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 — 18 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, 18 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

Uno de los principios más importantes de la ingeniería del software es aprovechar al máximo los componentes existentes [14]. El empleo de librerías externas para disminuir los tiempos de desarrollo y costes es prácticamente universal, en todos los lenguajes y tipos de proyecto software. En una encuesta realizada por Sonartype en 2014, más de once mil arquitectos y desarrolladores revelaron que el 90 % del código de una aplicación típica en la actualidad está compuesto por componentes externos. Además, más del 80 % de los proyectos utilizan repositorios centralizados de componentes. Estos repositorios están configurados por defecto en las herramientas que usamos para instalar y administrar los paquetes de cada lenguaje. Son como almacenes gigantes que contienen componentes los cuales aportan alguna utilidad. Algunos ejemplos famosos son PyPI (Python Package Index), npm (Node Package Manager) para JavaScript (Node.js), Packagist para PHP, CPAN (Comprehensive Perl Archive Network) y Maven Central para Java.

El problema reside en que hay veces que utilizar componentes externos tiene sus riesgos. Esos riesgos pueden ser difíciles de notar para los desarrolladores, ya que solo importan explícitamente una pequeña parte de las librerías que se incluyen en cada proyecto.

Ilustraremos la afirmación anterior con un suceso que hizo temblar npm. En marzo de 2016, un desarrollador llamado Azer Koçulu, de California, recibió un correo de unos abogados de Kik Interactive. Le pedían amablemente que cambiara el nombre de su librería llamada Kik, que había publicado en npm. Resulta que los abogados decían que la marca "Kik Messenger"les pertenecía a sus representados [5] por derechos de propiedad intelectual. Koçulu no se quedó de brazos cruzados, se negó rotundamente a cambiar el nombre. Los abogados, entonces, se pusieron en contacto con Isaac Schuleter, el director de npm, quien accedió a transferir el nombre.

Koçulu muy decepcionado con la decisión tomó medidas, y eliminó más de 250 paquetes del repositorio que habían sido desarrollados por él.

Entre los paquetes eliminados por Koçulu había uno llamado "left-pad", que era un simple script para justificar cadenas de texto. Pero... Muchos de los paquetes más populares de npm dependían directa o indirectamente de "left-pad". Casi un millón de sitios web comenzaron a tener problemas para actualizar sus dependencias o lanzar nuevas versiones, e incluso gigantes como Facebook o Netflix se vieron afectados. Fue un verdadero caos.

```
module.exports = leftpad;
function leftpad (str, len, ch) {
    str = String(str);
    var i = -1;
    if (!ch && ch !== 0) ch = ' ';
    len = len - str.length;
    while (++i < len) { str = ch + str; }
    return str;
}
```

Finalmente, npm tuvo que intervenir y, aunque generó mucha polémica, decidieron restaurar el paquete "left-pad". Laurie Voss, el director de tecnología de npm, explicó que tomaron esta decisión sin precedentes debido a la gravedad y la amplia repercusión del problema. No lo hicieron a la ligera, la decisión se tomó pensando en la comunidad de usuarios de npm y en sus necesidades.

Después del incidente, se hizo evidente lo frágiles que son los sistemas de paquetes en los que depende una gran parte del desarrollo de software en todo el mundo. Surgieron muchas preguntas sobre el uso indiscriminado de librerías externas [2] y el modelo de gestión centralizada. Es como darse cuenta de lo mucho que dependemos de ciertas cosas y preguntarnos si estamos siendo prudentes al hacerlo. La problemática no se limita solo a la retirada voluntaria de un paquete en particular, como sucedió con left-pad. También puede ser causada por todo tipo de defectos en el software o ataques maliciosos que se propagan a través de la estructura de dependencias. Según un informe llamado "The state of open source security report", las vulnerabilidades de seguridad en el software de código abierto prácticamente se duplican cada dos años, y aproximadamente el 80 % de estas vulnerabilidades se introducen a través de dependencias indirectas. Las conclusiones de este informe son preocupantes, e hicieron que el estudio de la seguridad y estabilidad de estos

repositorios, que antes no había recibido mucha atención se pusiese en el punto de mira de la comunidad [1] [9] [4].

