TACKLING THE CHALLENGES OF SOFTWARE PROVISION

Peter Vogt [1*], Pierrick Rambaud [2]

[1] European Commission, Joint Research Centre (JRC), Ispra, Italy – Peter.Vogt@ec.europa.eu
[2] United Nations FAO, Rome, Italy – Pierrick.Rambaud@fao.org

Commission IV, WG IV/4

KEY WORDS: Software, packaging, open source, dissemination, application outreach.

ABSTRACT:

In this perspective, we investigate the less documented topic of software provision, aimed at bridging the community of software developers to the one of software end-users. We outline aspects of the circular flow of software development, starting from the source code, software packaging, the target platform, licensing, program documentation, and feedback. Next, we highlight challenges and opportunities of these aspects and how they contribute to the overall success and adoption of a software application. Finally, we exemplify and illustrate how these aspects were addressed with the provision of the software GWB on FAO's cloud computing platform SEPAL. The outlined reflections on software provision are of generic nature and, depending on a given software, may include many more, or different aspects. Yet, we hope that this perspective may trigger more interest and dedication to the topic of software provision and its integral function to promote and improve software development.

1. INTRODUCTION

For most end-users, the term 'software' is equivalent with executing a given application to obtain a desired result. Moreover, the highest importance is usually attributed to the software being free to use. Besides intuitive use, a key requirement for success and wider acceptance of a software application is easy access, which is often facilitated through open-source projects. While most end-users naturally only care about stability and functionality of the software, software developers often see their task completed once the application reaches a certain degree of maturity and its source code is made available. However, in addition to ease of use and targeted software development, a third component in the life cycle of software design (Vogt, 2019) is the software provision. The importance of adequate software dissemination entails a wide range of aspects, which are often undervalued but are crucial to best meet end-user expectations and to achieve the highest application acceptance.

In this manuscript, we outline a perspective on approaches to appropriately address issues of software provision aimed at promoting software in an efficient way. We illustrate the motivation and features of various aspects of software provision on the recently published software GWB ([1]) (Vogt et al., 2022) and its implementation on the FAO cloud computing platform SEPAL ([2]), the System for Earth observations, data access, processing & analysis for land monitoring.

2. SOFTWARE PROVISION

This section summarises reflections on various aspects when disseminating a software application for a given operating system (OS). Figure 1 illustrates six individual sub-sections, which address generic, major aspects only. These aspects are not meant to be exhaustive because any given application can have a different focus area, which in turn requires dedicating more time and effort into respective aspects of software provision. In general, the overarching goal of software provision is to optimise the usage of the software application and to maximise its outreach into the user community.

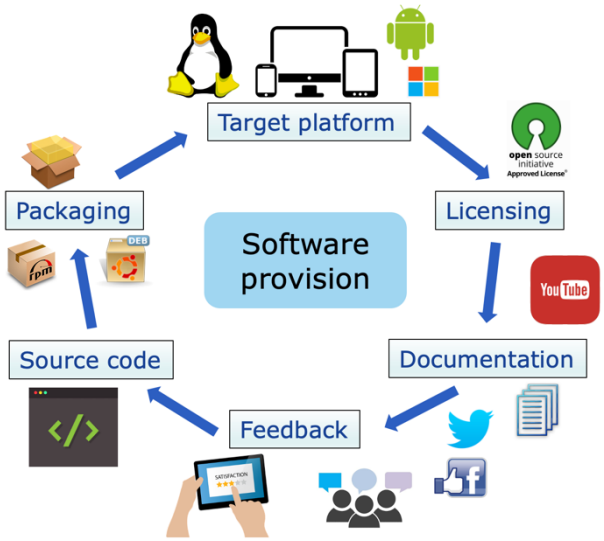

Figure 1. Typical aspects of software provision.

2.1 Source code

The provision of the source code is often perceived as a final product delivery. The main reason for this perception is that a fully transparent access to the source code will allow any interested user to analyse all processing steps, or to modify, or re-use sections of the code for any related purpose. Open-source access is also important from an educational perspective because software developers can use the source code to improve their programming skills or learn how to code in a different

* Corresponding author

programming language. Furthermore, full access to the source code implies that many experts can review and test the code, a critical aspect for security and building trust in the software application. On the other hand, the majority of end-users cannot make any use of the source code itself because they either do not understand the programming language used, or even if so, they may well be overextended with the complexity of a given source code. In addition, end-users may not have the skills to compile the source code or know how to properly link required dependencies. The large number of Linux distributions provide an additional challenge due to varying inter-dependencies of distribution-specific compiled libraries and packaging policies. In most cases, the process of compiling source code is quite different across various operating systems and often requires choosing and setting up an appropriate and dedicated OS-specific compilation environment. In brief, packaging - the conversion of compiled source code into a functional executable binary - is a science on its own and is beyond the abilities of a typical end-user.

2.2 Software packaging

The scope of packaging is to bundle the entire application into a single installation archive, including and/or linking OS-specific dependencies, pre- and post-installation instructions, and the integration into the OS via menu entries. Examples are rpm and deb-packages (Linux), dmg-packages (macOS), and exe/msi-packages (MS-Windows). Packaging allows for efficient system-wide software management: installation, upgrades, and removal of the application. System-wide installation also provides application access to all OS users in a coherent way.

A particular advantage of the Linux OS is the availability of, and easy access to, various compiler suits, programming languages, and libraries. Virtually all Linux distributions provide a package manager, which can be used to query, install, and uninstall any software or libraries of interest for direct use or temporary evaluation. The Linux package manager is a unique and very powerful tool: browse the repository and select all packages of interest and once done, click the Apply button. The same is true for updating the entire system. A simple mouse click, or the respective installation command, is all it needs to automatically download all selected software packages including any depending libraries. And all packages to be installed or upgraded are tested, validated, and officially approved. In fact, it is quite puzzling to see that no other OS has adopted such a user-friendly, efficient, and secure software management tool. Yet, a disadvantage of software packaging on Linux is the vast number of Linux distributions. In addition to various packaging formats, the high variety of custom compiled libraries and distribution-specific dependencies makes packaging under Linux a challenging task. One solution, though time-consuming and tedious, is to provide distribution-specific packages for the most common Linux distributions. Another alternative is to generate statically linked executables or use generic installation formats such as Snap ([3]), AppImage ([4]) or Flatpak ([5]), each having its own advantage or disadvantage.

In contrast to Linux, setting up a specific compiler environment and required libraries on any other OS can be a time-consuming and rather daunting exercise. It requires browsing the Internet for information, comparing various options against each other, manual download and subsequent execution of individual installer packages, and often custom actions to setup required elements in the packaging chain. Yet, once established, the resulting compiled binary is likely to work without issues across various editions of the MS-Windows or macOS operating systems. Finally, OS-agnostic compiling is facilitated when the application is coded in platform independent, mostly scripting languages such as JAVA ([6]), R ([7]), or Python ([8]). The TIOBE ([9]) programming community index is a measure of popularity of various programming languages. This index is updated monthly and may help to select a future-proof programming language when starting to build a new software system. Figure 2 shows a snapshot of the TIOBE index, illustrating the strong increase for the Python programming language (light blue colour) over the past 20 years, which in April 2022 has become the most popular programming language.

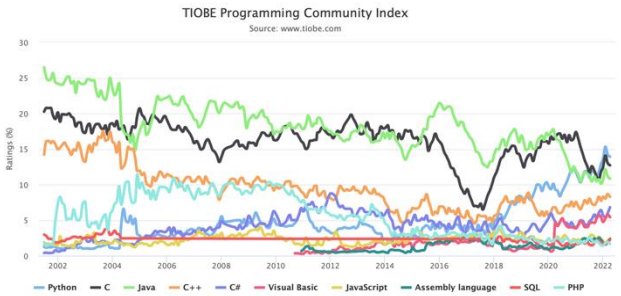

Figure 2. TIOBE ([9]) index showing the most used programming languages since June 2001.

Various tools exist for packaging the binary and adding package maintenance utilities. They differ in program features, ease of use, and price, from fully fledged proprietary commercial applications to powerful and free open-source solutions, for example the Nullsoft Scriptable Install System – NSIS ([10]) for MS-Windows. The Wikipedia website ([11]) provides an overview of notable software package management systems for a series of OS and packaging scopes.

In general, the installation of software into an operating system requires administrator rights, which are not available on many secured or closed IT environments, such as in government agencies, where users may fully access a limited OS-space only, for example their $HOME directory. Yet, this situation can be addressed by setting up the software and all required components in a self-contained single directory, which is then compressed into a self-extracting installer. Any user can then download such a standalone installer, extract it in a private location with write and execute permissions, and have full access to the application without administrator rights. An additional benefit of this way of software distribution is that the application in a self-contained directory can be copied to and executed from any external device, for example a USB flash drive, providing maximum portability of the given application.

Finally, the provision of a single application, or even a suite of applications up to a fully customised operating system, can also be setup within a Docker ([12]) container, or in a virtual machine setup, see the overview in this Wikipedia website ([13]). Nowadays, virtualization software is an ideal way for developers to draft, test, deploy, and run applications in a controlled and isolated environment with all its dependencies, and configured for a variety of operating systems without the need to purchase any additional hardware.

2.3 Target platform

Maximum outreach is achieved through a software setup that will work on as many platforms and operating systems as possible. In addition to personal computing, an even wider outreach may be achieved by using a cloud computing platform, such as GEE ([14]), the Google Earth Engine (Gorelick et al.,

2017) or BDAP ([15]), the Big Data Analytics Platform (Soille et al., 2018). Yet another alternative is designing the application as a web-based online application or, where possible, as an Android/iOS application for portable devices. According to Statcounter ([16], Figure 3), the use of portable devices has increased steadily over the past years and its global market share has surpassed the desktop market share in 2017. This trend is likely to continue, due to the growing adoption of smartphones (Almunawar et al., 2018) and the CPU performance increase in mobile devices (Halpern et al., 2016). Another deciding factor for choosing the appropriate target platform may also be the age class of the envisioned end-user audience (Bröhl et al., 2018).

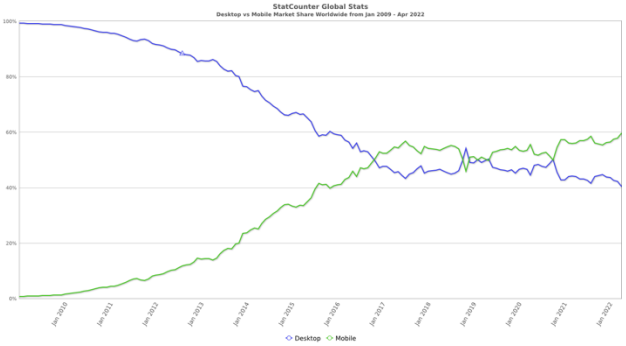

Figure 3. Global market share ([16]): desktop (blue line) and mobile devices (green line) from Jan 2009 to April 2022.

2.4 Licensing

A software license agreement is a legally binding agreement between the software creator and the software end-user. There are two main types of software license agreements. A proprietary or closed source license, usually outlined in an End User License Agreement (EULA), must be applied when using a proprietary programming language or the developer wishes to maintain full ownership and exclusive control over future developments. The other main type is Open-Source licenses. Open-Source Software (OSS) is a widely used term describing software that comes with all necessary permissions granted in advance to permit its use, improvement, and redistribution (modified or unmodified) by anyone and for any purpose. These permissions are often called the four freedoms ([17]) of software users. The ability of individuals and organisations to freely reuse software under standardised open-source licences has many benefits: it avoids the need for custom license negotiations, enables business innovation through rapid deployment of existing technology, increases sharing and popularity of software development methodologies, and supports sharing of development costs, resulting in improved code quality. The Open-Source Initiative (OSI) website ([18]) provides extensive information on over 100 open-source licenses and answers to related questions. The choice of an appropriate license scheme may be further complicated by having to review license compliance checks when a software project combines various modules or libraries, each with its individual license scheme. Here, the OSS Review Toolkit ([19]) can assist with license compliance checks. An alternative introduction and guideline, outlines the European Union Public License (EUPL) and other common OSS licenses (Schmitz, P. 2021). The Joinup Licensing Assistant ([20]) is yet another very helpful online tool to interactively compare and select an adequate open-source licensing scheme, while maintaining license compatibility. Finally, OSOR ([21]), the Open-Source Observatory, is a place where the open-source software community can publish news, find relevant open-source software solutions and read about the use of free and open source in public administrations across and beyond Europe in 24 languages. In summary, choosing the appropriate license for a given software project requires a good deal of thought and rationale to ensure all legal aspects are addressed in an adequate way, and covered also for future software developments.

2.5 Documentation

Documentation is crucial for software adoption. The most obvious scope is explaining application features and provide detailed usage instructions. Yet, documentation has many facets and can be fine-tuned in various ways to best match customer expectations, for example as a user manual, product sheets with application examples, guided instructions in workshop material, providing and distributing flyers, offering online training courses, or by using social media services, for example promoting the software via Twitter or recorded YouTube video sessions.

From a strictly technical perspective, any application is fully functional by executing correct command-line instructions. The addition of a Graphical User Interface (GUI) does not provide any new program functionality but will simply act as a user-friendly way to collect the information for setting up the underlaying command-line instructions for execution. By providing only valid options to choose from, a GUI has the additional benefit to reduce potential wrong user input. The scope of a GUI is to increase program usability, and, in this sense, a GUI can be seen as a form of program documentation aimed at simplifying program interaction and execution. While GUI coding is an extra burden for software developers, adequate GUI design is one of the most important aspects for the success and adoption of any application. A GUI should be intuitive, easy to interact with, and logically structured to flatten the learning curve, encourage, facilitate, and stimulate using the application to its full extent. In fact, the success of personal computing was triggered through the introduction of a mouse-driven GUI for Apple Macintosh in 1984 and MS-Windows 3.0 in 1990. Equally, Linux desktop environments such as KDE, Gnome, XFCE, etc. are nothing else but GUIs to the underlaying operating system. Another example, illustrating the importance and impact of GUIs, are geographic information system (GIS) applications such as ArcGIS ([22]), QGIS ([23]), GRASS ([24]), gvSIG ([25]), or simplifying coding interfaces such as RStudio ([26]) and JupyterLab ([27]), which greatly increased the wider adoption of image analysis techniques and the exploitation of spatial data. Furthermore, please note that nowadays virtually all portable or mobile devices use GUI-driven applications.

2.6 Feedback

End-users interact with the software application. This statement is not trivial at all, but in fact crucial because it entails many possibilities to enhance software adoption and improve software outreach into the user community. In fact, feedback is likely the most fundamental aspect in software provision because it concerns and serves as an indicator for all other aspects in the entire software cycle (see Figure 1), from product development, dissemination, and usage. It is therefore critical that software developers should be actively involved in tackling end-user feedback (Srba et al., 2016) to avoid misunderstandings and ensure efficient target-driven software development.

Ideally, the application should be accompanied with various feedback options to allow the end-user sending back constructive comments and suggestions. A good example is the GDAL project ([28]), which uses a mailing list, GitHub, chat, social media, and conferences (FOSS4G 2022) to exchange information between developers and the user community. Stack Overflow ([29]) is another prominent example serving the same purpose. Typically, feedback will result in improved documentation, bug fixing, or adding new program features. However, it may also trigger setting up various versions of the application, to specifically address targeted end-user groups. For example, the application, or modular parts of it, can be provided as a simple script (R, Python, etc), GIS-plugins, standalone-, mobile-, or webserver-based application. Feedback promotes transparency, security, and trust by actively involving the user community in the software life cycle. Feedback opens an efficient communication channel between all participants. It highlights areas of improvement, streamlines efficient development, and fosters end-user engagement. Ultimately, feedback contributes to increased application performance and acceptance of the software. Closed source projects often have a dedicated marketing department, which monitors, measures, and channels user feedback in dedicated customer satisfaction studies (Farris et al., 2010), considered essential for ensuring product success and end-user loyalty. In open-source projects ([30]), feedback is a critical component for efficiently facilitating code revisions, allowing for security screening, and even attracting new contributors from developers to translators to educators. Feedback is a natural way of communication, starting from sharing an idea to finding participants having the same interests and setting up a new project for implementation. Feedback then continuous as a stimulating collaborative tool for monitoring, listening, and constantly improving the software.

3. APPLICATION EXAMPLE

The GuidosToolbox Workbench (GWB) (Vogt et al., 2022) provides various generic image analysis modules as command line scripts on 64-bit Linux systems. In this section, we use GWB as an example to illustrate how we addressed the software provision points mentioned before.

3.1 Source code

In addition to the distribution-independent compiled executables, we provide the plain text source code for all IDL modules in a dedicated subdirectory of the application. The source code and dependencies of the IDL and all other C-programs, including the very popular spatial pattern analysis routine MSPA (Soille and Vogt, 2009; Soille and Vogt, 2022), has been placed on the public GitHub project page ([31]). We believe that the inclusion of the simple plain text IDL source code within the application facilitates the access to the source code, for example in the absence of internet access. All modules are launched via customised Bash scripts and setup in IDL ([32]), the Interactive Data Language. IDL is an interpreter programming language ([33]), meaning that every source statement will be executed line by line. While compiled object code will run faster, interpreter language will be easier to understand for the novice user and facilitate modification, debugging or extracting code sequences of interest. All GWB modules include a program version checker routine, which will test for, and if available, inform the user about a new GWB version. For the user, using the most recent program version is important to benefit from up-to-date program features and latest bug fixes. For the developer, program feedback and bug reports are most meaningful when reported for the current program version only.

3.2 Software packaging

When coding GWB, Bash and IDL were chosen due to personal coding preference. With respect to packaging, IDL provides an additional advantage because it provides its own set of highly efficient OS-specific processing libraries. This means that all scripts and required IDL libraries can be stored in a single application directory, which then works on all Linux distributions. Combined with customised packaging setup-files, this archive is then converted into distribution-specific installation packages in rpm- and deb-format for the most common Linux distributions. In addition, we provide a generic standalone installer using the open source makeself ([34]) archiving tool. The standalone installer can be used on any Linux distribution for either, system-wide installation, or installation in user space on restricted systems without administrator privileges, for example under $HOME.
All installer packages include two sample images and module-specific usage descriptions, aimed at generating sample output illustrating the features of each application module.

3.3 Target platform

With its focus as a server-application, GWB is setup for the Linux OS, but it can also be used on a regular desktop PC running Linux. The installation on cloud computing platforms, including an interface to upload/download personal data, greatly enlarges the outreach into the user community and allows usage of the software from any device having a Web browser and Internet access. First, GWB was installed, and can easily be maintained, via the respective deb-package on FAO's cloud computing platform SEPAL. To further facilitate end-user interaction, we developed a Jupyter (Kluyver et al., 2016) based graphical user interface for GWB on SEPAL. This GUI uses widgets and interactive displays to help the end-user provide personal data to the individual software routines. The interface is developed in Python using the sepal-ui framework ([35], Rambaud et al., 2022), embedding a fully independent set of requirements. Because the GUI application is run via the voilà dashboarding tool (QuantStack, 2019), the end-user is never confronted with the underlaying command-line interface. However, the command line access to individual GWB image analysis modules is always available for the interested user for direct use or for inclusion into custom scripts setup by the SEPAL user.

3.4 Licensing

GWB consists of C and IDL source code, which are subject to various license conditions. A license compatibility check resulted in choosing GPLv3 ([36]) as the general applicable license for GWB. This choice was triggered due to using one GSL, Gnu Scientific Library ([37]) subroutine within a C-code module. GSL is released under GPLv3, which is known as a strong copyleft license, Copyleft is an arrangement, where software may be used, modified, and distributed freely but under the condition that anything derived from it is bound by the same GPLv3 conditions. As a result, this requirement enforced that all other licence conditions had to be merged under GPLv3. The respective licensing information was then included in the software and its GitHub repository.

3.5 Documentation

The GWB project homepage, as well as the GWB application itself, provides a brief overview and installation instructions. To foster understanding and exemplify proper usage we provide highly detailed usage instructions on a dedicated SEPAL documentation page for the GWB command-line tools ([38]). The instructions on this page outline the use and configuration of features of each module with the help of the application-provided sample images. In addition, and to further facilitate using the application especially for novice users and non-IT experts, we wrote a second documentation page ([39]) illustrating the use of the individual GWB modules via the interactive Jupyter ([40]) dashboard.

3.6 Feedback

In fact, GWB itself is a prime example of user feedback. GWB is a spin-off of the desktop application GTB ([41], Vogt and Riitters, 2017). GTB-users suggested to extract the most popular GTB image analysis modules into individual command-line driven modules. This suite of modules then turned into the new application GWB, allowing GUI-free execution of dedicated GTB images analysis routines on webservers, or executing a GWB module from custom scripts setup by the user in any other programming language. Initial user feedback via email to the GWB lead developer then triggered setting up the much-detailed command-line documentation page on SEPAL. Further feedback from SEPAL users then lead to developing the Jupyter GUI application on SEPAL, which with its own dedicated documentation page, further simplifies interacting with the GWB image analysis modules. Both SEPAL documentation pages are setup on GitHub for efficient inclusion of end-user feedback and updating its content for new features in future versions of GWB. With GWB being a spin-off of GTB, GWB users may also consult the extensive documentation in form of thematic product sheets and workshop material available on the GTB homepage. Consulting these documents will enhance understanding of the existing modules and their application fields, which in turn may prompt suggestions for improvements or new functionality to be included in a future GWB version.

4. CONCLUSIONS

Software provision is an often overlooked yet critical component in software design. It comprises various aspects, which when addressed appropriately, can make a great impact in the promotion, outreach, and acceptance of a software application. Obviously, the number, type, and content of software provision aspects will be directly related to the size and complexity of a given software project. For example, very simple software may only require downloading a Bash-script and running it in a terminal. However, the need for tailored concepts and dedicated programming efforts in software provision will become paramount when dealing with large and complex software projects. In fact, at enterprise and commercial level, aspects of software provision become the driving principles in development, efficient dissemination, quality management, progress monitoring, and inclusion of end-user feedback to increase customer satisfaction and maximise the overall success of a software project. Software provision may even expand into areas that are no longer strictly related to the software itself. Efficient software provision efforts contribute to disseminating knowledge and analytical methods beyond the traditional user community. For example, an intuitive and user-friendly application for digital image analysis, which originally was targeted for the GIS user community, may then become an interesting, new analysis tool in medical image analysis: analysing the shape of forest patches or blood cells in a digital image is technically the same task but its implications for the respective user-community are rather different, yet they could both use the same software doing the same type of analysis. Another important aspect of software provision is the fact that it acts as a bridge between software developers and end-users, and as such contributes to a closer interaction for the benefit of both communities. Finally, lessons learned in software provision will also be beneficial and often directly applicable to future or other related software projects. We hope that this perspective has shed some light on the topic of software provision and that it will stimulate further discussions to improve communication and efficient dissemination of software throughout various end-user communities.

REFERENCES

Almunawar, M. N., Anshari, M., Susanto, H., & Chen, C. K., 2018. How People Choose and Use Their Smartphones. In P. Ordóñez de Pablos (Ed.), Management Strategies and Technology Fluidity in the Asian Business Sector (pp. 235-252). IGI Global. doi.org/10.4018/978-1-5225-4056-4.ch014.

Bröhl, C., Rasche, P., Jablonski, J., Theis, S., Wille, M., Mertens, A., 2018. Desktop PC, Tablet PC, or Smartphone? An Analysis of Use Preferences in Daily Activities for Different Technology Generations of a Worldwide Sample. In: Zhou, J., Salvendy, G. (eds) Human Aspects of IT for the Aged Population. Acceptance, Communication and Participation. ITAP 2018. Lecture Notes in Computer Science, vol 10926. Springer, Cham. doi.org/10.1007/978-3-319-92034-4_1.

Farris, P. W., Bendle, N. T., Pfeifer, P. E., Reibstein, D. J., 2010. Marketing Metrics: The Definitive Guide to Measuring Marketing Performance. Upper Saddle River, New Jersey: Pearson Education, Inc. ISBN 0-13-705829-2.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., Moore, R., 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, 202, pp 18-27, doi.org/10.1016/j.rse.2017.06.031.

Halpern, M., Zhu Y., Reddi, V. J. 2016. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. IEEE International Symposium on High Performance Computer Architecture (HPCA), doi.org/10.1109/HPCA.2016.7446054.

Kluyver, T., Ragan-Kelley, B., Fernando Pérez, Granger, B., Bussonnier, M., Frederic, J., Willing, C., 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90)

QuantStack 2019. And voilà! Jupyter Blog. https://blog.jupyter.org/and-voil%C3%A0-f6a2c08a4a93

Rambaud, P., Guerrero, D., d'Annunzio R., 2022. sepal-ui: a framework to transform Python based GIS workflows into

stand-alone web applications. Zenodo, doi.org/10.5281/zenodo.6467835.

Schmitz, P., 2021. European Union Public Licence (EUPL): guidelines July 2021, European Commission, Directorate-General for Informatics, Publications Office, doi.org/10.2799/77160.

Soille, P., Burger, A., De Marchi, D., Kempeneers, P., Rodriguez, D., Syrris, V., Vasilev, V., 2018. A versatile data-intensive computing platform for information retrieval from big geospatial data. *Future Generation Computer Systems*. 81: 30–40, doi.org/10.1016/j.future.2017.11.007.

Soille, P., Vogt, P., 2009. Morphological segmentation of binary patterns. *Pattern Recognition Letters*, 30(4), 456-459.
Soille, P., Vogt, P., 2022. Morphological spatial pattern analysis: open source release. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, this volume.

Srba, I., Bielikova, M., 2016. A Comprehensive Survey and Classification of Approaches for Community Question Answering. ACM Trans. Web 10, 3, Article 18 (August 2016), 63 pages. doi.org/10.1145/2934687.

Vogt, P., Riitters K., 2017. GuidosToolbox: universal digital image object analysis. *European Journal of Remote Sensing*, 50, 1, pp. 352-361, doi.org/10.1080/22797254.2017.1330650.

Vogt, P., 2019. Patterns in software design. *Landscape Ecology*, doi.org/10.1007/s10980-019-00797-9.

Vogt, P., Riitters, K., Rambaud, P., d'Annunzio, R., Lindquist, E., Pekkarinen, A., 2022. GuidosToolbox Workbench: spatial analysis of raster maps for ecological applications. *Ecography*, doi.org/10.1111/ecog.05864.

[1] https://forest.jrc.ec.europa.eu/en/activities/lpa/gwb/
[2] https://sepal.io/
[3] https://snapcraft.io/
[4] https://appimage.org/
[5] https://flatpak.org/
[6] https://www.java.com/
[7] https://www.r-project.org/
[8] https://www.python.org/
[9] https://www.tiobe.com/tiobe-index
[10] https://nsis.sourceforge.io/
[11]https://en.wikipedia.org/wiki/List_of_software_package_management_systems
[12] https://www.docker.com/
[13] https://en.wikipedia.org/wiki/Virtual_machine
[14] https://earthengine.google.com/
[15] https://jeodpp.jrc.ec.europa.eu/bdap
[16] https://gs.statcounter.com
[17] https://www.gnu.org/philosophy/free-sw.en.html
[18] https://opensource.org/licenses
[19] https://github.com/oss-review-toolkit/ort
[20]https://joinup.ec.europa.eu/collection/eupl/solution/joinup-licensing-assistant/jla-find-and-compare-software-licenses
[21] https://joinup.ec.europa.eu/collection/open-source-observatory-osor/about
[22] https://www.esri.com/arcgis
[23] https://qgis.org/
[24] https://grass.osgeo.org/
[25] http://www.gvsig.com/
[26] https://www.rstudio.com/
[27]https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html
[28] https://gdal.org/community/index.html
[29] https://stackoverflow.com
[30] https://en.wikipedia.org/wiki/Open-source_software
[31] https://github.com/ec-jrc/GWB
[32] https://www.l3harrisgeospatial.com/Software-Technology/IDL
[33] https://en.wikipedia.org/wiki/Interpreter_(computing)
[34] https://makeself.io/
[35] https://pypi.org/project/sepal-ui/
[36] https://www.gnu.org/licenses/gpl-3.0.en.html
[37] https://www.gnu.org/software/gsl/
[38] https://docs.sepal.io/en/latest/cli/gwb.html
[39] https://docs.sepal.io/en/latest/modules/dwn/gwb.html
[40] https://jupyter.org
[41] https://forest.jrc.ec.europa.eu/en/activities/lpa/gtb/