PHILOSOPHICAL TRANSACTIONS A

royalsocietypublishing.org/journal/rsta

Opinion piece

Cite this article: Fortunato L, Galassi M. 2021 The case for free and open source software in research and scholarship. *Phil. Trans. R. Soc. A* **379**: 20200079.
https://doi.org/10.1098/rsta.2020.0079

Accepted: 8 February 2021

One contribution of 15 to a theme issue 'Reliability and reproducibility in computational science: implementing verification, validation and uncertainty quantification *in silico*'.

Subject Areas:
software, computational mathematics, mathematical modelling

Keywords:
free and open source software (FOSS), GNU Scientific Library (GSL), open research, open scholarship, open science, reproducibility

Author for correspondence:
Laura Fortunato
e-mail: laura.fortunato@anthro.ox.ac.uk

The case for free and open source software in research and scholarship

Laura Fortunato[1,2] and Mark Galassi[3]

[1]Institute of Cognitive and Evolutionary Anthropology, University of Oxford, 64 Banbury Road, Oxford OX2 6PN, UK
[2]Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA
[3]Space Science and Applications Group, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

LF, 0000-0001-8546-9497; MG, 0000-0002-3279-2693

Free and open source software (FOSS) is any computer program released under a licence that grants users rights to run the program for any purpose, to study it, to modify it, and to redistribute it in original or modified form. Our aim is to explore the intersection between FOSS and computational reproducibility. We begin by situating FOSS in relation to other 'open' initiatives, and specifically open science, open research, and open scholarship. In this context, we argue that anyone who actively contributes to the research process today is a computational researcher, in that they use computers to manage and store information. We then provide a primer to FOSS suitable for anyone concerned with research quality and sustainability—including researchers in any field, as well as support staff, administrators, publishers, funders, and so on. Next, we illustrate how the notions introduced in the primer apply to resources for scientific computing, with reference to the GNU Scientific Library as a case study. We conclude by discussing why the common interpretation of 'open source' as 'open code' is misplaced, and we use this example to articulate the role of FOSS in research and scholarship today.

This article is part of the theme issue 'Reliability and reproducibility in computational science: implementing verification, validation and uncertainty quantification *in silico*'.

THE ROYAL SOCIETY PUBLISHING

1. Motivation and plan

Sharing software is to computing what sharing recipes is to cooking ([1], p. 53). Just as individual cooks may be more or less willing to reveal the secret ingredient to a favourite dish, individual coders may be more or less inclined to make their programs available to others. Personal dispositions aside, programmers also have to grapple with what has been dubbed '[t]he fundamental tension—how to publish software, comment on it, encourage learning from it, yet still retain commercial and technical control' [2].

This tension arises from the opposition of two camps, which can be caricatured as follows. One maintains that code ought to be accessible and shared with no restrictions; the other insists that code be closely guarded for the sake of intellectual property and, more prosaically, in the prospect of turning that property into profit [2]. What we now term 'free and open source software' (FOSS) emerged piecemeal, starting in the 1970s, along a path unfolding between these two camps.

Nowadays, many researchers are no more than a few clicks or taps away from a large body of literature, data, and software. It can therefore be hard for them to imagine just how different the process of accessing those resources was even 10 years ago—let alone in the early 1970s, when the tension between the two camps began to manifest. Possibly as a result of this discrepancy in day-to-day experience, only a minority across the research community today seem to appreciate the close ties between FOSS and academia. Exceptions include those who work at the interface with FOSS, and a handful of others with a specific interest in the topic. For example, few in the community seem aware that key elements of FOSS emerged in a culture of openness and collective effort, in keeping with long-established academic principles of knowledge-sharing. Fewer still seem to realize that FOSS provides the technical backbone of the infrastructure that enables their effortless access to literature, data, and software (e.g. high-speed networks, responsive search engines).

In fact, today there is often even confusion among researchers as to what FOSS actually is. A case in point is the common interpretation of 'open source' as 'open code'—that is, software for which the human-readable form of the code, or source code, is openly available, with unrestricted access by all. Misled perhaps by similarity in the name (think, for example, of open access in academic publishing), advocates of open science have turned their attention to the software that generates findings reported in scientific papers. In this context, availability of code is widely regarded as essential to reproducibility of the results (e.g. [3–5]). Therefore, a common recommendation is that researchers make the software 'open source', by hosting it in an online repository that is publicly accessible (e.g. [6]). And yet public access to the source code is neither necessary nor sufficient for a computer program to qualify as FOSS.

This paper is aimed at researchers with an interest in FOSS, and specifically in the intersection between FOSS and reproducibility. To the extent that reproducibility is linked to research quality and sustainability, the discussion is relevant to several others in the research community—including support staff, administrators, publishers, funders, and any other group with a stake in the improvement of research culture [7]. Accordingly, we begin by providing definitions of reproducibility and related notions, followed by the argument that the discourse surrounding FOSS needs extending to encompass research and scholarship in the broadest sense (§2).

The rest of the paper comprises two substantive parts. The first is a primer to FOSS (§3). The material is pitched at novices, but it is also suitable for readers beyond this group—for example, researchers who have some familiarity with relevant terms and concepts, but who are now wondering whether the code they shared in a public repository hosted on, say, https://github.com/ is indeed FOSS. We encourage anyone puzzled by this example to engage with §§3a and b in particular.

The second part illustrates the relevance of FOSS to the research community, with reference to the GNU Scientific Library [8–10] as a case study (§4). More advanced readers, and anyone with a specific interest in numerical software for scientific computing, can skip to this part.

We conclude with brief remarks about the role of FOSS in research and scholarship (§5), including the 'solution' to the puzzle above about software hosted publicly online.

Throughout, we draw on our experience advocating for FOSS, which collectively spans several decades across multiple professional contexts, different countries, and disparate fields beyond our own.

2. Setting the scene

There is ambiguity, and some confusion, in terminology linked to reproducibility and related notions. We rely on the definitions provided in the recent report on *Reproducibility and replicability in science* by the US National Academies of Sciences, Engineering, and Medicine [11]. Here, *reproducibility* is defined as 'obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis' ([11], p. 1). Reproducibility is distinct from *replicability*, defined as 'obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data' ([11], p. 1). In turn, both are distinct from *generalizability*, namely 'the extent that results of a study apply in other contexts or populations that differ from the original one' ([11], pp. 1–2).

In line with the remit of the issuing organization, the report relates to computational reproducibility specifically with respect to scientific research [11]. By contrast, our interest is in research generally, which we define loosely as the process of knowledge production and dissemination. Anyone who actively contributes to this process today is effectively a *computational researcher*, to the extent that they use computers to manage and store information. This designation shifts the emphasis from viewing software as a tool, to viewing it as an integral part of the day-to-day workflow for any researcher in any field. By broadening the scope of the discussion in this way, we aim to highlight the relevance of FOSS to disciplines in which the software-as-tool perspective has tended to prevail—including, for example, many fields in the humanities and in the social sciences, and some fields in the natural sciences, in engineering, and in medicine. In other words, we seek to extend the discourse surrounding FOSS from the usual context of *open science* to the broader context of *open research*.

The rationale here is that FOSS is relevant to research practice beyond the domain of scientific programming: any researcher in any field is a computational researcher, whether or not they engage in software development. Against this rationale, a further step is to extend the discourse surrounding FOSS from open research to *open scholarship*—that is, to encompass both the research process itself and the various resources (e.g. intellectual, financial, administrative) that enable it. To our knowledge, related discussions to date have focused on the interplay between FOSS and open access in academic publishing (e.g. [12]). Yet with the ever-growing reliance on digital tools and services across the full range of research activity, it seems timely to explore the role of FOSS in other domains—for example, as applied to the systems used to publish scholarly content and teaching materials, or to those used for the day-to-day management of research funding. In our experience, the conversation is often framed in terms of trade-offs between the cost-effectiveness of FOSS solutions and the convenience of non-FOSS products. Such narrow framing obscures the many parallels between FOSS and open scholarship, and their convergence of interests in contributing to the public good [13].

3. A primer to free and open source software

This primer aims to equip readers with a working knowledge of FOSS. First, we provide a technical definition, and in the process we unpack several common misconceptions that arise from it (§3a). Next, we frame the definition in legal terms, outlining relevant notions with respect to copyright and to software licencing (§3b). Finally, we introduce key figures and events in the history of FOSS (§3c).

A specialized literature exists on each of these topics, and we refer to it where relevant. Some of the resources we include are aimed specifically at researchers—typically researchers working in quantitative fields, and often in the context of open science (e.g. [14–17] for the social sciences). We seek to complement this literature with a primer suitable for novices. Therefore, we assume

no background knowledge beyond the level of computer literacy than can be expected of those in the research community with an interest in open scholarship—including researchers in any field, as well as support staff, administrators, publishers, funders, and so on (§§1 and 2). Within this community, anyone who wishes to engage productively with FOSS stands to benefit from familiarity with the material covered here.

(a) What's in a name?

Throughout this paper we use 'free and open source software' and its acronym 'FOSS' as generic labels, resorting to 'free software' and 'open source software' where it is useful to do so—for example, to distinguish between these two categories of software. It is however important to note at the outset that the overlap between the two categories far outweighs the discrepancies: therefore, bar few exceptions, the corresponding labels refer to the same body of computer programs (§3biii).

To explain why multiple terms exist, in §3ai we introduce two social movements active in the FOSS community, together with the definition of FOSS provided by each. In §3aii we outline common misconceptions linked to the different terms, and we address the resulting confusion. A summary of key points is in §3aiii, including a justification of the approach to terminology we take in this paper.

(i) Technical definition(s)

The term 'free and open source software' is a sort of compromise between the philosophical and political views of two social movements ([18], p. 81): the *free software movement*, stewarded by the Free Software Foundation [19], and the *open source (software) movement*, stewarded by the Open Source Initiative [20]. We provide historical context for each in §3c; for now, they can be briefly characterized as follows. The free software movement is 'a campaign for freedom for users of software' [21]—that is, freedom from restrictions on use, study, modification, and distribution of software. The open source movement is, effectively, 'a marketing program for free software' ([22], p. 7), centred on the practical advantages that can derive from that freedom.

The Free Software Foundation and the Open Source Initiative provide, respectively, the Free Software Definition [23] and the Open Source Definition [24]. Each definition reflects the views of the corresponding movement, outlining the requirements a computer program must fulfil to qualify as free software, in one case, and as open source software, in the other.

The requirements outlined in the two definitions are formalized in the licence that accompanies the software, which technically determines whether the software is free and/or open source. Generally, then, *free and open source software* is any computer program released under a licence that grants users rights to run the program for any purpose, to study it, to modify it, and to redistribute it in original or modified form ([25], p. 3). Software that does not meet this definition is termed *proprietary* (also *non-free* and *closed source*, as antonyms to 'free' and to 'open source', respectively). All of these terms apply both to the software itself and to the licence that accompanies it (§3bii).

