[PD-02-037] Open Source Software Development

Abstract

Open source geospatial software is now ubiquitous – it is used and supported across industries, in government agencies, as well as research institutions and academia. This entry describes general principles of open source software development and provides an overview of the development platforms and tools. Specific focus is on the Open Source Geospatial Foundation's software stack, its development principles, practices, and initiatives. Several additional major open source software systems with geospatial support are also briefly discussed with examples of open source applications developed by integrating multiple libraries and packages.

Keywords: code spring, FOSS, license, OSGeo, programming, programming language, software ecosystem, software library, software stack, version control

Author & citation

Petras, V., Mitasova, H., and Petrasova, A. (2021). Open Source Software Development. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2021 Edition), John P. Wilson (ed.). DOI: 10.22224/gistbok/2021.2.4

Explanation

1. Definitions

branch: a version of the source code within a repository dedicated to a particular purpose, e.g., working on a development version, specific release, or new feature.

computational notebooks: documents combining source code, computation outputs, and text including figures which, given the right environment, can be interactive and allow recomputation.

continuous integration: effort or practice of combining and testing developer's changes with the rest of the latest source code as often as possible. More generally, any automated or mostly-automated testing performed on any and all versions of the source code.

fork, soft fork, hard fork: a separate version of software or its source code. Soft fork is meant for inclusion back into the original source while hard fork is meant to be a new independent piece of software.

issue tracker, bug tracker, ticketing system: a system for creating and managing

reports about bugs and requests for features (features are desired behaviors of the software, bugs refer to problems or issues with the software).

open source license: a software license fulfilling the Open Definition, the Open Source Definition, and/or the Free Software Definition ("four freedoms"). Typically, a license approved by the Open Source Initiative.

software ecosystem: a collection of software projects and their relationships which are developed within or tied to the same community, company, service, or a key software project.

pull request, merge request: proposal to maintainers of a project to include a new piece of source code.

software organization: a formal, usually legal and non-profit, entity which oversees or otherwise supports development of open source software creating a vendor-neutral platform for developers and users.

software stack: a collection of interoperable programs that can be combined to develop a workflow, to solve a set of problems, or to work towards a given goal.

source code: structured text assembled, compiled, or interpreted as or to an executable program. Often more generally, a collection of all files of a specific software project including documentation, figures, and data.

source code repository: directory or web-hosted archive with source code almost always managed using version control.

source code hosting service: cloud-based service with a source code repository and associated services for developers, such as version control system, issue tracker, discussion board, code and contribution analytics, and continuous integration.

version control: system for management of source code, its changes, and versions.

2. Introduction

Open source software is now ubiquitous – it is used and supported across industries, in government agencies, as well as research institutions and academia. The open source software development model has been adopted by major software companies (e.g., Google, IBM, Apple, Microsoft), and it is well established in computing related research and education. The geospatial domain is no exception as documented in topics within the GIS&T Body of Knowledge such as Python for GIS, R for Geospatial Analysis and Mapping, WebGIS Programming, GIS APIs, GDAL/OGR and IO Libraries, GIS and Computational Notebooks, and Openness.

Open source geospatial software includes a broad range of libraries, tools, applications, and platforms developed and released under different Open Source Initiative (OSI) approved licenses. These licenses allow for software to be freely used, modified, and shared (Open Source Initiative, 2021). Open source software development has many forms but there are

common principles along with infrastructure, platforms and tools, with several core projects providing foundation for the open source geospatial software ecosystem (Coetzee, 2020).

3. Open Source Software Development

3.1 General Concepts and Principles

Open source software can be described from a theoretical, license perspective, and from a software development perspective. The theoretical perspective is covered by the Body of Knowledge topic on Openness and the licenses are described in the topics on Commercialization of GIS Applications and Licensing of GIS Applications (forthcoming). This topic focuses on the software development aspect. Although the term open source software is used throughout this entry, the ideas of free and open source software (FOSS), sometimes referred to as free, libre, and open source software (FLOSS), are covered as well, highlighting the freedoms to use the software for any purpose, modify the software to suit your needs, and share the software and the changes you make (Stallman, 2002). This entry covers the development where all source code is released under OSI licenses. However, many licenses allow combining proprietary applications with open source extensions and many core open source libraries, such as GDAL and PDAL, use licenses which allow their incorporation into proprietary software.

