OWNERSHIP AND CONTROL OF THE SOURCE CODE

Alexander S. POPA [*]

Abstract

This article examines the legal ownership and control of source code as an intellectual asset at the intersection of copyright doctrine and corporate practice. It clarifies why computer programs are protected by copyright as a form of expression, whilst ideas, principles, algorithms, and functionality remain outside the scope of protection.

The analysis distinguishes between source code, object code, and (where original) technical documentation, and explains the software-specific operation of moral rights (paternity and integrity) alongside economic rights. Building on this framework, the article maps the structural tension between the human developer—always the only possible "author" under copyright—and the organisational settings in which software is produced (employment, commissioned work, and services). It evaluates the presumptions and limits of transfers of economic rights to employers, focusing on code created outside contractual duties, the interpretative role of job descriptions and internal IP policies, and the practical meaning of "control" over modification, integration, sublicensing, and commercialisation.

Finally, it addresses contemporary grey areas: co-authorship in team-based and agile development; AI-assisted code generation (excluding human–AI co-authorship and stressing the need for human creative choices); and layered rights arising from open-source components, derivative works, and licence compatibility and "contamination" risks. The article proposes criteria and contractual governance techniques to reduce recurrent disputes.

Keywords: *intellectual property, AI, doctoral research, artificial intelligence, law, machine learning.*

1. Introduction. Source code as a strategic intellectual asset and the issue of legal control

In the current context, source code has moved beyond the status of a mere technical tool to become a strategic intellectual asset with significant economic, competitive and legal value. Within the digital industries, software represents not only the functional backbone of products and services, but also the expression of a complex creative endeavour, the result of the competence, experience and intellectual choices of the developers involved. From this perspective, source code lies at the intersection between intellectual creation protected by copyright and the investment logic specific to the commercial environment, giving rise to legal disputes regarding the ownership, control and exploitation of[1] .

This reality has led to an intensification of the structural tension between the human author of the source code—the programmer, in their capacity as an intellectual creator—and the organisational framework. When we discuss the organisation, we are referring to any legal entity, from commercial companies and research institutes to universities, where creation takes place on the basis of a relationship, ranging from an employment contract, a contract for work or a service contract, and including the context of a research grant.

Whilst copyright traditionally rests on the premise that a work belongs to its author, the practice of software development is characterised by collaborative projects, specific economic objectives[2] and the legitimate expectation of employers to exercise full control over the results of the work they have funded.

This tension is not purely theoretical. It manifests itself in concrete legal questions, such as: Who holds the economic rights to the source code? To what extent does the programmer retain their moral rights? What are the limits of the employer's or organisation's control over the modification, integration or licensing of the code? What rights are transferred from the supplier of a model to the programmer-user?[3] How are rights allocated

[*] PhD Candidate, Faculty of Law, „Nicolae Titulescu" University of Bucharest (e-mail: alexspopa@gmail.com).

[1] P. Ohm and B. Reid, *Regulating Software When Everything Has Software,* p. 4, the George Washington Law Review, 2016, available at https://www.gwlr.org/wp-content/uploads/2016/11/84-Geo.-Wash.-L.-Rev.-1672.pdf, accessed 05.02.2026, 11:30.

[2] R. Davis, P. Samuleson, M. Kapor, J. Reichman, *A new view of Intellectual property and software,* Communications of the ACM, March 1996, vol. 39, no. 3, p. 23, available at https://people.csail.mit.edu/davis/cacm96.pdf, accessed 02.03.2026, 20:00.

[3] Available at https://openai.com/ro-RO/policies/row-terms-of-use/.

when the software creation is the result of multiple contributions? The answers to these questions cannot be formulated solely by reference to the legal framework, but require a contextual analysis that integrates specialist doctrine and relevant judicial practice.

The issue of control over the source code is a key point in this debate. Control is not limited to the mere formal ownership of economic rights, but involves the actual decision regarding how the code is commercially exploited: modified, adapted, integrated into other software products, sub-licensed or made available to third parties. From a legal perspective, this control is often exercised by the employer or the entity coordinating the project, but from a creative perspective, it directly affects the author's work and, by extension, their relationship with their creation.

Copyright doctrine has consistently emphasised that software, although subject to special rules, is not excluded from the general framework of intellectual property protection. Established analyses of the legal nature of computer programs highlight the fact that the functional specificity of the code does not negate its creative character, but rather distinguishes it. At the same time, the specialist literature has shown that the employer's involvement in the creative process, through the setting of technical requirements, the general architecture or commercial objectives, does not automatically result in the developer losing their status as an author, but requires a fine distinction to be drawn between intellectual contribution and the organisational framework of the work.

Another factor complicating the legal analysis is the development of software through multiple contributions, which is characteristic of modern projects. Agile methodologies, teamwork, and the use of existing code or open-source libraries make it difficult to identify precisely who the authors are and the extent of each person's contribution. In this context, the concept of co-authorship takes on particular importance, but also presents specific challenges, distinct from those encountered in traditional artistic creations. Legal doctrine has repeatedly highlighted the risk of confusion between co-authorship proper and mere technical or executory participation, with direct consequences for the legal regime governing copyright.

Case law, at both national and European level, has reflected these difficulties, oscillating between the protection of employers' economic interests and the recognition of developers' creative role. Without offering uniform solutions, case law highlights the need for a balanced interpretation that takes into account the specific nature of each software project, the content of the contractual obligations and the nature of the actual contributions made by the parties involved.

The law frequently employs mechanisms to structure complex legal relationships or to facilitate the functioning of institutions; however, their legitimacy depends on their acceptance in legal theory and case law, as well as on their compatibility with the fundamental principles of the field in which they operate. From this perspective, this article also aims to distinguish between established and functional legal fictions and another conceptual construct, the validity of which is not recognised by the legal interpretative community. This latter fiction, although prevalent in technological or media discourse, has not attained normative recognition nor doctrinal acceptance within the interpretation of positive law. Its critical analysis becomes necessary precisely to clearly distinguish the legitimate instruments of legal technique from conceptual extensions that risk distorting the meaning and coherence of the applicable rules. In this context, it becomes necessary to clarify the legal framework applicable to source code, through a structured analysis of the rights of developers and employers, with a focus on the issue of control over the use of software and on situations of co-authorship and multiple contributions. The aim is not to formulate rigid solutions, but to highlight relevant legal criteria and recurring practical risks, which can be managed through an appropriate contractual and organisational approach.

Through this analysis, the article aims to provide a useful theoretical and practical tool for both legal professionals and those involved in software development, contributing to a better understanding of the necessary balance between the protection of intellectual property and the demands of the digital economy.

2. Source code as a work protected by copyright

2.1. The legal basis for the protection of source code

The legal protection of computer programs under copyright law is now a well-established regulatory framework both at the level of European Union law and in the domestic law of Member States. This legislative option was not an obvious one from the outset, given the technical, utilitarian and functional nature of software,

but it gradually became established due to the need to provide creators of computer programs with an effective protection regime, tailored to the specific nature of digital creation. The aspects concerning the protection afforded to computer programs spanned several legal levels, ranging from the protection provided by the rules governing patents for inventions, as was the case in the US, with an interesting development that culminated in 1976 with the Copyright Act, which established that computer programs, generically referred to, are copyright works[4] , in the sense of 'literary works'. The 1980 amendment to the Copyright Act explicitly clarified the inclusion of computer programs[5] .

At European level, the explicit inclusion of computer programs within the scope of copyright was justified by the notion that they constitute a specific form of intellectual expression, even though such expression does not take the traditional forms of literary or artistic creation. Through the provisions of Directive 91/250/EEC, the Council of the European Communities proposed a level of protection for computer programs, setting out basic rules for this legal framework. Furthermore, what Directive 91/250/EEC contributed from a general perspective was the set of definitions relating to computer programs, particularly regarding their communication and operational functions[6] . Romanian domestic law, primarily Law No. 8/1996, adopted this concept, expressly enshrining the protection of computer programs as literary works within the meaning of copyright law. Thus, the legislator opted for legal assimilation, rather than ontological identity, between software and classical literary works, recognising the specific nature of source code whilst integrating it into the general framework of copyright law.

This legal classification has direct implications for the regime of rights granted to the author, the duration of protection, the manner in which economic rights are transferred, and the applicability of moral rights. At the same time, it justifies the application of the criterion of originality as an essential element of protection, a criterion which, in the case of software, raises significant conceptual and practical difficulties, at least from this perspective.

2.2. Source code as a literary work: legal significance and limitations

The classification of source code as a literary work must be understood in a functional-legal sense, and this classification does not seek to equate the programmer with a literary author in the classical sense of the term, but rather to enable the application of a coherent set of legal rules to an intellectual creation expressed through a formalised language. In this context, the work – the source code – is written in an artificial language governed by strict syntactic rules; however, this does not preclude the existence of the author's freedom of expression. The choice of algorithmic structure, the organisation of the code, the technical solutions adopted to solve a given problem, and the use of specific programming paradigms or data structures are the expression of intellectual choices that may reflect the developer's personality and competence.

At the same time, case law and its interpretation have highlighted the fact that copyright protection does not extend to ideas, principles, algorithms or functionalities in themselves, but exclusively to the form in which they are expressed[7] . This distinction is essential in the field of software, as the risk of unjustified extension of protection to technical solutions could lead to the stifling of competition and the hindrance of technological progress.