The generic definition of FOSS given above paraphrases the Free Software Definition, which centres on the *four essential freedoms* ([25], p. 3)—that is, the rights to run, study, modify, and redistribute a computer program. The Open Source Definition was derived from this framework in a more or less direct line of descent ([26], pp. 172–174; [18], p. 77), and it is equivalent for the purpose of defining FOSS in relation to the licence. However, it is not as easily paraphrased, nor is it readily summarized, being a 10-clause 'bill of rights for the computer user' ([26], p. 171), linked to a trademark on the term 'Open Source' administered by the Open Source Initiative (§3ciii).

(ii) Misconceptions

We can now proceed to introduce terminology to address key misconceptions surrounding FOSS. These misconceptions stem from use of the terms 'free' and 'open (source)' in relation to software, and they are common among supporters and detractors of FOSS alike.

Confusion linked to 'free'

One misconception arises from ambiguity in the English adjective 'free', which can refer both to liberty (as in 'freedom') and to price (as in 'free of charge') ([25], p. 3). To address this ambiguity, the French/Spanish adjective 'libre' is sometimes used in addition to 'free', or in alternative to it (i.e. *free/libre software* or *libre software*; also *free/libre and open source software* and its acronyms, *FLOSS* or *F/LOSS*). In other words, the reference here is to 'free as in freedom', rather than to 'free as in gratis'—a distinction colloquially captured by the expression 'Think of "free speech," not "free beer".' ([27], p. 43).

The term 'freeware' is easily confused with 'free software', but the two are not synonymous ([28], p. 73). In particular, *freeware* and its antonym *payware* are sometimes used to indicate software that is obtained, respectively, at no cost and in exchange for money (or, more commonly, software distributed to users *non-commercially* and *commercially*). Crucially, FOSS and proprietary software can both be distributed for free or for a fee ([27], p. 43); the difference is as follows. In the case of FOSS, anyone who has a copy of a computer program can decide whether to distribute copies of the program, and whether and how much to charge for them (§3bii). In the case of proprietary software, the proprietor has sole discretion as to whether copies of the program can be distributed, and under what conditions (the proprietor is the 'owner' of the software—technically, the holder of the copyright or, equivalently, the licensor; §3bi).

Therefore, it is incorrect to juxtapose FOSS and commercial software as mutually exclusive concepts: there is commercial software that is FOSS, and there is non-commercial software that is proprietary ([28], p. 74). In fact, large fortunes have been accumulated over the past three decades from the commercial distribution of FOSS, together with provision of related services (e.g. support).

Confusion linked to 'open (source)'

A relevant notion here is the distinction between distribution of software with or without source code. In one case, users have access to a human-readable version of the software; in the other, access is restricted to the executable machine code compiled from the source code. The term *source-available* designates any computer program for which the source code is available to view. Access to the source code is a prerequisite for the four essential freedoms, in that it enables users to readily study and/or modify the program (§3ai). By definition, then, all FOSS is source-available, and any software distributed only in compiled form is proprietary. However, not all source-available software is free and open source. For example, if the source code can be viewed for reference only, then the software is source-available and proprietary, in that modification and redistribution are not allowed.

Much confusion arises from interpretation of the term 'open source' to mean, effectively, 'source-available'. To address the confusion, the Open Source Definition states explicitly at the outset that "[o]pen source doesn't just mean access to the source code" [24]—rather, availability of the source code is only one of multiple requirements a computer program must fulfil to qualify as open source software (§3ai).

A related issue is confusion about who has access to the source code. A common misconception is that 'open source' implies that the source code is publicly available. That is not the case—more narrowly, the four essential freedoms granted by FOSS apply to users of the software (technically, the licencees; §3bi). In a way, this is a moot point, in that if a computer program is FOSS, there is nothing to prevent users from redistributing its source code, and thus from making it available to the public. At the same time, it is important to understand that public availability of the source code is not required for the program to qualify as FOSS—and, conversely, that not all programs for which the public has access to the source code are FOSS.

The misplaced notion that FOSS necessarily involves public access to the source code extends also to the development model. In particular, another common misconception arises from conflation of 'open source' with 'open development', i.e. collaborative development of a piece

of software, typically through volunteer effort. The defining features of FOSS make it especially suited to development in this way, and several successful FOSS projects have leaned heavily on it (e.g. development of the Linux kernel; §3ciii). Indeed, the Open Source Initiative explicitly highlights this approach to developing computer programs as one of the key practical advantages linked to software freedom (§3ai). However, as in the case of public access to the source code, open development is not required for a program to qualify as FOSS—and, conversely, not all programs developed in the open are FOSS.

(iii) Summary

To summarize, free and open source software, or FOSS, is any computer program that is freely modifiable and redistributable, in the sense that users are granted rights to run the program for any purpose, to study it, to modify it, and to redistribute it in original or modified form. To this end, users must be given access to the software in human-readable form—that is, to the program's source code.

Any program that is not freely modifiable and redistributable in this way is proprietary software. Broadly, whether a program is free and open source versus proprietary is determined by the licence that accompanies it. The licence specifies what restrictions apply to the program in terms of use, study, modification, and distribution (§3b).

Several misconceptions arise from this definition, linked to confusion in terminology. In particular, the terms 'free' and 'open (source)' lead to misunderstandings about the price of the software, about access to the source code, and about the development model. Crucially, whether a program is FOSS or proprietary software is independent of whether it is distributed for free or for a fee—that is, non-commercially or commercially. It is also independent of whether the source code is open, such that the public has access to it. Similarly, it is independent of whether development occurs in the open, i.e. involving scrutiny and contributions by users, and by the broader community, as part of a collaborative effort.

In part, the confusion in terminology reflects the historical tension between two social movements active in the FOSS community: the free software movement (§3cii) and the open source movement (§3ciii). One centres on freedom, in terms of the rights granted to users to run, study, modify, and redistribute the software (namely, the 'four essential freedoms'). The other centres instead on the practical advantages that can derive from that freedom—for example, to enable (and, possibly, motivate) users of a piece of software to work collaboratively to add a desired feature, or to continually review and fix security issues.

Whatever the differences between the two movements, both are firmly grounded on the premise that humanity as a whole is better served by software that is free and open source. Advocates of FOSS generally fall somewhere on 'a multidimensional scattering' ([22], p. 6) of opinions, underscoring the perspective of each movement to varying degrees. In the interest of full disclosure, we note that our philosophical and political stance tends to gravitate towards the free software portion of that state space.

In an effort to mitigate the confusion, the approach to terminology we take in this paper departs somewhat from our own 'FOSS orientation'; a summary of the approach is in table 1. Broadly, we use 'free and open source' versus 'proprietary' as generic terms, with specific reference to 'free' and to 'open source' in the context of licences (§3bii). We also resort to the specific labels in outlining the history of FOSS (§3c), and to highlight the views of one movement in particular (§4).

We close by reiterating a point we raised at the beginning of the section—namely, that the terms 'free software' and 'open source software' effectively refer to the same body of computer programs. Therefore, the philosophical and political nuances underlying variation in terminology are of limited practical relevance to most, including the intended readers of this paper (§1). In everyday practice, the research community stands to benefit from FOSS through the rights granted to users of a program (§3b)—not from the specific views of one movement, nor from subtleties in the use of vocabulary (see [18] for an alternative take on this issue). At the same time,

Table 1. Summary of key terms.[a]

label	definition	synonym(s)	antonym(s)
free and open source software[b] (*FOSS*)	any computer program released under a licence that grants users rights to run the program for any purpose, to study it, to modify it, and to redistribute it in original or modified form	*free/libre and open source software* (*FLOSS* or *F/LOSS*)	*proprietary software*
free software[c]	FOSS distributed with a licence approved by the Free Software Foundation	*free/libre software* or *libre software*	*proprietary software* or *non-free software*
open source software[d]	FOSS distributed with a licence approved by the Open Source Initiative	N/A	*proprietary software* or *closed source software*
freeware[e]	any computer program, FOSS or proprietary, distributed to users free of charge	*non-commercial software*	*payware* or *commercial software*
source-available software[f]	any computer program, FOSS or proprietary, for which the source code is available to view	N/A	N/A

[a]Terms discussed in §§3ai and ii are reported in italics. Note that alternative definitions of these terms exist, and that usage varies; a justification of the approach taken in this paper is in §3aiii. [b]The generic definition of FOSS given here is based on the Free Software Definition (§3ai). The rights to run, study, modify, and redistribute a computer program are the 'four essential freedoms', which rest on availability of the program's source code (i.e. a human-readable version of the program; §3aii). [c]See §3biii for discussion of FOSS licences approved by the Free Software Foundation, i.e. licences compliant with the Free Software Definition (§3ai). [d]See §3biii for discussion of FOSS licences approved by the Open Source Initiative, i.e. licences compliant with the Open Source Definition (§3ai). Note that the term 'Open Source' is a trademark administered by the Open Source Initiative; the Open Source Definition spells out the trademark conditions (§3ciii). [e]Note that the term 'freeware' is not univocal, as its usage has changed over time ([29], p. 96). [f]By definition, all FOSS is source-available, and any software that is not source-available is proprietary. However, not all source-available software is free and open source (§3aii).

exploring the intersection between FOSS and reproducibility, as we set out to do in this paper (§1), requires clarity on the concepts outlined here.

(b) Legal framework

In §3a we defined FOSS and its antithesis, proprietary software, in terms of the licence that accompanies a computer program. Here we outline the legal basis for this distinction. Perhaps counterintuitively, the existence of FOSS is predicated on the notion of copyright, so we begin by outlining relevant concepts (§3bi). We then discuss how these concepts apply to software (§3bii), alongside some practicalities related to releasing code as FOSS (§3biii).

Our aim is to provide a brief overview of these issues aimed at the research community (§1). We emphasize that the full picture is considerably more complicated than what we present here, owing in part to the ambiguities in terminology discussed in §3a. Additional contributing factors are variation in legislation by country and across jurisdictions, and old and new debates about interpretation and application. Relevant technical detail is beyond the scope of this paper, however, so we point readers to ([32], pp. 10–14) and ([22], ch. 9) for useful summaries, and to [33] for a practical guide aimed specifically at scientists.

(i) Basics of copyright

The vast majority of countries worldwide adhere to a framework to coordinate copyright legislation internationally. Within this framework, the original expression of an idea through sound, imagery, or text is automatically protected by copyright, such that authors have exclusive rights to commercial exploitation of their work (e.g. by selling copies of it, or by displaying it publicly). Broadly, copyright law places restrictions on use, modification, and distribution of the work, and of works derived from it, for a set period of time. Once that period lapses, the work is in the public domain—it is uncopyrighted. At this point, no one can claim ownership of it, and anyone can exploit it commercially.

Generally, if the work is produced by an employee to their employer's specification, then the holder of the copyright is the employer, not the employee. There are exceptions, however: for example, in some countries the work of government employees 'defaults' to the public domain. Furthermore, in some jurisdictions the work can be placed in the public domain before the required time has passed. To this end, the copyright holder must provide an explicit statement that they voluntarily waive their rights in the work (§3biii).