Open source projects start in different ways. Some are initiated by a single developer, later attract more contributors, and then create formal contributor structures. Others start as government funded research projects, become widely used and developed beyond the initial project. Finally, there are projects originated by companies as part of their software development model that are later transformed into community-driven projects as other companies, institutions, or individuals become contributors. Although the software evolutions and contributor community structures vary a lot, successful projects usually grow beyond their initial developer community or institution into projects with many contributors with different affiliations and create their own or join formal vendor-neutral organizations such as Open Source Geospatial Foundation (OSGeo) or Eclipse Foundation.

The open source software development process typically involves a broad community of **contributors** and a tighter, smaller group of **maintainers** or **core developers** collaborating on a software project. Anybody can obtain and modify a copy of the source code, however, only the maintainers control what is included in the software releases. The decision process is open and anybody can comment on the proposed changes. Project maintainers are often selected based on merit and approved by current maintainers, a **steering committee**, or a **core team of developers**.

Developers and users of open source software usually form a tightly knit community. The developers themselves are often also users of the software while users have access to communication channels used by developers such as ticket systems or mailing lists. Any user can open issues or tickets to report problems with the software, i.e., bugs, or to ask for new features. In this way, users have direct access to developers and can influence development of the software. Nevertheless, the most direct way to influence the development is to contribute source code.

The high level of freedom in shaping the software is well demonstrated in the process of

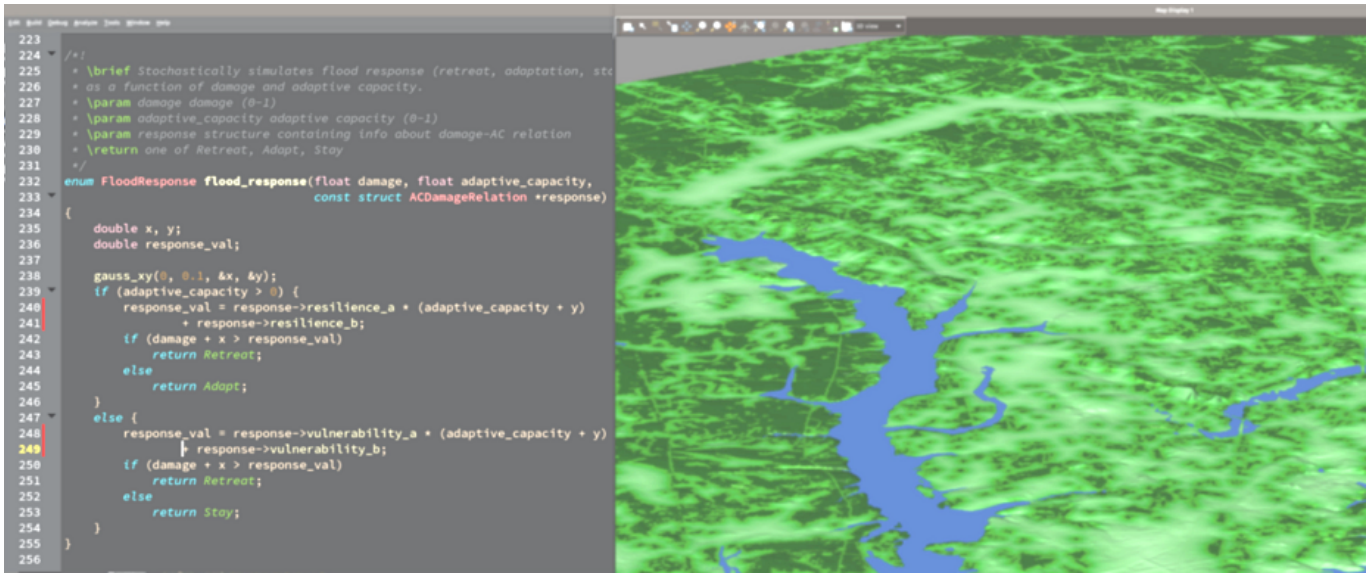
creating forks. To propose a change to an open source project, contributors typically create a **fork**, a **branch**, or simply a copy of the software source code. This copy can be freely modified and when the changes are made, the contributor can propose for these changes to be included in the original source code by a process often referred to as **pull request**, **merge request**, or **patch submission**. When the intention is to include changes into the original source code, we can call the fork a **soft fork**. However, the developer has freedom to keep the changes temporarily internal to an organization. An individual developer, organization, or a part of the community, has freedom to go even further and, when there is a need, create what we can call a **hard fork**, a completely new software project based on the source code of the original software.

3.2 Development Management Tools

Open source software development is a collaborative process supported by systems and tools for management of the source code contributions, its documentation, quality control, and testing. The presence and state of these tools in the development process of a project shows the level of its maturity.

Version control systems and source code hosting are essential tools for management of source code by tracking changes by each contributor. Current software development is often managed using **Git** through cloud-based hosting services such as **GitHub** or **GitLab**. The hosting services provide additional capabilities including issue tracking, continuous integration, as well as analytics of contributions and code evolution. They provide tools to build, test and deploy the code directly from the hosting service. Examples of GitHub repositories for open source geospatial projects are QGIS, GRASS GIS, GDAL, Fiona, or pywps.

Quality control, testing, and reviews ensure consistent quality of open source geospatial software. New contributions and changes are first tested automatically using quality control tools such as **static code analysis tools** and other so-called **linters** which check different aspects of the code. Next phase is automated testing where a set of predefined functions runs with the new code and the result is compared to reference values. Two more steps are then performed: a **code review** by maintainers and a **manual test** of the change by maintainers or interested users. The goal of the code review is to improve the quality of the code design beyond what automated tools can do and evaluate possible interactions with other features or components of the code. Manual testing (Figure 1), on the other hand, focuses on the behavior of the proposed change. Uniquely to open source, these experimental versions, latest development version, and any pre-release versions are available to the public as source code or as installable binaries so that any user can test and evaluate the new version of the software before a new release is made. Also uniquely to open source, results of the tests and reviews are publicly available from the time a change was first proposed. For example, the QGIS project (QGIS, 2021) automatically builds the proposed source code on different operating systems, tests compliance with Python code best practices, runs a series of testing functions including an Open Geospatial Consortium (OGC) compliance tests of QGIS Server, and performs several other automated actions besides a review and manual code testing process which is centered around a pull request on GitHub.



```
/*!
 * \brief Stochastically simulates flood response (retreat, adaptation, sta
 * as a function of damage and adaptive capacity.
 * \param damage damage (0-1)
 * \param adaptive_capacity adaptive capacity (0-1)
 * \param response structure containing info about damage-AC relation
 * \return one of Retreat, Adapt, Stay
 */
enum FloodResponse flood_response(float damage, float adaptive_capacity,
                                  const struct ACDamageRelation *response)
{
    double x, y;
    double response_val;

    gauss_xy(0, 0.1, &x, &y);
    if (adaptive_capacity > 0) {
        response_val = response->resilience_a * (adaptive_capacity + y)
                + response->resilience_b;
        if (damage + x > response_val)
            return Retreat;
        else
            return Adopt;
    }
    else {
        response_val = response->vulnerability_a * (adaptive_capacity + y)
                + response->vulnerability_b;
        if (damage + x > response_val)
            return Retreat;
        else
            return Stay;
    }
}
```

Figure 1: Examination of source code for modeling flood response and a simulated flood. Such examination of an open source software is not only possible, but can also lead to further development of the software or its extensions as the code can be modified and built upon. Source: authors.

Release, packaging, and distribution are crucial steps for delivering the software to users. Prior to a release, the development team and the users community focus on testing and inclusion of the most critical features or fixes as well as several non-coding tasks such as software documentation and translation. When the source code is ready, a new release is **tagged** in the version control system which formally marks creation of the release followed by the build and packaging. In addition to building binaries and executables from the source code, packaging involves management of compile-time and runtime dependencies, integration with other software or operating systems, and creation of **installers** or packages. The packaging and distribution depends on the platform ranging from independent executables and installers for Microsoft Windows downloaded from web sites to packages well-integrated with the system downloaded from centralized repositories of Linux distributions. Given the complexity of software packaging and subsequent distribution, many variations exist. Several projects focus on distribution of software for specific domains or operating systems such as Anaconda for data science and MacPorts for macOS. R and Python themselves include tools to install new packages. Yet another approach is **containerization** (more precisely, operating-system-level virtualization) with engines such as Docker or Singularity which encapsulate software including all its dependencies in a single binary which can be executed anywhere where a given container engine is available. Typically, more than one distribution option is available to support various user needs, for example, GDAL is available through Anaconda, Docker, and several other distribution channels.

Documentation, manuals, and tutorials are essential for continuous, team-based development and for user communities. Each project maintains a developer (programmer) manual or an Application Programming Interface (API) documentation using tools for management of the code documentation such as Doxygen and Sphinx. **Wikis**, or more recently interactive notebooks such as **Jupyter Notebooks** and **JupyterLab**, are used to

collaboratively develop tutorials as “living documents” (see GIS and Computational Notebooks). Specifically, Jupyter Notebooks and JupyterLab can be published through the cloud service Binder which allows readers to view the documentation and interact with the code. Recent developments in Binder, GitHub, and GitLab promise that not only learning, but also contributing to the projects will be possible through these environments. Overall, services such as GitHub and GitLab are particularly suitable for individual developers and small teams because in addition to code management, they provide all the tools needed for creating public documentation and presentation of the software, especially when combined with external tools such as Binder.

Extensions, addons, modules, packages, and plugins extend the functionality of open source geospatial software by integrating code from a broader community, usually published through a centralized repository. These extensions, contributed and maintained by independent developers follow given project standards to ensure seamless integration with the core code. While different technologies are used in the background and different terms are used to describe the system, the basic principles are usually the same. For example, QGIS has plugins (e.g., Vitalis et al., 2020) which provide additional tools through extension of the main user interface and GRASS GIS has addons which, after installing, are indistinguishable from the core tools (Neteler et al., 2012, Petrasova et al., 2016).

Collaboration and communication tools facilitate team work by contributors who are often spread across the globe. Source code hosting services, such as GitHub have become the main collaboration hubs while planning and coordination of initiatives is often done on wikis (for example, OSGeo Wiki). Mailing lists are a traditional communication tool, popular for its universality while other tools such as IRC or Gitter are used for their web interfaces or instant messaging features. Long-standing projects have extensive, searchable, on-line archives going back several decades.

The usage and status of software development management tools in a project is usually a good indicator of its health and level of maturity. For example, new, starting projects often lack formal releases or packaging. Mature projects have a long list of past releases, user-focused and developer-focused documentation, and extensive history of source code changes from multiple contributors. Unmaintained projects are characterized by unclear access to the source code, little or no visible activity, and lack of opportunities for submitting feedback such as missing issue tracking. Healthy projects measure source code quality, use continuous integration, and possibly have a stream of new contributions. Number of working extensions, their recent updates, mailing list traffic, or other discussions often indicate the size or activity of the community. Umbrella software organizations such as The Apache Software Foundation, the GNU Project, or OSGeo set formal standards which the member projects must meet if they wish to join the organization. In turn, a project’s membership in such an organization demonstrates its compliance with set standards indicating a certain level of quality and sustainability of the software. The OSGeo approach, as an example of open source geospatial software umbrella organization, is discussed in more detail in the following section.

