UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

título del TFG
Documentación Técnica

Presentado por nombre alumno
en Universidad de Burgos — 26 de junio de 2023
Tutor: nombre tutor

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

El proyecto propuesto se centra en el desarrollo de la herramienta *Olivia-Finder*, la cual ha sido diseñada con el objetivo de extraer datos de paquetes y sus dependencias de los repositorios de software *CRAN, Bioconductor, PyPI y npm.*

Dado que la herramienta en sí misma carece de elementos visuales o atractivos más allá de la presentación de los datos en forma de una lista de enlaces entre nodos de la red, se ha decidido complementarla con un análisis básico de las redes generadas desde la perspectiva de la ciencia de redes. Esto se realiza con el propósito de evitar que el trabajo resulte monótono y brindar un enfoque más completo y enriquecedor al proyecto.

En general, el proyecto se puede describir en tres etapas fundamentales. La primera etapa consiste en una exhaustiva investigación y documentación, donde se realiza un estudio en profundidad de los repositorios *CRAN, Bioconductor, PyPI y npm*, así como de las técnicas y herramientas utilizadas para la extracción de datos y análisis de redes de dependencias. Esta etapa sienta las bases teóricas necesarias para comprender el contexto en el que se desarrollará la herramienta y el análisis posterior.

La segunda etapa se enfoca en el diseño y desarrollo de la herramienta *Olivia-Finder*. Aquí, se aplican los conocimientos adquiridos durante la etapa de investigación para implementar una solución eficiente y robusta que permita la extracción de datos de los repositorios mencionados. Se deben tener en cuenta diversos aspectos técnicos, como el manejo de solicitudes web, el procesamiento de datos y la manipulación de estructuras de red.

La tercera etapa del proyecto implica el análisis experimental de las redes generadas. Una vez obtenidos los datos de los paquetes y sus dependencias, se aplican técnicas de la ciencia de redes para examinar características importantes, como la *centralidad de grado*, el algoritmo *PageRank* y otras métricas relevantes. Este análisis proporciona una comprensión más profunda de la estructura y las propiedades de las redes de dependencias en los repositorios estudiados, permitiendo identificar paquetes críticos, vulnerabilidades potenciales y relaciones significativas entre los elementos de la red.

A.2. Planificación temporal

En el ámbito de los proyectos modernos de software, es común utilizar metodologías ágiles, como *Scrum* o *Kanban*. Estas metodologías reconocen la naturaleza dinámica del proceso iterativo que implica el establecimiento de requisitos, diseño, desarrollo y validación de un producto de software. Por lo tanto, se enfocan en la planificación adaptativa y la mejora continua a través de entregas tempranas.

Sin embargo, en nuestro caso, debemos cuestionar la aplicabilidad práctica de las metodologías ágiles tal como están concebidas. Esto se debe a que el conjunto de partes interesadas, que se limita a un cliente académico prototípico, y sobre todo al hecho de que el equipo de trabajo es unipersonal. Estas circunstancias particulares plantean dudas sobre la eficacia de la implementación de las metodologías ágiles en nuestro contexto.

No obstante, podemos establecer similitudes entre nuestro proceso y el marco de trabajo *Scrum*, debido a la presencia de *sprints*. Aunque no hemos seguido rigurosamente la estructura de los *sprints* debido a la falta de objetivos claramente definidos en el proyecto, los *sprints* nos han permitido representar la actividad realizada y las etapas en las que hemos dividido el trabajo.

Por otro lado, se ha de tener en cuenta que las tareas de investigación incluidas en el proyecto son difíciles de planificar. El proceso de investigación implica continuos replanteamientos y el alcance de los resultados debe ser constantemente modelado o redefinido a medida que avanzamos dentro del límite temporal del que se dispone.