Consequently, the classification of source code as a literary work must be understood as a functional legal fiction, designed to ensure the protection of creative expression, without conferring a monopoly over ideas or computer functions. The existence of legal fictions which, although constructive and artificial in nature, produce real normative effects and are coherently integrated into the architecture of the legal system is a relevant factor for the present analysis.

2.3. The condition of originality in the case of computer programs

Compared to literary works, computer programs present fundamental differences, and this aspect leaves no room for solid counter-arguments. Nevertheless, the inclusion of computer programs within the scope of

[4] V. Roș, *Intellectual Property Law – University Course,* Global Lex 2001, p. 199.

[5] Available at https://www.copyright.gov/title17/title17.pdf.

[6] Available at https://eur-lex.europa.eu/legal-content/RO/TXT/PDF/?uri=CELEX:31991L0250.

[7] See the BGH case, judgement of 3 May 2005 – I ZR 111/02 – Fash 2000, article available at https://www.glawe.de/en/european-court-of-justice-cjeu-on-protection-of-software-functionalities, accessed 03.03.2026, 12:00.

protection afforded to literary works was a step enshrined in the TRIPS Agreement, which, under the provisions of Article 10(1), specifies that they shall be governed, from the perspective of both source code and object code, by the provisions of the Berne Convention[8] .

The fundamental differences also concern the interpretation of originality. Starting from a generally accepted notion of originality as: a fundamental legal criterion that determines the recognition of copyright protection and concerns the result of an intellectual activity specific to the author, reflecting their free and creative choices, not to be confused with absolute novelty, artistic value, practical utility or technical complexity. From the perspective of the author-work connection, originality expresses the personal link between the two elements, manifested in the form of its expression.

In the case of a computer program, originality does not stem merely from the mere application of intellectual effort, labour and skill, but from a certain relevant level of these, a process which results in going beyond a level of mechanical execution or a purely functional contribution. Originality does not, in fact, operate as an absolute and uniform criterion, but as a gradual standard dependent on the particularities of each individual case. For this reason, originality cannot be reduced to clear and rigid formulas, requiring a contextual analysis that is sensitive to the nature of the premise and the type of contribution incorporated therein.

In this regard, I have deliberately avoided the term 'human contribution', based on the notion of a combined contribution by both human and machine in producing a result[9]

Example 1

```
numbers = [5, 7, 3, 10]
total = 0
for n in numbers:
total += n
average = total / len(numbers)
print("Total:", total)
print("Average:", average)
```

Example 2

```
def pulse(text):
words = text.split()
transformed = []
for i, w in enumerate(words):
if i % 2 == 0:
transformed.append(w.upper())
else:
transformed.append(w[::-1])
return " | ".join(transformed)
print(pulse("Intellectual property is the field of the
future"))
```

The examples presented allow for an applied analysis of the fact that originality is not synonymous with complexity.

Whereas in the first case the programmer follows the most direct and predictable path to achieve a functional result, in the second case we see the programmer making their own choices regarding expression. In this example, the programmer-author does not merely process the text, but decides in a particular way how exactly to transform it: they vary the treatment of words, impart a certain rhythm to the result, and construct a form of presentation that was not dictated by technical necessity alone.

Unlike the first example, which is limited to performing a simple operation in an almost inevitable, standardised algebraic form—[10] —this second code no longer appears to be a mere mechanical transposition of a technical requirement. Consequently, the difference from the first code does not necessarily lie in the technical difficulty or the length of the sequence, but in the fact that the second example suggests a genuine margin for creative freedom. Whilst the first code could have been written, almost identically, by any programmer faced with the same basic task, the second reflects a certain combination of choices, arrangements and phrasing options that were not strictly necessary for its operation. It is precisely this difference in approach to the technical function that supports the argument that, in the second case, the code can be viewed not merely as a functional tool, but also as an expression of one's own intellectual contribution.

Originality is the central criterion for copyright protection and, at the same time, one of the most problematic aspects when it comes to software. Unlike traditional literary or artistic works, where originality is

[8] V. Roș, A. Livădariu, C.P. Romițan, *Intellectual Property Law. Copyright and Related Rights and Industrial Property Rights,* C.H. Beck 2025, p. 125.

[9] L. Bently, B. Sherman, *Intellectual Property Law – Third edition*, Oxford University, 2009, p. 97.

[10] C. Petzold, *CODE The hidden language of computer hardware and software*, Second edition, Pearson Education, 2023, p. 42.

often manifest in an intuitively perceptible form, in the case of source code it must be analysed in terms of the author's freedom of choice and intellectual contribution[11] .

Several authors have articulated the idea that originality presupposes the existence of an intellectual creation that is unique to the author[12] , reflecting their personality. When applied to software, this definition implies a careful assessment of the degree of technical constraint. Where a solution is dictated solely by functional requirements, technical standards or interoperability, the scope for creative expression is limited or even non-existent, which may preclude copyright protection.

Conversely, where the author has genuine scope for discretion – in choosing the application's architecture, structuring the code, or implementing functionalities – originality may be recognised, even if the programme's ultimate purpose is utilitarian or merely operational.

This distinction between functional solutions and creative expression is of major practical importance, as it determines the limits of protection and influences the analysis of potential copyright infringements in the field of software.

2.4. The distinction between source code, object code and technical documentation

When analysing the legal framework governing computer programs, it is essential to draw a clear distinction between their various components, each of which has a distinct significance from a copyright perspective.

Source code	Object code (disassembly)	Technical documentation
int add(void) { *int a = 5;* *int b = 7;* *return a + b;* *}*	*push rbp* *mov rbp, rsp* *mov DWORD PTR [rbp-4], 5* *mov DWORD PTR [rbp-8], 7* *mov edx, DWORD PTR [rbp-4]* *mov eax, DWORD PTR [rbp-8]* *add eax, edx* *pop rbp* *ret*	*This code snippet defines the add function, which initialises two local integer variables, a and b, with the values 5 and 7, and then returns their sum. The source code is written in a form that is understandable to the programmer, whilst the object code represents the compiled version of the same instructions in a form that can be executed by the processor. The exact representation of the object code depends on the technical compilation environment.*

Source code is the human-readable form of a programme, written in a programming language and capable of reflecting the author's intellectual choices. For this reason, source code is the primary subject of copyright protection in the field of software.

Object code, produced as a result of compiling source code, takes the form of a sequence of instructions in binary format or in a derived hexadecimal representation, intended for interpretation and execution by computer systems. Although unreadable to most users, object code is consistently regarded as a derivative form

[11] C-604/10, Football Dataco Available at https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A62010CJ0604_SUM&utm_source=chatgpt.com.

Secondly, as is apparent from recital 16 of Directive 96/9, the concept of the author's own intellectual creation refers to the criterion of originality. As regards the creation of a database, that criterion of originality is satisfied when, through the selection or arrangement of the data it contains, its author expresses his creative ability in an original manner by making free and creative choices and thus leaves his 'personal mark'. By contrast, that criterion is not satisfied when the creation of the database is dictated by technical considerations, rules or constraints leaving no room for creative freedom. Accordingly, no other criteria than that of originality are to be applied to determine the eligibility of a database for the copyright protection provided for by the directive.

[12] D.I. Bainbridge, Intellectual Property, 8th Edition, Pearson Education, 2010, p. 41.

of the source code and enjoys protection to the extent that it reproduces the expression thereof. Technical documentation, including functional specifications, user manuals or software architecture documents, may in turn constitute a protected work, to the extent that it meets the condition of originality. Legal doctrine has established that technical documentation is not automatically protected, but only to the extent that it expresses an original structure of information, going beyond a mere standardised technical description.

2.5. Moral and economic rights applicable to software

The copyright regime governing computer programs is characterised by a specific application of moral and economic rights. Although moral rights are recognised for software authors, the exercise of these rights is often limited by the functional nature of computer programs and the professional context in which they are created.

The right to authorship of a work remains, in principle, applicable, with the author having the right to be recognised as the developer of the source code; however, in practice, this right is often exercised in a limited form, particularly in the context of collective works or programmes developed within the framework of employment relationships.The right to the integrity of the work raises specific difficulties, as software is, by its very nature, subject to continuous modifications, updates and adaptations. Legal doctrine has shown that the exercise of this right must be reconciled with the functional need to maintain and develop the programme, which justifies a flexible interpretation of the concept of infringement of the integrity of the work. Economic rights take on central importance in the field of software, being closely linked to the economic exploitation of the programme. The rights of reproduction, distribution, adaptation and making available to the public are essential for the commercial exploitation of the source code and constitute, in practice, the main subject matter of assignments or licences granted to employers or third parties. This dual structure of rights – moral and economic – forms the legal foundation upon which the analysis of the relationship between developers and employers, as well as the issues of co-authorship and multiple contributions in software creation.

3. The rights of the programmer – the author of the source code

3.1. Authorship of the source code. Natural persons and the impossibility of legal persons being authors

In the field of copyright, authorship is consistently and unequivocally reserved for the natural person who created the intellectual work. But is this statement true?

Within the framework of copyright law, the concept of authorship has a consistently and strictly defined meaning: the author is the natural person who creates an intellectual work. This concept is not a mere accident of the lawmaker, but reflects the very foundation of copyright protection.