4. OSGeo Software Ecosystem

Open source geospatial software has been developed since the 1980s and by the year 2000

numerous packages were available. Coordination of projects was limited and the software was difficult to navigate and evaluate. The need to organize the rapidly growing field and address interoperability issues has driven the founding of the OSGeo Foundation in 2006. OSGeo is a not-for-profit organization with the mission to “support the collaborative development of open geospatial technologies, data, and education, and to promote their widespread use” (OSGeo, 2021). OSGeo serves as an umbrella organization for projects that have become the foundation of the open source geospatial software ecosystem providing libraries, platforms, and applications used by major geospatial software systems within and outside of OSGeo.

4.1 OSGeo Projects

OSGeo supports two groups of projects: **Graduated OSGeo projects** which passed the incubation process and **OSGeo Community Projects** not yet fully compliant but fulfilling the basic open source software requirements (OSGeo Incubation Committee, 2021).

Graduated OSGeo projects (Table 1) passed a rigorous incubation to ensure that the project is mature and sustainable, fulfills the conditions stated by its open source license and has an open and active user and developer community. The incubation covers both the management and technical aspects of development and the details are provided in the OSGeo graduation checklist (OSGeo Incubation Committee, 2021).

Management of software development supports an open developer community with transparent communication and active collaboration. Developers must come from several different organizations to ensure long-term project sustainability. The project steering committee (PSC, for example, PostGIS PSC, MapBender PSC, or Geoserver PSC) must offer both transparent decision-making and opportunities for new members to participate. Open decision-making encourages the developers to take ownership of the project and facilitates knowledge transfer from experienced to new team members. Projects must have a clear contribution policy and code of conduct document. All code contributors agree to abide by the project's license policy, and this agreement must be documented and archived. Technical aspects of development require that the code development must be supported by version control and an issue tracker, both user and developer documentation must be available, and the procedures for release and testing of the software must be well-defined.

Table 1. Graduated OSGeo Projects: a software stack of interoperable packages that can be combined to develop geospatial tools and applications.

Project	Purpose	Implementation and API languages	Link
deegree	OGC web services	Java	https://www.deegree.org/
GDAL (GDAL/OGR)	Format translator library, GIS processing engine	C++, C	https://gdal.org/
GeoMoose	Web GIS portal	JavaScript	https://www.geomoose.org/
GeoNetwork	Metadata catalogue	Java	https://geonetwork-opensource.org/
GeoNode	Geospatial content management system	JavaScript, Python	https://geonode.org/

Project	Purpose	Implementation and API languages	Link
GEOS	Spatial library	C++, C	https://geos.osgeo.org/
GeoServer	OGC web services	Java	http://geoserver.org/
GeoTools	GIS processing engine	Java	https://geotools.org/
GRASS GIS*	GIS processing engine, desktop GIS	C, C++, Python	https://grass.osgeo.org/
gvSIG Desktop*	Desktop GIS	Java	http://www.gvsig.com/
Mapbender	Geoportal framework	PHP, JavaScript	https://mapbender.org/en/
MapServer	OGC web services	C	https://mapserver.org/
Marble	Virtual Globe	C++	https://marble.kde.org/
OpenLayers	Browser mapping library	JavaScript	https://openlayers.org/
Orfeo Toolbox	image processing library	GLSL, C++	https://www.orfeo-toolbox.org/
OSGeoLive	Open source geospatial GNU/Linux distribution	n/a	https://live.osgeo.org
PostGIS	Spatial Database	C, PL/pgSQL, SQL	https://postgis.net/
PROJ	CRS transformation library	C, C++	https://proj.org/
pycsw	Metadata catalogue	Python	https://pycsw.org/
PyWPS	Web processing service	Python	https://pywps.org/
QGIS*	Desktop GIS	C++, Python	https://qgis.org/

* projects with server or mobile implementations outside the graduated OSGeo project

OSGeo Community projects (Table 2) have lower requirements and usually are newer, less established projects developed under an open source license. The projects must be open to contributions but broader users community is not required. These projects foster innovation and build their developers and users community until they are mature enough to enter incubation and become a Graduated OSGeo project. Several current community projects are going through the incubation process.