Es aquí donde entra la ciencia de redes a jugar un papel importante en el mundo del software. Dado que las dependencias entre los proyectos de software, especialmente entre los paquetes de un repositorio, se pueden representar como una red o un grafo dirigido y analizar desde una perspectiva estructural, la ciencia de las redes se ha convertido en un enfoque reciente para comprender mejor la estructura y evolución [10] [5] de los repositorios de paquetes de software. Es como mirar la forma en que todas estas piezas se conectan y se influyen mutuamente. La ciencia de redes es una disciplina que se dedica a estudiar las redes complejas que podemos encontrar en muchos campos como la tecnología, la biología y las ciencias sociales. Cuando decimos çomplejas"nos referimos al tamaño de las estructuras que se investigan y a la falta de un patrón obvio de relaciones entre sus elementos. En los modelos de ciencia de redes, representamos estos elementos como nodos o vértices, mientras que las conexiones entre ellos se simbolizan con enlaces o aristas. No es de sorprender que la ciencia de redes se base fundamentalmente en la teoría matemática de grafos, pero no es la única ciencia que interviene en este análisis. Es una materia interdisciplinaria que incorpora conocimientos de estadística, teorías de control e información, algoritmos, minería de datos y muchas otras áreas de la informática. Los principales logros de la ciencia de redes se centran en la construcción de modelos que explican la síntesis y evolución de las redes reales. También se estudia la centralidad o importancia de los nodos, es decir, qué nodos son los más cruciales dentro de una red. Además, se investiga la detección de agrupaciones significativas, como clústeres y comunidades, que revelan patrones interesantes dentro de una red. Y por si fuera poco, se exploran fenómenos de difusión, es decir, cómo se propaga la información a través de las redes.

Alexandre Decan junto a otros investigadores [7], realizaron un estudio en el que compararon el número de dependencias directas y transitivas, así como el número de componentes débilmente conectados en los repositorios CRAN, PyPI y npm. En otros estudios posteriores [11], sugieren que las estrategias actuales de versionado de paquetes pueden causar problemas de falta de robustez en los repositorios. Esta conclusión fue demostrada experimentalmente en un análisis de CRAN [8], apareciendo un patrón de comportamiento que a priori se mantiene común en los repositorios de software. Por si eso no fuera suficiente, otro estudio [15] analiza la evolución del número de dependencias y la importancia relativa de cada componente en npm. Utiliza PageRank como métrica para evaluar la centralidad de los paquetes y tienen en cuenta cómo se utilizan en proyectos de GitHub.

Otro aspecto que se estudia en la ciencia de redes es la vulnerabilidad de la red. La vulnerabilidad se refiere a la incapacidad de la red para funcionar correctamente cuando se eliminan algunos nodos o enlaces importantes [13]. Para medir la vulnerabilidad, necesitamos conocer el tipo de red en particular. Por lo general, distinguimos entre casos en los que los problemas ocurren de manera aleatoria o cuando son ataques intencionales [3]. En el último caso, se supone que los atacantes utilizan información de la red para planificar estrategias que provoquen el mayor daño posible o maximicen algún objetivo que generalmente va en contra de los intereses de los actores de la red. En cuanto a los repositorios de paquetes, un estudio [15] revela que la eliminación de un solo paquete puede afectar al 30 % de los paquetes y aplicaciones en los ecosistemas de JavaScript, Rust y Ruby. Otro estudio [9] concluye que el número de vulnerabilidades encontradas en el código fuente de los paquetes de npm ha estado aumentando constantemente desde 2012, propagándose a través de la red de dependencias.

Las redes de dependencias de paquetes presentan características distintivas en comparación con las redes comúnmente estudiadas en la ciencia de redes, como las redes sociales, de información, tecnológicas o biológicas [12]. En primer lugar, los enlaces en estas redes pueden tener múltiples interpretaciones dentro de la misma red. Por un lado, actúan como canales de comunicación o difusión, similar a cómo ocurre en muchas redes sociales (lo cual explica, por ejemplo, la propagación de errores en la red). Por otro lado, representan requisitos necesarios para el funcionamiento de los nodos, pero de una manera diferente a otras redes tecnológicas. Tomemos como ejemplo una red eléctrica o de comunicaciones, donde los enlaces entrantes son necesarios para el funcionamiento de los nodos en la red. Sin embargo, en general, un solo enlace es suficiente para establecer una conexión entre un nodo y el conjunto de la red. En cambio, en las redes de dependencias, todos los enlaces son necesarios para el funcionamiento del nodo. Esto se debe a su naturaleza inherentemente transitiva, lo que significa que las relaciones son transitivas por definición, sin necesidad de que esta propiedad se exprese explícitamente en la estructura de la red. Esta característica es poco común en otros tipos de redes. Además, es posible considerar una red de dependencias como una red de información, donde los enlaces representan votos, en el sentido de que reflejan algún tipo de reconocimiento, al igual que los enlaces en Internet o las citas en las redes bibliográficas. En este sentido, las redes de dependencias comparten la propiedad de poder ser modeladas mediante una estructura acíclica sin introducir restricciones importantes, al igual que ocurre con las redes bibliográficas. La investigación sobre las redes de dependencias de paquetes es un campo relativamente nuevo y nuestra

