Proc. of International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA 2024)
1-2 February 2024, Victoria-Seychelles

On Openness in Service Stacks

Peter Baumann
Department of Computer Science
Constructor University
Bremen, Germany
pbaumann@constructor.university

Abstract – Openness has become a widely requested property, mostly of data and software. Generally, being "open" is seen as advantageous; sometimes a demand is stated that all software and data should be free, in other words: non-open (or less open) data and software should not even exist. On the other hand, neither all data, nor all software is openly available today. Vendors and research centers often regulate data access, and software users still oftentimes decide to invest into licenses of proprietary software – obviously, with a reasons for doing so.

In this contribution, based on a strong own involvement in and experience with the open-source community, we provide a balanced analysis of open data and software and additionally consider the role of open standards. Ultimately, it turns out that open data and software models constitute particular business models with particular properties to be considered before deciding about a specific model. Further, we motivate that there is a variety of possible business models which all can coexist in the market. As such, this paper can serve for clarification in an often heated, sometimes emotional discussion, contributing to more fact-based choices.

Keywords — open data, open-source software, open services, open standards

I. INTRODUCTION

The movement for Open Science has been around for two decades and is supported by learned societies, research organisations, funding agencies, and others. Openness has become a widely requested property, mostly of data and software. Generally, being "open" is seen as advantageous; in places even a demand is stated that all software and data should be free, in other words: non-open (or less open) data and software should not even exist. However, even within the supporter communities, "Openness may sound self-evident, but in fact it can mean different things, even within the Open Science community" [1]. A typical understanding of open knowledge is that the provider of knowledge should offer it for uninhibited access [2].

Conversely, after decades of active promotion of openness into society and even political communities, in practice neither all data nor all software is openly available. Vendors and research centers often regulate data access, and users still decide to invest into licenses of proprietary software. There seem to be valid reasons for restricting openness.

Which is debated heavily. While balanced discussions exist [3], oftentimes they are conducted very emotionally: proponents of unlimited openness diagnose "digital feudalism" [4] while impositions like "all software should be open source" by others are called "software communism". Also the "right" way of doing open source is being debated heatedly: "Open source is an amoral, depoliticized substitute for the free-software movement" and "it's much easier to be a supporter of open source, because it doesn't commit you to anything" [5]. Interestingly, free and open provisioning typically is tied to intangible products. Free and open VLSI chips or other engineering artifacts are not nearly addressed as intensively as are software engineering artifacts.

In this contribution, we discuss openness of data, software, standards, and services, thereby assessing the impact of each facet on Open Science. Goal is to collect a solid factual basis in a discussion which oftentimes is dominated by emotional arguments. This is done based on the author's own strong involved involvement in both proprietary and free and open-source software (FOSS) development since decades, in academic and industrial setups, and as Open-Source Geospatial Foundation (OSGeo) Charter Member.

We attempt to clarify what kind of openness is most critical for a satisfying user experience, where "user" can mean someone accessing some online service, but also operators of such a service in the sense that they utilize some (open or closed) software and data. We will want to make a point for thinking in terms of interfaces rather than implementations, which leads to the adherence to standards as a critical factor not only for interoperability, but also vendor independence. As such, for decision makers this should make it easier to define requirements, separate concerns, and ultimately make more fact-based decisions.

The remainder of this paper is organized as follows. In Section 2 we inspect "dimensions" of openness relevant in software world. In Section 3 we discuss the factual contribution of each such dimension towards the stated goals. Section 4 concludes the plot.

979-8-3503-9452-8/24/$31.00

II. Dimensions of Openness

The business models under which the technologies investigated are made available vary widely – from proprietary over dual-license to open-source. Therefore, and because we sometimes observe confusion about the meaning and consequence of an "open-something", we briefly discuss these core terms heavily debated in the science data domain. Note that the goal is not to define or even explain in all detail – this has been done many times elsewhere already – but to specifically talk about the terms open *data*, open *source*, open *services*, and open *standards*.