Copyright protects the expression of human intellectual activity, and the link between the author and the work is conceptualised as direct, immediate and personal; and in Romanian law, this principle is expressly enshrined in Article 3(1) of Law No. 8/1996, according to which ‘the author is the natural person or persons who created the work’. The text leaves no room for a broad interpretation that would include legal persons or technical entities within the scope of authorship. Furthermore, the provisions on computer programs, included in Title III of the law, do not derogate from this principle. On the contrary, Article 73(1) provides that computer programs are protected as literary works, which entails the application of the same rules regarding the original ownership of rights.

At the level of European Union law, the same approach is reflected in Directive 2009/24/EC on the legal protection of computer programs. Recital (7) and Article 1(3) emphasise that protection is granted to computer programs insofar as they constitute the author’s own intellectual creation. The concept of one’s own intellectual creation has been consolidated in the case law of the Court of Justice of the European Union as an expression of human intellectual intervention and the existence of free and creative choices. In the field of software, this rule remains fully applicable. The source code, even when developed within a corporate structure, remains the result of intellectual choices made by individuals – programmers, software architects, developers – who decide on the structure, organisation, formulation and implementation of technical solutions. The fact that the work is carried out within the framework of an employment relationship or a service contract does not transform the legal entity into the author. Pursuant to Article 74 of Law No 8/1996 and Article 2 of Directive 2009/24/EC, the employer may, under certain conditions, acquire the economic rights to the programme created by the employee;

however, this acquisition is of a derivative nature and does not affect the original status as author of the individual who created the work.

Legal doctrine has consistently emphasised that a legal person cannot, in a legal sense, be regarded as an author, as it lacks the capacity for intellectual creation, in contrast to the human individual who possesses qualities such as: intelligence, personality, feelings or creative power[13] . Legal personality is a normative fiction that allows a collective entity to participate in civil transactions, but it does not equate to the existence of a creative consciousness. Authorship implies a direct link between the work produced and the intellectual activity of a natural person, which cannot be attributed to an organisational entity, regardless of the complexity of its structure.

In practice, the field of software development tends to blur this distinction, promoting the idea of the organisational 'product'. Collective projects, fragmented workflows, and the use of standardised libraries and automation tools can create the impression of an impersonal creation. However, from a legal perspective, authorship is not diluted by organisation, subordination or technical cooperation. In the case of collaborative works, Law 8/1996 recognises the co-authorship of individuals who have contributed creatively, without attributing authorship to the organising entity.

In the age of artificial intelligence, the analysis is complicated by the emergence of code generated on the basis of a prompt formulated by a programmer. Even in this context, the same criterion remains applicable: the identification of a human intellectual contribution capable of conferring on the work the status of a protected creation.

If the programmer formulates instructions, selects results, modifies and integrates the code into their own architecture, their intervention may establish authorship. If, on the contrary, the result is generated entirely by an automated system, without human creative intervention, the issue is not the attribution of authorship to the legal entity or the machine, but the possible absence of copyright protection. Within this conceptual framework, the notion of a 'machine-author' would represent a radical departure from the logic of copyright law. This notion entails attributing authorship to an entity lacking legal personality and creative consciousness, which would contravene both the legal texts and the theoretical foundations of the institution. Romanian and European positive law provides no basis for such a classification, and recourse to this idea can only be of the nature of a legal fiction not enshrined in law.

3.2. The developer's moral rights in the source code

3.2.1. The right to authorship of the work

The right of authorship is the most obvious expression of the personal dimension of copyright and remains applicable in the field of computer programs. Even if the software is created within the framework of an employment relationship or a contract for the provision of services, the developer remains the author of the source code to the extent that their contribution is creative in nature. This right reflects the intellectual link between the individual and their creation and, by its very nature, cannot be transferred or waived by agreement between the parties, regardless of the regime governing economic rights.

In practice, however, the exercise of the right of authorship in the field of software is not without its peculiarities. Software programmes are usually marketed under the name of the product or the company, and the public interface does not explicitly indicate the names of the developers. This economic reality does not, in itself, negate authorship, provided that authorship is not falsely attributed to another person and that the recognition of the actual contribution is not impeded in relevant contexts, such as technical documentation, internal reports or potential disputes. In the case of team projects, the right to authorship takes a form adapted to the collective structure of the creation. Recognition of authors does not necessarily entail individual mention in every form of exploitation, but rather ensuring the possibility of identifying their contribution where this is relevant. This solution strikes a balance between the protection of moral rights and the practical requirements of software development and marketing, without undermining the personal nature of authorship.

[13] V. Roș, *Intellectual Property Law – University Course,* Global Lex 2001, p. 56

3.2.2. The right to the integrity of the work and the specific features of software

The right to the integrity of the work, in the sense of the author's right to object to modifications that affect their creation, takes on particular significance in relation to source code. Unlike traditional literary or artistic works, software is, by its very nature, subject to a continuous process of adaptation, optimisation and correction.

The analysis of the provisions of Article 78 of Law No 8/1996 on copyright and related rights must be carried out in conjunction with the special regime applicable to computer programs, a regime which is not autonomous but derives directly from European Union law, in particular from Directive 2009/24/EC on the legal protection of computer programs (which codified Directive 91/250/EEC). The provisions of Article 78 transpose Articles 5 and 6 of Directive 2009/24/EC into national law, establishing a set of specific limitations and exceptions tailored to the technical characteristics of software.

This approach must be understood as the expression of a deliberate legislative choice to treat computer programs differently from other categories of protected works. This distinction is not accidental, but stems from the profoundly functional nature of software and the need to ensure a genuine balance between the exclusive rights of the right holder and the legitimate interests of the authorised user. The principle essentially establishes three distinct regulatory mechanisms, which shape the specific legal regime governing computer programs. Firstly, it recognises the authorised user's right to make an archive or backup copy, to the extent that this is necessary for the use of the program. Secondly, it enshrines the right to analyse, study or test the functioning of the program in order to identify the ideas and principles underlying its elements. Thirdly, it provides that the moral right of withdrawal does not apply in the case of computer programs.

The legislative provisions under consideration are not merely technical exceptions, but reflect a coherent European model. They transpose into national law the solutions established by Directive 2009/24/EC on the legal protection of computer programs, which, in turn, established an autonomous regime within copyright law. Article 5(2) of the Directive enshrines the right of the lawful user to make a backup copy, whilst paragraph 3 permits the observation, study and testing of the functioning of the program in order to determine the ideas and principles underlying it. Romanian law reproduces these regulatory solutions almost verbatim, confirming their binding nature within the national legal order.

By recognising the right to make a backup copy, European and national legislators have acknowledged that the technical reproduction necessary to ensure the use of the program cannot be left to the sole discretion of the rightholder. That copy does not confer any additional right of exploitation and cannot be used for independent purposes, but serves solely to protect the legitimate user against the risk of loss or damage to the program.

A more significant aspect concerns the right to analyse, study and test; which enshrines, in the field of software, the fundamental principle of the separation of idea and expression, with the authorised user having the right to observe and examine the programme's operation, the sequence of lines of code, the connections and the interfaces, in order to understand the ideas and principles underlying it, without prejudice to the author[14] . This possibility is essential both for ensuring interoperability and preventing the monopolisation of technical concepts, as well as for rapid technological development. However, the recognised right does not extend to the reproduction of the protected form of expression; it allows access to the logic and functionality, not to the copying of the original expression.

In the case of computer programs, the inapplicability of the right of withdrawal reflects the specific economic and technical nature of this type of work. Unlike literary or artistic works, where withdrawal may be seen as an expression of the author's personality, in the case of software such a possibility would seriously undermine legal certainty and the stability of commercial relations. Once integrated into IT systems and technological infrastructures, the program becomes part of a functional ecosystem, and unilateral withdrawal could have disproportionate effects.

From the user's perspective, special legal provisions do not create independent rights in their favour, but rather limit the exercise of the rightholder's exclusive rights within a precisely defined framework, and their applicability is logically and legally contingent upon the existence of a protected work. In this regard, if a code is generated entirely by an automated system, without the creative intervention of a natural person, and the result does not meet the criterion of originality, it cannot be classified as a work within the meaning of copyright law.

[14] V. Roș, A. Livădariu, C.P. Romițan, *Intellectual Property Law. Copyright and Related Rights and Industrial Property Rights,* C.H. Beck 2025, p. 260.

In such a situation, there is no exclusive right that can be limited, and the legal mechanisms do not come into play. In other words, the rule cannot function in the absence of the object of protection. Exceptions and limitations do not exist independently of the exclusive right; they presuppose its existence.

The situation is different where the code is developed through the interaction between a developer/programmer and a large-scale, even specialised, language model. If the programmer formulates complex instructions, selects options, restructures solutions and integrates the results into their own architecture, their intervention may amount to a significant creative contribution. To this extent, the program acquires the status of a protected work, and the specific legal regime for computer programs becomes applicable.

In such a context, the authorised user of a copy of the program is entitled to make a backup copy, to the extent necessary for its use, as well as to analyse, study or test the program's operation in order to determine the ideas and principles underlying its components. These rights are, however, strictly limited to legitimate use and cannot be extended to the reproduction or exploitation of the protected expression beyond the legal limits. The right to understand the principles does not equate to the right to reproduce the form of expression.

