U.C.L.A. Law Review

When Disciplines Disagree: The Admissibility of Computer-Generated Forensic Evidence in the Criminal Justice System

Gregory D. Schwartz

ABSTRACT

Criminal trials increasingly rely on computer programs to generate forensic evidence. But experts in the fi lds of computer science and forensic science often disagree over whether programs are sufficiently trustworthy to meet the legal admissibility standards for scientific evidence. When adjudicating between these disciplines, courts overwhelmingly side with forensic experts—even when considering technical, software-specific concerns. Usurping the intended application of the *Daubert* and *Frye* admissibility standards, courts blur the distinctions between scientific disciplines. Software experts are rendered unnecessary to establish the validity of software.

This Essay argues that courts habitually overestimate the reliability of software-generated evidence, and that courts do so because they grossly underestimate the specialized expertise involved in software validation. Instead, courts should treat the computer science and forensic science communities as discrete groups with distinct expertise over diffe ent aspects of forensic procedures. This would allow courts to develop a more robust application of current admissibility standards when considering increasingly interdisciplinary methods of generating evidence.

AUTHOR

J.D. Candidate, Stanford Law School, 2025.

I am deeply grateful to Professor Lisa Larrimore Ouellette for supervising this research. Thank you to Larissa Bersh and Matthew Bova for their helpful comments and suggestions, and to the editors of the *UCLA Law Review*, Kendall Chappell, Michael Dickerson, Samuel Dorsey, Viviana Gonzalez, Kyler McVoy, Sara Orton, Ana Siqueira, Alyssa Stolmack, Alondra Vazquez, and Mariam Zahran, for their detailed editing work. All errors are my own.

TABLE OF CONTENTS

INTRODUCTION

A strange pattern has emerged in criminal evidentiary hearings. Across states, legal standards, and technologies, courts regularly discount software experts when evaluating the validity of forensic software.

In New York, for example, a defendant was on trial for murder and sought to preclude evidence generated by a DNA matching program.[1] Software experts, including a computer science professor specialized in algorithm accountability, testified that the program did not adhere to standard software verification practices, such as independent verification and validation.[2] The government provided no experts with formal computer science credentials to refute this claim.[3] Still, the court decided that because forensic science standards do not require programs to follow computer science standards, the evidence was admissible.[4]

In Minnesota, software experts—including a computer science professor specialized in software verification—reviewed a different DNA matching program and identified several critical deficiencies in the software's development and testing.[5] Again, the government offered no experts with formal computer science credentials.[6] And again, the court decided that adherence to software development standards was unnecessary.[7]

1. People v. Burrus, 200 N.Y.S.3d 655 (Sup. Ct. 2023).
2. *Id.* at 727–28.
3. *Id.* at 657–98. The court deemed one of the prosecution's experts qualified in "DNA testing as well as probabilistic genotyping tools including software development and coding." *Id.* at 688. This expert's undergraduate degree and PhD were in chemistry. *Id.* He was also one of the creators of the DNA matching software. *Id.*
4. *Id.* at 731.
5. Special Master's Report on the Scientific Foundations of STRmix™ at 12–13, United States v. Lewis, 442 F. Supp. 3d 1122 (D. Minn. 2020) (No. 18-CR-194 (ADM/DTS)) ("Among the deficiencies Prof. Heimdahl cited were: (1) lack of a 'hazard analysis' to identify possible failure points in the software for special scrutiny; (2) lack of 'formal specifications' of what the software is expected to do that can provide a basis for independent assessment of whether it is doing what is expected; (3) insufficient documentation of changes in the software which interferes with the 'traceability' (i.e., the effort to link performance characteristics of the software to particular features of the code); and (4) the lack of a formal 'code inspection' by an independent party to test and confirm that the software is operating according to specifications.").
6. United States v. Lewis, 442 F. Supp. 3d 1122, 1126 (D. Minn. 2020).
7. *Id.* at 1131.

In California, the prosecution relied heavily on software-generated evidence, and the court sentenced the defendant to life without parole.[8] On appeal, the court determined that the prosecution had met their burden to show the software was accepted by the "relevant scientific community."[9] Yet, the government had offered no experts with formal computer science credentials.[10]

In Pennsylvania, a homicide defendant's computer science experts made a plea for the court to recognize their expertise:

> We emphasize that forensic DNA and computing disciplines both recognize and emphasize the need for validated products and systems, but that the label of "validated" is achieved through different processes in each discipline. We do not intend to diminish the significance or relevance of [forensic] guidance and standards . . . but suggest that practices common to software development and described in software standards, guidance, articles, and texts are also relevant considerations.[11]

As the criminal justice system becomes increasingly automated, this tension between disciplines and the ramifications of this pattern are only going to grow.

Software is already employed for predictive policing,[12] forensic investigations,[13] sentencing,[14] and parole.[15] These developments have sparked

8. *See* Hannah Zhao, *EFF Tells California Court That Forensic Software Source Code Must Be Disclosed to the Defendant*, ELEC. FRONTIER FOUND. (May 14, 2021), https://www.eff.org/deeplinks/2021/05/eff-tells-california-court-forensic-software-source-code-must-be-disclosed [https://perma.cc/4BG8-VL3V].
9. People v. Davis, 290 Cal. Rptr. 3d 661, 679–80 (Ct. App. 2022).
10. *Id.* at 677–79.
11. Declaration of Nathaniel Adams and Jeanna Matthews at 2, United States v. Ellis, No. 19-369, 2021 WL 1600711 (W.D. Pa. Apr. 23, 2021) (emphasis omitted).
12. *See, e.g.*, Lyria Bennett Moses & Janet Chan, *Algorithmic Prediction in Policing: Assumptions, Evaluation, and Accountability*, 28 POLICING & SOC'Y 806 (2018); Jeff Asher & Rob Arthur, *Inside the Algorithm That Tries to Predict Gun Violence in Chicago*, N.Y. TIMES (June 13, 2017), https://www.nytimes.com/2017/06/13/upshot/what-an-algorithm-reveals-about-life-on-chicagos-high-risk-list.html [https://perma.cc/K7RC-85RS].
13. *See, e.g.*, U.S. GOV'T ACCOUNTABILITY OFF., GAO-20-479SP, FORENSIC TECHNOLOGY: ALGORITHMS USED IN FEDERAL LAW ENFORCEMENT 5–6 (2020).
14. *See, e.g.*, Rebecca Wexler, *Code of Silence: How Private Companies Hide Flaws in the Software That Governments Use to Decide Who Goes To Prison and Who Gets Out*, WASH. MONTHLY (June 11, 2017), https://washingtonmonthly.com/2017/06/11/code-of-silence [https://perma.cc/NX5F-BF2H].
15. *See, e.g.*, Rebecca Wexler, *When a Computer Program Keeps You in Jail*, N.Y. TIMES (June 13, 2017), https://www.nytimes.com/2017/06/13/opinion/how-computers-are-harming-criminal-justice.html [https://perma.cc/W2WF-JA52].

critiques regarding the fairness[16] and transparency[17] of these systems, with growing concern over the role of ownership and trade secrets.[18] In court, software developers often claim that their source codes are trade secrets and thus exempt from review by criminal defendants or their attorneys.[19] Such claims, and related concerns to protect proprietary interests, encompass diverse technologies, including image-identification programs,[20] recidivism predictors,[21] gunshot detection mechanisms,[22] breathalyzers,[23] and DNA matching software.[24]

These tools interact with a justice system dominated by expert testimony,[25] but automation has narrowed the scope of expert authority considerably. Previously, experts testified about experiments that they had personally conducted by hand, such as a forensic scientist describing their fingerprint-identification procedure. Today, expert witnesses tend to speak to automated processes that they

16. *See, e.g.*, Julia Angwin & Jeff Larson, *Bias in Criminal Risk Scores Is Mathematically Inevitable, Researchers Say*, *in* ETHICS OF DATA & ANALYTICS 265 (2022) (identifying concerns over accuracy, objectivity, errors, and bias).
17. *See* Wexler, *supra* note 14. *Contra Research News: Scientific Risk Assessments May Result in More Equitable Sentences*, VANDERBILT UNIV. (Sept. 11, 2014, 12:38 PM), https://news.vanderbilt.edu/2014/09/11/scientific-risk-equitable-sentences/ [https://perma.cc/4YVR-WMUP].
18. *See, e.g*, Rebecca Wexler, *Life, Liberty, and Trade Secrets: Intellectual Property in the Criminal Justice System*, 70 STAN. L. REV. 1343, 1429 (2018).
19. *See, e.g.*, State v. Wakefield, 9 N.Y.S.3d 540, 543 (Sup. Ct. 2015); People v. Superior Ct. (Chubbs), No. B258569, 2015 WL 139069, at *2 (Cal. Ct. App. Jan. 9, 2015).
20. *See, e.g.*, Jack Gillum, *Prosecutors Dropping Child Porn Charges After Software Tools Are Questioned*, PROPUBLICA (Apr. 3, 2019, 5:00 AM), https://www.propublica.org/article/prosecutors-dropping-child-porn-charges-after-software-tools-are-questioned [https://perma.cc/E6FA-UEHA]; United States v. Ocasio, No. EP-11-CR-2728-KC, 2013 WL 2458617 (W.D. Tex. June 6, 2013) (discussing TLO's, a software company, Child Protection System); United States v. Rosenschein, CR. No. 16-4571 JCH, 2020 WL 3572662 (D.N.M. July 1, 2020) (discussing Microsoft's PhotoDNA); United States v. Miller, 982 F.3d 412 (6th Cir. 2020) (discussing Google's hashing algorithm for Gmail).
21. *See, e.g.*, State v. Loomis, 881 N.W.2d 749 (Wis. 2016) (discussing Equivant's Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) program).
22. *See, e.g.*, Brendan Max, *SoundThinking's Black-Box Gunshot Detection Method: Untested and Unvetted Tech Flourishes in the Criminal Justice System*, 26 STAN. TECH. L. REV. 193, 240–41 (2023).
23. *See, e.g.*, Edward J. Imwinkelried, *Computer Source Code: A Source of the Growing Controversy Over the Reliability of Automated Forensic Techniques*, 66 DEPAUL L. REV. 97, 110–11 (2016).
24. *See, e.g.*, Wakefield, 9 N.Y.S.3d at 543 (discussing Cybergenetics's TrueAllele); United States v. Gissantaner, 990 F.3d 457 (6th Cir. 2021) (discussing the New Zealand Crown Research Institute's STRmix).
25. *See* Peter J. Neufeld, *The (Near) Irrelevance of* Daubert *To Criminal Justice and Some Suggestions for Reform*, 95 AM. J. PUB. HEALTH S107, S107 (2005) ("[T]here is rarely a criminal trial that does not rely on some form of expert testimony.").

have merely overseen, such as testifying to the results of DNA matching software, such as TrueAllele, without ever seeing the underlying source code.[26]

Despite the shrinking expertise of the witnesses conducting forensic tests, state and federal courts rarely acknowledge the need for robustness and validity checks on forensic software or for independent examinations by defendants. Criminal defendants have been largely unsuccessful in seeking source code disclosure under the Confrontation Clause,[27] the Due Process Clause,[28] and admissibility standards.[29]

During admissibility hearings, defendants have argued that without source code disclosure, software evidence cannot be verified to the extent required by the legal standards for admitting scientific evidence.[30] Such claims are bolstered by scholarly arguments that source code disclosure is necessary for many programs to find acceptance in the scientific community[31]—a common requirement across admissibility standards.[32]

For many technologies, courts have been reluctant to accept these arguments. Courts routinely admit breathalyzer results without source code review, even after discovery from an outlier decision uncovered "astonishing and highly disturbing" defects in the Alcotest 7110 MKIII-C breathalyzer.[33] While the New Jersey Supreme found that these defects made the breathalyzer