Sprint 0: Kick of meeting

Este *sprint* representó nuestra primera aproximación a la temática del proyecto. Desde el principio, quedó claro que el modelo matemático propor-

cionado por Olivia en el Trabajo de Fin de Grado anterior estaba más allá de nuestra comprensión absoluta, y nuestro enfoque se centraría principalmente en la experimentación.

Por lo tanto, fue necesario realizar un esfuerzo inicial para comprender los fundamentos básicos de lo que el modelo de *OLIVIA* nos permitía hacer. En este sentido, los cuadernos de trabajo proporcionados en el TFG anterior fueron de gran utilidad, ya que mostraban la funcionalidad en casos de estudio concretos y ofrecían un análisis complementario.

Desde el principio, nos encontramos con problemas técnicos. En primer lugar, las dependencias de *OLIVIA* requieren una actualización, ya que se están utilizando versiones algo desactualizadas de algunos paquetes y herramientas de análisis, y el *Dependabot de GitHub* insiste en su actualización *"Bump numpy from 1.18.5 to 1.22.0"*. En concreto, se ha identificado que esta versión de *NumPy* presenta vulnerabilidades de alto riesgo, como la referida al *NumPy NULL Pointer Dereference*.

Sin embargo, no es posible actualizar la biblioteca, ya que existen métodos esenciales para la funcionalidad implementada en *OLIVIA* que se han vuelto obsoletos en la versión actualizada. Además, en cuanto a la versión de Python, el código debe ejecutarse en la versión 3.8 debido a la presencia de dependencias clave para OLIVIA, como *intbitset*, que no están disponibles para otras versiones de Python. Esto supuso un problema al ejecutar los cuadernos de trabajo en *Google Colab* debido a las dificultades para instalar la versión específica de Python que requeríamos.

Además, fue necesario realizar una introducción a la temática mediante una lectura superficial de la memoria del TFG de OLIVIA. Esto también nos permitió familiarizarnos con el hecho de que estas memorias se redactan utilizando LaTeX, una tecnología que nos resultaba completamente desconocida en ese momento.

En conclusión, este *sprint* inicial implicó la familiarización con el modelo de OLIVIA, la comprensión de sus fundamentos y la resolución de desafíos técnicos relacionados con las dependencias y las versiones de Python. Además, se llevó a cabo una introducción a la temática a través del estudio de la memoria del TFG anterior, lo que también nos permitió adquirir conocimientos sobre el uso de LaTeXen la redacción de documentos científicos.

La duración de este *sprint* ha sido de 30 días aproximadamente, realizando 2 reuniones y con unas 25 horas de trabajo.

Sprint 1 - Data collection

En esta etapa de investigación nos enfocamos en el manejo de datos. Inicialmente, utilizamos el conjunto de datos de *libraries.io*, el cual resultó útil desde una perspectiva histórica. Sin embargo, presentó importantes limitaciones, como la falta de actualización periódica. Al enfrentarnos a los desafíos derivados de este conjunto de datos, pudimos constatar que trabajar con grandes volúmenes de datos no es trivial. Fue necesario utilizar un disco duro externo para almacenar el conjunto de datos, del cual solo nos interesaba la lista de enlaces entre paquetes y dependencias contenida en uno de los archivos CSV. Debido a la gran cantidad de líneas y su considerable tamaño, resultó difícil manipular y filtrar los datos.

La falta de actualización de los datos fue un aspecto clave a mejorar, por lo cual exploramos otras fuentes de información, como la API de *libraries.io* y el conjunto de datos de *BigQuery* proporcionado por los mismos desarrolladores. El uso de la API quedó descartado debido a sus limitaciones técnicas para realizar un escaneo completo del repositorio. Por otro lado, el conjunto de datos de *BigQuery* también presentó problemas similares a los archivos CSV en términos de falta de actualización.

