Code-ifying the Law: How Disciplinary Divides Afflict the Development of Legal Software

NEL ESCHER, University of Michigan, USA
JEFFREY BILIK, University of Michigan, USA
NIKOLA BANOVIC, University of Michigan, USA
BEN GREEN, University of Michigan, USA

Proponents of legal automation believe that translating the law into code can improve the legal system. However, research and reporting suggest that legal software systems often contain flawed translations of the law, resulting in serious harms such as terminating children's healthcare and charging innocent people with fraud. Efforts to identify and contest these mistranslations after they arise treat the symptoms of the problem, but fail to prevent them from emerging. Meanwhile, existing recommendations to improve the development of legal software remain untested, as there is little empirical evidence about the translation process itself. In this paper, we investigate the behavior of fifteen teams—nine composed of only computer scientists and six of computer scientists and legal experts—as they attempt to translate a bankruptcy statute into software. Through an interpretative qualitative analysis, we characterize a significant epistemic divide between computer science and law and demonstrate that this divide contributes to errors, misunderstandings, and policy distortions in the development of legal software. Even when development teams included legal experts, communication breakdowns meant that the resulting tools predominantly presented incorrect legal advice and adopted inappropriately harsh legal standards. Study participants did not recognize the errors in the tools they created. We encourage policymakers and researchers to approach legal software with greater skepticism, as the disciplinary divide between computer science and law creates an endemic source of error and mistranslation in the production of legal software.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI)**; • **Applied computing** → **Law**.

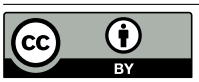
Additional Key Words and Phrases: law, automated legal systems, legal software development

ACM Reference Format:
Nel Escher, Jeffrey Bilik, Nikola Banovic, and Ben Green. 2024. Code-ifying the Law: How Disciplinary Divides Afflict the Development of Legal Software. *Proc. ACM Hum.-Comput. Interact.* 8, CSCW2, Article 398 (November 2024), 37 pages. https://doi.org/10.1145/3686937

1 Introduction

Proponents of legal automation believe that translating the law into software makes legal processes more accessible, efficient, and consistent. The public sector has adopted algorithms to manage vital government services from social welfare [46] to criminal justice [2]. Nonprofits offer free do-it-yourself applications that guide laypeople through legal processes such as eviction [44]

Authors' Contact Information: Nel Escher, kescher@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Jeffrey Bilik, jbilik@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Nikola Banovic, nbanovic@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Ben Green, bzgreen@umich.edu, University of Michigan, Ann Arbor, Michigan, USA.



ACM 2573-0142/2024/11-ART398
https://doi.org/10.1145/3686937

and immigration [32]. New products aim to replace many fields of law with consumer software, including tax preparation [64], wills [74], and prenuptial agreements [28].

Despite the optimism motivating these tools, recent reporting and research suggest that legal software systems often contain flawed translations of the law, resulting in serious harm. Automated legal systems have misapplied rules, informing eligible families that they did not qualify for subsidized child care [18], extracting payments for debts that were not owed [42, 75], and erroneously cutting access to necessary social services [19]. Errors can cause fatal deprivations, forcing people to go without healthcare or shelter to which they are legally entitled, and can take years to resolve [75]. Many of these problems are traceable to flaws that are directly embedded in the software [12, 15, 18]. These incidents suggest that when developers translate law into code, they often alter legal rules that were established through legitimate legislative processes. These changes evade democratic oversight and circumvent the rule of law.

In response, scholars have suggested strategies for improving the translation process. One proposal focuses on delineating which laws are suitable for translation into software. Ambiguous laws have multiple meanings and so require interpretation and discretion; attempting to translate them into code has led to problems, as it is unclear how a formal programming language should represent the plural possible interpretations. Thus, some scholars argue that only well-defined, concrete laws are amenable to automation [12, 17, 25, 49]. For instance, they claim tax law is a good candidate for translation due to its predictable rules and formulas, while criminal law, which involves less predictable judgments around culpability, is not [23]. This approach explains problems in legal software as the result of inappropriate law selection. Provided that a clear, determinate law is selected, much computational law scholarship assumes that translation into code is straightforward [40, 61].

Another proposal for improving the translation process focuses on who should be involved in translating laws into software. This response explains problems in legal software as the result of computer programmers who lack sufficient legal expertise [9, 12]. Therefore, one commonly suggested intervention is incorporating a legal expert into development teams [5, 12, 36, 71]. This advice is motivated by a belief that interdisciplinary teams would create software that better encodes the accepted meaning of the law and considers existing institutional practices [7].

Despite the support for these two strategies, a lack of empirical evidence about the translation process itself presents a significant barrier to improving legal software. Since we do not know how errors actually get embedded into legal software, we cannot develop reforms that reliably prevent those errors. Notably, beyond the concerns raised by advocates of the prior two strategies, one additional challenge looms large: the divide between legal and computer science reasoning. While lawyers are taught to balance economic, cultural, moral, and political considerations, computer scientists are taught to follow internally rule-bound reasoning and dismiss concerns outside of this frame [26]. Even if development teams include lawyers, these epistemic differences may lead to practical and ideological conflicts in the translation process. In other words, the development of legal software might be shaped by the disciplinary assumptions and practices of the specific people involved. Ultimately, to improve legal software, we must first learn more about the dynamics and challenges that emerge within the translation process.

In this work, we investigate the behavior of computer scientists and legal experts as they develop legal software. We conducted an experimental study framed around four research questions:

RQ1: How do computer scientists and lawyers approach the task of translating law into code?
RQ2: What processes and strategies do teams use to complete the task?
RQ3: How do teams encode the law in software?
RQ4: How do computer scientists and lawyers reflect on their work?

We provided fifteen teams with a statutory excerpt from the U.S. Bankruptcy Code to translate into legal software.[1] We employed two conditions for our study: computer science teams, composed of 1–2 computer scientists, and interdisciplinary teams, composed of a computer scientist and a legal expert. Of the fifteen groups that completed the task, five were composed of individual computer scientists, four were pairs of computer scientists, and six were one computer scientist with one lawyer. Computer scientists were advanced undergraduates or graduate students with web programming experience, and legal experts were upper-level law students. We presented all teams with a translation task that follows a common design for legal software: a lay user provides information about their circumstances and then receives a determination about how a law applies to them. At the end of each study session, we asked participants how their tools should be used in practice. We captured the entire development process, collecting over 100 hours of laptop screen captures, audio transcripts, text communications between teammates, and snapshots of programming code. We then performed interpretative qualitative analysis on the composite transcripts to answer our four research questions.

Our findings demonstrate that the epistemic divide between computer science and law makes the translation of law into software a fertile ground for errors, misunderstandings, and policy distortions. Practitioners conceived of the translation task through distinct modes of reasoning, supplied by their disciplinary background in computer science or law (RQ1). Consequently, computer scientists and legal experts employed disparate approaches for interpreting the law and designing software, and lacked strategies to bridge the gap between their approaches (RQ2). These breakdowns meant that, even when development teams included a legal expert, they produced software that typically provided incorrect legal advice and overstated how strict the law is (RQ3). Finally, because they lacked knowledge about the other field, participants (especially computer scientists) expressed unwarranted confidence in the utility of their tools (RQ4).

These results suggest the need for greater skepticism about the desirability and reliability of legal software. The details of a translation depend not just on the structure of a law and the capabilities of code, but also on the practitioners involved, their disciplines, and their interactions with one another. Divergent modes of reasoning make it difficult for development teams to produce correct software and to recognize mistranslations in their tools. There is some evidence to suggest that these dynamics are present in real-world development contexts and can help explain the legal distortions that persistently emerge in legal software. First, the modes of reasoning we observed align with widespread educational practices for computer scientists and lawyers. Second, many legal software tools are developed under similar circumstances, with short timelines and practitioners who lack interdisciplinary training. Thus, policymakers and researchers should view flaws in legal software as an endemic issue rather than one-off bugs. Attempts to ameliorate the real and extensive harms of these flaws must be grounded in empirical evidence about why mistranslations emerge and what interventions work to prevent them.

The rest of the paper proceeds as follows. Section 2 reviews prior research related to the flaws of legal software. In Section 3, we present the methods for our experimental study and the data we collected about each team's development process. The following four sections present findings for our four research questions: Section 4 characterizes the modes of reasoning that participants exhibited, Section 5 examines the development strategies used across team types, Section 6 focuses on how the final tools represented the bankruptcy statute, and Section 7 presents participant attitudes about the utility of their software. In Section 8, we discuss the implications of our results and suggest next steps for research and reform.

[1] A statute is a written law created by a legislative body. Congress enacts federal statutes in the United States, meaning these laws affect the entire country.

2 Background and Related Work

Section 2.1 overviews the harms and proposed causes of flawed translations of the law in legal software. Section 2.2 presents scholars' guidance on which laws ought to be translated into code, and who ought to engage in translation work. Section 2.3 discusses difficulties for the translation process that challenge the efficacy of existing guidance.