Across jurisdictions the copyright holder can specify which rights, if any, are granted to recipients of the work, by way of a licence. The licence is a legal document that stipulates how the work and its derivatives may be used (technically, the copyright holder is the licensor, the recipients are the licencees). For example, if the licence states 'All rights reserved', then the rights to use and repurpose the work remain with the copyright holder ([32], p. 11).

Licences can be grouped into two broad categories, based on what restrictions they impose on further distribution of the work and of its derivatives, whether non-commercially or commercially. Specifically, copyleft licences require anyone who distributes copies of the work, with or without changes, to do so under the same terms as the original copy, or under equivalent ones; non-copyleft licences do not include a requirement to this effect.

At a conceptual level, the significance of copyleft is to protect the rights originally granted to recipients of the work by the copyright holder. The outcome is that no one, not even the copyright holder, can ever deprive others of those rights. By contrast, non-copyleft allows anyone to remove some or all of those rights in redistribution. In practice, then, recipients of a derivative of the work may face greater restrictions than recipients of the original work. For this reason, copyleft licences are considered reciprocal and protective (also, 'share-alike'); non-copyleft ones are considered permissive (also, 'not share-alike').

Effectively, the distinction between copyleft and non-copyleft licences captures a trade-off between the freedom of any one individual to use and repurpose the work at any time, and everyone's freedom to do so at all times (§3aii). In a way, copyleft is a 'hack' on copyright, in that it secures the opposite of what the law seeks to achieve: a copyleft licence ensures that copyrighted work can be freely used, modified, and distributed, and that everyone can do so in perpetuity. Hence the play on words: 'copyleft' is intended as the inverse of 'copyright' ([34], pp. 184–185).

(ii) Software licences

We can now discuss how the general provisions outlined in §3bi apply to computer programs, with reference to the simplified classification of software categories in figure 1. We point readers to ([28], p. 68) for an extended representation; that level of detail is beyond the scope of this paper.

To the extent that a computer program embodies its author's original creation, it is subject to the automatic protection provided by copyright law ([28], p. 70). With few exceptions (e.g. the work of government employees in some jurisdictions; §3bi), by default any program is thus *copyrighted software*. As discussed in §3bi, a licence can be used to stipulate what restrictions apply to the program and to its derivatives. Through the licence, the author specifies which rights they retain and which they grant to others, in terms of use, modification, and distribution.

Building on the discussion in §3ai, a *FOSS licence* can be defined generically as one that grants users rights to run the software, to study it, to modify it, and to redistribute it in original or modified form. A *proprietary licence* is one that does not meet this definition, implying that some

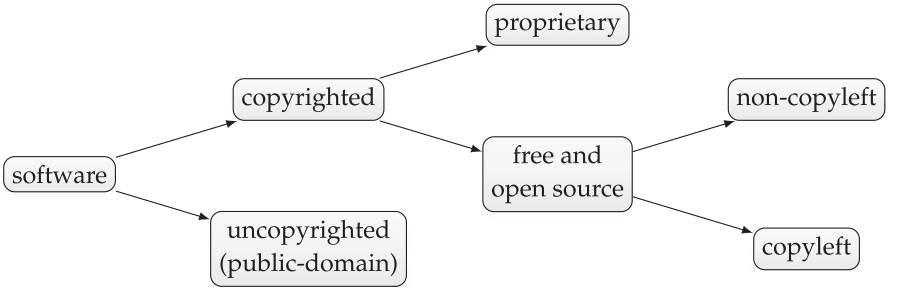

Figure 1. Simplified classification of software categories.

of those rights remain with the copyright holder. For example, the licence may state that recipients are allowed to use the program, but they are not allowed to modify it, nor to distribute copies of it.

A FOSS licence is termed *copyleft* if it requires that distribution of copies of the program, with or without changes, occur under the same terms as the original copy (i.e. with the same licence, or an equivalent one). Conversely, a FOSS licence is termed *non-copyleft* (or, more commonly, *permissive*) if it does not include such a requirement. In practice, the difference is as follows. In one case, modified versions of the program cannot be distributed as proprietary software; in the other, they can. In other words, copyleft does not allow derivative programs that are proprietary, whereas non-copyleft does.

Widely used copyleft licences are the GNU General Public License (GPL) [35], the GNU Lesser General Public License (LGPL) [36], and the Mozilla Public License (MPL) [37]. Widely used non-copyleft ones are the Apache License [38], the 2-clause BSD licence (also called 'simplified BSD' or 'FreeBSD') [39,40], and the MIT licence (more precisely, the Expat licence) [41,42]. Within each category, some licences are more restrictive (and thus more reciprocal and protective), others less so (and thus more permissive) (§3bi). Overall, the GPL and the MIT licence sit at either end of this spectrum ([32], p. 14).

Finally, a program can be *uncopyrighted software* (or, more commonly, *public-domain software*); we outlined some mechanisms to this effect in §3bi. Any program for which the source code is in the public domain is FOSS, in that it can be freely used, studied, modified, and distributed. There are no restrictions on distribution, therefore derivative programs can be proprietary ([28], p. 70). A point to note here is that since public-domain software is not covered by copyright, by definition it cannot be subject to a licence (§3bi). As we discuss in §3biii, this is an issue to consider when releasing code as FOSS.

(iii) Practicalities

In practice, all that is required to specify the licencing terms for a computer program is to include a file reporting the full text of the licence; typically, this file has the name 'LICENSE' or 'COPYING'. Additionally, a short note is added as a comment at the top of each source code file, stating the copyright date, the name of the copyright holder, the name of the licence, and the location of the full text of the licence ([22], pp. 20–21). If a piece of software does not include a licence, it cannot be legally used, modified, or distributed (§3bi): doing so would be an infringement of copyright, potentially resulting in legal action ([32], p. 30).

The simple procedure outlined above is complicated by the large number of licences available. Anyone seeking to release a program as FOSS can consult information compiled by licence-certifying organizations in the FOSS community, such as the list of licences produced by the Free Software Foundation [43], the Open Source Initiative [44], the Debian Project [45], the Fedora Project [46], and the Linux Foundation [47]. These organizations review existing licences to determine which meet the requirements for free and/or open source software, and to examine issues relating to licence compatibility. Some organizations are also directly involved in managing

specific licences. For example, the Free Software Foundation holds the copyright to the GPL (§3bii), and it coordinates work relating to this licence.

The lists of FOSS licences approved by the various organizations do not overlap fully. In some cases, the discrepancy arises for practical reasons, for instance if a licence is yet to be reviewed by a given organization ([22], p. 20). In other cases, the discrepancy reflects variation in philosophical and political views. For example, licences approved by the Free Software Foundation are compliant with the Free Software Definition; those approved by the Open Source Initiative are compliant with the Open Source Definition (§3ai). Generally, most of the licences approved by the Free Software Foundation are also approved by the Open Source Initiative, and many of those approved by the Open Source Initiative are also approved by the Free Software Foundation ([28], p. 69). In this context, the terms 'free' versus 'non-free', and 'open source' versus 'closed source', can be used to indicate whether a licence is approved by one or the other organization (table 1). We emphasize, however, that the overlap between the two lists far outweighs the discrepancies—with few exceptions, then, the terms 'free' and 'open source' refer to the same body of computer programs (§3aiii). In most cases, software that is free is also open source; in slightly fewer cases, software that is open source is also free ([32], p. 8).

One notable discrepancy between the two lists relates to public-domain software (§3bii). Both organizations view such software as free and/or open source, but they differ in their approach to it [48,49]. In particular, the Creative Commons CC0 public domain dedication [50] is a widely used legal instrument for releasing material into the public domain. Technically, CC0 is a waiver text, not a licence (it includes a fallback licence option for application in jurisdictions that have no concept of dedicating work to the public domain, by way of the copyright holder waiving their rights in the work; §3bi). Creative Commons CC0 is approved as a free software licence by the Free Software Foundation [51]. However, it is not approved as an open source licence by the Open Source Initiative [52], on the grounds that public-domain software cannot be subject to a licence (§3bii).

This example illustrates only one of the many complications that surround software licencing, and we refer readers to the resources listed at the beginning of the section for additional information. §§4 and 5 include discussion of the issue in relation to research and scholarship specifically.

(c) Historical framing

In §3ai we introduced two social movements active in the FOSS community. Here we frame the two in historical context, as background to concepts discussed in §§3a and b, and to the case study we present in §4. Our account is necessarily incomplete, focusing on key developments directly relevant to the material covered in those sections. We point readers to ([22], pp. 3–7) for a useful overview of the history of FOSS, and to [53–60] and various chapters in [61] for detailed treatments of specific topics.

(i) Origins and key developments through the early 1980s

The origins of today's FOSS culture can be traced back to the tradition of openness, collaboration, and knowledge-sharing that characterized the early decades of computing, sometimes referred to as 'Hacker Ethic' ([58], ch. 2). In §1 we pinpointed to the early 1970s the emerging tension between the hacker tradition and the commercial interests of a nascent software industry. Rather than a specific event, this window captures several significant interrelated changes that occurred around that time.

The 'unbundling'

One such change was the 'unbundling', i.e. separation of the sale of software and services from the sale of hardware, starting in 1969 with IBM ([57], pp. 106–107). Up until that point, corporate profits had been driven by manufacture and marketing of hardware. Therefore, the cost of software and services had been combined, or 'bundled', with the cost of hardware. Each

hardware model was highly specialized, as consequently was the software written for it. As a result, manufacturers had little incentive to sell software separately from hardware. They also had no incentive to prevent software from being shared widely. In fact, the widespread distribution of programs for a specific model often led to user-contributed updates, which improved sale prospects for that model ([32], p. 5). This dynamic affected early corporate motivations for investment in software development. It also affected how companies distributed source code, and the relationship they kept with volunteer developers (e.g. users in academia operating the hardware, typically researchers and technicians).

The unbundling decision by IBM was precipitated by the threat of an antitrust lawsuit by the US government, on the charge that the bundling of hardware and software was anticompetitive. More broadly, the decision captured an inevitable transition in the computer industry ([57], p. 169). Hardware manufacturers had begun to realize that software development required substantial effort, and that it held potential as a revenue stream in and of itself ([32], p. 4). For example, the introduction of high-level programming languages meant that software could be ported to different hardware models, creating the conditions for a commercial software industry separate from the sale of hardware ([22], p. 3). Beginning with the unbundling by IBM in 1969, software effectively became merchandise.

Rise of Unix

That same year saw another key development, namely creation of the Unix operating system [62]. It is no exaggeration to say that Unix provides the foundation to the world's digital infrastructure ([60], pp. 158, 165). Today, the majority of web servers and the top 500 supercomputers run operating systems based on the Unix design. Unbeknown to many, practically every 'smart' device we interact with on a daily basis also runs some variant of Unix—including our phones, computers, routers, headsets, and so on. For example, both Android and iOS are variants of Unix, as are the GNU/Linux operating system in all its flavours (e.g. Debian, Ubuntu), Chromium/Chrome OS, and macOS.

