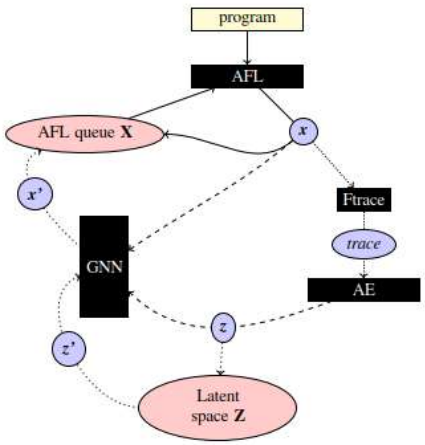
   La segunda parte de la tesis investiga el uso de un espacio latente como base potencial para una función de aptitud en SBST. El propósito de este trabajo es crear un paisaje de búsqueda fluido a partir de datos discretos, lo que proporcionaría una idoneidad para una propiedad de interés. El segundo resultado muestra que el bloqueo de la red neuronal la estimación de probabilidad se puede utilizar eficazmente como base para una función de aptitud específica. El efecto de la estrategia específica se evaluó midiendo la cantidad de bloqueos descubiertos, la cantidad de bloqueos únicos descubiertos y la cantidad total de rutas encontradas. El fuzzing se ejecutó para un número fijo de ejecuciones, en una submuestra de programas del conjunto de datos de prueba, y se comparó con los resultados del fuzzer de referencia sin modificar. Se dio una puntuación de rendimiento basada en el área bajo la curva para cada medida. El propósito de acumular los valores de esta manera es recompensar tanto la cantidad de bloqueos (o caminos) encontrados como encontrarlos antes.

3. Aumento de SBST con un modelo generativo:

   La tercera parte de la tesis explorará el uso de un modelo generativo junto con un fuzzer, utilizando la aptitud espacial latente de la Parte 2. Se propone una versión preliminar dónde se modela una arquitectura para la generación de imágenes o secuencias.

Figura 28

Artículo 18, Framework Experimental



Nota. Adapado de "Machine Learning Augmented Fuzzing", por Joffe, 2018 (https://doi.org/10.1109/ISSREW.2018.000-1).

Principal Resultado:

El autor del artículo relata los resultados para cada parte de su proceso. Para la primera es que los rastros de ejecución de las llamadas a la biblioteca C estándar son un fuerte predictor de fallas. El segundo resultado muestra que el bloqueo de la red neuronal la estimación de probabilidad se puede utilizar eficazmente como base para una función de aptitud específica. Finalmente, el fuzzing se ejecutó para un número fijo de ejecuciones, en una submuestra de programas del conjunto de datos de prueba, y se comparó con los resultados del fuzzer de referencia sin modificar. Se dio una puntuación de rendimiento basada en el área bajo la curva para cada medida. El propósito de acumular los valores de esta manera es recompensar tanto la cantidad de bloqueos (o caminos) encontrados como encontrarlos antes. Una estrategia dirigida superó una línea base en 80.1%, 63.5% y 61.5% de los casos considerando accidentes totales, únicos bloqueos y caminos descubiertos respectivamente (She et al., 2020).

4.3.3.6 MTFuzz: Fuzzing with a Multi-task Neural Network

Aporte: En el artículo científico, los autores abordan las problemáticas mediante el aprendizaje multitarea, un paradigma de aprendizaje popular utilizada para aprender de manera efectiva características comunes compartidas en tareas relacionadas a partir de datos de entrenamiento limitados. En este marco, diferentes tareas participantes permiten que un modelo de Machine Learning aprenda de manera efectiva una representación de características compacta y más generalizada mientras ignora los ruidos específicos de la tarea. Para aprender conjuntamente una incrustación compacta de las entradas, en nuestro entorno, usamos diferentes tareas para predecir la relación entre las entradas del programa y diferentes aspectos del comportamiento del programa relacionado con fuzzing (por ejemplo, diferentes tipos de cobertura de borde). Tal arquitectura aborda tanto la escasez de datos como el problema de falta de diversidad. El modelo puede aprender simultáneamente de diversos comportamientos de programa de diferentes tareas, así como enfocarse en aprender las características importantes (bytes calientes en nuestro caso) en todas las tareas (She et al., 2020).

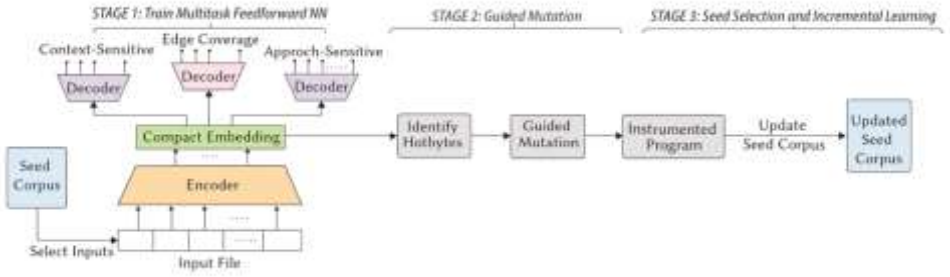
Proceso:

MTFuzz que tiene como objetivo maximizar la cobertura perimetral con la ayuda de dos medidas de cobertura adicionales: cobertura perimetral sensible al contexto y cobertura perimetral sensible al enfoque mediante aprendizaje multitarea.

La primera etapa entrena una MTNN para producir una incrustación compacta de un espacio de entrada escaso mientras conserva la información sobre los bytes activos, es decir, los bytes de entrada tienen la mayor probabilidad de afectar la cobertura del código. La segunda etapa identifica estos bytes calientes y se enfoca en mutarlos. Finalmente, en la tercera etapa, el corpus semilla se actualiza con las entradas mutadas y retiene solo las entradas nuevas más interesantes (She et al., 2020).

Figura 29

Artículo 10, Descripción General de MTFuzz



Nota. Información de "MTFuzz: Fuzzing with a Multi-task Neural Network", por She et al., 2020 (https://doi.org/10.1145/3368089.3409723).

Principal Resultado:

En el artículo los autores presentan MTFuzz, un marco de fuzzing de red neuronal multitarea. MTFuzz aprende de múltiples medidas de cobertura de código para reducir un espacio de entrada disperso y de gran dimensión a una representación compacta. Esta representación compacta se usa para guiar al fuzzer hacia regiones inexploradas del código fuente. Además, esta representación compacta se puede transferir entre programas que operan en el mismo formato de entrada. Los hallazgos sugieren que MTFuzz puede mejorar significativamente la cobertura perimetral mientras descubre varios errores nunca vistos.

4.3.3.7 Fuzzing-base d hard-lab el black-box attacks against machine learning models

Aporte:

Los autores buscan evaluar proyectos de caja negra basados en fuzzing y se aprovecha la idea básica de las pruebas de fuzzing para explorar el espacio de ejemplos adversarios. mediante la realización iterativa de selección de semillas, mutación, nueva evaluación de ejemplo, y registro de observación. El principal desafío de aplicar pruebas de fuzz a la generación de ejemplos contradictorios está explorando el gigantesco espacio de búsqueda. Teóricamente, un fuzzer es capaz de explorar todo el espacio de búsqueda, pero esto es computacionalmente inviable Los autores abordan este desafío utilizando una imagen de guía y ciertas estrategias de orientación para mejorar la eficiencia del ataque, es decir, solo usando un pequeño número de consultas. Una imagen de guía es una imagen limpia de una clase de destino específica para realizar ataques dirigidos, o cualquier imagen que no sea de la clase de imagen de origen para realizar ataques no dirigidos. Las estrategias de orientación controlan cómo son los pasos tomados para caminar desde una imagen de origen a una imagen de guía. La imagen de guía y las estrategias de guía juntas reducen la confusión (Qin & Yue, 2022).

Proceso:

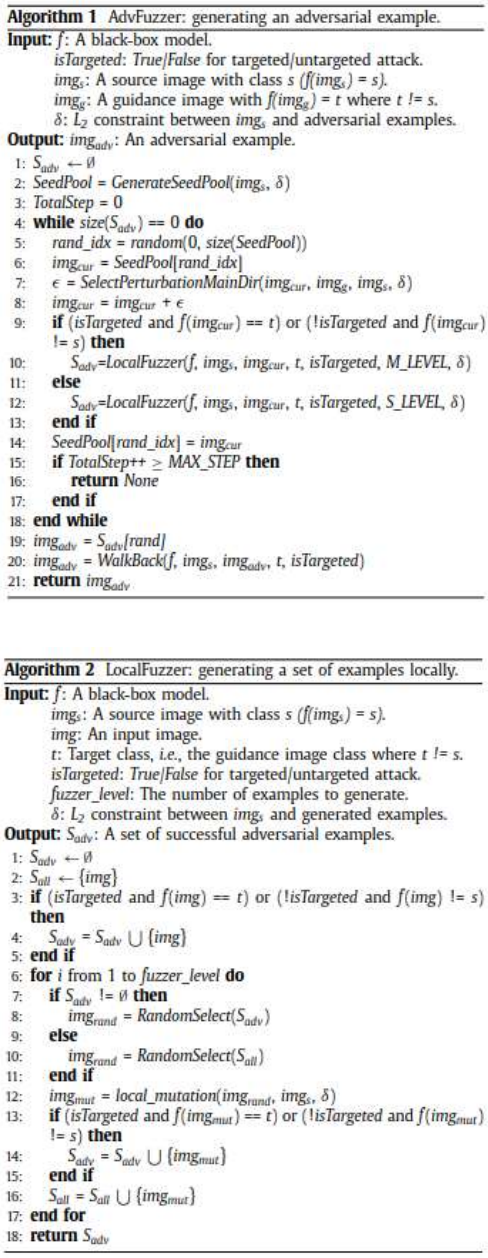
Se diseñó dos fuzzers: un fuzzer adversario (denominado AdvFuzzer) y un fuzzer local (denominado LocalFuzzer). Partiendo de una imagen fuente que está limpia, AdvFuzzer se aleja de ella lentamente (por lo tanto, con menos cantidad de perturbaciones) paso a paso seleccionando aleatoriamente un camino hacia una imagen guía. Con un grupo de semillas

AdvFuzzer tomará múltiples caminos aleatorios con el objetivo de explorar diferentes regiones del espacio de ejemplos adversos entre las dos imágenes. LocalFuzzer tiene como objetivo explorar los puntos de datos cercanos para identificar potenciales ejemplos adversos alrededor de una imagen actual mientras AdvFuzzer da los pasos principales (Qin & Yue, 2022).

- Algoritmos: Ahora presentamos el Algoritmo 1 para AdvFuzzer y el Algoritmo 2 para Local Fuzzer. AdvFuzzer es el algoritmo principal y LocalFuzzer se utiliza en AdvFuzzer.

Figura 30

Artículo 20, Algoritmo AdvFuzzer y Local Fuzzer



Algorithm 1 AdvFuzzer: generating an adversarial example.

Input: f: A black-box model.
$isTargeted$: $True|False$ for targeted/untargeted attack.
img_s: A source image with class s ($f(img_s) = s$).
img_g: A guidance image with $f(img_g) = t$ where $t != s$.
δ: L_2 constraint between img_s and adversarial examples.
Output: img_{adv}: An adversarial example.

