

TRABAJO DE FIN DE MÁSTER

**Cálculo de métricas del paisaje a partir
del SIOSE: Una propuesta escalable
basada en Postgres/PostGIS**

Andrea Rosado Abad

Directores: Raquel Montorio Llovería y Daniel Borini Alves

*Máster Universitario en Tecnologías de la Información Geográfica para
la Ordenación del Territorio: Sistemas de Información Geográfica y
Teledetección*

Universidad de Zaragoza

Noviembre 2017

Agradecimientos

Este Trabajo de Fin de Máster ha sido posible gracias al apoyo y ayuda de muchas personas a las que me gustaría agradecer y también por todo el conocimiento que he obtenido gracias a ellos a lo largo de esta etapa.

En primer lugar, he de dar las gracias a mis directores Raquel Montorio Llovería y Daniel Borini Alves de la Universidad de Zaragoza quienes han dirigido este trabajo, y Alfredo Ramón Morte por darme la oportunidad de volver a realizar por segundo año consecutivo las prácticas de empresa en el Laboratorio de Geomática de la Universidad de Alicante.

También me gustaría agradecer especialmente a los compañeros del laboratorio por su colaboración y paciencia a lo largo de las prácticas como también por su apoyo y ayuda cuando lo necesitaba: B.M. Zaragozaí Zaragozaí. J. Torres Prieto y J.T. Navarro Carrión. Gracias por hacerme sentir como si fuera una compañera más.

Finalmente, y no menos importante, a mi familia por su apoyo y comprensión, como también a todos mis amigos y compañeros del máster de la Universidad de Zaragoza.

Prólogo

Presentación

El presente Trabajo Fin de Máster en Tecnologías de la Información Geográfica para la Ordenación del Territorio: SIG y Teledetección, expone todas las tareas que se han realizado durante la colaboración en el Proyecto SIOSE-INNOVA: Innovaciones técnicas y metodológicas en el Sistema de Información sobre Ocupación del Suelo de España (SIOSE) y su aplicación en estudios geográficos. El investigador principal de este proyecto es Alfredo Ramón Morte, que cuenta con la participación de los miembros del Laboratorio de Geomática y otros profesionales, en colaboración con el equipo de investigación responsable de la base de datos del SIOSE.

Todo este trabajo se ha realizado mediante el marco de convenio de prácticas de empresa entre la Universidad de Zaragoza y el Laboratorio de Geomática del Instituto Interuniversitario de Geografía de la Universidad de Alicante. El periodo de prácticas ha tenido una duración desde julio hasta noviembre de 2017, de un total de 440 horas (ver en el Anexo).

El Laboratorio de Geomática se encarga de administrar el Sistema de Información Geográfica de la Universidad de Alicante (SIGUA), o como era conocido al principio, Laboratorio de SIG y Cartografía Automatizada. Este cambio de nombre fue razón por la orientación del laboratorio al desarrollo de soluciones basadas en la geomática o informática aplicada a la Geografía.

Su origen data de 1997, año en el que además, se pone en marcha el servicio SIGUA, que gran parte de los empleados se centran en el mantenimiento del sistema, como también de la creación de nuevas utilidades adaptadas a las necesidades de la Universidad de Alicante y, compartir recursos e interconectar sistemas de información por otras unidades. Cabe mencionar que, está formado por licenciados en Geografía y en Informática que desarrollan su trabajo en las Tecnologías de la Información Geográfica basado en software libre.

Proyecto SIOSE-INNOVA

SIOSE-INNOVA es un proyecto de investigación financiado por el Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, dentro del marco Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016. Los objetivos principales tienen una parte innovadora, que consiste en comprobar qué tecnologías NoSQL (no sólo SQL) pueden aportar mejores soluciones para la explotación de la base de datos del SIOSE, y una parte aplicada, que consiste en poner en práctica las nuevas tecnologías en casos de estudios reales.

Durante el desarrollo del proyecto, se quieren alcanzar los siguientes objetivos específicos:

1. Crear un marco de experimentación reproducible y fácilmente utilizable por un gran número de usuarios.
2. Analizar las necesidades y rendimiento de distintas tecnologías de bases de datos NoSQL para la explotación del SIOSE.
3. Desarrollar e implementar un nuevo modelo de datos auxiliar que permita extender las posibilidades de análisis del SIOSE con técnicas de Big Data o Data Mining.
4. Evaluar la usabilidad de los datos SIOSE en distintas plataformas tecnológicas, mediante su aplicación en casos de uso reales en los que utilizar datos de ocupación del suelo resulte esencial.

A partir de estos objetivos, el proyecto SIOSE-INNOVA tiene como objetivo final crear un visor cartográfico donde se pueda consultar y comparar resultados entre distintos paisajes (ver figura 1). Por este motivo, se desarrolla una nueva extensión denominada *pg_landmetrics* capaz de calcular métricas del paisaje, papel fundamental para este trabajo.

Las métricas de paisaje son importantes para el estudio del paisaje en su estructura, comportamiento o modificación temporal, ya sea por factores naturales como artificiales. Es por este motivo la necesidad de crear una extensión que permita aplicarlas y calcularlas sobre el paisaje.



Figura 1 Prototipo del visor cartográfico.

Estructura del trabajo

Este trabajo se organiza en cuatro capítulos, a parte las referencias bibliográficas y los anexos, y su estructura es la siguiente:

- En el capítulo 1 se analiza la estructura del paisaje a partir del uso y cobertura del suelo en relación con las métricas de paisaje. Además, se desarrollan los objetivos que se quieren alcanzar en el trabajo.
- En el capítulo 2 se describen todos los procesos necesarios para implementar y desarrollar la nueva extensión PostgreSQL/PostGIS, desde el conjunto de datos hasta las tareas diarias, la incorporación de funciones y la documentación de la extensión.
- En el capítulo 3 se detallan todos los resultados que se obtienen a partir de la extensión en casos reales.
- Por último, en el capítulo 4 se realiza un análisis conclusivo para Además, se describen las aportaciones interesantes para trabajos futuros.

Resumen

Las métricas de paisaje se utilizan para analizar la estructura y comportamiento del paisaje como también las modificaciones temporales, ya sea por factores naturales o humanos. Dada la utilidad para una variedad de aplicaciones, existen muchos softwares diseñados para ofrecer cálculos y análisis de patrones de paisaje. El objetivo principal de este trabajo es crear una extensión PostgreSQL/PostGIS reproducible y extensible capaz de calcular métricas de paisaje para datos de entrada vectoriales. Más adelante, esta extensión debería permitir añadir nuevas métricas e investigar nuevos estudios relacionados con la estructura del paisaje. Finalmente, se pone en valor la implementación de la extensión ya que requiere de una metodología de trabajo colaborativo basado en una serie de herramientas de contenerización y orquestación.

Palabras clave: métricas de paisaje, extensión, reproducibilidad, contenerización, orquestación.