comprensión de estas estructuras es limitada [10]. El conjunto de trabajos de referencia en este tema es aún reducido, y lo es aún más si nos centramos en aspectos específicos como el análisis de la vulnerabilidad, donde solo hemos encontrado publicaciones descriptivas. Hasta donde se sabe, no se ha propuesto un modelo teórico que permita avanzar en la comprensión de la vulnerabilidad de este tipo de redes, lo cual podría tener un impacto significativo en la seguridad y estabilidad del software a nivel global. Es necesario realizar más investigaciones en este campo para mejorar nuestra comprensión y desarrollar enfoques teóricos que aborden eficazmente los desafíos relacionados con la seguridad de las redes de dependencias de paquetes. En este trabajo, nos centraremos en los aspectos distintivos de las redes de dependencias de paquetes, como las múltiples interpretaciones de la red, las relaciones inherentemente transitivas y la posibilidad de modelarlas como redes acíclicas. Nuestro objetivo será estudiar la vulnerabilidad de estas redes frente a fallos. Definiremos la función de la red como la capacidad de proporcionar de manera ininterrumpida un conjunto de componentes de software reutilizables, de calidad y que funcionen según sus especificaciones. A través de nuestra investigación, intentaremos responder a preguntas como:

- ¿Hasta qué punto se ven afectados los proyectos software que emplean el repositorio ante la incidencia de fallos aleatorios (como por ejemplo la introducción de errores en los componentes)?
- ¿Cuál es la sensibilidad de una red determinada ante estas situaciones?
- ¿Existen estrategias óptimas para reducir el efecto perjudicial de los fallos en la red?

Por lo tanto, nuestra contribución consistirá en desarrollar un modelo teórico que permita analizar la vulnerabilidad frente a fallos en las redes de dependencias de paquetes. Este modelo se utilizará para implementar un conjunto de herramientas que ayudarán a responder las preguntas planteadas anteriormente. Estas herramientas se aplicarán a una captura de la estructura de dependencias de varios repositorios de paquetes reales. Por último, se realizará un análisis de los resultados obtenidos para obtener conclusiones relevantes. Los resultados de estos estudios proporcionan a la comunidad una visión muy poco conocida y novedosa desde el punto de vista de la redes de dependencias de paquetes, siendo como mínimo de interés académico para: Gestores centralizados de paquetes, para establecer políticas y procesos de control manuales o automáticos que mejoren la seguridad y estabilidad de

los repositorios. Desarrolladores de software en general, para valorar los diferentes riesgos introducidos por las dependencias empleadas en sus proyectos, y a los desarrolladores de paquetes en particular para comprender mejor su responsabilidad sobre el ecosistema. Desarrolladores de herramientas de calidad e integración continua, para definir cuantitativamente el concepto de vulnerabilidad en función del modelado de la red de dependencias de paquetes.

OLIVIA [6], 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.

Este TFG pretende ser una continuación de la labor realizada por su predecesor. Una vez que tenemos el modelo de análisis definido, lo que se necesitan son los datos a analizar mediante este. La obtención de un conjunto de datos actualizado y el análisis a bajo nivel de los mismos, han aportado una actualización de los resultados experimentales obtenidos.

La recopilación de datos sobre las dependencias de los paquetes de software es un desafío complejo que puede resultar difícil de abordar. Uno

de los principales problemas radica en la falta de una única fuente confiable de información sobre estas dependencias. En muchos casos, no existe un repositorio centralizado o una base de datos completa que contenga todos los detalles necesarios. Debido a esta falta de una fuente única, recopilar datos sobre las dependencias de los paquetes de software a menudo requiere un esfuerzo manual y exhaustivo. Los desarrolladores y analistas deben investigar y rastrear las dependencias de cada paquete individualmente, lo que puede llevar una cantidad considerable de tiempo y recursos. Además, la información sobre las dependencias de los paquetes de software puede dispersarse en diferentes fuentes, como documentación oficial, repositorios de código, foros de desarrolladores y otras fuentes en línea.

Esta dispersión puede dificultar aún más la recopilación de datos y aumentar la posibilidad de omitir o malinterpretar información relevante. El análisis de las redes de dependencias de paquetes de software es un proceso complejo debido a la naturaleza de estas redes, que pueden ser enormes y altamente interconectadas. Estas redes pueden contener miles o incluso millones de nodos y conexiones, lo que hace que el análisis manual sea prácticamente imposible. Por lo tanto, se requieren herramientas especializadas y experiencia en análisis de datos para abordar este desafío.

Otro desafío asociado con la recopilación de datos es mantener la información actualizada. Las dependencias de los paquetes de software pueden cambiar con el tiempo debido a actualizaciones, nuevas versiones o cambios en los requisitos del sistema. Por lo tanto, es crucial realizar un seguimiento constante de los cambios y actualizar la información de las dependencias de manera regular para garantizar la precisión de los datos recopilados.