$S_{adv} \leftarrow \emptyset$
$SeedPool = GenerateSeedPool(img_s, \delta)$
$TotalStep = 0$
while $size(S_{adv}) == 0$ **do**
  $rand\_idx = random(0, size(SeedPool))$
  $img_{cur} = SeedPool[rand\_idx]$
  $\epsilon = SelectPerturbationMainDir(img_{cur}, img_g, img_s, \delta)$
  $img_{cur} = img_{cur} + \epsilon$
  if ($isTargeted$ and $f(img_{cur}) == t$) or (!$isTargeted$ and $f(img_{cur})$!= s) **then**
    S_{adv}=$LocalFuzzer(f, img_s, img_{cur}, t, isTargeted, M\_LEVEL, \delta)$
  else
    S_{adv}=$LocalFuzzer(f, img_s, img_{cur}, t, isTargeted, S\_LEVEL, \delta)$
  end if
  $SeedPool[rand\_idx] = img_{cur}$
  if $TotalStep++ \geq MAX\_STEP$ **then**
    return $None$
  end if
end while
$img_{adv} = S_{adv}[rand]$
$img_{adv} = WalkBack(f, img_s, img_{adv}, t, isTargeted)$
return img_{adv}

Algorithm 2 LocalFuzzer: generating a set of examples locally.

Input: f: A black-box model.
img_s: A source image with class s ($f(img_s) = s$).
img: An input image.
t: Target class, *i.e.*, the guidance image class where $t != s$.
$isTargeted$: $True|False$ for targeted/untargeted attack.
$fuzzer\_level$: The number of examples to generate.
δ: L_2 constraint between img_s and generated examples.
Output: S_{adv}: A set of successful adversarial examples.

$S_{adv} \leftarrow \emptyset$
$S_{all} \leftarrow \{img\}$
if ($isTargeted$ and $f(img) == t$) or (!$isTargeted$ and $f(img)$!= s) **then**
  $S_{adv} = S_{adv} \cup \{img\}$
end if
for i from 1 to $fuzzer\_level$ **do**
  if S_{adv} != $\emptyset$ **then**
    $img_{rand} = RandomSelect(S_{adv})$
  else
    $img_{rand} = RandomSelect(S_{all})$
  end if
  $img_{mut} = local\_mutation(img_{rand}, img_s, \delta)$
  if ($isTargeted$ and $f(img_{mut}) == t$) or (!$isTargeted$ and $f(img_{mut})$!= s) **then**
    $S_{adv} = S_{adv} \cup \{img_{mut}\}$
  end if
  $S_{all} = S_{all} \cup \{img_{mut}\}$
end for
return S_{adv}

Nota. Adaptado de "Fuzzing-base d hard-lab el black-box attacks against machine learning models", por Qin et al., 2022 (https://doi.org/10.1016/j.cose.2022.102694).

Principal Resultado:

El artículo nos muestra que los atacantes pueden usar una gran cantidad de ejemplos adversarios generados en masa en diferentes momentos o en diferentes situaciones sin iniciar un proceso de nueva generación que requiere mucho tiempo. Además, la generación masiva

se puede aplicar a ejemplos contradictorios generados a partir de cualquier método de ataque para refinar sus ejemplos adversarios "optimizados" al reducir la distancia L2 del ejemplo de origen.Ahora se cuenta con el LocalFuzzer para aplicar en ejemplos adversarios exitosos generados a partir del ataque Boundary, el ataque Opt, el ataque HSJA, el ataque QEBA, los ataques C&W L0 y L2, y nuestros ataques fuzzing para evaluar más intensamente la capacidad de generación masiva y la capacidad de refinamiento de distancia de LocalFuzzer (Qin & Yue, 2022).

4.4 Conclusiones

A continuación, se presentan un resumen detallado de cada uno de los artículos seleccionados, respetando el orden del cuadro presentado previamente y agrupados según las tipologías descritas en el apartado de prefacio:

4.4.1 Conclusiones de artículos de la categoría de fuzzing

Dentro de la primera categoría seleccionada, la cual agrupa diferentes trabajos de investigación relacionados al fuzzing y sus aplicaciones, se puede concluir que, de manera general, esta técnica ha tenido logros significativos en diferentes enfoques del desarrollo de software. Asimismo, se ha demostrado que usar fuzzing mejora el porcentaje de la cobertura de pruebas y también abre la posibilidad de utilizar dichas referencias técnicas en diferentes contextos.

Por último, concluimos que también existen limitaciones en el fuzzing, por ejemplo: algunos de los sistemas requieren un hardware en específico y/o necesitan que el detalle de los datos de entrada sea detallado o estén bajo una estructura de datos específica.

4.4.2 Conclusiones de artículos de la categoría de modelo generativo

En esta categoría, se agrupa diferentes trabajos de investigación relacionados con el uso de diferentes técnicas basados en modelos generativos para el fuzzing. Concluimos de manera general que la investigación realizada en los diferentes portales de artículos científicos nos brindó un panorama de algunos modelos generativos dotados de algoritmos que poseen la capacidad de aprender cualquier tipo de distribución de datos. Para la investigación que se está realizando, se trabajará en un modelo que contemple algoritmos de machine learning par ala generación automática de casos de prueba, a fin de tener una mayor cobertura sobre los escenarios posibles que puedan ser usados para pruebas.

Gracias a la bibliografía revisada, procederemos a realizar un análisis sobre los algoritmos planteados y seleccionar aquel que se ajuste a nuestro objetivo presentado como la generación de casos para que otros aplicativos puedan utilizar aquellas salidas como parte de sus escenarios de prueba.

4.4.3 Conclusiones de artículos de la categoría de machine learning

Para la siguiente categoría, concluimos de manera general que, si es eficaz utilizar modelos de aprendizaje con algoritmos basados en fuzzing para la generación de casos de pruebas, con el fin de detectar vulnerabilidades y/o errores en el sistema. Asimismo, varios de los trabajos realizados han mejorado la precisión de los datos de entrada y con ello, incrementado la cobertura de las pruebas del sistema.

Por un lado, las revisiones sistemáticas realizadas fueron de gran ayuda, ya que nos brindaron un panorama general de las técnicas y herramientas que se han aplicado a diferentes contextos del desarrollo de software, así como también las ventajas y desventajas del uso de machine learning.

Por otro lado, una parte de las investigaciones fueron aplicadas directamente al código (caja negra) y a compiladores en el lenguaje de C, debido a que, las librerías actuales de fuzzing en su mayoría están orientadas a pruebas de seguridad y rendimiento.

En la actualidad, la tecnología de procesamiento del lenguaje natural es relativamente madura, por lo que podemos considerar el uso de tecnologías avanzadas en el campo del procesamiento del lenguaje natural para extraer información útil, como atributos de código, características semánticas y gramaticales de programas para fuzzing.

Finalmente, concluimos que todas las técnicas mencionadas en los artículos nos servirán de conocimiento para definir nuestro enfoque de generación de casos de pruebas basados en fuzzing.

4.4.4 Conclusiones generales

Tras realizar un exhaustivo proceso de investigación y de analizar los artículos expuestos previamente, concluimos que el fuzzing es un método eficiente para descubrir las debilidades del software. Además, la combinación de las pruebas de fuzzing y algún modelo de aprendizaje automático proporcionan una nueva idea para aliviar el problema de los cuellos de botella de las técnicas de fuzzing tradicionales.

De igual forma, la revisión sistemática de cada técnica nos encamino a un sin fin de posibilidades de implementación, así como también la capacidad para conocer sus límites y ventajas. Asimismo, nos ha orientado sobre qué pasos seguir para la implementación de un nuevo enfoque basado en fuzzing.

Según la investigación realizada, el primer paso que debemos tomar en cuenta es la generación de archivos semilla (DataSets) con el fin de alimentar nuestro modelo predictivo. Como segundo paso, debemos preparar el modelo generativo dotado de algoritmos de machine learning que permitan la generación de los casos de prueba erróneos. Para ello, podemos utilizar algún algoritmo de machine learning que se ajuste a nuestro contexto. Como tercer paso está el filtrado de casos de prueba, debido a que se busca contar con la mejor precisión posible para evitar caer en redundancia de data. Y, por último, pero no menos importante, la selección de operadores de mutación y análisis de explotabilidad.

Asimismo, concluimos que una de las limitaciones de la revisión sistemática es que la literatura no es del todo suficiente, debido a que, muchos de los autores mencionan que hace falta más investigaciones para poder tener una mayor precisión en las comparaciones técnicas, con el fin de analizar y evaluar el rendimiento del fuzzing en diferentes contextos.

Finalmente, se puede concluir que, si bien es cierto el fuzzing nació para aplicarse en las pruebas de seguridad (caja negra) nosotros consideramos que está técnica también puede ser eficaz en un enfoque basado en fuzzing utilizando algoritmos de machine learning para pruebas de caja blanca.

CAPÍTULO 5. DESARROLLO DEL PROYECTO

El presente capítulo del proyecto tiene por finalidad presentar el desarrollo del modelo y sus componentes mostrando el análisis y desarrollo.

5.1 Análisis de las soluciones para la construcción de la arquitectura

5.1.1 Análisis de la necesidad

La necesidad de proponer una arquitectura cloud basado en algoritmos fuzzing y redes neuronales surge, debido al gran impacto que se ocasiona cuando un bug es detectado en el ambiente de producción del sistema, en base a nuestra experiencia, el costo para solucionar los bugs puede ocasionar la pérdida de la disponibilidad del sistema y con ello, retrabajo por parte de los involucrados en el desarrollo de software.

5.1.2 Análisis de herramientas

Al tener conocimiento de la necesidad se realizó un primer análisis de posibles tipos algoritmos basados en fuzzing para la generación de escenarios de pruebas. Las opciones principales son las siguientes:

Tabla 12

Análisis de los tipos de algoritmos

Criterios	**Mutational and generational**	**Input Structure Aware Fuzz Algorithms**	**Input Structure Aware "Smart" fuzz algorithms**
Permite generar datos de entrada desde cero	20%	10%	10%
Basado en la generación (GA Algorithm)	20%	10%	10%
Hibrido	10%	10%	10%
Muta semillas de entrada	20%	10%	10%
Permite la exploración de varios elementos disjuntos y mutuamente incompatibles	10%	20%	10%
Escalable	10%	10%	10%
Total	**100%**	**70%**	**60%**

Nota. Leyenda: 10% No cumple con los requerimientos del Proyecto, **20**% Cumple en cierta medida, 30% Totalmente alineado al proyecto

En base a la tabla mostrada, hemos decidido utilizar un algoritmo fuzzing del tipo **Mutational and generational fuzz Algorithms,** debido a que cumple con la mayoría de los criterios mostrados.

En segundo lugar, se realizó un análisis sobre los diferentes tipos de pruebas que existen a nivel de fuzzing.

Tabla 13

Análisis de los tipos de pruebas basados en fuzzing

Criterios	Black-Box Fuzzer	White-Box Fuzzer	Gray-Box Fuzzer
Escalabilidad	20%	30%	10%
Tiempo de desarrollo	20%	20%	10%
Datos de entrada variables	30%	30%	20%
Reutilización de código	20%	20%	20%

Nota. Leyenda: 10% No cumple con los requerimientos del Proyecto, 20% Cumple en cierta medida, 30% Totalmente alineado al Proyecto

En base a la tabla mostrada, hemos decidido atacar las pruebas de caja blanca del sistema, es decir a nivel de microservicios (API), debido a que, la información puede ser manejable y de acuerdo con la actual arquitectura del proyecto de software, el desarrollo de la implementación de los algoritmos será viable.

