Page 652 - 664 **DOI:** *https://doi.org/10.63108/IJLMH.12815*

INTERNATIONAL JOURNAL OF LAW MANAGEMENT & HUMANITIES

[ISSN 2581-5369]

Volume 9 | Issue 5

2026

The Merger Doctrine as an Ex-Ante Solution: Reclaiming Software Interoperability from the 'Fair Use Trap'

AISHWARYA SINGH* AND UTKARSH JAISWAL**

ABSTRACT

A decade long court battle in Oracle v. Google *enunciating the copyright mechanism of controlling software interfaces failed to answer specific questions on software interoperability. The case has sparked a debate on the copyrightability of Application Programming Interfaces, or APIs, throughout the world. The American courts that were meant to settle the dispute have, in turn, converted the dispute into ex post litigation. They have decided on classifying the reuse of functional interfaces as a probable infringement, and condemned interoperability into the "Fair Use Trap." This imposition places burdens of disproportionality on smaller developers, as interoperability is not only a technical requirement but also a legal one. This article illustrates how the Federal Circuit's intervention deviated from the Ninth Circuit's well-established copyright foundations by bringing patent-style reasoning into a field ill-suited for monopoly-based analysis, through a close examination of the procedural trajectory in* Oracle v. Google*. This article argues that the fair use paradigm is organisationally incompetent to regulate functional aspects which exist as technical standards. The implication of treating APIs as expressive property is that APIs serve an infrastructural purpose in facilitating competition. Therefore, it advocates for the merger doctrine as a superior ex ante solution. The merger doctrine ensures predictability of market, protects competition, and brings software copyright back to its constitutional purpose; promoting technology instead of creating proprietary lock-in.*

Keywords: *Copyright, Interoperability, Merger Doctrine, Fair Use, Software Interfaces, Oracle v. Google*

I. INTRODUCTION

Software copyright disputes rarely trigger conventional questions on creativity. Instead, they arise from conflicts over compatibility, reuse, and the ability of independently developed programs to communicate within shared technical ecosystems. These conflicts place unique strain on copyright doctrine, which was not designed to regulate functional interfaces that

* Author is a Student at National University of Study and Research in Law, Ranchi, Jharkhand, India. ORCID: https://orcid.org/0009-0004-5795-3459

** Author is a Student at National University of Study and Research in Law, Ranchi, Jharkhand, India. ORCID: https://orcid.org/0009-0004-6126-1834

 [ISSN 2581-5369]

operate as industry standards. As software development increasingly depends on interoperability, the traditional balance between protection and access is no longer theoretical but structurally contested.

Copyright law operates as both a grant of rights and a structural restraint.[1] Software is cumulative, interoperable, and functional unlike the traditional literary or artistic works.[2] It is not just to be read, but to be used, stretched out, and linked. Opposing the common intuition that protection of copyright is to be extended to the maximum in this domain, it will not necessarily serve to reward authorship; it will also run the risk of adding technical standards to the repertoire of chokepoints owned privately.

This tension was revealed in the litigation in *Oracle v. Google*.[3] What started as a clash over a shared resource of Application Programming Interfaces (APIs) on Java, turned into a decade long battle on the legality of the software interfaces themselves. It was not only the question of whether a certain use was permissible, but whether the fundamental vocabulary, on which programs are built, could be subject to copyright at all.[4] The ramifications of the answer to that issue go well beyond the parties involved.

This tension was enhanced by the nature of the procedure that the dispute took. The litigation history was novel in terms of copyright analysis. This controversy began under the copyright law of the Ninth Circuit, but the Federal Circuit's interference caused the jurisdictional change. Nonetheless it affected the legal framework applied to the case and moved the inquiry away from traditional copyright principles, with broader implications for the treatment of software copyright in contemporary law.

The resulting legal ambiguity places a burden on contemporary software markets that they are poorly equipped to bear. Software development requires a high degree of technical predictability, however the current framework introduces ambiguity for the developers regarding the potential reuse of functional interfaces.