Development at Bell Labs. Unix was created beginning in 1969 by Ken Thompson, Dennis Ritchie, and other researchers at Bell Telephone Laboratories, 'Bell Labs' for short. Bell Labs was a division of AT&T, the American Telephone and Telegraph Company; at the time, it was co-owned with Western Electric, which was itself a subsidiary of AT&T ([53], p. 57).

Unix was initially released for internal use at Bell Labs. From the early 1970s it was licenced to educational institutions for a nominal 'media fee', and to a limited number of corporate costumers for a commercial fee ([60], pp. 132, 143). For legal reasons, the company was prohibited from turning the system into a product and marketing it ([53], pp. 57–61). Therefore, the licences included all of the source code, but on an 'as is' basis ([60], pp. 134–135). Furthermore, there were restrictions in place: for example, licencees in universities could use the system exclusively for educational purposes ([60], p. 132).

The earliest versions of Unix were written in assembly language (i.e. the machine-dependent language for the specific hardware model available to the researchers at Bell Labs). Relatively early on, most of it was rewritten in C, a high-level programming language created by Ritchie at around the same time ([60], pp. 76–78). As a result, Unix was the first operating system to be ported to a wide variety of platforms ([60], pp. 140–141).

Development at Berkeley. The system's portability allowed the source code to be easily adapted to different machines, including the low-end models often available in university departments. Coupled with the inexpensive licencing, this feature led to widespread circulation of Unix in academia, starting in the early 1970s ([60], p. 134). In turn, uptake of the system in this setting contributed to an upward spiral of technical innovation. For example, from the mid-1970s a group of researchers at the University of California, Berkeley developed a version of Unix with expanded scope and capabilities ([63], pp. 31–33). This version was released beginning in the late 1970s as the Berkeley Software Distribution (BSD) under a permissive licence (§3bii).

Unix was thus adopted as an educational tool across institutions in several countries (e.g. [64]): its relative simplicity was well suited to demonstrating the implementation of a real-world operating system. As computer science graduates ventured from university into industry, they contributed to the growing popularity of the system outside academia ([60], pp. 144–145).

Commercialization. Unix development was run through the 1970s 'as a loosely proprietary research project at AT&T' ([22], p. 5). Given that the company did not provide any support, users started coming together on a regular basis to share ideas and software ([53], p. 65). Improvements were fed back to the researchers at Bell Labs for integration into later releases, and this group acted as an informal 'clearing house' for exchange of the system in the US.

By the late 1970s Unix had accrued thousands of users. The surge in popularity led AT&T to cease distribution on the previous terms, and to begin enforcing its copyright on the source code. This shift in posturing was followed by a change in the company's legal position in the early 1980s, which lifted the restrictions that had prevented the full commercialization of the system until then ([60], pp. 143–145).

Unix was thus promptly taken to market. The new licencing terms were not as favourable as the previous ones had been, so the Berkeley researchers continued to develop and distribute the BSD version as an alternative to the commercial products available from AT&T ([60], pp. 154–155). The researchers at Bell Labs were no longer able to act as a clearing house, even informally, and this role was taken up by their peers at Berkeley ([63], pp. 34–35).

(ii) Key developments in the 1980s

The free software movement

By the time Unix was taken to market, the commercial software industry had converged on a clear position in relation to the distribution of source code. Companies needed to produce reliable products in order to gain an edge over the competition ([57], p. 108). To this end, they needed to stem the unrestricted sharing of software that had prevailed in the early decades of computing. For example, they would now require users to sign non-disclosure agreements on the code, or they would simply not distribute it ([22], p. 3).

The changes occurring in industry spawned lucrative employment prospects for programmers. They also led to frustration, however, arising from the cost of acquiring software and, more importantly, from no longer being able to customize programs and then share the modified, improved versions ([32], p. 5). One person particularly aggrieved by the situation was Richard Stallman, a programmer on staff at the AI Lab, the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT). Stallman had thrived in the hacker culture of the lab through the 1970s, but this community had progressively disbanded over the years, as many of its members joined companies to work on proprietary software.

As the changes that were underway in industry finally caught up with MIT in the early 1980s, the situation became intolerable for Stallman. Resenting the growing restrictions on both himself and others, he resolved to create GNU, a Unix-compatible system that was free software (§3ai). 'GNU' is a recursive acronym for "Gnu's Not Unix", indicating that the system would be Unix-like in design, but that it would not include any Unix code.

Stallman announced this plan in a posting to two Usenet newsgroups (`net.unix-wizards` and `net.usoft`) in 1983 [65]. He then resigned from the AI Lab, to prevent MIT from interfering with release of GNU as free software, and in 1984 he began work on what would later become known as 'GNU Project' [1,66]. The project would produce a complete operating system from scratch, including (i) a kernel (i.e. the program that allocates resources and interfaces with the hardware), and (ii) a full collection of developer and user-level utilities, to allow both development and general use. The aim was for Stallman to 'be able to get along without any software that is not free' ([65], p. 27).

Opening with 'Free Unix!', the Usenet posting invited '[c]ontributions of time, money, programs and equipment' ([65], p. 26). This 'call to arms' marks the birth of the free software

movement (§3ai). In 1985 Stallman issued the GNU Manifesto [21,67], in which he expanded on the initial announcement, with more detail on the project's philosophical underpinnings. For example, he explained that GNU software is not in the public domain, to ensure that all versions remain free in perpetuity (§3bii). That same year he established the Free Software Foundation to support the movement, and in 1986 he published an early version of the Free Software Definition [68] (§3ai). He then introduced the notion of copyleft (§3bi), through release of the first version of the GPL in 1989 (§3bii). Effectively, several key concepts outlined in §§3a and b were first explicitly articulated by Stallman in the 1980s, and they delineate FOSS as we understand it today.

Demise of Unix

In many ways, the free software movement resembled the community of Unix users established in the 1970s. Together with the researchers working on the system at Bell Labs and at Berkeley, this community had been putting into practice the culture of knowledge-sharing that would drive Stallman's ideological campaign beginning in the 1980s ([22], p. 5). What had been missing in the early phases of Unix development, however, was a clearly stated philosophy around software licencing (§3bii), and the system suffered uncertainty and loss of opportunity as a result ([60], p. 157).

One issue was the proliferation of Unix and Unix-like variants, following commercialization of the system by AT&T in the early 1980s ([60], pp. 153–157). A multitude of implementations appeared, some based on versions available from AT&T, some based on BSD, some based on combinations of the two ([53], pp. 209–210). In particular, the licencing of BSD allows proprietary derivatives (§3bii), and BSD code was thus often incorporated into proprietary variants (notably, for example, Apple's macOS and iOS are proprietary operating systems based on derivatives of BSD). Such proliferation dispersed effort, creating competition, fragmentation, and incompatibility. Indeed, one of the benefits for 'all computer users' that Stallman envisioned from releasing the GNU system as free software was precisely to avoid 'much wasteful duplication of system programming effort'—effort that could be directed 'instead into advancing the state of the art' [21].

(iii) Key developments in the 1990s

Enter Linux

The GNU system was nearly complete by the early 1990s, comprising a large collection of packages. However, the design that had been selected for the kernel proved particularly difficult to implement. Therefore, development of this component lagged behind ([22], p. 4).

The system was made operational through inclusion of a kernel based on a less ambitious design. This kernel, called 'Linux', was created by Linus Torvalds beginning in 1991 ([69], pp. 103–104). Torvalds, then a computer science student at the University of Helsinki, had started the project as a hobby, but he managed to produce a working kernel from scratch in a relatively short period of time. In part, he was able to do so with contributions by developers around the world, leveraging the growing capability of the Internet, which was getting off the ground in those years ([22], p. 3).

Linux was released as free software under the GPL from 1992 (§3bii). At this point, work began to integrate the kernel with GNU software, so as to produce a fully functional operating system ([69], p. 107])—technically, this is the GNU/Linux operating system, although it is commonly referred to simply as 'Linux' (see discussion in [70,71]).

Legal troubles

Had a kernel been available as part of the BSD version, Torvalds would likely not have bothered with his hobby project in the first place [72]. However, development of BSD had been slowed down in the early 1990s by legal turmoil around licencing ([60], p. 157).

Early versions of BSD included Unix code, and they were therefore subject to an AT&T licence. After AT&T began clamping down on distribution of the system, the researchers at Berkeley put substantial effort towards reimplementation of various utilities which used Unix code, so that BSD and its derivatives would no longer be subject to the AT&T licencing requirement ([63], pp. 40–43). Despite this effort, a legal quagmire ensued between 1992 and 1994 over intellectual property rights to the software. In particular, a subsidiary of AT&T accused BSD developers at Berkeley and elsewhere of violating the copyright on Unix; the University of California responded with a counterclaim, accusing AT&T of breaching the terms of the BSD licence in using code developed at Berkeley ([63], pp. 44–45).

The legal troubles resolved largely in favour of the BSD developers, and work on the project continued at Berkeley through the mid-1990s ([63], pp. 45–46). Yet by the time the legal ambiguity had cleared, the market had shifted to GNU/Linux as the leading Unix-like operating system that was free software, with BSD and its derivatives relegated to relatively minor roles.

The open source movement

By the late 1990s, GNU/Linux had also gained large market shares as an alternative to proprietary operating systems, notably Windows, both among individual users and in business. Free software quickly gained momentum from then on, in industry as well as outside the developer community. Some in the community felt that this turning point called for a 'rebranding', so as to make free software palatable to mainstream corporate types. A sticking point was the ambiguity arising from use of the term 'free' in relation to software (§3aii). In particular, the need to explain the concept with reference to 'free speech' versus 'free beer' diverted attention from the core issue of source code availability [73,74]. Besides being confusing to newcomers, the term failed to convey the possible commercial advantages arising from software freedom [73], and in particular the greater levels of innovation enabled by an open development model (§3aii).

Brainstorming on the topic at a strategy session in 1998 converged on 'open source software' as the preferred alternative [73,74]. Following extensive promotion, the term quickly gained traction in industry, in the media, and among the public, with early support from prominent figures in the community, including Torvalds [74]. By contrast, Stallman rejected the term on the grounds that it carries its own ambiguity ([18], pp. 77–78) (§3aii) and, more importantly, for shifting the focus away from 'the ideas of freedom, community, and principle' ([1], p. 70).

The Open Source Initiative was founded as part of this rebranding effort (§3ai). The term 'Open Source' was registered as a certification mark (i.e. a trademark for application to other people's products) ([26], p. 174), with the trademark conditions spelled out in the Open Source Definition [24] (§3ai). Use of the term 'Open Source' is thus linked to the Open Source Definition, and the Open Source Initiative was established primarily to manage the trademark ([26], p. 174).