Abstract

Landscape metrics are used to analyze the structure and behavior of the landscape as well as the temporary modifications, either by natural or human factors. Given the utility for a variety of applications, there are many softwares designed to offer calculations and analysis of landscape patterns. The main objective of this work is to create a reproducible and extensible PostgreSQL/PostGIS extension capable of calculating landscape metrics for vector input data. Later, this extension should make it possible to add new metrics and investigate new studies related to landscape structure. Finally, the implementation of the extension is valued because it requires a collaborative work methodology based on a series of tools for containerization and orchestration.

Key words: landscape metrics, extension, reproducibility, containerization, orchestration.

Índice general

Índice de figuras	XV
Índice de tablas	XVII
Ejemplos de código	XVIII
1. Introducción	1
1.1. Análisis de la estructura del paisaje a través del SIOSE	1
1.2. Métricas de paisaje y software que las calcula	2
1.3. Objetivos	2
2. Metodología	3
2.1. Software y plataformas	5
2.1.1. Control de versiones	5
2.1.2. Contenerización y orquestación de servicios	8
2.1.3. Extensibilidad	11
2.1.4. Aplicaciones	13
2.2. Conjunto de datos	13
2.3. Selección de métricas	16
2.4. Implementación/desarrollo de funciones en PostgreSQL	16

2.5. Documentación de la extensión	16
3. Resultados y Discusión	27
3.1. Resultados	27
3.2. Discusión	27
4. Conclusiones y trabajo futuro	29
Bibliografía	31
Apéndice A. Anexo	33

Índice de figuras

1.	Prototipo del visor cartográfico.	VII
2.1.	Flujo de proceso de actualización de ficheros.	7
2.2.	Flujo de proceso de trabajo colaborativo entre repositorios.	9
2.3.	Contenerización del sistema operativo.	10
2.4.	Flujo de proceso de integración continua.	12

Índice de tablas

2.1. Atributos del primer conjunto de datos.	13
2.2. My caption	14
2.3. Atributos del primer conjunto de datos.	15
2.4. My caption	15

Ejemplos de código

2.1. Crear una función para calcular el IDW (I)	17
2.2. Crear una función para calcular el IDW (I)	17
2.3. Crear una función para calcular el IDW (I)	17
2.4. Crear una función para calcular el IDW (I)	17
2.5. Crear una función para calcular el IDW (I)	17
2.6. Crear una función para calcular el IDW (I)	18
2.7. Crear una función para calcular el IDW (I)	18
2.8. Crear una función para calcular el IDW (I)	19
2.9. Crear una función para calcular el IDW (I)	22
2.10. Crear una función para calcular el IDW (I)	23
2.11. Crear una función para calcular el IDW (I)	23
2.12. Crear una función para calcular el IDW (I)	23
3.1. Crear una función para calcular el IDW (I)	27
A.1. Crear una función para calcular el IDW (I)	33

1. INTRODUCCIÓN

Puntos de interés:

- El SIOSE es una valiosa **base de datos de ocupación del suelo** que contiene un gran volumen de información territorial de toda España.
- Desde su aparición en 2005, SIOSE se ha convertido en un repositorio de referencia para sus homólogos europeos, llegando a ser un **modelo para la iniciativa EAGLE** (*SIOSE europeo*).
- A pesar de su gran potencial, el SIOSE presenta ciertos problemas de *usabilidad* debidos a su gran volumen y complejidad (p.ej. desde aplicaciones SIG de escritorio).
- Las **métricas de paisaje** son métodos cuantitativos que sirven para analizar la estructura del paisaje y otros fenómenos (p.ej. evolución del paisaje, conectividad de ecosistemas, entre otros).
- FRAGSTATS, Conefor Sensinode, Patch Analyst, entre otros, son aplicaciones de escritorio muy utilizadas para el cálculo de métricas del paisaje. No obstante, **no hay ninguna aplicación** que sea fácilmente **escalable y extensible** como para realizar análisis sobre una geodatabase similar a la del SIOSE.
- El principal objetivo de este trabajo es **desarrollar una extensión** (PostgreSQL/PostGIS) capaz de calcular métricas de paisaje a partir de geodatabases tan voluminosas o más como la del SIOSE *actual* (2014).

1.1. Análisis de la estructura del paisaje a través del SIOSE

El SIOSE características

El SIOSE importancia/alcance

Aplicaciones de los usos/coberturas del suelo...

Usabilidad del SIOSE. En jornada de girona (2011)

SIOSE-INNOVA

1.2. Métricas de paisaje y software que las calcula

Métricas del paisaje como técnica/metodología para el estudio del paisaje... aplicaciones agua, biodiversidad, riesgos naturales (), estructura urbana, etc

Software. En [1] se establece una comparativa entre programas específicos para el cálculo de métricas del paisaje, destacando entre ellos FRAGSTATS. Otros programas como... Evidentemente, esta lista no puede estar completa. Existen muchos otros programas especializados que permiten el cálculo de este tipo de métricas/índices

También es posible calcular determinadas métricas con herramientas típicas de un SIG...

1.3. Objetivos

2. METODOLOGÍA

La implementación de `pg_landmetrics` requiere de una metodología de trabajo colaborativo basado en una serie de herramientas de contenerización y orquestación de manera eficiente, extensible y reproducible.

Puntos de interés:

- Este trabajo aplica una metodología de **integración continua y desarrollo colaborativo** de acuerdo con los objetivos del proyecto SIOSE-INNOVA.
- El trabajo colaborativo se ha coordinado utilizando *Git* que es el sistema de **control de versiones** más popular de los últimos años (p.ej. utilizado en , PostGIS, QGIS, CARTO y decenas de proyectos ESRI, entre muchos otros).
- La **contenerización o *dockers*** es una novedosa tecnología para la virtualización de software/servicios, frente a la virtualización de sistemas operativos (p.ej. máquinas virtuales). La orquestación de *dockers* permite organizar complejos sistemas de información con muchas facilidades.
- PostgreSQL/PostGIS es la *geodatabase libre* más potente del mercado, destacando por sus opciones de **extensibilidad** (p.ej. PostGIS en sí misma es una extensión de PostgreSQL).
- Los desarrollos en PostgreSQL se pueden realizar en lenguajes de programación como ANSI C, SQL y/o distintos lenguajes procedurales (p.ej. PLpgSQL, PL/R, PL/Python, entre muchos otros), **dependiendo de las necesidades**.
- En este trabajo se han utilizado dos conjuntos de datos, **un paisaje de ejemplo y el SIOSE de 2011 completo**, para poner a prueba la extensión *pg_landmetrics* propuesta en los objetivos.
- El número potencial de métricas del paisaje es indeterminado y depende de muchos factores (p.ej. objetivos del estudio, modelos de datos como *raster/vector* o en red, niveles de agregación y/o escala, etc). Resulta esencial determinar unas **métricas representativas** para esta primera propuesta.
- Una parte fundamental de esta metodología es la **documentación** del desarrollo y uso de la extensión. Una buena documentación con ejemplos facilitará el cálculo de métricas del paisaje en grandes repositorios, **sobretudo para aquellos usuarios con menos experiencia** en PostgreSQL/PostGIS.