26. *See* Joe Palazzolo, *Judge Denies Access to Source Code for DNA Software Used in Criminal Cases*, WALL ST. J. (Feb. 5, 2016, 11:03 AM), https://www.wsj.com/articles/BL-LB-53075 (reporting comments from TrueAllele's developer Dr. Mark Perlin that "[s]cientists test executable software programs on real data [in validation studies]; they do not read source code text."); *see, e.g.*, Commonwealth v. Foley, 38 A.3d 882 (Pa. Super. Ct. 2012) ("[S]cientists can validate the reliability of a computerized process even if the 'source code' underlying that process is not available to the public.").
27. *See, e.g.*, State v. Lindner, 252 P.3d 1033, 1036 (Ariz. Ct. App. 2010).
28. *See, e.g.*, State v. Loomis, 881 N.W.2d 749, 753, 760 (Wis. 2016).
29. *See, e.g.*, People v. Bullard-Daniel, 42 N.Y.S.3d 714, 715 (Niagara Cnty. Ct. 2016).
30. *See* Order at *6–9, State v. Shaw, No. CR-13-575691 (Ohio Ct. Com. Pl. Oct. 10, 2014).
31. *See, e.g.*, BRIAN CARRIER, OPEN SOURCE DIGITAL FORENSICS TOOLS: THE LEGAL ARGUMENT 9 (2002); Darrel C. Ince, Leslie Hatton & John Graham-Cumming, *The Case for Open Computer Programs*, 482 NATURE 485, 485 (2012) ("[A]nything less than release of actual source code is an indefensible approach for any scientific results that depend on computation . . . ."); A. Morin, J. Urban, P.D. Adams, I. Foster, A. Sali, D. Baker & P. Sliz, *Shining Light Into Black Boxes*, 336 SCIENCE 159, 159 (2012) ("In the absence of source code, the inner workings of a program cannot be examined, adapted, or modified.").
32. *See infra* Part II.
33. Steven M. Bellovin et al., *Seeking the Source: Criminal Defendants' Constitutional Right to Source Code*, 17 OHIO STATE TECH. L.J. 1, 11 (2021). For further description of Alcotest's acceptance and defects, see *id.* at 9–11; *infra* notes 49–51 and accompanying text.

inadmissible,[34] and the Supreme Court of Minnesota cited this finding when it required the disclosure of another breathalyzer's source code,[35] evidence from undisclosed code continues to enter courts nationwide.[36] For years, courts similarly admitted evidence without source code review from the DNA matching program FST, until discovery uncovered errors so substantial that FST was discontinued and its source code unsealed.[37] Later, in discovery regarding FST's replacement, STRmix, a defendant's review of the source code revealed an error that had affected the program's calculations in sixty other cases.[38] Thus far, the majority of admissibility hearings on STRmix's primary competitor,[39] TrueAllele, have not required source code disclosure.[40] A recent decision in New Jersey may signal a shift[41]—after TrueAllele has been used for decades in over 850 criminal cases.[42]

Using DNA-matching software as a case study, this Essay argues that courts habitually overestimate the reliability of software-generated evidence, and that courts do so because they inadequately appreciate the different forms of expertise needed to validate software. In particular, closed-source software is prone to several kinds of errors that courts do not adequately consider—omissions that could make the evidence from these programs inadmissible under the current standards for scientific evidence. The implications of this oversight would shift the current trade secret debate: Where admissibility standards mandate disclosure,

34. State v. Chun, 943 A.2d 114, 153 (N.J. 2008) (finding that the breathalyzer was not "sufficiently scientifically reliable to be admissible").
35. State v. Underdahl, 767 N.W.2d 677, 685 (Minn. 2009) (drawing from *Chun* that "integrity of the source code is essential to the scientific reliability").
36. Stacy Cowley & Jessica Silver-Greenberg, *These Machines Can Put You in Jail. Don't Trust Them*, N.Y. TIMES (Nov. 3, 2019) https://www.nytimes.com/2019/11/03/business/drunk-driving-breathalyzer.html [https://perma.cc/G25M-H38W].
37. *See* discussion accompanying *infra* notes 52–55.
38. *See* discussion accompanying *infra* note 56.
39. STRmix and TrueAllele are "the two most widely used probabilistic genotyping (PG) software systems used in the United States." Susan A. Greenspoon, Lisa Schiermeier-Wood & Bradford C. Jenkins, *A Tale of Two PG Systems: A Comparison of the Two Most Widely Used Continuous Probabilistic Genotyping Systems in the United States*, 69 J. FORENSIC SCIS. 1840, 1841 (2024).
40. *See TrueAllele® Admissibility*, CYBERGENETICS, https://www.cybgen.com/information/admissibility/page.shtml [https://perma.cc/49J4-K97S] (listing the hearings in which TrueAllele evidence was admitted).
41. *See* State v. Pickett, 246 A.3d 279, 283–84 (N.J. Super. Ct. App. Div. 2021).
42. Lauren Kirchner, *Powerful DNA Software Used in Hundreds of Criminal Cases Faces New Scrutiny*, MARKUP (Mar. 9, 2021, 9:59 AM), https://themarkup.org/news/2021/03/09/powerful-dna-software-used-in-hundreds-of-criminal-cases-faces-new-scrutiny [https://perma.cc/Q2SB-3AYG].

courts do not need to evaluate developers' trade secret claims. It would be the developer's choice whether they want to relinquish their claim in order to make their product admissible.

This Essay begins in Part I by outlining the different kinds of software errors, the difficulties in assuring software performance, and how the computer science community has addressed these difficulties. Part II explains the requirements of *Daubert* and *Frye*—the dominant standards for scientific admissibility in federal and state courts. Part III covers how courts' analysis of software reliability fails to meet the *Daubert* and *Frye* standards. Part IV offers ways in which courts can improve their analysis by incorporating computer science expertise.

I. UNDERSTANDING SOFTWARE

Courts tend to assume that computers are objective, accurate, and reliable.[43] But software errors are common—they exist in many forms across multiple levels of abstraction. It is challenging to verify the absence of errors, so the computer science community has developed specialized standards and knowledge within the field of software assurance. Additionally, many errors are specific to the programs in which they occur, so the academic computer science community rarely evaluates individual programs. That software assurance experts have specialized expertise, and that computer science researchers focus on generalizable assurance methods, undermines most courts' approaches to evaluating software.

A. Software Errors

Software misbehavior is common, even in code developed by leading experts for applications where reliability is critical. For instance, the hole in the ozone layer went unnoticed for years because NASA programmers set their software to ignore unrealistic outlier data.[44] Likewise, a misplaced less-than symbol (<) in Ireland's National Integrated Medical Imaging System caused "potentially thousands of patient records from MRIs, X-rays, CT scans and ultrasounds" to be recorded

43. *See* Eric Van Buskirk & Vincent T. Liu, *Digital Evidence: Challenging the Presumption of Reliability*, 1 J. DIGIT. FORENSIC PRAC. 19, 20–21 (2006) (collecting cases).

44. *See Research Satellites for Atmospheric Sciences, 1978–Present: Serendipity and Stratospheric Ozone*, NASA EARTH OBSERVATORY (Dec. 10, 2001), https://earthobservatory.nasa.gov/Features/RemoteSensingAtmosphere/remote_sensing5.php [https://perma.cc/S89U-ACT2].

incorrectly.[45] A software malfunction caused the cancer-treatment device Therac-25 to give patients hundreds of times the intended radiation.[46] Faulty code led a large Australian bank to misreport transactions for almost three years, facilitating widespread money laundering.[47] In rare cases, software is intentionally deceptive, as when Volkswagen programmed its vehicles to cheat on emissions tests.[48]

These mistakes were made by large, trusted entities handling critical and sensitive tasks. Even for them, software assurance is challenging, and forensic software developers face the same difficulties. Yet courts have repeatedly admitted software-generated evidence without source code review—a fundamental software assurance method—only for eventual analysis to uncover serious defects in the software's programming. In a series of cases where defendants were charged with driving under the influence of alcohol, courts consistently maintained that there was no need to disclose the source code of the breath-testing devices because the defendants had "other avenues of challenge."[49] The courts noted that defendants could access the devices' calibration records and operator checklists to determine whether the devices were working and properly used.[50] Yet an eventual review of the source code for New Jersey's breath-testing devices by a paid defense expert "uncovered a variety of defects that could impact

45. Jack Power, *Software Company Behind HSE Scan Glitch Begins Investigation*, IRISH TIMES (Aug. 5, 2017, 5:33 PM), https://www.irishtimes.com/news/ireland/irish-news/software-company-behind-hse-scan-glitch-begins-investigation-1.3178349 [https://perma.cc/MW32-R8W9]; *see* HEALTH SERV. EXEC., NIMIS '<' SYMBOL INCIDENT: FINAL REPORT (2018).

46. Nancy G. Leveson & Clark S. Turner, *An Investigation of the Therac-25 Accidents*, 26 COMPUTER 18, 18 (1993).

47. *See* Allie Coyne, *CBA Blames Coding Error for Alleged Money Laundering*, ITNEWS (Aug. 7, 2017, 8:47 AM), https://www.itnews.com.au/news/cba-blames-coding-error-for-alleged-money-laundering-470233 [https://perma.cc/9F4X-5Q6K].

48. *See* Guilbert Gates, Jack Ewing, Karl Russell & Derek Watkins, *How Volkswagen's 'Defeat Devices' Worked*, N.Y. TIMES (March 16, 2017), https://www.nytimes.com/interactive/2015/business/international/vw-diesel-emissions-scandal-explained.html [https://perma.cc/UK9A-Q7HE]; Russell Hotten, *Volkswagen: The Scandal Explained*, BBC (Dec. 10, 2015), http://www.bbc.com/news/business-34324772 [https://perma.cc/K2BY-2E6B]; Sonari Glinton, *How a Little Lab in West Virginia Caught Volkswagen's Big Cheat*, NPR (Sept. 24, 2015, 5:04 AM), http://www.npr.org/2015/09/24/443053672/how-a-little-lab-in-west-virginia-caught-volkswagens-big-cheat [https://perma.cc/3333-9ULP].

49. *See* People v. Robinson, 860 N.Y.S.2d 159, 165 (App. Div. 2008) (quoting People v. Alvarez, 515 N.E.2d 898, 900 (N.Y. 1987)).

50. *Id.* at 166 ("The defendant thus has not argued, and cannot argue, that he was denied the calibration records of the Intoxilyzer .... Thus, the People satisfied the disclosure and evidentiary requirements for proffering the Intoxilyzer's measurement of the defendant's BAC at trial.").

the test result."[51] The courts insufficiently estimated the risk of software defects, and defendants paid the price.

Courts repeated this error with the Forensic Statistical Tool (FST), a probabilistic genotyping program used to match the DNA of suspects to the DNA found on evidence. It was developed in 2010 by New York City's Office of the Chief Medical Examiner (OCME). Despite OCME's status as a public entity with weakened proprietary interests, the courts denied independent review of FST's source code for years, even under a protective order. In 2016, a federal judge finally ordered OCME to turn over FST's source code for review by Nathaniel Adams, a systems engineer and expert for the defense.[52] In his review, Adams identified critical issues, including a "secret function" that "tend[ed] to overestimate the likelihood of guilt."[53] He found that the software did not use the methodology described in sworn testimony and peer-reviewed publications.[54] Following these findings, FST was discontinued, and its entire source code was unsealed.[55]

Similarly, STRmix—the software chosen to replace FST in New York—was used in 4500 cases globally before its source code was analyzed by independent researchers and found to have programming errors that produced false results in sixty cases.[56] Such examples are not exceptional cases that will lose relevance over time. Fundamental aspects of programming can cause errors at multiple levels of abstraction due to various persistent mechanisms.[57] These errors can remain

51. State v. Underdahl, 767 N.W.2d 677, 685 (Minn. 2009) (describing the report in *State v. Chun*, 943 A.2d 114, 132–33 (N.J. 2008)).
52. Order, United States v. Johnson, No. 15-CR-565 (VEC) (S.D.N.Y. June 7, 2016).
53. Stephanie J. Lacambra, Jeanna Matthews & Kit Walsh, *Opening the Black Box: Defendants' Rights to Confront Forensic Software*, 5 CHAMPION 28, 32 (2018).
54. *Id.*
55. Lauren Kirchner, *Federal Judge Unseals New York Crime Lab's Software for Analyzing DNA Evidence*, PROPUBLICA (Oct. 20, 2017, 8:00 AM), https://www.propublica.org/article/federal-judge-unseals-new-york-crime-labs-software-for-analyzing-dna-evidence [https://perma.cc/Z8BR-2ZJK].
56. David Murray, *Queensland Authorities Confirm 'Miscode' Affects DNA Evidence in Criminal Cases*, COURIER MAIL (Mar. 20, 2015, 10:00 PM), www.couriermail.com.au/news/queensland/queensland-authorities-confirm-miscode-affects-dna-evidence-in-criminal-cases/news-story/833c580d3f1c59039efd1a2ef55af92b [https://perma.cc/A592-NQZM] ("Queensland authorities confirmed the 'minor miscode' had affected DNA likelihood ratios in 60 cases . . . ."). Granted, STRmix's developers contest the significance of these miscodings. *Incorrect Comments Relating To STRmix in* State of New Jersey v. Corey Pickett, STRMIX (Feb. 16, 2021), https://www.strmix.com/assets/STRmix/STRmix-PDFs/STRmix_Response_State_of_NJ_v_Pickett_160221.pdf [https://perma.cc/9P49-RDZ7] ("No miscodes in STRmix have been identified by independent code review.").
57. *Infra* Subpart I.A.1.

benign in most conditions and have unpredictable effects, making them challenging to find or guard against.[58]

1. Kinds of Errors

Programmers can introduce errors into forensic software at multiple levels of abstraction. At the highest level, programmers can err if they rely on incorrect scientific principles or misunderstand correct scientific principles. When designing the details of their implementation, programmers can err if they misunderstand their code, misunderstand code from other programmers, do not foresee certain inputs, or do not foresee interactions between sections of code. At the lowest level, programmers can err if they make typos or other simple errors. Finally, outside of the linear development process, programmers can err if they do not account for software updates.

Forensic software often depends on the validity of many scientific principles. Probabilistic genotyping software (PGS), like FST and STRmix, relies on empirical facts about the variability of human DNA to determine the likelihood that two samples match.[59] If human DNA varies less than these programs presume, PGS would overestimate the likelihood of a DNA match. This reliance on flawed scientific principles is the courts' foremost focus when they evaluate forensic software.[60] But there are many other sources of error.

Forensic software developers can also err in the implementation of correct scientific principles. This risk is heightened when software deals with technical subject areas such as physics, chemistry, and biology, as is frequently the case with forensic software. Unless the programmers have expertise in the relevant related fields, they may have an incomplete understanding of the concepts they are trying to implement and are liable to err.[61]

To be sure, some members of forensic software development teams often hold advanced degrees outside of computer science.[62] But programmers can still misunderstand critical scientific principles when their coworkers possess all the

58. *See infra* Subpart I.A.2.
59. *See* Michael D. Coble & Jo-Anne Bright, *Probabilistic Genotyping Software: An Overview*, 38 FORENSIC SCI. INT'L: GENETICS 219 (2019) (describing the numerical thresholds used to evaluate possible contributors).
60. *See infra* Part II.
61. *See* Christian Chessman, Note, *A "Source" of Error: Computer Code, Criminal Defendants, and the Constitution*, 105 CALIF. L. REV. 179, 188 (2017).
62. *See, e.g.*, United States v. Lewis, 442 F. Supp. 3d 1122, 1133–34 (D. Minn. 2020) (describing STRmix co-developer John Buckleton's scientific background).

relevant expertise. Software projects tend to be an amalgamation of code developed by different members of a programming team,[63] and it is often unrealistic for a resident expert to fully check a project's code.[64]

Even if the underlying science is understood properly, software can still be designed incorrectly because of imperfect foresight and misunderstandings over critical software components. Programmers can fail to anticipate interactions between components of a system, or between a system and its environment.[65] Commercial developers no longer program with 1s and 0s or write instructions from scratch. Instead, developers use shorthand to reference modules of code created by strangers, team members, and themselves. The multiplication character (*), for example, is shorthand that tells the computer to run code that was created by somebody else. As is the shorthand to access a computer file (fopen) or read input from a user (fgetc).[66] Programmers use these generic modules to create custom modules, which they use to make increasingly higher-level custom modules, until they have their final product.[67]

As such, a modern software product is code that runs other code that runs other code. And each reference to "other code" is an opportunity for a developer to misunderstand or misremember the behavior of the module they are referencing.[68] For example, the programming terms "==" and "===" will perform the same function in many cases—but not in every case.[69] Misunderstanding how a component will behave, or failing to foresee its behavior in a certain situation, can cause the entire system to err.

63. *See* Thomas Chau & Frank Maurer, *Knowledge Sharing in Agile Software Teams*, *in* LOGIC VERSUS APPROXIMATION 173, 174 (Wolfgang Lenski ed., 2004) (arguing that substantial relevant information is inevitably lost in communication chains).
64. *See infra* Subpart I.B.1.
65. John Rushby, *Formal Methods and Their Role in the Certification of Critical Systems*, *in* SAFETY AND RELIABILITY OF SOFTWARE BASED SYSTEMS 1, 2 (Roger Shaw ed., 1997).
66. *Standard C Library Functions Table, by Name*, IBM (May 7, 2024), https://www.ibm.com/docs/en/i/7.5?topic=extensions-standard-c-library-functions-table-by-name [https://perma.cc/2FG2-K3UQ]. Note that the available shorthand differs between programming languages.
67. The accepted best practice in programming is to make a module, or "function," for every distinguishable step in program. ROBERT C. MARTIN, *Chapter 3: Functions*, *in* CLEAN CODE: A HANDBOOK OF AGILE SOFTWARE CRAFTSMANSHIP 31, 35 (2009) ("Functions should do one thing. They should do it well. They should do it only."). This helps keep programs organized and understandable.
68. *See* Chau & Maurer, *supra* note 63 (arguing that substantial relevant information is inevitably lost in communication chains).
69. *See* Sobit Prasad, *How Is == Different From === in JavaScript? Strict vs Loose Equality Explained*, FREECODECAMP() (Feb. 14. 2023), https://www.freecodecamp.org/news/loose-vs-strict-equality-in-javascript [https://perma.cc/V7EV-9TS9].

Next, simple errors can cause software to function differently than intended. Even when programmers use correct scientific principles, apply those principles correctly, conceive of logically sound program design, thoroughly consider relevant use-cases, and correctly conceive "imported" code, they can still make typos and confuse words. Even experienced programmers make typos at a high frequency.[70] This can lead to tens of thousands of errors in any given program.[71] Recall the previous example of programming terms: small errors such as typing "==" instead of "===" produce critical errors all the same.

In addition, properly designed and implemented software can develop errors when its outside code is updated. Such updates are often necessary for security, improvements, and compatibility with newer products. But by design, updated code is changed code, and these changes can interact with programs in unexpected ways.[72] The complex mix of old and new code in modern programs means that each update carries the risk of introducing new errors.[73]

2. Handling Errors

While programming, developers employ various methods to reduce the risks from the errors discussed above. But software has several characteristics that make removing and mitigating errors particularly challenging.

First, software errors are hard to remove because they are hard to find. The inputs and outputs of software are generally connected by a series of discrete decisions: if a given condition is true, one section of code is used; otherwise, another section is used.[74] As such, the change in output across changes in inputs is discontinuous. That is, small changes in inputs can flip a program's decision, causing the software to run entirely different code and create a radically different

70. *See* Chessman, *supra* note 61, at 186–89.
71. *Id.* at 187.
72. *See, e.g.*, Jamie Lynch, *The Worst Computer Bugs in History: The Ariane 5 Disaster*, BUGSNAG (Sept. 7, 2017), https://www.bugsnag.com/blog/bug-day-ariane-5-disaster [https://perma.cc/VE2D-BGUH] (explaining how incompatible code reused from the Ariane 4 caused the Ariane 5 to experience rocket failure and crash, costing $370 million).
73. Erik Dietrich, *Learning a Healthy Fear of Legacy Code*, DAEDTECH (June 26, 2024), https://daedtech.com/learning-healthy-fear-legacy-code [https://perma.cc/38LA-XLT9] ("[M]ost of our efforts in software development involve a blend of new and old code. We write some new code, stuff it into some existing code, and then try to figure out how the two things will behave together in production.").
74. *See* Joshua A. Kroll, Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson & Harlan Yu, *Accountable Algorithms*, 165 U. PA. L. REV. 633, 648 n.41 (2017) (asserting that source code analysis can reveal different conditional behaviors for inputs above or below a threshold).

output.[75] This discontinuous relationship between inputs and outputs makes it difficult to verify that a program is error-free by testing it. Accuracy on a sample of inputs cannot be reliably extrapolated to other inputs.[76] As such, software verification cannot rely on testing to find errors like other disciplines.[77]

Second, software errors are hard to mitigate because their effects are hard to predict. Mitigation works by reducing the impact of unknown errors.[78] If a program still generates the correct output when an error occurs, then the risk from the error is low even if it is never removed. But error-tolerant techniques are difficult to implement and assess.

Compare, for example, multiple-version programming with overengineering. An engineer can preempt errors in the design and construction of a bridge by "overengineering"—adding more material and constructing the bridge to withstand far more than its expected load, but programmers cannot similarly "add more material" and rerun identical code to mitigate an error: duplicating the program would duplicate the error. Instead, in "multiple-version programming," developers create different code to perform the same task and compare the results.[79] But this is still of questionable efficacy.[80] To mitigate the error, the versions must differ where the error exists. Because it is challenging to

75. *See* Rushby, *supra* note 65, at 5–6.
76. Igor Ushakov, *Reliability: Past, Present, Future*, 1 RELIABILITY: THEORY & APPLICATION 10, 11 (2006) ("There is no such concept as a 'sample' for software . . . .").
77. Dick Hamlet, *Continuity in Software Systems*, 27 ACM SIGSOFT SOFTWARE ENG'G NOTES 196, 196 (2002) ("Most engineering artifacts behave in a continuous fashion, and this property is generally believed to underlie their dependability. In contrast, software systems do not have continuous behavior, which is taken to be an underlying cause of their undependability."); Rushby, *supra* note 65 ("[T]he traditional disciplines are founded on science and mathematics and are able to model and predict the characteristics and properties of their designs quite accurately, whereas software engineering is more of a craft activity, based on trial and error rather than calculation and prediction.").
78. Michael R. Lyu, *Software Reliability Engineering: A Roadmap*, FUTURE OF SOFTWARE ENG'G 153, 157 (2007) (outlining common fault tolerant techniques and models).
79. Lorenzo Strigini, *Fault Tolerance Against Design Faults*, *in* DEPENDABLE COMPUTING SYSTEMS: PARADIGMS, PERFORMANCE ISSUES, AND APPLICATIONS 213, 217–19 (Hassan B. Diab & Albert Y. Zomaya eds., 2005); Luping Chen & John H. R. May, *A Diversity Model Based on Failure Distribution and its Application in Safety Cases*, 65 IEEE TRANSACTIONS ON RELIABILITY 1149, 1150 (2016) ("Applications of software diversity can be found in critical flight-controllers, railway signaling and control systems, and nuclear reactor protection systems.").
80. Rushby, *supra* note 65, at 4 (outlining scholars' doubts as to whether multiple-version software "provides any significant additional assurance of safety"); JOHN THOMAS, FRANCISCO LUIZ DE LEMOS, NANCY LEVESON, EVALUATING THE SAFETY OF DIGITAL INSTRUMENTATION AND CONTROL SYSTEMS IN NUCLEAR POWER PLANTS 4–5 (2012) ("[People] do not make mistakes in a random fashion: Therefore, independently developed software is very likely to contain common cause failure modes.").

predict the location of an unknown error, experts worry that multiple-version programming may only give developers false confidence in their products' reliability.[81] Recall again the previous example of programming terms: a programmer can verify their output across a hundred different versions of a program and still produce faulty data if they misuse the term "==="in every version.

B. Evaluating Software

Software errors are prevalent, arise from multiple sources, are hard to remove, and are hard to mitigate. As such, assessing the reliability of software is challenging. To overcome these challenges, developers commonly use two methods: validation testing and source code review. These methods have different limitations and complement each other, although error-resistant programming practices are also typically needed to make programs sufficiently reliable.

Through these methods and practices, the software industry has developed standards for determining when programs are sufficiently reliable. But, due to the wide range of potential implementation errors in software development,[82] these assessments are highly individualized inquiries. Thus, while the computer science scientific community routinely accepts the validity of new concepts and ideas, it does not typically evaluate or accept specific software products.

These characteristics of software assessment and computer science practices have three important implications, which will be drawn out in Parts II and III. First, they help explain the relevance of computer science expertise. Software errors are common, handling software errors is hard, and—as this Subpart will show—computer science experts have developed substantial specialized expertise in calculating software risk. This suggests that courts are wrong when they determine that forensic expertise is sufficient to assess forensic software. Second, these characteristics help explain the risks from overreliance on validation studies. Implementation errors are common and—as this Subpart will show—validation studies have significant shortcomings. This suggests that courts are conducting insufficient analyses when they only check for implementation errors via validation studies. Third, these characteristics help explain the importance of

81. *See* Rushby, *supra* note 65, at 4 (questioning whether any assurance from multiple-version programming is quantifiable).

82. *See supra* Subpart I.A.1 (exploring how software built upon valid scientific principles can still contain errors if the developers misunderstand the science, misunderstand their software, fail to foresee certain inputs, misunderstand the outside software they rely upon, make simple errors, or mishandle software updates).

computer science standards. The computer science scientific community—as this Subpart will show—typically evaluates the effectiveness of validation methods, not the validity of specific programs. As such, when courts ask whether software-generated evidence is deduced from principles that are generally accepted within the relevant scientific community, they should review the methods the developers used to address and analyze error risk—not just software-specific experiments.

1. Assessing Reliability

One method for assessing the reliability of software is validation testing. This method entails providing a program with inputs for which there are known correct outputs and checking whether the program's actual outputs match. Since testing every plausible situation is typically impossible, it is common to group "essentially similar" behaviors for testing.[83] But the discontinuity of software behavior makes such groupings difficult.[84] Seemingly minor changes in inputs can trigger serious software errors.[85] Additionally, depending on the kind of error, it may not be obvious when an error has occurred.[86]

Beyond these general limits, validation tests are particularly ill-suited to evaluating probabilistic genotyping software (PGS). Developers of DNA matching software frequently argue that access to source code is unnecessary because validation studies in peer-reviewed journals are sufficient to establish efficacy.[87] Yet compared to other software tests, validation involving DNA is especially narrow in scope. While some programs can be tested on massive databases, DNA samples are considerably harder to acquire. The most recent TrueAllele validation studies had samples derived from twenty, four, and nine people, respectively.[88] Testing is less effective at catching errors when conducted

83. Rushby, *supra* note 65, at 5.
84. *Id.* at 13–14; *see* Ricky W. Butler & George B. Finelli, *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software*, 19 IEEE TRANSACTIONS ON SOFTWARE ENG'G 3 (1993).
85. *See, e.g.*, Vishal Singh & Prerna Chaudhary, *Y2K38: The Bug*, 2 INT'L J. ENG'G & ADVANCED TECH. 157, 157 (2012) (predicting, akin to Y2K, mass systems failure at the seemingly arbitrary time 03:14:07 UTC on January 19, 2038).
86. *See, e.g.*, NASA EARTH OBSERVATORY, *supra* note 44.
87. *See, e.g.*, *Computers Are Helping Justice*, CYBERGENETICS (June 16, 2017), https://www.cybgen.com/information/newsroom/2017/jun/Cybergenetics-to-New-York-Times-Computers-are-helping-justice.shtml [https://perma.cc/3D3J-6EUL]. The company has presented similar arguments in court. *See, e.g.*, Declaration of Mark W. Perlin, State v. Fair, No. 10–1-09274–5 SEA (Wash. Super. Ct. Apr. 1, 2016).
88. David W. Bauer, Nasir Butt, Jennifer M. Hornyak & Mark W. Perlin, *Validating TrueAllele® Interpretation of DNA Mixtures Containing Up To Ten Unknown Contributors*, 65 J. FORENSIC

over a small sample size, especially errors specific to races that comprise only a small proportion of the population.[89]

Limited testing also narrows the extent to which validation studies can verify empirical assumptions built into the software. PGS relies on assumptions about DNA samples,[90] but validation studies only verify these assumptions for samples similar to those tested. For example, if the blood from a crime scene contains contamination or DNA from a large number of people, but the validation studies did not test the software in similar circumstances, then those studies provide limited support for the software's efficacy.[91]

Due to these limitations, the computer science community often supplements validation tests with direct review of the program's source code.[92] Access to source code alleviates many of the issues with validation studies, as the outcomes from previous disclosure orders have demonstrated.[93] It allows experts to identify typos and understand the logic of the software. This helps them identify whether the logic is incorrect, whether the logic is predicated on a misunderstanding, and whether there are relevant conditions that the validation studies did not test.[94] For these reasons, some scholars doubt that meaningful software assessment is possible without source code review.[95]

Certainly, source code review is a time-consuming process. But reviewers do not have to analyze a program in its entirety for the review to be effective. It is quicker to verify the overarching logic of a program than to comb through each

SCI. 380, 381 (2020) (sampling from "20 preset male and female individuals of predominantly Caucasian descent"); Mark W. Perlin, *Efficient Construction of Match Strength Distributions for Uncertain Multi-Locus Genotypes*, 4 HELIYON, Oct. 2018, at 11 (using a sample derived from four people); Nancy A. Stokes, Cristina E. Stanciu, Emily R. Brocato, Christopher J. Ehrhardt & Susan A. Greenspoon, *Simplification of Complex DNA Profiles Using Front End Cell Separation and Probabilistic Modeling*, 36 FORENSIC SCI. INT'L: GENETICS 205, 206 (2018) (using a sample size of nine). For a full list of TrueAllele studies, see *Publications*, CYBERGENETICS, https://www.cybgen.com/information/publication/page.shtml [https://perma.cc/S9HV-HKAA].

89. *See* Lacambra et al., *supra* note 53.
90. *See* JILL R. PRESSER & KATE ROBERTSON, AI CASE STUDY: PROBABILISTIC GENOTYPING DNA TOOLS IN CANADIAN CRIMINAL COURTS 14 (2021).
91. *Id.*
92. Ince et al., *supra* note 31.
93. *Supra* notes 44–56 and accompanying text.
94. *See* Kroll et al., *supra* note 74, at 648–49, 650–52.
95. *See, e.g.*, Ince et al., *supra* note 31 ("[A]nything less than release of actual source code is an indefensible approach for any scientific results that depend on computation . . . ."); Morin et al., *supra* note 31 ("In the absence of source code, the inner workings of a program cannot be examined, adapted, or modified."); *see* CARRIER, *supra* note 31 ("[O]pen source tools may more clearly and comprehensively meet the [*Daubert*] requirements [for admissibility] . . . .").

line of code. This lesser verification can still catch many logical errors—including misunderstandings of scientific principles, misunderstandings of code, and unforeseen inputs.

A rushed source code review can also be used with relative ease to ensure compliance with certain industry standards. To alleviate the limitations of standalone source code analysis and validation tests, best practices in software development can include methods such as hazard analysis, traceability, and defensive programming.[96] Built-in checks, for example, are a form of defensive programming which can catch common incorrect behaviors by specifically checking whether they are occurring.[97] If a program uses sensors with an expected range of readings, a programmer can add a check which warns the user if the readings ever exceed that range.

The success of such practices, however, relies on whether they are adopted, and whether they are adopted thoroughly. Without source code disclosure, reviewers are unable to evaluate the implementation of error-resistant code, which compromises incentives for developers and the potential for these standards as a solution.[98] Combined, source code disclosure and compliance with best practices can complement validation tests. It can be far less resource-intensive for a reviewer to determine whether formal standards are being followed than to evaluate an entire program. Industry standards and source code disclosure together can help address the limitations of each and provide a potent means of verifying forensic software.

Still, even with validation tests, source code review, and best programming practices, there is often a substantial gap between the trustworthiness that users need and computer scientists' capacity to measure and achieve that trustworthiness.[99] Determining the size of this gap is not a trivial task. Meaningful

96. *See Software Reliability Techniques* in APPLIED R&M MANUAL FOR DEFENCE SYSTEMS 5, 7, 14–15 (2012).

97. *See, e.g.*, Erlend Oftedal, *Code-Level Defenses*, *in* SQL INJECTION ATTACKS AND DEFENSE 365, 379–80 (Justin Clarke-Salt ed., 2d ed. 2012) (discussing the specific errors to check for when "validating input").

98. The risk of even public employees lying about the formal standards and logic that they implement is not hypothetical. *See, e.g.*, Lacambra, et al., *supra* note 53 ("A Forensic Statistical Tool (FST) was developed in 2010 by New York City's Office of the Chief Medical Examiner (OCME). For years, OCME fought any independent review of FST's source code and other software development materials, even under a protective order . . . . The actual functioning of the software, revealed upon inspection of the source code, did not use the methodology publicly described in sworn testimony and peer-reviewed publications.").

99. Jonathan P. Bowen & Victoria Stavridou, *Formal Methods and Software Safety*, 25 IFAC SYMP. SERIES 93, 97 (1992).

assessments of software must consider the known or potential error rate, but the error rate in a program can be complex due to its discontinuity across different inputs. Seemingly trivial differences between inputs can make the risk of error change from nonexistent to guaranteed.[100] Not knowing what those high-risk circumstances are, or if they even exist, makes validating software a challenging task. But, while guaranteed software performance is elusive, strict industry standards allow reputable companies to reach the certainty required for even safety-critical programs, such as software for airplanes, medical devices, and weapons systems.

Taken together, the techniques discussed in this Subpart allow computer science experts to evaluate the likelihood of software errors, but the complexity and nuance of these problems mean that such determinations require specialized expertise. When forensic experts alone opine on these issues, they are liable to misunderstand why this nuance makes certain assurance methods necessary. When the norms differ between the computer science and forensic science academic communities, an undiscerning court might ignore relevant standards and fail to require necessary tests.

2. Scientific Acceptance

The wide range of potential implementation errors in software development also affects what it means for the computer science scientific community to accept software. This has important implications for courts because the dominant admissibility standards ask whether the evidence is derived from principles with general acceptance from the relevant scientific community. Most courts center this inquiry around experiments, or validation tests,[101] but the role of experiments within the academic computer science community is complicated.[102] The

100. *Supra* notes 74–76 and accompanying text.

101. *See, e.g.*, United States v. Gissantaner, 990 F.3d 457, 464–65 (6th Cir. 2021); *see infra* Part III. TrueAllele's developers push for this publicly as well. MARK W. PERLIN, CYBERGENETICS, INNOVATION AND TRANSPARENCY FOR RELIABLE FORENSIC SOFTWARE 4 (2022), https://www.cybgen.com/information/presentations/2022/SCU/Perlin-Innovation-and-transparency-for-reliable-forensic-software/handout.pdf [https://perma.cc/6N92-5GJ4] ("[Validation tests are] how real scientists assess reliability.").

102. Matti Tedre & Nella Moisseinen, *Experiments in Computing: A Survey*, SCI. WORLD J., Jan. 2014, at 1 ("The view that computing is an inseparable combination of three very different intellectual traditions—theory, engineering, and empirical science—complicates many debates about computing. One such debate is the 'experimental computer science' debate. The words 'experiment' and 'experimental' are understood very differently between the traditions . . . .").

community generally evaluates the effectiveness of validation methods, not the validity of specific programs. As such, to properly determine whether software has general acceptance, courts should emphasize computer science standards and not expect in-depth software-specific critiques from scholars.

While experiments are central in science, they provide limited information to computer scientists. Software discontinuity makes it challenging to generalize the behavior of a specific program,[103] and implementation error makes it challenging to generalize behaviors across programs. Indeed, some scholars go so far as to argue that there can be "no academic discipline of computing but just eclectic knowledge about particular machines."[104] While other computer scientists view experiments more favorably, they rarely investigate the efficacy of specific programs.[105] In academia, specific programs tend to be used as a proof of concept rather than the object of review.[106] In cases where specific products are tested, the limits of empirical tests are widely recognized. For life-critical software, scholars have found that it is infeasible for experimental validation alone to provide adequate verification.[107]

In contrast, there is computer science scholarship on more generalizable principles, such as the effectiveness of error-handling methods.[108] As such, insofar as the computer science community finds general acceptance for principles from which software-generated evidence is deduced, those principles will typically be generalized standards regarding the sufficiency of error-handling methods.[109] The computer science community will very rarely develop general acceptance of a specific program. And even when such an acceptance develops, it will be

103. *See supra* notes 74–77 and accompanying text.
104. *See* Tedre & Moisseinen, *supra* note 102, at 5.
105. *See id.* at 6–7 (discussing five views on experiments in computer science); Marvin V. Zelkowitz & Dolores R. Wallace, *Experimental Models for Validating Technology*, COMPUTER, May 1998, at 23. "Experimentation is one of those terms that is frequently used incorrectly in the computer science community." *Id.* Here, "experiment" really means an example that the technology exists or an existence proof that the technique can be employed. "Very rarely does [it] involve any collection of data to show that the technology adheres to some underlying model or theory of software development or that the software is effective." *Id.*
106. *Id.* at 23, 29.
107. *See* Butler & Finelli, *supra* note 84, at 7–10.
108. *See, e.g.*, Algirdas Avižienis, Jean-Claude Laprie, Brian Randell & Carl Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, 1 IEEE TRANSACTIONS DEPENDABLE & SECURE COMPUTING 11, 24–29 (2004).
109. *See, e.g.*, Raj kamal Kaur, Babita Pandey & Lalit Kumar Singh, *Dependability Analysis of Safety Critical Systems: Issues and Challenges*, 120 ANNALS OF NUCLEAR ENERGY 127 (2018) (surveying studies which sought to analyze and improve the dependability of safety critical systems).

predicated on generalizable principles regarding the sufficiency of the error-handling methods used on that program.

Notably, because of the computer science community's focus, what experiments exist for specific forensic software tend to be in publications published and peer-reviewed by the forensic science community, not the computer science community.[110] Such studies are subject to general critiques that have been made of forensic science as a discipline.[111] More specifically, forensic science standards do not address the possibility of discontinuous implementation errors or the utility of standard software validation practices.[112] Determining when validation studies are sufficient to evaluate software dependability requires specialized expertise.[113] This expertise, however, is notably absent in courtroom admissibility hearings.

II. ADMITTING SOFTWARE

To protect against junk science, courts tend to evaluate the admissibility of scientific evidence through multiple distinct prongs. The diversity of these prongs is essential to the robustness of the evaluation. However, in hearings on forensic evidence created by computer programs, courts have consistently found that litigants can satisfy every prong with the same two sources of evidence: validation studies and the soundness of software's underlying principles. As discussed above, validation studies have substantial limitations, and underlying principles are only one potential source of software error.

Two standards of admissibility dominate the landscape. Federal courts use the five-factor standard established in *Daubert v. Merrell Dow Pharmaceuticals,*

110. *See, e.g.*, Mark W. Perlin, Matthew M. Legler, Cara E. Spencer, Jessica L. Smith, William P. Allan, Jamie L. Belrose & Barry W. Duceman, *Validating TrueAllele® DNA Mixture Interpretation*, 56 J. FORENSIC SCIS. 1430 (2011); Bauer et al., *supra* note 88.

111. *See, e.g.*, Harry T. Edwards, *Solving the Problems That Plague the Forensic Science Community*, 50 JURIMETRICS 5 (2009); Claude Roux, Frank Crispino & Olivier Ribaux, *From Forensics to Forensic Science*, 24 CURRENT ISSUES CRIM. JUST. 7 (2012).

112. *See, e.g.*, AM. ACAD. FORENSIC SCI. STANDARDS BD., ANSI/ASB STANDARD 018, STANDARD FOR VALIDATION OF PROBABILISTIC GENOTYPING SYSTEMS (2020), https://www.aafs.org/sites/default/files/media/documents/018_Std_e1.pdf [https://perma.cc/4UHQ-KKPX].

113. *See generally* Avižienis et al., *supra* note 108 (describing the numerous intricate considerations when designing and maintaining a secure and dependable computing system).

Inc.,[114] while state courts are split between *Daubert*, the earlier standard from *Frye v. United States*,[115] *Daubert-Frye* hybrids,[116] and the occasional local statute.[117]

The *Frye* standard relies on the scientific community as gatekeepers, using the extent of the community's acceptance to determine the weight and status of the expert's theory or technique. In contrast, the *Daubert* standard treats acceptance by the scientific community as one of many factors to be considered by the judge. The other factors are disconnected from the scientific community's practice and encourage a more holistic judicial consideration by the court.[118] This Part will analyze the treatment of DNA-matching software in *Frye* and *Daubert* hearings, respectively.

A. *Frye* Hearings

In *Frye*, the D.C. Circuit held that evidence must have "gained general acceptance" from the relevant scientific community to be admitted.[119] This was intended to be a higher bar than expert testimony: since general community acceptance is necessary, one or even several expert opinions may be insufficient to establish evidence as admissible.[120] In practice, however, this higher bar has not been translated into a thorough analysis of software. *Frye* hearings routinely evaluate forensic software based only on the results of validation studies and the soundness of underlying scientific principles.

Validation studies test software by providing the program with known samples and recording the error rate. While independent researchers may not be able to directly observe how a trade secret program works, they can test samples to determine how often the software provides the correct output. In doing so, these researchers treat the software like a "black box:" the inner workings of the system

114. 509 U.S. 579 (1993).
115. 293 F. 1013 (D.C. Cir. 1923).
116. David E. Bernstein & Jeffrey D. Jackson, *The* Daubert *Trilogy in the States*, 44 JURIMETRICS 351, 351–52 (2004); *see also* Chessman, *supra* note 61, at 219 (arguing that evidence prohibited by each test standing alone is likely also prohibited by their combination).
117. MATTHIESEN, WICKERT & LEHRER, S.C., ADMISSIBILITY OF EXPERT TESTIMONY IN ALL 50 STATES 3–9 (2023), https://www.mwl-law.com/wp-content/uploads/2018/02/ADMISSIBLITY-OF-EXPERT-TESTIMONY-CHART-00220033x9EBBF.pdf [https://perma.cc/CV8W-EGAF] (finding that only seven states—Maine, Minnesota, Missouri, Nevada, North Dakota, South Carolina, Virginia—do not use either the *Daubert* or *Frye* standard, but many apply standards "substantially similar to *Daubert*" or rely on *Daubert* as persuasive authority).
118. *See* discussion accompanying *infra* notes 163–168.
119. *Frye*, 293 F. at 1014.
120. *See* Paul C. Giannelli, *The Admissibility of Novel Scientific Evidence:* Frye v. United States*, a Half-Century Later*, 80 COLUM. L. REV. 1197, 1205 (1980).

are hidden, but by identifying the program's outputs for various known inputs, researchers can develop an understanding of its behavior.[121]

Meanwhile, the practice of verifying underlying scientific principles tests software by evaluating the "rules" about the natural world that the software relies on to operate correctly. For example, in *Frye*, the defendant took a lie detection test, claimed he was innocent, and presented the results of the test as evidence of his innocence.[122] One of the test's underlying principles was that a person's blood pressure level spikes when they lie and remains steady when they tell the truth. This principle was not accepted in *Frye*.[123] But if it had been sufficiently established, then—combined with the evidence of James Frye's blood pressure remaining steady—it would have been reasonable under the *Frye* standard for an expert to deduce that James Frye's statements were truthful.

Validation studies and the verification of underlying principles cannot reliably detect many kinds of software errors.[124] Yet admissibility hearings often consider little else. Decisions from New York courts illustrate the widespread judicial dependence on validation studies and underlying principles. These courts often hold *Frye* hearings to evaluate DNA matching software like FST, TrueAllele, and STRmix. Defendants rarely succeed in excluding evidence at these hearings. In New York, there is only one case of note in which such evidence was not admitted.[125] This case was an outlier and has been criticized by later rulings.[126]

Much more frequently, the hearings resemble that of *State v. Wakefield*,[127] where the trial court admitted evidence from TrueAllele software under the *Frye* test.[128] The court separately analyzed five categories of evidence: "peer review,"[129] "validation studies,"[130] the "scientific community,"[131] "legal acceptance,"[132] and

121. *See, e.g.*, Bauer et al., *supra* note 88, at 395–97.
122. *Frye*, 293 F. at 1013–14.
123. *Id.*
124. *See supra* Subpart I.B.
125. People v. Collins, 15 N.Y.S.3d 564, 629 (Sup. Ct. 2015) (concluding that "evidence derived both from high sensitivity analysis and from the [Forensic Statistical Tool, a new software program] are not yet proved to be admissible under the *Frye* test"); *see also* United States v. Wilbern, No. 17-CR-6017-CJS, 2019 WL 5204829, at *8 (W.D.N.Y. Oct. 16, 2019) ("The decision in *Collins* appears to be an 'outlier among the forensic DNA software program cases in New York.'" (quoting People v. Bullard-Daniel, 42 N.Y.S.3d 714, 724 (Niagara Cnty. Ct. 2016))).
126. People v. Carter, No. 2573/14, 2016 N.Y. Misc. LEXIS 166, at *9–19 (N.Y. Sup. Ct. 2016).
127. 9 N.Y.S.3d 540 (2015).
128. *Id.* at 543–46.
129. *Id.* at 543.
130. *Id.* at 543–44.
131. *Id.* at 544–45.
132. *Id.* at 545.

"expert testimony."[133] But, despite being different in name, the sole bases of the analysis under each category were validation studies and the verification of underlying scientific principles.[134] Indeed, so long as TrueAllele's developers shield its source code[135] and do not report any error-mitigation techniques,[136] it is unclear what else can be analyzed.

The first two categories of evidence in *Wakefield* were "peer review[ed]" articles that described validation studies and "validation studies" themselves.[137] Evidence in the third category—acceptance from the "scientific community"—comprised approval from two bodies whose determinations were based on validation studies: the New York State Commission on Forensic Science[138] and the New York State Police.[139] Evidence from the fourth category—"legal acceptance"—reviewed the findings of previous admissibility hearings. These hearings examined factors similar to those used in *Wakefield*. As such, their decisions were based on validation studies and acceptance of scientific principles as well.[140] Finally, the fifth category, "expert testimony," contained testimony from TrueAllele inventor Dr. Perlin and evidence that researchers had presented their validation studies on TrueAllele in conferences.[141] Apart from Dr. Perlin's personal knowledge of TrueAllele—which the court did not discuss—the evidence

133. *Id.* at 546.
134. It is probable that the expert opinions from TrueAllele developers were also based on their access to proprietary information on TrueAllele, but the *Wakefield* court does not consider this in its analysis. *See generally id.* Even if the court had considered it, the information is biased and uncontestable by the defendant.
135. Commonwealth v. Foley, 38 A.3d 882, 889–90 (Pa. Super. Ct. 2012).
136. *See New York State Subcommittee*, CYBERGENETICS (May 20, 2011), https://www.cybgen.com/information/presentations/2011/NYSDNASUB/Subcommittee/page.shtml [https://perma.cc/TH4D-LRAY] [hereinafter NY TrueAllele Review].
137. *Wakefield*, 9 N.Y.S.3d at 543–44; *see also* CYBERGENETICS, *supra* note 88. As the TrueAllele peer-reviewed publications are validation studies—either validating aspects of the TrueAllele technique or the software as a whole—the difference between the two categories is unclear.
138. Press Release, Cybergenetics, New York State DNA Subcommittee Scientists Unanimously Recommend Regulatory Approval of Cybergenetics TrueAllele for Forensic Casework (May 20, 2011), https://www.cybgen.com/information/newsroom/2011/may/New-York-State-DNA-Subcommittee-scientists-unanimously-recommend-regulatory-approval-of-Cybergenetics-TrueAllele-for-forensic-casework.shtml [https://perma.cc/5CUS-TYAB]; *see also* NY TrueAllele Review, *supra* note 136.
139. NY TrueAllele Review, *supra* note 136; *Wakefield*, 9 N.Y.S.3d at 544–45 (describing the New York State Police's approval of TrueAllele based on the New York State Commission on Forensic Science's determination and "three separate validation studies").
140. *Wakefield*, 9 N.Y.S.3d at 545 (relying on People v. Wesley, 633 N.E.2d 451, 456 (N.Y. 1994), which based its determination on peer reviewed articles and the fact that the underlying principles are "widespread in biology").
141. *Id.* at 545–46.

in this category, just like all the others, was ultimately based only on validation studies and underlying scientific principles.

While each category of evidence relied heavily on validation studies, the *Wakefield* court did not address their limitations in evaluating software, despite these shortcomings being well recognized by computer science experts.[142] And when listing the Commission members' credentials, the court identified no expertise in computer science or software.[143] When discussing expert testimony, there was again no mention of computer science or software expertise.

The trial court in *People v. Bullard-Daniel*[144] admitted evidence from STRmix under *Frye* on similar grounds.[145] Their decision was largely based on the testimony of the People's witness, Dr. Simich, who was "thoroughly familiar with the *application* of the STRmix software."[146] His lab conducted validation studies, and he reviewed numerous articles that had done the same. The court also noted that the mathematical models—the underlying scientific principles—were non-controversial and have been widely used in fields such as "weather forecasting, computational biology, linguistics, genetics, engineering, physics, aeronautics, finance, and social sciences."[147]

But the trustworthiness of underlying principles, especially principles as general as these, does not protect against implementation errors. Like in *Wakefield*, validation tests were the only evidence that the developers had not misunderstood the underlying principles, misdesigned the software, used outdated code, or introduced simple errors.[148] Yet again, there was no inquiry into error mitigation or the sufficiency of validation studies given software discontinuity.[149]

Still, *Bullard-Daniel* is an influential decision that has persuaded federal and state courts to admit evidence from STRmix in cases like *People v. Lopez*[150] and

142. *See supra* Subpart I.A.2.
143. *Wakefield*, 9 N.Y.S.3d at 544–45.
144. 42 N.Y.S.3d 714 (Niagara Cnty. Ct. 2016).
145. *Id.* at 723–26.
146. *Id.* at 721 (emphasis added).
147. *Id.*
148. All of which are substantial risks in programming. *See supra* Subpart I.A.1.
149. *See supra* Subpart I.A.2; JOHN RUSHBY, NASA-CR-4551, FORMAL METHODS AND DIGITAL SYSTEMS VALIDATION FOR AIRBORNE SYSTEMS 8 (1993) ("Tests provide information on only the state sequences actually examined; without continuity there is little reason to suppose the behavior of untested sequences will be 'close' to tested ones, and therefore little justification for extrapolating from tested cases to untested ones.").
150. Indictment No. 3927/16 (N.Y. Sup. Ct. Apr. 27, 2018). !

People v. Yates.[151] The consistency of these decisions is such that many courts have stopped making *Frye* inquiries entirely, admitting the evidence without any independent analysis.[152]

Because traditional legal databases offer limited coverage of state trial courts, it is unclear precisely how often *Frye* analyses rely solely on validation studies and underlying scientific principles. But the practice is likely widespread. First, a variety of courts engage in the practice, including federal courts,[153] state courts outside of New York,[154] and an international court which almost exclusively relied on the New York State Commission on Forensic Science's recommendation to adopt TrueAllele.[155] Second, it is unclear what else admissibility could be based on. TrueAllele's developers refuse to provide the information needed for additional verification methods,[156] such as overengineering through multiple-version software or performing traceability[157] This leaves the courts with only validation studies, underlying scientific principles, and derivatives thereof.

For DNA matching, this pattern might reverse. In 2021, *State v. Pickett*[158] marked New Jersey's first precedential appellate holding requiring developers to provide the source code to their DNA matching program.[159] It recognized the implementation gap between theoretical validity and actual effectiveness, and so

151. Indictment No. 10663-2016 (NY Sup. Ct. Oct. 4, 2018). *See* United States v. Lewis, 442 F. Supp. 3d 1122, 1163 (D. Minn. 2020) (describing *Bullard-Daniel*'s influence on *People v. Lopez*, Indictment No. 3927/16 (N.Y. Sup. Ct. Apr. 27, 2018) and *People v. Yates*, Indictment No. 10663-2016 (N.Y. Sup. Ct. Oct. 4, 2018)).

152. *See, e.g.*, People v. Carter, No. 2573/14, 2016 N.Y. Misc. LEXIS 166, at *9 (N.Y. Sup. Ct. 2016) ("[T]he vast majority of the courts of this state that have considered the admissibility of FST have concluded that the techniques used to develop FST are not new, and instead are based on well-established mathematical and statistical principles.").

153. *See infra* Subpart II.B.

154. *See, e.g.*, People v. Superior Ct. (Chubbs), No. B258569, 2015 WL 139069, at *4 (Cal. Ct. App. Jan. 9, 2015) (using the *Kelly/Frye* test); Commonwealth v. Foley, 38 A.3d 882, 888 (Pa. Super. Ct. 2012) (admitting TrueAllele because it is "a refined application of the 'product rule' . . . method" and "scientific evidence based on the product rule is admissible").

155. *See* Ruling on Voir Dire at 16–17, Regina v. Duffy, ICOS No. 09/143857 (N. Ir. Crown Ct. Dec. 1, 2011).

156. NY TrueAllele Review, *supra* note 136.

157. *See infra* Subpart IV.B.1; *see also* Rushby, *supra* note 65, at 12 (discussing multiple-version software); *About Traceability*, CTR. EXCELLENCE FOR SOFTWARE & SYS. TRACEABILITY, http://sarec.nd.edu/coest/aboutTraceability.html [https://perma.cc/Q8ED-9RMX] [hereinafter COEST] ("[The U.S. Food and Drug Administration (FDA) requires] that all aspects of the design are traceable to software requirements . . . . the [U.S. Federal Aviation Administration (FAA)] states that software developers need to have ways of demonstrating traceability between design and requirements . . . .").

158. 246 A.3d 279 (N.J. Super. Ct. App. Div. 2021).

159. Tamar Lerer, *Check the Sources: Why Secret Computer Code Matters and How Defense Counsel Can Get It*, 45 CHAMPION 14 (2021).

addressed the need to ensure that the source code "functions as the science underpinning probabilistic genotyping necessitates."[160] But, while this decision may signal the start of a shift in *Frye* hearings on DNA matching software, for now it remains an anomaly.[161] And should such a change occur, like New Jersey's shift towards disclosing the source code of breathalyzers,[162] courts appear liable to insufficiently scrutinize whatever forensic technology comes next.

B. *Daubert* Hearings

While the *Daubert* standard is generally considered more restrictive than *Frye*,[163] in hearings on forensic software the analyses and results have been largely the same. Federal courts (and state courts in *Daubert* jurisdictions) also habitually rely only on validation studies and the soundness of the software's underlying principles, despite *Daubert*'s multi-pronged test.

In 1993, the U.S. Supreme Court created *Daubert*'s five-factor balancing test.[164] The test is flexible, creating more room for judicial discretion as *Daubert* definitively shifted *Frye's* "gatekeeping role" for the admission of scientific evidence from the scientific community to the judge.[165] Under *Daubert*, it should be examined whether the theory or technique: (1) is scientific knowledge or falsifiable; (2) "has been subjected to peer review and publication"; (3) has a known or potential rate of error; (4) has "the existence and maintenance of standards controlling [the technique's] operation"; and (5) has "widespread acceptance" among the relevant scientific community.[166]

160. *Pickett*, 246 A.3d at 310–11.
161. *See, e.g.*, People v. Burrus, 200 N.Y.S.3d 655, 727–28 (Sup. Ct. 2023) (holding that the IEEE software standard was not applicable because forensic standards "do not require probabilistic genotyping programs to meet the IEEE standard"). Over two years after *Pickett*, courts still fail to independently ask whether forensic software "has gained general acceptance in the computer science community to which it also belongs." *Pickett*, 246 A.3d at 323.
162. *See supra* note 51 and accompanying text.
163. *See* Edward K. Cheng & Albert H. Yoon, *Does Frye or Daubert Matter? A Study of Scientific Admissibility Standards*, 91 VA. L. REV. 471, 472 n.6 (2005).
164. Katherine L. Moss, Note, *The Admissibility of TrueAllele: A Computerized DNA Interpretation System*, 72 WASH. & LEE. L. REV. 1033, 1040–43 (2015).
165. *See* Daubert v. Merrell Dow Pharmaceuticals., Inc., 509 U.S. 579, 597 (1993); David L. Faigman, *The* Daubert *Revolution and the Birth of Modernity: Managing Scientific Evidence in the Age of Science*, 46 U.C. DAVIS L. REV. 893, 907–08 (2013); *see also* John Eric Smithburn, *The Trial Court's Gatekeeper Role Under* Frye, Daubert, *and* Kumho*: A Special Look at Children's Cases*, 4 WHITTIER J. CHILD. & FAM. ADVOC. 3, 18 (2005).
166. *Daubert*, 509 U.S. at 580.

This shift from *Frye* to *Daubert* was intended to address the problem of junk science in criminal cases.[167] The additional factors allow judges more discretion in distinguishing between valid and invalid expert opinions in response to growing concerns that juries were unsuited for the task.[168] In hearings on DNA matching software, however, the analysis under *Daubert* and *Frye* is interconnected to the extent that some *Daubert* hearings consider the decisions made in *Frye* hearings.[169]

The *Daubert* hearings on genotyping software also follow a similar pattern to those under *Frye*. Courts routinely find that *Daubert's* prongs can be satisfied by only validation studies and verification that the software is based on principles widely accepted in scientific communities.[170] They hold that these two sources of information are sufficient to evaluate the effectiveness of the forensic software.

Courts could look to computer science experts to determine when additional software verification is needed.[171] But when such expertise is offered, it is regularly ignored. In a *Daubert* hearing over STRmix, the district court heard from three defense experts, including a software engineer and the Dean of the University of Minnesota Department of Computer Science and Engineering.[172] None of the opposing experts had degrees in computer science. But while the court analyzed STRmix's biological and mathematical underpinnings at length,[173] it gave no such treatment to the software design practices. Instead, it relied on standards from forensic science groups over the Dean's recommendation.[174] Such disregard of

167. *See* Neufeld, *supra* note 25, at S109 ("Many thought *Daubert* would be the meaningful standard that was lacking in criminal cases and that it would serve to protect innocent defendants.").
168. *See* Barefoot v. Estelle, 463 U.S. 880, 916 (1983) (Blackmun, J., dissenting). Justice Blackmun would go on to write the *Daubert* decision.
169. In *United States v. Wilbern*, for example, "[w]ith respect to prior case law on the admissibility of LCN DNA test results generated by OCME, the Court focused on, carefully read, and considered three very well reasoned decisions." No. 17-CR-6017 CJS, 2019 WL 5204829, at *6 (W.D.N.Y. Oct. 16, 2019). One of the decisions assessed OCME under the *Daubert* standard, the other two applied "the more stringent *Frye*." *Id.*
170. On December 8, 2023, I ran the following search on Westlaw: [(STRMix TrueAllele "Forensic Statistical Tool") AND Daubert]. This located fifty-four cases, twenty-three of which were federal or state cases where forensic genotyping tools were evaluated under *Daubert* (the other cases applied the *Frye* standard, applied a hybrid standard, or addressed an unrelated matter). These twenty-three cases consistently demonstrated reliance on validation studies and scientific principles. *See, e.g.*, United States v. Gissantaner, 990 F.3d 457, 463–68 (6th Cir. 2021) (organizing its analysis by *Daubert*'s five prongs and using a combination of validation studies and scientific theory to satisfy each one).
171. *See supra* Subpart I.B (discussing the computer science expertise developed to understand and address software risks).
172. United States v. Lewis, 442 F. Supp. 3d 1122, 1126, 1134 (D. Minn. 2020).
173. *Id.* at 1135–43 (discussing this at length in Part III, "The Science").
174. *Id.* at 1131 ("Those guidelines include standards published by the Scientific Working Group on DNA Analysis Methods, the Forensic Science Regulator, and the International Society for

computer science standards and expertise is commonplace in *Daubert* hearings.[175] Although again, most often computer science testimony is not heard at all.

As such, the witnesses who testify in *Frye* and *Daubert* hearings tend to be exclusively from the field in which the software operates. That is, DNA experts provide testimony regarding DNA identification software. These experts tend to base their understanding on validation studies that they or their peers have conducted. This leaves a significant gap in the information before the courts, namely expertise regarding the proper extent to rely on validation studies.[176]

III. SHORTCOMINGS OF MODERN HEARINGS

As demonstrated in Part II, during admissibility hearings for trade-secret protected, software-generated evidence, courts tend to rely on validation studies and the soundness of underlying principles.[177] They ignore the risk of implementation error and hear near-exclusively from forensic experts. When computer science experts are present, these experts consistently raise the possibility of implementation error and are refuted by unresponsive claims.

The software examples in Subpart I.A—the NASA ozone monitoring system, Ireland medical imaging system, cancer-treatment device Therac-25, and Australian bank—contained serious hidden errors despite having sound underlying principles, successful validation tests, and years of use. These errors were not caught for decades because validation studies cannot test every possible circumstance that may cause a bug to arise,[178] and depending on the kind of error, it may not be obvious that an error has occurred.[179]

Forensic Genetics."). The *Lewis* Court notes that STRmix does not comply with the standards from the technology organization Institute of Electrical and Electronics Engineers (IEEE), but determines that it comes close enough. *Id.*

175. *See, e.g.*, State v. Watkins, 648 S.W.3d 235, 263–65 (Tenn. Crim. App. 2021); *Gissantaner*, 990 F.3d at 468; United States v. Jones, No. S4 15-CR-153 (VSB), 2018 WL 2684101, at *8–12 (S.D.N.Y. June 5, 2018), *aff'd*, 965 F.3d 149, 154 (2d Cir. 2020).

176. See, for example, *People v. Bullard-Daniel*, where the People "rel[ied] solely on the testimony of Dr. Simich and the accompanying exhibits." 42 N.Y.S.3d 714, 719 (Niagara Cnty. Ct. 2016). When evaluating Dr. Simich's testimony, the court identified no computer science background, *see id.* at 716–19, and it decided that his extensive forensic expertise was sufficiently comprehensive. *Id.* at 721 ("As the director of a forensics lab, Dr. Simich is well-qualified to critique software programs like STRmix."); *accord* Affidavit of John P. Simich, United States v. Pettway, No. 12-CR-103S(1), (2) (W.D.N.Y. Oct. 10, 2016).

177. *Supra* Part II.

178. *See supra* notes 74–76 and accompanying text.

179. *See supra* notes 74–76 and accompanying text.

These are known limitations in computer science, and as such, the computer science discipline understands experiments differently. Experiments are frequently used as proofs of concept rather than as validation for a specific product.[180] In cases where researchers test specific products, the limits of empirical tests are widely recognized.[181] In response to the limits of mere experimentation, software assurance experts have developed complex methods to assess reliability more accurately and various programming techniques to improve it.

These established limitations of validation studies are relevant considerations under current admissibility standards. Both *Frye* and *Daubert* consider general acceptance by the scientific community.[182] And under both standards, multidisciplinary evidence requires acceptance by multiple disciplines.[183] Identifying the relevant communities across disciplines is difficult, especially under the narrower considerations in *Frye*.[184] But capturing all relevant expertise is critical and required nonetheless.[185] If, when evaluating forensic software, software experts possess specific pertinent expertise, then courts are missing a substantial portion of the relevant analysis.

Indeed, computer science expertise is critical to understand the likelihood of software errors, whether software is being applied outside of its scope, and the available supplements to validation studies. Critically, treating forensic scientists as experts on software overlooks the complexity of software validation and grants

180. *See* Zelkowitz & Wallace, *supra* note 105, at 23. "Experimentation is one of those terms that is frequently used incorrectly in the computer science community." *Id.* Here, "experiment" really means an example that the technology exists or an existence proof that the technique can be employed. "Very rarely does [it] involve any collection of data to show that the technology adheres to some underlying model or theory of software development or that the software is effective." *Id.*

181. Butler & Finelli, *supra* note 84, at 7–10.

182. *See* Smithburn, *supra* note 165, at 6–7, 9.

183. Simone A. Cole, *Out of the* Daubert *Fire and Into the* Frye*ing Pan? Self-Validation, Meta-Expertise and the Admissibility of Latent Print Evidence in* Frye *Jurisdictions*, 9 MINN. J.L. SCI. & TECH. 453, 480–81 (2008) ("[V]irtually all [*Frye*] courts have articulated a preference construing the 'relevant scientific community' broadly, rather than narrowly."). *Daubert* analysis is broader still. *See* Kerri N. Polizzi, *How Long Do We Keep* Frye*ing?: The Future of Expert Scientific Evidence in California*, 20 CHAP. L. REV. 393, 394 (2017).

184. Polizzi, *supra* note 183 ("This problem [of determining scientific acceptance] is particularly prevalent where multiple scientific communities claim a technique as their own.").

185. *See* United States v. Porter, 618 A.2d 629, 634 (D.C. 1992) ("It simply is not creditable to argue . . . that general acceptance may be premised simply on the opinion of forensic scientists . . . . While views of forensic scientists have weight and must be considered, 'members of the relevant scientific field will include those whose scientific background and training are sufficient to allow them to comprehend and understand the process and form a judgment about it.'" (quoting Reed v. State, 391 A.2d 364, 368 (Md. 1978))).

computers undeserved deference.[186] It leads courts to underappreciate the difficulty of generalizing software behavior from a few tests, the risk of subsets of defendants triggering untested behavior, and the risk of errors going unnoticed.

To be clear, the issue is not that software evidence is uniquely complex or difficult to generalize, but that the courts have not applied the same scrutiny to software evidence that they have applied to other similarly complex scientific evidence. For example, consider medical evidence. Medical principles can also be hard to generalize. It is not uncommon for drugs to have adverse effects on a subset of people,[187] and determining causality can be extraordinarily complex. But the medical community has developed sophisticated expertise that allows them to agree on generally appropriate treatments, and this acceptance rightly carries weight in court. The key difference is that when determining the admissibility of medical evidence, the courts hear medical testimony regarding the complexity of biological reactions and the likelihood of error. Courts evaluating a DNA matching technology rarely hear from a software expert or recognize these specialists' distinct expertise.

Prior to computerized DNA matching, forensic experts could understand all the steps in their process. The risks of human error and contamination were within these experts' purview, so they could fully speak to the reliability of their results. But when software becomes highly integrated into the procedure, forensic experts' scope of authority narrows.[188] Courts have not brought in additional expertise to maintain a complete understanding of the evidence's reliability. Consequently, courts convict and exonerate defendants with only partially verified evidence, as they leave entire categories of errors unchecked.

IV. POTENTIAL SOLUTIONS

The *Daubert* and *Frye* standards require courts to evaluate the risk of implementation error.[189] But courts are failing to apply these standards properly because they are too reliant on theoretical principles and validation studies. To

186. *See* Rushby, *supra* note 65, at 2; People v. Bullard-Daniel, 42 N.Y.S.3d 714, 720 (Niagara Cnty. Ct. 2016) ("[T]he Court agrees[] that the only question before it is whether the scientific principles underlying the STRmix software are accepted generally in the relevant scientific community.").

187. *See, e.g.*, MARK KESTER, KENT E. VRANA & KELLY D. KARPA, ELSEVIER'S INTEGRATED REVIEW PHARMACOLOGY 64 (2d ed. 2012) (describing how "slow acetylators" are at greater risk of drug-related toxicities from taking isoniazid).

188. *See* Imwinkelried, *supra* note 23, at 97 ("In the early 20th century, the normal pattern was that an individual expert would personally conduct by hand a single test . . . . [Today] the witness will testify about the results of an automated forensic technique that he or she oversaw.").

189. *Supra* Parts II, III.

protect the legitimacy of criminal trials, courts must require additional sources of verification. This Part will analyze three additional measures that courts can take: (1) mandating source code disclosure; (2) scrutinizing programming practices; and (3) emphasizing software expertise.

Source code review has been the focus of most scholars and defendants thus far. It is an effective complement to verification studies for many kinds of programs, but not all of them. Due to this limitation, industry standards often include certain error-resistant programming practices. These practices can be particularly useful when software is too complex for effective code review, but it is challenging to assess how much reliability they provide. As such, this Part ends with an argument for a greater emphasis on software expertise. To navigate the complex field of software assurance, courts should use software experts' testimonies to flexibly determine the risk of software error and the need for additional verification.

A. Disclosing Source Code

Access to source code alleviates many of the issues with relying on validation studies. It allows defendants to understand the basic logic of the opposing software, helping them identify whether there are relevant conditions that validation studies did not test.[190] The value of source code disclosure for uncovering errors missed by validation studies is not speculative. As noted in Part I, previous disclosure orders for source code have repeatedly uncovered serious errors.[191]

This notion—that adversarial review is a powerful tool for discovering the truth—is hardly foreign to American courts. The Supreme Court has noted that "[t]he very premise of our adversary system of criminal justice is that partisan advocacy on both sides of a case will best promote the ultimate objective that the guilty be convicted and the innocent go free."[192] Likewise, faced with great difficulties in assuring software performance, the computer science community has long recognized the importance of adversarial and open-source review.[193]

190. *See* Kroll et al., *supra* note 74, at 648 n.41.
191. *Supra* notes 49–56 and accompanying text.
192. Herring v. New York, 422 U.S. 853, 862 (1975).
193. ERIC S. RAYMOND, THE CATHEDRAL & THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY 19 (2008) ("Given enough eyeballs, all bugs are shallow."); *see also* Jim Hamerly, Tom Paquin & Susan Walton, *Freeing the Source: The Story of Mozilla*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 197, 197–206 (Chris DiBona, Sam Ockman & Mark Stone eds., 1999) (describing Mozilla's shift to open source); Steven

Attempting to break software is the "most frequent[] and common[]" method to secure it.[194] Bug bounty programs—which reward the public for finding errors—are well established, originating with sophisticated entities like the Pentagon and Google.[195] While the effectiveness of source code review can vary,[196] it boasts strong advocates in the computer science community.[197]

Some scholars have argued that the time constraints in criminal trials will prevent defendants from performing meaningful source code analysis.[198] The errors found following previous disclosure rulings suggest otherwise.[199] First, source code review does not have to analyze a program in its entirety to be effective. Line-by-line analysis is not necessary to understand the basic logic of a program, which on its own can reveal if key conditions were not tested or if the program's overarching logic is flawed. Second, source code review does not have to be comprehensive to help ensure compliance with development standards.[200] Third, even if only a minority of defendants have the resources to conduct a meaningful analysis, the results of their review can help anyone accused by the same software. When one defendant identified errors in FST, the existence of these errors became widely known.[201] The program was discontinued, and its source code was unsealed—helping everyone wrongfully accused due to FST errors and everyone who would have been if the program had continued.

Certainly, the degree to which source code is used to find simple errors, design errors, and compliance failures will vary from case to case. For simpler

Vaughan-Nichols, *Coverity Finds Open Source Software Quality Better Than Proprietary Code*, ZDNET (Apr. 16, 2014, 11:01 AM), https://www.zdnet.com/article/coverity-finds-open-source-software-quality-better-than-proprietary-code [https://perma.cc/Y5CA-9DRY] ("[T]he numbers don't lie and the 2013 Coverity Scan Open Source Report . . . found that open source had fewer errors per thousand lines of code (KLoC) than proprietary software."); Chessman, *supra* note 61, at 223–24 n.323 (reporting that "78 percent of companies in the world rely on open-source software in their computer programs").

194. Brad Arkin, Scott Stender & Gary McGraw, *Software Penetration Testing*, 3 IEEE SEC. & PRIV. 84, 84–87 (2005).

195. Akemi Takeoka Chatfield & Christopher G. Reddick, *Crowdsourced Cybersecurity Innovation: The Case of the Pentagon's Vulnerability Reward Program*, 23 INFO. POLITY 177, 177–78 (2018).

196. *See, e.g.*, Jing Wang, Patrick C. Shih & John M. Carroll, *Revisiting Linus's Law: Benefits and Challenges of Open Source Software Peer Review*, 77 INT'L J. HUMAN-COMPUTER STUD. 52 (2015).

197. *See, e.g.*, Ince, Hutton & Graham-Cumming, *supra* note 31 ("[A]nything less than release of actual source code is an indefensible approach for any scientific results that depend on computation . . . .").

198. *See* Wexler, *supra* note 18, at 1373–74.

199. *See supra* notes 49–56 and accompanying text.

200. *See* Gates et al., *supra* note 48; Hotten, *supra* note 48; Glinton, *supra* note 48.

201. *See supra* notes 52–55 and accompanying text.

programs, source code review alone may be sufficient to augment validation studies. In other instances, verification may rely more heavily on proper development practices,[202] and source code review could mainly serve to ensure compliance with these practices.

The role of source code can be different still for AI-based systems, where the training data and parameters heavily influence the end product's behavior. In these cases, other forms of disclosure may be more relevant.[203] Source code review can still provide relevant information, such as how the training was performed.[204] But the usefulness of this information again varies depending on the software.

Against source code disclosure, developers often claim trade secret protection or point to the commercial costs of disclosure.[205] While these arguments may have relevance in other contexts,[206] they are inapplicable where disclosure is necessary to meet admissibility standards. That is, admissibility standards do not force developers to relinquish their trade secrets. It is the developers' choice whether they want to open their products to level of review needed to comply with admissibility standards. Likewise, expense is not a part of the *Daubert* or *Frye* inquiry. If a developer's business model cannot survive complying with evidentiary rules, then they are simply not capable of making an admissible product.

B. Examining Programming Practices

While source code review and validation testing are the focus of legal scholarship, they are not the only tools for software verification. This Subpart explains how developers can use techniques like defensive programming and multiple-version software to write code that is less likely to fail when errors occur.[207] Developers can also reduce error formation in the first place using practices like traceability analysis.[208] As both categories result in more trustworthy

202. *See infra* Subpart IV.B.
203. Brandon L. Garrett & Cynthia Rudin, *Interpretable Algorithmic Forensics*, PROC. NAT'L ACAD. SCIS., Oct. 10, 2023, at 1, 4 (describing the elements of interpretable forensic AI).
204. Such details include the parameters, hyperparameters, and the model itself. *See* Li Yang & Abdallah Shami, *On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice*, 415 NEUROCOMPUTING 295, 296–99 (2020).
205. *See, e.g.*, Commonwealth v. Foley, 38 A.3d 882, 888–89 (Pa. Super. Ct. 2012).
206. For a discussion of the trade secrets objection outside of admissibility hearings, *see generally* Wexler, *supra* note 18.
207. *See* NY TrueAllele Review, *supra* note 136.
208. *See* COEST, *supra* note 157.

code, courts—like the computer science community[209] and multiple federal agencies[210]—can look to the presence of such practices to determine whether software is sufficiently reliable.

1. Error-Resistant Code

There is a wide variety of techniques aimed at making code more error resistant. One illustrative example is multiple-version software, which operates on a similar principle to overengineering. Akin to using multiple independently sufficient beams to hold up the same bridge, a multiple-version program can use multiple independent methods to answer the same question. By checking first that those methods provide the same answer, the program can offer more certainty.

As noted above, the promotion of this method is controversial among software reliability experts.[211] The different versions must be meaningfully different, which is difficult to verify. As such, the technique risks increasing a developer's confidence in the software without actually improving reliability.

Another technique, defensive programming, faces a similar problem. Defensive programming adds code that checks for errors preemptively.[212] Such checks are standard and effective where errors are predictable,[213] but again rely on some prediction of what kinds of errors are likely to arise.

Ultimately, error-resistant code can be effective in some cases, but understanding its effectiveness requires a complicated analysis of method diversity and error risks.

2. Error-Resistant Coding

Software can also be more reliable when it is created using less error-prone development processes.[214] Such processes are typically implemented through

209. Patrick Rempel & Parick Mäder, *Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality*, 43 IEEE TRANSACTIONS ON SOFTWARE ENG'G 777 (2017) ("Requirements traceability is broadly recognized as a critical element of any rigorous software development process . . . .").
210. *See* COEST, *supra* note 157.
211. *See supra* notes 79–81 and accompanying text.
212. Jean-Louis Boulanger, *Technique to Manage Software Safety*, *in* CERTIFIABLE SOFTWARE APPLICATIONS 135–36 (2016); *see, e.g.*, Marco Guarnieri, Boris Köpf, Jan Reineke & Pepe Vila, *Hardware-Software Contracts for Secure Speculation*, IEEE SYMP. ON SEC. & PRIV. (2021).
213. *See, e.g.*, Oftedal, *supra* note 97 (discussing the specific errors to check for when "validating input").
214. *See* U.S. FOOD & DRUG ADMIN., GENERAL PRINCIPLES OF SOFTWARE VALIDATION; FINAL GUIDANCE FOR INDUSTRY AND FDA STAFF 20 (2002) ("Firms frequently adopt specific coding

guidelines that require developers to document specific information on how their code works, create detailed explanations of the expected behavior, and evaluate whether each segment of code operates as expected.[215] These checks have important internal functions and can dramatically reduce the rate of errors.[216] They improve communication and make programs more comprehensible by humans, which reduces the various kinds of misunderstandings discussed above.[217]

One practice for improving program comprehension is maintaining traceability.[218] It is a well-recognized method, mandated by federal regulators such as the Food and Drug Administration (FDA) and Federal Aviation Administration (FAA).[219] At the highest level, traceability is a formal method of tracking software requirements and the corresponding code. It requires that any high-level requirements can be traced to specific low-level requirements, and vice versa. In the development of software for an autopilot system, a high-level requirement might be that the system can maintain a specified altitude during flight. Low level requirements could be (1) accurately acquiring the current altitude from the aircraft's altimeter, (2) defining a permissible deviation from the desired altitude, and (3) communicating with the aircraft's control surfaces for altitude corrections. Tracking and tracing these requirements allows developers

guidelines that establish quality policies and procedures related to the software coding process."); *see also* NANCY G. LEVESON, SAFEWARE: SYSTEM SAFETY AND COMPUTERS (1993) ("One obvious lesson is that most accidents are not the result of unknown scientific principles but rather of a failure to apply well-known, standard engineering practices. A second lesson is that accidents will not be prevented by technological fixes alone, but will require control of all aspects of the development and operation of the system.").

215. *See* U.S. FOOD & DRUG ADMIN., *supra* note 214 ("Code comments should provide useful and descriptive information for a module, including expected inputs and outputs, variables referenced, expected data types, and operations to be performed. Source code should also be evaluated to verify its compliance with the corresponding detailed design specification. Modules ready for integration and test should have documentation of compliance with coding guidelines and any other applicable quality policies and procedures.").

216. Such explanations, such as "code comments," are important for effective internal development. *See* Sebastian Nielebock, Dariusz Krolikowski, Jacob Krüger, Thomas Leich & Frank Ortmeier, *Commenting Source Code: Is It Worth It for Small Programming Tasks?*, 24 EMPIRICAL SOFTWARE ENG'G 1418 (2019).

217. *See supra* notes 62–73 and accompanying text.

218. *See generally* B. Scott Andersen & George Romanski, *Verification of Safety-Critical Software*, 54 Commc'ns ACM 52 (2011) (explaining key concepts in traceability).

219. FED. AVIATION ADMIN., ORDER 8110.49A, SOFTWARE APPROVAL GUIDELINES 1–3 (2018) ("Inspecting the traceability from system requirements to software requirements to software design to source code to object code to test cases and procedures to test results."); U.S. FOOD & DRUG ADMIN, *supra* note 214, at 21 ("A source code traceability analysis is an important tool to verify that all code is linked to established specifications and established test procedures.").

to identify and address high-risk sources of error. If, for example, a lower-level item cannot be directly traced to a higher-level item, then the lower-level item may have unintended behaviors and merit further review.[220] If part of the altitude maintenance code communicates with the landing gear, the developers should figure out why. This process can also help development teams retain a better understanding of their product's overall logic, making vulnerabilities more obvious.

C. Emphasizing Software Expertise

Source code review, error-resistant coding techniques, and error-resistant development practices all have merit in different circumstances. Source code review can be useful for identifying simple errors, design errors, or compliance failures—depending on factors such as the software's complexity, structure, and use of AI. The effectiveness of multiple-version software depends on a complicated diversity analysis, and the effectiveness of traceability depends on the precision with which it is implemented.

To understand when these practices are necessary and whether they have been sufficiently implemented, courts must emphasize software expertise in their assessments of forensic software. Heavier reliance on computer science experts—through both their testimony in court and guidance creating standards—would address concerns that peer review without source code is illegitimate[221] and allow for software testing that is more aligned with the recommended practices in computer science.[222]

Additionally, by using software experts' testimonies, courts can flexibly require source code disclosure and error-resistant programming practices when those methods are appropriate and only when they are appropriate. Contrast this with the recently-proposed legislative requirement for developers to always disclose the source code for their products.[223]

220. *See* Andersen & Romanski, *supra* note 218.

221. *See* CARRIER, *supra* note 31; *see, e.g.*, Ince et al., *supra* note 31; Morin et al., *supra* note 31.

222. *See, e.g.*, Kroll et al., *supra* note 74, at 661 n.91 (discussing a technique called "white-box testing"); *see* Sean Gallagher, *Microsoft Launches "Fuzzing-as-a-Service" To Help Developers Find Security Bugs*, ARS TECHNICA (Sept. 27, 2016, 8:21 AM), https://arstechnica.com/information-technology/2016/09/microsoft-launches-fuzzing-as-a-service-to-help-developers-find-security-bugs [https://perma.cc/24W7-JRF4].

223. Justice in Forensic Algorithms Act of 2021, H.R. 2438, 117th Cong. (2021) ("[T]he defendant shall be accorded access to both an executable copy of and the source code for the version of the computational forensic software . . . ."). Note that the bill would also charge the National

To be clear, given the computer science literature on source code disclosure,[224] this proposal is likely to bring courts closer to the assurance standards accepted in the computer science scientific community. But it would be closer still—and better aligned with current law—to determine the need for source code disclosure on an individualized basis using computer science testimony and standards.

This is not a stretched reading of *Daubert* and *Frye*. Indeed, a handful of courts have already recognized that current admissibility standards require judges to treat computer science as an independent, necessary scientific discipline. In *Pickett*, the court held in its review of TrueAllele that techniques which integrate multiple scientific disciplines must be verified across each discipline.[225]

And, as explored above, this reading is correct. Software evaluation is sufficiently complicated to fall outside of the expertise of forensic scientists. Without accounting for the prevalence and elusiveness of software errors, courts are liable to underestimate error rates and permit insufficient standards. But by distinguishing between scientific disciplines, courts can ensure that they are not overextending the testimony of forensic specialists and that admitted scientific evidence has general acceptance in each relevant scientific community.

CONCLUSION

This Essay has argued that courts should use computer science expertise to help determine whether software-generated evidence meets the standards for scientific admissibility. Software errors can occur through many different mechanisms—regardless of whether the underlying scientific principles are correct. These errors can be hard to catch or mitigate, so the computer science community has developed sophisticated methods for finding errors and reducing the danger from unfound errors. To correctly apply their admissibility standards for scientific evidence, courts must require the moving party to present testimony or standards from experts who have knowledge of these issues in software verification. This additional testimony will allow

Institute of Standards and Technology (NIST) with creating standards for evaluating forensic software. *Id.* § 2(a). Transitioning this standards-setting role to a technology-oriented agency is in line with the judicial recommendations in this Part.

224. *See supra* Subpart IV.A.

225. State v. Pickett, 246 A.3d 279, 311 (N.J. Super Ct. App. Div. 2021) ("TrueAllele's software integrates multiple scientific disciplines, therefore requiring cross-disciplinary validation to determine reliability.").

for a more meaningful determination of where source code review and other assurance measures are necessary to properly gatekeep scientific evidence in the courtroom.

More broadly, an appreciation of the distinct expertise within software assurance can prevent courts from continuing to over-rely on novel technologies. While DNA matching systems are currently the subject of scrutiny, they were preceded by fingerprint analyzers, which were preceded by breathalyzers. Each time, courts held that source code disclosure was unnecessary to sufficiently vet the software, and each time, serious errors were found when the code was eventually released. Recognizing the limits of forensic experts—and, potentially more important, recognizing the expertise of computer science experts—can provide the missing perspective that is needed for courts to properly balance the opportunities and limitations of emerging forensic technologies.