More broadly, the birth of the open source movement in the late 1990s finally made explicit the 'multidimensional scattering' ([22], p. 6) of opinions that had emerged across the FOSS community by that point (§3aiii). Importantly, the movement introduced vocabulary and concepts necessary for education and advocacy about FOSS on pragmatic grounds [74], distinct from the ideological framing of the free software movement ([22], p. 7).

4. The GNU Scientific Library

We illustrate how the notions introduced in §3 apply in the context of research and scholarship with reference to the GNU Scientific Library (GSL) [8–10], a widely used collection of numerical routines for scientific computing. This choice of example is opportunistic: one of us (Galassi) was involved in initial conception of the library in the second half of the 1990s, and in its subsequent development over several years. Therefore, we can draw on his first-hand account of the motivations driving the project, and of how these informed the design of the library.

This section is in three parts. We begin by sketching the landscape of resources for scientific computing available in the 1990s (§4a). We then frame GSL against this landscape, which provided the technical backdrop to conception and development of the library (§4b). We conclude by outlining lessons learnt from the case study (§4c).

(a) The landscape

The earliest motivation for development of GSL came in the early 1990s when Galassi, then a visiting graduate student at Los Alamos National Laboratory, attempted to obtain source code for the Common Los Alamos Mathematical Software (CLAMS) library for numerical analysis. Instead of the code he received a series of bureaucratic answers: in practice, the maintainers of the library had not yet decided under what terms to release the software. The CLAMS library included all of the routines in the public-domain SLATEC library (described below), alongside some routines unique to Los Alamos National Laboratory ([75], p. 8-5). Without the source code it was impossible to determine which routines came from which codebase. It was therefore impractical, at best, to reproduce results generated using the CLAMS library (e.g. [76–78]).

Further motivation for development of GSL came from the landscape of software for numerical analysis available in the early to mid-1990s—a motley collection of resources varying on several dimensions, such as code quality and terms of use [79]. We can sketch this landscape with reference to two of those resources, which sit at opposite ends of the spectrum for the dimensions most relevant to our discussion.

At one end of the spectrum is the SLATEC library—in full, the SLATEC Common Mathematical Subroutine Library, or some variant thereof [80–82]. The library was developed beginning in the late 1970s across several US government research laboratories: the 'SLA' in 'SLATEC' stands for three federal research facilities based in New Mexico (namely, Sandia National Laboratories, Los Alamos National Laboratory, and Air Force Weapons Laboratory); 'TEC' is an acronym for 'Technical Exchange Committee'. The committee had been established in 1974 as a loose consortium of the computing departments of those three sites, to foster the circulation of technical knowledge across them; it was later expanded to include other members, with others still joining specifically in the effort to develop the library [80,82].

The effort was driven by the aspiration to procure high-quality numerical software for use on the various systems, including supercomputers, in operation at the facilities involved in the collaboration [80]. It was framed at the outset as 'an experiment in resource sharing by the computing departments of several Department of Energy Laboratories', with the aim 'to cooperatively assemble and install at each site a mathematical subroutine library characterized by portability, good numerical technology, good documentation, robustness, and quality assurance' ([80], p. 16).

Multiple versions of the codebase were released between 1982 and 1993 [82], 'available to anyone on request [in the] hope that the library will enjoy usage by universities and laboratories engaged in scientific computation' ([81], p. 310). With the launch of Netlib [83–85] in 1985, specific packages of the library were additionally made available via the network. Short for 'network library', Netlib had been set up to provide 'quick, easy, and efficient distribution of public-domain software to the scientific computing community on an as-needed basis' ([83], p. 403). The aim was to facilitate the aggregation of relevant computer programs and other resources, thereby promoting re-use as best practice among researchers [84]. The material was initially collected and distributed via 'electronic mail' [83], and later predominantly via the World Wide Web [84].

At the other end of the spectrum is *Numerical recipes: the art of scientific computing*, a series of books published beginning in 1986 [86]. The books covered a broad range of algorithms and methods of scientific computation. They proved popular with researchers for the informal style of the prose, and for implementing the procedures discussed in the text as printed computer routines (e.g. [87]). For example, the first edition featured approximately 200 programs, implemented in both Fortran and Pascal ([86], p. 1). The 'package' also included example books [88,89] and diskettes, all available for separate purchase.

Inevitably, there was a trade-off between breadth and depth of the material covered in the books, both in thoroughness of the explanations and in sophistication of the implementations [90]. Indeed, experts in numerical analysis generally considered the *Numerical recipes* codebase poor in terms of efficacy, efficiency, and reliability (e.g. [90]).

Yet it became common practice among researchers to incorporate snippets of *Numerical recipes* code into software they developed. This practice was problematic, because releasing any such software is an infringement of copyright (§3b). For example, the licencing terms for the first edition of the book prohibit distribution of the machine-readable code, whether typed from the text or inputted from related products: the reader is allowed to make only one copy of each program, for personal use ([86], p. xiii).

(b) Conception and design of GSL

The SLATEC library and *Numerical recipes* both provided impetus to conception of GSL in the mid-1990s, although pulling in opposite directions. One was a library of high-quality software, in the public domain and accessible via the network, and aimed at specialists in numerical analysis. The other was a codebase of low quality, available under the restrictive terms of a commercial proprietary licence, and aimed at non-specialists. More broadly, the software in Netlib could handle difficult computational problems, which were out of reach of the small algorithms that researchers would incorporate as snippets of code from the *Numerical recipes* package [84]. Overall, then, these resources diverged both in scope and in target audience. Curiously, if for different reasons, they all raised concerns of a 'black-box' approach in the hands of users who had limited familiarity with the underlying mathematical theory, or with relevant numerical techniques [90].

For all its positive features, the SLATEC library presented one major shortcoming in the mid-1990s: it was written in Fortran. Fortran had been the programming language of choice in science and engineering since the late 1950s; beginning in the 1980s, however, it was increasingly replaced with C. Therefore, GSL was framed explicitly as 'a modern version of SLATEC' [79] (evidently, a reference to the SLATEC library, not the committee; §4a). Galassi and James Theiler began work on design and early implementation of GSL, developing the first modules, in 1996. The pair were soon joined by Brian Gough, who contributed to both aspects of the project. Once the initial elements were in place, it was possible to recruit several others to join in the effort. In particular, Gough went on to co-lead with Gerard Jungman on overall development of the library, including design and implementation of the major modules [10]. All four were based at Los Alamos National Laboratory: Galassi, Theiler, and Jungman as research scientists, Gough working on development and modernization of the web interface to the facility's preprint server (the predecessor to arXiv.org). Multiple versions of the library were released beginning in 1996, with the most recent stable release put out in 2019 [10].

A crucial initial step in development of GSL was to outline an explicit framework in a design document, which was refined over several years [79]. The document began by identifying the need for a numerical library of high quality that was free software (§3ai). The library would include state-of-the-art algorithms, which would be described pedagogically in comprehensive documentation. The code would be written in C, using up-to-date coding conventions and standards, and using a build and release system that would ensure portability and robust configuration on different platforms. Overall, the library would be aimed at general users, not specialists. At the same time, it would provide a framework to which experts in numerical analysis would be able to contribute.

These principles were conceived to build on the positive features of related resources available at the time, while addressing various shortcomings, as outlined above. In all other respects, the general approach to the design of GSL was derived from established practices of the free software movement, and of the GNU Project in particular (§3cii)—including release of the library under the GPL (§3bii), adoption of the GNU coding standards [91], and use of the GNU Autotools [92] to manage the complexities of porting software to different platforms [93]. As stated in the design

document, 'basically, [the library] is GNUlitically correct' [79]. The GSL team never lost sight of these principles, even as Galassi moved on from the role of maintainer, succeeded first by Gough, and then by Patrick Alken, a research scientist at the University of Colorado Boulder.

(c) Lessons learnt

We can now review the outlook for the resources introduced in §§4a and b, separately for different categories of software: software in the public domain (§4ci), proprietary software distributed commercially (§4cii), and free software distributed non-commercially (§4ciii).

(i) Public-domain software

Work authored as part of the duties of officers or employees of the US government is not covered by copyright, therefore it is automatically in the public domain (§3bi). At various times, however, federal research laboratories have been run as 'government-owned, contractor-operated' facilities, with personnel employed by the contractor, not the government. Exploiting this technicality, administrators of those facilities have often 'experimented' with approaches to the commercialization of software, variously restricting use, modification, and distribution of computer programs developed in-house.

A misguided attempt at commercialization may explain why the maintainers of the CLAMS library dithered when, in the early 1990s, Galassi approached them about the source code (§4a). Whether through administrative shortsightedness, or as an unintended consequence of it, in the end the source code was not released, and the library was therefore never ported to modern operating systems. A key resource for scientific computing was thus lost through a bureaucracy that lacked a robust framework for licencing software.

In and of itself, demise of the CLAMS library represents a waste of the effort (intellectual, financial, administrative, etc.) that had enabled development and maintenance of the codebase over several years. There are also enduring repercussions for the research community more broadly, in that it is not possible to reproduce published results that relied on the library for calculations (§2).

Conception of the SLATEC library in the late 1970s predates such attempts at commercialization. Nearly 40 years since its initial release (in 1982, Version 1.0, written in Fortran 66), the library is available to researchers worldwide as public-domain software. The latest release (in 1993, Version 4.1, written in Fortran 77) features over 1400 general-purpose mathematical and statistical routines [82].

This outcome is not the fortuitous byproduct of an auspicious set of bureaucratic circumstances. As discussed in §4a, development of the SLATEC library was driven by awareness of the benefits that arise from the pooling of technical expertise across research facilities [80]. Accordingly, there was an explicit focus from the outset on 'free software'—both in terms of incorporating existing high-quality code into the library, and in terms of making the programs available to all, encouraging uptake of the library across the research community ([81], pp. 304–306, 310). For example, stringent standards were initially set for programming, documentation, error handling, and testing, but they were quickly reframed as recommendations, so as to facilitate integration into the codebase of 'free software' available from other sources ([81], pp. 304–306, 314). Additionally, new programs considered for inclusion in the codebase were required to be in the public domain, which was 'generally not a problem since most authors are proud of their work and would like their routines to be used widely' [82].

The absence of restrictions on distribution of the software enabled access via the network, practically as soon as the relevant technology allowed. As noted in §4a, in 1985 specific packages of the library were included in the initial release of Netlib, a repository that aggregated public-domain software for scientific computing, to encourage sharing across the research community [83–85]. The temporal overlap in development of the two resources is likely not causal, and it is likely no coincidence that both efforts involved scientists based at US government

research facilities—possibly reflecting a culture that promoted the (relatively) free circulation of knowledge, echoing the culture established in the early days of military funding for computing projects [94].