2.1. Software y plataformas

A lo largo de este proyecto se han utilizado una serie de softwares y plataformas para el desarrollo de la nueva herramienta, clasificados en diferentes grupos. Dentro de estos conjuntos se encuentran explicados con más detalle los procesos que corresponden según las aplicaciones que se utilizaron. Estas tareas se llevaban a cabo de forma diaria y se solían efectuar cada cierto tiempo. Estos procesos se dividen en tres: (1) actualización de ficheros, (2) trabajo colaborativo e (3) integración continua.

Antes que nada, se debe de tener en cuenta que los procesos están representados mediante diagramas de secuencia. Por ello, la lectura debe realizar de izquierda a derecha y de arriba a abajo. Las líneas discontinuas corresponden a las notificaciones o mensajes intercambiables que son enviados y recibidos entre distintos actores, y las líneas continuas corresponden a las operaciones que realiza y recibe cualquier actor. Además, con ello podemos relacionar la activación y desactivación del actor cuando éste envía o realiza cualquier operación. Finalmente, entre actores aparecen expresiones como condicionantes o alternativas (alt), opcionales (opt) y bucles o ciclos (loop), es decir, una operación que se repite tantas veces sea necesario o quiera el usuario. Y no menos importante, los actores son los softwares o plataformas utilizados durante el proceso.

2.1.1. Control de versiones

Todas las modificaciones que surgían de uno o varios archivos a lo largo de una tarea, se utilizaba el control de versiones que se encargaba de registrarlas. Gracias a este sistema, se tenía la posibilidad de recuperar una versión antigua en cualquier momento. Para ello se han utilizado dos tipos de sistemas de control:

- **Git:** permite tener un control de las versiones que se han distribuido con código abierto y libre, con mayor rapidez y eficacia, sobre el repositorio (lugar en el que se almacenan todos los archivos necesarios de una investigación) en el que estamos trabajando. Por otro lado, este sistema proporciona la posibilidad de replicar y actualizar el repositorio de trabajo, además de añadir, eliminar, mover o reemplazar cualquier archivo. La ventaja de este tipo de sistemas es que se puede crear una rama (branch) colaborativa del repositorio original para tener un aseguramiento de los datos, ya que si en cualquier momento se registra algún problema o éstos se dañan, existe la posibilidad de volver a restaurarlos sin ningún problema desde el repositorio original. Así pues, se creó

una nueva rama colaborativa paralela¹ a la rama original del proyecto para operar y modificar sin el riesgo de que los archivos originales fueran dañados. Una vez finalizados y revisados todos los cambios, se procedía a unirlos al repositorio original.

- **GitHub:** es una plataforma de desarrollo que alberga proyectos de software, como es el caso del proyecto SIOSE-INNOVA², y almacena de forma pública todo el código, trabajando de forma colaborativa utilizando el sistema de control de versiones de Git. En este caso, su presentación es más visual e interactiva, a través de una interfaz web.

Una de las tareas diarias que corresponde a esta subsección es la actualización de ficheros de la extensión. El proceso consistía en que desde la máquina local Git(local) el usuario ejecutaba un comando pull que se encargaba de obtener la última versión del propio repositorio y de revisar si han habido cambios desde la última actualización (este paso es opcional en el caso de que no se tengan los ficheros actualizados pero si se tiene la última versión, no hace falta ejecutar el anterior comando). Cuando la máquina local tenía la última versión, se iniciaba un loop, es decir, el usuario editaba los ficheros de su sistema las veces que eran necesarias. Hechas estas modificaciones, se añadían y actualizaban los ficheros mediante el comando commit, obteniendo a continuación un hash (identificador único) por cada cambio que se había realizado durante el proceso. A partir de aquí, el loop finalizaba si el usuario decidía que no habían más modificaciones que realizar. Finalmente, desde la máquina local, el usuario subía todos los cambios realizados a su propio repositorio y así tener una nueva versión actualizada (ver figura 2.1).

Otro de los procesos, que comparte esta y la siguiente subsección, corresponde al trabajo colaborativo de los usuarios entre sus propios repositorios y el repositorio Upstream (original) del proyecto. Desde el punto de vista más técnico, una de las primeras tareas era realizar un fork, es decir, una copia de todos los ficheros del repositorio Upstream a un nuevo repositorio del propio usuario. La ventaja de esto es que si en cualquier momento el repositorio del usuario sufría algún contratiempo, se podía volver a realizar un fork del repositorio Upstream. A continuación, si no había una copia en la máquina local Git (local), se ejecutaba el comando clone para obtener una copia de la última versión de todos los ficheros del repositorio del usuario. En el caso de que los ficheros no estaban actualizados, se ejecutaba una nueva instrucción donde Git(local) revisaba las últimas modificaciones y, si no los tuviera actualizados, se obtenía su última versión (estos últimos procesos son alternativos, ya que depende del estado de los ficheros). A partir de aquí, el sistema de ficheros quedaba actualizado y se iniciaba el proceso de trabajo colaborativo. En este punto, se producía un

¹https://github.com/andrearosado/pg_landmetrics

²https://github.com/siose-innova/pg_landmetrics

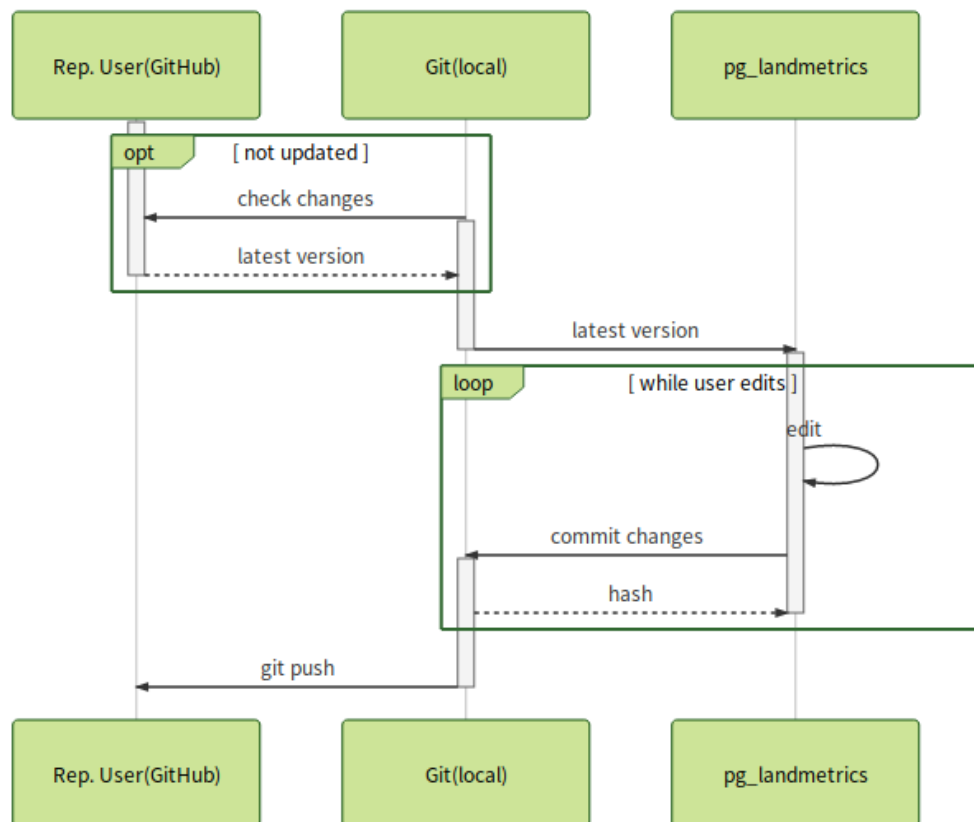


Figura 2.1 Flujo de proceso de actualización de ficheros.

loop cada vez que el usuario editaba ficheros, añadía y actualizaba en la máquina local y finalmente, recibía un hash correspondiente a cada modificación que se había realizado. Una vez se hechos todos los cambios, éstos se subían al propio repositorio y en este momento el propio repositorio del usuario estaba por delante del repositorio Upstream ya que éste no tenía los últimos cambios guardado y el loop terminaba ya que no habían más modificaciones que realizar. Ahora es cuando interviene el trabajo colaborativo, es decir, desde el propio repositorio del usuario se enviaba una petición al repositorio Upstream para emparejar ambos a través de la función pull request desde la plataforma de GitHub. Si la petición era aceptada por el personal encargado del repositorio Upstream, el usuario ejecutaba el comando merge (unión) y recibía una notificación de sincronización completada. Finalmente los repositorios quedaban sincronizados y los ficheros con la última versión. Se volvían a subir los cambios al propio repositorio del usuario y se recibía una notificación de que los cambios habían sido correctos. Todo esto queda dentro de un loop hasta que el usuario decidía que no habían más cambios que realizar o ficheros que actualizar (ver figura 2.2).

2.1.2. Contenerización y orquestación de servicios

Uno de los objetivos de la extensión era obtener un alto grado de reproducibilidad y, por ello se ha llevado a cabo la virtualización de un sistema operativo a través de contenedores³ Docker (ver figura 2.3).

Se han utilizado los siguientes elementos:

- **Docker:** es un servicio de integración continua de código abierto que automatiza el despliegue de ficheros de configuración de aplicación en contenedores facilitando la virtualización en el sistema operativo en cualquier máquina con Docker instalado. Este servicio permitió la portabilidad de las aplicaciones y la reproducibilidad del proyecto en un entorno de trabajo con todos los softwares, configuraciones y datos necesarios, con la ventaja de reproducirlo en cualquier máquina.

³El uso de contenedores es relevante para la reproducibilidad, por eso se evalúan los siguientes criterios objetivos: portabilidad (se pueden reproducir en máquinas cuyos contenedores despliegan recursos compartidos), empaquetado (el software y los datos están compilados en una imagen binaria), reutilización de componentes (distribución entre usuarios de las imágenes sin necesidad de progresos intermedios), distribución (distribución de imágenes a través de plataformas) y versionado (integración de sistema de control de versiones en los archivos para la automatización la compilación de datos) (Boettiger, 2015).

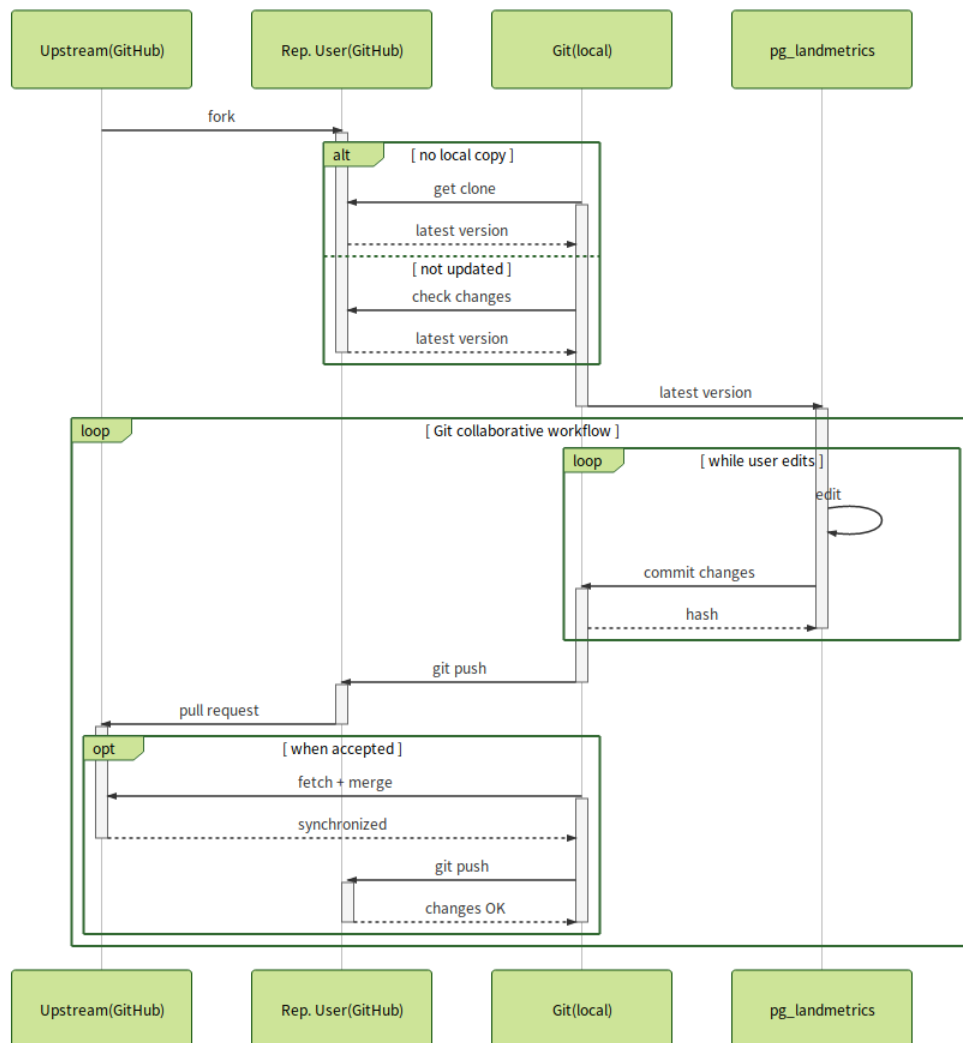


Figura 2.2 Flujo de proceso de trabajo colaborativo entre repositorios.

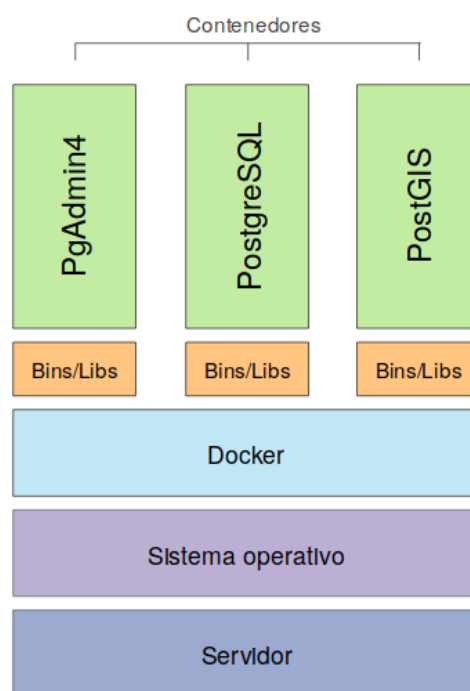


Figura 2.3 Contenerización del sistema operativo.

- **Docker Hub:** es una biblioteca digital donde se almacenan todas las imágenes Docker de aplicaciones en distintos repositorios. Desde aquí se obtuvieron las imágenes de las aplicaciones que se utilizaron.
- **Docker-compose:** es un fichero de configuración que despliega todos los servicios que sean necesarios durante el desarrollo de manera rápida y eficaz.

Otro de los procesos, que corresponde a esta subsección, es la integración continua del proyecto que conlleva la ejecución de control de versiones y la automatización. Desde el punto de vista del usuario, se ejecutaba el comando `docker-compose build` si la extensión había sido modificada. En el caso de que no estuviera modificada y no fuera necesario volver a construir el Docker-compose, solamente era necesario ejecutar el comando `docker-compose up` para iniciarlo. A partir de aquí, el Docker-compose orquestaba y se ocupaba de lanzar dos contenedores: PostgreSQL/PostGIS y PgAdmin. Una vez han sido lanzados dichos contenedores y se han recibido las notificaciones de que la conexión había sido satisfactoria, se iniciaba un loop mientras el usuario realizaba consultas a los contenedores. A continuación, ambos contenedores realizaban las consultas entre ambos y luego se enviaban al usuario los resultados de las consultas acompañado de una vista previa de ellos. Cuando ya eran adquiridos todos los resultados deseados, el loop se terminaba y se ejecutaba el comando `docker-compose down` para que el Docker-compose dejase de funcionar y el usuario recibiese una notificación por parte de este actor de que había sido apagado sin problema (ver figura 2.4).

2.1.3. Extensibilidad

El *makefile* (fichero encargado de organizar todo el código compilado de todos los programas que se deseen utilizar en la extensión) obtiene la capacidad para ampliar las funcionalidades de cualquier aplicación, como es el caso de la siguiente extensión:

- **PostgreSQL/PostGIS:** por un lado, PostgreSQL es un poderoso sistema de bases de datos relacional el cual ayuda a organizar todos los objetos en un conjunto de tablas y, en cuanto a PostGIS es una extensión que añade objetos geográficos a la base de datos relacional de PostgreSQL, pasando a ser una base de datos espacial. Ambas extensiones han permitido la construcción de consultas sobre las bases de datos utilizadas en el proyecto, tanto los de prueba como de los casos de estudio a partir de datos del SIOSE.

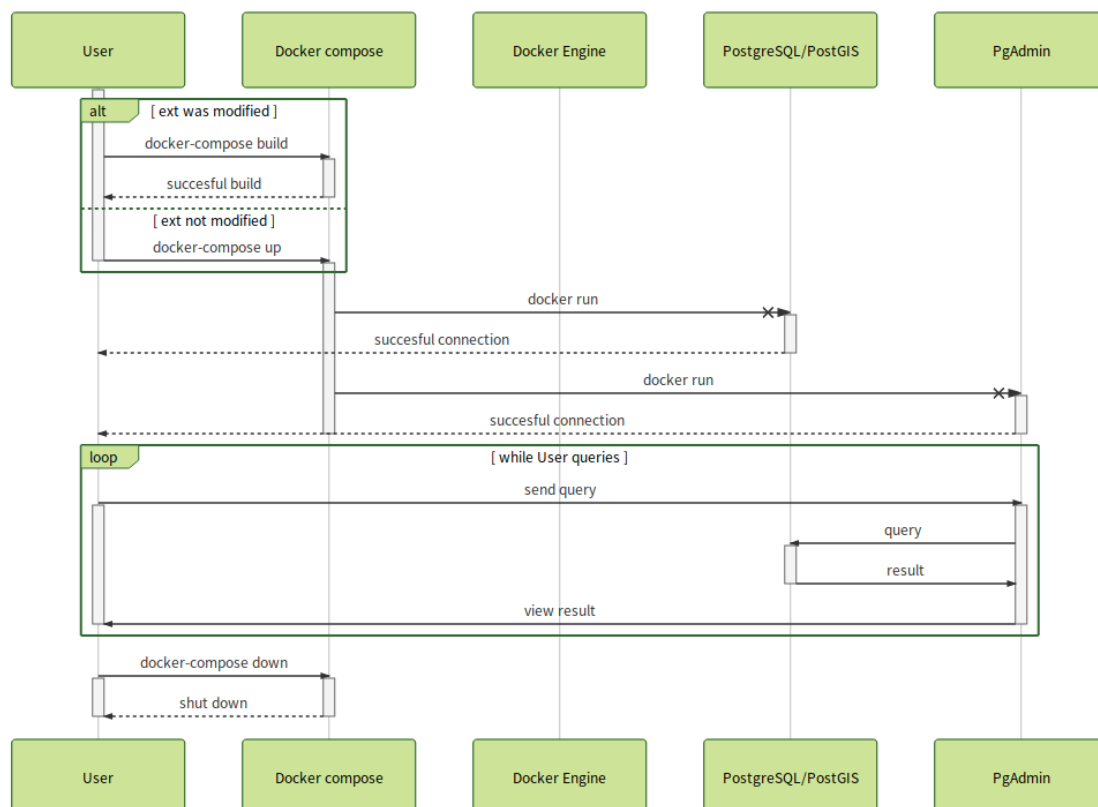


Figura 2.4 Flujo de proceso de integración continua.

Tabla 2.1 Atributos del primer conjunto de datos.

Nombre	Tipo de campo	Descripción
gid	integer	Identificador único de cada polígono
geom	geometry	Geometría del polígono
category	character varying (text)	Clasificación de la cobertura del suelo
svg_color	character varying (text)	Color según tipo de categoría

2.1.4. Aplicaciones

Tanto para el conjunto de datos como para las consultas y funciones de SQL, se utilizaron aplicaciones que realizan unas determinadas tareas:

- **PgAdmin4:** es una aplicación de código abierto capaz de administrar y gestionar bases de datos PostgreSQL. En este caso, desde una interfaz web donde se han construido todas las consultas, funciones, agregados,... etc.
- **QGIS 2.18:** es una aplicación de escritorio SIG, de código abierto o libre, que analiza, maneja y opera con datos vectoriales, datos ráster y bases de datos. Además facilita la conexión entre las bases de datos espaciales como PostGIS. Gracias a este software se ha elaborado un conjunto de datos que se ha utilizado para comprobar el funcionamiento de las métricas de paisaje.

2.2. Conjunto de datos

Antes de seleccionar las métricas de paisaje que han sido utilizadas para complementar el nuevo software, ha sido necesario elaborar y procesar conjuntos de datos que posteriormente se han utilizado para realizar comprobaciones del funcionamiento de las métricas. Así pues, por una parte se ha creado un paisaje ficticio y por otra, se han utilizado los datos del SIOSE de dos zonas de estudio como casos de estudio reales.

Para el paisaje ficticio, se ha digitalizado desde cero todos los polígonos que comprenden esta zona de estudio a partir de herramientas de geoprocésamiento, geometría, vectorial y edición, utilizando QGIS. En la tabla de atributos del shapefile, cada polígono tiene identificador único, geometría, clasificación según el tipo de cobertura del suelo y color según el tipo de categoría al que pertenece.

Tabla 2.2 My caption

Category	svg_color
Agricultural area	lightsalmon
Continuous urban	lightcoral
Discontinuous urban	lightpink
Forest	forestgreen
Main road	gray
Secondary road	lightgrey
River	royalblue
Water body	mediumblue

En este paisaje se ha optado por una clasificación de etiquetas simples. En cuanto al color de cada tipo de categoría se ha definido y escogido a partir de los 147 colores que presenta Scalable Vector Graphics (SVG) Specification⁴, además del apoyo de la clasificación de colores que especifica la leyenda del Corine Land Cover.v

Los colores definidos han sido utilizados posteriormente para elaborar y colorear las figuras de la documentación, tanto del trabajo como para la plataforma del proyecto en GitHub, a partir de la función SVG que se ha creado expresamente para ello (véase).

En el segundo conjunto de datos, se han utilizado los datos del SIOSE de dos zonas de estudio distintas para realizar una comprobación de los resultados de las métricas de paisaje a partir de casos reales y comparar ambos paisajes. La estructura de la tabla de atributos se compone de identificadores únicos, geometría, tipos de coberturas, información en XML, superficie, dos campos para almacenar nueva información y códigos numéricos de producción.

⁴<http://www.december.com/html/spec/colorsvg.html>

Tabla 2.3 Atributos del primer conjunto de datos.

Nombre	Tipo de campo	Descripción
ID_POLYGON	integer	Identificador único de cada polígono
GEOMETRY	geometry	Geometría del polígono
SIOSE_CODE	character varying (text)	Tipo de cobertura de cada polígono
SIOSE_XML	character varying (text)	Información completa del tipo de cobertura al que pertenece cada polígono, en formato XML
SUPERF_HA	character varying (text)	Superficie del polígono en hectáreas
OBSER_C	character varying (text)	Campo información auxiliar (texto)
OBSER_N	character varying (text)	Campo información auxiliar (numérico)
CODBLQ	character varying (text)	Código numérico al bloque de producción del SIOSE al que pertenece

Tabla 2.4 My caption

Nombre	Tipo de campo	Descripción
ID_POLYGON	string	Identificador único de cada polígono
GEOMETRY	geometry	Geometría del polígono
SIOSE_CODE	string	Tipo de cobertura de cada polígono
SIOSE_XML	string	Información completa del tipo de cobertura al que pertenece cada polígono, en formato XML
SUPERF_HA	real	Superficie del polígono en hectáreas
OBSER_C	string	Información auxiliar (texto)
OBSER_N	integer	Información auxiliar (numérico)
CODBLQ	integer	Código numérico al bloque de producción del SIOSE al que pertenece

2.3. Selección de métricas

2.4. Implementación/desarrollo de funciones en PostgreSQL

2.5. Documentación de la extensión

A lo largo del trabajo, se aplican una serie de lenguajes de marcado para la documentación de la extensión. Pero antes se obtienen conocimientos previos sobre ellos y su funcionamiento para utilizarlos durante el proyecto. Los lenguajes utilizados son:

- **Markdown**⁵ es un lenguaje ligero capaz de convertir texto plano a lenguaje HTML. Permite una escritura sencilla y conserva un diseño fácil de lectura. Es compatible con muchas plataformas. Este tipo de lenguaje es utilizado para documentar la extensión en la plataforma de GitHub y la descripción de cada una de las métricas de paisaje.
- **TeX**⁶ es el lenguaje que se utiliza en el sistema de textos LaTeX y que crea documentos con una alta calidad tipográfica. Se utiliza para escribir artículos o libros científicos, y desde hace tiempo este lenguaje se emplea por un gran número de usuarios. Este tipo de lenguaje es utilizado para redactar este trabajo. Para trabajar con este lenguaje, se utiliza la aplicación Texmaker que se ejecuta desde un contenedor Docker.
- **Scalable Vector Graphics (SVG)** es un lenguaje capaz de crear gráficos basados en vectores escalables a partir de archivos vectoriales en 2D y en formato XML. En esta década muchos de los navegadores web utilizan este tipo de lenguaje para sus gráficos. Gracias a este lenguaje, los gráficos no pierden calidad, pueden ser escalables y ocupan menos espacio en la memoria. Este tipo de lenguaje es utilizado para crear las figuras del trabajo.

Por ello, se desarrolla una función SQL, como primera propuesta, que dibuja y colorea los gráficos vectoriales que acompañan en el trabajo y en la documentación de la extensión en la plataforma GitHub. Para que las figuras tengan el color correspondiente al tipo de cobertura al que pertenecen, se aplica el código SVG que acompaña a cada polígono en la tabla de atributos del primer conjuntos de datos. La función SVG puede consultarse en el Anexo II.

⁵<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

⁶<https://www.latex-project.org/>

- **Mermaid**⁷ es un lenguaje de secuencia, que utiliza etiquetas similares a las que son empleadas en el lenguaje de marcado, capaz de generar gráficos a partir de texto por medio de JavaScript. Este tipo de lenguaje se ha utilizado para crear los diagramas de secuencia y el diagrama de Gantt.

Ejemplo de código 2.1 Crear una función para calcular el IDW (I)

```
1 SELECT St_Area(geom)/10000 FROM sample_patches_25830;  
2 SELECT St_Area(geom)/10000 FROM sample_patches_4326;
```

Ejemplo de código 2.2 Crear una función para calcular el IDW (I)

```
1 SELECT SUM(St_Area(geom))/10000, category FROM  
   sample_patches_25830 GROUP BY category;  
2 SELECT SUM(St_Area(geom))/10000, category FROM  
   sample_patches_4326 GROUP BY category;
```

Ejemplo de código 2.3 Crear una función para calcular el IDW (I)

```
1 SELECT SUM(St_Area(geom)) FROM sample_patches_25830;  
2 SELECT SUM(St_Area(geom)) FROM sample_patches_4326;
```

Ejemplo de código 2.4 Crear una función para calcular el IDW (I)

```
1 SELECT St_Distance(p1.geom, p2.geom)  
2 FROM sample_patches_25830 AS p1, sample_patches_25830 AS p2  
3 WHERE p1.id = 1 AND p1.id <> p2.id AND p2.category= "category"  
4 ORDER BY St_Distance (p1.geom, p2.geom)  
5 LIMIT 1;  
6  
7 SELECT St_Distance(p1.geom, p2.geom)  
8 FROM sample_patches_4326 AS p1, sample_patches_4326 AS p2  
9 WHERE p1.id = 1 AND p1.id <> p2.id AND p2.category= "category"  
10 ORDER BY St_Distance (p1.geom, p2.geom)  
11 LIMIT 1;
```

⁷<https://mermaidjs.github.io/>

Ejemplo de código 2.5 Crear una función para calcular el IDW (I)

```

1  SELECT SUM(St_Area(St_Buffer(geom, -100)))/10000 FROM
    sample_patches_25830 GROUP BY category;
2  SELECT SUM(St_Area(St_Buffer(geom, -100)))/10000 FROM
    sample_patches_4326 GROUP BY category;

```

Ejemplo de código 2.6 Crear una función para calcular el IDW (I)

```

1  SELECT SUM(St_Perimeter(geom)/St_Area(geom))*10000 FROM
    sample_patches_25830;
2  SELECT SUM(St_Perimeter(geom)/St_Area(geom))*10000 FROM
    sample_patches_4326;

```

Ejemplo de código 2.7 Crear una función para calcular el IDW (I)

```

1  /*
2  Patch Area - devuelve la suma del área del polígono dividido por
    10.000 (unidades: Hectáreas).
3  */
4  -- SAMPLE USAGE:
5  /*
6  SELECT (p_area(geom)).value As p_area FROM sample_patches_25830;
7  SELECT (p_area(geom)).value As p_area FROM sample_patches_4326;
8  */
9
10 CREATE OR REPLACE FUNCTION p_area(geom geometry)
11 RETURNS metric AS
12 $$
13
14 SELECT (1, St_Area(geom)/10000)::metric;
15
16 $$
17 LANGUAGE SQL
18 IMMUTABLE
19 RETURNS NULL ON NULL INPUT;
20
21 COMMENT ON FUNCTION p_area(geom geometry) IS 'Divide el área en
    metros cuadrados de un polígono por 10.000 para devolver un
    valor en Hectáreas.';

```

```

22
23
24 CREATE OR REPLACE FUNCTION p_area(geom geography)
25 RETURNS metric AS
26 $$
27
28 SELECT (1, St_Area(geom)/10000)::metric;
29
30 $$
31 LANGUAGE SQL
32 IMMUTABLE
33 RETURNS NULL ON NULL INPUT;
34
35 COMMENT ON FUNCTION p_area(geom geography) IS 'Divide el área en
    metros cuadrados de un polígono por 10.000 para devolver un
    valor en Hectáreas.';

```

Ejemplo de código 2.8 Crear una función para calcular el IDW (I)

```

1  /*
2  Total (Class) Area - devuelve la suma de las áreas (mš) de todos
    los polígonos correspondientes al tipo de polígono, dividido
    por 10.000 (unidades: Hectáreas).
3  */
4  -- SAMPLE USAGE
5  /*
6  SELECT c_totalarea_state(ARRAY[('Agricultural area',('Total
    Class Area'::text, 10, 'Ha.'::text))::labeled_metric], geom,
    category) FROM sample_patches_25830;
7  SELECT c_totalarea_state(ARRAY[('Agricultural area',('Total
    Class Area'::text, 10, 'Ha.'::text))::labeled_metric], geom,
    category) FROM sample_patches_4326;
8  */
9
10 CREATE OR REPLACE FUNCTION c_totalarea_state(
11     current_state metric_labeled,
12     geom geometry,
13     category text)
14 RETURNS metric_labeled

```

```
15     LANGUAGE 'sql'
16
17 AS
18 $BODY$
19
20 WITH inputs AS (
21     SELECT current_state AS cstate
22 ), melt AS (
23     SELECT unnest((cstate).pairs) AS m2 FROM inputs
24     UNION
25     SELECT (category, (p_area(geom)).value)::
26         metric_labeled_pair AS m2
27 ), summarize AS (
28     SELECT (m2).label, SUM((m2).value) AS value FROM melt
29     GROUP BY (m2).label
30 )
31 SELECT (9, ARRAY(SELECT (label, value)::metric_labeled_pair FROM
32     summarize))::metric_labeled;
33
34 $BODY$;
35
36 -- SAMPLE USAGE
37 -- SELECT c_totalarea(geom,category) FROM sample_patches_25830;
38
39 CREATE AGGREGATE c_totalarea(geometry, text)(
40     SFUNC=c_totalarea_state,
41     STYPE=metric_labeled,
42     INITCOND='(9,{})'
43 );
44
45 COMMENT ON AGGREGATE c_totalarea(geom geometry, category text)
46 IS 'Calcula la suma de las áreas de los polígonos de la misma
47     categoría dividido por 10.000 para devolver un valor en
48     Hectáreas.';
49
50 CREATE OR REPLACE FUNCTION c_totalarea_state(
```



```

48         current_state metric_labeled,
49         geom geography,
50         category text)
51     RETURNS metric_labeled
52     LANGUAGE 'sql'
53
54 AS
55 $BODY$
56
57 WITH inputs AS (
58     SELECT current_state AS cstate
59 ), melt AS (
60     SELECT unnest((cstate).pairs) AS m2 FROM inputs
61     UNION
62     SELECT (category, (p_area(geom)).value)::
63         metric_labeled_pair AS m2
64 ), summarize AS (
65     SELECT (m2).label, SUM((m2).value) AS value FROM melt
66     GROUP BY (m2).label
67 )
68 SELECT (9, ARRAY(SELECT (label, value)::metric_labeled_pair FROM
69     summarize))::metric_labeled;
70
71 $BODY$;
72
73 -- SAMPLE USAGE
74 -- SELECT c_totalarea(geom,category) FROM sample_patches_4326;
75
76 CREATE AGGREGATE c_totalarea(geography, text)(
77     SFUNC=c_totalarea_state,
78     STYPE=metric_labeled,
79     INITCOND='(9,{})'
80 );
81
82 COMMENT ON AGGREGATE c_totalarea(geom geography, category text)
83 IS 'Calcula la suma de las áreas de los polígonos de la misma
84     categoría dividido por 10.000 para devolver un valor en
85     Hectáreas.';

```

Ejemplo de código 2.9 Crear una función para calcular el IDW (I)

```
1  /*
2  Total Area - devuelve el total del área (mš) del paisaje
    dividido por 10.000 (unidades: Hectáreas).
3  */
4  -- SAMPLE USAGE:
5  /*
6  SELECT (l_totalarea(geom)).value FROM sample_patches_25830;
7  SELECT (l_totalarea(geom)).value FROM sample_patches_4326;
8  */
9
10 CREATE OR REPLACE FUNCTION l_totalarea_state(metric,geometry)
11     RETURNS metric AS
12 $$
13     SELECT $1 + (p_area($2)).value;
14 $$
15 LANGUAGE 'sql' IMMUTABLE;
16
17
18 CREATE AGGREGATE l_totalarea(geometry)(
19     SFUNC=l_totalarea_state,
20     STYPE=metric,
21     INITCOND='(0,0)',
22 );
23
24 COMMENT ON AGGREGATE l_totalarea(geometry) IS 'Calcula el área
    total del paisaje dividida por 10.000 para devolver un valor
    en Hectáreas.';
25
26
27 CREATE OR REPLACE FUNCTION l_totalarea_state(metric,geography)
28     RETURNS metric AS
29 $$
30     SELECT $1 + (p_area($2)).value;
31 $$
32 LANGUAGE 'sql' IMMUTABLE;
```

```

33
34
35 CREATE AGGREGATE l_totalarea(geography)(
36     SFUNC=l_totalarea_state,
37     STYPE=metric,
38     INITCOND='(0,0)'
39 );
40
41 COMMENT ON AGGREGATE l_totalarea(geography) IS 'Calcula el área
    total del paisaje dividida por 10.000 para devolver un valor
    en Hectáreas.';