The issue of training large-scale language models using protected code lies in a separate category. It cannot be subsumed under provisions concerning the lawful use of a programme by its authorised user, but must be analysed in relation to the exclusive right of reproduction and any exceptions provided for under European law, including those introduced by Directive (EU) 2019/790 on copyright in the Digital Single Market, relating to text and data mining activities. However, these exceptions do not alter the special regime for computer programs established by Directive 2009/24/EC, which remains the specific regulatory framework for software.

In the case of Bartz et al. v. Anthropic[15] , the argument set out above is consistent in legal terms, even though the Anthropic case was decided in the light of the fair use doctrine, rather than within the European framework of reproduction rights and the special regime governing computer programs. In that case, the court held that training based on legally acquired books may constitute fair use, and here we can draw a parallel with code sequences acquired through a legal process; however, it clearly distinguished the issue of pirated copies, for which Anthropic did not obtain the same protection. The main point of convergence between this article and the Anthropic case is that both start from the premise that training a model should not be confused with the mere legitimate use of the software copy by the authorised user. The issue of training a large-scale language model cannot be subsumed under the provisions concerning the legitimate use of the programme, but must be addressed within the context of acts of reproduction.

In the case in question, the real dispute was not about the normal exploitation of the work, but about the copying of the body of the protected work for training and storage purposes. An analysis of this case in two different jurisdictions would reveal the difference that the US court raised the question of whether that copying constitutes transformative fair use, whereas a court in Romania or the European Union would examine whether the reproduction falls within a legal exception or remains reserved to the copyright holder.

3.3. Initial economic rights over the source code

3.3.1. The rule of initial attribution to the author

Within the framework of copyright law, the general rule is that economic rights initially belong to the author of the work, a natural person; this is the fundamental principle enshrined in all statutory provisions. Why was it necessary to specify 'natural person' earlier? Contemporary issues raise a question to which the vast majority of legal experts object, namely, the 'machine author'. Copyright has historically been built on a foundation whereby the author is not merely the holder of an economic right, but the subject of a legal relationship that presupposes a link between the work and the creator's personality. Moral rights, the link to authorship, the integrity of the work, and disclosure — all express this anthropocentric conception. By definition, the work is protected because it reflects human intellectual activity, a product of creation.

The concept of the 'machine-author' represents a radical break, a rupture in the logic constructed by practitioners and theorists alike, which entails attributing authorship to an entity lacking legal personality and creative consciousness. Of course, this view is recognised as being nothing more than a legal fiction.

[15] Available, including history, at https://admin.bakerlaw.com/wp-content/uploads/2025/02/ECF-1-Complaint-3.pdf, accessed on 10 February 2026, 14:00.

Legal fictions are classic tools of legislative technique, used to organise complex social realities by attributing legal effects to entities or situations which, in factual terms, do not directly correspond to traditional legal concepts; they are legitimate insofar as they serve a coherent purpose and do not alter the axiological structure of the institutions within which they are embedded. A relevant example is the legal personality of commercial companies, in that the law attributes legal capacity to a collective entity to facilitate economic activity, without denying that the corporate will is expressed through the natural persons who constitute it. Applying this logic to copyright law, through the recognition of a machine-author, raises far more profound conceptual difficulties, as authorship is not merely a technical means of allocating economic rights, but rather the expression of a personalist conception of creation, the result of complex neural processes based on creative processes, impossible to duplicate, generated by years of evolution[16] .

Attributing authorship to a machine would necessarily entail recognising a capacity to create in the legal sense—that is, to make free and creative choices attributable to a legal person—and would imply the possibility of exercising moral rights and, at least in theory, assuming a form of liability. However, a technical system, regardless of its degree of complexity, cannot manifest will, cannot claim authorship of a work, and cannot invoke an infringement of its integrity. It does not possess legal personality and cannot participate in legal relationships autonomously; moreover, the recognition of a machine-author would not merely represent a terminological innovation, but would have structural effects on the architecture of copyright law. From this perspective, the analysis must shift from the rhetorical level of technological novelty to that of systemic coherence.

Firstly, attributing authorship to a technical entity would lead to a significant broadening of the concept of a work. Of course, not every result generated by automated processes would qualify as a protected work, solely by invoking the fiction of the machine-author, but the scope of protection would extend beyond the realm of human intellectual creation, and assessing originality would be difficult to achieve. In such a case, copyright could end up covering products resulting from simple algorithmic processes, regardless of the existence of creative choices attributed to a natural person. Such an extension could exponentially increase the number of protected works, leading to a proliferation of situations of legal exclusivity, with a direct impact on the circulation of information and on competition. In such a case, protection would cease to be a mechanism for stimulating creation and would become a tool for monopolising technical production.

Another issue concerns the criterion of originality, which is the central element of copyright protection under European law. There is a risk that copyright could be transformed into a system for protecting automatically generated results, without it being possible to identify the link between the work and human intellectual activity. In this context, the very notion of creation would become ambiguous, and the distinction between protected expression and mere functionality would be blurred. Last but not least, recognising the machine as an author would create ambiguities regarding liability. Copyright is not merely a system for allocating exclusive rights, but also a framework within which liability may arise for infringing rights or for disseminating unlawful content. In a situation where the machine could be classified as an author but could not bear the legal consequences of its own ‘acts’, liability would have to be redistributed to other persons involved — developers, users or infrastructure owners. This redistribution would confirm the artificial nature of the fiction: the entity formally designated as the author could neither exercise rights autonomously nor be held liable for infringements, which would demonstrate its legal futility.

In such a scenario, economic rights would inevitably have to be assigned or transferred to a natural or legal person—the system developer, the programmer, the user-author of the prompt, or the owner of the infrastructure. This redistribution confirms the artificial nature of the fiction: the machine would formally appear as the author, but without actually exercising any rights and without being able to bear any obligations. Authorship would become a purely formal intermediate stage, devoid of normative substance.

Irrespective of the above, this rule currently applies to computer programs, regardless of the context or the means by which they were created. The developer or programmer acquires, by the mere act of creation, all economic rights to the source code, including the right to reproduce, distribute, adapt and exploit it commercially. This initial attribution has fundamental legal significance, as it constitutes the starting point for any subsequent analysis regarding the transfer or limitation of rights. Even in situations where the law establishes presumptions of transfer in favour of the employer, these operate as exceptions to the general rule and must be interpreted restrictively.

[16] N. Toon, How AI Thinks, Penguin Random House UK, 2024, p. 144.

3.3.2. Assignment and licensing of economic rights

In the field of computer programs, economic rights relate to the reproduction, distribution, adaptation or incorporation of the source code into a larger software product. Moral rights, on the other hand, reflect the personal connection between the author—as a human individual—and the work. They protect authorship, integrity and the right to disclose the creation, and are, in principle, inalienable and non-transferable. Even when economic rights are transferred to the employer or a third party, the author remains the holder of moral rights, which reflects the personalist conception of intellectual creation. The distinction between the two categories is essential to understanding the difference between authorship and economic ownership of the work.

The transfer of economic rights to the source code may be effected by assignment or by the granting of a licence. The distinction between these two mechanisms is essential, from both a legal and an economic perspective. Assignment involves the permanent transfer of economic rights, with the author losing control over the subsequent exploitation of the work, within the agreed limits. A licence allows the beneficiary to exploit the source code without depriving the author of ownership of the rights. In software practice, licences are frequently used to permit the integration or use of programs in commercial products, whilst retaining a certain degree of control over the work.

In this context, the question arises as to whether it is possible to assign specific code sequences from a protected programme, and what the legal effect of such a transaction would be. From a copyright perspective, there is nothing to prevent the transfer of economic rights in an identifiable part of a work, provided that it is specific or determinable and constitutes, in turn, a protected expression. As source code is capable of technical and functional segmentation (modules, libraries, functions), it allows for such a contractual demarcation.

The assignment may relate to a distinct module or a set of identifiable functionalities, without entailing the full transfer of rights to the original programme. If the sequence of code thus transferred is subsequently integrated into another program, the result is not necessarily a derivative work in the legal sense. A derivative work involves the transformation or adaptation of a pre-existing work, whilst retaining a recognisable link to it. In the event that the economic rights to the sequence have been assigned and the assignee uses it as an autonomous element within a new software architecture, the resulting product may constitute a distinct work, without being classified as a derivative of the original program. The legal link to the original work is, in this case, mediated by the transfer of rights, not by a relationship of creative dependence.

The issue at hand concerns the distinction between the mere legitimate reuse of a transferred sequence and the creation of a derivative work; this distinction depends on the nature of the intervention on the pre-existing code, since the new programme involves substantial modifications to the structure or logic of the original sequence, raising the question of whether a derivative work has been created based on that component. Conversely, when the sequence is used as such, on the basis of a valid transfer of economic rights, the new creation constitutes an autonomous work, within which the pre-existing element functions as a legitimate integrated contribution, not as a transformation of the original work.

3.4. Problematic situations in software development practice

3.4.1. Lack of a clear contract regarding rights to the source code

One of the most common disputes between developers and employers is the absence of clear contractual clauses regarding copyright over source code. In the absence of explicit provisions, the general rules of copyright law apply, which can lead to unexpected outcomes for the parties, particularly for employers, who often assume that the mere remuneration of the employee's work equates to the full acquisition of rights over the work created within the employment relationship. However, this presumption is not necessarily in line with the applicable legal regime. From this perspective, in the case of computer programs, Romanian law establishes a specific rule whereby, if the program is created by an employee in the course of their duties, the economic rights belong to the employer, in the absence of a stipulation to the contrary. However, this acquisition relates exclusively to economic rights and does not affect the author's moral rights. The programmer remains, from a legal point of view, the author of the source code, even if the employer becomes the holder of the economic exploitation rights. The right of authorship and the right to integrity are not absorbed by the corporate structure and cannot be nullified by the mere payment of a salary.