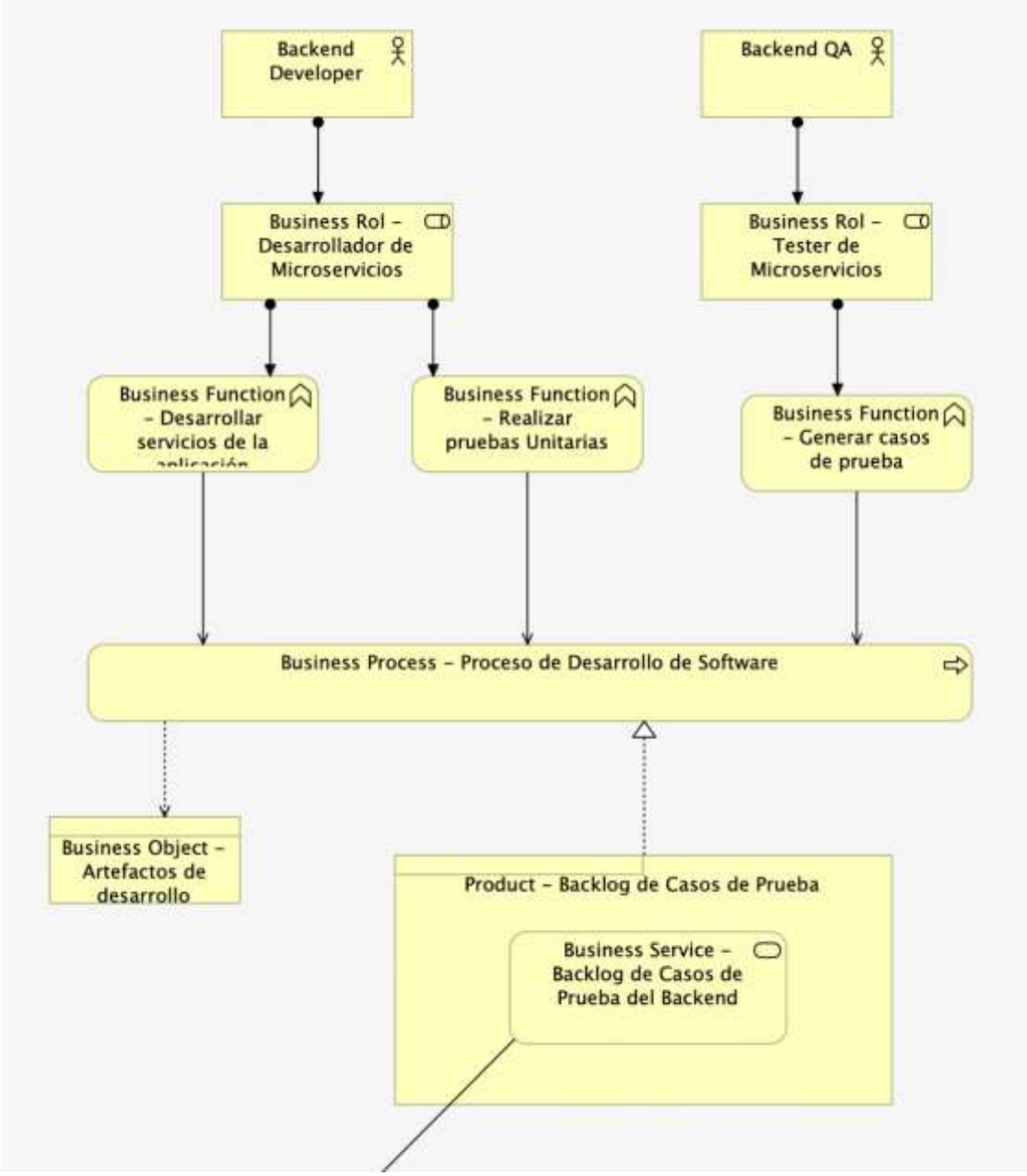
5.2 Diseño del modelo

5.2.1 Capa del negocio

Son dos roles los principales involucrados en el proceso desarrollo del software de la organización, el Developer se encarga de desarrollar los microservicios del sistema y de realizar las pruebas unitarias. El QA está encargado en asegurar la calidad del sistema realizando pruebas manuales a los microservicios con el fin de encontrar defectos en el sistema.

Figura 31

Capa de Negocio

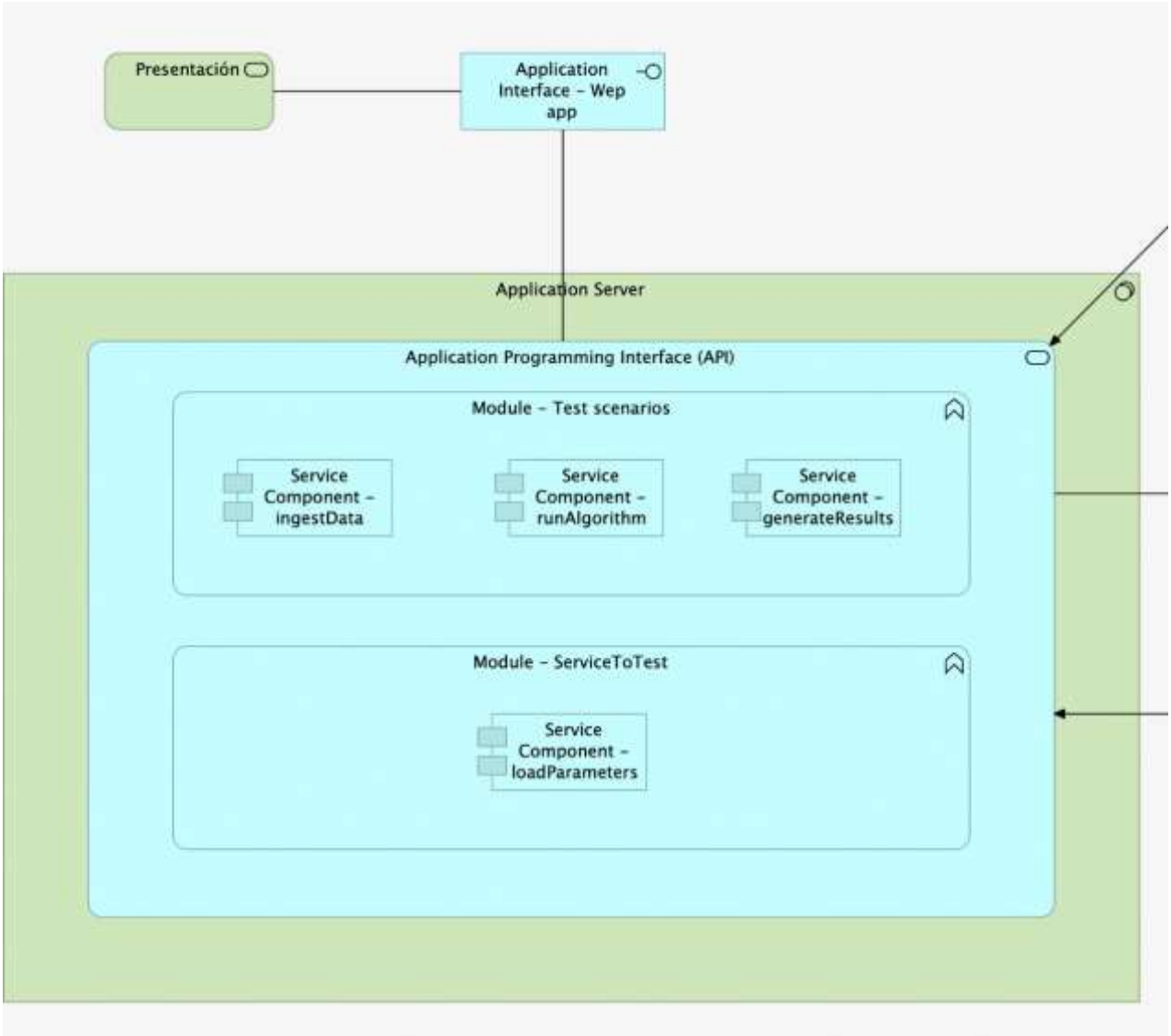


5.2.2 Capa de aplicación y datos

Como se muestra en la siguiente imagen, en la capa de aplicación se integrarán diferentes servicios para realizar la ingesta de la información, la ejecución del algoritmo propuesto y la obtención de los escenarios de pruebas como resultados. Asimismo, se contará con una interfaz para comunicar los resultados con la capa de presentación, en la cual, se mostrarán los resultados de manera clara y concisa.

Figura 32

Capa de Aplicación y Datos

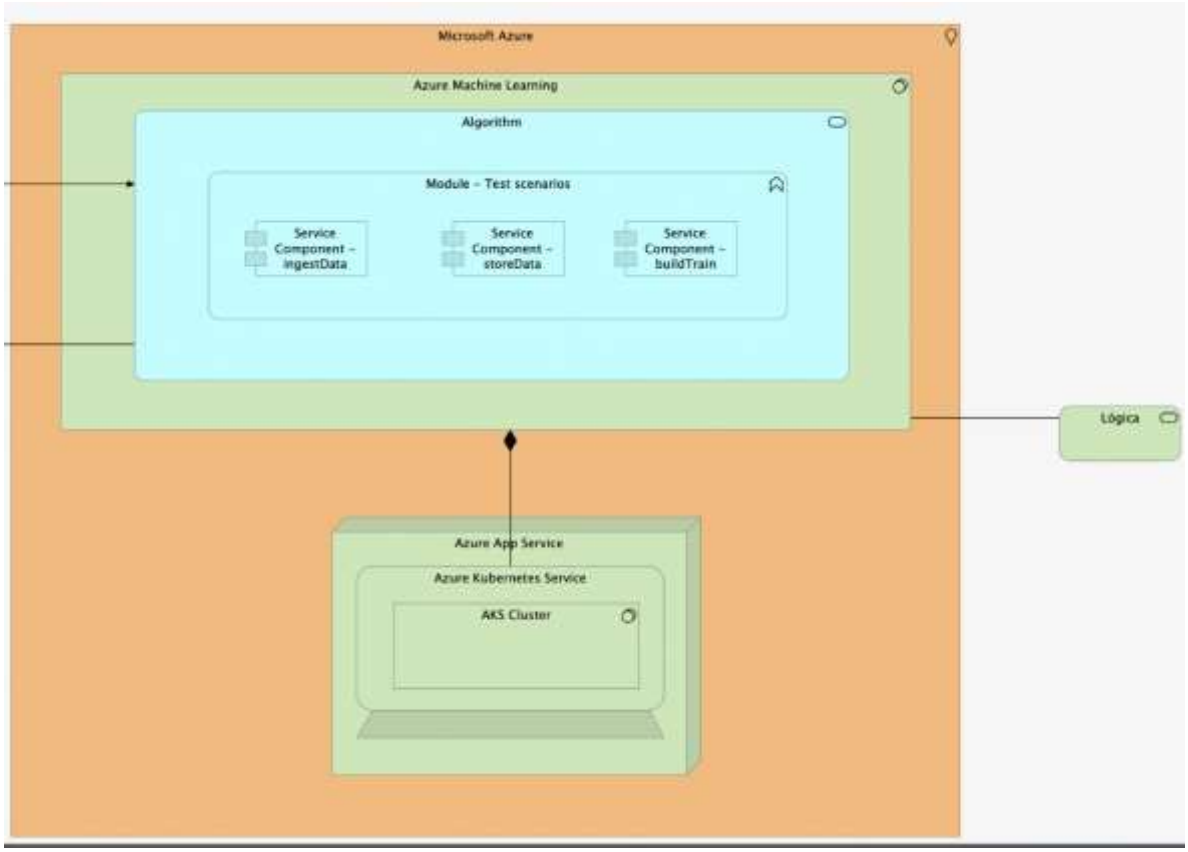


5.2.3 Capa tecnológica

Para el desarrollo de la capa tecnológica, proponemos el uso de recursos de **Azure Machine Learning,** debido a que, se acopla a la solución planteada y a los recursos usados actualmente por la organización. Asimismo, implementamos el uso de **Azure Kubernetes Service** para desplegar el sistema en pequeños componentes con el fin de facilitar la automatización y la configuración declarativa del sistema.