Para abordar estos desafíos, y como uno de los principales puntos fuertes de este TFG se pretende ilustrar al lector de cómo poder enfrentarse a esta tarea, identificando las distintas vías de obtención de datos, y concluyendo con el desarrollo de una herramienta que facilita el proceso de obtención de los mismos desde los repositorios oficiales de los gestores de paquetes que hemos elegido como caso de estudio. En concreto nos referimos a la recopilación de datos de las dependencias en los paquetes de R (Bioconductor y CRAN), python (Pypi) y javascript (NPM)

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

3.1. 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.

3.2. 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.

3.3. 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.

3.4. Redes de dependencias de paquetes de software

Una red de dependencias de paquetes de software consiste en un grafo en el que se representan las relaciones de dependencias entre paquetes de software. En una red de dependencias, cada nodo representa un paquete de software, y cada enlace representa una relacion de dependencia entre dos paquetes. Existe una dependencia entre dos paquetes si un paquete requiere del otro para funcionar.

Importancia de las redes de dependencias de paquetes de software

En primer lugar, estas redes se pueden utilizar para identificar posibles problemas en proyectos de software, ya que cuando se actualiza un paquete, puede introducir cambios incompatibles que afecten a otros paquetes en la red.

Desde un punto de vista de control de calidad, las redes de dependencias de paquetes de software se pueden utilizar para mejorar la calidad de los proyectos de software. Por ejemplo, al analizar las dependencias entre paquetes, los desarrolladores pueden identificar áreas potenciales de mejora. Por ejemplo, pueden identificar paquetes que ya no son necesarios o que están causando problemas.

Además cabe destacar que se pueden utilizar para hacer que los proyectos de software sean más seguros, ya que, al analizar las dependencias entre paquetes, se pueden identificar posibles vulnerabilidades de seguridad o que son utilizados con frecuencia con fines maliciosos.

Por último, permiten gestionar eficientemente las actualizaciones de software. Al comprender cómo los cambios en un paquete pueden afectar a otros, los equipos de desarrollo pueden evaluar el impacto potencial de las actualizaciones y tomar decisiones informadas sobre cuándo y cómo implementarlas.

Necesidades de las redes de dependencias de paquetes de software

Recopilacion de datos

La recopilación de datos sobre las dependencias de los paquetes de software es un desafío complejo que puede resultar difícil de abordar. Uno de los principales problemas radica en la falta de una única fuente confiable de información sobre estas dependencias. En muchos casos, no existe un repositorio centralizado o una base de datos completa que contenga todos los detalles necesarios.

Debido a esta falta de una fuente única, recopilar datos sobre las dependencias de los paquetes de software a menudo requiere un esfuerzo manual y exhaustivo. Los desarrolladores y analistas deben investigar y rastrear las dependencias de cada paquete individualmente, lo que puede llevar una cantidad considerable de tiempo y recursos. Además, la información sobre las dependencias de los paquetes de software puede dispersarse en diferentes fuentes, como documentación oficial, repositorios de código, foros de desarrolladores y otras fuentes en línea. Esta dispersión puede dificultar aún más la recopilación de datos y aumentar la posibilidad de omitir o malinterpretar información relevante.

Otro desafío asociado con la recopilación de datos es mantener la información actualizada. Las dependencias de los paquetes de software pueden cambiar con el tiempo debido a actualizaciones, nuevas versiones o cambios en los requisitos del sistema. Por lo tanto, es crucial realizar un seguimiento constante de los cambios y actualizar la información de las dependencias de manera regular para garantizar la precisión de los datos recopilados. Para abordar estos desafíos, se han desarrollado herramientas y técnicas específicas. Algunas soluciones automatizadas, como analizadores de dependencias y herramientas de gestión de paquetes, pueden ayudar a simplificar el proceso

de recopilación de datos al extraer automáticamente información sobre las dependencias de los paquetes de software. Sin embargo, incluso con estas herramientas, es posible que se requiera intervención manual para verificar y corregir posibles discrepancias o lagunas en los datos recopilados

Análisis de datos

El análisis de las redes de dependencias de paquetes de software es un proceso complejo debido a la naturaleza de estas redes, que pueden ser enormes y altamente interconectadas. Estas redes pueden contener miles o incluso millones de nodos y conexiones, lo que hace que el análisis manual sea prácticamente imposible. Por lo tanto, se requieren herramientas especializadas y experiencia en análisis de datos para abordar este desafío.

Una de las principales dificultades del análisis de datos en las redes de dependencias de paquetes de software es ser capaz de poder trabajar con la masividad de estas y poder visualizar como son. Dado que las redes pueden ser muy grandes, es esencial utilizar herramientas de visualización que permitan comprender la estructura y las interconexiones de manera más sencilla. Estas herramientas ayudan a identificar patrones, clusters y dependencias importantes dentro de la red.

Además, la complejidad de las redes de dependencias también implica la necesidad de algoritmos y técnicas de análisis avanzados. Estos algoritmos pueden abordar desafíos como la detección de comunidades de paquetes, la identificación de paquetes críticos o de alto impacto, la detección de ciclos o bucles de dependencia y la identificación de flujos de información crítica.

Si nos enfocamos en el punto de vista de calidad y robustez de la red, es necesario medir la estabilidad de las dependencias y evaluar los posibles puntos de falla o vulnerabilidades en la red. Estas métricas nos permiten identificar posibles puntos débiles en la infraestructura de software.

Actualizaciones y volatilidad

Las actualizaciones del código fuente son una parte crucial y muy frecuente, ya que proporcionan mejoras, correcciones de errores y nuevas funcionalidades. Sin embargo, estas actualizaciones también pueden tener un impacto significativo en las dependencias entre paquetes de software.

Cuando se realiza una actualización en un paquete de software, es posible que se modifiquen los requisitos o las dependencias del mismo. Por ejemplo, una actualización puede introducir una nueva versión de una biblioteca o

un componente, lo que podría requerir que otros paquetes también se actualicen para mantener la compatibilidad. Esto puede afectar a las conexiones existentes en la red de dependencias de paquetes de software.

Es fundamental que las redes de dependencias se actualicen regularmente para reflejar estos cambios. Esto implica realizar un seguimiento de las actualizaciones de cada paquete y revisar que todo funcione como se espera. Sería conveniente y una buena práctica para los mantenedores de la red, que esta esté alineada con las versiones y requisitos actualizados de los paquetes individuales.

Como consecuencia de mantener las redes actualizadas, se asegura que los desarrolladores tengan una visión más cercana a la evolución tecnológica del software lo que facilita la toma de decisiones informadas sobre futuras actualizaciones y mejoras en un determinado proyecto.

3.5. Analisis de redes de dependencias de paquetes de software

El análisis de redes es una disciplina que se enfoca en el estudio y comprensión de la estructura, interconexiones y propiedades de los sistemas complejos representados como redes. Estas redes pueden ser cualquier tipo de sistema compuesto por elementos interconectados (como redes sociales, carreteras, red eléctrica, o como en nuestro caso redes de dependencias de paquetes de software)

En el contexto de las redes de dependencias de paquetes de software, realizamos este análisis por medio de métricas. Las métricas son medidas cuantitativas que se utilizan para evaluar y describir diferentes aspectos de la red. Estas métricas proporcionan información clave sobre la importancia, la centralidad y las características de los nodos y las conexiones en la red.

Algunas de las métricas comunes en el análisis de redes de dependencias de paquetes de software incluyen el grado, que indica el número de conexiones que tiene un nodo. La centralidad de intermediación, que mide cuánto un nodo se encuentra en los caminos más cortos entre otros nodos. La centralidad de cercanía, que evalúa la distancia promedio entre un nodo y los demás nodos de la red. La centralidad de vector propio, que mide la importancia de un nodo basándose en la importancia de sus vecinos.

El grado

El grado de un nodo en una red es el número de aristas conectadas a ese nodo. En una red de dependencias de paquetes de software, el grado de un nodo representa el número de paquetes que dependen de ese paquete junto con las dependencias de este. En redes dirigidas se hace diferencia entre grado de entrada y grado de salida.

Un alto grado en la red de dependencias de paquetes de software revela el nivel de emparejamiento que posee un determinado paquete. Esta circunstancia puede tener consecuencias tanto positivas como negativas. Podría considerarse un indicio alentador cuando el paquete en cuestión ha sido rigurosamente probado y goza de una reputación fiable en términos de su desempeño. No obstante, también es importante reconocer que un alto grado puede encerrar ciertos riesgos, especialmente si el paquete se caracteriza por su complejidad o la presencia de errores.

La presencia de un alto grado de entrada en un paquete sugiere que una gran cantidad de otros paquetes dependen de él. Esta situación puede interpretarse como un signo de popularidad y amplio uso en el entorno del software. Cuando un paquete ha demostrado ser confiable, está bien mantenido y ha sido sometido a pruebas exhaustivas, su alto grado se convierte en una señal de que ha ganado la confianza de los desarrolladores y es considerado una opción sólida para satisfacer diversas necesidades funcionales.

Sin embargo, se debe considerar el riesgo potencial asociado a un alto grado de salida. Existe la posibilidad de que un alto grado de salida indique complejidad o presencia de errores en el paquete. Si un paquete es altamente complejo, es probable que su comprensión y mantenimiento sean una tarea dificil, lo que podría derivar en problemas de rendimiento, escalabilidad o incluso fallos en el sistema. Asimismo, si un paquete presenta errores o fallas, su alta dependencia implica que los problemas pueden propagarse rápidamente a otros paquetes, lo que afecta negativamente la estabilidad y confiabilidad del sistema en su conjunto.

Por lo tanto, resulta crucial considerar tanto el grado de un paquete como su calidad y confiabilidad. Un alto grado por sí solo no garantiza la calidad o idoneidad de un paquete, sino que debe evaluarse en conjunto con otros factores, como la presencia de test de funcionalidades, validación y pruebas, la existencia de documentación, el mantenimiento por parte de los desarrolladores y el feedback de la comunidad de desarrollo.

Centralidad de intermediación (Betweenness centrality)

La centralidad de intermediación mide la importancia de un nodo en una red según la frecuencia con la que se encuentra en el camino más corto entre otros dos nodos. En una red de dependencias de paquetes de software, la centralidad de intermediación se puede utilizar para identificar paquetes que son críticos para el funcionamiento general del sistema.

Esto implica que paquetes con alta centralidad de intermediación son puntos clave intermedios entre los paquetes de ese sistema. Identificar estos paquetes críticos puede ayudar a los desarrolladores a entender dónde se encuentran potenciales problemas intermedios o cuellos de botella y dónde pueden surgir problemas si esos paquetes fallan o se ven afectados.

Es por eso que debemos ser proactivos y estar alerta cuando se trata de este paquete con alta centralidad de intermediación. Es necesario establecer medidas de seguridad sólidas para protegerlo y garantizar su integridad. Esto implica llevar a cabo una supervisión constante, identificar posibles vulnerabilidades y aplicar actualizaciones y parches de seguridad de manera oportuna si fuese necesario para garantizar la calidad de la red.

Centralidad de cercanía (Closeness centrality)

La centralidad de cercanía mide la distancia media entre un nodo y todos los demás nodos de la red. En nuestro caso de estudio, la centralidad de cercanía se puede utilizar para identificar paquetes que son fácilmente accesibles desde otros paquetes.

Los paquetes con alta centralidad de cercanía son aquellos que están cerca de muchos otros paquetes en términos de distancia. Esto significa que son habitualmente usados por otros paquetes y pueden tener un impacto significativo en la propagación de información o cambios a través de la red de dependencias. Identificar estos paquetes puede ayudar a los desarrolladores a comprender dónde se encuentran los puntos clave de nexo común entre otros paquetes y cómo se propagan las dependencias en el sistema.

Como consecuencia, un paquete con alta cercanía debería de estar bien testado y documentado, ya que es fácilmente posible que sea dependencia en un proyecto software.

Centralidad de vector propio (Eigenvector centrality)

La centralidad de vector propio mide la importancia de un nodo en una red según la importancia de sus vecinos. En una red de dependencias de paquetes de software, la centralidad de vector propio se puede utilizar para identificar paquetes influyentes en la red.

Los paquetes con alta centralidad de vector propio son aquellos que están conectados a otros paquetes importantes en la red. Esto significa que su importancia se deriva de su conexión con otros paquetes influyentes. Identificar estos paquetes puede ayudar a los desarrolladores a comprender qué paquetes tienen un impacto significativo en la estructura y el funcionamiento de la red y que los cambios que se produzcan en estos paquetes pueden afectar a otros paquetes en la red.

Existen variaciones de esta métrica, como la métrica de Katz calcula la importancia de un nodo teniendo en cuenta tanto la cantidad como la calidad de sus conexiones. Se basa en la idea de que un nodo es importante si está conectado con otros nodos importantes. Por lo tanto, asigna puntuaciones más altas a los nodos que tienen conexiones con otros nodos de alta importancia. Por otro lado, el algoritmo de PageRank se utiliza para evaluar la relevancia de un nodo en función de la estructura de enlaces en la red. PageRank asigna puntuaciones a los nodos según la probabilidad de que un navegante aleatorio termine en ese nodo al seguir los enlaces de la red. En otras palabras, un nodo obtendrá una puntuación más alta si es enlazado por nodos importantes y relevantes en la red.

4. Técnicas y herramientas

4.1. Ánalisis de los datos disponibles

Para llevar a cabo el análisis de las redes de dependencias de paquetes de software, contamos con los valiosos datos proporcionados por libraries.io en formato CSV. Libraries.io nos ofrece una amplia gama de conjuntos de datos que contienen listas de enlaces, a partir de los cuales podemos extraer la red de dependencias de los principales repositorios de paquetes de software. Estos conjuntos de datos incluyen una diversidad de repositorios populares, entre los que se encuentran:

NPM con 3.88 millones de paquetes, Maven con 551 mil paquetes, Go con 458 mil paquetes, PyPI con 453 mil paquetes, etc...

Sin embargo, es importante destacar que estos datos no se encuentran actualizados a la fecha actual debido a la alta volatilidad de los cambios en los proyectos de software. Por lo tanto, los análisis que realicemos con estos datos no reflejarán la situación actual y actualizada de la red de dependencias.

No obstante, contamos con la ventaja de tener un conjunto de datos que abarca un gran número de repositorios y, además, incluye un histórico de las dependencias de los paquetes para sus distintas versiones. Esto nos permite realizar análisis retrospectivos y estudiar la evolución de las dependencias a lo largo del tiempo.

Para aprovechar al máximo este conjunto de datos, es importante tener en cuenta la temporalidad de la información y considerar su contexto histórico al realizar análisis y conclusiones. Esto nos permitirá comprender mejor la dinámica de las redes de dependencias y sus cambios a lo largo del tiempo.

Como conclusión final, es evidente la necesidad de obtener un nuevo conjunto de datos que nos permita realizar un estudio más preciso y actualizado de las relaciones de paquetes en los repositorios ya que por la rapidez con la que evoluciona el ecosistema del software y la constante introducción de nuevas versiones y cambios en los paquetes, es crucial contar con información que refleje de manera precisa el panorama actual de las dependencias entre los paquetes. Obtener un nuevo conjunto de datos actualizados nos brindará una visión más completa y confiable, lo que contribuirá a un análisis más sólido y a la toma de decisiones más informadas.

4.2. Exploracion de los repositorios

La exploración de los repositorios de paquetes es un paso fundamental en el estudio que nos conlleva. Los principales puntos de interés que encontramos dentro de esta temática son los siguientes:

Diversidad de repositorios:

Existen numerosos repositorios de paquetes, cada uno especializado en un lenguaje de programación o plataforma específica. Es importante conocer la variedad de repositorios relevantes para nuestra área de interés, como npm para JavaScript, PyPI para Python o Maven para Java. Cada repositorio tiene su propia comunidad, conjunto de reglas y mejores prácticas.

Estructura y metadatos de los paquetes:

Cada paquete de software en un repositorio está acompañado de metadatos que proporcionan información importante. Esto incluye el nombre del paquete, versión, descripción, autor, licencia, dependencias y más. Estos son los datos relevantes para nuestra investigación y en lo que se centrara la extraccion

Popularidad y reputación:

Algunos repositorios proporcionan métricas de popularidad, como el número de descargas o el número de estrellas en GitHub. Además, es útil tener en cuenta la reputación de los paquetes, basada en la retroalimentación de la comunidad y las revisiones de otros usuarios.

Versionado y mantenimiento:

Los repositorios de paquetes suelen manejar diferentes versiones de un paquete, cada una con sus propias mejoras, correcciones de errores y posibles cambios en las dependencias.

Documentación y ejemplos:

La documentación oficial, tutoriales y ejemplos de uso nos ayudará a entender cómo utilizar el paquete de manera efectiva, qué funcionalidades ofrece y qué limitaciones pueden existir.

Comunidad y soporte:

Suele ser común que la comunidad que rodea a un repositorio nos sirva de ayuda para conectarnos con otros desarrolladores, obtener ayuda sobre cuestiones técnicas de los paquetes y participar en debates o discusiones relevantes.

Seguridad y confiabilidad:

Los repositorios de paquetes también pueden ser susceptibles a problemas de seguridad. Es importante estar informado sobre las políticas de seguridad del repositorio, las prácticas de revisión de código, las actualizaciones de seguridad y la presencia de alertas o vulnerabilidades conocidas en los paquetes. Cabe mencionar que los puntos anteriores suelen ser comunes para casi todos los repositorios, aunque todo depende de la naturaleza del mismo, su finalidad y la comunidad de desarrolladores. En los tiempos que corren es raro no encontrar repositorios serios que no implementen estas características

4.3. Extracción de datos

Datasets de libraries.io:

Los datasets proporcionados por libraries.io constituyen una importante fuente de información, ya que contienen una amplia gama de datos sobre los repositorios de paquetes. Estos conjuntos de datos pueden ser adquiridos y utilizados para extraer información relevante, como las dependencias entre los paquetes y sus metadatos asociados. Estos datasets permiten un acceso estructurado a la información y facilitan el análisis de la red de dependencias.

En contra, la exploración de repositorios de paquetes puede presentar desafíos en términos de manejo y visualización de datos debido a su volumen masivo. El almacenamiento y procesamiento de conjuntos de datos grandes pueden requerir recursos computacionales significativos y capacidad de almacenamiento adecuada.

Para abordar este desafío, es común aplicar técnicas de manejo de datos, como la división de archivos en conjuntos más pequeños o el filtrado de

datos irrelevantes. Dividir los archivos en partes más manejables permite una mejor organización y acceso eficiente a los datos. Esto facilita la realización de análisis y visualizaciones en secciones más pequeñas del conjunto de datos completo.

API de libraries.io:

Además de los datasets, libraries.io también ofrece una API que permite acceder a los datos de forma programática. La API proporciona métodos para realizar consultas específicas y obtener información en tiempo real sobre los repositorios de paquetes. Esto brinda la posibilidad de realizar extracciones personalizadas, adaptadas a las necesidades del análisis o estudio en curso. La utilización de la API de libraries.io puede facilitar la obtención de datos, aunque no necesariamente actualizados. Existen ciertos inconvenientes a tener en cuenta al utilizar la API de libraries.io para la extracción de datos. Estos incluyen:

Autenticación:

Para realizar cualquier solicitud a la API, es necesario incluir la API KEY que se obtiene desde la página de tu cuenta. Lo que implica un registro en el servicio. Hay distintas versiones del servicio, siendo gratuita el servicio básico y las caracteristicas extra de pago

Límite de velocidad:

Todas las solicitudes están sujetas a un límite de velocidad (de 60 solicitudes por minuto en la versión gratuita) basado en tu API KEY. Si se realizan más solicitudes dentro de ese período de tiempo, se recibirá una respuesta de error 429. Este límite está diseñado para garantizar un uso equitativo de los recursos y evitar una carga excesiva en los servidores. Como consecuencia de esta limitación de velocidad, el uso de la API para la recopilación completa de datos para todos los paquetes del repositorio, si existen numerosos paquetes es una operación inviable temporalmente puesto que la cantidad de tiempo que se necesitaría para obtener estos datos sería considerablemente alto. Por ejemplo, un repositorio como NPM (aproximadamente 2 millones de paquetes) -> ((2000000 / 60) / 60) / 24 = 23.14 días

Paginación:

Algunas solicitudes pueden devolver múltiples resultados, y para manejar estos casos, se puede utilizar la paginación. El valor predeterminado de es

30, pero se puede ajustar hasta un máximo de 100 resultados por página. Esto podría suponer resultados truncados, cosa que no es deseable.

Web scraping de las listas de paquetes publicadas en los sitios web de los repositorios

El web scraping es una técnica ampliamente utilizada para extraer datos directamente de dichos sitios. En nuestro caso, permite recopilar información específica, como nombres de paquetes, descripciones y metadatos asociados, directamente de las páginas web públicas.

Esta técnica resulta útil cuando no se cuenta con conjuntos de datos completos o actualizados, ya que permite obtener información en tiempo real.

Sin embargo, el web scraping también presenta ciertos inconvenientes, como las políticas de los sitios web que pueden bloquear el acceso repetitivo desde una misma dirección IP. Al consultar la página de datos de cada paquete alojado en el servidor web, se corre el riesgo de ser denegado el servicio. Una forma de evitar este problema es utilizando técnicas como ocultar la dirección IP detrás de un proxy. No obstante, es importante destacar que el web scraping debe realizarse de manera ética y respetando las políticas de los sitios web objetivo. Otro desafío a tener en cuenta es la velocidad de obtención de datos. Para acelerar esta tarea, se han empleado técnicas de concurrencia que permiten realizar múltiples solicitudes de forma simultánea y recopilar una mayor cantidad de información en un menor período de tiempo. Esto resulta especialmente beneficioso al tratar con grandes volúmenes de datos en repositorios con numerosos paquetes.

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] Pietro Abate, Roberto Di Cosmo, Louis Gesbert, Fabrice Le Fessant, Ralf Treinen, and Stefano Zacchiroli. Mining component repositories for installability issues. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, page 24–33. IEEE Press, 2015.

[2] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 385–395, New York, NY, USA, 2017. Association for Computing Machinery.

[3] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 7 2000.

[4] Christopher Bogart, Christian Kastner, and James Herbsleb. When it breaks, it breaks: How ecosystem developers reason about the stability of dependencies. pages 86–89, 11 2015.

[5] Paolo Boldi. How network analysis can improve the reliability of modern software ecosystems. In *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pages 168–172, 2019.

[6] Daniel. dsr0018/olivia: OLIVIA - Open-source Library Indexes Vulnerability Identification and Analysis, nov 2022.

[7] Alexandre Decan, Tom Mens, and Maelick Claes. On the topology of package dependency networks: A comparison of three programming language ecosystems. In *Proccedings of the 10th European Conference*

on Software Architecture Workshops, ECSAW '16, New York, NY, USA, 2016. Association for Computing Machinery.

[8] Alexandre Decan, Tom Mens, Maelick Claes, and Philippe Grosjean. When github meets cran: An analysis of inter-repository package dependency problems. 03 2016.

[9] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 181–191, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 102–112, 2017.

[11] Tom Mens, Alexandre Decan, and Maelick Claes. An empirical comparison of dependency issues in oss packaging ecosystems. 02 2017.

[12] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[13] Marton Posfai and Albert-Laszlo Barabasi. *Network Science*. Citeseer, 2016.

[14] J. Sametinger. *Software Engineering with Reusable Components*. Springer Berlin Heidelberg, 1997.

[15] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. A look at the dynamics of the javascript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 351–361, New York, NY, USA, 2016. Association for Computing Machinery.