```

Ejemplo de código 2.10 Crear una función para calcular el IDW (I)

```

1 SELECT (p_corearea(geom, 50)).value FROM sample_patches_25830;
2 SELECT (p_corearea(geom, 50)).value FROM sample_patches_4326;

```

Ejemplo de código 2.11 Crear una función para calcular el IDW (I)

```

1 SELECT c_totalarea(geom,category) FROM sample_patches_25830;
2 SELECT c_totalarea(geom,category) FROM sample_patches_4326;

```

Ejemplo de código 2.12 Crear una función para calcular el IDW (I)

```

1 CREATE TABLE metric_meta_unit (
2     id serial primary key,
3     sunit text NOT NULL,
4     lunit text NOT NULL
5 );
6
7 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('Ha.', '
    Hectáreas');
8 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('m.', 'Metros'
    );
9 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('mš.', 'Metros
    ');
10 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('%', '
    Porcentaje');
11 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('', 'None');

```

```

12 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('m/Ha.', '
    Metros por Hectárea');
13 INSERT INTO metric_meta_unit(sunit,lunit) VALUES ('num/100 Ha.',
    'Número por 100 Hectáreas');
14
15
16 CREATE TABLE metric_meta (
17     id serial primary key,
18     sname text NOT NULL,
19     lname text NOT NULL,
20     unit_id integer NOT NULL
21 );
22
23 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('AREA', '
    Patch Area',1);
24 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PERIM', '
    Patch Perimeter',2);
25 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PARA', '
    Perimeter Area Ratio',3);
26 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('SHAPE', '
    Shape Index',5);
27 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('CORE', '
    Core Area',1);
28 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('NCORE', '
    Number of Core Areas',5);
29 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('CAI', '
    Core Area Index',4);
30 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('ENN', '
    Euclidean Nearest Neighbour Distance',2);
31 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('CA', '
    Total (Class) Area',1);
32 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PLAND', '
    Percentage of Landscape',4);
33 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('TE', '
    Total Edge',2);
34 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('ED', 'Edge
    Density',6);
35 INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('TCA', '
    Total Core Area',1);

```

```
36| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('CPLAND', '
    Core Area Percentage of Landscape',4);
37| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('NP', '
    Number of Patches',5);
38| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PD', '
    Patch Density',7);
39| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('TA','Total
    Area',1);
40| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('TE','Total
    Edge',5);
41| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('ED','Edge
    Density',6);
42| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('NP','
    Number of Patches',5);
43| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PD','Patch
    Density',7);
44| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PR','Patch
    Richness',5);
45| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PRD','
    Patch Richness Density',7);
46| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('SHDI','
    Shannon Diversity Index',5);
47| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('SHIDI','
    Simpson Diversity Index',5);
48| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PC','
    Proportion Class',4);
49| INSERT INTO metric_meta(sname,lname,unit_id) VALUES ('PL','
    Proportion Landscape',4);
50|
51|
52|
53| -- id: metric_meta id
54| CREATE TYPE metric AS (
55|     id integer,
56|     value numeric
57| );
58|
59|
60| CREATE TYPE metric_labeled_pair AS
```

```
61  (  
62      label text,  
63      value numeric  
64  );  
65  
66  -- id: metric_meta id  
67  CREATE TYPE metric_labeled AS  
68  (  
69      id integer,  
70      pairs metric_labeled_pair[]  
71  );
```

3. RESULTADOS Y DISCUSIÓN

Puntos de interés:

- What?
- How?
- When?
- Who?
- Where?
- why or for what?

3.1. Resultados

Ejemplo de código 3.1 Crear una función para calcular el IDW (I)

```
1  -- Function: _plr_idw2ascii(text, text, integer, boolean)
2  -- DROP FUNCTION _plr_idw2ascii(text, text, integer, boolean);
3  CREATE OR REPLACE FUNCTION _plr_idw2ascii(text, text, integer,
4      boolean)
5      RETURNS text AS
6  $BODY$
7  --codigo de R que realiza una interpolacion
8  $BODY$
9      LANGUAGE 'plr' VOLATILE
10     COST 100;
11 ALTER FUNCTION _plr_idw2ascii(text, text, integer, boolean)
12     OWNER TO postgres;
```

3.2. Discusión

4. CONCLUSIONES Y TRABAJO FUTURO

Puntos de interés:

- What?
- How?
- When?
- Who?
- Where?
- why or for what?

Conclusiones

Trabajo

Bibliografía

- [1] B. Zaragozí, A. Belda, J. Linares, J. E. Martínez-Pérez, J. T. Navarro, and J. Esparza. A free and open source programming library for landscape metrics calculations. *Environmental Modelling and Software*, 31:131–140, 2012. ISSN 13648152. doi: 10.1016/j.envsoft.2011.10.009. URL <http://dx.doi.org/10.1016/j.envsoft.2011.10.009>.

A. ANEXO

Windows OS

Ejemplo de código A.1 Crear una función para calcular el IDW (I)

```
1  -- SELECT patches_toSVG(id::integer, geom, category, svg_color,
   p_area(geom)) FROM sample_patches_25830;
2  -- SELECT patches_toSVG(id::integer, geom, category, svg_color,
   p_area(geom)) FROM sample_patches_4326;
3
4  CREATE OR REPLACE FUNCTION patches_toSVG(gid integer, geom
   geometry, category text, svg_color text, metric metric)
5  RETURNS TEXT AS
6  $func$
7
8  WITH sp(gid,geom,category, svg_color) AS (VALUES
9                                     (gid,geom,category,
                                       svg_color)),
10 paths (svg) AS (
11     SELECT array_to_string(
12         array_agg(
13             concat(
14                 '<path d= "', ST_AsSVG(geom,0), '" ', ' stroke='
15                 'black" stroke-width="2" fill=" ', svg_color, '
16                 " />'
17             ), ''
18         )
19     FROM sp
20 ),
21 texts (svg) AS(
22     SELECT array_to_string(
23         array_agg(
24             concat(
25                 '<text x=" ', st_x(st_pointonsurface(geom)), '" y="
26                 ', -st_y(st_pointonsurface(geom)), '" font-size
```

```

                                ="100px" fill="black" text-anchor="middle">',
                                metric.value, ' ', metric.units, '</text>'
23                                )), '')
24    FROM sp
25 ),
26 env AS (
27     SELECT st_extent(geom) AS extent
28     FROM sp
29 ),
30 dims AS (
31     SELECT st_ymin(extent) AS bottom, st_ymax(extent) AS top,
32            st_xmin(extent) AS lefthand, st_xmax(extent) AS righthand
33     FROM env
34 )
35 SELECT concat(
36     '<svg width="100%"
37     preserveAspectRatio="xMinYMin meetOrSlice"
38     style="border: 1px solid #cccccc;">',
39     '<g transform= "translate(0, 0) scale (0.1, 0.1) translate('
40     , - dims.lefthand, ',', dims.top, ')">',
41     paths.svg, texts.svg,
42     '</g>',
43     '</svg>')
44 FROM dims, paths, texts;
45 $func$
46 LANGUAGE sql;

```