

The Mythical Code-Authorship

by Ewa Laskowska-Litak *

Abstract: Modern copyright law emerged during the transition from the 18th to the 19th centuries, centred on the figure of the author as the originator of unique, creative works. Authors were seen as deserving legal protection, both because of their creative labour and their vulnerable position in publishing contracts. Over time, this framework has expanded to include various creative outputs, including musical works, films, and computer programs. However, the incorporation of software into copyright law occurred without sufficient understanding of its development process, leading to conceptual tensions.

This paper addresses two interrelated aims. First, it examines, from a primarily European Union perspective supplemented by illustrative United Kingdom authorities, whether changing perceptions of software—particularly its collaborative, iterative and functional nature—require a recalibration of copyright analysis. Second, it challenges two persistent misconceptions: (1) that software authorship is straightforward and legally unproblematic, and (2) that most source code qualifies as protectable under copyright. These assumptions, foundational to ex-

isting copyright regimes, fail to reflect the realities of programming and contribute to a growing disconnect between legal theory and software practice. The paper is divided into two major parts: the first critiques prevailing copyright assumptions using sociological and psychological insights into authorship and creativity, revealing the mismatch between the solitary-author model and the inherently collective, structured nature of programming. The second part proposes a revised framework that accounts for the collaborative and social character of code creation. Here, the main observation is that authorship is understood more technically, through programmatic attribution, rather than through traditional legal concepts.

In conclusion, the paper advocates for a more nuanced and interdisciplinary approach to copyright law—one that acknowledges the complexities of software development and better aligns legal doctrine with technical and social realities. This shift is crucial to addressing the growing misfit between copyright's foundational assumptions and the nature of modern software production.

Keywords: Authorship; code authorship; copyright law; collaborative authorship; software development

© 2026 Ewa Laskowska-Litak

Everybody may disseminate this article by electronic means and make it available for download under the terms and conditions of the Digital Peer Publishing Licence (DPPL). A copy of the license text may be obtained at <http://nbn-resolving.de/urn:nbn:de:0009-dppl-v3-en8>.

Recommended citation: Ewa Laskowska-Litak, The Mythical Code-Authorship, 17 (2026) JIPITEC 161 para 1.

A. Introduction

- 1 The genesis of copyright thought can be traced back to the transition from the 18th to the 19th centuries, a period that witnessed the establishment of foundational principles that persist to the present day: that it is the author, as the central figure of copyright law, who deserves protection because of their creative effort; that they are the weaker party to the contract and the legislator should protect them to a certain extent from exploitation by publishers; that the object of protection should be an original and unique work, carrying the genius of the creator,

distinctive from others.¹ In the course of time, only

* Ewa Laskowska-Litak, PhD habilit., LL.M. (Heidelberg); Faculty of Law and Administration, Future Law Lab, Jagiellonian University, Kraków, Poland
e-mail: ewa.laskowska@uj.edu.pl

Acknowledgments: This work was supported by the National Science Centre in Poland (Narodowe Centrum Nauki) under grant no. 2023/49/B/HS5/00772.

1 Martha Woodmansee, 'The Genius and the Copyright: Economic and Legal Conditions of the Emergence of the "Author"' (1984) 17 *Eighteenth-Century Studies* 425.

further protection instruments growing out of these foundations have been added. This evolution has encompassed the establishment of normative protection over author's personal interests, the protection of musical, and subsequently, also of cinematographic works, computer programs, and, to a certain extent, the protection of artistic performers. Nevertheless, the decision to extend the scope of copyright protection to computer programs, without conducting a thorough examination of the software development process, may have inadvertently changed the perspective of copyright law as such. The purpose of this paper is twofold: firstly, to challenge a hypothesis that a shift in the manner in which software is perceived may facilitate a concomitant change in perspective with regard to copyright law. And secondly, to address two misconceptions relating to software protection in copyright law:

(1) that software authorship constitutes a straightforward and uncomplicated case; and

(2) that the majority of source code is subject to copyright protection.

- 2 Both are foundational assumptions of all copyright instruments. Both also, in my view, not only fail to capture the specificity of computer programs, but also lead to a significant and progressive disconnect between programming pragmatics and legal evaluation.
- 3 In other words, the main aim of this research is to dispel the prevailing myths surrounding the copyright narrative concerning software, with a particular focus on these two gravitational points: authorship and copyright subject matter. In the first, mythological part, we will confront these assumptions in turn by addressing the issues of authorship and the protection of computer program code. Utilising examples drawn from sociological and psychological research, this paper aims to highlight the inaccuracies in the understanding of authorship and the dissonance that arises when confronted with the reality of programming and with doctrinal analysis of authorship, joint authorship and contractual allocation of rights. The second (demythologising) part will be devoted to an in-depth analysis of two elements that copyright law does not take into account, but which it should systemically include in this assessment: the collaborative element of code authorship and the sociality of programming. In the latter case, the focus is primarily on the technical understanding of authorship (as programmatic attribution, not about authorship in the legal sense). The primary postulate emphasises the necessity for enhanced caution in the formulation of legal authorship assessments. Additionally, it underscores the imperative for

a more interdisciplinary and community-based approach in copyright law, particularly in the context of assessing collaborative authorship. The concluding part then develops a more concrete reform agenda directed at legislation, adjudication and contracting practice.

- 4 This article proceeds primarily from the legal framework of the European Union, especially the Software Directive and the CJEU case law on originality and the scope of protection for computer programs,² while using United Kingdom doctrine on joint authorship as a comparative illustration where it sharpens the analysis. The point is not that one jurisdiction can simply be universalised, but that the EU/UK materials make especially visible the tension between copyright's inherited assumptions and contemporary software production.

B. Myth No. 1: Software Authorship Does Not Require Special Treatment

- 5 The status of author is generally attributed to the individual who created the work. The veracity of this assertion remains unassailable, irrespective of the legal system under consideration, be it common law³ or continental law.⁴ In principle, therefore, one can assert that the creator of a work can be considered the author, irrespective of geographical location, provided that the country in question has adopted the Berne Convention.⁵ Although the dogmatics of copyright law are, in general, very conservative on this aspect, the legal scholarship completes the system. The prevailing tendency is to conceptualise authorship as a romantic phenomenon⁶ (or not

2 Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs (Codified version) [2009] OJ L111/16.

3 Jane C Ginsburg, 'The Concept of Authorship in Comparative Copyright Law' (2003) 52 DePaul L Rev 1063.

4 Lionel Bently, 'Authorship of Popular Music in UK Copyright Law' (2009) 12 *Information, Communication & Society* 179; Lior Zemer, *The Idea of Authorship in Copyright* (Ashgate 2007); Mireille van Eechoud (ed), *The Work of Authorship* (Amsterdam University Press 2014).

5 Sam Ricketson, *The Berne Convention for the Protection of Literary and Artistic Works: 1886-1986* (Centre for Commercial Law Studies, Queen Mary College 1987); Berne Convention for the Protection of Literary and Artistic Works (adopted 9 September 1886, as revised at Paris 24 July 1971 and amended 28 September 1979) 1161 UNTS 3.

6 Megan Richardson and Julian Thomas, 'Rethinking "Romantic" Authorship' in *Fashioning Intellectual Property: Exhibition, Advertising and the Press, 1789-1918* (Cambridge University Press 2012) 103; Isabella Alexander, 'The Genius and the Labourer: Authorship in Eighteenth and

at all⁷). Another well-documented assumption on authorship is that the grant of copyright to the creator is of paramount importance, as it serves to safeguard the creator's work and creative genius,⁸ or that the author's character requires special protection because, as an independent yet inexperienced person in the trade, they are vulnerable to exploitation.⁹

- 6 Yet, even at this basic level, software forces a more careful doctrinal analysis than the orthodox narrative often suggests. In the EU framework, computer programs are protected as literary works, but only if they are original in the sense of being the author's own intellectual creation;¹⁰ the legal question is therefore never merely who contributed, but which contributions count as authorial expression and under what doctrinal test. At the same time, the law of joint authorship, employee creation and contractual allocation of rights shows that copyright has long had to mediate between creativity, collaboration and market organisation.
- 7 Current copyright frameworks do address collaborative creation, though unevenly. A useful illustration is the United Kingdom notion of a "work

of joint authorship", which requires collaboration and non-distinct contributions. In *Kogan v Martin* [2019] EWCA Civ 1645, the Court of Appeal restated the inquiry in a way that is particularly relevant for software disputes: the court must ask whether the putative co-author made an authorial contribution that formed part of the creation of the work, whether that contribution is quantitatively and qualitatively sufficient, and whether it is part of a collaborative process rather than merely advisory criticism or editing. The attraction of this approach is its flexibility; its weakness, for software, is that it still presupposes a relatively intelligible chain between contribution and expressive outcome. In iterative software projects that chain is often dispersed across tickets, pull requests, reviews, architecture choices and dependency management.

- 8 The situation becomes especially difficult where there is more than one person working on the work and where contractual or employment structures detach legal ownership from factual creation. Work-for-hire and employee-creation rules differ across jurisdictions, but their common function is to facilitate exploitation by reallocating rights away from the natural person who physically created the work.¹¹ Co-creation raises a different problem: not who will own the rights after creation, but which participants count as authors at all. That question has long been difficult in copyright law—in music, photography, film and social media alike—and software does not create the problem from nothing. What software does is intensify it by making cumulative, interdependent and partially invisible contributions the rule rather than the exception.
- 9 Whilst the problems presented by social media, music and other forms of collaborative authorship to the existing copyright paradigm have been well-discussed and researched for years,¹² it is contemporary software pragmatics, particularly those based on agile frameworks, that represent a particularly salient threat.¹³ For instance, one can agree with the assertion that a co-author is a person who exerts the most significant influence on the final form of the work.¹⁴ However, the extent to which

Nineteenth-Century Copyright Law' in Lionel Bently, Jennifer Davis and Jane C Ginsburg (eds), *Copyright and Piracy: An Interdisciplinary Critique* (Cambridge University Press 2010) 300; Margaret Chon, 'The Romantic Collective Author' (2012) 14 *Vand J Ent & Tech L* 829; Mark A Lemley, 'Romantic Authorship and the Rhetoric of Property' (1997) 75 *Tex L Rev* 873.

- 7 cf Michael J Madison, 'Beyond Creativity: Copyright as Knowledge Law' (2010) 12 *Vanderbilt Journal of Entertainment and Technology Law* 817; Annemarie Bridy, 'Coding Creativity: Copyright and the Artificially Intelligent Author' (2012) 2012 *Stan Tech L Rev* 5; Olufunmilayo B Arewa, 'From J.C. Bach to Hip Hop: Musical Borrowing, Copyright and Cultural Context' (2006) 84 *NC L Rev* 547; Elena Cooper, 'Copyright and Mass Social Authorship: A Case Study of the Making of the Oxford English Dictionary' (2015) 24 *Social & Legal Studies* 509.
- 8 Ole-Andreas Rognstad, *Property Aspects of Intellectual Property* (Cambridge University Press 2018); Carys J Craig, 'Locke, Labour and Limiting the Author's Right: A Warning against a Lockean Approach to Copyright Law' (2002) 28 *Queen's LJ* 1; Lior Zemer, 'Three Arguments on Locke, Authorship, Communication and Solitude' (2015) 7 *The WIPO Journal* 40.
- 9 Diane Leenheer Zimmerman, 'Copyrights as Incentives: Did We Just Imagine That?' (2011) 12 *Theoretical Inquiries in Law* 29; James Griffin, 'The Need for a New Paradigm in IP Law: A Focus on Authorship' (2005) 14 *Information & Communications Technology Law* 267; Ronan Deazley and Bartolomeo Meletti, 'Copying, Creativity and Copyright' (CREATE Working Paper, University of Glasgow, February 2016).
- 10 Directive 2009/24/EC (n 2) art 1(3); Case C-393/09 *Bezpečnostní softwarová asociace – Svaz softwarové ochrany v Ministerstvo kultury* ECLI:EU:C:2010:816.

- 11 Catherine L Fisk, 'Authors at Work: The Origins of the Work-for-Hire Doctrine' (2003) 15 *Yale Journal of Law & the Humanities* 1.
- 12 Peter Jaszi, 'Toward a Theory of Copyright: The Metamorphoses of "Authorship"' (1991) 40 *Duke LJ* 455; Jason Borenstein and Adil E Shamoo, 'Rethinking Authorship in the Era of Collaborative Research' (2015) 22 *Accountability in Research* 267.
- 13 Philippe Kruchten, 'Contextualizing Agile Software Development' (2013) 25 *Journal of Software: Evolution and Process* 351; Alistair Cockburn, *Agile Software Development: The Cooperative Game* (2nd edn, Addison-Wesley 2006).
- 14 Peter Jaszi, 'On the Author Effect: Contemporary Copyright

this (causal?) influence can justify the co-authorship status is still not yet determined. Since we have not so far extended co-author status to the producer of a film, can a scrum master be a co-author of software? After all, their role is to facilitate the success of the team, to ensure effective communication within the team, or to remove impediments that hinder the team's progress. Their role, although not directly related to the writing of code, facilitates the same act:

- 10 the Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team¹⁵.
- 11 The aforementioned concerns are often managed in practice through contracts rather than through fine-grained judicial inquiry into authorship. In software projects, common contractual devices include employment clauses assigning economic rights, commissioned-development agreements, contributor licence agreements, open-source licence terms and repository governance rules. These instruments reduce transactional uncertainty and enable exploitation, but they do not eliminate the underlying doctrinal question of authorship, because author status is generally a matter of fact and law rather than pure contract.¹⁶ Nor do they fully solve evidentiary difficulties. IT contracts are typically multi-layered and multi-party, regulating scope, methodology, security, risk allocation, change management, acceptance criteria and delivery structures.¹⁷ In that environment, the practical success of contracting can mask the conceptual instability of authorship doctrine: the project proceeds, rights are allocated for business purposes, yet the legal map of creative contribution remains obscure unless a dispute forces the issue.
- 12 Despite the fact that the role of figures such as scrum masters and product owners does not directly pertain to the creation of software code, it is challenging to refute their, at times even direct, influence on the eventual functionality and appearance of the software. In their study, Shastri et al.¹⁸ examined the impact of the presence of scrum masters in IT teams, highlighting a positive association between specific agile practices and the effectiveness of the scrum master (definition of done, iteration (sprint) planning, project velocity measurement, and short iterations (sprints)).¹⁹ In particular, the ability to mediate (negotiate) between client expectations and the team's capabilities seems to have a crucial role in the context of copyright, as it always translates into the final shape of the work—a requirement for the copyright protection to subsist. The coordination of collaborative efforts between teams from diverse organisational domains, the safeguarding of teams against external interferences, and the rebuff of scope creep can also be of equal importance.²⁰
- 13 Nevertheless, even if the causal link between the role of the scrum master and software production is considered to be too speculative, the exclusive focus on software engineers alone generates further uncertainty. The GitHub platform is recognised as one of the most prevalent tools for software development.²¹ The platform itself offers a range of options for storing computer program code in dedicated repositories (generally either commercial-private or public). Research conducted on the methods of software development within this platform has yielded several noteworthy insights in this regard. Firstly, a small proportion of public repositories (i.e. those created by a single individual for personal, scientific or development purposes) are single-author: only one quarter of all developers

and Collective Creativity' (1992) 10 *Cardozo Arts & Ent LJ* 293.

- 15 Ken Schwaber and Jeff Sutherland, *The Scrum Guide: The Definitive Guide to Scrum — The Rules of the Game* (Scrum.org and Scrum Inc 2016).
- 16 van Eechoud, *The Work of Authorship* (n 4); Bently (n 4); Arewa (n 7).
- 17 Michael J Gallivan and Wonseok Oh, 'Analyzing IT Outsourcing Relationships as Alliances among Multiple Clients and Vendors' (*Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences, Maui*, IEEE Computer Society 1999); L Guan, 'Managing Risk in IT Contracts' (2012) 32 *Government News* 40; John Warchus, 'Avoiding Litigation: Drafting Successful IT Contracts' (1999) 15 *Computer Law & Security Report* 117.