In practice, conflicts arise particularly in situations where the scope of job responsibilities is not clearly defined, or where the developer contributes to parallel projects, reuses previously created components, or integrates libraries and code snippets with different legal statuses. In the absence of express clauses regarding assignment, the scope of the rights transferred and the regime governing subsequent creations, the analysis is carried out by reference to the specific nature of the creative contribution, the job description and the circumstances in which the work was created. This approach, whilst consistent with the principles of copyright law, increases legal uncertainty and may lead to divergent outcomes. The analysis becomes even more complex in the context of the use of automated means or large-scale language models in the development process. If the programmer formulates a prompt and obtains a fragment of automatically generated code, the question arises as to who holds the rights to the result. To the extent that human intervention consists of formulating creative instructions, selecting and adapting the result, and integrating it into a proprietary architecture, the programmer's contribution may establish their status as an author, and the regime of economic and moral rights applies as usual. Conversely, if the result is generated entirely without any identifiable creative intervention, questions may arise regarding the very existence of copyright protection.

For the employer, the risk is twofold. On the one hand, they may assume that they automatically acquire economic rights over code whose protection is uncertain. On the other hand, they may overlook the fact that the programmer's moral rights remain, including in the context of the use of automated tools. In the absence of contractual clauses expressly regulating the use of AI tools, the distinction between human contribution and the status of pre-existing or automatically generated components, legal uncertainty is amplified.

3.4.2. Contributions made outside the scope of duties established under the employment contract

A distinct yet common situation in software development practice concerns contributions made by a programmer outside the scope of their job duties. In technological environments characterised by a rapid pace of innovation and a high degree of individual creative involvement, the distinction between work carried out in the performance of the employment contract and the developer's personal initiatives often becomes difficult to draw. This difficulty is compounded by the fact that programming work typically involves intellectual continuity and the reuse of professional experience acquired within the employer's organisation. The legal regime governing computer programs does not result in the automatic transfer of any creation made by an employee to the employer's assets. Although the law provides, in the case of programmes created in the course of employment, for a rule assigning economic rights to the employer, the decisive criterion remains the link between the creation and the contractual obligations undertaken. The mere fact that a sequence of code was written during working hours or using the employer's technical infrastructure is not, in itself, sufficient to justify the transfer of rights, in the absence of a genuine connection with the job description or with the specific tasks entrusted to the developer.

In the absence of this functional and legal link, the author – a natural person – retains, in principle, the economic rights to the code they have created, as well as the associated moral rights. This distinction has relevant practical implications in situations such as the development of parallel projects, the creation of personal prototypes, contributions to open-source projects, or the use of automated tools, including large language model systems or AI agents, outside the contractual framework. In such contexts, determining ownership of rights requires a careful analysis of the nature of the contribution, the degree of integration into the employer's business, and any applicable contractual clauses. The absence of clear internal policies regarding the use of resources, secondary developments or external contributions increases legal uncertainty and may give rise to disputes, particularly when the subsequent success of a project confers significant economic value on the original code.

4. The employer's rights over the source code

4.1. The legal basis for the employer's rights over source code created within the employment relationship and the presumption of transfer of economic rights

In the field of computer programs, the employer's rights over source code lie at the intersection of employment law, civil law and copyright law, constituting one of the most sensitive areas of regulatory overlap. Although the general rule in copyright law is that economic rights are originally held by the author as a natural

person, the specific economic nature of software development has led to the establishment of a special regime for works created within the context of employment relationships. This regime does not negate the personal nature of authorship, but adapts the mechanisms for the allocation of economic rights to the reality of the collective organisation of software production[17] .

The legislative solution is not based on the idea of 'employer authorship', which would contravene the personalist conception of copyright, but on the recognition of the economic role of the enterprise. The employer bears the financial risks of the project, provides the technical infrastructure, organises the work and pursues the commercial exploitation of the result. The transfer of economic rights over the source code, alongside the liability[18] , is thus conceived as a functional consequence of the employment relationship, intended to enable the efficient exploitation of the programme without affecting the developer's status as author. The author remains the natural person who made the creative contribution, and the moral rights belong to them by origin.

Under Romanian law, this solution is enshrined through the establishment of a legal presumption of transfer of economic rights to the employer, where the computer program is created by an employee in the course of their duties or within the scope of tasks set out in the individual employment contract. The rule reflects a legislative policy choice aimed at ensuring legal certainty in the exploitation of software. In the absence of such a presumption, each project would require individual negotiations regarding exploitation rights, which would affect the functioning of complex projects and generate contractual instability.

However, this presumption is not absolute and does not amount to a blanket deprivation of the author's rights. It applies only to the extent that there is a direct link between the software creation and the developer's job duties. The analysis must be carried out on a case-by-case basis, with reference to the job description, the instructions received and the purpose of the work for which the employee was hired. The mere existence of an employment relationship does not, in itself, justify the transfer of economic rights over any creation made by the employee.

In the current context, characterised by the use of automated tools and systems such as large language models and specialised AI agents in the programming process, this analysis becomes even more complex. If the developer uses such tools within the scope of their duties, the result – insofar as it constitutes a protected creation – will, in principle, fall under the legal presumption. However, if the use takes place outside the scope of contractual duties or the creative contribution cannot be subsumed under the employment relationship, the application of the presumption may be challenged. In all cases, the distinction between the employer's economic ownership and the individual's status as author remains essential to maintaining the coherence of the legal regime governing computer programs.

4.2. The limits of the presumption: the subject matter of the contract, the nature of the duties and the degree of creative autonomy

The first key limitation on the legal presumption of the transfer of economic rights to the employer is determined by the specific subject matter of the individual employment contract. The special regime applicable to computer programs does not operate in an abstract or automatic manner based solely on the status of employee, but in relation to the activities for which the employee was hired, and property rights are transferred to the employer only to the extent that the software creation, or what is generally referred to as a computer program, falls within the scope of the duties established by the contract, the job description or other internal documents defining the content of the employment relationship.

Where the subject matter of the contract explicitly concerns the development of software applications, the writing of source code or related activities, it is reasonable for the programs produced in the performance of these tasks to fall within the scope of the presumption. Conversely, if the developer creates a program that clearly goes beyond the scope of the contractually agreed activities – whether in terms of its subject matter or economic purpose – the application of the presumption becomes questionable. In such situations, the analysis must be carried out on a case-by-case basis, through a systematic interpretation of the contract and the actual duties, and overly general or ambiguous wording may give rise to significant uncertainty regarding the extent of the employer's rights.

---

[17] L. Bently, B. Sherman, *Intellectual Property Law – 6th edition*, Oxford University, 2022, p. 640.

[18] AI Agents in action, white paper, November 2025 – World Economic Forum, p. 11, Available at https://reports.weforum.org/docs/WEF_AI_Agents_in_Action_Foundations_for_Evaluation_and_Governance_2025.pdf, 10 March 2026, 11:00.

A second major limitation relates to the nature of the actual duties and the nature of the developer's contribution. Not every technical activity carried out by an employee constitutes, in and of itself, a protected work— —and not every work created by an employee can automatically be subsumed under their job duties. What is relevant is the degree of creative autonomy the programmer enjoys during the development process. When the programmer carries out precise technical instructions, with no real scope for choice, their contribution may be predominantly of an executive nature. Conversely, when they have the freedom to design the application's architecture, structure the modules, select technical solutions and devise original functionalities, their involvement takes on a clear creative dimension.

This distinction does not override the legal presumption of transfer, provided that the relevant conditions are met, but it does influence how any disputes regarding authorship, the integrity of the work, or the reuse of code snippets in other projects are resolved. Even if economic rights are transferred to the employer within the scope of the employment relationship, the developer retains their status as author and the associated moral rights, and the assessment of their creative contribution remains relevant in any disputes concerning the scope of the creation, the reuse of components or their integration into distinct contexts, including in situations where automated tools or systems such as large language models were used in the development process.

The reuse of components or their integration into different contexts raises specific legal issues, particularly in the field of software development, where code modularity and portability are standard practice. A function, a library, a module or even an architectural structure may be extracted from an initial project and used in another software product, either within the same organisation or in a different context; from a legal perspective, such an operation requires an analysis of the ownership of economic rights over the reused component, as well as any limitations imposed by the author's moral rights.

Where the economic rights to the code have been transferred to the employer under the terms of the employment contract, the employer may, in principle, reuse the component in other internal or commercial projects, provided that such reuse falls within the scope of the rights acquired. However, even in this situation, the programmer's moral rights – in particular the right of authorship and the right to the integrity of the work – remain. Substantial modifications to a reused component or the removal of any reference to the author may give rise to legal disputes, particularly if they affect the identification of the original creative contribution.

The situation becomes more complex when reuse takes place outside the original employment relationship or involves personal projects or external collaborations, including open-source contributions. If a component was created outside the scope of employment or does not fall under the presumption of transfer, the economic rights may remain with the author, and its integration into another project without authorisation may constitute an infringement. At the same time, in the context of the use of automated tools or systems such as large language models, the issue arises of distinguishing between pre-existing, protected elements and sequences generated or adapted subsequently, which requires a careful assessment of the creative contribution and the legal regime applicable to each integrated component.