A related term is "free" meaning that access or use is free of cost; “open” tentatively differentiates itself against that: Open items not necessarily are free of cost, and free-of-cost access is no guarantee for openness [6].

A. Open Data

Open data is the idea that data should be freely available for everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control such as constraining access to certain user groups. Like the concepts below, we will discuss in detail in the next section.

B. Open Source

The Open Source Initiative (OSI) lists over 110 different license types [7] and an annotated definition [8]. This term refers to the software used, e.g., to serve or access data (i.e., servers and clients in Web-based information systems). Programs typically are written in some high-level language. For each language it needs compilers or interpreters translating this source code into machine code which can be executed by a particular CPU. Note that for one and the same language often different compilers exist, not to speak of the manifold options of a compiler impacting the code generated – we will need this fact later. In practice, interpreted code like python is easier to handle, maybe one reason for its recent embracing by data scientists (so not trained computer scientists). Compiled like C++ may still comprise substantial hurdles which typically requires substantial technical skills.

Furthermore, even open source code runs in the particular "environment" of some computer hosting the service. As such, the program will use external libraries whose source code may or may not be open, and it has been derived from the source code through a compiler which itself may or may not be open. Dependencies are manifold, essentially reaching down to the hardware and networks.

C. Open Service

Whether it is data access or processing, always services are involved. Even in the most trivial data download of files the server (sic!) must provide APIs like *ftp* or *scp* to accomplish the download. This gets us to a hitherto unaddressed aspect, that of open services, which actually can mean two independent things:

- All functionality the API advertises is available to clients without restrictions on region, user group, etc. – very much like with open source.
- The service API is published so that own clients can be built, rather than shrink-wrapped client / server applications. This addresses interoperability.

We concentrate on the second option because it is closely related to the open standards: if the service API follows open standards then third parties can build clients which, in turn, may connect to further services offering the same API.

Stating the obvious, open service does not mean open access to the complete software installation. Typically service operators will – for good reason – not publish details about their hardware/software stack. For example, it is not usual (and of no particular interest to most users) what database system a service uses underneath. We come back to this in the next section.

D. Open Standards

In IT world, standards typically establish data formats and interfaces between software components so that software artifacts written by different, independent producers (say, different companies or different departments within a company) still can communicate and perform a given task jointly, like building complex systems. Building software based on only interface knowledge and without knowledge about the internals of how a component establishes the behavior described by the interface definition is a key practice in Software Engineering; without such boxed and layered thinking the complexity of today's software would be absolutely intractable and unmanageable.

Like with data, a standard is called open if it is accessible to everybody without discriminating; some of those standards additionally are free of cost (such as with the Open Geospatial Consortium, OGC) while others are available against a (usually moderate) fee, such as with ISO.

Some standardization bodies offer compliance suites allowing validation of implementations against an official test suite, such as the OGC compliance test suite [9].

III. Assessment

In this section we argue that the categories data, code, service, and standards are independent from each other, trying to provide a differentiated picture.

A. Open Data

As our society at large today depends so much on data it seems natural to make data available for public use where this public needs it (which, like with software, does not necessarily mean free of cost – for example, the European Union funds acquisition of satellite imagery bought from the satellite operators, and then made available for free). In academia, a free flow of data and knowledge has been a key concept in science since long, fostering peer review, reproducibility, societal impact, and collaborative research [10].

Still, there are valid reasons for withholding data or limiting their reuse. One example is the thesis work done by a PhD researcher. Would all the data acquired with often substantial effort be forced to get published immediately it could happen that some other researcher took the data, published insights themselves faster (because of more experience or better evaluation resources like a larger team or computational power), and thereby kill the PhD topic for the student. Therefore, a limited "embargo time" is common. Further reasons include: military intelligence is not to be shared publicly; data obtained through some effort by a company should be allowed to generate revenues; privacy protection reasons, etc. In this context a session "The Limits of Open" was held at the Annual European Geosciences Union General Assembly in 2021 [11]. Remarkable is also the Aarhus Convention [12], an international legal framework on access to environmental information. In practice, our experience is that by giving data providers the freedom to design access control policies they are more inclined to operate basically open services.