Figura 33

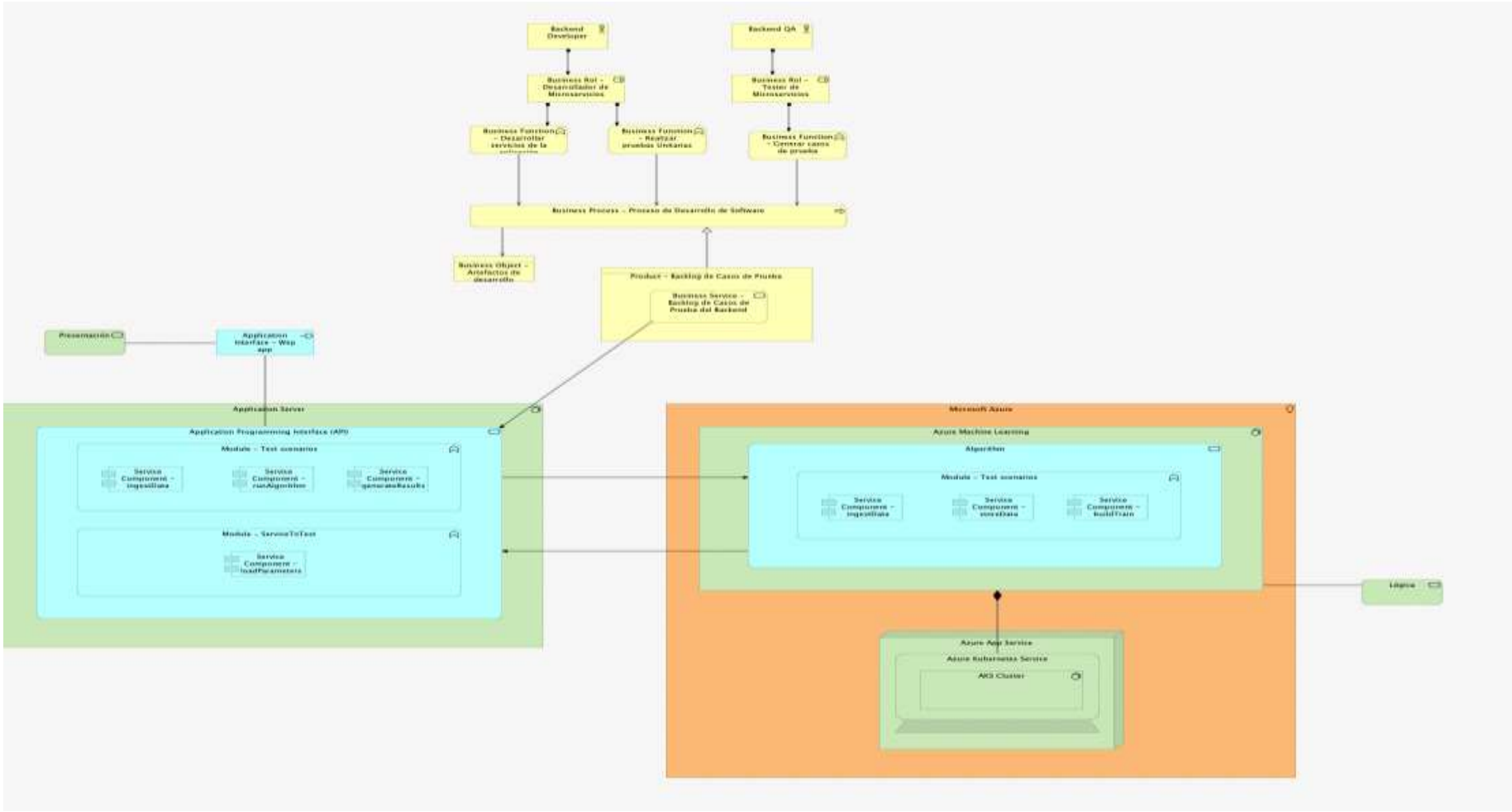
Capa Tecnológica



5.2.4 Capa Integrada

Figura 34

Capa Integrada – Arquitectura en Capas



5.2.5 Proceso del desarrollo de la arquitectura

5.2.5.1 Metodología Usada

5.2.5.1.1 Nombre del proyecto de software

Arquitectura cloud basada en la técnica fuzzing para la generación de escenarios de pruebas en el sector de desarrollo de software.

5.2.5.1.2 Metodologías/Normas Utilizadas

Para el desarrollo de la tesis planteamos el uso del marco de trabajo **SCRUM (**The Scrum GuideTM), ya que nuestro objetivo es entregar un producto mínimo viable en corto tiempo.

5.2.5.1.3 Nombre del proceso

Proceso de desarrollo de software

5.2.5.1.4 Lista de subprocesos y roles

Tabla 14

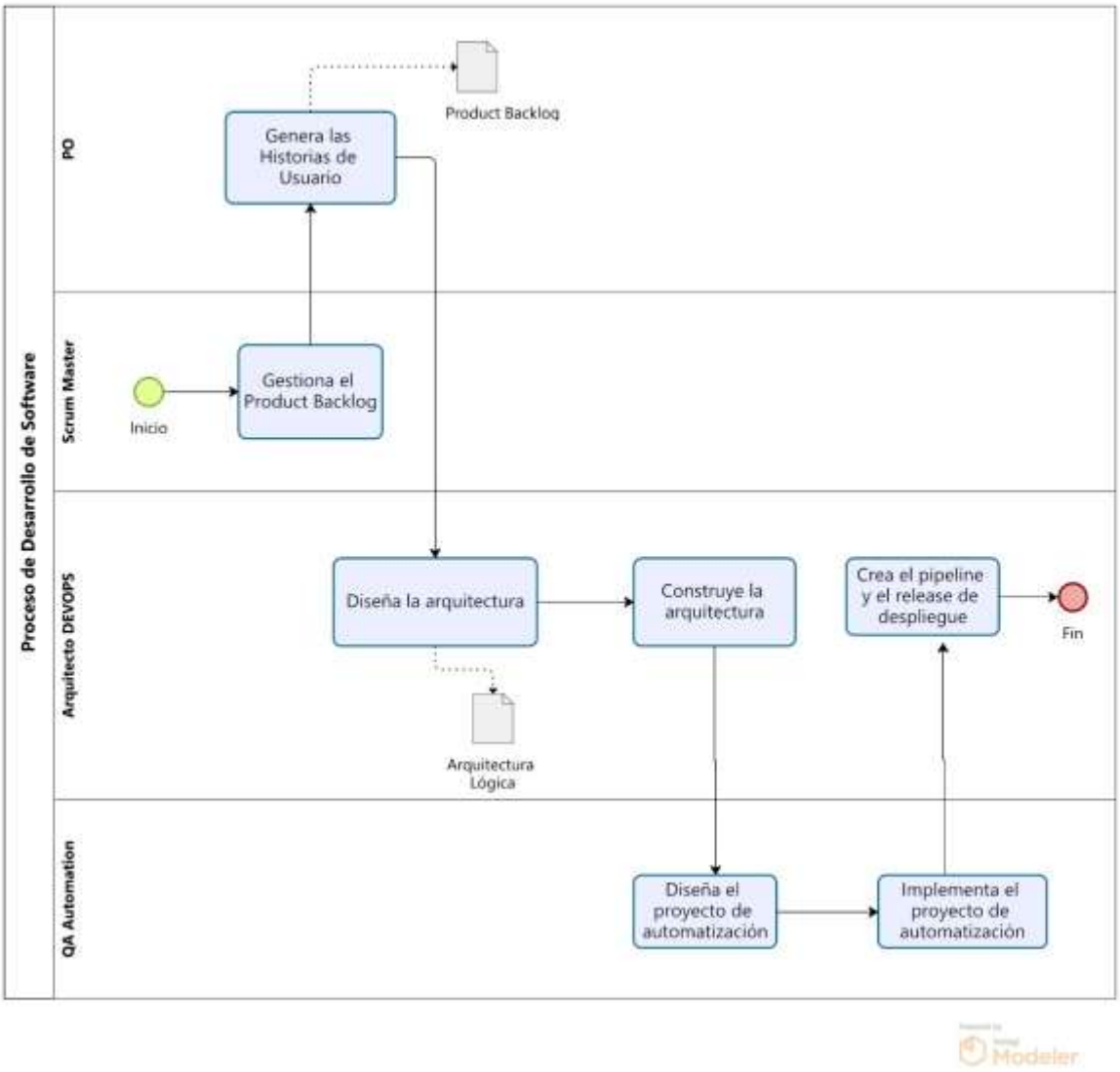
Lista de subprocesos y roles

SubProcesos	Rol
Creación de las historias de usuario	Product Owner
Gestionar al equipo de desarrollo, ayudar a eliminar impedimentos que afecten la entrega del producto. Facilitador del marco de trabajo Scrum. Gestiona el Product Backlog	Scrum Master
Diseño y construcción de la arquitectura cloud Capacitación constante de Integración continua Generación de pipelines y recursos de automatización	Arquitecto DevOps
Diseño e implementación del proyecto de automatización para la generación de escenarios de pruebas utilizando algoritmos de machine learning.	QA Automación