4.3. Code created during working hours or using the employer's resources, but unrelated to work duties

In the context of employment relationships in the field of software development, the term 'employer's resources' refers to all the material, technical, informational and organisational means made available to the employee for the performance of their duties. This typically includes hardware infrastructure, equipment, servers, workstations, licensed software, access to internal platforms, databases, code repositories, development tools, as well as technical documentation and even organisational know-how.

Even access to large-scale language models or AI agents, for which the employer has a payment commitment or for which they have uploaded a digital library or carried out a training process, falls within this concept. Thus, the employer's resources may also have an informational and strategic dimension, consisting of access to proprietary architectures, internal projects under development, commercial plans or other confidential information relevant to the technical work. In a broad sense, they reflect the material and organisational framework that enables the developer to carry out their professional work. However, the use of these resources, in itself, does not automatically entail the transfer of economic rights over a creation produced by the employee, but constitutes a relevant factor in analysing the link between the creation and the employment relationship.

The decisive factor remains the correlation between the use of resources and the contractual obligations undertaken, not merely the use of the employer's infrastructure.

4.4. The employer's control over the source code. The right to modify the code, to sub-license it and to incorporate it into complex commercial products

One of the most important rights acquired by the employer is the ability to modify the source code by any means. In the field of software, this prerogative is a practical necessity. Programs must be updated, adapted, corrected, made compatible with other systems, or adjusted to market requirements. Without the right to modify the code, economic exploitation would become rigid and, over time, inefficient. For this reason, the right to modify is naturally regarded as an integral part of the transferred economic rights. However, this freedom is not unlimited. Even if the employer can adapt and transform the code, the author – the programmer – remains the holder of moral rights. The right to integrity cannot be ignored, nor can it be interpreted in a way that hinders the technical evolution of the software. In practice, a balance is struck through a functional approach: modifications necessary for exploitation are permissible, provided they do not damage the developer's professional reputation or distort their contribution in a misleading manner.

Equally important is the right to grant sub-licences. Software is rarely used in isolation; it is usually integrated into more complex products or made available to business partners. If the employer is the holder of the economic rights, they must be able to authorise the use of the software by third parties. Of course, the scope of this prerogative depends on the contractual terms and the limits established at the time of transfer. In the absence of express restrictions, sub-licensing is considered a normal consequence of commercial exploitation. However, the integration of the code into larger systems raises other questions. Combining it with other modules, adapting the architecture, or developing successive versions can make it difficult to distinguish individual contributions. From a legal perspective, integration is permitted where economic rights have been transferred, but it cannot justify the attribution of improper authorship or the disregard of the contribution of the developers involved. Even in a complex product resulting from extensive collaboration, authorship remains tied to the individual who made the initial creative contribution.

5. Co-authorship in software creation

5.1. Co-authorship in copyright law. General conceptual overview

The concept of co-authorship occupies a distinct place within the framework of copyright law, having been designed to address situations where a work of intellectual creation is the result of the creative contributions of several individuals. In its classical sense, co-authorship presupposes the existence of a single work, created through the combined efforts of several authors, whose contributions cannot be separated without affecting the unity of the work. The mere participation of several individuals in the creation of a work is not sufficient to qualify as co-authorship; what is essential is the existence of a creative contribution, eligible for copyright protection, and its integration into a final result conceived as a joint work. This theoretical framework, which is relatively stable in the field of traditional artistic creations, becomes problematic, however, when applied to software creation, where the fragmentation of labour, hierarchical organisation and technical constraints profoundly alter the manner in which the work is produced.

In such a case, copyright belongs to the co-authors of the computer program, and it may be exercised only with the consent of all of them. In the case of a computer program composed of divisible code sequences, each author's contribution may also be exercised separately, in relation to each segment of code created[19] . However, in the field of software products, we most often encounter the type of collective work, namely when several programmers work together on a finished product, with specialised expertise sometimes being the criterion for selecting the authors. The contributions are merged in such a way that it is impossible to attribute individual rights[20] .

[19] V. Roș, A. Livădariu, C.P. Romițan, *Intellectual Property Law. Copyright and Related Rights and Industrial Property Rights,* C.H. Beck 2025, p. 99.

[20] *Idem*, p. 100.

5.2. The cumulative conditions of co-authorship: creative contribution and shared intention

Under copyright law, not every intellectual contribution is relevant; only those that reflect the free and creative choices of the person involved are considered so. When applied to software, this condition raises specific difficulties. The development of a program involves a succession of extremely diverse activities: designing the general architecture , writing the source code, testing functionalities, technical documentation, and integration with other systems. Not all of these activities, however, involve a creative contribution within the meaning of copyright law. Thus, a clear distinction must be made between those contributions that reflect one's own intellectual choices, manifested in original solutions, structural options or algorithmic mechanisms, and those activities that are predominantly technical, executory or verification-based in nature. Only the first category is capable of supporting recognition as an author or co-author. Following this logic, a developer who designs a key module of the application or devises an original algorithmic solution participates creatively in the shaping of the work and may be classified, as appropriate, as an author or co-author. Conversely, a person who merely implements pre-established specifications, in the absence of genuine creative freedom, does not meet the necessary conditions for acquiring such status.

The second essential condition of co-authorship is the existence of a shared intention to create a work. This intention need not be expressly stated, but must be apparent from the specific circumstances of the collaboration.

In the case of works created deliberately in collaboration, such as musical or cinematographic works, the existence of a common intention can, as a rule, be identified more easily. In the case of computer programs, however, this aspect is more difficult to ascertain, as the organisation of software projects often involves the involvement of large teams, within which participants work on distinct segments of code, without always having an overall view of the final work and without being in a direct relationship of creative collaboration with the other contributors.

From this perspective, mere participation in the same IT project is not, in itself, sufficient to demonstrate the existence of a shared intention to create a single work. Such an intention must be assessed in concrete terms, by reference to the actual role of each person, the existence of a genuine process of creative coordination, and the extent to which their contribution is substantially integrated into the overall architecture of the programme.

5.3. The Specific Features of Co-authorship in Software Development

Modern software development methodologies, particularly agile ones, have emphasised the fragmentation of contributions and the incremental nature of creation. Source code is often the result of successive iterations, carried out by different people at different times, based on constantly evolving requirements. This dynamic calls into question the traditional applicability of the concept of co-authorship. In many cases, contributions are functionally autonomous, being subsequently integrated into a coherent whole through automated processes or managerial decisions. Legal doctrine has observed that, in such situations, it is difficult to uphold the existence of a work created 'jointly', in the classical sense of copyright.

Another characteristic feature of software is the difficulty in identifying and distinguishing individual contributions. Source code is often modified, refactored or rewritten, and the final version may differ significantly from the initial contributions. This fluidity raises legal issues regarding the determination of co-authorship and the scope of each person's rights. Not every contribution, regardless of its scope, automatically leads to recognition of co-authorship. To produce such a legal effect, the contribution of the person in question must have genuine creative significance and be identifiable, in a recognisable manner, within the structure of the final work. In the absence of these elements, the contribution remains ancillary, lacking the necessary relevance to influence the legal regime of authorship.

5.4. Distinctions required in software practice

One of the most important distinctions in the analysis of co-authorship in software is that between a co-author and a mere contributor. A co-author participates in the creation of the work through creative contributions that influence the structure or expression of the source code. A mere contributor, on the other hand, may play an essential role from a technical or organisational perspective, without making a contribution that is protectable by copyright. This distinction has major legal implications. The co-author acquires copyright in the work, whilst the contributor can only claim, at most, contractual or professional rights. Case law has shown

that ignoring this difference leads to significant disputes, particularly in the context of the commercial exploitation of software.

Recent developments in computer tools capable of generating source code based on instructions formulated in natural or semi-structured language have brought to the fore a new issue with profound implications for copyright theory: can a machine's contribution be classified as authorship or co-authorship? And, correspondingly, how should the legal relationship between the human developer and the automated system used in the software creation process be classified?

The current approach remains, for the most part, focused on rejecting the idea that a machine could be considered an author, regardless of the level of technological autonomy or algorithmic sophistication of the system used. Such an approach stems from a fundamental premise of copyright law, namely that a work is the expression of human intellectual activity, and its original owner can only be a natural person. Consequently, computer systems, including those based on artificial intelligence, cannot be classified as authors or co-authors, as they possess neither legal personality nor the capacity to create within the meaning recognised by legal rules.

In this context, the machine's contribution should not be assessed from the perspective of authorship, but rather from that of its instrumental function. The computer system does not, in the legal sense, create a work, but generates an output following the execution of algorithmic processes configured on the basis of training data, technical parameters and instructions provided by the human user. Even in the event that the result obtained exhibits, from a technical point of view, a certain degree of novelty or complexity, the legally relevant originality cannot be separated from the human intervention that guided, determined or made possible the generation of that result.

Where automated tools are used to generate source code, the human developer remains, from a legal perspective, the sole potential author, provided that they exercise genuine control over the creative process. Control should not be understood in an absolute technical sense, but in a legal sense, as the exercise of relevant intellectual choices.