In this article, the vagaries brought out by *Oracle v. Google* are not discussed as an exception but rather as a symptom of a basic doctrinal imbalance within the judicial conception of software interfaces. The structural constraints of copyright have not been considered through the idea-

[1] U.S. CONST. art. I, § 8, cl. 8.
[2] "The unique nature of software as a cumulative, interoperable and functional medium has long been recognised as a departure from traditional copyright subjects." *See generally* Pamela Samuelson, Randall Davis, Mitchell D. Kapor & J.H. Reichman, *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308, 2310 (1994).
[3] *Oracle Am., Inc. v. Google Inc.*, 872 F. Supp. 2d 974 (N.D. Cal. 2012).
[4] *Id.*

 [ISSN 2581-5369]

expression distinction and the merger doctrine, but courts have provided the by-default protection to functional elements through the fair use doctrine.

II. The Anatomy of Interoperability

> "Any fool can write code that a computer can understand. Good programmers write code that humans can understand." – Martin Fowler.

This quote can be broken down into two parts. The first part is referring to the "functional" code and the latter is referring to the "expressive" code. While Fowler gave the quote to underscore the uniqueness of a good programmer, we will use this quote to set the foundation for analysing the API conundrum. In 2021, the US Supreme Court finally decided the landmark case of *Google LLC v. Oracle America, Inc.*, famously quoted as the 'Copyright Case of the Century.'[5] It addressed the perplexing issue of defining Application Programming Interfaces as non-copyrightable. But first, what is an API and why is it relevant in the realm of Intellectual Property Law?

For a layman, an API can be too technical to make sense of. By definition it means a set of rules or protocols that enable software applications to communicate with each other to exchange data, features or functionality.[6] Precisely, it is the code that governs the access points for the server. Therefore, it has a functional use. So, when does it have a creative use? In *Google v. Oracle*, Oracle argued that it spent years developing a programming library which successfully attracted software developers, enhancing the value of Oracle's products.[7] Their API library was a vast creative taxonomy that required hundreds of idiosyncratic design choices and diligent human effort, thus constituting an original literary work. In contrast, Google argued that it only copied the "structure, sequence and organisation" (or SSO) of Java's API to develop earlier versions of Android OS, that is, that its copying of declaring code was a functional necessity.[8]

There's a fine line between the distinction made here. Let's understand it with the help of an illustration. Imagine a world where a restaurant could copyright not just its recipes, but the very name of the dishes on the menu.[9] A chef at a competing bistro would be legally barred from

---

[5] *Google LLC v. Oracle Am., Inc.*, 593 U.S. 1 (2021).

[6] Michael Goodwin, *What Is an API (Application Programming Interface)?*, IBM (Apr. 9, 2024), https://www.ibm.com/think/topics/api.

[7] *Google*, 593 U.S. at 43 (Thomas, J., dissenting).

[8] *Oracle Am., Inc. v. Google Inc.*, 750 F.3d 1339, 1354–61, 1368–72 (Fed. Cir. 2014).

[9] *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 49 F.3d 807, 815 (1st Cir. 1995), *aff'd by an equally divided Court*, 516 U.S. 233 (1996). "We think that 'method of operation,' as that term is used in § 102(b), refers to the means by which a person operates something, whether it be a car, a food processor, or a computer. Thus a text describing how to operate something would not extend copyright protection to the method of operation itself; other people would be free to employ that method and to describe it in their own words. Similarly, if a new method of operation is used rather than described, other people would still be free to employ or describe that method."

 [ISSN 2581-5369]

serving 'Tomato Soup' or 'Grilled Cheese' simply because a predecessor claimed ownership over the labels. This is the API conundrum at the heart of *Google v. Oracle*. There is a dispute over whether the linguistic menus of software, the labels that tell a machine what to do, can be owned as literary works or whether they are merely the functional buttons of a machine that must remain free for all to press.