We saw in §3cii that free software and related concepts were not articulated in full until the late 1980s. In particular, copyleft licencing had not been introduced at the time the SLATEC library and Netlib were first released. Therefore, the researchers involved in development of the two resources had no other option to make the programs 'free software', but to place the source code in the public domain (§3bii). There are several advantages to this approach: anyone can obtain the programs at no cost, and anyone is allowed to distribute copies of the source code, with or without changes. It follows that anyone can make improvements and contribute them back to the community, by distributing modified versions of the programs as FOSS. However, there is nothing to prevent distribution of the modified versions as proprietary software instead.

Distributing proprietary derivatives amounts to 'free-riding' on collective effort ([34], p. 184). Allowing (enabling?) such an uncooperative outcome is the one drawback to the approach. In all other respects, the 'experiment' set out in establishment of the SLATEC library was a resounding success, likely bolstered by confluence with the creation of Netlib (§4a). Both resources remain relevant today. As of February 2021, Netlib reports over 1.24 billion cumulative requests to its repositories at the University of Tennessee and Oak Ridge National Laboratory [95], with over 17.76 million of those requests being accesses to the SLATEC library [96]. These figures relate to only one of four main Netlib servers, and there are several mirrors worldwide [97]. The email interface to Netlib continues to operate, more than 35 years on, alongside the web-based one [83,84].

(ii) Commercial proprietary software

There is a clear parallel between the spirit of collaboration that culminated in release of the SLATEC library in 1982—and, separately, in the launch of Netlib in 1985—and the culture of knowledge-sharing that animated the free software movement in the early 1980s (§3cii).

This sentiment was not universally shared across the research community, however. For example, it is striking that those various initiatives emerged at around the same time as publication of the first edition of *Numerical recipes*, in 1986 [86] (§4a). In particular, the approach to licencing taken by the authors of the book provides a jarring contrast: as noted in §4a, the commercial proprietary licence imposes stringent restrictions on use and repurpose of programs in the package, such that distributing software that includes *Numerical recipes* code is a violation of copyright.

Whether intentionally or inadvertently, the licencing terms hindered collaboration among researchers. By the time the second edition of the book appeared, in 1992 [98], this stance seemed untenable—irksome even, since the authors acknowledged public funding for related work. Ironically, they also acknowledged relying on various free software tools for composition of the book ([98], pp. xiii, xv).

Whatever gains (reputational, financial?) were obtained in the short to medium term, the approach to licencing eventually backfired, exposing an embarrassing lack of foresight on the part of the authors. High-quality free software for scientific computing had become increasingly available through the 1990s and the early to mid-2000s, and the *Numerical recipes* codebase was simply no match for it. The third edition of the book was published in 2007 [99]; by admission of the authors, printed code was now included in the text exclusively as a pedagogical tool ([99], p. xi). More generally, the package was relegated to a relatively minor role against its earlier popularity (§4a), occupying a narrow niche in the flourishing landscape of scientific computing.

Overall, then, the outlook for the *Numerical recipes* codebase is analogous to the outlook for the CLAMS library (§4ci): a waste of the disparate set of resources that had enabled the work, with repercussions for the research community at large in terms of reproducibility (§2).

(iii) Non-commercial free software

We can only speculate what alternative fate the *Numerical recipes* package would have enjoyed, had its authors not restricted use of the programs, nor prohibited their distribution—perhaps even inviting users, and the broader community, to improve the software collaboratively.

To this end, we can draw a comparison with GSL, a project that began between publication of the second and third editions of the book. The library is in active development 25 years since its initial release (in 1996, Version 0.0). The most recent stable release (in 2019, Version 2.6) includes over 1000 mathematical routines, together with an extensive test suite [10]. Current efforts focus on maintaining the stability of the codebase, on extending or improving existing functionality, and on introducing new capabilities by incorporating useful algorithms that have been independently tested and thoroughly documented. Wrappers for the library are available from C to several high-level programming languages. Finally, GSL is readily installed from source code, and it is ported to different platforms efficiently and robustly—including the major GNU/Linux packaging approaches (Debian, Red Hat) and proprietary operating systems (macOS, Windows).

Where possible, the routines included in GSL were based on public-domain software of high quality, reimplemented in C using up-to-date coding conventions and standards (e.g. name canonicalization); others were instead written from scratch, following the same approach in terms of implementation [10]. The stability over time of C coding conventions and standards has lent robustness to GSL. Additional robustness has grown out of the explicit design decision to adopt rigorous engineering practices linked to the GNU Project (§4b). For example, use of the GNU Autotools [92] as the build system ensures longevity alongside portability. This suite of tools generates source code distributions that can be run virtually unchanged decades after they are released, through the rigid application of standards (e.g. use of the POSIX shell instead of GNU Bash for shell scripts), and through the assumption of minimal features on the target system. As a result, it is still possible to build the earliest versions of GSL, 25 years on, with almost no changes [93].

Another example of best practice derived from the GNU Project is inclusion in GSL of a test suite for all routines—for instance, a program that uses the library to check the output of a routine against known results, or one that invokes the library several times and performs a statistical analysis on the results (e.g. for random number generators). The test suite is run automatically as part of the building of the library [10]. Such complete testing was a new feature in the late 1990s, preceding widespread uptake of the practice in industry and, eventually, in academic research.

As discussed in §4b, GSL was conceived as a modern replacement to the SLATEC library. It was also intended as an alternative to a host of other resources for numerical analysis available in the 1990s, which included both free and proprietary software, and both commercial and non-commercial packages [10]. The advantages to using a tool like GSL that is free software, distributed non-commercially, are as follows. First, the library is available to everyone, at no cost, and therefore without conditions on in-house use (e.g. a limit on the number of users, which commonly applies to commercial products). Second, availability of the source code means that users can customize the routines in GSL to their needs. Third, users are allowed to release, as source code, any software they develop that is derivative of GSL—including software that makes use of the library, and software that incorporates a non-trivial amount of code from it. Overall, then, GSL enables and promotes scientific collaboration [10].

One restriction that arises from licencing of GSL under the GPL is that the library can only be redistributed in software that is itself under the GPL (§3bii). For example, users are allowed to combine GSL code with a program under a different licence, if the licence is compatible with the GPL. They are also allowed to distribute the combined program, including the source code, provided that the program is released under the GPL. It follows that all derivatives of GSL grant users the same rights that are granted by GSL [10]. By design, then, the freedom built into GSL by way of the GPL begets more freedom (§3ai), in the sense that it gets passed on to any software that relates to the library, in perpetuity (§3bi). In practice, a positive side-effect is that improvements to GSL are automatically contributed back to the community, to the extent that modified versions

of routines in the library can only be distributed as FOSS. Therefore, copyleft licencing averts the free-riding on collective effort enabled by FOSS that allows proprietary derivatives (§4ci)—namely, FOSS in the public domain, and FOSS under a non-copyleft licence (§3bii).

The restriction on distribution imposed by the GPL effectively serves to prevent the imposition of further restrictions (§3bii), and it applies equally to all users, whether they distribute software commercially or non-commercially (§3aii). In the case of GSL, it is the 'price' for access to a high-quality numerical library at no cost ([8], pp. 1–2). Opinions differ as to whether such a restriction should extend to software developed in the context of research. For example, a recent recommendation is to avoid the GPL in scientific computing, and to use a permissive licence instead (i.e. a non-copyleft licence; §3bii). The rationale is that permissively-licenced FOSS is more easily integrated into other projects [100]. Another view is that where the overarching aim is knowledge production and dissemination, as in the context of research (§2), then convenience should give way to principle — in this case, the principle of software freedom ([101], p. 235).

In setting up GSL, Galassi and colleagues were able to draw on the licencing philosophy that had emerged from the free software movement in the late 1980s (§3cii). As discussed in §4ci, such a carefully thought-out approach to the distribution of software had not been available to the researchers involved in establishment of the SLATEC library in the late 1970s, and in the creation of Netlib in the early 1980s. In many other respects, the parallels between the SLATEC library and Netlib outlined in §4ci extend to GSL. GSL also sought to prevent duplication of effort, both by re-using existing programs, and by enabling the research community to share software—not reinventing the wheel, so to speak, being a recurrent theme in the free software tradition (§3cii). And, as had been the case for the SLATEC library and for Netlib, development of GSL was driven by scientists based at US government research facilities (§4b). This observation raises questions about the incentives that motivated those involved in the three projects, compared to the incentives that may prevail in other contexts. For example, contemporary university settings have long suffered from an emphasis on 'publish or perish' [13], which leads to framing the research process as a 'winner-takes-all' activity; in turn, such framing rewards short-termism and performative novelty, at the expense of research quality and sustainability.

5. Closing

We conclude with two remarks, to illustrate the relevance of the preceding discussion to research and scholarship today (§2). The first remark is aimed specifically at researchers. To this end, we return to the common interpretation of 'open source' as 'open code' outlined in §1, and in particular to the scenario relating to software hosted in an online repository that is publicly accessible. It should be clear from the material in §§3a and b why, as noted in §1, public access to the source code is neither necessary nor sufficient for a computer program to qualify as FOSS.

We can make this scenario more concrete with reference to software hosted on https://github.com, a platform popular among researchers. The website is run by GitHub, Inc., a subsidiary of Microsoft Corporation. The GitHub platform provides online services for software development, and specifically distributed version control and source code management using Git [102]. Git is FOSS under the GPL (§3bii); it was created by Torvalds and others for development of the Linux kernel (§3ciii). Additional features on the platform, including its 'social networking' functions and other tools for collaborative development, are based on a proprietary web infrastructure—that is, the web interface itself is not FOSS.

In the context of open science, there is growing pressure on researchers to share programs they develop (§1). A common recommendation is to host the code on the GitHub platform (e.g. [100,103,104]). As noted in §1, simply hosting source code in a publicly accessible repository on https://github.com does not amount to 'open sourcing' a piece of software (e.g. [6]): the software is free and open source only if it is released under an appropriate licence (alternatively, the source code must be in the public domain; §3bii).

And yet, in our experience, this approach to 'open sourcing' software is common among researchers. For example, a convenient way to share code with others is to upload it to a public

repository on GitHub. Crucially, all too often there is no licence (§3biii). Without a licence, nobody has permission to use, modify, or distribute the code in the repository [105]. If multiple collaborators contributed to development of the code, then 'nobody' includes each one of them. Under the GitHub terms of service, other registered users on the platform can (i) view code hosted in a publicly accessible repository, and (ii) make a copy of the repository into their own account (via GitHub's 'fork' function). Technically, however, they are not allowed to use, modify, or distribute the code—doing so is a violation of copyright.

Clearly, researchers who share code in this way do not intend to disallow its use and repurpose. Rather, they typically aim to facilitate collaboration, enabling others to reproduce their work and to build on it—precisely the reasons the code should be released as FOSS in the first place! To be clear, we are not dwelling on this example as a criticism to well-intentioned researchers. Instead, we wish to highlight that it is important for anyone who contributes to the research process to acquire a working knowledge of FOSS, so that they can make informed decisions about software as an integral part of their day-to-day workflow (§2). We hope that the primer in §3 proves useful in this regard. Increasing awareness seems key, given that between 2008 and 2015 only around 20 to 30% of all repositories hosted on the GitHub platform included a licence [106]. These figures are all the more striking considering that, since 2013, the web interface has actively encouraged users to specify a licence when creating a new repository [107]: the initialization menu includes a 'Choose a license' checkbox, which brings up a 'license picker' with a list of commonly used options (the drop-down menu defaults to 'License: None').