An indirect obstacle to open access – aside and independently from organizational restrictions and holding back – can be the difficulty of access due to reasons such as uncommon data formats, unwieldy data granules (such as 100 GB TIFF files which cannot be loaded into a desktop RAM), access interfaces requiring high technical backgrounds, or interfaces posing particular hardware requirements (high client-side hardware resource needs, high-bandwidth connection, etc). Hence, offering open data also has an implication on the ease of use of the data offered. In this context, an interesting and widely embraced initiative has been launched by the USGS Landsat team coining the term Analysis Ready Data (ARD) [13], and much research is being investigated meantime, such as by CEOS [14] and OGC [15], among others. In this approach, data centers tentatively undertake high effort in preparing (homogenizing, restructuring, cleansing, etc.) data in a way that reduces such intrinsic obstacles to data access.

Bottom line, open data sometimes are highly desirable, but sometimes are not, for good reason. This more or less is common sense today.

B. Open Source

As the open source idea addresses software we discuss it in the following from a Software Engineering perspective, in particular the facets source code inspection, malicious code injection, determining correctness, protection, version control, software-only limits, legal liability.

Hardware vs software. It has been argued [6] that hardware is substantially different from software, and therefore selling of software licenses similar to hardware buys is not adequate. Looking closer we find several misconceptions (see in particular Table 2.1, quoting from there):

- "If hardware breaks or ceases to function beyond easy repair, it becomes useless. Software cannot break in the same sense." – ignoring the unspecified attribute "easy", there are obvious reasons to make software break, including incompatible changes in the execution environment, configuration changes for worse, bugs surfacing only once hitherto unused but contractually assured functionality is used, etc.
- "Hardware cannot be duplicated. Every copy needs the same amount of raw material and energy as any other. Copies of complex hardware will always be imperfect. Software can be duplicated completely. Each successful copy of a software product is an identical reproduction of the original (the "raw material" is the source code, it does not run out)." – Here we observe a mismatch in the focus: While on software side the availability of the source code is stressed, on the hardware side the particular piece is in focus whereas the corresponding level would be the blueprint for the hardware, such as VLSI designs, lithography masks, etc. This "raw material" – as the paper calls it – likewise "does not run out" as it is not distributed. Just as the source code is not (to be exact, the software product is not a reproduction, but a derivation from the source code).
- "Hardware can wear out, rust, or decay, and will eventually break and cease to function. Software does not wear down, rust, decay or break. It may fall out of use, but it never loses its basic functionality." – While some hardware may wear out indeed (in particular when containing moving parts, such as spinning disks) there is a large share in hardware which is outdated and replaced long before any functional wear-out could occur. From this perspective, both sides match again.

We feel that pulling up a special position of software in the general market of goods is based on (i) unsubstantiated technical arguments and (ii) a deeply rooted feeling of "what I cannot touch I do not want to pay for". Notably, in consequence this should affect all non-physical goods, such as legal contracts, for example.

While this might be perceived more as a philosophical discussion we next address "hardcore" software engineering aspects.

Source Code Inspection. It is a common observation in undergraduate Computer Science education that students have difficulties with the abstraction of an interface – which, typically, on introductory level means calls to functions provided by libraries. Recurringly students request to "show me the source code so that I can understand what the function does". Only gradually they learn that this is futile in several respects. First, this means going down a rabbit hole where the function whose understanding is required itself invokes other functions, and so on – ultimately the whole software stack has to be investigated until at operating system level the final frontier is reached where, again, functions have to be understood from their documentation alone. For a human brain such comprehensive understanding of the code obviously has natural limits and will not work for most of today's services at least (some clients admittedly

are simplistic enough). Second, in industrial practice there is simply not sufficient time to go through extensive source code resources before becoming productive – code production has to start based on the documentation (which is why it was written in the first place).

It is common wisdom that, for proper use of software components, not their source code but the pertaining documentation is essential (which in turn underlines the importance of high-quality documentation). We therefore argue that the same applies to open-source software: except for trivial code it is not feasible to dive into the source code.