- 18 Yogeshwar Shastri, Rashina Hoda and Robert Amor, 'Spearheading Agile: The Role of the Scrum Master in Agile Projects' (2021) 26 *Empirical Software Engineering* art 3.
- 19 *ibid.*
- 20 Shastri, Hoda and Amor (n 18); Maja Due Kadenic, Konstantinos Koumaditis and Louis Junker-Jensen, 'Mastering Scrum with a Focus on Team Maturity and Key Components of Scrum' (2023) 153 *Information and Software Technology* 107079; Hannes Salin, Felix Albrecht and John Skov, 'The Power of Scrum Mastery: An Analysis of Agile Team Performance and Scrum Master Influence' (*Proceedings of the 18th International Conference on Software Technologies, Rome*, SCITEPRESS 2023) 377.
- 21 Laura Dabbish and others, 'Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository' (*Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, Seattle*, ACM 2012) 1277; Antonio Lima, Luca Rossi and Mirco Musolesi, 'Coding Together at Scale: GitHub as a Collaborative Social Network' (*Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*, AAAI Press 2014) 295.

within GitHub create their code independently.²² And even when they do work together on a public project, research indicates that their participation is usually very cautious and rather small: limited to an average of 10 files.²³ In contrast, private (commercial) repositories are overwhelmingly multi-author and decidedly more extensive.²⁴ Furthermore, an examination of the relationship between the choice of programming language and the personnel complexity of the project suggests that the decision in this regard predefines collaborative authorship:²⁵ some programming languages require collaboration (such as JavaScript or Ruby), while others allow for limited communication between teams. Therefore, while for other creative fields, the decision to include another author in a project may be driven by creative invention, for some programming languages this becomes a key element of running a technology project.²⁶ Another study on public and private GitHub repositories suggests that it is unlikely to be possible to identify one person responsible for specific parts of the source code: at least several people are more or less engaged in every smaller or bigger part of the code.²⁷ The above studies very clearly challenge the copyright paradigms, based on the underlying assumption that authors tend to work independently on the work and co-authorship is generally the exception to this rule.

14 The concerns mentioned earlier may be alleviated through the application of contractual clauses and the mutual consent of the parties with regard to the collaborative nature of the creative process and its extent. Whilst there is a degree of consensus that author status is a matter of fact rather than contract,²⁸ it is suggested that some regulatory guidance could be helpful in the event of a legal dispute. However, the software development context does not facilitate this resolution either. Contracts in the IT sector are among the most multi-faceted, multi-level and hence multi-party.²⁹ They typically regulate not only the project's commercial object, but also development methodologies, security obligations, compliance requirements, data protection, allocation of implementation risks, testing and acceptance procedures, change-management mechanisms, maintenance expectations, and delivery schedules. Even so, no contract is capable of exhaustively capturing the full reality of software production. Software development is not a linear execution of pre-defined obligations, but an iterative and dynamic process shaped by continuous adjustment, unforeseen dependencies, emergent technical constraints, and evolving business expectations. As the project progresses, new functionalities are proposed, bugs reveal architectural weaknesses, integrations with external services reshape technical priorities, and security or regulatory requirements may require substantial redesign. In such an environment, not every relevant contribution can be anticipated *ex ante*, named in advance, or reduced to a stable contractual category. The legal document may define roles formally, but it cannot fully predict how influence over the final form of the software will in fact be exercised throughout the lifecycle of the project. Software development has evolved into a multifaceted process that demands a substantial investment of resources, particularly human capital. Modern software is rarely created within a single homogeneous team. Rather, it emerges from the interaction of developers, testers, UX designers, DevOps engineers, security specialists, data analysts, compliance officers, project managers, product owners, and, frequently, external vendors or client-side stakeholders. Their contributions operate on different levels, in different temporal phases, and according to different institutional logics. Some shape the literal code directly; others determine the architecture, priorities, feasibility, sequencing, or permissible scope of implementation. In practice, the final work is therefore often the result of distributed and cumulative acts of coordination, negotiation, review, acceptance, and revision. For that reason, the significance of particular individuals may exceed what is visible from the contractual allocation of

22 Lima, Rossi and Musolesi (n 21); Guilherme Avelino and others, 'Measuring and Analyzing Code Authorship in 1 + 118 Open Source Projects' (2019) 176 *Science of Computer Programming* 14.

23 Avelino and others (n 22).

24 Avelino and others (n 22); Dabbish and others (n 21); Fabian Stephany, Fabian Braesemann and Mark Graham, 'Coding Together – Coding Alone: The Role of Trust in Collaborative Programming' (2021) 24 *Information, Communication & Society* 1944.

25 Gabriel P Oliveira and others, 'How Do Developers Collaborate? Investigating GitHub Heterogeneous Networks' (2023) 31 *Software Quality Journal* 211; Natércia A Batista and others, 'Collaboration Strength Metrics and Analyses on GitHub' (*Proceedings of the International Conference on Web Intelligence, Leipzig, ACM 2017*) 170.

26 Batista and others (n 25); Dabbish and others (n 21).

27 Casey Casalnuovo and others, 'Developer Onboarding in GitHub: The Role of Prior Social Links and Language Experience' (*Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, ACM 2015*) 817; Pedro Henrique Pereira Braga and others, 'Not Just for Programmers: How GitHub Can Accelerate Collaborative and Reproducible Research in Ecology and Evolution' (2023) 14 *Methods in Ecology and Evolution* 1364; Ines Mergel, 'Open Collaboration in the Public Sector: The Case of Social Coding on GitHub' (2015) 32 *Government Information Quarterly* 464; Nikolas Zöllner, Jonathan H Morgan and Tobias Schröder, 'A Topology of Groups: What GitHub Can Tell Us about Online Collaboration' (2020) 161 *Technological Forecasting and Social Change* art 120291.

28 van Eechoud, *The Work of Authorship* (n 4); Bently (n 4); Arewa (n 7).

29 Gallivan and Oh (n 17); Guan (n 17); Warchus (n 17).

duties alone. Persons, not merely clauses, influence the shape of the resulting software product. Accordingly, while contractual arrangements remain indispensable for reasons of legal certainty and risk allocation, they should not be assumed to resolve, in any definitive manner, the problem of authorship in software development. The complexity, variability and human embeddedness of software production mean that authorship-related questions cannot always be settled by reference to formal provisions alone. In many instances, the decisive factors will lie in actual collaborative practice: who formulated the decisive solution, who redirected the project at a critical moment, who resolved a bottleneck that altered the final implementation, or who exercised meaningful control over the program's structure and functionality. The software environment therefore demonstrates particularly clearly that legal categories of authorship may be underinclusive when confronted with a creative process that is fluid, collective, and only partially foreseeable. From this perspective, contractual regulation of authorship should be treated as necessary but inherently incomplete, since the practical significance of human relationships, situated decision-making and cross-functional cooperation cannot be fully translated into *ex ante* legal language.³⁰

- 15 In conclusion, it seems that the question of authorship of a computer program is by no means a simple and straightforward matter. It is part of a very complex and multifaceted technological process involving various actors, to varying degrees and intensities, making the determination of who has rights, and to what, an important question. Despite the fact that the prevailing legal paradigm asserts that copyright is inherently attributed to the creator, this approach appears to be inadequate when confronted with the complexity, hybrid nature and multi-faceted character that are increasingly prevalent in the domain of modern computer programs and mobile applications.

C. Myth No. 2: The Majority of Source Code Is (and Should Be) Protected

- 16 Any claim that copyright overreaches in relation to software must begin with the concept of originality. Copyright does not protect all code, but only code that satisfies the applicable originality threshold. In the EU framework, that threshold is whether the relevant expression is the author's own intellectual creation; by contrast, elements dictated solely by technical function, constraints of the programming language, interoperability requirements or standard

coding conventions fall outside protection. Framed this way, the argument is not that copyright law automatically protects all source code and therefore needs radical reduction. Rather, the problem is that software discourse often moves too quickly from "there is code" to "there is protected expression", without sufficiently testing originality, functionality and the narrow scope of protectable expression in computer programs.