Following this line of reasoning, the developer may be recognised as the author to the extent that their human intervention remains a decisive factor in the process of generating and configuring the code. Such a conclusion is justified when the person formulates instructions precise enough to guide the outcome, selects or adapts the solutions generated by the system, integrates these results into an autonomously designed architecture, and ultimately decides on the final form of the source code. In these circumstances, the contribution of the computer system must be understood as being of an instrumental nature. From a legal perspective, the use of such a system is akin to the use of advanced technical tools, such as compilers, code generators or standard libraries, which intervene in the process of creating the programme without, in themselves, affecting the authorship of the person using them. What remains decisive is the fact that the human user takes responsibility for the result obtained and plays an essential role in shaping the final expression of the work.

Conversely, if the developer merely accepts, without significant creative input, code generated entirely by the system, the question of authorship becomes more nuanced. Some legal scholars argue that, in such situations, human authorship may be entirely absent, leading to the conclusion that no work protected by copyright exists. This approach is based on the idea that originality cannot be deduced solely from the functionality of the code, but from the intellectual choices of an individual.

From a copyright perspective, the concept of co-authorship implies multiple authors, not multiple tools used in the creative process. However, since a machine cannot be considered an author, the possibility of co-authorship between a human developer and an automated system is logically and legally ruled out.

At this point, attention must also be paid to the conceptual risk of using phrases such as 'human–AI co-creation' or 'hybrid author', which, although they may have descriptive value in technological discourse, are incompatible with the normative structure of copyright law. Accepting such terminology would lead to a dilution of the concept of authorship and to a dangerous confusion between intellectual creation and automated production. Consequently, the machine's contribution can only be classified as a technical contribution, regardless of its complexity. Even in situations where the system generates solutions not anticipated by the user, the absence of intention and creative consciousness excludes any form of authorship or co-authorship. This distinction has direct implications for the analysis of the relationship between developers and employers. In software projects where automatic code generation tools are used, the employer cannot claim the existence of

a 'non-human author' to justify an extension of control over copyright. Ownership of the rights remains with the human developers involved, under the conditions established by law and contract.

At the same time, the use of artificial intelligence does not automatically transform the development team into a group of co-authors. Co-authorship remains dependent solely on the creative input of individuals, and contributions made with the aid of machines must, from a legal perspective, be attributed to those individuals who exercised creative control over the result.

In software projects, roles are strictly defined in functional terms, but not always in legal terms. The developer is, of course, the prime candidate for authorship, insofar as they write the original source code. The tester, on the other hand, although playing a crucial role in ensuring product quality, does not usually contribute to the creative expression of the code and cannot be classified as an author. The software architect occupies an intermediate position. Insofar as they design the general structure of the application and make creative decisions regarding the organisation of the code, their contribution may be eligible for protection. However, if their role is limited to defining functional or technical requirements, without any concrete creative expression, their status as a co-author is debatable. Finally, the product owner has a predominantly managerial and decision-making role, linked to the definition of business requirements. It is unanimously recognised that this role does not normally involve a protectable creative contribution, even if it significantly influences the direction of the project.

In the context of employment relationships, the recognition of co-authorship can have complex legal consequences for both developers and employers. Co-authorship generally entails the joint exercise of economic rights, which may affect the employer's ability to exploit the software without the consent of all co-authors.

6. Multiple contributions and derivative works

One of the defining features of contemporary software is its composite nature. Computer programs are rarely the result of a creation ex nihilo, but are usually built by integrating pre-existing elements: libraries, frameworks, modules, APIs or reusable code snippets.

This technical reality has far-reaching legal implications, as it challenges traditional concepts of copyright, which are centred on the idea of a unified and coherent work created by one or more identifiable authors.

In the field of software, the line between an original work and a derivative work is often difficult to draw. The incorporation of pre-existing code is not, in itself, legally problematic, but it becomes relevant when it influences the structure, functionality or legal status of the final work. In this context, multiple contributions can no longer be analysed solely through the lens of co-authorship, but must be placed within a broader framework that includes the concept of a derivative work and the relationship between different layers of copyright.

The use of open-source code is one of the most widespread practices in software development. From a technical perspective, this enables faster development, lower costs and greater interoperability. From a legal perspective, however, open-source does not equate to the absence of copyright, but rather implies the existence of a specific licensing regime, which governs the use, modification and redistribution of the code. Legal doctrine has consistently emphasised that open-source licences are not mere declarations of intent, but legally binding licence agreements that establish clear limits on exploitation. The integration of open-source code into a commercial project therefore entails an obligation to comply with the conditions imposed by the licence, regardless of the commercial or private status of the resulting project.

This reality creates a first grey area: developers and employers tend to view open-source code as a 'free good', ignoring the fact that the freedom to use it is, in fact, subject to legal conditions.

The integration of open-source components can have a significant impact on the legal status of the final work, depending on the manner, extent and type of licence under which they are used. Some licences permit their incorporation without extensive legal consequences for the application as a whole, whilst others may entail broader obligations, including with regard to the distribution or making available of the source code relating to larger parts of the programme. The issue of integrating open-source components must also be analysed in relation to the concept of a derivative work, since, in certain situations, the integration of an open-source module may influence the legal classification of the final work and, by extension, the licensing conditions that apply to it. However, such a conclusion does not apply automatically, but depends on several factors, including the actual degree of integration of the components, the nature of the interaction between the modules, and the specific terms of the applicable licence.

In practical terms, this issue poses significant risks for the holders of rights to commercial software, particularly in the absence of a clear record of the open-source components used and the legal regime applicable to each of them. In the absence of such a mapping, serious legal difficulties may arise in assessing compliance obligations and the limits within which the code may be used, modified or distributed.

From a technical perspective, plugins and modules are ways of extending the core software, allowing new functionalities to be added without directly modifying the application's core. From a legal perspective, the question that arises is how these components should be classified: whether they should be regarded as separate works or, on the contrary, as integral parts of the main work.

The answer cannot be formulated in the abstract, but requires a concrete assessment of the degree of functional and creative autonomy enjoyed by the plugin or module in question. Where the component can function independently and expresses its own creative choices, it may be treated as a distinct work, even if it was designed to interact with a specific main software programme. Conversely, when the module has no functional autonomy and remains inseparable from the programme into which it is integrated, the legal classification may lead either to the concept of a single work or to that of a derivative work. This distinction is not merely of theoretical interest, but has direct implications for the copyright regime, the possibility of separate licensing, and the control exercised by the owners of the various software components.

In this analysis, APIs occupy a special position, as they serve as communication interfaces between programmes. In principle, the functionality of an API and its logical structure are not, as such, protected by copyright, insofar as they express ideas, principles or methods of operation. However, the specific elements of expression associated with an API, such as its documentation or certain specific implementations, may be eligible for legal protection when they meet the requirement of originality. It is precisely this subtle yet essential distinction that explains the difficulties that frequently arise in practice, particularly in the areas of interoperability and the development of compatible software.

In the analysis of multiple contributions, the distinction between pre-existing code and newly created code is of paramount importance, since the protection afforded by copyright applies only to the original contribution, and not to elements taken as they stand from previous works. In the field of software, this distinction is often difficult to make, partly because of the common practice of reusing code, and partly because of the fragmented structure of programmes, which makes it difficult to identify the exact origin of each component. The mere technical adaptation of pre-existing code is not, in itself, sufficient to give rise to a new protected work; an additional creative contribution is required, capable of reflecting the author's own intellectual choices and conferring an individualised expression on the result. In the absence of such an element, the result obtained may remain within the realm of reproduction or a mere technical adaptation, without acquiring legal autonomy in terms of copyright protection.

The issue takes on particular significance in copyright infringement disputes, where determining the originality of the allegedly newly created code usually requires a detailed analysis of the software's architecture and, not infrequently, the commissioning of complex technical expert reports designed to distinguish between what has been copied and what constitutes a genuine creative contribution.

The concept of licence propagation is used to describe situations in which the incorporation of code subject to a particular licensing regime has legal implications for other components of the software into which it is incorporated. Although the expression is metaphorical in nature, it denotes a genuine legal risk, associated in particular with certain categories of open-source licences, the terms of which may extend beyond the strict scope of the original module used. However, such an effect does not occur automatically; it depends on the specific content of the applicable licence, as well as on the manner in which the code is integrated into the application's structure. For this very reason, ignoring these elements can have significant consequences, including the loss of control over the commercial exploitation of the software or the emergence of unforeseen obligations regarding the disclosure of the source code.

A distinct but closely related difficulty concerns the legal compatibility of different licences. In practice, software applications frequently combine components subject to different licensing regimes, which raises the question of whether they can legally coexist within the same product. In certain situations, incompatibility between licences may make it impossible to legally distribute the programme as a whole, even if each component, considered separately, has been used in accordance with its own licensing terms. This reality highlights the fact that software architecture must be assessed not only from a technical perspective, but also

through a prior legal analysis capable of identifying in advance any potential contradictions between the regimes applicable to the integrated components.

The entire analysis leads to one key conclusion: in the context of multiple contributions and derivative works, ownership of software can no longer be conceived as an absolute and indivisible right, but rather as a layered set of rights and obligations. Ultimate control over the source code is often limited by the rights of other authors, by licensing conditions and by the derivative nature of the work. Clearly, this fragmentation of rights is one of the defining characteristics of modern software creation and requires a sophisticated legal approach, combining copyright law with contractual analysis and sound intellectual property governance.