Hence, even when inspected by experts, openness of the source code of the tool under consideration is not necessarily a guarantee for completely overseeing its effects.

For example, the list where PostgreSQL core development is discussed is "a must read to follow along for a few months before you start contributing yourself" [16]; actually, the article introducing on how to put fingers on the PostgreSQL source code lists several more steps, such as studying wikis and reviewing patches, before it is time for a first own patch. This in addition to deep pre-existing skills on database system implementation required to understand and, ultimately, contribute productively. As per OpenHub the 1.1 million lines of code of the PostgreSQL database server offering a signature functionality breadth have seen just teams of up to 21 at a time [17]. MariaDB with its 2.24 million lines of code is maintained over the last ten years by teams less than 40 [18]. The rasdaman datacube engine consists of about 600k lines of code [19], and similarly rigorous rules apply for code contributions to ensure quality. Consequently, we observe that patches by externals are usually contributed on the periphery of the engine where complexity is far less than in the core.

Software quality rules are laudable, but in practice drastically limit the circles who are willing and able to contribute. Paul Starkey observes "there's an argument that having the source available gives users guarantees they don't get from proprietary software, but with something as complicated as a database, most users aren't going to try to master the sources" [20]. In particular for data scientists (i.e., not computer scientists) it is generally not possible to verify the validity of open source code - and be it just for the lack of time to inspect all source code involved.

Hence, the fact that source code *can* be inspected does not necessarily mean it *will* be inspected and scrutinized. In practice, except for very shallow and simple projects, externals will normally not deeply look into it. The claim of assessing and fixing software "by everybody" is largely hypothetical. This puts users of complex open source tools de facto into a situation similar to proprietary software: upon problems, the hotline gets contacted.

Finally, the fact that some service engine, say PostgreSQL, is open source does not guarantee that the version deployed in a particular service is based on the unmodified, vanilla source code or is some forked version. We conclude that in practice ordinary users will not be able to assess the code some service operates, even if that is open-source.

Determining Correctness. Along a similar line, it is often argued that software bugs can be found better in open than in closed software. We have not found any quantitative evidence for this statement, it rather seems to be a widespread assumption (or based on comparisons of Unix and Microsoft Windows alone). Inspection of the source code, called *glass-box testing*, can indeed spot opportunities for performance improvement, for example, by determining that particular code pieces are executed very often and hence represent a worthwhile target for optimization. Additionally, code coverage analysis can spot problematic situations like code never executed ("dead code"), missing *else* branches in *if* statements, uninitialized variables, etc.

In contrast, *black box testing*, which does not know about the code, but solely tests the externally observable behavior based on the specification (which typically is or relates to the user documentation), can detect deviations from the expected behavior. This is what effectively a user is most interested in: does it perform as expected, or not? Assuming realistically complex systems, for a user it is close to impossible to dive into the code to find a bug. For example, assume a user suspects that some SQL statement delivers a wrong result. SAP R/3 already a long time ago had around 7,000,000 lines of code, 100,000 function calls, and 15,000 database tables. So what to look at instead, if not the source code? Ultimately, is defined through the standard, so that is what we look at below.

Protection. Assets of value to somebody can be distributed without restriction, can be held back, or anything in between (such as being released after some time, at certain conditions, etc.). This holds for data as well as software. Development of advanced software requires high skills (read: salaries) and constitutes significant Intellectual Property (IP). It has been stated [21] that IP rights provide small and medium sized companies (SMEs) with the opportunity to protect their technical innovations and the flexibility to optimize their business pursuits. SMEs in particular depend on Intellectual Property protection for their innovation against large, multi-national players. For example, rasdaman, marketed by an SME, has Google Earth as a key competitor.

We conclude: open-source is not a software quality criterion, but one business model family out of many possible where each model comes with its individual merits, and with a blurred border to closed-source – see, e.g., dual-licence models adopted by rasdaman. Therefore, it seems worthwhile to not adopt a radical all-or-nothing approach, but to accept that different situations beg different, individual rules. This view is supported by the observation that there is a spectrum of business models between the extremes of completely open and completely closed software.