The lessons learnt from the case study in §4 underscore that lack of clarity in licencing leads to lost opportunity and wasted effort: the demise of the CLAMS library is a clear example (§4ci). In the long run, the restrictions imposed by proprietary licencing may lead to an analogous outcome, as demonstrated by the downward trajectory of the *Numerical recipes* package (§4cii). Both lessons echo the Unix story (§3c). The long-term prospects of the operating system that 'changed the entire path of computer technology' ([60], p. ix) were thwarted by the lack of a clearly defined approach to licencing: the 'liberal' distribution of source code in the early years (e.g. to educational licencees) clashed with later attempts to restrict circulation for commercial gain ([60], p. 157) (§3ci). The legacy of Unix is substantial; sadly, the system itself succumbed to the confusion and 'legal wrangling' ([60], p. 157) that ensued from the clash (§§3cii and iii).

A related lesson to emerge from the case study is that a careful approach to licencing is key to the resilience of software projects and, ultimately, to research quality and sustainability. In particular, development of the FOSS resources discussed in §4 involved technical and organizational challenges linked to creating, maintaining, and distributing large and complex codebases, and to managing projects that rely heavily on feedback between developers and researchers. Arguably, the success of any such effort is best assessed in terms of longevity and widespread uptake, rather than by popularity of a piece of software over a short period of time. Decades on, the SLATEC library, Netlib, and GSL are still robust, relevant, and widely used (§§4ci and iii)—evidence that upfront investment in FOSS can lead to substantial long-term dividends in terms of research quality and sustainability (§2).

Building on this lesson, the second concluding remark is aimed more broadly at researchers, support staff, administrators, publishers, funders, and anyone else in the research community with an interest in open scholarship (§2). The implication here is that the community as a whole stands to benefit from a 'FOSS-first' approach to both the research process itself and the surrounding infrastructure. For example, we can ask whether well-intentioned researchers based at a university should indeed rely on services provided by a company like GitHub, Inc. in their day-to-day work. In our experience, many do so out of convenience, because equivalent FOSS infrastructure is not available through the university.

As noted in §2, related discussions are often framed in terms of the convenience of proprietary products versus the cost-effectiveness of FOSS solutions. The principle of software freedom is typically ignored, and with it the issue of control over the digital tools and services that enable the research process. To the extent that FOSS is itself a contribution to human knowledge ([101], p. 234), reframing the discourse in terms of principle, based on an ethical perspective, can bolster

other 'open' initiatives in research and scholarship, in support of the overarching aim: knowledge production and dissemination.

Data accessibility. This article has no additional data.

Authors' contributions. Both authors contributed to conceptualization and writing of the paper.

Competing interests. LF leads Reproducible Research Oxford (RROx; https://ox.ukrn.org/) and she is a member of the Steering Group of the UK Reproducibility Network (UKRN; https://www.ukrn.org/). She also sits on the Board of Directors of the Software Freedom Conservancy (https://sfconservancy.org/), which is chaired by MG. These are all volunteer efforts.

Funding. No funding has been received for this article.

Acknowledgements. We thank Jeremy Allison, Jim Blandy, and Tom Tromey for discussion, and Malika Ihle, Adam Kenny, and Rowan Wilson for feedback on the manuscript. We are also grateful to those who took part in discussions hosted in September 2020 by Reproducible Research Oxford and by `anthrologue`, and to the reviewers for providing positive, constructive comments.

References

1. Stallman R. 1999 The GNU operating system and the free software movement. In *Open sources: voices from the open source revolution* (eds C DiBona, S Ockman, M Stone), pp. 53–70. Sebastopol, CA: O'Reilly Media, Inc.
2. Ritchie DM. 1996 Foreword. In *Lions' commentary on UNIX 6th edition with source code*, p. x. Charlottesville, VA: Peer-to-Peer Communications LLC.
3. Stodden V, Guo P, Ma Z. 2013 Toward reproducible computational research: an empirical analysis of data and code policy adoption by journals. *PLoS ONE* **8**, e67111. (doi:10.1371/journal.pone.0067111)
4. Stodden V, Seiler J, Ma Z. 2018 An empirical analysis of journal policy effectiveness for computational reproducibility. *Proc. Natl Acad. Sci. USA* **115**, 2584–2589. (doi:10.1073/pnas.1708290115)
5. Culina A, van den Berg I, Evans S, Sánchez-Tójar A. 2020 Low availability of code in ecology: a call for urgent action. *PLoS Biol.* **18**, e3000763. (doi:10.1371/journal.pbio.3000763)
6. Jiménez R *et al.* 2017 Four simple recommendations to encourage best practices in research software [version 1; peer review: 3 approved]. *F1000Research* **6**, 1–8 (doi:10.12688/f1000research.11407.1)
7. Munafò MR, Chambers CD, Collins AM, Fortunato L, Macleod MR. 2020 Research culture and reproducibility. *Trends Cogn. Sci.* **24**, 91–93. (doi:10.1016/j.tics.2019.12.002)
8. Galassi M, Davies J, Theiler J, Gough B, Jungman G, Alken P, Booth M, Rossi F. 2009 *GNU Scientific Library reference manual*, 3rd edn, for version 1.12. Bristol: Network Theory Ltd.
9. Galassi M, Davies J, Theiler J, Gough B, Jungman G, Alken P, Booth M, Rossi F, Ulerich R. 2019 *GNU Scientific Library release 2.6*. Aug. 21. https://www.gnu.org/software/gsl/doc/latex/gsl-ref.pdf (visited on 07/26/2020).
10. GNU Operating System. *GSL - GNU Scientific Library*. https://www.gnu.org/software/gsl/ (visited on 02/28/2021).
11. National Academies of Sciences, Engineering, and Medicine. 2019 *Reproducibility and replicability in science*. Washington, DC: The National Academies Press.
12. Willinsky J. 2006 *The access principle: the case for open access to research and scholarship*. Cambridge, MA: The MIT Press.
13. Willinsky J. 2005 The unacknowledged convergence of open source, open access, and open science. *First Monday* **10**. (doi:10.5210/fm.v10i8.1265).
14. Christensen G, Freese J, Miguel E. 2019 *Transparent and reproducible social science research: how to do open science*. Oakland, CA: University of California Press.
15. Healy K. 2019 *The plain person's guide to plain text social science*. Oct. 4. https://plain-text.co/ (visited on 07/04/2020).
16. Yalta AT, Lucchetti R. 2008 The GNU/Linux platform and freedom respecting software for economists. *J. Appl. Econometrics* **23**, 279–286. (doi:10.1002/jae.990)
17. Yalta AT, Yalta AY. 2010 Should economists use open source software for doing research? *Comput. Econ.* **35**, 371–394. (doi:10.1007/s10614-010-9204-4)

18. Stallman RM. 2015 Why open source misses the point of free software. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 14, pp. 75–82. This essay was originally published on http://gnu.org, in 2007. Boston, MA: GNU Press.
19. Free Software Foundation. https://www.fsf.org (visited on 08/11/2020).
20. Open Source Initiative. https://opensource.org (visited on 08/11/2020).
21. GNU Operating System. *The GNU Manifesto*. https://www.gnu.org/gnu/manifesto.html (visited on 08/01/2020).
22. Fogel K. 2017 *Producing open source software: how to run a successful free software project*, 2nd edn, version 2.3168. Originally published in 2005 by O'Reilly Media, Inc. Sebastopol, CA: O'Reilly Media, Inc. https://producingoss.com/en/producingoss-a4.pdf.
23. GNU Operating System. *What is free software?* https://www.gnu.org/philosophy/free-sw.html (visited on 08/11/2020).
24. Open Source Initiative. *The Open Source Definition*. https://opensource.org/osd (visited on 08/11/2020).
25. Stallman RM. 2015 What is free software? In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 1, pp. 3–8. The free software definition was first published in 1996, on http://gnu.org. Boston, MA: GNU Press.
26. Perens B. 1999 The Open Source Definition. In *Open sources: voices from the open source revolution* (eds C DiBona, S Ockman, M Stone), pp. 171–188. Sebastopol, CA: O'Reilly Media, Inc.
27. Stallman RM. 2015 Selling free software. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 8, pp. 43–45. This essay was originally published on http://gnu.org, in 1996. Boston, MA: GNU Press.
28. Stallman RM. 2015 Categories of free and nonfree software. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 13, pp. 68–74. This list was originally published on http://gnu.org, in 1996. Boston, MA: GNU Press.
29. Stallman RM. 2015 Words to avoid (or use with care) because they are loaded or confusing. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 17, pp. 89–103. This list was first published on http://gnu.org, in 1996. Boston, MA: GNU Press.
30. St. Laurent AM. 2004 *Understanding open source and free software licensing*. Sebastopol, CA: O'Reilly Media, Inc.
31. Rosen L. 2005 *Open source licensing: software freedom and intellectual property law*. Upper Saddle River, NJ: Prentice Hall PTR.
32. Brasseur VM. 2018 *Forge your future with open source*. Raleigh, NC: The Pragmatic Bookshelf.
33. Morin A, Urban J, Sliz P. 2012 A quick guide to software licensing for the scientist-programmer. *PLoS Comput. Biol.* **8**, e1002598. (doi:10.1371/journal.pcbi.1002598)
34. Stallman RM. 2015 What is copyleft? In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 29, pp. 184–186. This essay was originally published on http://gnu.org, in 1996. Boston, MA: GNU Press.
35. GNU Operating System. *GNU General Public License*. https://www.gnu.org/licenses/gpl-3.0.html (visited on 08/11/2020).
36. GNU Operating System. *GNU Lesser General Public License*. https://www.gnu.org/licenses/lgpl-3.0.html (visited on 08/11/2020).
37. Mozilla. *Mozilla Public License*. https://www.mozilla.org/en-US/MPL/ (visited on 08/11/2020).
38. The Apache Software Foundation. *Apache License*. https://apache.org/licenses/LICENSE-2.0 (visited on 08/11/2020).
39. Open Source Initiative. *The 2-Clause BSD License*. https://opensource.org/licenses/BSD-2-Clause (visited on 08/11/2020).
40. GNU Operating System. *Various licenses and comments about them*. https://www.gnu.org/licenses/license-list.en.html#FreeBSD (visited on 08/11/2020).
41. Open Source Initiative. *The MIT License*. https://opensource.org/licenses/MIT (visited on 08/11/2020).
42. GNU Operating System. *Various licenses and comments about them*. https://www.gnu.org/licenses/license-list.en.html#Expat (visited on 08/11/2020).
43. GNU Operating System. *Various licenses and comments about them*. https://www.gnu.org/licenses/license-list.en.html (visited on 08/11/2020).