A. Interoperability as a Technical Necessity

In the software ecosystems, interoperability allows each system on a network to communicate with its peers to share, exchange, combine, and use data. Each interoperable system can also allow the other systems to read, update, modify, and analyse that data with minimal human interaction.[10] Application Programming Interfaces (APIs) serve as the primary mechanism through which interoperability is achieved. APIs can be found in cell phones, laptops, computers, and even television applications such as Netflix, Hulu+, and Amazon Instant.[11] They are useful because they can prevent a software programme from wasting time by having to rewrite existing code.[12] They supply programmers with pre-packaged declarations to write programming that is compatible with other applications.[13] As a result, it reduces redundancy, improves efficiency and enables different software components to operate cohesively within a shared technological environment.

Interoperability is a requirement for competition, and innovation rather than a vague ideal of software markets.[14] This technical necessity is further reinforced by the structure of modern software platforms. Most applications do not operate in isolation but exist within broader ecosystems, such as Android, iOS or Windows, that provide foundational services like memory management and networking. Developers are, therefore, platform-dependent, relying on the underlying system's pre-written code and interfaces to perform essential functions, which is one of the reasons why Google resorted to copying 37 API packages of Java. This eliminated the need for programmers to learn a whole new language for developing apps for Android.[15] Further, it exponentially increased the value of Android platforms as more users and developers adopt

---

[10] Margaret Lindquist, *What Is Interoperability?*, Oracle (May 20, 2024), https://www.oracle.com/in/interoperability/.

[11] Brian Proffitt, *What APIs Are and Why They're Important*, ReadWrite (Sept. 19, 2013), https://readwrite.com/api-defined/.

[12] *Oracle Am.*, 872 F. Supp. 2d at 982.

[13] *Id.*

[14] "The success of software products strongly depends on network effects, and thus on interoperability, the degree to which the products are able to interact with other computer systems or software in an increasingly interconnected computing landscape." Suzanne Van Arsdale & Cody Venzke, *Predatory Innovation in Software Markets*, 29 HARV. J.L. & TECH. 243, 261 (2015), https://jolt.law.harvard.edu/articles/pdf/v29/29HarvJLTech243.pdf.

[15] *Google*, 593 U.S. at 8, 13–14.

it. Developers are attracted to large user bases with an established set of tools, libraries and conventions.

B. APIs as Reusable Interfaces

A declaring code functions as the interface. It consists of the specific names, commands and parameters required to invoke a particular function. Thus, it operates as a form of shorthand, enabling developers to trigger complex computational processes through a simple, recognisable command.[16] It provides the link between the method call and implementing code.[17] In contrast, implementing code constitutes the underlying logic that performs the task. It is the detailed, step-by-step instruction set that determines how a function is executed. This is where a developer's technical skill, efficiency choices and creative problem-solving are most clearly expressed.

APIs are deliberately designed so that their interfaces can be reused.[18] For third party developers to create applications that interact seamlessly with a platform or with other software, they must use the exact declaring code that the system recognises. Altering the names, structures or parameters of these declarations would defeat the very purpose of an interface, rendering the software incompatible. Therefore, reuse of interfaces is not a form of appropriation but a functional necessity that allows diverse programs to plug into shared systems without necessitating wholesale duplication of the underlying infrastructure.

C. The Myth of Unlimited Design Choice

The fact that the developers of software are assumed to be given total freedom to design APIs ignores the process of technical lock-in. Technical lock-in happens when a certain interface becomes so popular that it becomes virtually an industry standard.[19] When an API has become learned and entrenched in the practices of millions of developers, its structure stops being a question of discretionary design, and becomes the functional necessity of a software.

In such circumstances, deviation is not a smart option. If a developer seeking interoperability were to use different labels, commands or organisational structures, their software would fail to

---

[16] *Google*, 593 U.S. at 43–44 (Thomas, J., dissenting). "To save space and time, legislatures define terms and then use those definitions as a shorthand. For example, the legal definition for 'refugee' is more than 300 words long. 8 U.S.C. § 1101(42). Rather than repeat all those words every time they are relevant, the U.S. Code encapsulates them all with a single term that it then inserts into each relevant section. Java methods work similarly. Once a method has been defined, a developer need only type a few characters (the method name and relevant inputs) to invoke everything contained in the subprogram."

[17] *Google*, 593 U.S. at 10–11.

[18] Brad A. Myers & Jeffrey Stylos, *Improving API Usability*, 59 COMM. ACM 62 (2016), https://cacm.acm.org/research/improving-api-usability/, https://doi.org/10.1145/2896587.

[19] W. Brian Arthur, *Competing Technologies, Increasing Returns, and Lock-In by Historical Events*, 99 ECON. J. 116 (1989), https://doi.org/10.2307/2234208.

 [ISSN 2581-5369]

communicate with the existing ecosystem. Compatibility would be lost due to the lack of shared vocabulary required for interaction. The result is exclusion from the market rather than innovation within it.

From this reality, emerges the concept of constrained expression. Unlike literary or artistic works, where ideas can be expressed in countless forms, software interfaces often permit only one, or a limited number of ways to express a particular function if interoperability is to be preserved. Where expression is dictated by functional requirements and compatibility goals, the traditional distinction between idea and expression begins to collapse. Ultimately, developers have to face what amounts to a tax on innovation,[20] a gruelling and expensive process of clean-room engineering that small startups cannot afford.

As Lemley and Samuelson observe, "the terms 'compatible/compatibility' and 'interoperable/interoperability' were rarely mentioned in the *Google* decision."[21] Instead the term interface appeared occasionally, only to define declaring code as one in providing a "user interface" between human and the machine. It is fair to read the decision as having recognised "interoperability of humans with software, such that programmers could continue to use declarations that they already knew when writing apps for the Android platform as well as for Java-compliant platforms."[22]

D. Courts' Decisions in Existing Cases

What *Oracle v. Google* brought out, was the less known debate over the ability to interoperate without having to face legal backlash for software companies. This question has loomed over the US jurisprudence for decades. In *SAS Institute, Inc. v. World Programming Ltd.*, the court's declining to settle the core question of the copyrightability of interfaces illustrates how legal barriers can entrench 'lock-in',[23] while *Synopsys* serves as a cautionary tale of the 'innovation tax,' where the defendant was driven into bankruptcy after a copyright verdict returned against it for attempting to navigate a shared technological landscape through interoperability.[24] *Cisco v. Arista* provided a refreshing change of direction towards the realisation of technical necessity by presenting the *scènes à faire* doctrine,[25] which recognised that there are some parts of an

[20] JAMES BESSEN & MICHAEL J. MEURER, PATENT FAILURE: HOW JUDGES, BUREAUCRATS, AND LAWYERS PUT INNOVATORS AT RISK 22–23 (Princeton Univ. Press 2008).
[21] Mark A. Lemley & Pamela Samuelson, *Interfaces and Interoperability After Google v. Oracle*, 100 TEX. L. REV. 1 (2021), https://texaslawreview.org/interfaces-and-interoperability-after-google-v-oracle/.
[22] *Id.* at 39.
[23] *SAS Inst., Inc. v. World Programming Ltd.*, 64 F.4th 1319 (Fed. Cir. 2023) (No. 21-1542, decided Apr. 6, 2023).
[24] *Synopsys, Inc. v. ATopTech, Inc.*, No. 13-cv-02965-MMC, 2016 WL 6782028 (N.D. Cal. Nov. 16, 2016), published at 319 F.R.D. 293; *see also In re ATopTech, Inc.*, No. 17-10111 (Bankr. D. Del. filed Jan. 13, 2017).
[25] "Certain scenes or situations are inherently necessary to a form of expression, and thus cannot be copyrighted independently of the work as a whole." Gavin M. Strube, *Gotham Skylines: The Intersection of Scènes à Faire and*

interface which, through mechanical necessity, or standard industry practice, should be left free to be reused, lest the market should be shut out.[26] That reprieve, however, was never consolidated on appeal. Cisco's challenge to the verdict was argued before the Federal Circuit in June 2018 but was never decided: the parties settled in August 2018, and the judgment was vacated on their joint motion in October 2018, leaving the district court's treatment of *scènes à faire* without precedential force.[27]

These cases demonstrate that even now, the US courts have treated interoperability concerns in the context of software copyright cases in an inconsistent manner, with many cases having been litigated by relying on more limited doctrinal instruments without tackling the structural significance of interoperability in the context of the contemporary software system.

The technical reality of limited expressions and mechanical need implies that functional interfaces ought to be positioned beyond the circumference of copyright protection to make sure that there is rivalry on the market. However, instead of addressing this at the threshold of copyrightability, the American legal system has largely funnelled the problem into a post-infringement framework. By failing to provide ex ante clarity on whether these 'bridges' can be owned, the courts have forced developers to rely on a reactive safety valve: the fair use doctrine. This shift from structural exclusion to a case-by-case defence creates a trap, where the fundamental right to interoperate is transformed into an unpredictable legal gamble.

III. THE FAIR USE TRAP

Fair use logic was first introduced in *Folsom v. Marsh*, where the court noted that in deciding questions of copyright infringement, the deciding court must "look to the nature and objects of the selections made, the quantity and value of the materials used, and the degree in which the use may prejudice the sale, or diminish the profits, or supersede the objects, of the original work."[28] However, one needs to note that fair use is an affirmative defence and not a right.[29] It is notoriously expensive, fact-specific and unpredictable; factors that explain why the Supreme Court refrained from delivering a widely applicable ruling and instead granted an escape to

---

Fictional Facts in Comic Books, Graphic Novels, and Their Derivative Works, 5 PACE INTELL. PROP. SPORTS & ENT. L.F. 54, 64 (2015), https://digitalcommons.pace.edu/pipself/vol5/iss1/5/; *see generally* Leslie A. Kurtz, *Copyright: The Scenes a Faire Doctrine*, 41 FLA. L. REV. 79 (1989) (for a more in-depth overview of the *scènes à faire* doctrine).

[26] *Cisco Sys., Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344-BLF (N.D. Cal. filed Dec. 5, 2014) (ECF No. 1).

[27] *Arista Networks, Inc. v. Cisco Systems, Inc.*, 904 F.3d 981 (Fed. Cir. 2018); *cf. Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 2017-2145 (Fed. Cir. argued June 6, 2018) (appeal from the *scènes à faire* verdict; settled Aug. 6, 2018 before decision, judgment vacated on the parties' joint motion in Oct. 2018).

[28] *Folsom v. Marsh*, 9 F. Cas. 342, 348 (C.C.D. Mass. 1841) (No. 4,901).

[29] 17 U.S.C. § 107 (2018); *Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569, 590 (1994) ("Since fair use is an affirmative defense, its proponent would have difficulty carrying the burden of demonstrating fair use without favorable evidence about relevant markets.").

Google on grounds of fair use, exclusively. Nonetheless, by enforcing the *Oracle* case into a "redundant fair use retrial," the Federal Circuit ensured that only companies with Google sized bank accounts could afford to fight for interoperability.

In glaring contrast, the Ninth Circuit has always advocated an open-door policy, meaning interoperability should not be locked in to allow the emergence of monopolies and restrict competition. In instances such as 'reverse engineering' a copyrighted object code when it's the only way to access the unprotected functional elements of code, or creating a software that is designed to interoperate with a copyrighted work, or copying declaring code for interoperability purposes, the Ninth Circuit has consistently held these practices as fair use.[30] While the term 'fair use' is used to describe these practices, it does not automatically attract the fair use defence since there is no "infringement" to begin with when copying functional interfaces.

Universal's argument in the *Sony* case appropriately explains the idea. "Productive uses bring a new work, such as a critical review that quotes from a copyrighted novel, into the world, thereby adding to the corpus of knowledge, which furthers the ultimate purpose of copyright law."[31] "An intrinsic use merely consumes the work in the same way as if a copy had been purchased."[32]

To sum up, the Federal Circuit's procedural manoeuvre has effectively created a legal grey zone that jeopardises the future of software development. By refusing to settle the question of copyrightability, the Supreme Court has allowed a "lock" to remain on the fundamental building blocks of software, with a specialised patent court holding the key to a door that the Ninth Circuit had intended to keep wide open for the sake of competition.

The Fair Use Trap serves as a definitive procedural bait-and-switch. While the Ninth Circuit's precedents offered developers prophylactic immunity by ruling that functional interfaces were never copyrightable to begin with, the Federal Circuit's reversal stripped away this armour. The court did not simply make the law harder by reclassifying APIs as protectable expression; it put the entire industry in an affirmative defence quagmire into which a developer is virtually guilty of infringement until a decade of exhausting legal litigation confirms the use to be fair. Due to this flaw, interoperability is no longer considered a typical technical standard, or a '*lingua franca*,' but instead a luxury that only can be afforded by those rich enough to spend money to undergo an unfruitful fair use lawsuit.

---

[30] *Sega Enters. Ltd. v. Accolade, Inc.*, 977 F.2d 1510 (9th Cir. 1992), reaffirmed in *Sony Computer Entertainment, Inc. v. Connectix Corp.*, 203 F.3d 596 (9th Cir. 2000).

[31] Pamela Samuelson, *Fair Use for Computer Programs and Other Copyrightable Works in Digital Form: The Implications of Sony, Galoob and Sega*, 1 J. INTELL. PROP. L. 49, 61 (1993), https://digitalcommons.law.uga.edu/jipl/vol1/iss1/6.

[32] *Id.*

 [ISSN 2581-5369]

IV. Is the Merger Doctrine a Way Out?

To safeguard only the expression rather than concepts, operational procedures or functional limitations is the basic premise of copyright law.[33] Such limitation is essential for keeping copyright from being used as a means of monopolising knowledge or technique. The difference between expression and idea is then not just a conceptual difference, but is structural in nature, working as a protection against over-protection. A more straightforward example of this protection is the doctrine of merger. It is based on the premise that if the expression of an idea is only possible in one or very limited number of forms, the expression 'merges' with the idea and is no longer protectable.[34] To give copyright in such a situation would practically mean granting a de facto monopoly to the idea itself and would violate the goals of copyright law. Merger can be said to have its doctrinal root in the case of *Baker v. Selden*,[35] where the US Supreme Court held that even though copyright protection was available to the explanatory part, it cannot be used to regulate the functional process.

This principle has also been reflected in Indian copyright jurisprudence in the case of *R.G. Anand v. Delux Films*,[36] where the Supreme Court restated that a copyright is only infringed when there is a similarity in the means of expression, and not the idea or theme. This argument implicitly acknowledges that in cases where expression is defined by the idea behind it, protection should not be given.

Software interfaces present a textbook scenario for the application of this doctrine. Some of its aspects, such as the declaring code of APIs in this case, are not based on the creative choice, but on the technical and compatibility needs. When an interface is established as an industry standard or a necessary bridge to a popular platform, the only real choice that the developers seeking interoperability have, is to replicate them,[37] and any deviation will result in a failure to communicate, setting aside the separability of the form of expression and its function. Giving such elements the protection of copyright transforms copyright into a gatekeeping device from an incentive device. This creates a profound chilling effect. It enables the rightsholder to control

---

[33] "To this end, copyright assures authors the right to their original expression, but encourages others to build freely upon the ideas and information conveyed by a work. . . . This principle, known as the idea/expression or fact/expression dichotomy, applies to all works of authorship." *Feist Publ'ns, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 349–50 (1991).

[34] Anuttama Ghose & S.M. Aamir Ali, *The Principle of Idea-Expression Dichotomy in Copyright Laws: Legal Scenario in India Compared to the Laws of U.S.A. and United Kingdom*, 7 JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH 594, 600 (2020), https://www.jetir.org/papers/JETIR2007070.pdf.

[35] *Baker v. Selden*, 101 U.S. 99 (1879).

[36] *R.G. Anand v. Delux Films*, (1978) 4 SCC 118, AIR 1978 SC 1613.

[37] Mark A. Lemley & David W. O'Brien, *Encouraging Software Reuse*, 49 STAN. L. REV. 255 (1997), https://doi.org/10.2307/1229298.

 [ISSN 2581-5369]

the technical standard, which other people are bound to use, thereby distorting competition and adding barriers to entry. The choice remaining to them will be either to forgo interoperability, and thus viable competition, or pay a tax on innovation that only the largest incumbents can afford.

Resolving these disputes under the umbrella of fair use doctrine is structurally unsound. Fair use is an equitable, post-infringement defence involving a complex, fact-specific balancing test.[38] Although it is useful in situations that relate to expressive works, this framework is ill-suited in circumstances that relate with the functional interfaces that act as infrastructural elements of the digital economy. Exposing interoperability to an affirmative defence regime creates legal ambiguity where predictability is essential. It may provide a safety valve for giants like Google, but does not offer any ex ante clarity for the industry at large.

In contrast, the merger doctrine provides a more coherent alternative by preventing the functional interfaces from reaching the protected domain altogether as it deals with the copyrightability at the threshold level. This does not remove copyright protection of the software, but creates a coherent 'thin copyright' regime which simply limits the protection to those elements where creative choices remain meaningful, namely the implementing code.[39]

By acting as a structural limitation rather than a procedural exception, the merger doctrine returns copyright to its original intent. It guarantees that creativity is protected by copyright without allowing control over the functional mechanisms that underpin technological advancement.

V. Fair Use and Ex-Ante Clarity: Why Rigidity Works in Functional Copyright

There is more to the instability around software copyright than merely doctrinal ambiguity. One of the primary issues is a misaligned institutional design. *Oracle v. Google* reveals a fundamental flaw in the decision-making and distribution of copyright cases that involve software in the American court. When a Ninth Circuit copyright case was routed through the Federal Circuit, a court specialised in patents, instead of resolving a functional compatibility through the copyright's inherent limitations, the appellate structure exacerbated the confusion,

[38] Alex R. Figares, Daniel P. Fernandez & H. Wayne Cecil, *Copyright Infringement and the Fair Use Defense: Navigating the Legal Maze*, 27 U. Fla. J.L. & Pub. Pol'y 135 (2016), https://scholarship.law.ufl.edu/jlpp/vol27/iss1/5. "The four statutory Fair Use factors: (1) the purpose and character of the use, including whether such use is of a commercial nature or is for non-profit educational purposes; (2) the nature of the copyrighted work; (3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and (4) the effect of the use upon the potential market for or value of the copyrighted work."
[39] *Sega*, 977 F.2d at 1524.

as a court whose docket is built on limited monopolies for functional inventions was entrusted with interpreting fundamental questions of copyright law.

Historically, unlike patents, copyright law has been formed on the basis of regional appellate interpretation.[40] And because of this decentralised evolution, copyright doctrine has been able to be context dependent especially where the technology and expression overlap. However, software copyright issues are increasingly being funnelled into a judicial structure known for being more focused on patent uniformity, than copyright intricacies. This misalignment was evident in *Oracle v. Google*. The Federal Circuit's involvement was not because it had copyright expertise but because it had exclusive jurisdiction.[41] The court tried to solve a copyright issue by patent method. Rather than relying on the existing Ninth Circuit principles, that treated functional interfaces as outside copyright's core, the Federal Circuit reclassified APIs as expressive structures, warranting protection, with the only restriction of fair use.

Unlike a film or a novel, software is part of a stack of dependencies. One program relies on the operating system, which relies on the driver, which relies on the firmware.[42] If a court introduces a "maybe" into the copyright status of any layer in that stack, the entire tower becomes legally unstable. When we apply a flexible standard to a functional interface, we are essentially asking a machine to operate on a "fuzzy" logic that it wasn't built for. The merger doctrine, as discussed above, works precisely because it is rigid: if there is only one way to express a function, that expression is out of bounds for copyright.[43] There is no balancing test, no weighing of intent, just a binary determination that matches the binary world of the code itself. Moreover, the 'innovation tax' discussed above suffers from an inherent lack of clarity. When rules are flexible, the party with the larger legal budget usually wins by default. A dominant tech firm can threaten a smaller entrant with a "fact-heavy" fair use trial even though that entrant's use is likely to be found fair in the end. The cost of proving fairness is often greater than the value of the startup itself.

That action overnight altered the legal perspective on interoperability. The non-infringing behaviours were presumed to be unlawful and lawful only under the cover of an affirmative defence. This institutional impasse did not get resolved even after the Supreme Court intervened

[40] *See, e.g.*, *Computer Assocs. Int'l, Inc. v. Altai, Inc.*, 982 F.2d 693 (2d Cir. 1992) (adopting the abstraction-filtration-comparison test and noting the divergent reception of *Whelan* among the circuits).
[41] 28 U.S.C. § 1295(a)(1) (2006) (amended 2011). The provision conferring Federal Circuit jurisdiction where the district court's jurisdiction rested in whole or in part on 28 U.S.C. § 1338 was rewritten by the Leahy-Smith America Invents Act, Pub. L. No. 112-29, § 19(b), 125 Stat. 284, 331 (2011).
[42] Sujit Nagarajan, *Firmware vs. Software: What's the Real Difference?*, Elemental Electronics (July 9, 2025), https://elementalelec.com.au/firmware-vs-software/.
[43] *Oracle Am.*, 872 F. Supp. 2d at 997–98.

 [ISSN 2581-5369]

in the end. While the court altered the immediate decision, it expressly declined to decide the copyrightability question, assuming it in Oracle's favour for argument's sake. So, future conflicts will continue to face similar uncertainty.

The United States is not the only country to have this institutional issue. In the case of *R.G. Anand v. Delux Films*,[44] the Supreme Court of India had noted that it is important to first see whether the pirated material is even protected expression. It is evident from this sequence that there are no ideological boundaries to be replaced with equitable defences.

The procedural redirection in the *Oracle v. Google* litigation produced significant economic consequences. Instead of providing legal certainty, the reliance on a fair use framework generated doctrinal instability concerning litigation risk for software developers. According to scholarly commentary, such uncertainty as a cost of innovation can be incurred by pre-existing firms in a way that is likely to be cost-effective, and in effect, blocks the entry of independent developers and startups into the market. The legal structure is turning a technical requirement into a legally ambiguous and economically costly, and disproportionately incumbent-favouring, exercise by treating interoperability as a matter of fair use instead of a preliminary establishment of copyrightability. The API litigation thus indicates that functional aspects existing not only pose doctrinal problems to the copyright law but also create unequal protection of the law due to the lack of consistency in jurisdictions.

VI. Conclusion: Beyond the Google Verdict

The *Oracle v. Google* lawsuit revealed a structural weakness in the contemporary copyright law, that is, the lack of ability to differentiate between the creative expression and the functional infrastructure in software. Once the basic interfaces on which the functionality of systems is grounded, is questioned by the legal system, copyright ceases to be a measured incentive and begins to act as a kind of regulatory choke point on technological innovation. Therefore, when discussing software interfaces, the real challenge facing the law is not in individual cases, but in preserving the legitimacy of copyright in the face of a rapidly changing economy. The approach, from now on, should be to provide ex ante clarity to prevent and resolve disputes on interoperability, as opposed to using ex post affirmative defences. One is the functional limitation, technical standards and compatibility provisions that demand the use of doctrinal instruments that may actually draw strict boundaries at the boundary of copyright protection. The merger doctrine is one of such boundary setting mechanisms; it is not yet losing competitiveness and still gives significant protection to really creative software elements. The

[44] *R.G. Anand*, (1978) 4 SCC at 122.

 [ISSN 2581-5369]

question of whether interoperability should be accepted as a principle of architecture that should be preserved or tolerated as an exception by the court and legislators is the one that will ultimately determine the fate of copyright. Extending the concept of copyright to the extent that it is impossible to distinguish any expression and functionality will be counterproductive since it is purported to encourage innovation. The question that remains is whether the law can adapt at a pace such that it seeks to offer protection and yet does not license the digital world to block its creators.

*****

 [ISSN 2581-5369]