Version Control and Its Limits. Most open-source projects today use professional-grade version management, such as git, alongside with further productivity and quality

enhancements. It goes by itself that reusing the code by others is invited – forking an open-source project is considered a signal of success for the project as others use the code and even invest into developing it further. However, from a product maintenance perspective key elements are missing. From the moment the derivative is forked it is an independent artifact under a different maintainer; hence, earlier pledges made, such as compatibility with a particular standard or support commitments, will not necessarily be inherited to the new package. Among others, this may break interoperability. For a provider running mission-critical services, therefore, a critical scrutiny of the pledges applicable to the particular fork is required.

Limits of Software-Only Scrutiny. A service can be compromised on hardware-level despite running open-source software. With the same rationale for inspection of the source code is required to check for proper, expected behavior, one also needs to inspect the compiler generating machine code from the source code and, in particular, the underlying hardware that executes the machine code. Bugs in machine instructions [22] as well as firmware attack vectors [23] are not unknown. Validation of hardware down to transistor level typically is a destructive process, so establishing trust in hardware requires prohibitive effort [24] and hence is infeasible in practice. Bottom line, trust in a service requires trust not only in software but also in hardware, and source code analysis can cover only a limited part of system verification. And we did not even address all the network components involved when accessing a service via Internet.

Legal liability. A recent development is the European Union proposal for a regulation on cybersecurity requirements for products with digital elements, known as the Cyber Resilience Act (CRA) [25]. Relevant for our discussion is primarily the objective to "ensure that manufacturers take security seriously throughout a product's life cycle" which, among others, results in liability of the software provider irrespective of open or closed source. This is in line with US DoD fears that FOSS "potentially creates a path for adversaries to introduce malicious code into DoD systems. This concern requires a careful supply chain risk management (SCRM) approach for OSS, which must meet the same rigorous standards for SCRM and cyber threat testing as any other product" [26].

This strive for liability has raised severe concerns by the FOSS community [27] who requested to "exempt" FOSS, i.e, have the CRA apply only when the code leaves the open source "commons" (a term not rigorously defined) and then continue to apply throughout the entire commercial supply chain.

The EU, on the contrary, argues that currently closed-source IT sellers have to assume full liability on their complete delivery, including all open-source components, which is infeasible (not only) for SMEs. Therefore, European SMEs should only have to pay to certify the code they are responsible for; and rely on others upstream (including FOSS providers) to certify the rest.

Notably, this discussion is not about feasibility of CRA (its draft having 40,000 words), which can be questioned indeed; rather, it is about whether different rules apply to open and closed source. The latter industry is used to carry the burden of legal responsibility, FOSS is not; we believe that, while the attitude of "no money – no responsibility" is understandable from a FOSS perspective, it is not balanced to leave the burden with IT industry only, and it is dangerous for society at large when left alone with software issues.

The 2023 Synopsis report [28] highlights importance of finding a solution: from the 1,703 FOSS codebases scanned across 17 industries, over 80% were found to have at least one vulnerability, about 50% to have high-risk vulnerabilities. The general public realizing that recently shows some anxiety [29].

A. Open Services

Building services typically is done by composing preexisting packages, libraries, etc. plus potentially adding glue code or functionality enhancements. Importantly, it is sufficient for developers to know the interface specifications ("on input of X the result invariably will be Y"). If this specification is an open standard, and if the tool has been confirmed to pass the corresponding compliance test, then the behavior of this tool can be anticipated with respect to this standard which creates trust in the tool.

Examples are manifold: we trust SQL query language implementations, regardless whether the database management system is open or closed source; we trust our C++ compilers, python engines, numerical libraries, operating systems, etc. – at least concerning the core question addressed here: does this code provide me with the true, valid data? And, for that matter, we trust the underlying hardware which ultimately executes the code.