5.2.5.2 Proceso Preliminar

Figura 35

Proceso Preliminar



5.2.5.3 Diagramas específicos de subprocesos

Figura 36

Subproceso: Diseño y Construcción de la Arquitectura

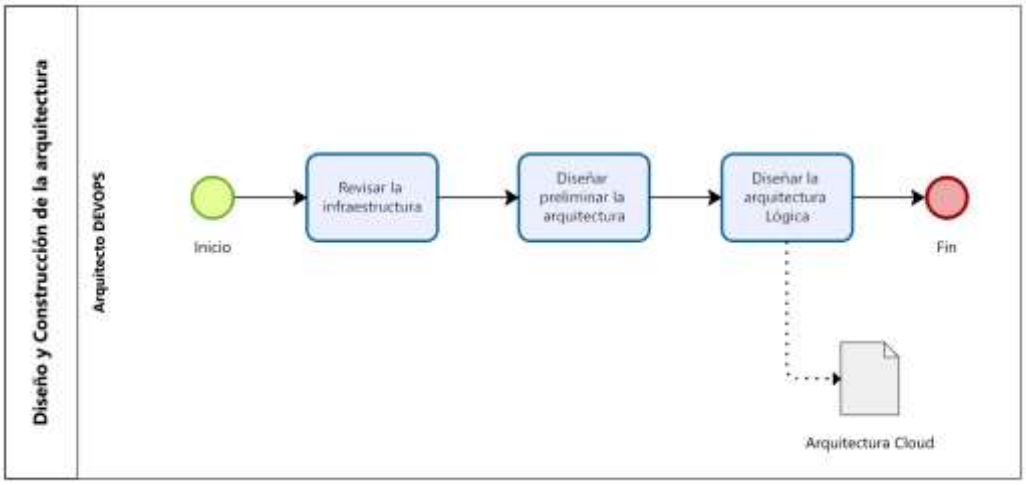


Figura 37

Subproceso: Generación del Producto Backlog

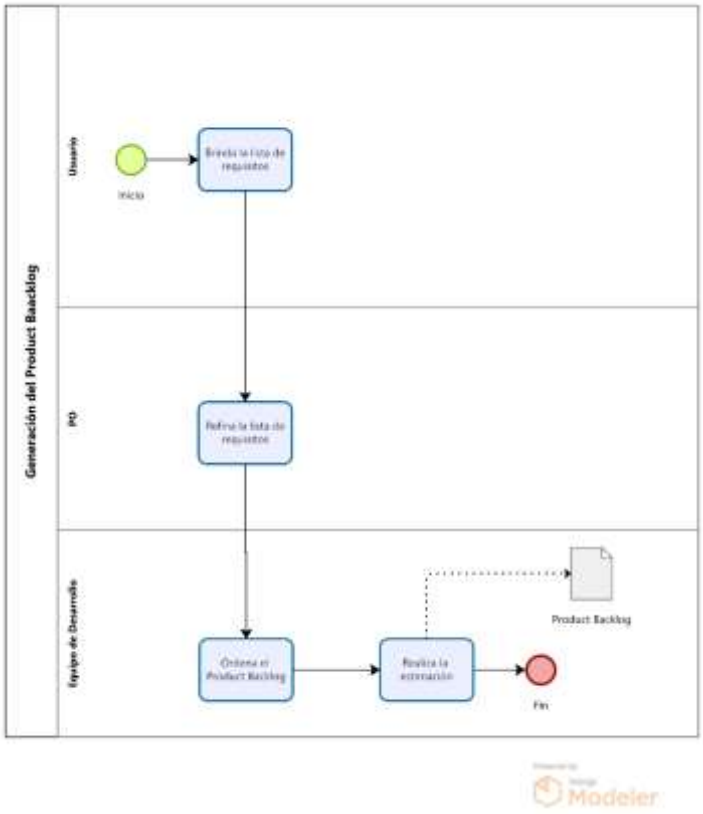
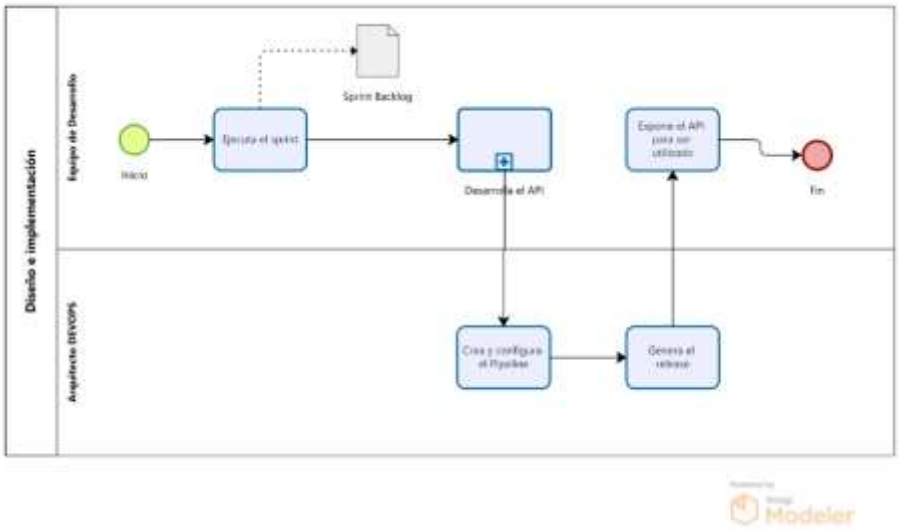


Figura 38

Subproceso: Diseño e Implementación



5.2.5.4 Entradas

Las entradas para el desarrollo de la solución propuesta son las siguientes:

- Documentación de las API del sistema

  Uno de los requerimientos principales del proyecto es la creación de un archivo en formato .yml para documentar todas las API del sistema que serán utilizadas por el algoritmo basado en fuzzing. El detalle completo de los parámetros de entrada y categorización se encuentra en la especificación de las historias de usuario.
- Documentación para el desarrollo del algoritmo fuzzing

  Para el desarrollo del algoritmo, se deben utilizar los siguientes métodos y funciones de la siguiente documentación:

  1. **https://www.fuzzingbook.org/html/Fuzzer.html**
  2. **https://llvm.org/docs/LibFuzzer.html**
- Documentación para el desarrollo de la red neuronal

  Para el desarrollo de la red neuronal, se plantea el uso de las siguientes referencias para el correcto desarrollo del modelo generativo.

  1. **https://programmerclick.com/article/5871883966/**
- Cuentas con licencia de Azure

  Para la creación de un ambiente de desarrollo en el portal de Azure, es necesario requerir al cliente la creación de las credenciales para los diferentes roles del equipo.

5.2.5.5 Salidas

Las principales salidas de algoritmo propuesto son las siguientes:

- **Dashboard**: Con el objetivo de la visualización de los resultados (Escenarios de pruebas generados por el algoritmo fuzzing) se plantea la generación de un Dashboard con el reporte de los escenarios encontrados, ejecutados y los defectos encontrados por el mismo. Actualmente, existen frameworks que permite la generación de un reporte automático para su visualización. Uno de ellos es **Serenity Reports**, se plantea el uso de esta herramienta debido a que tiene integración con las tecnologías que estamos presentando y así como también, los lenguajes de programación y la nube.

  La segmentación de la información facilitará dar seguimiento a los defectos encontrados y los nuevos escenarios encontrados por el algoritmo.

5.3 Procesos identificados y tiempos

Del levantamiento de la información al realizar el análisis de los procesos se pudo determinar la siguiente información:

5.3.1 Costo de personal en el proceso

Tabla 15

Costo de personal en el proceso

Personal	**Costo por hora (8h) en dólares**
Product Owner	15
Scrum Master	13
SDET	15
Arquitecto DevOps	20

Nota. Información al 12 de diciembre del 2022. Adaptados de "Salarios de puestos de tecnología en el Perú", por Glassdoor, 2022, (https://www.glassdoor.com/)

5.3.2 Obtención de requisitos

En esta sección se detallará los requisitos del sistema.

5.3.2.1 Product backlog

Tabla 16

Product Backlog

Código	Descripción
HU-01	Como usuario quiero tener la documentación de todos los servicios **REST** full del sistema que serán involucrados en las pruebas para la segmentación de la información
HU-02	Como usuario quiero contar con un algoritmo de machine learning basado en fuzzing para generar escenarios de prueba de cada servicio REST del sistema
HU-03	Como usuario quiero utilizar una red neuronal para alimentar el algoritmo fuzzing.
HU-04	Como usuario quiero visualizar los escenarios de pruebas generados por el algoritmo en una aplicación web para ejecutar los escenarios de prueba.
HU-05	Como usuario quiero visualizar los resultados de la ejecución de las pruebas para conocer los posibles bugs del sistema encontrados por el algoritmo.
HU-06	Como usuario quiero guardar los casos de pruebas generados por el algoritmo en una base de datos no relacional en Azure para gestionar la información.
HU-07	Como usuario quiero utilizar el recurso de **Azure Machine Learning** para instanciar al algoritmo desarrollado con la integración continua del proyecto

5.3.2.2 Especificación de las historias de usuario

5.3.2.2.1 HU-01

Tabla 17

Historia de Usuario 01

Historia de Usuario - 01
Nombre de historia:
Como usuario quiero tener la documentación de todos los servicios rest full del sistema que serán involucrados en las pruebas para la segmentación de la información

Responsables: Product Owner, Tech Lead, QA

Descripción:

Se necesita que las API del sistema sean accesibles, categorizadas y entendibles.

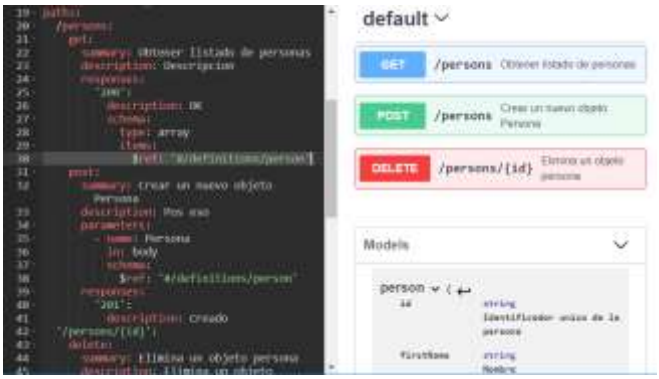
Consideraciones:

1. Utilizar el lenguaje YML para documentar los servicios.
2. Verificar que todos los servicios tengan la especificación y/o códigos de error.
3. Antes de agregar el api al consolidado de información, verificar que el happyPath del API funcione correctamente.
4. Categorizar los servicios por funcionalidad, utilizar la siguiente documentación para eso. https://swagger.io/specification/
5. El archivo. yml con el consolidado de APIs, no debe contar con errores de sintaxis; para ello verificar en swagger.io.

Prototipos o Mockup:

Figura 39

Prototipo – Historia de Usuario 01



Criterios de Aceptación:

Cuando	**Espero**
Compilo el archivo. yml en un swagger.io	***Debo ver la documentación completa de cada api (request, parameters, headers, body, expected results, body response, etc)***

5.3.2.2.2 HU-02

Tabla 18

Historia de Usuario 02

Historia de Usuario - 02
Nombre de historia: *Como usuario quiero contar con un algoritmo de machine learning basado en fuzzing para generar escenarios de prueba de cada servicio REST del sistema* **Responsables:** Tech Lead, QA Automation, QA **Descripción:** ***El algoritmo debe ser capaz de generar data dummy que posteriormente servirán como escenarios de pruebas.*** **Consideraciones:** 6. Revisar la documentación en formato. yml e identificar los parámetros de entrada de las APIs del sistema. o Crear una clase "Params", y generar atributos para cada **parámetro** del api (Considerar: pathParams, queryParam, header, body). o Ejemplo: API **X**, tiene el parámetro **A** de tipo String [3-5] o Agregar a la clase params el atributo String **A [3-5]** 7. Una vez identificado los parámetros de entrada, verificar que no exista información duplicada. 8. Para generar los datos de entrada de manera dinámica se requiere implementar los siguientes métodos basados en fuzzing de la siguiente librería: **https://www.fuzzingbook.org/html/Fuzzer.html** 9. Cada parámetro necesitará una función/método diferente para generar data de prueba aleatoria.

Prototipos o Mockup:

Figura 40

Prototipo – Historia de Usuario 02

Criterios de Aceptación:

Cuando	**Espero**
Ingrese los parámetros de un api al algoritmo	***El algoritmo devuelva data dummy para probar el api.***

5.3.2.2.3 HU-03

Tabla 19

Historia de Usuario 03

Historia de Usuario - 03
Nombre de historia:
*Como **usuario** quiero utilizar una red neuronal para alimentar el algoritmo fuzzing.*
Responsables: Tech Lead, QA Automation, QA
Descripción:
La red neuronal permitirá que los datos generados por el algoritmo basado en fuzzing sean lo más reales posibles, es decir, data con sentido.
Consideraciones:
10. Se implementará la red neuronal **APIDataOrchestrator** para simplificar la data generada por el algoritmo basado en fuzzing. 11. Utilizar la siguiente documentación como referencia. **https://programmerclick.com/article/5871883966/**

Prototipos o Mockup:

Figura 41

Prototipo – Historia de Usuario 03

```
import java.util.Random;
/**
 *
 * @author Wagner
 */
public class Estructura {
    int nK[] = {2, 3, 3, 1};
    double[][][] w = new double[nK.length][9][9]; //peso
    double[][] s = new double[nK.length][9]; //salida

    // tabla de entrenamiento
    final int x1[] = {34}; // entrada 1 (x1)
    final int x2[] = {23}; // entrada 2 (x2)
    final int di[] = {57}; // salida deseada 1 (d1)

    public void estruct() {

        int row = 1;
        for (int i = 0; i < nK.length; i++) {
            row *= nK[i];
        }

        // inicializar vector con pesos aleatorios.
        double[] vw = new double[row];
        for (int i = 0; i < vw.length; i++) {
            vw[i] = new Random().nextDouble();
        }

        // Añadir los pesos(w) y cálculo salidas(s) - Propagación hacia delante -
        propagacion_hacia_delante(vw);

    }

    private void propagacion_hacia_delante(double[] vw) {

        // vector a lista
        List<Double> lw = new ArrayList<>();
        for (int i = 0; i < vw.length; i++) {
            lw.add(vw[i]);
        }
        Iterator wi = lw.iterator();

        System.out.println("* Entradas (x1, x2): ");
        int k = 0;
        s[k][0] = x1[0] / 100.0;
        s[k][1] = x2[0] / 100.0;
        System.out.println("s[" + k + "][" + 0 + "] = " + s[k][0]);
```

Criterios de Aceptación:

Cuando	**Espero**
Ingrese los parámetros de un api al algoritmo	***El algoritmo devuelva data smart para probar el api.***

5.3.2.2.4 HU-04

Tabla 20

Historia de Usuario 04

Historia de Usuario - 04
Como usuario quiero visualizar los escenarios de pruebas generados por el algoritmo en una aplicación web para ejecutar los escenarios de prueba. **Responsables:** Tech Lead, QA

Descripción:

Para la creación de un Dashboard se plantea el uso del framework Serenity para generar reportes dinámicos de los escenarios de pruebas generados por el algoritmo

Consideraciones:

12. Para el uso del framework **Serenity** utilizar la siguiente documentación:
    https://serenity-bdd.github.io/docs/reporting/the_serenity_reports
13. El reporte deberá ser generado una vez finalizada la ejecución del algoritmo basado en fuzzing.
14. El reporte debe incluir los siguientes puntos:

- Número de escenarios de prueba generados y la especificación de cada uno.
- Un diagrama para ver el número de casos encontrados y los que se detectaron anteriormente (Información de la base de datos)

Prototipos o Mockup:

Figura 42

Prototipo – Historia de Usuario 04

Criterios de Aceptación:

Cuando	**Espero**
Ingrese terminé la ejecución del algoritmo	***Se generará un reporte con los resultados de la ejecución, en una carpeta del proyecto de automatización.***

5.3.2.2.5 HU-05

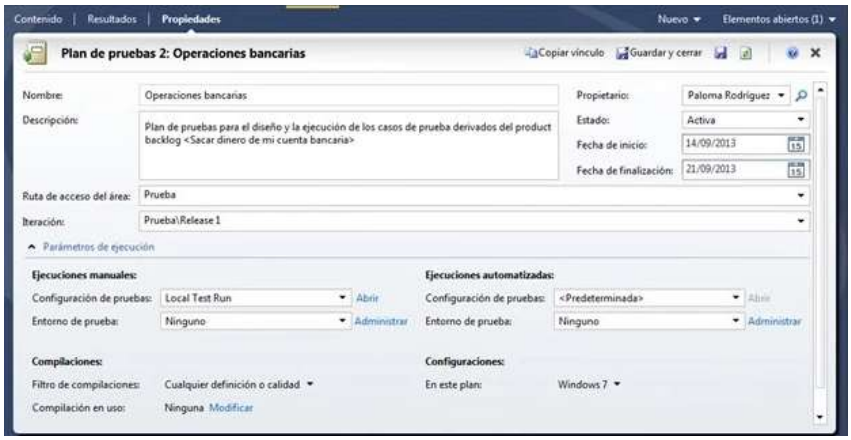
Tabla 21

Historia de Usuario 05

Historia de Usuario - 05
Como usuario quiero visualizar los resultados de la ejecución de las pruebas para conocer los posibles bugs del sistema encontrados por el algoritmo. **Responsables:** Tech Lead, QA Automation **Descripción:** ***Después de la implementación del Dashboard, es necesario incluir una pantalla para ver el estado de la ejecución de las pruebas.*** **Consideraciones:** En el reporte generado por serenity agregar una pantalla para verificar los siguientes puntos: • Estado de la ejecución del escenario de prueba (ACTIVO, NO_ACTIVO, NUEVO_ERROR) • El reporte debe tener una sección para replicar cada escenario y verificar si hay un issue o no. • La pantalla debe permitir ingresar un estado al escenario • La pantalla debe permitir realizar el CRUD de los escenarios generados. **Prototipos o Mockup:**

Figura 43

Prototipo – Historia de Usuario 05



Criterios de Aceptación:

Cuando	Espero
Ingreso al reporte y me dirijo a la pestaña "Escenarios generados"	*Ver los escenarios generados y el detalle de cada uno de ellos para poder replicarlo y agregar un estado de su ejecución.*

5.3.2.2.6 HU-06

Tabla 22

Historia de Usuario 06

Historia de Usuario - 06
Como usuario quiero guardar los casos de pruebas generados por el algoritmo en una base de datos no relacional en Azure para gestionar la información. **Responsables:** Tech Lead, QA Automation, Arquitecto Devops **Descripción:** ***Se utilizarán base de datos no relacionales para guardar la información de los escenarios de pruebas generados, con la finalidad de centralizar la información con fines de futuros desarrollos.*** **Consideraciones:** Se deberá de generar la siguiente BD y la colección: **Database: test-cases-generator** **Collection: smart-data** **Attributes:**

Figura 44

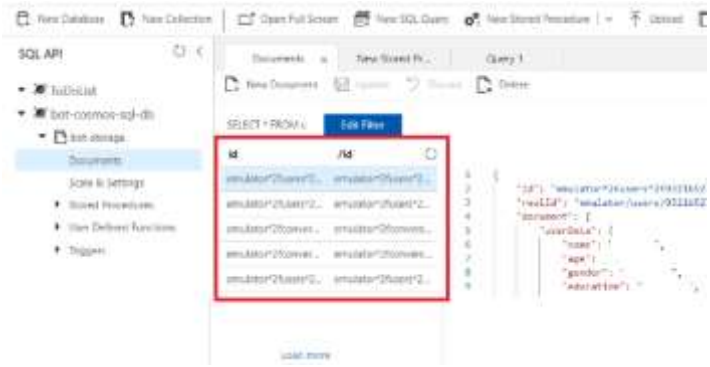
Prototipo – Test Cases Generator Collection

id	HashAttribute
APIName	String
Parameters	JsonBody
Status	Boolean (1,0)
isDefect	Boolean (1,0)
initialDate	dateTime

Prototipos o Mockup:

Figura 45

Prototipo – Historia de Usuario 06



Criterios de Aceptación:

Cuando	**Espero**
Se generé un escenario de prueba	***Guardar el caso de prueba generado en la base de datos no relacional***

5.3.2.2.7 HU-07

Tabla 23

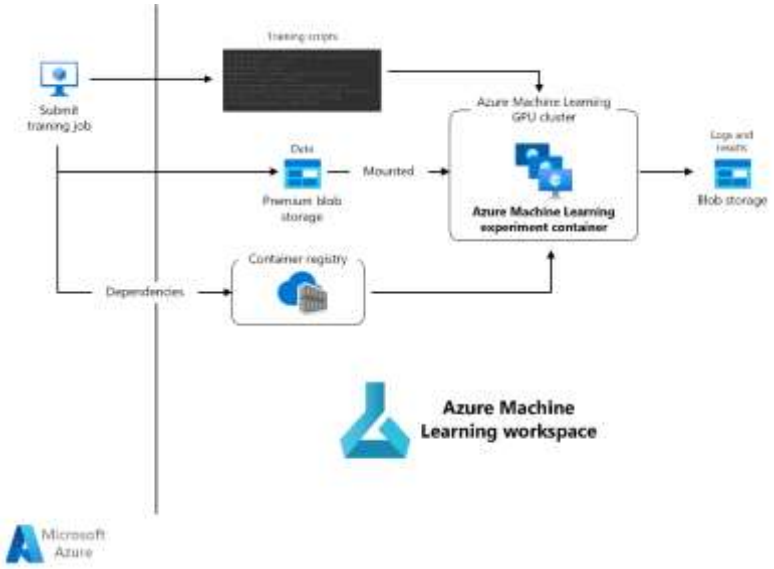
Historia de Usuario 07

Historia de Usuario - 07
Como usuario quiero utilizar el recurso de **Azure Machine Learning** para instanciar al algoritmo desarrollado con la integración continua del proyecto **Responsables:** Tech Lead, Arquitecto DevOps **Descripción:** El uso del recurso de Azure Machine Learning permitirá desarrollar la aplicación en un entorno Cloud, así como también, la integración de diferentes recursos que permitirán la escalabilidad del proyecto de tesis.

Prototipos o Mockup:

Figura 46

Prototipo – Historia de Usuario 07



5.3.3 Plan de integración

5.3.3.1 Lenguajes de programación

Tabla 24

Lenguajes de programación

Lenguaje de programación	Descripción
Java	Java es uno de los lenguajes de programación más populares de todos los tiempos, es de software abierto y tiene una comunidad muy grande en todo el mundo. Muchas de las aplicaciones para páginas web y teléfonos fueron desarrolladas utilizando este famoso lenguaje de programación basado en objetos.

5.3.3.2 Herramientas y tecnologías

Tabla 25

Herramientas y tecnologías

Herramientas y tecnologías	Descripción
American Fuzzy Lop	Es un fuzzer de software gratuito que emplea algoritmos genéticos para aumentar de manera eficiente la cobertura de código de los casos de prueba. Hasta ahora, ha detectado docenas de errores de software significativos en los principales proyectos de software libre. El código fuente de American fuzzy lop está publicado en GitHub. Su nombre es una referencia a una raza de conejo, el American Fuzzy
Cucumber	Cucumber es un framework para el desarrollo de pruebas automatizadas, utiliza el lenguaje gherkin para realizar las notaciones de las funciones, así como también, tiene una integración con diferentes librerías como serenity y bdd.

5.3.3.3 Estándares y plantillas

Tabla 26

Nomenclaturas

Estándares y plantillas	Nomenclatura
Historia de Usuario	HU
Criterios de aceptación	CA
Red Neuronal	RN
Algoritmo Fuzzing	AF

5.3.3.4 Cronograma preliminar del proyecto y entregables

Tabla 27

Cronograma del proyecto

Actividad	Entregable	Rol	Inicio	Fin
Sprint 1				
Elección y creación de la base de datos no relacional del proyecto. Creación del recurso cloud machine-learning. Creación del repositorio de automatización para la generación de escenarios de pruebas. Creación de usuarios cloud para el dev team. El equipo de QA debe mapear los servicios web que se utiliza en cada flujo del sistema.	Creación de recursos Cloud Mapeo de los servicios web de la aplicación Creación del repositorio de trabajo Creación de usuarios y accesos de red.	PO, SM, QA, DevOps	20/02/23	03/03/23
Release 1: 03/03/23				
Sprint 2				
Investigación y documentación del algoritmo fuzzing Investigación y documentación de la red neuronal para el aprendizaje automático del algoritmo fuzzing Convocar a una reunión con expertos para decidir el lenguaje de programación a utilizar para la automatización	Integración del repositorio de automatización con los pipelines de la aplicación. Documentación de los algoritmos que se utilizarán para la generación de casos de prueba.	PO, SM, QA, DevOps	06/03/23	17/03/23

Implementar la librería del algoritmo fuzzing y lanzar una prueba de su ejecución. Integrar el repositorio del proyecto con el pipeline del sistema. ***Release 2: 17/03/23***	MVP del funcionamiento del algoritmo.			
Sprint 3				
Generación de data inputs para alimentar la red neuronal. Primera integración de la red neuronal al algoritmo basado en fuzzing. Integración de la primera suite de servicios web con el algoritmo desarrollado. Subir los cambios a través de la integración continua del proyecto. Ejecutar algoritmo desde el pipeline de azure devops. ***Release 3:31/03/23***	Primer despliegue del algoritmo utilizando la primera suite de servicios web. Ejecución del pipeline de manera manual desde el azure devops.	PO, SM, QA, DevOps	20/03/23	31/03/23
Sprint 4				
Integración de la segunda suite de servicios web con el algoritmo desarrollado. Ejecutar el algoritmo y generar los escenarios de pruebas. Guardar los casos de pruebas ejecutados en la base de datos manualmente. Probar manualmente los escenarios generados y	Generación de escenarios de pruebas automáticos para la segunda suite de pruebas.	PO, SM, QA, DevOps	03/04/23	14/04/23

verificar si es un happy o unhappy path. Despliegue de los cambios. Programar un horario en el pipeline, para su ejecución automática.				
Release 4: 14/04/23				
Sprint 5				
Integración de la **tercera** suite de servicios web con el algoritmo desarrollado. Ejecutar el algoritmo y generar los escenarios de pruebas. Guardar los casos de pruebas ejecutados en la base de datos **automáticamente**. Implementar un reporte en html para la visualización de los escenarios de pruebas generados y los resultados de su ejecución.	Generación de escenarios de pruebas desde el pipeline y generación de un reporte en html para la visualización de los casos generados y los resultados de su ejecución.	PO, SM, QA, DevOps	17/04/23	28/04/23
Release 5: 28/04/23				
Sprint 6				
Generación de las métricas tras los resultados de todos los casos generados por el algoritmo para cada suite de pruebas. Implementación de un plan de escalabilidad para futuros servicios web desarrollados. Mantenimiento del proyecto de automatización y evaluar la	Reporte de la generación de escenarios de pruebas automatizado. Fase final del proyecto, resultados de la automatización y la generación de métricas.	PO, SM, QA, DevOps	01/05/23	12/05/23

posibilidad de integrar el algoritmo con las pruebas automatizadas del sistema. Ejecutar el pipeline después de cada pase del equipo desarrollo al ambiente de pruebas.	Desarrollo de un plan de trabajo futuro para la continuidad del proyecto.	

Release 6: 12/05/23

Nota. Se detallan las actividades por sprint y por rol, las fechas de inicio y fin de cada iteración y el porcentaje de avance total.

CAPÍTULO 6. VALIDACIÓN DE LA PROPUESTA

6.1 Validación de factibilidad económica

6.1.1 Costo de RRHH

Se realizó una estimación sobre la duración de cada actividad en horas hombre por cada rol involucrado en el proceso de desarrollo, A continuación, el detalle:

Tabla 28

Roles y actividades para medir el costo RRHH

Rol Abreviatura	**Rol**	**SubProcesos**
PO	Product Owner	Creación de las historias de usuario
SM	Scrum Master	Gestionar al equipo de desarrollo, ayudar a eliminar impedimentos que afecten la entrega del producto.
		Facilitador del marco de trabajo Scrum. Gestiona el Product Backlog
DevOps	Arquitecto DevOps	Diseño y construcción de la arquitectura cloud
		Capacitación constante de Integración continua
		Generación de pipelines y recursos de automatización
QA	QA Automación	Diseño e implementación del proyecto de automatización para la generación de escenarios de pruebas utilizando algoritmos de machine learning.

La estimación contempla el desarrollo en 6 Sprint de dos semanas de trabajo cada una, utilizando la metodología de Scrum.

En la siguiente tabla se muestra el detalle de las horas por cada uno de los roles para cada sprint.

Tabla 29

Detalle de horas

Sub-Proceso	Sprint 01				Sprint 02				Sprint 03			
	PO	SM	DEVOPS	QA	PO	SM	DevOps	QA	PO	SM	DevOps	QA
Creación de las historias de usuario	40											
Gestionar al equipo de desarrollo, ayudar a eliminar impedimentos que afecten la entrega del producto.		20				20				20		
Facilitador del marco de trabajo Scrum. Gestiona el Product Backlog		30				30				30		
Diseño y construcción de la arquitectura cloud			80				80				80	
Capacitación constante de Integración continua			20				20				20	
Generación de pipelines y recursos de automatización			10									
Diseño e implementación del proyecto de automatización para la generación de escenarios de pruebas utilizando algoritmos de machine learning.				30				30				30

	Sprint 04				Sprint 05				Sprint 06			
	PO	SM	DEVOPS	QA	PO	SM	DevOps	QA	PO	SM	DevOps	QA
Creación de las historias de usuario												
Gestionar al equipo de desarrollo, ayudar a eliminar impedimentos que afecten la entrega del producto.		20				20				20		
Facilitador del marco de trabajo Scrum. Gestiona el Product Backlog		30				30				30		
Diseño y construcción de la arquitectura cloud			80				80				80	
Capacitación constante de Integración continua			20				20				20	
Generación de pipelines y recursos de automatización												
Diseño e implementación del proyecto de automatización para la generación de escenarios de pruebas utilizando algoritmos de machine learning.				10				10				10

Se estimó un total de 1070 horas hombre de desarrollo con los 4 roles definidos previamente.

Seguidamente, se realizó el cálculo de costo por horas de cada rol, este monto se dividió con el salario de cada uno, según las propuestas indicada por Glassdoor (s.f) https://www.glassdoor.com/Salaries/index.htm, sobre la cantidad de horas trabajadas por semana, estimadas en 40 horas, obteniendo el costo por hora de cada rol y multiplicándolo por las horas requeridas nos da el costo total.

Tabla 30

Costo total x rol

Rol Abreviatura	Rol	Horas dedicadas	Salario según Glassdoor	Costo por hora	Costo total
PO	Product Owner	40	S/ 5,750.00	S/ 35.94	S/ 1,437.50
SM	Scrum Master	300	S/ 6,000.00	S/ 37.50	S/ 11,250.00
DevOps	Arquitecto DevOps	610	S/ 7,500.00	S/ 46.88	S/ 28,593.75
QA	QA Automación	120	S/ 4,000.00	S/ 25.00	S/ 3,000.00
Total		**1070**			**S/ 44,281.25**

Nota. Información al 12 de diciembre del 2022. Adaptados de "Salarios de puestos de tecnología en el Perú", por Glassdoor, s,f. (https://www.glassdoor.com/).

La estimación concluye que el costo por horas hombres en total es de S/ 44,281.25.

6.1.2 Costo de los recursos en Azure cloud

Se realizó una cotización utilizando la calculadora de Azure para obtener el costo mensual aproximadamente que incurriría los servicios a utilizar. A continuación, el detalle:

Tabla 31

Costo total – recursos Cloud

Service category	Service type	Region	Description	Estimated monthly cost
Analytics	Azure Machine Learning	East US 2	1 D4ds v4 (4 Core(s), 16 GB RAM) x 730 Hours, Pay as you go	$164.98
Compute	App Service	West US	Basic Tier; 1 B1 (1 Core(s), 1.75 GB RAM, 10 GB Storage) x 730 Hours; Windows OS; 0 SNI SSL Connections; 0 IP SSL Connections	$54.75
Databases	Azure Cosmos DB	East US	Standard provisioned throughput (manual), Always-free quantity disabled, Single Region Write (Single-Master) - East US (Write Region); 400 RU/s x 730 Hours; 0 GB transactional storage, 2 copies of periodic backup storage; Dedicated Gateway not enabled	$23.36
		Licensing Program	**Microsoft Customer Agreement (MCA)**	
		Total		**$243.09**

Nota: Adaptado por "Calculadora de Azure", Azure, s.f., (https://azure.microsoft.com/es-es/pricing/calculator/)

6.1.3 Costo total del proyecto

El costo final para realizar el proyecto, considerando el costo de los RRHH y los precios mensuales de los servicios en el entorno Cloud con el tipo de cambio (S/. 3.87) un total de S/. 45,222.01

Tabla 32

Costo total del proyecto

Nro.	Costo	Importe
1	Costo de RRHH	S/ 44,281.25
2	Costo Servicios Cloud (Mensual)	S/ 940.76
	Total	**S/ 45,222.01**

6.1.4 Estructuras de ganancias por cliente

Las empresas del rubro cuentan con un área de desarrollo de software para construir nuevos requerimientos y para brindar mantenimiento a los aplicativos. Se realizó una evaluación sobre los desarrollos realizados puestos en producción y cuánto cuesta las horas de soporte invertidas en los errores de producción.

A continuación, se presenta el número de errores encontrados en el ambiente de QA por un equipo durante un mes.

Tabla 33

Horas de soporte

Nro. de Equipo	**Nro. de Bugs**	**Horas de Soporte**	**Total, Horas**
1	183	16	2928

Considerando los costos de soporte invertidas por cada recurso y el valor de ello, se observa que resolver las incidencias conlleva un total de S/ 98,820.00.

Tabla 34

Costo del proyecto para la resolución de incidencias

Recurso	**Salario según Glassdoor**	**Costo por Hora**	**Costo Total**
Desarrollador	S/ 6,000.00	S/ 37.50	S/ 76,860.00
QA	S/ 4,000.00	S/ 25.00	S/ 21,960.00
	Total		**S/ 98,820.00**

La inversión de S/ 45,222.01 representa el 46% de lo que la organización gasta en horas de soporte de solo un equipo en resolver aproximadamente 183 errores en el ambiente de QA.

6.1.5 Cálculo de la VAN y TIR

Fórmula del valor actual neto (VAN)

Figura 47

Fórmula del valor actual neto (VAN)

$$VAN = -I_0 + \sum_{t=1}^{n} \frac{F_t}{(1+k)^t} = -I_0 + \frac{F_1}{(1+k)} + \frac{F_2}{(1+k)^2} + \cdots + \frac{F_n}{(1+k)^n}$$

Nota: Valor Actual Neto: ¿Qué es y cómo se calcula de manera correcta?, 2022, (https://economia3.com/valor-actual-neto/)

Donde:

Ft: son los flujos de dinero en cada periodo t

I0 es la inversión realiza en el momento inicial (t = 0)

n es el número de periodos de tiempo

k es el tipo de descuento o tipo de interés exigido a la inversión

Para nuestra propuesta se consideró los siguientes datos, los montos están en dólares:

- Flujos por cada año (USD 4674.10) equivale al 40% de ganancia esperada
- Inversión de USD 11685.27
- Se consideró 3 años
- La tasa de descuento del 8% según la constante planteada por el Ministerio de Economía y finanzas.

Se obtuvo el VAN con el monto de USD 360.36. y la TIR de 9.28%. lo cual evidencia la viabilidad y rentabilidad del proyecto.

6.2 Encuesta de satisfacción de la arquitectura propuesta

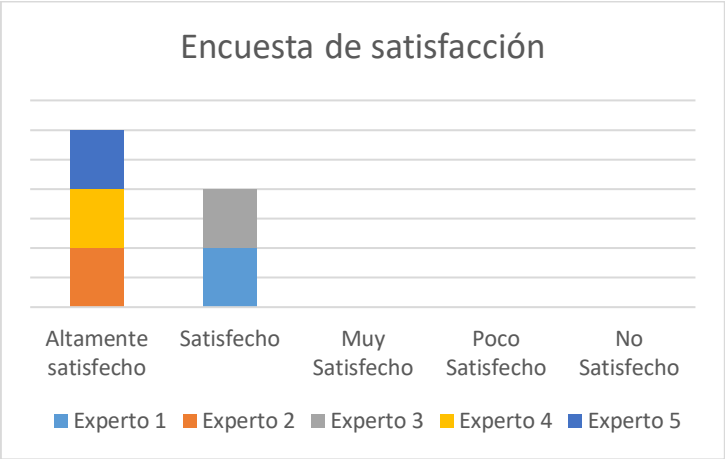
La última comunicación con los stakeholders del proyecto fue la presentación de la arquitectura de capas de la solución propuesta.

En base a ello, se finalizó la presentación realizando una encuesta de satisfacción a los diferentes roles del proyecto (2 QA Lead, 2 Backend Developer, 1 PO)

Como se muestra en la siguiente imagen, se obtuvo un grado de satisfacción alto por parte de nuestros expertos, lo cual, evidencia la viabilidad del proyecto.

Figura 48

Encuesta de Satisfacción de la Solución Planteada



Nota: Resultados de la encuesta de satisfacción sobre la arquitectura de la solución, fue realizada a través de un meeting a 5 expertos de la organización

CONCLUSIONES

Se concluye lo siguiente:

- Después de realizar el análisis y la revisión sistemática de la investigación, se concluye que el uso de algoritmos basados en la técnica fuzzing. funcionaron eficazmente en diferentes contextos del ámbito tecnológico, ya que, tienen la capacidad de identificar vulnerabilidades en el sistema e incrementar el porcentaje de cobertura de pruebas. Asimismo, aplicar fuzzing en pruebas de caja blanca es más eficaz que en caja negra, debido a la complejidad de la información de entrada y salida.
- El diseño de la arquitectura basada en capas fue desarrollado basado en las tecnologías que posee la organización, y su implementación permitirá generar escenarios de pruebas de las API de la aplicación. Para ello, se propone la implementación de una red neuronal que tenga conocimiento sobre la lógica de la aplicación, para permitir la selección de escenarios de prueba con el fin de que la data sea lo más real posible. Finalmente, la información estará segmentada y deberá ser llamada smart data.
- Como consecuencia de lo descrito en el informe, el líder técnico de desarrollo y de QA validan la viabilidad de realizar la implementación futura de la arquitectura propuesta, puesto que cumple con la infraestructura que posee la organización y con los recursos a utilizar en la nube. Asimismo, respaldan que las horas de soporte para atender errores en el ambiente productivo son costosos y reducir ello, sería beneficioso para la organización.
- La implementación de un reporte con los resultados una vez ejecutado el algoritmo, permitirá que los involucrados en el proyecto tengan trazabilidad y se realice una correcta gestión de los casos de pruebas y los errores encontrados por su ejecución.
- La integración de cloud computing es favorable para el desarrollo del proyecto, debido a que, existen recursos virtuales que permiten la implementación de una arquitectura de algoritmos, el uso de base de datos no relacionables y lo más importante, poder ejecutar el script de manera remota.

Se recomienda lo siguiente:

- La curva de aprendizaje para los nuevos integrantes del proyecto es de suma importancia, debido a que, es importante conocer el negocio y los flujos del sistema a tiempo para no impactar con los entregables del equipo.
- Existen limitaciones por parte del equipo de Microsoft para la creación de recursos en azure, sobre todo tiempos de demora en la gestión.
- Documentar los algoritmos utilizados y su integración con los componentes del sistema, para ellos se puede utilizar Confluence/Jira para ello.

GLOSARIO

- **Fuzz testing**: Es una técnica de pruebas utilizado frecuentemente en el mundo de la seguridad informática, tiene como objetivo detectar vulnerabilidades en el sistema a través, de algoritmos.
- **Azure**: Plataforma informática en la nube operada por Microsoft para la gestión de aplicaciones a través de centros de datos distribuidos en todo el mundo.
- **Dashboard**: Tablero para visualizar, analizar y extraer conclusiones de distintos indicadores determinados previamente.
- **API**: Las API son interfaces que permiten comunicar la capa de aplicación con la capa de datos de un sistema.

REFERENCIAS

Azure. (s.f.). *Calculadora de recursos de azure.* Azure. Recuperado el 15 de enero del 2023, de https://azure.microsoft.com/es-es/pricing/calculator/.

Chen, C., Baojiang, C., Jinxin M., Runpu, W., Jianchao, G. & Wenqian, G. (2018). A systematic review of fuzzing techniques. *Computers & Security, 75*, 118-137. https://doi.org/10.1016/j.cose.2018.02.002

Cheng, L., Yang, Z., Zhang, Y., Wu, C., Li, Z., Fu, Y. & Li, H. (2019). Optimizing Seed Inputs in Fuzzing with Machine Learning. *ICSE-Companion, 41*, 244-245. https://doi.org/10.1109/ICSE-Companion.2019.00096

Çınar, Z., Nuhu, A., Zeeshan, Q., Korhan, O., Asmael, M. & Safaei, B. Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing. *Industry 4.0. Sustainability, 12*(19), 8211. https://doi.org/10.3390/su12198211

Cummins, C., Petoumenos, P., Murray, A. & Leather, H. (2018). Compiler fuzzing through deep learning. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. *Association for Computing Machinery, 27*, 95–105. https://doi.org/10.1145/3213846.3213848

Eisele, M., Maugeri, M., Shriwas, R., Huth, C., & Bella, G. (2022). Embedded fuzzing: a review of challenges, tools, and solutions. *Cybersecurity, 5*(18), 03-15. https://doi.org/10.1186/s42400-022-00123-y

Finance, J. (2019). *Clasificación por el número de capas.* Tipos de redes neuronales (Clasificación). Recuperado el 20 de octubre de 2022, de https://inteligencia-artificial.dev/tipos-redes-neuronales/

Geetha, U., Sankar, S. & Sandhya, M. (2021). Acceptance testing based test case prioritization. *Cogent Engineering, 8*(1), 01-20. https://doi.org/10.1080/23311916.2021.1907013

Ghani, I., Wan, M., Wan-Kadir, N., Arbain, A. & Ibrahim, N. (2022). Improved test case selection algorithm to reduce time in regression testing. *Computers, Materials & Continua, 72*(1), 635-650. https://doi.org/10.32604/cmc.2022.025027

Gibson, S., Issac, B., Zhang L. & Jacob, S. M. (2020). Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms. *IEEE Access*, 8, 187914-187932. https://doi.org/10.1109/ACCESS.2020.3030751.

Glassdoor. (s.f.). *Salarios de puestos de tecnología en el Perú*. Salarios en Lima. Recuperado el 12 de diciembre del 2022, de https://www.glassdoor.com/

Great Learning Team. (s.f.). *What is Machine Learning? Defination, Types, Applications, and more*. Machine Learning: Definition, Types, Applications, and more Recuperado el 12 de enero del 2023, de https://www.mygreatlearning.com/blog/what-is-machine-learning.

Jitsunari, Y. & Arahori, Y. (2019). Coverage-Guided Learning-Assisted Grammar-Based Fuzzing. IEEE International Conference on Software Testing, Verification and Validation Workshops. *ICSTW*, *5*, 275-280. https://doi.org/10.1109/ICSTW.2019.00065.

Joffe, L. (2018). Machine Learning Augmented Fuzzing. *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 178-183, doi: https://doi.org/10.1109/ISSREW.2018.000-1

Karamcheti, S., Gideon, M. & Rosenberg, D. (2018). Adaptive Grey-Box Fuzz-Testing with Thompson Sampling. *AISec '18: Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, 37-44, doi: https://doi.org/10.1145/3270101.3270108.

Liu, X., Li, X., Prajapati, R., & Wu, D. (2019). DeepFuzz: Automatic Generation of Syntax Valid C Programs for Fuzz Testing. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(01), 1044-1051. https://doi.org/10.1609/aaai.v33i01.33011044

Madhavarapu, V., Bhattacharjee, S. & Dasy, S. (2022). A Generative Model for Evasion Attacks in Smart Grid. *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 1-6. https://doi.org/10.1109/INFOCOMWKSHPS54753.2022.9798325.

Microsoft. (s.f.). *Inteligencia artificial en el perímetro con Azure Stack Hub*. Arquitecturas de Azure. Recuperado el 12 de diciembre del 2022, de https://learn.microsoft.com/es-es/azure/architecture/solution-ideas/articles/ai-at-the-edge

Nasrabadi, Z., Parsa, S. & Kalaee, A. (2021). Format-aware learn&fuzz: deep test data generation for efficient fuzzing. *Neural Comput & Applic, 33*, 1497–1513. https://doi.org/10.1007/s00521-020-05039-7.

Olsthoorn, M. *(*2022). More Effective Test Case Generation with Multiple Tribes of AI. *IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings*, 286-290. https://doi.org/10.1145/3510454.3517066.

Qin, Y., Yue, C. (2022). Fuzzing-base hard-label black-box attacks against machine learning models. *Computers & Security, 117*, 01-13 https://doi.org/10.1016/j.cose.2022.102694

Santaella, J. (s,f). *Valor Actual Neto: ¿Qué es y cómo se calcula de manera correcta?* Economía3. Recuperado el 17 de enero del 2023, de https://economia3.com/valor-actual-neto/

Serpanos, D. & Katsigiannis, K. (2021). Fuzzing: Cyberphysical System Testing for Security and Dependability. *Computer*, *54*(9), 86-89. https://doi.org/10.1109/10.1109/MC.2021.3092479.

She, D., Krishna, R., Yan, L., Jana, S. & Ray. B. (2020). MTFuzz: fuzzing with a multi-task neural network. *In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, 737–749. https://doi.org/10.1145/3368089.3409723

Universal CPA Review. (s.f.). *Neural Networks*. What is a neural network in accounting and finance? Recuperado el 10 de enero del 2023, de https://www.universalcpareview.com/ask-joey/what-is-a-neural-network-in-accounting-and-finance/).

Wang, Y., Jia, P., Liu, L., Huang, C. & Liu, Z. (2020) A systematic review of fuzzing based on machine learning techniques. *PLoS ONE 15*(8), 01-35. https://doi.org/10.1371/journal.pone.0237749

Xu, H., Wang, Y., Fan, S., Xie, P. & Liu, A. (2020). DSmith: Compiler Fuzzing through Generative Deep Learning Model with Attention. *International Joint Conference on Neural Networks (IJCNN)*, 1-9. https://doi.org/10.1109/IJCNN48605.2020.9206911.

Zhang, P., Ren, B., Dong, H. & Dai, Q. (). CAGFuzz:Coverage-Guided Adversarial Generative Fuzzing Testing for Image-based Deep Learning Systems. *IEEE Transactions on Software Engineering, 48*(11), 4630-4646. https://doi.org/10.1109/TSE.2021.3124006.