44. Open Source Initiative. *Licenses & Standards*. https://opensource.org/licenses (visited on 08/11/2020).
45. Debian. *License information*. https://www.debian.org/legal/licenses/index.en.html (visited on 08/11/2020).
46. Callaway T. 2020 *Licensing:Main*. Aug. 19 https://fedoraproject.org/wiki/Licensing:Main#Software_License_List (visited on 02/28/2021).
47. SPDX. *SPDX License List*. url: https://spdx.org/licenses/ (visited on 08/11/2020).
48. GNU Operating System. *Various licenses and comments about them*. https://www.gnu.org/licenses/license-list.en.html#PublicDomain (visited on 08/11/2020).
49. Open Source Initiative. *Frequently Answered Questions*. https://opensource.org/faq#public-domain (visited on 08/11/2020).
50. Creative Commons. *CC0 1.0 Universal (CC0 1.0) Public Domain Dedication*. https://creativecommons.org/publicdomain/zero/1.0/ (visited on 08/11/2020).
51. GNU Operating System. *Various licenses and comments about them*. https://www.gnu.org/licenses/license-list.en.html#CC0 (visited on 08/11/2020).
52. Open Source Initiative. *Frequently Answered Questions*. https://opensource.org/faq#cc-zero (visited on 08/11/2020).
53. Salus PH. 1994 *A quarter century of UNIX*. Reading, MA: Addison-Wesley Publishing Company.
54. Raymond ES. 2001 *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*, revised edn. Originally published in 1999 by O'Reilly Media, Inc. Sebastopol, CA: O'Reilly Media, Inc.
55. Moody G. 2002 *Rebel code: the inside story of Linux and the open source revolution*. Originally published in 2001 by Perseus Publishing; issued with a new afterword by the author. New York, NY: Basic Books.
56. Williams S. 2002 *Free as in freedom: Richard Stallman's crusade for free software*. Sebastopol, CA: O'Reilly & Associates, Inc.
57. Ceruzzi PE. 2003 *A history of modern computing*, 2nd edn. Cambridge, MA: The MIT Press.
58. Levy S. 2010 *Hackers: heroes of the computer revolution*, 25th anniversary edn. Originally published in 1984 by Anchor Press/Doubleday. Sebastopol, CA: O'Reilly Media, Inc.
59. Coleman EG. 2013 *Coding freedom: the ethics and aesthetics of hacking*. Princeton, NJ: Princeton University Press.
60. Kernighan BW. 2020 *Unix: a history and a memoir*. Kindle Direct Publishing.
61. DiBona C, Ockman S, Stone M, eds. 1999 *Open sources: voices from the open source revolution*. Sebastopol, CA: O'Reilly Media, Inc.
62. Ritchie DM, Thompson K. 1974 The UNIX time-sharing system. *Commun. ACM* **17**, 365–375. (doi:10.1145/361011.361061)
63. McKusick MK. 1999 Twenty years of Berkeley Unix: from AT&T-owned to freely redistributable. In *Open sources: voices from the open source revolution* (eds C DiBona, S Ockman, M Stone), pp. 31–46. Sebastopol, CA: O'Reilly Media, Inc.
64. Lions J. 1996 *Lions' commentary on UNIX 6th edition with source code*. Charlottesville, VA: Peer-to-Peer Communications LLC.
65. Stallman RM. 2015 The initial announcement of the GNU operating system. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 3, pp. 26–27. Boston, MA: GNU Press.
66. GNU Operating System. https://www.gnu.org (visited on 08/11/2020).
67. Stallman R. 1985 The GNU Manifesto. In *Dr. Dobb's Journal of Software Tools*, **10**, 30.
68. Stallman RM. 1986 What is the Free Software Foundation? *GNU'S BULLETIN* **1**, 8–9.
69. Torvalds L. 1999 The Linux edge. In *Open sources: voices from the open source revolution* (eds C DiBona, S Ockman, M Stone), pp. 101–111. Sebastopol, CA: O'Reilly Media, Inc.
70. Stallman RM. 2015 What's in a name? In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 11, pp. 61–63. This essay was originally published on http://gnu.org, in 2000. Boston, MA: GNU Press.
71. Stallman RM. 2015 Linux and the GNU system. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 12, pp. 64–67. This essay was originally published on http://gnu.org, in 1997. Boston, MA: GNU Press.

72. Linksvayer M. 1993 *The choice of a GNU generation: an interview with Linus Torvalds*. Originally published late 1993 in Meta Magazine. https://gondwanaland.com/meta/history/interview.html (visited on 10/19/2020).
73. Peterson C. 2018 *How I coined the term 'open source'*. Feb. 1. https://opensource.com/article/18/2/coining-term-open-source-software (visited on 06/15/2020).
74. Open Source Initiative. *History of the OSI* https://opensource.org/history (visited on 08/11/2020).
75. *BITS: Computing & Communications News.* 1999 Los Alamos, NM: Computing, Information, and Communications (CIC) Division, Los Alamos National Laboratory. May. ftp://hpc-ftp.lanl.gov/history/bits/1999/1999-annual-LALP-99-42.pdf.
76. Wienke BR. 1985 Relativistic photon-Maxwellian electron cross sections. *Astron. Astrophys.* **152**, 336–342.
77. Wienke BR, Lathrop BL, Devaney JJ. 1986 Relativistic photon-Maxwellian electron cross sections. *Radiat. Eff.* **94**, 303–306. (doi:10.1080/00337578608208397)
78. Wienke BR. 2009 On validation of a popular sport diving decompression model. *Open Sports Sci. J.* **2**, 76–93. (doi:10.2174/1875399X00902010076)
79. Galassi M, Theiler J, Gough B. *GNU Scientific Library–Design document*. https://www.gnu.org/software/gsl/design/gsl-design.html (visited on 07/06/2020).
80. Vandevender WH, Haskell KH. 1982 The SLATEC Mathematical Subroutine Library. *ACM SIGNUM Newsletter* **17**, 16–21. (doi:10.1145/1057594.1057595)
81. Buzbee BL. 1984 The SLATEC Common Mathematical Library. In *Sources and development of mathematical software* ed. WR Cowell, ch. 11, pp. 302–320. Englewood Cliffs, NJ: Prentice Hall, Inc.
82. Fong KW, Jefferson TH, Suyehiro T, Walton L. 1993 *Guide to the SLATEC Common Mathematical Library*. July. https://www.netlib.org/slatec/guide (visited on 07/04/2020).
83. Dongarra JJ, Grosse E. 1987 Distribution of mathematical software via electronic mail. *Commun. ACM* **30**, 403–407. (doi:10.1145/22899.22904)
84. Dongarra J, Golub GH, Grosse E, Moler C, Moore K. 2008 Netlib and NA-Net: building a scientific computing community. *IEEE Ann. Hist. Comput.* **30**, 30–41. (doi:10.1109/MAHC.2008.29)
85. Netlib. *Netlib Repository at UTK and ORNL*. https://www.netlib.org/ (visited on 02/14/2021).
86. Press WH, Flannery BP, Teukolsky SA, Vetterling WT. 1986 *Numerical recipes: the art of scientific computing*. Cambridge, UK: Cambridge University Press.
87. Frolkovič P. 1990 William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling: *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, New York, New Rochelle, Melbourne, Sydney, 1986, 20 + 818 pp. *Acta Appl. Math.* **19**, 297–290. (doi:10.1007/BF01321860)
88. Vetterling WT, Teukolsky SA, Press WH, Flannery BP. 1985 *Numerical recipes example book (FORTRAN)*. Cambridge, UK: Cambridge University Press.
89. Vetterling WT, Teukolsky SA, Press WH, Flannery BP. 1985 *Numerical recipes example book (Pascal)*. Cambridge, UK: Cambridge University Press.
90. Shampine LF. 1987 *Numerical Recipes, The Art of Scientific Computing*. By W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. V. Vetterling. Cambridge University Press, 1986. xi + 817 pp. *Am. Math. Mon.* **94**, 889–892. (doi:10.1080/00029890.1987.12000737)
91. GNU Operating System. *GNU coding standards*. June 21, 2020. http://www.gnu.org/prep/standards/ (visited on 08/11/2020).
92. Vaughan GV, Elliston B, Tromey T, Taylor IL. 2000 *GNU Autoconf, Automake, and Libtool: expert insight into porting software and building large projects using GNU Autotools*. Indianapolis, IN: New Riders Publishing.
93. Galassi M. 2018 *A self-referential HOWTO on release engineering*. Tech. rep. LA-UR-14-21151. Los Alamos, NM: Los Alamos National Laboratory.
94. Nofre D, Priestley M, Alberts G. 2014 When technology became language: the origins of the linguistic conception of computer programming, 1950–1960. *Technol. Cult.* **55**, 40–75. (doi:10.1353/tech.2014.0031)
95. Netlib. *Netlib Statistics at UTK/ORNL*. https://www.netlib.org/utk/misc/counts.html (visited on 02/14/2021).

96. Netlib. *Breakdown of requests to each Netlib library*. http://www.netlib.org/master_counts2.html#slatec (visited on 02/14/2021).
97. Netlib. http://www.netlib.org/bib/mirrors.html (visited on 02/14/2021).
98. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. 1992 *Numerical recipes in C: the art of scientific computing*, 2nd edn. Cambridge, UK: Cambridge University Press.
99. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. 2007 *Numerical recipes: the art of scientific computing*, 3rd edn. Cambridge, UK: Cambridge University Press.
100. Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK. 2017 Good enough practices in scientific computing. *PLoS Comput. Biol.* **13**, e1005510. (doi:10.1371/journal.pcbi.1005510)
101. Stallman RM. 2015 Releasing free software if you work at a university. In *Free software, free society: selected essays of Richard M. Stallman*, 3rd edn, ch. 39, pp. 234–235. This essay was originally published on http://gnu.org, in 2002. Boston, MA: GNU Press.
102. Git. https://git-scm.com/ (visited on 08/11/2020).
103. Blischak JD, Davenport ER, Wilson G. 2016 A quick introduction to version control with Git and GitHub. *PLoS Comput. Biol.* **12**, e1004668. (doi:10.1371/journal.pcbi.1004668)
104. Perez-Riverol Y *et al.* 2016 Ten simple rules for taking advantage of Git and GitHub. *PLoS Comput. Biol.* **12**, e1004947. (doi:10.1371/journal.pcbi.1004947)
105. GitHub. *No License*. https://choosealicense.com/no-permission/ (visited on 08/11/2020).
106. Balter B. 2015 *Open source license usage on GitHub.com*. March 9. https://github.blog/2015-03-09-open-source-license-usage-on-github-com/ (visited on 10/19/2020).
107. Haack P. 2013 *Choosing an open source license*. July 15. https://github.blog/2013-07-15-choosing-an-open-source-license/ (visited on 10/19/2020).