Table 2. OSGeo Community Projects

Project	Example or Applications
Geospatial libraries	FDO, Mesh Data Abstraction Layer (MDAL), OWSlib, OSSIM, Pronto Raster
Browser-facing GIS	GC2/Vidi, Geomajas, GeoExt, GeoStyler, mapfish, MapGuide Open Source, Oskari*
Web services	actinia, GeoWebCache, ZOO-project*, istSOS*, pygeoapi
Spatial database / Data stores	MobilityDB, Rasdaman, pgRouting
Image analysis platform	Opticks*
Distribution platform	OSGeo4W, Portable GIS
Monitoring and testing tools	GeohealthCheck, TEAM Engine*
Interaction with web services	GeoServer Client PHP, Loader

* projects in OSGeo incubation

4.2 OSGeo Software Development Initiatives

Software development is independently managed by the individual OSGeo projects or entities contributing to these projects, however, many aspects of project and software development infrastructure are similar. OSGeo helps to pool resources to, for example, host project websites or manage a GitHub organization as a common space to support collaboration. OSGeo also plays an important role in several development initiatives.

Code sprints or community sprints are self-organized events bringing together, virtually or in-person, developers from one or more projects to discuss important development issues, such as new features or interoperability, and to work on bug fixes, implementation, or documentation. Hands-on coding is an important part of the sprint, however, the fact that the event is synchronous and that both developers and users are present creates unique opportunities for resolving complex issues. There is an annual OSGeo sprint, sprints associated with conferences, numerous project-specific sprints as well as sprints co-organized between OSGeo and other organizations with similar goals (for example Apache Software Foundation-OGC-OSGeo code sprint). Open source geospatial developers and users meet annually at a global FOSS4G conference and at numerous regional FOSS4G meetings.

OSGeo serves as a mentor organization for geospatial projects (OSGeo projects and smaller guest geospatial projects) in the Google Summer of Code program providing funded opportunities for students to contribute to open source geospatial software under the mentorship of experienced open source software developers.

Software distribution is another area where combining efforts resulted in three major OSGeo-centered software distribution projects. OSGeo4W is a package manager for installing OSGeo and related software on Microsoft Windows. Homebrew-osgeo4mac, although not officially recognized at the time of writing, is focused on bringing OSGeo software to macOS using the Homebrew package manager. Finally, OSGeoLive is a virtual machine aimed at teaching and exploring OSGeo and other open source geospatial software in an ad-hoc lab setting. Although these hardly cover all the distribution options, many users encounter at least one of those.

5. Other Open Source Geospatial Software Development

There are numerous geospatial FOSS projects developed independently from OSGeo, but many rely on the core OSGeo projects' libraries for interoperability and several are distributed on OSGeoLive. The largest projects are covered as separate topics including comprehensive references to libraries and applications, for example R, Python, JavaScript, or PySAL.

Other large projects include libraries to support development of applications across many disciplines and technologies, such as point cloud data visualization and processing (PDAL, plas.io, potree), virtual globes (Cesium, NASA WorldWind, Pirotti et al., 2017), field mapping (Geopaparazzi), ocean and atmospheric sciences (Generic Mapping Tools - GMT), or modeling earth surface processes (Landlab, Barnhart et al., 2020). GeoBlacklight provides a

discovery engine for geospatial data and uDig is a desktop application framework, built with Eclipse Rich Client technology. Many general tools provide specialized geospatial functionality such as JuliaGeo for the data science language Julia or mapping functions in D3.js.

OpenStreetMap (OSM), which provides open local map data collected by communities across the world, has a full stack of tools and data formats dedicated to collecting, analyzing, and visualizing OSM data (Mocnik et al., 2018). Its API supports development of GIS applications for fetching and saving raw geospatial data from and to the OpenStreetMap database.

Many open source geospatial applications and tools combine domain specific libraries with relevant libraries from the OSGeo software stack to create open source geospatial solutions. For example, OpenDroneMap and WebODM combine GDAL, PDAL, GRASS GIS, potree, Leaflet, PostGIS, and computer vision libraries to support UAS mapping and associated data analysis (Figure 2). Tangible Landscape (Petrasova et al., 2018) integrates sensor communication, point cloud data processing, GRASS GIS, domain specific models, and 3D rendering with Blender to provide tangible interaction with geospatial data and simulations through physical models.

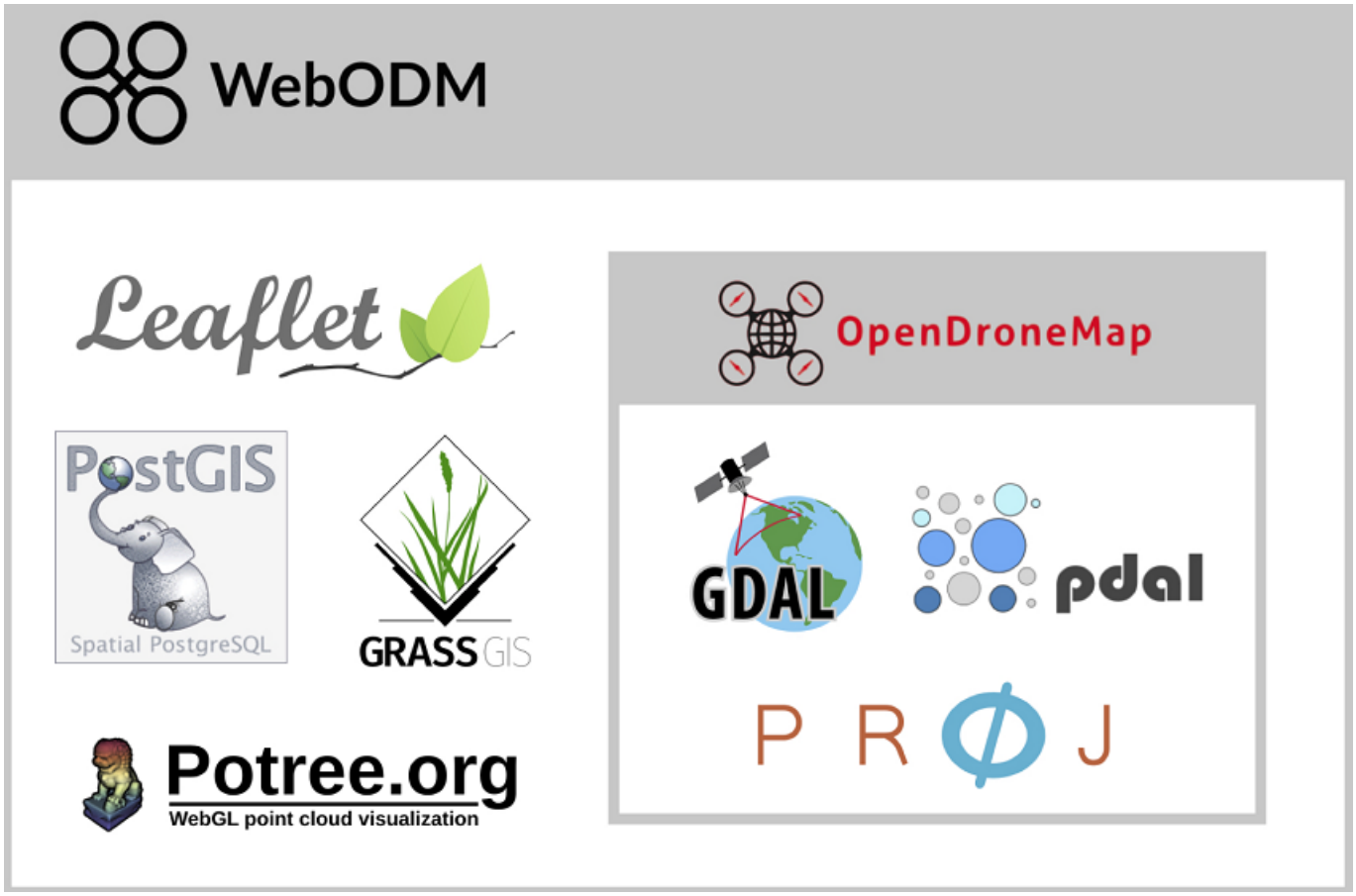


Figure 2: WebODM and OpenDroneMap projects building on other open source geospatial software (figure includes only selection of software most relevant for this topic; number of other components play an important role too). Source: authors.

OSI licenses also allow domain specific software platforms and applications to add geospatial capabilities. For example, GDAL is used in the BlenderGIS plugin of 3D rendering

engine Blender as well as in the LANDIS-II landscape change model to support geospatial data formats. These examples demonstrate how open formats and translation libraries facilitate interoperability across diverse projects and support development of innovative applications and tools.

References

Barnhart, K. R., Hutton, E. W. H., Tucker, G. E., Gasparini, N. M., Istanbulluoglu, E., Hobley, D. E. J., Lyons, N. J., Mouchene, M., Nudurupati, S. S., Adams, J. M., and Bandaragoda, C. (2020). Short communication: Landlab v2.0: a software package for Earth surface dynamics, Earth Surface. Dynamics, 8, 379–397,

Coetzee, S., Ivanova, I., Mitasova, H. & Brovelli, M.A. (2020). Open Geospatial Software and Data: A Review of the Current State and A Perspective into the Future. ISPRS International Journal of Geo-Information, 9(2), 90.

Mocnik, F.B., Mobasheri, A. & Zipf, A. (2018). Open source data mining infrastructure for exploring and analysing OpenStreetMap. Open Geospatial Data, Software, and Standards 3(7).

Neteler, M., Bowman, M. H., Landa, M., & Metz, M. (2012). GRASS GIS: A multi-purpose open source GIS. Environmental Modelling & Software, 31, 124-130.

Open Source Geospatial Foundation (OSGeo). (2014). Project Graduation Checklist, Version 2.0. Accessed January 22, 2021.

Open Source Geospatial Foundation (OSGeo). (n.d.). About OSGeo, Accessed January 26, 2021.

Open Source Initiative (n.d.). OSI Approved Licenses. Retrieved January 26, 2021.

Petrasova, A., Harmon, B., Petras, V., Tabrizian, P., & Mitasova, H. (2018). Tangible Modeling with Open Source GIS. New York, NY: Springer International Publishing.

Petrasova, A., Petras, V., Van Berkel, D., Harmon, B. A., Mitasova, H., & Meentemeyer, R. K., (2016). Open Source Approach to Urban Growth Simulation. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLI-B7, 953-959.

Pirotti, F., Brovelli, M.A., Prestifilippo, G. Zamboni, G., Kilsedar, C.E.,Piragnolo, M. & Hogan, P. (2017). An open source virtual globe rendering engine for 3D applications: NASA World Wind. Open Geospatial Data, Software and Standards 2(4).

Stallman, R. M. (2002). Free software, free society: selected essays. Ed. by J. Gay. 1st. ed. Boston, Mass: Free Software Foundation. 220 pp.

Vitalis, S., Arroyo Ohori, K., & Stoter, J. (2020). CityJSON in QGIS: Development of an open-source plugin. Transactions in GIS, 24(5), 1147-1164.