Como alternativa, se propuso utilizar técnicas de *web scraping* para recolectar información de los sitios web de los repositorios de software de nuestro interés, comenzando con la recolección de información de CRAN. Este esfuerzo dio resultados positivos, y logramos obtener una lista actualizada de la red de CRAN. Los éxitos obtenidos en este proceso nos llevaron a considerar el desarrollo de una herramienta que pudiera obtener esta información para los repositorios de interés, ya que esto generaría un nuevo conjunto de datos utilizable en Olivia y en trabajos anteriores que hayan utilizado datos de *libraries.io*, lo que permitiría actualizar sus resultados de manera significativa.

Como resultado al finalizar este *sprint*, logramos obtener una red actualizada de CRAN gracias a que desarrollamos un prototipo básico pero todavía inmaduro de un código en Python que nos permitió realizar la recolección de datos.

Es importante destacar que durante este *sprint* se abordó uno de los principales desafíos que hemos enfrentado. Como era de esperar, los servidores web implementan medidas de seguridad para evitar comportamientos maliciosos. Debido a la naturaleza del proceso de escaneo de un repositorio, que implica realizar numerosas solicitudes web a un mismo servidor desde una misma dirección IP en un período de tiempo relativamente corto, esta

actividad a menudo resulta en la prohibición temporal de acceso al servidor web para los equipos asociados a esa dirección IP. Para solucionar este problema, se implementó la funcionalidad de ocultar las solicitudes detrás de servidores proxy, que enmascaran la dirección IP de origen al servidor web. Por lo tanto, fue necesario desarrollar esta funcionalidad para que fuera posible llevar a cabo esta tarea.

En conclusión, este *sprint* fue clave para obtener un conjunto de datos actualizado y establecer los primeros pasos en el desarrollo de una herramienta de recolección de datos. Además, se logró resolver el desafío de la prohibición de acceso a los serv

idores web mediante la implementación de rotación de proxy. Estos avances sientan las bases para continuar con el desarrollo del proyecto y alcanzar los objetivos planteados.

La duración de este *sprint* ha sido aproximadamente de 30 días, que en nuestra metodología de trabajo equivalen aproximadamente a 2 reuniones y a 60 horas de trabajo.

Sprint 2 - Library implementation and interpretation of some data

En esta etapa, nuestro enfoque se centró en el desarrollo de una herramienta genérica con el propósito de llevar a cabo una extracción sencilla de la red de dependencias de los repositorios *CRAN*, *Bioconductor*, *PyPI* y *npm*. Nos enfrentamos al primer desafío de obtener una lista completa de los paquetes disponibles en cada repositorio, que serviría como punto de partida para nuestra recopilación de datos. Cabe destacar que esta tarea no esulta sencilla en la mayoría de los casos, ya que dichas listas no siempre están disponibles.

Es importante mencionar que cada repositorio de software presenta sus peculiaridades distintivas. En el caso particular de *CRAN* y *Bioconductor*, el proceso se basó exclusivamente en técnicas de *web scraping*, aprovechando los datos de interés que se encuentran directamente en las listas de paquetes publicadas en sus respectivos sitios web.

El análisis de *CRAN* resultó ser el más sencillo de todos, ya que su página web es simple, robusta y aparentemente sólida a lo largo del tiempo, lo que proporciona una implementación bastante estable para este repositorio. Por otro lado, inicialmente se esperaba que *Bioconductor* fuese más sencillo debido a la menor cantidad de paquetes en comparación con los otros

repositorios en los que estamos trabajando. Sin embargo, nos encontramos con un problema en el servidor web de *Bioconductor*, específicamente en la página que muestra el listado de paquetes disponibles, ya que no era accesible mediante una simple solicitud web, como las que solemos realizar utilizando la biblioteca *requests"* de Python.