B. Open Standards

Open standards invite to mix and match components at the discretion of both developers and service operators. To this end they need some evidence that a tool under consideration indeed fulfils expectations in their API conformance. Ideally, compliance of a tool with a particular standard can be proven automatically. Unfortunately, most standardization bodies provide only the specification documents and sometimes high-level advice on testing. One exception is the Open Geospatial Consortium (OGC) which provides high-level textual Abstract Test Suites (ATSs) and concrete Executable Test Suites (ETSs) which can be run against systems under test [9]. These tests – which of course are agnostic against open or closed source – perform a black-box testing on defined properties. If some tool has been confirmed to pass the corresponding compliance test, then the behavior of this tool to some extent can be trusted with respect to this standard (of course, there may be further unwanted behavior not addressed by the compliance test – for example, a test will typically concentrate on functionality but not on security, and also the mileage of comprehensiveness may vary).

Coming back to SQL. This is a standard supported by many widely used open as well as closed source systems. NoSQL databases consciously have left this safe harbor in favor of own query languages, such as MongoDB MQL, neo4j cypher, PigLatin, Hive and many more. If any of these NoSQL systems is picked then any migration to another system, SQL or NoSQL, requires a major rewrite and becomes immensely hard and costly.

We conclude: Open source tools, too, can create a complete "vendor" lock-in when not relying on community-adopted standards; proprietary SQL systems can retain freedom of choice and replacement, thanks to the standard (which is not even open in the strict sense, as ISO charges).

IV. A Business Model Perspective

Both open and proprietary tool providers need to address their customers, need to do marketing and claim competitive advantages. Based on the discussion in this contribution we believe that open source in its essence is not a software style, not a software quality assurance technique, but a business model. What is typically discussed are the extremes, completely proprietary and completely open.

However, in today's software ecosystem we find a wide spectrum of mixed business models: donation-based (ex: OpenStreetMap), industry sponsoring (ex: Apache), crowd-funding (ex: TopCoder), open-core (ex: GitHub), freemium (ex: Nintendo), software leasing (ex: Oracle, SAS), software as a service (ex: all major cloud providers), and several more. Often, companies build their own mix of open and closed offerings (including products and services), such as rasdaman, Red Hat, SuSE, etc. Additionally, we find proprietary and open components combined, such as running open-source clients on closed-source MS-Windows [6].

Open and closed source business models differentiate in various aspects, which can be relevant in the overall software selection process in addition to the software features as such, for example:

- Who has knowledge on the software to the depth required for doing not only cosmetic changes, but also general bug fixes and enhancements?
- Who will take on responsibility in case of malfunction?
- What kind of maintenance is provided?
- What is the provider's commitment to standards like?

Notably, timeline of support of a tool is not a differentiating criterion. Industry sometimes claims that software products live longer than open-source projects due to the company thrust behind the project. This is not necessarily true – for example, ESRI ArcSDE was a worldwide advertised geo service product which at some time was dropped in favor of a completely new product; SAP R/4 is not entirely backwards compatible with its predecessor R/3; Adobe switched to cloud license for its product suite; and many more examples can be found. Long-living companies in general may decide to terminate support for some product, forcing customers into costly IT infrastructure changes.

V. Conclusion

Open data, open source, and open standards – three terms widely used, from developers to politicians – are three fundamentally different and independent concepts, each one addressing separate concerns. A fourth, in current discussions not yet considered principle, we have brought on the table: open services.

Open policies generally foster proliferation, exchange, and reuse, although many obstacles today remain.

- Data are often open and yet insufficiently accessible due to technical obstacles with data not "analysis-ready"; this issue is tightly connected to service accessibility. In plain words: as long as a service is prohibitively complicated to use, open data are of not much value.
- Open source on principle allows a broader range of programmers to contribute as such, however the number of actual contributors (volunteering and admitted) is not necessarily broad. Contributions and fixes to smaller projects naturally are simpler whereas enterprise-scale tools require the deep knowledge of their developers which effectively leads to a handling similar to closed source tools.
- Open standards are helpful, but most valuable in combination with automatic, authoritative compliance testing; this is not yet in place to a sufficient degree.