- 17 This doctrinal baseline matters because the legal materials already contain resources for a more restrained analysis. The CJEU's originality jurisprudence, usually associated with *Infopaq*, insists on free and creative choices;³¹ the software acquis adds that no criteria other than originality apply, but also that protection does not extend to ideas and principles underlying any element of a program.³² Likewise, the case law on computer programs stresses that functionality, programming languages and data formats are not monopolised as such.³³ Properly understood, then, software copyright is neither absent nor all-encompassing: it protects expressive choices in code, but not the technical necessities, methods and standards through which software remains usable and interoperable.
- 18 Within the domain of copyright scholarship, the concept of 'kleine Münze' holds particular significance. It was proposed that this term should be used to denote types of works which, due to their 'insignificance', 'minimalist content' or 'utilitarian character', should not be considered as subjects of copyright protection.³⁴ Examples include, for example, the design of a simple kitchen tool (knife); a bus timetable; a monthly calendar; and, in modern times, also: emoji.³⁵ It is challenging to unequivocally identify the individual who first recognised the issue concerning the copyright protection of a computer program. It has been argued that it should not be subject to the provisions of copyright law, but rather those of patent law.³⁶ However, as Boyle has

31 Case C-5/08 *Infopaq International A/S v Danske Dagblades Forening* ECLI:EU:C:2009:465.

32 Directive 2009/24/EC (n 2) art 1(2).

33 Case C-406/10 *SAS Institute Inc v World Programming Ltd* ECLI:EU:C:2012:259.

34 Annette Kur, 'Die Alternativen zum Schutz durch das Urheberrecht in Deutschland' in Reto M Hilty and Christophe Geiger (eds), *Impulse für eine europäische Harmonisierung des Urheberrechts* (Springer 2007) 193; Marcel Bisges, *Die Kleine Münze im Urheberrecht: Analyse des ökonomischen Aspekts des Werkbegriffs* (Nomos 2014).

35 Eric Goldman and Gabriella Ziccarelli, 'Emojis and Intellectual Property Law' (Santa Clara University Legal Studies Research Paper, 2018).

36 James Boyle, *Shamans, Software, and Spleens: Law and the Construction of the Information Society* (Harvard University Press 1996).

30 Guan (n 17); Trif (2024).

noted, copyright protection is significantly more favourable for a number of reasons. These include its global nature, the limited formalities required to obtain protection, and the complexity and duration of the rights granted.³⁷ The benefit of copyright protection has ultimately led to a state in which we assume that each element of a computer program is treated in the same way as a literary work, i.e. it is protected primarily in its textual layer.³⁸ However, a closer inspection of this assertion reveals certain inaccuracies.

- 19 A single written line of source code does not necessarily translate into action or specific functionality for the entire system. Indeed, in the majority of cases, it can be considered a directive to invoke or transmit specific information to a designated location (e.g. when displaying a particular screen in a mobile application). In many instances it merely invokes a command, implements a standard library call, satisfies a syntactic requirement or reproduces a functional solution that is largely dictated by the chosen language or framework. For that reason, the originality inquiry in software must be conducted at an appropriate level of abstraction. Some protectable expression may lie in a larger sequence, in a distinctive arrangement, or in a specific implementation that reflects free and creative choices. But much code is better understood as constrained, interoperable and heavily shaped by function. That insight does not abolish copyright in software; it narrows it to those aspects that are genuinely expressive rather than technically inevitable.
- 20 Moreover, the formulation is predominantly contingent on the characteristics inherent to the selected programming language. In a manner analogous to the capacity to articulate the aesthetic appeal of weather through diverse linguistic expressions, a programming language possesses limited capacity to convey a single 'creative thought'.³⁹ Only the writing of a few (and

sometimes dozens) lines of code translates into a greater meaning for the entire program, and on top of that, it still has to correctly interact (compile) with other elements usually written by other software engineers. By default, then, developing software is far more limited, in comparison to other creative activities, in terms of freedom of expression. And even setting aside the question of protecting the literal part of the code, for some time the copyright jurisprudence has recognised the so-called non-literal elements of computer code, understood mostly as structure, sequence and organisation, provided they were adequately distinguishable from other programs of a specific type.⁴⁰ Although this position has been revised slightly over time,⁴¹ the fact that these non-literal blocks of source code may be the result of a need related to the compilation of the code, the implementation of a relevant API, or an accepted programming convention, is still not currently taken into account. It is possible to provide numerous examples of the misunderstanding of software in the copyright system, but these do not alter the dogma accepted in the literature as well as in the case law that the source code of a program is generally protected, even if it is only a line of such a code, especially if it contains 11 words.⁴²

- 21 This romantic notion of the creative genius, uniqueness and originality of computer program code does not align with the reality that a significant proportion of code is merely copied as part of software development. Golubev et al.⁴³ conducted a survey of 23,378 projects on GitHub, each containing at least one line of Java code. The study exclusively focused on projects that had amassed a minimum of 50 stars, a metric that serves as an indication of user interest on GitHub. The study found evidence of at least 11.7% of exact clones, defined as copied code snippets from one project to another, implemented without any alterations.⁴⁴ Meanwhile, Baltes et al.⁴⁵ conducted

37 Boyle, *Shamans, Software, and Spleens* (n 36); James Boyle, 'The Search for an Author: Shakespeare and the Framers' (1988) 37 *Am U L Rev* 625; James Boyle, 'A Politics of Intellectual Property: Environmentalism for the Net?' (1997) 47 *Duke LJ* 87.

38 Jon Crowcroft, 'Copyright, Piracy and Software' in Lionel Bently, Jennifer Davis and Jane C Ginsburg (eds), *Copyright and Piracy: An Interdisciplinary Critique* (Cambridge University Press 2010) 209; K Tumbragel and R de Villiers, 'Copyright Protection for the Non-Literal Elements of a Computer Program' (2004) 10 *Computer and Telecommunications Law Review* 34.

39 Mark Gabel and Zhendong Su, 'A Study of the Uniqueness of Source Code' (*Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, Santa Fe*, ACM 2010) 147.

40 *Whelan Associates Inc v Jaslow Dental Laboratory Inc* 797 F 2d 1222 (3d Cir 1986).

41 *Computer Associates International Inc v Altai Inc* 982 F 2d 693, 706–711 (2d Cir 1992); Joshua Bloch and Pamela Samuelson, 'Some Misconceptions about Software in the Copyright Literature' (*Proceedings of the 2022 Symposium on Computer Science and Law*, ACM 2022) 131.

42 Mireille van Echoud, 'Along the Road to Uniformity – Diverse Readings of the Court of Justice Judgments on Copyright Work' (2012) 3 *JIPITEC* 60; Eleonora Rosati, 'Closed Subject-Matter Systems Are No Longer Compatible with EU Copyright' (2014) *GRUR Int* 1112.

43 Yaroslav Golubev and others, 'A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub' (*Proceedings of the 17th International Conference on Mining Software Repositories, Seoul*, ACM 2020) 54.

44 *ibid.*

45 Sebastian Baltes, Richard Kiefer and Stephan Diehl, 'Attribution Required: Stack Overflow Code Snippets in

a study that analysed copied snippets from Stack Overflow, a forum where developers pose questions regarding specific issues in code development. These studies estimated that approximately 25% of Java projects contained exact clones. In a slightly earlier analysis, this figure was marginally elevated (27%), with the observation that the greater the popularity of a repository among developers, the higher the proportion of code duplicated from another project (7.33%). On average, within a single project, the proportion of clones from external sources is 3.22%.⁴⁶ Indicated in a study on cross-project clones that the proportion of clones varies between 10% and 30%.⁴⁷ Following the construction of tools capable of tracing clones across a number of widely utilised programming environments, and subsequent analysis of over 428 million files written in Java, C++, Python and JavaScript, a corpus containing 85 million unique files was identified by other researchers. This finding suggests that approximately 70% of the code on GitHub is derived from pre-existing files. It was also noted that JavaScript has the highest rate of file duplication.⁴⁸ The statistics demonstrate that, on average, 25–30% of the code in a project is likely to be a literal copy from external sources. In some repositories, this proportion may exceed half.⁴⁹ And while a certain amount of repetitiveness may be present as an element of style, artistic convention or motif, when it comes to programming that is protected without closer review by the courts, this becomes an example of a failure to adapt the law to reality, and consequently opens up the field to develop its own tools to resolve any dissonance. It also stands in contrast to the prevailing assumption that an author's individual creative invention is primarily protected because of the creative effort they have invested in creating an original work. According to Gharehyazie et al., the majority of clones stem from the same project. In the event that suitable code is absent from a project, developers will subsequently turn to code from projects in the same application domain, and finally from projects in different domains. An algorithmic solution within the same application is therefore sought first,

leading to the question of whether subsequent code fragments within the same software or application become repeatable. The larger and more elaborate the project, logically the smaller the proportion of clones that occur, and the larger the copied fragment, the more additional modifications have been made to it relative to the original.⁵⁰ Therefore, it seems that, at least in the initial development phase, much of the code is neither original nor creative, but largely a duplication of existing structures from this or other projects. The authors of the study observe that, interestingly, in the majority of cases, the rationale behind clone copying is relatively prosaic. It is common practice among developers to replicate the complete source code of popular libraries in order to utilise their APIs; in the absence of this, developers would be unable to implement them, or the implementation would be less efficient. Furthermore, in more than half of the cases (56.12%), such cloned files are never modified, and in 75% of cases they are modified by only a small fraction (2.94%). The role of authorship in these clones, or 'super-sources', does not appear to be a significant factor in the cloning process.⁵¹ In such cases, should such fragments, where they merely implement commonly available solutions, be protected from a copyright perspective? It seems not, especially as further research indicates that there are such projects that function directly as "hubs" for other projects, i.e. they provide more clones than the other projects.⁵²

22 Open-source licensing is especially important here. Copyright is not displaced by open-source development; rather, it is operationalised through licences that permit copying, modification and redistribution on defined terms. This means that the prevalence of copying in software ecosystems does not by itself demonstrate legal failure. What it does demonstrate is that software practice relies on norms of attribution, compatibility and compliance that are often more granular than the abstract figure of the solitary author. The mismatch, therefore, is not simply between law and copying, but between a doctrinal emphasis on author-centred originality and an operational environment organised around reuse, modularity and licence stewardship.

23 Consequently, a more precise account of the mismatch between copyright and software practice is required. The problem is not that copyright law invariably prevents software from being written—plainly it does not, as the scale of commercial and open-source production shows. The problem

GitHub Projects' (2017 *IEEE/ACM 39th International Conference on Software Engineering Companion*, IEEE 2017) 161; Sebastian Baltes and Stephan Diehl, 'Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects' (2019) 24 *Empirical Software Engineering* 1259.

46 *ibid.* In contrast, Mohammad Gharehyazie, Baishakhi Ray and Vladimir Filkov, 'Some from Here, Some from There: Cross-Project Code Reuse in GitHub' (2017 *IEEE/ACM 14th International Conference on Mining Software Repositories*, Buenos Aires, IEEE 2017) 291.

47 *ibid.*

48 Cristina V Lopes and others, 'DéjàVu: A Map of Code Duplicates on GitHub' (2017) 1 *Proceedings of the ACM on Programming Languages* art 84.

49 Gharehyazie, Ray and Filkov (n 47).

50 Gharehyazie, Ray and Filkov (n 47); Mohammad Gharehyazie and others, 'Cross-Project Code Clones in GitHub' (2019) 24 *Empirical Software Engineering* 1538.

51 Gharehyazie and others (n 51).

52 Gharehyazie and others (n 51); Lopes and others (n 49).

is subtler. First, uncertainty about authorship and scope of protection shifts risk into contracts, compliance procedures and private platform governance. Secondly, it complicates attribution and licence management when projects involve layered contributions from employees, contractors, occasional contributors and code borrowed under different licence conditions. Thirdly, it increases the cost of dispute resolution because courts must reconstruct a collaborative and technically mediated process using doctrinal categories developed for more linear forms of creation. In practice, software continues to be produced, but at the price of extensive private ordering and with recurring uncertainty at the margins—precisely where disputes over provenance, credit, relicensing and enforcement arise.

- 24 The primary nature of software source code is syntactic: programming languages tend to be fully defined by relatively simple formal grammars that specify structured and idiomatic source.⁵³ It is grammar that largely dictates that statements be confined to method bodies, which in turn must reside within class declarations, each of which contains strictly defined sequences of tokens that include keywords and required punctuation. The possibilities of free structuring of the content, as demanded by copyright law, are significantly limited and sometimes even impossible, leading to the fact that we are dealing with a typical ‘*kleine Münze*’.⁵⁴ Additionally, style conventions restrict the space of available ‘creative’ choices for a developer, and are constantly being worked out by the community on platforms such as GitHub. Gabel and Su⁵⁵ further point out the problem of commonality in software engineering, which may lead to further similarities in source code.⁵⁶ This is particularly evident in low-level granularity, where the repetitive and idiomatic character of the code predominates. An exemplar of this is C-like languages, which are frequently replete with simple indexed for loops over known bounds. Consequently, the researchers predict in the extreme case that every token type in the language has been used at least once, and it is likely true that every legal two-token sequence has been written as well. In their view, while the number of legal statements in the language is theoretically infinite, the number of practically useful statements is considerably smaller and potentially finite. This concept is pivotal in the context of copyright law, as it underscores

the legal protection afforded to unique codes, phrases, statements, or tokens.⁵⁷ Ultimately, Gabel and Su argue that the uniqueness of a code is in fact something extremely rare, precisely because of the specificity of the language and its use.

- 25 Consequently, a certain dissonance emerges when comparing the assumptions inherent in copyright law with those observed in programming practices. Copyright legislation treats a program as a literary work, thereby forcing us to assume that every literal line of code created is in principle subject to protection. However, it does not consider the possibility that such a line may in fact invoke or implement a function, library or other source to the program that is ultimately intended to define the value of such a program. The same line of code is repeated on GitHub itself, and over time, it becomes a ‘standard’ among the development community because its implementation does not entail an excessive number of bugs. Also, the same copyright law treats a single developer as the author, disregarding the circumstances, personnel structures and teams with which they work, or who and to what extent binds them with certain instructions regarding the content, functionality or quality of the code they create. A certain ‘workaround’ solution would probably be to regulate this appropriately in the implementation contract, but firstly, such a possibility is not always allowed by law at all, and secondly, it would significantly expand the already complicated contractual pragmatics and negotiation procedure.

D. Software Creation Is Not for Lone Wolves

- 26 What then is the programming reality? The studies suggest that private repositories and commercial ones function in a somewhat different manner than those run publicly (and for free). The majority of activity is centred around private repositories, which are established by software companies for the purpose of developing commercial software.⁵⁸ It could be tentatively assumed that a significant part of the creative activity in programming is initiated when it is linked to an incentive in the form of remuneration; whereas software created in open and accessible repositories is the part of the creative effort that is actually created for purposes other than economic (for oneself, for ‘romantic’ purposes, in any case different from utilitarian *kleine Münze*). However, at this time, there is insufficient evidence to support this position. Research findings also indicate that economic motivation is not the sole catalyst for software creativity. As Stephany et al.

53 Gabel and Su (n 39).

54 Stefan Martin and Jonathan Boyd, “‘Software’: Clear and Precise?” (2021) 16 *Journal of Intellectual Property Law & Practice* 459; Paolo Guarda, ‘Looking for a Feasible Form of Software Protection: Copyright or Patent, Is That the Question?’ (2013) 35 *EIPR* 445.

55 Gabel and Su (n 39).

56 *ibid.*

57 Boyle, *Shamans, Software, and Spleens* (n 36).

58 Batista and others (n 25).

noted,⁵⁹ a significant factor in software development is the comfort of working in a specific environment, usually local, in which social ties and personal acquaintances play an important role. It was the social trust that motivated Stack Overflow users to respond to queries and disseminate knowledge pertaining to code development. It also plays an important role in explaining the geography of contributions, even though it does not explain the popularity of specific questions or forums. From a legal perspective, this is significant because it shows that participation in software-related knowledge exchange is often structured by motivations that do not fit neatly within the economic assumptions traditionally associated with copyright. Contributors are frequently driven by non-economic considerations, such as reciprocity, peer recognition, community standing, and the maintenance of reliable professional relationships, rather than by the expectation of direct remuneration. These forms of engagement are embedded in networks of social trust, personal ties, and informal dependencies, which both enable and incentivise the sharing of knowledge, code fragments, and problem-solving strategies.⁶⁰ In many cases, contributors respond not because they are contractually obliged or financially rewarded, but because they operate within networks of mutual recognition and professional dependency, in which helping others reinforces one's own credibility and position within the community. In such settings, creative contribution is not adequately explained by ownership structures, contractual allocation, or market exchange alone, since the decision to contribute often arises from relational commitments and community norms rather than from formal legal entitlements. This, in turn, complicates any attempt to conceptualise software authorship exclusively through the prism of individual economic incentive, and suggests that legal frameworks based on that assumption may fail to capture the social conditions under which software is actually produced, improved, and circulated.

- 27 The study on the relevance of society and interpersonal relationships within programming communities seems to confirm previous intuitions and corresponds with similar studies, although so far social trust has not been considered as an element of support in the creative process. In this regard, Batista et al. observed the interactions amongst users of GitHub and noted that most of them do not interact directly, but instead they work independently until a certain event brings them together: they have a common problem to solve and communication is the only way to work together to circumvent or identify the source of the

problem. However, before they enter into direct communication, they are restrained and attempt to find the source of the problem independently. Thus, despite technological and organisational pressures to collaborate, certain psychological factors suggest that developers are cautious in their interactions with others and limit them to purely technological matters. Empirical evidence also suggests that, in direct interactions, individuals tend to demonstrate a preference for collaboration with those in close physical proximity (once more: it demonstrates the importance of social relations and local connections). This finding lends further support to the initial conclusion that social trust and local community are integral components of the programming environment.⁶¹ From the standpoint of authorship, this observation is particularly important, because it suggests that software creation depends not merely on the aggregation of technical inputs, but also on relational conditions that make such inputs possible in the first place. Mutual trust between developers facilitates the disclosure of unfinished ideas, the sharing of tentative solutions, and the acceptance of interventions by others into one's own work, all of which are central to collaborative coding practices but difficult to capture in formal legal categories. In this sense, programming appears as a creative domain with a distinctive social structure: unlike more traditional models of authorship, often built around the image of a comparatively isolated creator or a more clearly bounded act of expression, software development is frequently iterative, distributed, and dependent upon repeated acts of interpersonal coordination. The originality embodied in software may therefore emerge not only from individual expression in isolation, but also from a trusted environment in which contributors can refine, test, challenge, and rework each other's solutions. This specificity matters for copyright doctrine, because it complicates the application of authorship concepts developed with other forms of creative production in mind, and suggests that, in the software field, the social architecture of creation is often inseparable from the creative process itself.

- 28 The present findings correspond with the research conducted on the phenomenon of the popularity of certain repositories.⁶² Popularity among repositories essentially means that such a project becomes a reference point for many subsequent ones: This phenomenon can be attributed to the frequent replication of code components within the development of emerging projects.⁶³ It can thus be

59 Stephany, Braesemann and Graham (n 24).

60 ibid.

61 Batista and others (n 25).

62 Oskar Jarczyk and others, 'GitHub Projects: Quality Analysis of Open-Source Software' in Luca Maria Aiello and Daniel McFarland (eds), *Social Informatics: 6th International Conference, SocInfo 2014, Barcelona* (Springer 2014) 80.

63 Gharehyazie, Ray and Filkov (n 47).

deduced that the duration of a project's existence is positively correlated with its popularity, particularly if it attains a degree of renown during its initial phase. Nevertheless, the volatility of technology and rapid changes in circulation often result in even the most popular and copied projects becoming obsolete rapidly. Of greater significance is the affiliation of the project with a particular organisation. The more prominent the brand of the company under whose umbrella the repository is operated, the greater the interest in the content of the project. As Jarczyk et al. observe, two further significant attributes—the average number of followers of developers in the project and the average number of developers/projects followed by them—can be regarded as an approximation of social relations among project members. The popularity of the developer, as determined by the researchers based on the number of followers, does not exert a significant influence on the popularity of the project itself. However, a correlation was observed between the involvement of more developers following other users and the popularity of the project. In essence, the prevalence of a project is amplified in scenarios where multiple developers are engaged, as opposed to scenarios where a solitary developer (yet recognised in the community) exhibits substantial involvement. The study found that involving more people in a project, even if they were not yet strongly recognisable and connected to the community, was more beneficial than the involvement of a single recognisable developer, who nevertheless has a rich portfolio of interdisciplinary projects.⁶⁴ As demonstrated above, repositories with a demonstrable history are generally regarded as the most reliable and authoritative sources. In most cases, such repositories are linked to well-known organisations or to identifiable developers acting under their official name, which makes the authenticity of the author a factor of considerable importance. In the software environment, the value of a contribution is therefore rarely assessed in complete isolation from the person or entity behind it. What matters is not only whether the code works, but also who produced it, under what name, with what history of prior contributions, and with what degree of recognised competence. A visible authorial track record functions here as a mechanism of trust: it helps users and collaborators assess reliability, predict quality, and decide whether a given repository, package, or solution is worthy of adoption or further development. This is significant because it points to a model of authorship that departs, at least in part, from the canonical assumptions of copyright theory. Traditionally, authorship has been closely linked to intellectual creation, individual expression, personality, and the justificatory framework associated with freedom of expression and moral rights. In that picture, the

author is recognised primarily because he or she originates an expressive form that reflects individual creative choices. In software development, however, public recognition of authorship is often shaped by a wider and more socially mediated set of factors. Authorial visibility may depend not only on the act of creation itself, but also on authenticity, continuity of participation, responsiveness to the community, technical authority, and the capacity to maintain a credible and recognisable public presence over time.

- 29 It is also noteworthy that active participation in the development of communities and the expansion of projects does not necessarily translate into the popularity of the projects in which users are involved. Yet such participation still performs an important authorial function, because it enables individuals to establish themselves within the development community and to build a personal brand that becomes relevant to the reception of their work. The formation of authorial recognition is thus interwoven with many elements that are more closely related to the person of the author than to the abstract need to create or to express ideas. Reputation, recognisability, prior conduct, community standing, and perceived reliability all contribute to whether a contribution will be noticed, trusted, and attributed significance. From this perspective, software authorship appears as a partly networked and reputation-based phenomenon, in which the social construction of the author may be just as important as the creative act itself. This does not displace the traditional copyright image of authorship altogether, but it does suggest that, in the software field, that image is incomplete unless it also accounts for the relational, public, and credibility-based conditions under which authors come to be recognised as authors.
- 30 The above research demonstrates that within the domain of programming, genuinely solitary authorship is relatively rare. Contrary to intuitions drawn from other forms of creative activity, code development appears to be predominantly a collective practice, shaped by social trust, collaborative methodologies such as agile and scrum, and, to some extent, economic incentives, including remuneration for the work performed. This observation is particularly important because it indicates that software creation cannot be adequately understood through models that presuppose an individual creator acting in relative isolation. Even where financial compensation is present, the production of code remains deeply dependent on interpersonal coordination, iterative revision, and shared technical environments. In that sense, software development differs not only from more individualised forms of authorship, but also from forms of voluntary or undercompensated creativity in which collaboration may occur without becoming so structurally central to the act of creation itself.

64 Jarczyk and others (n 63).

At the same time, the use of the same technological tools, especially GitHub, appears to strengthen collaboration without necessarily increasing the discoverability or originality of solutions.⁶⁵ What gains importance instead are factors such as the authenticity of the author and the reliability of the author's track record. Affiliations with recognised organisations managing a project, as well as standing within the relevant community, substantially affect the project's popularity and reputation. This means that the reception and authority of software outputs are shaped not only by the intrinsic quality of the code, but also by the social identity and credibility of those associated with it. The practical construction of authorship in software therefore involves more than the production of original expression; it also depends on visibility, trustworthiness, continuity of contribution, and community recognition. These elements are not peripheral. On the contrary, they are often central to determining whose contributions will be accepted, developed further, or treated as authoritative within the collaborative environment. When these findings are juxtaposed with the prevailing legal framework, a significant gap becomes apparent. Copyright law, particularly in its treatment of co-authorship, continues to rely on assumptions that fit poorly with the realities of software production, including the expectation of a concluding agreement between creators. Yet, as the analysis of GitHub pull requests suggests, software creation often unfolds through open, incremental, and socially mediated interactions in which formal closure between contributors may be absent or legally irrelevant to the actual emergence of the work. The context of collaboration, the architecture of social relations, the presence of trust, and the openness of the development process—factors repeatedly identified as central by empirical research—remain largely invisible within the doctrinal model. This is precisely why the software context calls for a revision of the classical model of authorship. A legal framework developed around more bounded and individualised acts of creation may be insufficient where creative production is distributed, iterative, and dependent on relational infrastructures that the law does not presently recognise. The problem, therefore, is not merely that copyright law offers incomplete answers, but that its underlying image of the author may itself be too narrow to account for how authorship is actually formed in contemporary software practice.

E. The Social Reactions

- 31 A substantial proportion of software development occurs not only on the GitHub platform, but also on Stack Overflow. Unlike GitHub, however,

StackOverflow operates as a forum in which users pose questions and receive responses in the form of comprehensive solutions. These replies often include lines of code that users can directly incorporate into their own projects.⁶⁶ Research into the phenomenon of this platform (98% of developers report using it with high frequency⁶⁷) has indicated that users are motivated to respond not necessarily by an altruistic need to support others, but rather by a need to build their own brand and recognition.⁶⁸ In contrast to other instances of community creation and collaboration, Stack Overflow users primarily seek knowledge but are reluctant to share or assist others, unless they recognise in this an opportunity for social empowerment.⁶⁹ In essence, software engineers adopt a selective approach when it comes to making voluntary contributions to publicly accessible projects. Gabel and Su's study revealed that more experienced and active developers were more inclined to reproduce code from other projects than to generate it autonomously. This finding suggests that software development proficiency may not depend on illusory creative genius, but rather on a skilled capacity to locate information on Stack Overflow and GitHub, and to navigate between different projects. It points to two insights that have already been touched upon.

- 32 First, the motivations underlying creation may be even more significant in software engineering than in many other creative fields. This does not mean, however, that economic incentives remain the dominant driver; social recognition, professional credibility, and inclusion within trusted technical communities may operate as equally powerful catalysts. From a legal perspective, this matters because copyright doctrine has traditionally been built around a comparatively narrow image of the author as an individual creator moved either by economic reward or by the expressive logic associated with personality and freedom of expression. In software practice, by contrast, the incentive to contribute is often bound up with collaborative reputation, peer validation, and the accumulation of technical authority within a community. That broader motivational structure should not be treated as merely incidental, since it shapes both the conditions under which software is created and the ways in which particular contributors come to be recognised as authors in practice.

65 Braga and others (n 27).

66 Iraklis Moutidis and Hywel T P Williams, 'Community Evolution on Stack Overflow' (2021) 16 *PLOS ONE* e0253010; Sohaib Mustafa, Wen Zhang and Muhammad Mateen Naveed, 'What Motivates Online Community Contributors to Contribute Consistently? A Case Study on Stackoverflow Netizens' (2023) 42 *Current Psychology* 10468.

67 Baltes and Diehl (n 45).

68 Mustafa, Zhang and Naveed (n 67).

69 *ibid.*

33 Secondly, authenticity and a proven track record are important not only because they help to build and consolidate the popularity of projects, but also because they may themselves become objectives pursued within the software community. A visible history of reliable contributions functions as a marker of trustworthiness, making it easier for a developer's work to be accepted, reused, and extended by others. In that sense, authorial recognition in software often emerges through a process of public credibility-building rather than through a single, isolated act of creation. This sits somewhat uneasily with legal models that continue to focus primarily on the moment of fixation or on formal categories of authorship, while paying less attention to the social mechanisms through which authorship is actually attributed and validated. The point may be illustrated, indirectly, by UK case law: in *Nova Productions v Mazooma Games*,⁷⁰ the courts drew a firm distinction between the protected expression found in code and the broader "look and feel" of the resulting software, thereby revealing how copyright doctrine tends to isolate discrete acts of expression even where the practical value of software is shaped by a wider collaborative and reputational context. More recently, *THJ Systems v Sheridan*⁷¹ reaffirmed that originality depends on the author's own intellectual creation and on identifiable creative choices, yet the dispute also shows how strongly software authorship questions are intertwined with issues of technical control, contribution history, and attribution in ongoing collaborative settings. Taken together, these considerations suggest that, in software, the making of an "author" is often inseparable from authenticity, reputation, and community-based trust, even though such factors remain only imperfectly reflected in the classical legal account of authorship. In this context, Baltes et al.⁷² conducted a study on the social reactions of users who copied other people's code into their projects. The need to reinforce one's own authenticity contradicts the common practice of not attributing authorship to the person who actually created the code. Indeed, 49% of developers reported that the last time they copied or adapted a code snippet from Stack Overflow, they did not attribute its origin.⁷³ This apparent contradiction can be explained by the fact that authorship in the legal sense—that is, being the author of the code as a copyright matter—is not necessarily aligned with attribution as it functions in programming practice. Judging from developers' reactions, the latter is often more important. Even where programmers do not

identify the author of the copied code by name, more than 40% of respondents reported adding a source-code comment containing a link to the relevant question or answer. In this environment, classical attribution through naming or signing the work is frequently displaced by reference to a repository, thread, or platform entry, which may be associated with a person, but does not necessarily belong to that person in any legally meaningful sense. What circulates, therefore, is not always the author's name as such, but a traceable point of origin within the technical ecosystem. This form of attribution is usually invisible to outsiders without access to the code itself, yet it remains visible, intelligible, and normatively significant within the developer community, where it is treated as an ordinary and legitimate practice.

34 Although such attribution is not legally sufficient, it performs an important social function in programming pragmatics. In a field where authenticity must be established continuously rather than presumed, developers reinforce their credibility not primarily through "authored publications" in the traditional sense, but through visible activity, sustained participation, and the number of references, forks, citations, and reuses attached to their work. Authenticity is thus built cumulatively and relationally: it depends on being seen, referenced, and recognised across the network, rather than on the formal assertion of authorship over discrete works. This is an important departure from more classical understandings of authorial recognition, in which the author is identified through the work as a bounded expressive object. In software culture, by contrast, recognition often attaches to a contribution history, a repository presence, or a pattern of problem-solving that others learn to trust. From a legal perspective, this suggests that practical forms of attribution in software are organised around reputation and verifiability rather than around the canonical copyright model of authorial designation.

35 The survey results concerning the source of copied code illustrate this point further. The majority of respondents (58%) identified Stack Overflow as the source. By contrast, some of those who claimed that the code had not been copied, even where an exact equivalent could be found on that platform, stated either that they had written it themselves or appeared simply unaware that they had reproduced existing material. This may partly reflect social desirability bias,⁷⁴ but it may also indicate a deeper uncertainty within the community as to where reuse ends and authorship begins. That uncertainty is legally significant. If developers do not clearly

70 *Nova Productions Ltd v Mazooma Games Ltd* [2007] EWCA Civ 219, [2007] RPC 25, affirming *Nova Productions Ltd v Mazooma Games Ltd* [2006] EWHC 24 (Ch), [2006] RPC 14.

71 *THJ Systems Ltd v Sheridan* [2023] EWCA Civ 1354, [2024] ECDR 4.

72 Baltes and Diehl (n 45).

73 *ibid.*

74 Anton J Nederhof, 'Methods of Coping with Social Desirability Bias: A Review' (1985) 15 *European Journal of Social Psychology* 263.

distinguish between original creation, adaptation, and copying, then the doctrinal assumptions on which copyright attribution is based become difficult to apply in practice. The problem is not merely one of compliance, but of conceptual mismatch between legal categories and everyday programming behaviour.

- 36 This mismatch is reinforced by the strikingly low level of legal awareness revealed by the available research. As Baltes et al. observe, up to 75% of respondents were unaware of the licensing terms governing the code snippets shared on Stack Overflow, namely CC BY-SA 3.0.⁷⁵ More broadly, the evidence suggests that many developers remain uncertain not only about licensing conditions, but also about the basic boundaries of copyright protection itself—what is protected, what is not, and under what conditions reuse is lawful. This points to the distinctly secondary role of copyright law in the normative universe of the programming community. In practice, developers appear to orient themselves less by formal legal rules than by community customs, technical convenience, and shared expectations about acceptable reuse. Copyright may continue to structure legal risk *ex post*, once a dispute arises, but it seems to exercise only limited influence *ex ante* over the everyday behaviour of developers. For copyright theory, this is a significant finding: in the software context, legal authorship and licensing rules do not appear to function as the primary behavioural framework for creative practice. Instead, programming communities are governed to a substantial extent by informal norms of cooperation, authenticity, and reputational trust, which often matter more in practice than the law designed to regulate authorship itself.
- 37 These findings help answer the objection that current copyright law cannot be much of a hindrance because software production is flourishing. Flourishing output does not mean optimal legal fit. A legal regime may still be poorly calibrated if its frictions are absorbed by platforms, employers, maintainers and developers through extra-legal workarounds. The evidence surveyed here suggests exactly that: attribution is often technical rather than legal, authorship is often settled *ex ante* by contract rather than *ex post* by doctrine, and compliance with licence conditions is uneven because many developers are only partially aware of the legal terms attached to the code they reuse. The law's categories are therefore not irrelevant; they are redistributed into governance costs, evidentiary burdens and downstream disputes.

F. Conclusions

- 38 The foregoing analysis leads to four principal conclusions, each of which responds to a specific doctrinal weakness in current software copyright discourse. First, the concept of software authorship is not as simple as we imagine it to be for the purposes of copyright discourse. Rather, it is a form of social, multifaceted teamwork, whereby participants influence the entire software production process in various ways. The team's relations primarily affect the production process itself, thereby influencing the final shape of the work. Furthermore, the underlying assumptions about the programming language and platform on which the code is created and stored, as well as the pragmatics of program development, effectively force participants to collaborate. Comfort and social trust in the working environment (broadly defined, and primarily in the online space within repositories) are also important factors in this, although the latter is quite specifically built. Here, there are no transparent, classical agreements that are legally binding. The interactions are purely technical and are not the first choice of the developers, even though all the other elements of this environment compel them to cooperate.
- 39 Secondly, the assumption that all computer program code is automatically protected by copyright is incorrect. The nature of programming languages significantly limits the choices available to those creating code, and there are also restrictions relating to functionality, the implementation of specific elements and even style conventions. Furthermore, a large proportion of the code in almost every project contains elements that have been copied from elsewhere. The personal and economic motivations of those creating the code also play a considerable role in software engineering. Although no research has been conducted into the altruistic motives of professional software developers, the significant activity in commercial projects and reluctance to interact directly, except when unavoidable, suggests that incentives are necessary to encourage collaboration. The popularity, development and quality of a project may also be influenced by the authenticity of the author and the proven track record. In this environment, the author's position and the ability to verify their achievements and project portfolio seem to be more important professionally than in other creative fields, especially compared to artistic or musical activities. Thirdly, the mismatch between copyright theory and software practice is best understood as a problem of calibration rather than outright incompatibility. Software production continues at scale, but legal uncertainty is absorbed through private ordering: contracts, repository rules, contributor policies, compliance workflows and informal attribution practices. This does not negate the mismatch; it shows where its costs

⁷⁵ Baltes, Kiefer and Diehl (n 45); Baltes and Diehl (n 45).

are borne. As argued above, contractual practices undoubtedly play an important coordinating role, yet they cannot fully capture the fluid, iterative and multi-party character of software development, nor can they anticipate every socially relevant contribution made during the life of a project. For that reason, many of the tensions left unresolved by copyright doctrine are managed in practice not by law alone, but by community conventions grounded in trust, reputation, authenticity and shared expectations concerning acceptable reuse and attribution. These conventions often prove more immediate and behaviourally effective than either statutory rules or contractual provisions, precisely because they are embedded in the everyday pragmatics of programming. The resulting picture is not one in which law disappears, but one in which legal and contractual regulation is supplemented—and in important respects functionally displaced—by social norms that more closely track how software is actually created, shared and recognised.

- 40 Fourthly, and normatively, reform should proceed on several levels. At the interpretive level, courts should apply a stricter originality analysis in software cases and should resist collapsing technical contribution into legal authorship without evidence of authorial expression. They should also distinguish more carefully between joint creation, technical facilitation, editing, project management and code review. At the legislative or policy level, lawmakers and soft-law bodies should encourage clearer guidance on collaborative software authorship, evidentiary presumptions for repository-based contribution records, and the interface between code attribution practices and moral-rights discourse. At the transactional level, software contracts and open-source governance documents should be drafted not merely to assign rights, but also to document contribution pathways, licence provenance and responsibility for compliance. Taken together, these steps would not abolish software copyright. They would make it more modest, more transparent and better aligned with the realities of contemporary software development. The revised framework proposed here is therefore not a replacement for copyright, but a methodological correction to its application in the software field: specify the territorial baseline, test originality rigorously, separate legal authorship from technical attribution, and treat collaboration and licensing as central rather than peripheral features of code production.
- 41 Taken as a whole, the foregoing analysis points towards a broader conclusion for copyright law: in the software environment, authorship is not primarily organised through the legal grammar of exclusive creation, formal attribution, and clearly bounded ownership, but through conventions that are social, technical, and reputational in nature. What governs behaviour in practice is often not the

copyright rule or even the contractual clause, but the community norm—how copying is perceived, when reuse is accepted, what counts as proper acknowledgement, and whose contributions are treated as credible within the relevant network. The frequency of copying, and the relative normality with which it is socially received, show that the movement of code across projects is not experienced as an exceptional infringement-prone event, but as an ordinary feature of programming pragmatics, provided that it remains within recognised conventions of use, reference, and trust. At the same time, the motivations for creating and sharing code remain normatively ambiguous: economic reward matters, but so do reciprocity, peer recognition, technical curiosity, community standing, and the cultivation of a visible, authentic track record. This gives rise to a model of authorial recognition that differs markedly from the historical copyright image of the author as a solitary, expressive, romantic genius. In programming culture, recognition is often accumulated through iterative participation, problem-solving visibility, repository presence, and repeated citation by others, rather than through the production of a discrete and self-contained “work” tied to a single authorial personality. The implication for copyright law is not simply that doctrine requires minor adjustment, but that its underlying assumptions about authorship may be misaligned with one of the most important contemporary domains of creation. Where conventions are more behaviourally effective than legal rules, and where social legitimacy matters more than formal entitlement in structuring everyday practice, the law cannot plausibly continue to treat such conventions as peripheral. Rather, they should be understood as central to any serious rethinking of authorship in software.

Acknowledgments: This work was supported by the National Science Centre in Poland (Narodowe Centrum Nauki) under grant no. 2023/49/B/HS5/00772.