En *Bioconductor*, se utilizaba JavaScript para cargar dinámicamente la lista de paquetes en tiempo de ejecución sobre la página. Esta situación dificulta la obtención de los datos, ya que la biblioteca *requests* no es compatible con la carga de JavaScript. Como alternativa, tuvimos que recurrir a la biblioteca *selenium*, la cual ofrece funcionalidades más avanzadas al actuar como un navegador *headless* (sin interfaz gráfica) que se comporta de manera similar a un navegador de escritorio convencional al que estamos acostumbrados, pero que admite la automatización de tareas. Gracias a *selenium*, logramos extraer la lista de paquetes de *Bioconductor* y, a partir de ella, procedimos de manera similar a como lo habíamos hecho anteriormente, obteniendo los datos de interés mediante técnicas de *web scraping*.

El repositorio *PyPI*, también publica una lista de paquetes. Sin embargo, es importante destacar que esta lista contiene muchos paquetes obsoletos e inexistentes, lo que la convierte en un punto de partida necesario pero no del todo óptimo. Utilizando esta lista de paquetes y aprovechando la API proporcionada por *PyPI* para obtener metadatos de los paquetes, pudimos extraer la red de dependencias correspondiente.

En esta etapa, nos percatamos del tiempo considerablemente elevado requerido para llevar a cabo esta recopilación, así como de la necesidad de cuidar la implementación de la herramienta para evitar el desperdicio de memoria. Estos problemas surgidos debido al tamaño de los datos nos proporcionan una perspectiva clara de la importancia de gestionar eficientemente los recursos cuando se trabaja con cantidades masivas de información. En cuanto al tiempo necesario para la recolección, se realizó un esfuerzo por optimizar el proceso mediante técnicas de concurrencia, lo cual nos permitió realizar solicitudes web de forma concurrente en lugar de secuencial, como habíamos estado haciendo hasta ese momento. Estas mejoras significativas en el rendimiento de la herramienta se tradujeron en una reducción significativa del tiempo requerido y el consumo de memoria.

Finalmente, logramos generar el conjunto de datos para el repositorio *npm*. Es importante destacar que este repositorio ha sido el más desafiante de abordar. En primer lugar, no existe una forma sencilla de obtener una lista completa de los paquetes existentes en *npm*. Además, el repositorio oficial de paquetes no es único, ya que existen repositorios espejo (*mirrors*)

alternativos que difieren tanto en el número de paquetes como en los paquetes que contienen. Para abordar este problema, decidimos utilizar la lista de paquetes que pudimos extraer de uno de estos *mirrors* y complementar con los paquetes que teníamos en el conjunto de datos de *libraries.io*. De esta manera, obtuvimos una lista de nombres de paquetes a los cuales dirigir nuestros esfuerzos. A partir de esta lista, aplicamos la metodología utilizada previamente para el resto de los repositorios. En el caso de *npm*, gracias a su API, pudimos recopilar los metadatos necesarios para construir el conjunto de datos correspondiente.

Llevar a cabo este *sprint* ha supuesto una duración de 60 días, lo que aproximadamente corresponde con unas 3 reuniones y alrededor de 120 horas de trabajo.

Sprint 3 - Library refactoring

Tras el análisis de los datos recolectados, observamos que los conjuntos de datos de *libraries.io* proporcionan información sobre todas las versiones existentes de un paquete. En otras palabras, consideramos como *dependencias* de un paquete todas las dependencias que hayan existido para cada una de sus versiones. Sin embargo, desde una perspectiva de desarrollo de software, esto no es correcto, ya que sobrecargamos las dependencias de un paquete con dependencias de versiones antiguas que ya no se utilizan en el ciclo de vida actual de esas bibliotecas. Por lo tanto, es importante tener en cuenta esta información en los análisis posteriores que realicemos, donde será necesario aplicar un filtrado adecuado si deseamos utilizar los datos de *libraries.io*. Otro aspecto interesante es que no todas las *dependencias* de un paquete se encuentran presentes en el repositorio al que pertenece ese paquete, e incluso es posible que no utilicen el mismo lenguaje de programación. Tomemos como ejemplo un paquete en Python que depende de un binario escrito en C. Este fenómeno es muy común en la red de *Bioconductor*, cuyos paquetes dependen en gran medida de paquetes existentes en *CRAN*. A estas dependencias las hemos bautizado como *dependencias foráneas*.