7. Concluding remarks

An analysis of ownership and control over source code reveals that the law applicable to computer programs can no longer be understood solely by reference to the traditional models of the individual author and the work exploited in a linear manner. In today's technological reality, source code is created, circulated and exploited within a far more complex legal and economic ecosystem, in which human contributions, organisational structures, automation, the use of large-scale language models, as well as the reuse of components and licensing interdependencies create a stratification of rights and risks. It is precisely this stratification that means the issue of software ownership can no longer be treated as a simple matter of formal title, but rather as a question of the legal governance of digital creation.

In this context, one of the key ideas that emerges is that the concept of control has a broader meaning than simply holding economic rights. Effective control over the source code entails the actual ability to decide on its modification, integration, distribution, licensing, reuse and commercial exploitation. However, this ability may be affected not only by copyright law in the strict sense, but also by the contractual relationships between the parties, the limits of moral rights, the existence of prior contributions, the inclusion of open-source components, and the compatibility of the relevant licences. Consequently, legal control over software must be understood as the result of a cumulative normative and contractual framework, not as the automatic effect of a mere legal presumption or of a dominant economic position within the project.

At the same time, the research confirms that copyright remains, in essence, tied to an anthropocentric conception of creation. Even in a context where software development is aided by automated systems, including large-scale language models, authorship cannot be separated from the intellectual input of a natural person. This means that the central legal issue is not to identify a supposed creative personality of the machine, but to determine whether and to what extent human contribution retains the character of free and creative choices capable of underpinning copyright protection. From this perspective, the uncritical extension of technological vocabulary into the realm of legal concepts risks undermining the internal coherence of copyright law. A clear distinction between instrument, technical medium and author is therefore indispensable.

A further concluding remark concerns the relationship between the legal protection of source code and the economic functionality of software. The specific nature of computer programs requires a nuanced interpretation of the traditional institutions of copyright law. In this regard, the protection of expression cannot lead to the monopolisation of ideas, functionalities, logical principles or technical solutions in themselves. Therefore, the distinction between what falls within the scope of protected expression and what remains in the public domain is crucial not only for the author and the employer, but also for competition, interoperability, innovation and the circulation of technical knowledge. Without this distinction, copyright would risk becoming a barrier, incompatible with the demands of technological development.

In practical terms, the article also draws a legal and organisational policy conclusion: most conflicts concerning source code do not arise from the complete absence of rules, but from the inadequacy of contractual and internal safeguards. Disputes are frequently caused by vague wording regarding job responsibilities, the lack of policies on side projects, the absence of rules on the use of the employer's resources, the lack of clarity regarding the regime for reused components, and a failure to recognise the impact of open-source licences or AI tools on the development process. In this regard, good intellectual property governance in the software sector requires not only knowledge of copyright law, but also the establishment of clear organisational practices capable of documenting contributions, defining rights, and preventing the overlap of incompatible legal regimes.

In particular, the open-source component and the issue of licence compatibility demonstrate that ownership of software can no longer be conceived within the simplified paradigm of an exclusive, unitary and

absolute right. When code is built from successive layers of contributions, some proprietary and others open-source, legal control becomes fragmented. The risks associated with so-called licence 'contamination' or incompatibility between licences demonstrate that legal analysis must accompany the technical architecture right from the product design phase. To ignore this aspect is to shift the problem from the planning stage to the litigation stage or to the impossibility of subsequent commercial exploitation.

Ultimately, the study shows that the issue of ownership and control over source code should not be treated solely as a debate about ownership, but rather as one about balance. It concerns the balance between the developer's creative contribution and the organisation's legitimate interest in monetising the software product; between moral rights and the need for continuous modification of the code; between the protection of expression and the freedom to use ideas; between contractual autonomy and the limits imposed by the special legal regime governing computer programs; between innovation and legal certainty. Only by maintaining this balance can a regulatory framework be ensured that is appropriate for the contemporary digital economy.

8. Conclusions

This analysis has sought to clarify the legal regime governing ownership and control of source code, based on the premise that software is both a protected intellectual creation and a strategic economic asset. It is in this dual status that the main legal difficulty lies: copyright continues to be structured around the human author and original expression, whilst the practice of software development is dominated by collective processes, corporate organisation, the reuse of components, automation and the pressure for rapid commercial exploitation. The article has demonstrated that these two dimensions are not mutually exclusive, nor do they overlap perfectly; they must be harmonised through a careful interpretation of the rules and rigorous contractual discipline.

A key conclusion is that authorship, in the sense of copyright, remains inseparable from the natural person. The programmer is the only person who can originally hold the status of author of the source code, provided that their contribution meets the requirement of originality. A legal person may become the holder of economic rights, including under the special rules applicable to employment relationships, but cannot be an author in the legal sense. Similarly, the use of artificial intelligence systems does not justify abandoning this premise; it merely requires a more detailed analysis of the actual human contribution and the limits of protection when the result is generated automatically.

A second conclusion concerns the decisive nature of the distinction between the protected expression and the unprotected elements of software. Source code may constitute a protected work insofar as it reflects free and creative choices, but ideas, principles, functionalities and technical logic generally remain outside the scope of copyright. This distinction is essential for maintaining the balance between the protection of the author and the freedom of technological development. Without it, copyright would risk extending beyond the realm of expression and unduly affecting competition, interoperability and innovation.

A third conclusion concerns the relationship between the developer and the employer. The legal framework governing computer programs justifies the transfer of economic rights to the employer where the work is created in the course of the employee's duties; however, this approach must not be interpreted as an unlimited presumption that the employer automatically acquires all rights to the employee's creative output. The application of the rule depends on the specific subject matter of the contract, the nature of the actual duties and the genuine link between the creation and the tasks undertaken. Code created outside this scope, even if it draws on the developer's professional experience or is produced in a context closely related to their professional activity, cannot be automatically attributed to the employer. Hence the vital importance of the job description, intellectual property clauses and internal policies regarding side projects and the reuse of components.

At the same time, the article confirms that the concept of legal control over software goes beyond the issue of formal ownership of economic rights. Effective control is influenced by the author's moral rights, the existence of multiple contributions, the classification of certain developments as derivative works, the use of pre-existing components, and the licensing structure of the integrated code. In this context, control over the source code appears as a relational and functional concept, not as an absolute prerogative.

Finally, one of the most important practical conclusions of the study is that modern software development requires a shift from a static view of intellectual property to a dynamic one, focused on the legal management of the code's life cycle. Such an approach involves identifying contributions, ensuring component traceability, verifying licence compatibility, regulating the use of AI tools, distinguishing between projects undertaken outside

the scope of employment, and anticipating how the code may subsequently be exploited, adapted or transferred. In the absence of such legal management, conflict between the author, employer, collaborators and third parties becomes not an exception, but a predictable consequence of contractual and organisational opacity.

Consequently, ownership and control of source code must be understood as the expression of a complex legal regime, situated at the intersection of intellectual property protection, the organisation of work, investment logic and the technological infrastructure of the digital economy. In this regard, viable legal solutions can be neither exclusively formalistic nor exclusively technical. They must remain faithful to the principles of copyright law, yet sufficiently flexible to respond to the realities of contemporary software development. From this perspective, the challenge is not merely to establish the right holder, but to establish a balanced framework of recognition, exploitation and responsibility, capable of protecting human creativity without stifling innovation and without undermining the coherence of the legal system.

References

- Legislation and regulatory acts
- Regulation No 1689/2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828, published in the Official Journal of the European Union of 12 July 2024;
- Regulation No 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of individuals with regard to the processing of personal data and on the free movement of such data and repealing Directive 95/46/EC (General Data Protection Regulation) https://eur-lex.europa.eu/legal-content/RO/TXT/PDF/?uri=CELEX:32016R0679;
- Law No 8/1996 on copyright and related rights, republished in the Official Gazette, Part I No 489 of 14 June 2018, with subsequent amendments and additions;
- I. Bainbridge, Intellectual Property, 8th Edition, Pearson Education, 2010
- Bently, B. Sherman, Intellectual Property Law – Third edition, Oxford University, 2009
- Davis, P. Samuleson, M. Kapor, J. Reichman, A new view of Intellectual property and software, Communications of the ACM, March 1996, vol. 39, no. 3, p. 23, available at https://people.csail.mit.edu/davis/cacm96.pdf, accessed 02.03.2026, 20:00;
- Ohm and B. Reid, 'Regulating Software When Everything Has Software', p. 4, The George Washington Law Review, 2016,
- Petzold, *CODE: The Hidden Language of Computer Hardware and Software*, 2nd edition, Pearson Education, 2023, p. 42, available at https://www.gwlr.org/wp-content/uploads/2016/11/84-Geo.-Wash.-L.-Rev.-1672.pdf, accessed 05.02.2026, 11:30
- Roș, A. Livădariu, C.P. Romițan, Intellectual Property Law. Copyright and Related Rights and Industrial Property Rights, C.H. Beck 2025;
- Terms of Use _ OpenAI - Available at https://openai.com/ro-RO/policies/row-terms-of-use/
- BGH case, judgement of 3 May 2005 – I ZR 111/02 – Fash 2000, article available at https://www.glawe.de/en/european-court-of-justice-cjeu-on-protection-of-software-functionalities, accessed 3 March 2026, 12:00
- C-604/10, Football Dataco Available at https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A62010CJ0604_SUM&utm_source=chatgpt.com: