UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Vulnerabilidad de redes de paquetes software II

Presentado por Daniel Alonso Báscones
en Universidad de Burgos — 7 de junio de 2023
Tutor: Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Daniel Alonso Báscones, con DNI 71298886J, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 7 de junio de 2023

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Las redes y la teoría de grafos

Las redes, como estructuras que representan interconexiones entre entidades, son objeto de estudio de la teoría de grafos. Estas redes se componen de nodos o vértices conectados por enlaces o aristas. La teoría de grafos, por su parte, se encarga de analizar matemáticamente estas redes y proporcionar herramientas para comprender sus propiedades y comportamientos.

Los grafos son modelos abstractos que representan las relaciones y conexiones entre entidades mediante nodos y aristas. Estos modelos tienen aplicaciones en diversos campos, como la informática, la física, la biología y las ciencias sociales. Se utilizan para comprender fenómenos complejos, analizar la propagación de información, estudiar interacciones sociales y examinar las redes de transporte, entre otros aspectos.

En el campo de la teoría de grafos, destacan importantes autores que han contribuido significativamente a su desarrollo. Leonhard Euler es considerado el fundador de la teoría de grafos, gracias a su trabajo sobre el problema de los puentes de Königsberg en el siglo XVIII. Otros destacados autores incluyen a Paul Erdős, quien realizó contribuciones fundamentales a la teoría de grafos combinatorios, y Claude Shannon, quien aplicó la teoría de grafos en la teoría de la información. Estos investigadores han sentado las bases para el estudio y aplicación de la teoría de grafos en diversas disciplinas.

1.2. Los repositorios de paquetes de software

En el ámbito del desarrollo de software, los repositorios de paquetes desempeñan un papel fundamental al ofrecer un entorno centralizado donde

los desarrolladores pueden acceder, compartir y distribuir bibliotecas de código predefinidas. Estos repositorios están específicamente diseñados para diferentes plataformas y lenguajes de programación, proporcionando a los desarrolladores un acceso conveniente a una amplia gama de recursos.

A lo largo de los años, han surgido numerosos repositorios de paquetes de software para diversas plataformas y lenguajes. Por ejemplo, en el ámbito de la bioinformática, Bioconductor destaca como un repositorio importante que se centra en paquetes y herramientas para el análisis de datos genómicos. En el ecosistema de Python, PyPI (Python Package Index) es un repositorio central que alberga una gran cantidad de paquetes para una amplia variedad de aplicaciones y bibliotecas.

En el panorama actual, los repositorios de paquetes de software siguen siendo vitales para la comunidad de desarrollo. Proporcionan a los desarrolladores un acceso rápido y sencillo a una amplia gama de funcionalidades y bibliotecas de código predefinidas, lo que les permite acelerar el desarrollo de aplicaciones y proyectos. Además, estos repositorios fomentan la colaboración y el intercambio de código entre los desarrolladores, promoviendo un ecosistema de desarrollo más dinámico y eficiente.

1.3. Los gestores de paquetes de software

Los gestores de paquetes de software nos proporcionan herramientas y funcionalidades para gestionar la instalación, actualización y eliminación de bibliotecas y dependencias de un proyecto. Estos gestores se encuentran diseñados específicamente para diferentes plataformas y lenguajes de programación, brindando a los desarrolladores una forma eficiente de administrar y distribuir el código.

Entre los gestores de paquetes más populares, se destaca pip en el ecosistema de Python, que permite instalar y administrar fácilmente las bibliotecas necesarias para un proyecto Python. Por otro lado, mvn (Maven) es ampliamente utilizado en el mundo de Java para gestionar las dependencias y configuraciones de proyectos. Cada gestor de paquetes cuenta con su propia sintaxis y funcionalidades específicas, pero todos comparten el objetivo común de simplificar la gestión de bibliotecas y asegurar la resolución de dependencias.

En el panorama actual, los gestores de paquetes de software continúan desempeñando un papel crucial en el desarrollo de software. Proporcionan a los desarrolladores una forma conveniente y eficiente de administrar las

bibliotecas y dependencias necesarias para sus proyectos, lo que les permite centrarse en la implementación de funcionalidades sin preocuparse por la instalación manual y la gestión de las dependencias.

Además, estos gestores mejoran la reutilización de código de la comunidad y la colaboración, ya que permiten compartir y distribuir fácilmente bibliotecas y proyectos entre desarrolladores. También facilitan la actualización y el mantenimiento de las dependencias, asegurando que los proyectos estén siempre actualizados y protegidos contra vulnerabilidades conocidas.

1.4. Las redes de dependencias de paquetes de software

Una red de dependencias de paquetes software es un conjunto de conexiones entre diferentes paquetes o bibliotecas utilizadas en el desarrollo de software. Cada paquete depende de otros para funcionar correctamente, formando una red interdependiente.

En este contexto, cada paquete tiene requisitos específicos que deben cumplirse para que funcione correctamente. Estos requisitos son satisfechos por otros paquetes, creando conexiones que permiten que el software se comporte como se espera. Si alguna de estas conexiones se rompe o no se cumple, puede producirse un fallo en el funcionamiento del software.

La red de dependencias puede ser muy compleja, ya que un solo paquete puede depender de múltiples otros paquetes, y estos, a su vez, pueden tener sus propias dependencias. Es como un tejido intrincado en el que cada hilo está conectado con otros, formando una red interconectada.

1.5. Los repositorios de paquetes de software desde la perspectiva de la teoría de grafos

Al estudiar las dependencias de paquetes como grafos, obtenemos una visión más clara de las complejas relaciones que existen en nuestros proyectos. Podemos identificar los paquetes esenciales que sostienen todo el sistema y comprender cómo un cambio en uno de ellos puede afectar a otros. Esto nos ayuda a tomar decisiones informadas y anticiparnos a posibles problemas.

Además, la teoría de grafos nos permite detectar dependencias redundantes o ciclos indeseados en nuestra red de paquetes. Podemos evaluar la estabilidad y mantenibilidad del proyecto, analizando el impacto de agregar o eliminar un paquete en toda la red. Esta comprensión más profunda nos ayuda a optimizar y fortalecer nuestro código.

Afortunadamente existen herramientas y bibliotecas que nos proporcionan esta informacion de las redes. Con ellas, y usando el enfoque de la teoria de grafos, podemos ver de una forma más clara cómo se entrelazan los paquetes entre sí y cómo se relacionan con el resto de la red.

La aplicación de la teoría de grafos a las redes de dependencias de paquetes software nos permite tomar decisiones más informadas, comprender mejor las implicaciones de las dependencias y colaborar de manera más efectiva con nuestro equipo de desarrollo. Es una forma poderosa de optimizar y mejorar nuestros proyectos y mantener un ecosistema saludable en los gestores de paquetes.

1.6. OLIVIA: Open-source Library Indexes Vulnerability Identification and Analysis

OLIVIA, desarrollada por el alumno Daniel Setó Rey como parte de su Trabajo de Fin de Grado en la Universidad de Burgos y tutorizada por los profesores Carlos López Nozal y Jose Ignacio Santos Martín en 2021, es una herramienta de código abierto que se centra en la identificación y análisis de defectos en bibliotecas de software desde la perspectiva de la teoría de grafos. Estos defectos pueden provocar errores funcionales, problemas de rendimiento e incluso problemas de seguridad. Para los desarrolladores, comprender completamente el riesgo es complicado, ya que solo importan explícitamente una pequeña parte de las dependencias utilizadas en sus proyectos.

OLIVIA utiliza un enfoque basado en la vulnerabilidad de la red de dependencias de los paquetes de software para medir la sensibilidad del repositorio a la introducción aleatoria de defectos. Su objetivo es contribuir a la comprensión de los mecanismos de propagación de defectos en el software y estudiar estrategias factibles de protección.

En la actualidad, OLIVIA está en proceso de ser publicado a nivel académico. Después de su desarrollo como proyecto de fin de carrera, se

están realizando los esfuerzos necesarios para compartir sus resultados y contribuciones con la comunidad científica. Esta publicación permitirá que otros investigadores y profesionales del campo accedan a esta herramienta y se beneficien de su enfoque innovador en la identificación y análisis de vulnerabilidades en las bibliotecas de software. Además, sentará las bases para futuros avances en la comprensión de los mecanismos de propagación de defectos y la implementación de estrategias efectivas de protección en el desarrollo de software.

En este trabajo de fin de grado, no se profundiza en el modelo matemático subyacente de OLIVIA. Sin embargo, es importante tener conocimientos básicos de la teoría de grafos para comprender su funcionalidad. La teoría de grafos proporciona un marco conceptual para comprender las interconexiones y relaciones entre los paquetes de software, así como la propagación de posibles defectos en el ecosistema. Aunque no se abordan los aspectos matemáticos en detalle, tener una comprensión general de los grafos nos ayuda a apreciar y comprender mejor la utilidad y las implicaciones de OLIVIA en la identificación y análisis de vulnerabilidades en las bibliotecas de software.

2. Objetivos del proyecto

- Dar soporte a OLIVIA al proporcionarle los datos necesarios para enriquecer su modelo y seguir una estructura de datos similar que asegure la integración con OLIVIA.
- Proporcionar datos actualizados que mejoren su rendimiento y precisión, asegurando que OLIVIA esté equipado con los datos más recientes para su análisis y detección de defectos. Al proporcionar esta fuente constante de información actualizada, fortaleceremos la capacidad de OLIVIA para ofrecer resultados precisos y confiables, respaldando así su utilidad en la identificación y análisis de vulnerabilidades en bibliotecas de software.
- Realizar un análisis de los principales repositorios de software propuestos en OLIVIA, con el fin de buscar estrategias viables para extraer los datos de los paquetes que contienen. A través de este análisis, nos enfocaremos en identificar métodos eficientes y efectivos para obtener la información relevante de cada repositorio, considerando las restricciones y peculiaridades de cada plataforma.
- Enriquecer mediante la aportación de nuevos datos no disponibles anteriormente y usar fuentes de información adicionales.
- Publicar los datos obtenidos a través de nuestra investigación para que sirvan como referencia y recurso para investigaciones futuras. Al hacer públicos los datos recopilados en nuestro estudio sobre los repositorios de software, esperamos que otros investigadores puedan aprovecharlos y utilizarlos como base para sus propias investigaciones.
- Realizar una comparativa de la evolución de los paquetes y el estado de los repositorios tras el paso del tiempo.

3. Conceptos teóricos

En aquellos proyectos que necesiten para su comprensión y desarrollo de unos conceptos teóricos de una determinada materia o de un determinado dominio de conocimiento, debe existir un apartado que sintetice dichos conceptos.

Algunos conceptos teóricos de LaTeX [1].

3.1. Secciones

Las secciones se incluyen con el comando section.

Subsecciones

Además de secciones tenemos subsecciones.

Subsubsecciones

Y subsecciones.

3.2. Referencias

Las referencias se incluyen en el texto usando cite [3]. Para citar webs, artículos o libros [2], si se desean citar más de uno en el mismo lugar [1, 2].

[1] Créditos a los proyectos de Álvaro López Cantero: Configurador de Presupuestos y Roberto Izquierdo Amo: PLQuiz

3.3. Imágenes

Se pueden incluir imágenes con los comandos standard de LaTeX, pero esta plantilla dispone de comandos propios como por ejemplo el siguiente:

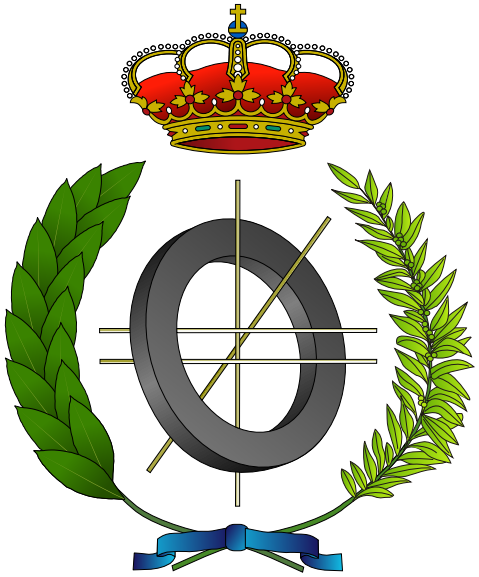

Figura 3.1: Autómata para una expresión vacía

3.4. Listas de items

Existen tres posibilidades:

- primer item.
- segundo item.

1. primer item.
2. segundo item.

Primer item más información sobre el primer item.

Segundo item más información sobre el segundo item.

-

Herramientas	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
JavaScript	X			
AngularJS	X			
Bower	X			
PHP		X		
Karma + Jasmine	X			
Slim framework		X		
Idiorm		X		
Composer		X		
JSON	X	X		
PhpStorm	X	X		
MySQL			X	
PhpMyAdmin			X	
Git + BitBucket	X	X	X	X
MikTEX				X
TEXMaker				X
Astah				X
Balsamiq Mockups	X			
VersionOne	X	X	X	X

Tabla 3.1: Herramientas y tecnologías utilizadas en cada parte del proyecto

3.5. Tablas

Igualmente se pueden usar los comandos específicos de LATEXo bien usar alguno de los comandos de la plantilla.

4. Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

6. Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Zachary J Bortolot and Randolph H Wynne. Estimating forest biomass using small footprint lidar data: An individual tree-based approach that incorporates training data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(6):342–360, 2005.

[2] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[3] Wikipedia. Latex — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=LaTeX&oldid=84209252, 2015. [Internet; descargado 30-septiembre-2015].