Con el objetivo de mejorar la funcionalidad y flexibilidad de la biblioteca, se llevó a cabo una refactorización del diseño para adaptarlo y permitir el uso y la combinación de diversas fuentes de datos. Se proporciona soporte para *web scraping*, conjuntos de datos en formato *CSV*, la API de *libraries.io* y repositorios de *GitHub*. El uso combinado de diferentes fuentes de datos nos brinda la capacidad de buscar en fuentes alternativas cuando un paquete solicitado no se encuentra en la fuente principal. El soporte de archivos *CSV* nos permite considerar los conjuntos de datos de *libraries.io* como fuente

de información, tanto a través de su API como de los archivos en sí. La implementación para repositorios de *GitHub* nos proporciona información adicional al utilizar *GitHub* como fuente de datos, ya que es el sistema de control de versiones por excelencia donde se encuentran la mayoría de los proyectos de software de código abierto publicados. De esta manera, tenemos acceso a un repositorio de un nivel inferior, ya que no pertenece a un gestor de paquetes específico de un lenguaje de programación, sino que se basa su relevancia principalmente en un ámbito más cercano al desarrollo. Además, la biblioteca ofrece otras funcionalidades, como la obtención en tiempo de ejecución de una red de dependencias transitiva para un paquete en particular. También proporciona persistencia de datos en forma de objetos serializados y la capacidad de exportar datos en formato *CSV* compatible con *OLIVIA*.

Una vez concluida esta etapa, procedimos a publicar los conjuntos de datos en *Zenodo*, con el fin de hacerlos accesibles para la comunidad científica. Además, dedicamos nuestros esfuerzos a mejorar la documentación del código fuente y generar una documentación completa de la biblioteca. Esta documentación está diseñada para ser accesible desde la web y se encuentra alojada en *GitHub Pages*. La publicación de los conjuntos de datos en *Zenodo* permite a otros investigadores y desarrolladores acceder a los datos recopilados y utilizarlos en sus propios proyectos o investigaciones. De esta manera, promovemos la transparencia y el intercambio de información entre la comunidad científica. En cuanto a la documentación de la biblioteca, nos esforzamos por ofrecer una guía clara y concisa sobre cómo utilizar la biblioteca, qué funcionalidades y características ofrece, así como ejemplos de uso. La documentación se ha estructurado de manera que sea fácil de navegar y buscar información relevante. Al alojarla en *GitHub Pages*, proporcionamos un acceso práctico y amigable para los usuarios, quienes pueden acceder a la documentación directamente desde el sitio web del proyecto en *GitHub*. Este *sprint* ha sido difícil de calcular el tiempo que ha llevado realizarlo, debido a que ha habido saltos hacia *sprints* anteriores cuando estamos trabajando en este. Ha sido uno de los más complejos. Se calcula un periodo de 60 días, 4 reuniones y aproximadamente 150 horas de trabajo.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una muestra de cómo podría ser una tabla de casos de uso:

B.2. Objetivos generales

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

CU-1	**Ejemplo de caso de uso**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla B.1: CU-1 Nombre del caso de uso.

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

Este anexo incluirá una reflexión personal del alumnado sobre los aspectos de la sostenibilidad que se abordan en el trabajo. Se pueden incluir tantas subsecciones como sean necesarias con la intención de explicar las competencias de sostenibilidad adquiridas durante el alumnado y aplicadas al Trabajo de Fin de Grado.

Más información en el documento de la CRUE https://www.crue.org/wp-content/uploads/2020/02/Directrices_Sosteniblidad_Crue2012.pdf.

Este anexo tendrá una extensión comprendida entre 600 y 800 palabras.

Bibliografía