An important point is that these "open X" criteria are not tightly connected, but rather independent. For example, open source is not a guarantee (and not a required prerequisite) for open data. Open data can be served through proprietary software, and open-source software may be configured to retain data.

Open source software often is pictured as better software; however, it is not: An indicator of scientific value or rigor; an indicator of software engineering quality; an indicator of quality and duration of support. Rather, it describes a particular business model under which software is provided and maintained.

For the user ultimately value comes from the ease of use, functionality power, and further quality criteria, in short: the quality of the interface – be they user-facing like Web services or internal like database APIs. How some software achieves its goals "under the hood" is secondary. We conclude that most important for a good user experience is reliance on adopted, community-relevant standards.

Well-crafted standard APIs, ideally coming with a solid mathematical semantics underpinning, can make a substantial contribution towards the Holy Grail of uninhibited access and use of data. ISO SQL and the ISO/OGC WCPS geo datacube query language [30] provide a role model.

Open standards are the single most effective measure against vendor lock-in, not open source. The interoperability established thereby – in this context: different servers using identical data will deliver identical results – constitutes a major advantage whose benefits are by far not leveraged in full today.

From a market perspective, open-source and proprietary software are two extremes spanning a spectrum of diverse business models. As they all have their own particular pros and cons they will continue to coexist; radically enforcing only one particular business model does not appear fruitful.

With our discussion we hope to contribute to a less ideological and more pragmatic decision making. In particular, we hope to stimulate awareness for a careful distinction between technical and business model requirements.

ACKNOWLEDGEMENT

This work has been supported partially by EU FAIRiCUBE and NATO SPS Cube4EnvSec.

REFERENCES

[1] I.V. Pasquetto, A.E. Sands, C.L. Borgman: *Exploring openness in data and science: What is "open," to whom, when, and why?* Proc. Association for Information Science and Technology 52(1)2015, doi 10.1002/pra2.2015.1450520100141

[2] N.n.: *Open Definition*. https://opendefinition.org/od/2.1/en, seen 2023-11-04

[3] P. Anagnostou, M. Capocasa, F. Brisighelli, C. Battaggia, G.D. Bisol: *The emerging complexity of Open Science: assessing Intelligent Data Openness in Genomic Anthropology and Human Genomics*. Journal of Anthropological Sciences, Vol. 99 (2021), pp. 135 – 152

[4] M. Mazzucato: *Preventing Digital Feudalism*. Project Syndicate, October 2, 2019, https://www.project-syndicate.org/commentary/platform-economy-digital-feudalism-by-mariana-mazzucato-2019-10, seen 2023-11-04

[5] Slashdot: *Richard Stallman Calls Open Source Movement 'Amoral', Criticizes Apple And Microsoft For 'Censoring' App Installation*. https://news.slashdot.org/story/18/10/28/0125221/richard-stallman-calls-open-source-movement-amoral-criticizes-apple-and-microsoft-for-censoring-app-installation, seen 2023-11-04

[6] A. Christl: *Free Software and Open Source Business Models*. Advances in Geographic Information Science, Springer, 2008, pp 21 – 48, DOI: 10.1007/978-3-540-74831-1_2

[7] OSI: *OSI Approved Licenses*. Open Source Initiative, https://opensource.org/licenses, seen 2023-11-04

[8] N.n.: *The Open Source Definition.* Open Source Initiative, https://opensource.org/osd-annotated, seen 2023-11-04

[9] OGC: *Compliance: Compliant standards and certified products for better-integrated solutions*. https://www.ogc.org/resources/compliance, seen 2023-11-04

[10] M.R. Munafò, B.A. Nosek, D.V.M. Bishop, K.S. Button; C.D. Chambers, N.P. du Sert, U. Simonsohn, E.J. Wagenmakers, J.J. Ware: *A manifesto for reproducible science*. Nature Human Behaviour 1(1)2017, doi: 10.1038/s41562-016-0021

[11] R. Huber, Maxi Kindling, Jens Klump (conveners): ESSI3.2 *The Limits of Open*. EGU 2021 session, https://meetingorganizer.copernicus.org/EGU21/session/40118

[12] N.n.: UNECE. https://unece.org/environment-policy/public-participation/aarhus-convention/introduction, seen 2023-07-09

[13] J.L. Dwyer, D.P. Roy, B. Sauer, C.B. Jenkerson, H. Zhang, L. Lymburner: *Analysis ready data—Enabling analysis of the Landsat archive*. Remote Sensing, 10(9)2018, art. no. 1363. Doi: 10.3390/rs10091363

[14] CEOS: *CEOS Analysis-Ready Data*. https://ceos.org/ard, seen 2023-11-04

[15] OGC: *Engineering Report for OGC Testbed 19 Analysis Ready Data*. http://ogc.pages.ogc.org/T19-ARD/documents/D051/document.html#_f67d9faa-417b-4913-a232-6c9d0c602313

[16] C. Kerstiens: *Contributing to Postgres*. https://www.citusdata.com/blog/2019/01/15/contributing-to-postgres, seen 2021-04-04

[17] n.n.: *PostgreSQL*. https://www.openhub.net/p?query=postgresql

[18] N.n.: *MariaDB*. https://www.openhub.net/p?query=mariadb

[19] OpenHub: *rasdaman*. https://www.openhub.net/p/rasdaman

[20] R. Zicari: *Database Challenges and Innovations. Interview with Jim Starkey*. https://www.odbms.org/blog/2016/08/database-challenges-and-innovations-interview-with-jim-starkey, seen 2023-04-11

[21] P. Wadsworth, J. Brant, P. Brown: *Key IP considerations for smaller enterprises*. WIPO Magazine, June 2021, https://www.wipo.int/wipo_magazine/en/2021/02/article_0008.html, seen 2023-11-04

[22] D. Goodin: *Intel fixes high-severity CPU bug that causes "very strange behavior"*. Ars Technica, 2023, https://arstechnica.com/security/2023/11/intel-fixes-high-severity-cpu-bug-that-causes-very-strange-behavior, seen 2023-11-04

[23] Microsoft: *New Security Signals study shows firmware attacks on the rise; here's how Microsoft is working to help eliminate this entire class of threats*. https://www.microsoft.com/en-us/security/blog/2021/03/30/new-security-signals-study-shows-firmware-attacks-on-the-rise-heres-how-microsoft-is-working-to-help-eliminate-this-entire-class-of-threats, seen 2023-11-04

[24] Bunnie, S. Cross, T. Marble: *36C3 - Open Source is Insufficient to Solve Trust Problems in Hardware*. https://www.youtube.com/watch?v=Hzb37RyagCQ, seen on 2021-06-02

[25] EU: *Proposal for a Regulation of the European Parliament and of the Council on Horizontal Cybersecurity Requirements for Products with Digital Elements and Amending Regulation (EU) 2019/1020*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52022PC0454, seen 2023-11-04

[26] J. Sherman: *Software Development and Open Source Software*. US Department of Defense, 2022, https://dodcio.defense.gov/Portals/0/Documents/Library/SoftwareDev-OpenSource.pdf, seen 2023-11-04

[27] D.W. vanGulik: *Save Open Source: The Impending Tragedy of the Cyber Resilience Act*. https://news.apache.org/foundation/entry/save-open-source-the-impending-tragedy-of-the-cyber-resilience-act, seen 2023-11-04

[28] Synopsis: *Open Source Security and Risk Analysis Report*. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2023.pdf, seen 2024-11-04

[29] Die Welt: *Der Feind im Code – wie verletzlich das freie Internet wirklich ist* (German). https://www.welt.de/wirtschaft/plus236260924/Kostenlose-

Software-Der-Feind-im-Code-So-leicht-wird-sie-zur-Gefahr.html, seen 2023—11-04.
[30] P. Baumann: *The OGC Web Coverage Processing Service (WCPS) Standard*. Geoinformatica, 14(4)2010, pp 447-479