2.1 Legal Software Failure

Legal researchers have documented many automated systems that do not correctly encode the law. These errors can insert new rules, such as when the Colorado Benefits Management System asked food assistance applicants whether they are "beggars," despite that term not appearing in any legal text or supplemental administrative materials [12]. Problems in modern software systems can strip people of their rights and impose arbitrary duties, all without human oversight and while being obscured by complex code. Getting the law wrong can have devastating effects, such as cutting off healthcare benefits for the elderly [9, 19], denying aid to immigrants experiencing domestic violence [70], and undermining government services by falsely accusing welfare recipients of owing debts [6].

Yet, it is difficult to detect and challenge such errors. Systems are often proprietary and closed off from public inspection [9, 12, 21]. Vulnerable people may not know how the law ought to function, as many areas of the law are difficult to understand without legal training. Even if errors are detected, it is difficult to contest them, as systems rarely build in routes for appeal [19]. People have died while waiting for their healthcare to be reinstated after an automated revocation [19]. Audit studies and court cases have identified bugs in algorithms [15, 18], but these retrospective analyses can only speculate about the source of flaws.

When scholars consider the source of problems, they theorize that flaws in legal software are attributable either to improper law selection or to programmer ignorance. Some computational law and legal scholarship argues that problems stem from misjudging whether a particular law is sufficiently unambiguous for computerization [23, 40, 61]. These scholars point out that legal software projects fail for domains that require construing legal ambiguities or making policy choices, both of which are inappropriate for a computer to decide [61]. Other legal scholars point to the propensity of programmers to make mistakes during the translation process—even when dealing with unambiguous legal rules, citing either the general ubiquity of bugs in code [8, 9, 54, 73] or developers' lack of expertise in law [11, 12, 31, 36]. These scholars also argue that programmers' ideological preferences and biases may lead to purposeful distortion of the law [8, 12, 53]. This latter critique falls in line with critical scholars' broader claims that any translation of law into code will displace the open texture of legal interpretation and cause harm [13, 29].

2.2 Guidance for Legal Translation

Recommendations about which laws ought to be translated into software theorize that computation is useful when legal practice in a given domain can be characterized as clear, consistent, or determinate. Surden's "variable determinacy thesis" provides an explicit and influential framework for conceptualizing which laws are translatable into code. Under this framework, a set of laws can be modeled computationally if legal decision-makers' choices are constrained such that legal outcomes in the domain are predictable. Such contexts can be articulated as explicit, fixed rulesets. For instance, personal income tax laws consist of a constrained ruleset; legal officials have little leeway to come to different conclusions when applying these rules to individual cases, so this is a context for which computation could be useful. Because legal contexts fall on a spectrum of determinacy, rules can be more mechanical, and thus more determinate and possibly representable

in code, or rules can be more open-ended, and thus more indeterminate and less likely to be a candidate for automatic application by software. [61]

Though many scholars acknowledge this divide between determinate and indeterminate law, they disagree about where laws tend to fall along this spectrum. Legal futurists argue that all law can be translated into code, as computational techniques promote accurate and consistent applications of law in part by eliminating or discouraging ambiguity and interpretive openness [1, 24, 48, 62]. Critical scholars often view computational techniques as solely useful for "the mechanical application of disambiguated rules," which renders most law unsuitable for automation since it is socially embedded in judicial institutions that perform open reasoning [29]. In practice, though, practitioners and governing bodies have supported legal software projects in a subset of domains. Their approach tracks Surden's, permitting computation where law provides rule-like instruction, but encouraging human intervention when applying relatively vague standards [12, 25, 49].

Scholars and practitioners have also made recommendations about who ought to translate law into code, suggesting that legal experts work together with software engineers to better align legal software with the operation and intent of the law [5, 12, 36, 71]. As software appears to deviate from established legal practice, the addition of lawyers is intended to improve fidelity to existing jurisprudence and guide development decisions according to legitimate considerations. Some studies have expanded on this recommendation with further directions, for instance: "legal analysts and software engineers must learn new ways to collaborate in their decision making using shared artifacts to build shared understanding" [7].

However, the guidance that scholars offer for developing legal software—which laws should be translated, and who should participate in translation—remains untested. As a result, it is unclear how these recommendations address recent empirical findings that uncover complex challenges in the development process of legal software.

2.3 Challenges of Legal Translation

Institutional conditions and constraints on the development process can impact how law is translated into software. Interviews with designers and lawyers working in the tech sector suggest that many development teams are homogenous, siloed groups of engineers and that lawyers have hesitated to insert themselves in the design process for fear of obstructing technical progress [71]. Development can depart from accepted software engineering practices; for instance, government contractors often omit testing from their project proposals to submit lower bids [12]. Institutional constraints such as limited funding can rush development, and team members' divergent expectations may lead to unclear project requirements for legal software [36]. Prioritizing certain policy outcomes can also lead to flawed translations. Post-mortem assessments of the Robodebt scandal in Australia, which charged citizens with false debts based on inaccurate assessments of welfare eligibility, attributed its haphazard design to designers' efforts to be "tough on welfare" and reduce government spending [10]. These attitudes may have contributed to rapid development and a lack of testing for the Robodebt system [52]. Though this work does not capture the precise mechanisms that produce flawed legal interpretations, it demonstrates that selecting a determinate law does not lead simply to an accurate software representation.

Challenges may arise due to the differences in the disciplinary training of lawyers and computer scientists. A recent report on interdisciplinary work in computer science and the law declares that "differences in standards, language, methods, and culture impede professors and other academic researchers who want to collaborate with colleagues on the other side of this divide" [3]. Dominant thinking in computer science tends toward formalism, an emphasis on bounded and rigid sets of rules with which to deduce next steps [26]. The legal field has developed a realist response to formalism, arguing that external considerations like politics or business practices influence judicial

decision-making [35]. For instance, a legal formalist approach would hold all noncompete clauses to be enforceable, following the plain language of employment contracts. In contrast, a realist approach might prioritize competition and worker welfare, and so release an employee from a promise not to work for competitors in some circumstances [37]. It is unclear how to reconcile the algorithmic formalism characteristic of computer science and the contextually-informed realism of the legal sphere when designing legal software.

Working across disciplines is a complex process that requires significant work. When development teams include legal experts and computer scientists, communicating across these perspectives often involves additional conceptual scaffolding. For instance, collaborators may need to construct "boundary objects"—shared spaces for embedding and relaying their disparate ideas and aims [22, 59]. Collaborators' distinct backgrounds may mean they prioritize different objectives that trade-off against those of their teammates. Such ideological mismatch meaningfully constrains the benefits of interdisciplinary collaboration [39]. For example, the developers of the system that gave rise to the Robodebt scandal could have implemented standards presented by lawyers and state officials, but instead prioritized an efficient design that saved the government money [10].

Together, these issues suggest that the selection of determinate law and promotion of interdisciplinarity may not ameliorate stubborn issues with the translation of law into code. It is imperative to investigate the translation process directly to address the serious harms that stem from software incorrectly representing the law.

3 Methods

We constructed an experimental study design that asked experts in computer science and law to translate a piece of legal text into legal software. Our experimental study followed how experts reason about and construct legal tools. We focused on the interpersonal collaboration, conflict, and iterative efforts involved in creating a legal application. To test advice on translatability, we selected a legal statute that contains both determinate and indeterminate elements. To test the effects of interdisciplinarity, teams in one condition were composed of computer scientists; in the other condition, a computer scientist was paired with a legal expert. Teams built legal advice tools that determine whether a particular statutory provision applies to a user's case.

3.1 Statutory Excerpt Selection

We began by selecting a legal statute for participants to translate into software. Having all teams work on the same legal statute allows us to compare development processes across teams. We sought an excerpt that includes both rule-like and standard-like elements. Rule-like elements in law are determinate, meaning that there is widespread agreement about their meaning such that their application produces predictable legal outcomes [61]. Standard-like elements in law are indeterminate, as they permit judges to use their discretion to fashion individualized outcomes [55]. Selecting for these qualities allows us to trace how determinacy affects the translation process for legal software development.

Based on these criteria, we selected an excerpt from the United States Bankruptcy Code, 11 USC 523(a), which has both rule-like and standard-like elements (see Table 1). Consumer bankruptcy is a legal process involving financial restructuring that intends to give individuals a fresh start. However, under U.S. law, there are certain types of debts whose cancellation presents hazards, and so are categorically prohibited from forgiveness. 11 USC 523(a) contains a list of debts that cannot be discharged through bankruptcy. For instance, a debtor will not be released from a "domestic support obligation," per 11 USC 523(a)(5), as canceling child support or alimony payments deprives others of necessary income.

Table 1. Statutory excerpt from the United States Bankruptcy Code that participants translated into software.

11 U.S. Code § 523—Exceptions to discharge

(a) A discharge under section 727, 1141, 1192, 1228(a), 1228(b), or 1328(b) of this title does not discharge an individual debtor from any debt—
  (...)
  (4) for fraud or defalcation while acting in a fiduciary capacity, embezzlement, or larceny;
  (5) for a domestic support obligation;
  (...)
  (8) unless excepting such debt from discharge under this paragraph would impose an undue hardship on the debtor and the debtor's dependents, for—
    (A)
      (ii) an obligation to repay funds received as an educational benefit, scholarship, or stipend

Our selection contains rule-like elements with a determinate structure because there is a predictable relationship that the subsections (a)(4), (a)(5), and (a)(8)(A)(ii) have to one another. The sole and undisputed approach used by courts to assess the separate statutory subsections in (a)(4), (a)(5), and (a)(8)(A)(ii) is that they describe different types of debts which must be separately considered. A debtor may have many distinct debts, such as medical bills, student loans, and child support payments. Each debt must be evaluated independently; for instance, medical bills and student loans may be forgiven, but child support payments cannot.

This excerpt also engages with indeterminate standard-like elements because it contains an ambiguous phrase—"undue hardship"—which requires judicial discretion to interpret and apply to each case. Though federal and private student loans are typically not dischargeable during bankruptcy, (a)(8) provides an exception if repaying the debt would "impose an undue hardship on the debtor and the debtor's dependents." Judges have attempted to give meaning to this phrase through practices of statutory interpretation, such as considering legislators' intent when enacting this law. However, courts have not agreed on a single approach for determining undue hardship. Two competing standards now predominate to guide court determinations of undue hardship (see Table 2) [20].

Table 2. Two judicial standards used to interpret "undue hardship" in 11 U.S.C. § 523(a)(8)(A).

The *Brunner* Standard [66]	**Totality-of-circumstances Standard** [51]
Debtor must prove all of the following: 1) that the debtor cannot, based on current income and expenses, maintain a "minimal" standard of living for himself or herself and his or her dependents if forced to repay the loans 2) that this state of affairs is likely to persist for a significant portion of the repayment period of the student loan 3) that the debtor has made good faith efforts to repay the loans	Judges consider each of the following: 1) the debtor's past, present, and reasonably reliable future financial resources; 2) the debtor's reasonable and necessary living expenses; and 3) any other relevant facts and circumstances

In construing undue hardship, all circuits[2] except for the First and Eighth Circuit follow the *Brunner* standard [66]. The remaining circuits follow the totality-of-circumstances standard, which

[2]Circuits are federal appeals courts in the United States. They review cases from lower courts within their jurisdiction. There are thirteen circuits in total, with twelve corresponding to geographic regions. Decisions made by a circuit court are binding on the lower courts within its jurisdiction.

is more lenient than *Brunner* in that it expands the sources of support for a finding of undue hardship [68, 69]. This split means that most judges are bound to follow a particular standard for determining undue hardship based on the geographic location of their court. These strategies for addressing indeterminacy give some structure to decision-making but still permit some discretion as judges must decide whether the debtor's circumstances satisfy these standards.

In this study, these determinate and indeterminate legal elements map onto particular software features. As scholars claim that determinate elements create consistent outcomes, we evaluate whether the eligibility predictions issued by the tools are consistent with the law. We comprehensively test the tool's responses by creating hypothetical debtors with different types of debt and evaluate if the tool correctly assesses bankruptcy eligibility for each case. As indeterminate elements require interpretation guided by judicial practice, we consider how tools reproduce the two dominant legal standards used by courts to determine undue hardship. We look at questions assessing undue hardship in the user interface of each tool, then at the internal logic that each tool uses to process user answers and produce a determination about undue hardship.

3.2 Study Design

We structured our study to test for the effects of interdisciplinarity within teams. There are two conditions in our study:

- **Computer Science Teams.** One computer scientist worked alone, or two computer scientists worked together.
- **Interdisciplinary Teams.** One computer scientist worked with one legal expert.

3.3 Participants

Following the study design, we recruited two types of participants: computer scientists and legal experts. We recruited both groups using email lists for a large public research university in the United States. We required computer science participants to be either advanced undergraduate or recent graduates, possess software engineering experience in an industry setting, and self-report intermediate Javascript proficiency. We also required that participating computer science students and software engineers had not completed any graduate coursework in law. For legal experts, we recruited second and third-year law students with professional legal experience who had not completed any college-level coursework in computer science. All participants were in the immediate geographic vicinity of the university and were at least 18 years old.

A screening survey enforced our inclusion/exclusion criteria by requiring prospective participants to report their educational and professional experience. After consenting to participate and completing the study, participants could optionally provide additional demographic information in a follow-up survey.

Our experiment included 25 participants—19 computer scientists and six legal experts. We divided these participants into 15 experimental trials: five trials with a computer scientist working alone, four trials with two computer scientists working together, and six trials with a computer scientist and a legal expert working together. Trials took place between April and December 2021.

We provided participants with open-text responses for demographic information. Among computer scientists, seven (37%) were women, 11 (58%) were men, and one decided not to share. Nine (47%) identified as Asian or South Asian, six (32%) identified as white, one identified as both Asian and white, and three chose not to share. Among legal experts, all were men. One identified as Black, and five identified as white. We compensated each participant $20 per hour of involvement in the study task, to a maximum of $100 for 5 hours.

3.4 Study Software and Apparatus

Given the ongoing COVID-19 pandemic during the months of the study, participants completed the experiment remotely rather than in a physical lab space. We facilitated this task virtually using study materials, which included a laptop, charger, headphones, instruction packet, compensation form, and snacks. Prior to the start of each trial, we delivered study materials inside disinfected plastic tubs to participants. The instruction packets contained detailed instructions to join a video conferencing meeting with a research coordinator and teammate, if applicable. The packets also described the tutorial and main task.

We preloaded laptops with software and files for tool development and data collection. All laptops were running Fedora, an operating system that is a Linux distribution[3]. To ensure that participants focused on the translation task rather than environment setup or styling, we provided a Nodejs[4] project with starter code for the legal application on the study laptops. We opted to use a web application to mirror many online interactive legal tools. The project provided stubs for three web pages: 1) a start page with a description of the application, 2) a survey page with questions, and 3) a result page with a message formulated after processing the answers (see Appendix A). The participants could edit three of the files in the project, which corresponded to the model–view–controller pattern.[5]

3.5 Tasks and Procedures

The participants first constructed a warm-up tool, which was designed to familiarize them with our project files. In this task, we instructed the participants to write an application to help students new to the region decide whether to wear a coat. Participants posed three or fewer questions to the user (e.g., "Is it raining?"), collected user responses (e.g., "Yes"), and then delivered advice based on the answers (e.g., "You should wear a coat."). After they declared their tool was finished, the research coordinator instructed them to proceed to the main task.

Our instruction packet prefaced the main task with a brief description of legal advice tools and consumer bankruptcy. It instructed participants to develop a program that asks users for information about their situation and predicts whether this section of the statute applies to their case. We provided the text of our statutory excerpt, 11 USC 523(a)(4)–(5), (8)(A)(ii), and asked participants to read this excerpt and construct an application that faithfully represents it. Participants were encouraged to consult internet resources. Teams were encouraged to work together.

When participants finished the task, we asked them to complete a brief survey. The survey asked participants about their willingness to install their tool in the legal system. We sought to determine participants' confidence in their tools, as well as their understanding of the transformation of legal processes that would occur if their tools were put in use. The first question asked about installing the tool as an advisor, a role typically filled by lawyers. The second question asked about installing the tool as a decider, a role typically filled by judges. Participants were provided space to elaborate on their answers.

We allotted participants up to five hours to complete all of the tasks. We selected this length based on piloting sessions of the task conducted prior to our study. During pilot sessions without any time limit, all groups finished well within five hours. We record the duration of each of our experiment trials in Appendix B. All teams finished the study tasks within four hours and thirty minutes. On average, teams completed both the warm-up and the study tasks in three hours and

[3]https://fedoraproject.org/

[4]https://nodejs.org/en

[5]This software engineering pattern is common in web design, as it allows for separation of concerns. The model manages data, the view visually represents data, and the controller accepts user input and uses it to update the data or view.

two minutes. The fact that all teams finished during the scheduled session suggests they did not require additional time to complete the study tasks.

The study was reviewed and deemed exempt by our Institutional Review Board.

3.6 Data Collection and Qualitative Analysis

We collected several data sources to document how participants interacted with one another, how they constructed the legal tool, and how they navigated issues of interpretation and implementation. We used the screen capture software OBS[6] to capture verbal communication between participants and their on-screen application use. To track the coding progress of the participants, we created a Python script that saved a copy of the program files every thirty seconds. To analyze the development process, we combined audio transcription, website visits, text communications, and diffs of the project files into a single master transcript for each trial. This compilation allowed us to trace design choices precisely, from communication into programming code.

We qualitatively analyzed our processual data, including laptop screen captures, audio, and snapshots of programming code, as well as the final tools submitted by each team. In analyzing processual data, we were informed by Timmermans and Tavory's approach to "abductive" analysis [63]. Abductive analysis facilitates the construction of theoretical insights that build on existing scholarly frameworks. Our analysis thus looked for patterns that are surprising given the theoretical expectations offered by scholarship on the translation of law into code.

The qualitative analysis involved the following steps. (1) The first lead author and a research assistant, holding graduate degrees in law and computer science, and the second lead author, holding graduate degrees in sociology and political science, individually conducted line-by-line open coding for one to five trials in NVivo. During this process, these team members wrote memos and compared work to identify repeating codes relevant to the broader literature on algorithmic and legal reasoning. (2) Team members met to discuss and refine codes. Once we reached a consensus, the team created a single master code book for focused coding. (3) The lead two authors independently applied the master codebook to each trial. These authors continued to develop the analytical memos alongside this final coding pass. (4) All team members collaborated to combine memos and fully coded trials into study findings.

To analyze the final tools submitted by our participants, we systematically documented every question, response, and logic path through each tool and created a set of test cases. We performed analysis on both the source code and the exhibited tool behavior. To determine how participants encoded determinate law, we constructed test suites for each tool that examined the treatment of borrowers with multiple sources of debt. This strategy identified tools that issued incorrect legal advice. We compiled the tool questions that corresponded to the ambiguous legal phrase "undue hardship" to identify patterns in how participants encoded indeterminate law.

3.7 Limitations

Although we designed our study to match key characteristics of legal software development, there are limits to the generalizability of our study, particularly regarding commercial development contexts. Legal software is developed under a wide range of conditions, varying in timescales, resources, team composition, professionalization, and size.

Our study aligns most closely with documented development practices from nonprofits and legal hackathons. Nonprofits often have geographically dispersed teams whose members have little or no experience in software development [36]; in our study, participants collaborated remotely and had limited professional experience. Nonprofit projects may be less rigorous, and their budgets

[6] https://obsproject.com/

typically cannot support full systems development cycles [36]. As we did not enforce software engineering standards, our study also reflects the informal design environment of nonprofits. At legal hackathons, small, one-shot teams develop tools within a few hours [14, 60]. Organizers prompt participants to tackle complex issues such as resolving inheritance disputes [58]. The tools produced at legal hackathons can be publicly released [34] and receive media attention [38]. These development conditions correspond to study conditions in terms of the compressed timeline and team members' inexperience with legal software.

However, our study may match less well with professionalized commercial contexts. There is little information about the organizational and institutional conditions under which private firms produce legal algorithms. Certain products, such as the tax preparation software Turbotax, are produced by companies which have accumulated the resources and expertise to refine development processes over decades of software release life cycles. Our trials are likely more compressed than their tool-design cycles, and our participants have not built up experience with prior computational law projects as their employees might. Thus, we caution against generalizing our results to development contexts with large budgets, long timelines, project managers, and standardized software engineering practices.

Further, the legal context of our study is not fully representative of all legal practice. Our study focuses on legal practice in the U.S., which emphasizes prior decisions in legal cases, and so our analysis is less informative for countries that use the civil law system [16].

4 RQ1. How do computer scientists and lawyers approach translating law into code?

Participants engaged in two distinct disciplinary modes of reasoning. In line with prior work [26], computer scientists adhered to an algorithmic formalist logic. The legal experts, on the other hand, exhibited a more contextualist approach to the task, drawing on their knowledge of jurisprudence and the social setting of legal counseling. These modes of reasoning manifested across six dimensions, summarized in Table 3. Internal to each discipline, participants initially approached the task through similar modes of reasoning.

Table 3. Summary of observed differences between the two disciplinary modes of reasoning.

	Computer Science Mode of Reasoning	**Legal Mode of Reasoning**
Assessment of Task Difficulty	Simple	Complex
Approach to Legal Statute	Textualist	Legal Practice
Characterization of People who Interact with Legal Software	Users	Clients
Perspective on Software	Inspectable and Understandable	Black Box that Transforms Formal Specifications into a User Interface
View of Translation from Law to Code	Direct Representation of Law	Simplified Representation of Law
Framing of Distortions	Software Bugs	Deviations from Existing Legal Practice

4.1 Reasoning about the Task

4.1.1 Computer Science: Simple. Computer scientists believed that building the tools was a straightforward task. One computer scientist participant expressed surprise about the perceived ease of the task, stating, "Maybe I'm missing something, but this seems very simple" (CS7, T4 CS).[7] Another

[7]We include anonymized identifiers that correspond to the participant number, the trial number, and the trial type. This quote is from participant 7, a computer scientist, who completed trial 4 in the solo computer scientist condition.

stated the time provided was excessive: "Five hours feels a bit like overkill. Maybe shorten it for later iterations?" (CS22, T13 CS+CS). Though unfamiliar with the domain, computer scientists spent much less time conducting online research about the law (median of 6.92 minutes, compared to legal participants' median of 30.67 minutes).

4.1.2 Law: Complex. Unlike the computer science participants, the legal experts thought the task was complex. One legal participant described the task as "tricky" (L15, T8 CS+L). While the computer scientists often finished quickly, legal experts felt they might not have enough time to complete the task. For instance, one legal participant drafted multiple contingency plans for running over time to ensure the project reached completion.

4.2 Reasoning about the Law

4.2.1 Computer Science: Textualist. Computer scientists privileged the text of the law and treated its meaning as evident from the words alone. Following a textualist approach, computer scientists believed they could derive the statute's meaning by reading the statutory excerpt or consulting definitions for words and phrases in its text. Most computer scientists performed Google searches for statutory terms, such as "domestic support obligation" and "dependents." Through these searches, they found a wide variety of sources on debt discharge, such as private law firm blogs and nonprofit help pages. Computer scientists often copied and pasted text from these non-authoritative resources directly into their code. Their emphasis on the text even overrode commonsense objections. For instance, several teams believed (incorrectly) that fraudsters could take out student loans in order to avoid responsibility for their debts:

> CS8: You are thinking if somebody engages in fraud, the only thing they have to do is to get a student loan and they don't have to pay back any debts?
> CS9: Yeah, which doesn't seem right. But that seems like what it's saying. (T5 CS+CS)

As computer scientists thought that the text supplied the law's meaning, they did not investigate the legal context of the statutory excerpt, such as the broader operation of bankruptcy. When formulating search queries, they did not incorporate the scope of the statutory provision (e.g., "dependents" rather than "dependents 11 USC 523"). This practice led participants to definitions and rules from legal domains outside of bankruptcy. One participant who searched "dependents" encountered tax law resources and did not seek further information, not realizing that bankruptcy law defines the term differently. In no instance was the relative credibility or relevance of any one site over another discussed. Even where teams found judicial standards from court cases through online research, they did not discuss these sources with their partners or incorporate them into their tools. Rather, they believed that it was essential to hew to the text alone.

4.2.2 Law: Legal Practice. Legal experts, conversely, used standard legal research methods to investigate the statute's meaning. Such methods are a critical component of legal disciplinary training, covered in mandatory first-year law courses [56]. Five of six legal participants conducted legal research on the statutory citation using specialized legal research tools (i.e., Westlaw or Lexis). Participants executed structured searches that produced legislative history, practice guides, and case law. These documents all shed light on the background and interpretation of laws: legislative history provides context about lawmakers' intent, practice guides supply step-by-step information about routine legal procedures, and case law features explicit judicial reasoning granting or denying individual claims for relief.

Further, legal experts not only read these sources but also evaluated them. For example, judicial decisions that higher-level courts have contradicted are not considered good law. Legal experts therefore used citation tools to verify that the court cases they consulted had not been later

overruled. Their legal research revealed how legal institutions handle the statutory excerpt in practice.

4.3 Reasoning about the People who Interact with their Tools

4.3.1 Computer Science: Users. The computer scientists understood their own level of legal comprehension to be representative of the general population, so they used themselves as barometers for accessibility and usability. Many assumed that the limited legal knowledge of users required them "to simplify [the statutory excerpt] to make it more accessible" (CS3, T1 CS), with seven of nine computer science teams removing or explaining legal jargon. The computer scientists selected which terms to unpack based on their own familiarity with the concepts. For instance, one computer science pair was unfamiliar with some of the terms in the fraud subsection, so they supposed that "no one's really gonna know what this means" (CS9, T5 CS+CS). They wrote a series of nine questions to help the user determine if they had previously committed fraud, querying the user about their financial responsibilities and potentially wrongful actions.

4.3.2 Law: Clients. Legal experts framed the people interacting with their tool as clients with a personal legal history and a need for actionable legal advice. In contrast to the computer scientists disassembling unfamiliar legal concepts into a heap of simpler components, legal experts reasoned such thorough treatment was unnecessary for concepts that implied previous contact with the legal system. As one legal expert stated, "if someone has a court-ordered restitution payment, they're gonna know, at least in my experience" (L15, T8 CS+L). Legal experts also acknowledged that the statutory text may not be intelligible for those exploring their bankruptcy options for the first time. However, most legal participants modified statutory language unique to the context of bankruptcy in their tools to improve client comprehension, rather than unpacking terms unfamiliar to themselves as computer scientists did. Further, several legal experts contextualized the advice dispensed by their tools by suggesting concrete steps a client could take to pursue their bankruptcy claim. While tools created by computer scientists only dispensed a prediction about eligibility (e.g., "You may be eligible for discharge of debt" [T1 CS]), legal experts also included more actionable advice, such as, "Please see your bank and lawyer to discuss your specific situation" (T8 CS+L).

4.4 Reasoning about Software

4.4.1 Computer Science: Inspectable and Malleable. Computer scientists understood program code as a comprehensible set of algorithmic instructions they could manipulate using their disciplinary skill set. They approached software development as an activity for crafting their tool through contact with the source code, rendered web application, development tools, and online programming communities. Since we provided starter code files, the computer scientists worked first to understand how these files produced the web application. The computer scientists interacted with the web application in the browser and then attempted to match user interface elements with the corresponding program code in the source files. Once they started to add their own program code, they would periodically check how the browser rendered their additions, exploring a set of troubleshooting approaches when application behavior did not conform to expectations. To detect and correct errors, they made changes in the source code, located programming tutorials and solutions online, and consulted specialized technical tools such as the inspector tool built into the browser.

4.4.2 Law: A Black Box that Transforms Formal Specifications into a User Interface. Legal participants viewed software as a black box that transformed formal logic into a user interface. Formal logic for legal experts meant breaking down the statutory text into individual concepts, defining relationships, and applying clear rules based on assigned truth values to achieve a definite categorical sorting.

One legal expert directed software behavior by sending a chat message specifying a conditional relationship: "If Domestic support, THEN cannot use Ch 11" (L6, T3 CS+L). The legal experts could also reason about software at a high level, as a user interface. They suggested using specific user interface components, such as a "pop-up" (L15, T8 CS+L). Thus, legal experts reasoned about software development from below, as formal logic, and from above, as a user interface.

The software connecting the formal logic to the user interface was opaque to legal experts. Legal participants typically did not build a deep understanding of the program files or software development practices. Although all legal experts lacked programming knowledge, only one attempted to learn more about programming through online research, and this effort was quickly abandoned. While several computer scientists tried to clarify their understanding of the law by directing questions to their partners, only one legal participant asked their computer science partner about code. Legal participants made statements indicating they did not understand the capabilities of computer code. One legal expert proposed that their partner abandon a design with twelve different responses in favor of an approach requiring "less typing" (L6, T3 CS+L). Their computer science partner explained that the responses could be composed quickly with basic programming concepts, such as conditional statements and loops, rather than being written out individually. Similarly, many legal participants appeared confused about the relationship between source code and web application. While one computer scientist demonstrated their tool in the browser, their law partner asked them to display "what the website would look like [...] without the code" (T8 CS+L). This sort of mistake indicates that legal experts were unsure how software was bounded.

4.5 Reasoning about the Translation of Law into Code

4.5.1 Computer Science: Software as a Direct Representation of Law. As they translated law into code, computer scientists did not understand their activity as a transformation. Instead, they felt that their final software products fully encompassed the meaning of the law, provided that there were no software bugs. Two overlapping views supported this conclusion. First, some computer scientists viewed the law as already structured similarly to computer code: "this US code [...] has a bunch of, like, if-statements built into it" (C9, T5 CS+CS). Second, many believed that repeating the statutory text prevented distortion of legal meaning, preventing the user from "missing out on information" or becoming "biased" (CS26, T15 CS+CS). Both approaches led the computer scientists to conclude there was an identity between the law and the software they wrote to encode it.

4.5.2 Law: Software as a Simplified Representation of Law. Legal participants framed the translation of law into code as a transformation that flattened legal practice to fit the demands of computation. Where computer scientists understood the process of developing software as a geometric translation, sliding the text of the law into the region of code without modifying its shape, the legal experts understood this process like 3D projection, where the textured, multidimensional law is mapped onto a flat plane of code. The legal participants felt that developing legal software meant representing only a subset of interpretations from those they considered computationally representable. Because the compressed logic misses "a lot of nuances to cases" (L11, T6 CS+L), legal participants expressed that their tools failed to capture the full meaning of the law.

4.6 Reasoning about Distortions

4.6.1 Computer Science: Distortions as Software Bugs. When computer scientists acknowledged their software could distort the law, they framed such deviations as software bugs. After completing the study, computer scientists believed their tools were ready to be released publicly, "as long as some testing [was] done to make sure that there is no bug in the system" (CS25, T12 CS+CS). In assuming

the remedy to bugs was software testing, computer scientists limited their inquiry to detecting code-level flaws and did not revisit the broader assumptions of the constructed applications.

4.6.2 Law: Distortions as Deviating from Legal Practice. Law participants understood distortions as a failure to capture the nuances of existing legal practice. As legal participants regarded the people using their tool as clients, they worried about providing insufficient legal guidance compared to attorney consultations. As one legal expert asserted, "our program does not process enough information to allow a user to make an informed decision" (L13, T7 CS+L). Following their understanding of legal software development as flattening legal practice, they believed their tools might be unable to assess more complex legal issues that "require greater finesse" (L21, T12 CS+L). To remedy these deviations, legal participants suggested reintroducing human decision-making:

> L18: I think there needs to be a human check of the facts at hand and given the niche nature—and realities of many of the caveats in the US Bankruptcy code section we did—it does not seem fitting to be automated. (T10 CS+L)

5 RQ2. What processes and strategies do teams use to complete the task?

The composition of the teams structured development processes and coordination strategies employed by participants. Computer scientists working alone proceeded quickly from legal research to implementation (see Appendix B for task duration across trials). Computer scientist pairs shared responsibility for each stage of development, but adding a partner with the same expertise provided limited benefits. Interdisciplinary teams divided labor along disciplinary lines, often requiring the teams to contend with boundary objects to integrate their separate efforts.

Some aspects of tool development were alike across experimental conditions. All teams sought at a base level to understand the law and implement a workable legal tool. Across nearly every trial, participants turned to external interpretive resources through internet research. Participants across all trials adapted the statutory language into a series of questions, and allowed input through radio buttons, checkboxes, or dropdown lists. No team considered parsing text input or otherwise permitting unstructured responses during the design process.

5.1 Computer Science Solo Teams

Computer scientists who worked alone exhibited similar behavior across trials. They issued a few Google searches, then started to write code. All individuals evaluated their software tools by testing a series of inputs. The primary stage during which these participants expressed difficulty, such as by using obscenities, was when writing code. These participants finished the study much quicker than other team compositions (on average, solo teams spent one hour and 28 minutes on the main task, while interdisciplinary teams spent two hours and 44 minutes; Appendix B). As this was not a think-aloud study, individual computer scientists did not explain or justify their design decisions.

5.2 Computer Science Pair Teams

Computer scientist pairs had overlapping expertise, so their specialization did not drive a division of labor. Teams shared responsibilities at each stage, with each member participating in legal interpretation, coding, and evaluation. Since partners entered the study with similar claims to expertise, other social factors shaped power dynamics within teams. For instance, male participants spoke 60.8% of the total words, while their female partners spoke 39.2% across the four computer scientist pair teams (though not an intentional aspect of our study design, all computer scientist pairs were mixed gender). Despite such imbalances, it appeared that the addition of a teammate provided some benefits over a computer scientist working alone.

Some approaches to coordination were more effective at building legal understanding, particularly when the partners advanced competing interpretations. Participants chose a paradigm for coordination at the outset, either decomposing each stage into subtasks or completing the full task together. In the first approach, teams explicitly divided a stage into parts, separately completed their assigned work, and then integrated their efforts. The teams that worked separately on the legal research task did not clash over the meanings of terms. Since the members built up disjoint knowledge, each accepted whatever their partner reported back. In contrast, the teams that completed the task together were engaged in deliberation, so either partner could challenge the other's actions or beliefs. In trial 15, one teammate indicated that "debtor" referred to a person who is owed money. Their partner presented a conflicting definition that they had encountered in their online research. This deliberation led both partners to express a correct understanding of the word, which is that a debtor is a person who owes money to someone else. Yet, as computer science participants had limited understanding of the law, they could not entirely dispel confusion about legal terms.

5.3 Interdisciplinary Teams

As the interdisciplinary teams had distinct areas of expertise, all of these teams set up a stark division of labor. Legal participants were responsible for interpreting the statute, and computer scientists were responsible for writing program code. This division was set up and reinforced by both partners, who were reluctant to engage in tasks that fell within their partner's realm of expertise. For instance, when one computer scientist asked for input about maintaining their codebase, their legal expert partner countered, "Uh, I'm doing no code. (laughs) I'm just here for the legal help" (L13, T7 CS+L). However, the partners could only keep their work separate for so long. Eventually, they had to condense the legal information collected by the legal expert into a tool design for the computer scientist to implement.

When computer scientists and legal experts discussed the design of their tools, they struggled to communicate due to their divergent modes of reasoning. Legal experts experienced trouble communicating law and tool designs to computer scientists, and computer scientists found it challenging to explain software to legal experts. To address these issues, five of six interdisciplinary teams converged on a common approach: constructing physical schematics to communicate the intended design and logic for the legal advice tool.

We term these "intermediate representations," as these boundary object artifacts occupy a transitory space between the law and the final software tool. Legal participants frequently turned to this strategy after encountering difficulties transmitting their interpretations to their partners. They would initially give instructions about constructing tools verbally, but computer scientists often repeated them back incorrectly.

> L21: If question one [is answered] "Yes," the section applies and we're done.
> CS20: Question one [answered] "Yes," it goes [to] two?
> L21: It doesn't go anywhere, we're done if they say "Yes." (T12 CS+L)

After failure to communicate intended meaning through discussion, legal participants typically pivoted to a more concrete record of their intended designs.

> L21: I just shared in the chat a link to a flowchart I'm using, and these will have some of the texts that you can use in your program. (T12 CS+L)

This concrete record was then used as a reference by both partners, which was useful for reducing misquotations:

> CS20: What is the response for the first question?
> L21: So I have it typed out in this flowchart.

> CS20 (reading from flowchart): "Unfortunately this debt cannot be discharged." Ok. (T12 CS+L)

The legal experts directed the contents of the intermediate representations in all interdisciplinary trials. Computer scientists willingly followed the lead of their legal partners, as they felt this design aspect of the task involved legal skills. As one computer scientist stated, "I don't really have any law knowledge, so I don't really know how to ask the questions" (CS14, T8 CS+L).

The most popular format of intermediate representation was a text-based outline of questions interleaved with branching path instructions. For example, after a draft of question text, there might be specific instructions about how the tool should respond based on possible user answers (see Figure 1). Other text-based approaches included writing formal logic and pseudocode. One team used a diagram-based intermediate representation, with the legal expert constructing a flowchart using a web-based application (Figure 2). Each question was a separate node, answers served as edges, and all paths terminated in specific legal advice.

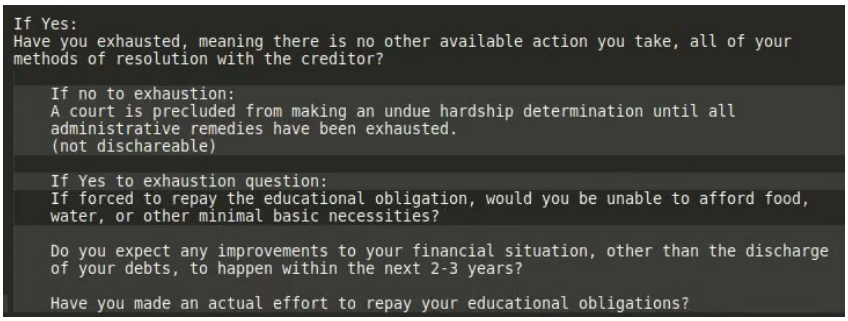


Fig. 1. Portion of the text-based schematic constructed by legal expert and modified by computer scientist. (T8 CS+L)

Participants updated intermediate representations as they revised their understandings through teamwork and legal research. Responsibility for maintaining the intermediate representation could shift during the trial, so an initial sketch provided by the legal expert could be copied and reformatted by the computer scientist. Some teams even scrapped their initial representations to pursue a different type of structure. Thus, in addition to providing an accessible record of design instructions, intermediate representations were flexible and revisable. Despite these advantages, intermediate representations were also ambiguous, incoherent, or ignored.

Resulting breakdowns, which we explore in the following section, often went unnoticed by the teams. Every team elected to test the final artifact that they built, but legal experts were reluctant to participate. On interdisciplinary teams, software evaluation was conducted and primarily directed by the computer scientists. Though legal participants directed a large part of tool design, they took a minor role in assessing the tool's conformance to their proposals. In half of the interdisciplinary trials, the legal participant did not take any part in the testing process. Testing software was often bound up in more technical tasks, such as bug fixing. This may have led legal experts to excuse themselves from this phase of development, unsure of how to contribute:

> CS12: Okay. I'm just like trying to check over some software and debug, basically.
> L13: Yeah. I mean, it doesn't seem like you need my help for that. (laughs) (T7 CS+L)

Legal experts uncritically accepted claims from their partners that the program code worked well. For example, one team found an error while testing, but the computer scientists then re-ran

the program with slightly different inputs. The new test case did not produce the same error, so the computer scientist proclaimed that the bug was gone:

> CS20: So, you saw, right? It worked. You saw, right? It worked completely fine. (T12 CS+L)

Accepting their teammate's assurances, the legal expert agreed to submit the tool without modification. When computer scientists misrepresented the technical challenges they were facing, they led their legal partners to believe that tools were more reliable than they in fact were.

6 RQ3. How do teams encode the law in software?

We follow the choices observed during the development process, tracking how the participants' modes of reasoning embed disciplinary patterns in legal software. All teams completed a legal tool. All teams elected to structure their tool to show a series of questions posed to a user, and then processed the user's answers to select a response from multiple possible recommendations. We organized our analysis of the software artifacts around the treatment of a determinate and an indeterminate feature of the statutory excerpt from 11 USC 523(a).

6.1 Representing Determinate Features of the Law

The legal provision that we presented to participants addresses three different sources of debt: fraud, domestic support obligations, and student loans. An individual debtor might have any combination of these debts: some might have to pay restitution for fraud, and others might owe both child support and student loans. To correctly consider the application of these three subsections to a debtor's situation, a tool should independently evaluate each of the three sources of debt and provide advice about every relevant type. We classify the final artifacts into two categories: those with correct structure and those with incorrect structure that gave erroneous or incomplete advice.

Though determinate law is thought to be computable [61], only two out of fifteen tools correctly and completely advised users about the three types of debts. The other thirteen, which spanned all team compositions, exhibited structural error. Though computer science teams arrived at these errors along different routes from interdisciplinary teams, no specific errors were unique to a particular team type. Thus, the inclusion of a legal expert did not lead to lower rates of error.

Table 4. Treatment of legal determinacy by team type.

	Interdisciplinary Teams	**Computer Science Teams**
Correct Structure: Provide Advice on Each of Three Subsections.	1 (17%)	1 (11%)
Incorrect Structure: Provide Erroneous or Incomplete Advice.	5 (83%)	8 (89%)

6.1.1 Computer Science Teams. The majority of computer science teams thought it was not necessary to understand the law to encode it into their tools. Thus, they misinterpreted the law and built their misinterpretations into the software. Eight of the nine final artifacts produced by computer science teams exhibited an incorrect structure. As they understood the given task as simple and took a strict textualist approach, computer scientists did not strive to comprehend the legal excerpt or its statutory context:

> CS25: I guess since we're just kind of just asking questions, we don't really need to understand the reasoning behind [the statute]. (T14 CS+CS)

This shallow engagement with the law meant that most computer scientists misunderstood its meaning. Computer scientists uttered factually untrue statements on many topics, from the meaning of statutory phrases to the scoping of subsections to the broader procedural operation of bankruptcy. Their tools followed these misperceptions. Multiple teams mistakenly believed that this provision provided a laundering scheme for debt incurred through fraud, where such debts would be forgiven if the debtor enrolled in school. The legal advice tools created by these teams encoded this incorrect dependency between the subsections. Others interpreted the provision to mean that anyone who had ever paid child support was ineligible to file for bankruptcy. Thus, when a user indicated that they owed child support, their tools would not solicit information about the user's other debts. This incomplete structure meant that the user would only receive information about a single type of debt, potentially suppressing advice about student loan relief.

Only one computer science team crafted a correct tool. This team arrived at the proper structural understanding by advancing competing interpretations that were encountered through legal research. CS22 initially proposed that the excerpt's subsections were dependent, but their partner, CS23, decided to investigate the citation online. CS23 found a discussion about discharge from the nonprofit National Consumer Law Center, after which they corrected CS22's understanding of the law. Through discussion, CS23 convinced CS22 that fraud debts were not related to domestic support debts and that each different type of debt had to be considered separately. The team went on to encode a tool that correctly considers the three types of debt independently.

6.1.2 Interdisciplinary Teams. The legal experts on interdisciplinary teams believed it was critical to comprehend the statutory excerpt in order to produce a correct tool. However, the intermediate representations that they prepared did not transmit their understanding perfectly to their computer science partners. These breakdowns led computer scientists to misunderstand the law, as in computer science teams, and then improperly encode the structural relationships of the statutory provisions. Five of six interdisciplinary teams built tools with an incorrect structure that issued misleading advice about the types of debts.

Many intermediate representations that legal experts produced were vague, so the correct tool behavior was not precisely defined. Intermediate representations in the form of text-based outlines confused computer scientists: the path descriptions did not explicitly identify the nodes that were termination points, so the tool flow was unclear. In Figure 1, for instance, when the third answer is "Yes," the outline does not indicate whether the application should proceed to the subsequent question or terminate. In several other trials, the legal expert never provided instructions for dealing with multiple types of debts, leaving their partner to guess at the structural relationship.

While legal participants did not provide incorrect information when directly discussing the law, their intermediate representations did not always fully specify how the law operated for the complete universe of potential applicants. In some cases, the legal experts made tool designs that did not provide advice to people with multiple types of debts. For instance, the diagram-based intermediate representation terminates the path if a person reports having fraud debt without asking questions about domestic support obligations or student loans (Appendix C). As this participant did not justify their design in conversation, we do not know whether this was due to a defect in legal understanding or the challenges in translating their knowledge into such specifications. However, several other law participants explicitly articulated a correct understanding of the law, suggesting that such problems were attributable to translation.

Though the intermediate representations were not perfectly drawn, the computer scientists regarded them as the controlling source for design specification. In other words, intermediate

representations could set the boundaries of communication. For instance, the computer science participants resisted the uptake of verbal modifications, implementing older instructions that were fixed in text instead. Additionally, computer scientists could appeal to intermediate representations to defeat critiques:

> L18: When you hit repayment of educational benefits, it jumps to question two, not the repayment question.
> CS17: Yeah. Uh, but I don't really know the logic inside of this. So if there's a problem, the problem's with the questions. (gestures with cursor to outline-based intermediate representation) (T10 CS+L)

This team did not make any further modifications to their tool, because the legal participant could not articulate their desired change in terms of the intermediate representation.

In the only interdisciplinary team that produced a tool with a correct structure, the legal expert and computer scientist worked to ensure that both members mutually understood the structure of computer code and the structure of the law. While most legal experts only explained the law briefly at the outset of the task, this team returned to the law and discussed its meaning at several points during the trial. As this legal expert conducted research, they often read aloud passages from

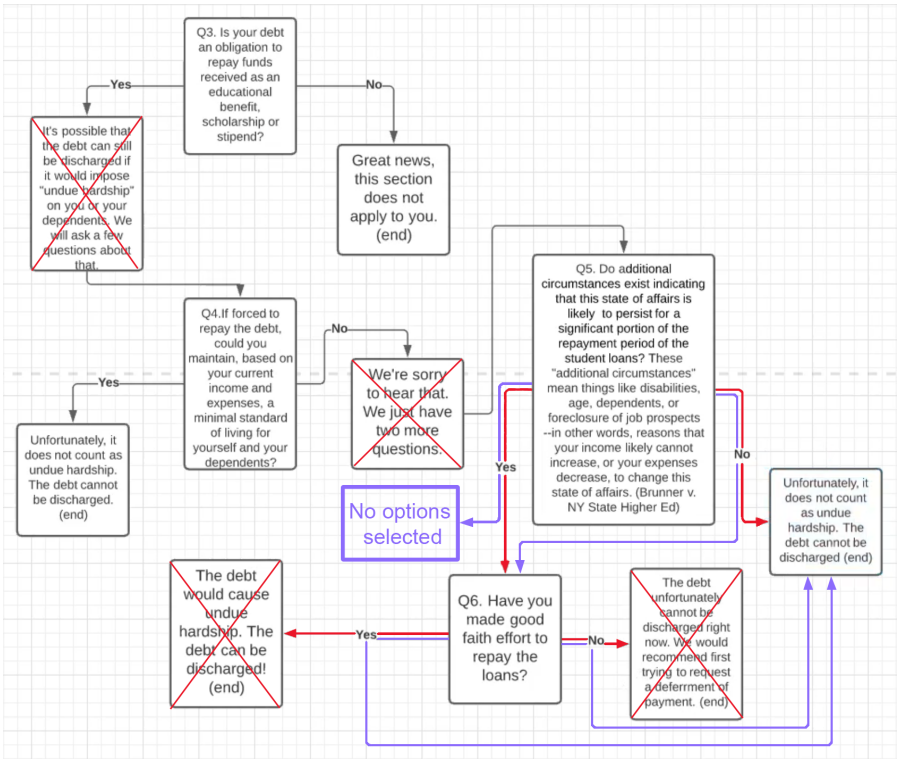


Fig. 2. Excerpt of flowchart constructed by legal expert, overlaid with colored annotations. Red represents messages and edges removed by computer scientist in final tool; purple represents messages and edges added by computer scientist. These annotations demonstrate how final tools could deviate from a legal expert's instructions, indicating that the translation between intermediate representations and software code is a potential site of error. For full flowchart, see Appendix C. (T12 CS+L)

judicial decisions that explained how the law had been applied to especially vulnerable debtors (e.g., an unemployed debtor who suffered from diabetic neuropathy was deemed capable of working sedentary jobs and thus ineligible for student loan forgiveness [65]). Their partner responded with interest and empathy, often asking clarifying questions that led to refinements on their tool design. Unlike other interdisciplinary trials, this legal participant made a sustained effort to understand the code their partner was writing, then drew upon their improved mental model of programming to inform revisions to their intermediate representations. This team worked together closely and repeatedly assessed each other's understanding of both the legal interpretation and programming subtasks. For instance, the legal expert initiated a conversation about the different files that their partner was editing:

> L6: Oh, okay. So HTML is the appearance stuff [...] saying this is what it looks like [...] Is that how that works? (T3 CS+L)

The computer scientist then presented each file and explained how they functioned together to compose a web application. From that point, the communications between the partners began to shift so that they discussed programming problems using programming terminology. The final intermediate representation that the legal expert transmitted took the form of pseudocode that used conditional branching if/else if statements. Their final artifact followed these instructions, composing the returned message by concatenating advice about each type of debt to cover all possible combinations of debts that might be implicated by the statutory excerpt. Their approach helped to align their disparate modes of reasoning.

6.2 Representing Indeterminate Features of the Law

When would repaying student loans impose undue hardship on a debtor? In many situations, this determination is not clear-cut. Yet, the judiciary has developed standards to guide their analysis of cases, which courts in most circuits are bound to apply under the doctrine of precedent. Here, we examine indeterminate features of the law by tracking how the teams wrote questions to determine eligibility for educational debt relief. Computer science teams shallowly engaged with ambiguity, typically repeating the ambiguous phrase verbatim. Interdisciplinary teams had a deeper appreciation of ambiguity in law, with most appealing to the judicial standards to make sense of the phrase "undue hardship." Disciplinary modes of reasoning thus shaped how ambiguity was encoded in legal advice tools.

Table 5. Treatment of legal indeterminacy by team type.

	Computer Science Teams	**Interdisciplinary Teams**
Resolved to Detriment of Debtor: All Educational Debt Nondischargeable	1 (11%)	0 (0%)
Ambiguity Unresolved	6 (67%)	0 (0%)
Resolved with Unrelated Legal Definition	2 (22%)	0 (0%)
Resolved by Using Strict Interpretations to Make Judicial Standards More Determinate	0 (0%)	5 (83%)
Resolved by External Attorney	0 (0%)	1 (17%)

6.2.1 Computer Science Teams. No computer science team recognized "undue hardship" as legally ambiguous. The theorized difference between indeterminate and determinate law did not impact how computer science experts translated the statute in practice. In line with their textualist approach,

computer scientists treated this term just as other words in the statute. In the teams that simply rephrased the text of the fraud and domestic support provisions as questions, they also rephrased the student loan provision as a question. In the teams that added examples to the fraud and domestic support provisions, they also added examples for the student loan provision. No teams incorporated existing judicial standards, departing from current legal practice. We observe three treatments of indeterminate law by computer science teams: deleting the ambiguous phrase, leaving it unresolved, or resolving it through appeal to an unrelated legal provision.

One tool produced by a single computer scientist construed the ambiguous phrase to have an unambiguous meaning—the most disadvantageous possible meaning for debtors. This tool contains no reference to undue hardship. The computer scientist only performed a single search query and never consulted a resource beyond the text of the statute itself. Their tool asks the user whether they have educational debts but does not ask whether repaying such debts would impose undue hardship. Instead, the tool considers all student debt to be nondischargeable. This approach to ambiguity does not accord with the statutory language or any judicial interpretation of the law.

Six final artifacts, three from single computer scientists and three from computer scientist pairs, did not resolve the ambiguity. Instead, their tools leave users to interpret the meaning of "undue hardship" by repeating snippets of the statutory text verbatim. Few tools provided additional guidance, which, when it did appear, consisted of hyperlinks to long articles or large databases, so the onus is still on the user to click the link and make sense of the resource. This approach emphasized fidelity to the legal text, even where participants discovered the judicial reasoning. In three computer science trials, participants who conducted online research on the statutory text encountered the *Brunner* standard. The sources they consulted described how courts applied the standards, and some participants highlighted these sections with their cursor. Yet, these teams did not discuss the meaning of the standard or attempt to incorporate it into their tools. All three teams instead built tools that directly asked the user whether they believed that "undue hardship" applied to them, discarding all extratextual interpretations.

As a final approach, two teams incorporated extratextual interpretations but broke from existing legal practice by drawing definitions from an irrelevant legal source. These teams did not recognize that statutory terms could have inconsistent definitions across the U.S. Code. For instance, the term "child" is defined several dozen times across the U.S. Code, as the word can take on different meanings in different contexts [50], such as a kinship relation in inheritance but an age threshold for alcohol purchases. When the teams searched "undue hardship," the results also included a definition from an unrelated legal domain—disability law. Though the sources they consulted referred to "employers" rather than "debtors," the teams ignored the inconsistency in context. They reproduced language from the Americans with Disabilities Act (ADA) in their final artifacts. By drawing a definition from an inappropriate source, the teams stripped away the bankruptcy context and ignored the explicit boundaries that circumscribe legal definitions.

6.2.2 Interdisciplinary Teams. All interdisciplinary teams identified "undue hardship" as legally ambiguous but prioritized approaches and interpretations that they thought could be implemented well in code. Legal experts believed computers were best at applying clear rules that produced a definite sorting of people into categories. Many of them doubted that computers could fully encode the complexity of law and legal practice. Thus, legal experts either worked to make the existing judicial standards more determinate or directed the user to consult with an external legal professional.

The most common strategy was to adopt strict interpretations and alter language to be more stringent in order to produce cleaner categorical boundaries. These changes produced a more concrete standard but also made the tools less generous, putting more limits on the discharge of

student loan debt than judges currently observe. For instance, though there are multiple different standards possible for assessing undue hardship, half of the interdisciplinary teams included only the *Brunner* standard in their tools. They explained that this standard was more restrictive than the alternatives:

> L13: My impression is that a lot of courts use the *Brunner* [standard] for the exact reason we don't like the totality-of-circumstances [standard]—[the *Brunner* standard] is much more specific [...] courts have started being over time really strict with it. (T7 CS+L)

They also rephrased language in the *Brunner* standard that they believed was not computable, narrowing vague terms since "there's no way our users here are gonna be able to understand what that means, and we won't be able to code it" (L13, T7 CS+L). As one modification, many legal participants replaced "minimal standard of living" with a reference to the "federal poverty line." The Department of Health and Human Services poverty guidelines set clear, determinate income thresholds that define whether a household is in poverty (e.g., $30,000 for a household size of four). This substitution is more restrictive than existing case law, though, as many courts have discharged loans of debtors who did not live in poverty (e.g., [33]). Another two teams combined the totality-of-circumstances and *Brunner* standards. Though such a synthesis has no precedent in the legal system, they also modified the language they borrowed from judicial standards to be more rigid and concrete. For instance, a rephrasing from one team asks users, "Would you be unable to afford food, water, or minimal basic necessities?" Yet, judges have stated that student loan discharge does not require "a lifestyle lacking in necessities"; courts permit debtors some comforts, such as the commonly cited allowance for "the ability to pay for some small diversion or source of recreation, even if it is just watching television or keeping a pet" [67].

One interdisciplinary team refused to construe ambiguity within their tool, directing student debtors to consult an external lawyer rather than making a prediction about eligibility. The legal expert contended that no computational tool could faithfully represent this ambiguous phrase in the statutory excerpt. They refused to cede any interpretative work to software, stating: "I don't per se believe in the automation of legal work—especially if it relates to bankruptcy and debt" (L18, T10 CS+L).

7 RQ4. How do computer scientists and lawyers reflect on their work?

Immediately after the participants completed the main study task, they individually filled out a survey about potential roles for the legal tools they constructed.

Proponents of automated legal advice systems argue that such tools can guide people who cannot afford to consult a lawyer. To gauge whether participants were comfortable installing their tools as an advisor, our survey first asked, "Would you be willing to include the piece of software your team created on a website that provides legal advice to laypeople?"[8] Participants could select "Yes," "No," or write-in a response, and then briefly explain their answers in a text entry form. Most participants across all conditions assented to this use (16 computer scientists, or 84% of them, and 4 legal experts, or 67% of them, selected "Yes").

Though automated legal systems have not yet replaced human decision-makers in American courts, there are some international efforts to conduct trials by replacing judges with computational

[8]U.S. law students are not permitted to practice law without attorney supervision. As these tools could be seen as dispensing legal advice, law students might have refused to share their work on the grounds that it violated legal ethics. So this concern would not direct their answers in the follow-up survey, we displayed the following message: "For the purposes of this question, assume that any ethical considerations regarding the unauthorized practice of law (see Model Rules of Professional Conduct, Rule 5.5) will be addressed following the study."

models of the law [72]. To gauge whether participants were comfortable installing their tools as a decider, our survey provided the following scenario: "The local court system is currently experiencing high caseloads. Would you like to offer your tool for use in determining the outcome of court cases? It would be used by court clerks without legal training. They would record your software output as the official outcome. This would reduce the workload of judges." Participants could select "Yes," "No," or write-in a response, and then briefly explain their answers. While computer scientists seemed not to mark much of a difference between the two proposed uses, legal experts uniformly responded that their legal advice tools should not be used as deciders (15 computer scientists, or 79% of them, and 0 legal experts, or 0% of them, selected "Yes").

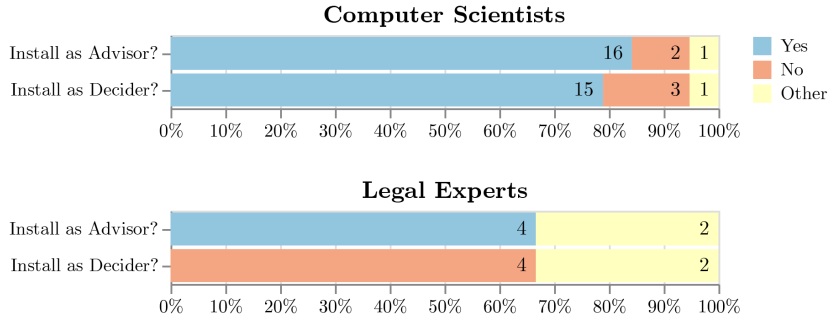


Fig. 3. Participant willingness to install their tools as advisors and deciders.

7.1 Computer Science Attitudes about Installation

Computer scientists across study conditions typically justified the deployment of their tools through the lens of helpfulness and did not differentiate between using the tool as an advisor or decider.

> CS10: yes it would be helpful[9] (T6 CS+L, agree to install as advisor, agree to install as decider)
>
> CS19: I would love to share my work and help anyone who is unsure whether they can discharge their debt. (T11 CS, agree to install as advisor, agree to install as decider)
>
> CS22: No issue, I hope this helps (T13 CS+CS, agree to install as advisor, agree to install as decider)

Occasionally, computer scientists acknowledged that their code could be flawed but believed that their tools would be ready to determine court outcomes after a round of verification by a legal expert. In keeping with their framing of distortions as software bugs, computer scientists assumed that the remedy would be testing, with only minor modifications at the level of code:

> CS7: There are only 10 or so possible outcomes; there are only 4 yes or no questions, with a bit of conditional rendering. As long as a lawyer or other certified individual checks that each one is correct, I would love to have the tool used. (T4 CS, agree to install as advisor, agree to install as decider)

[9]In Section 7, we reproduce direct quotations from written participant responses.

7.2 Legal Expert Attitudes about Installation

Legal experts marked a strong difference between using their tool as an advisor and as a decider, stating that the compression they performed to make law computable also constrained the capabilities of their tools. They were comfortable releasing their tools to the public, framing this use as a lightweight interaction in which their tools give a "quick and fair assessment" (L18, T10 CS+L, agree to install as advisor).

However, no legal experts agreed to using this tool to decide court cases, insisting that too much nuance was involved in making a determination. As one lawyer reflected:

> L13: The software only addresses cases that are unambiguous, and it was not coded to account for exceptions to even these clear-cut scenarios, let alone for ambiguous cases. There was nowhere near enough legal research conducted to be sure even as to [subsections] (4) [fraud] and (5) [domestic support] that all cases would be determined properly. I think the program at this point would be effective only as an informal guide to help laypeople navigate statutory interpretation, but should absolutely not be used to determine case outcomes. (T7 CS+L, disagree to install as decider)

7.3 Attitude Shifts and Collaboration

While the disciplinary training of the participants largely shaped their understanding of the translation of law into code, intensive interdisciplinary collaboration could expand their perspectives. Most interdisciplinary teams did not work together closely, and so each team member retained their distinct analytical lens. For instance, one legal expert spent less than a minute explaining the statutory language to their computer scientist partner. This computer scientist indicated a strong willingness to install their final artifact in the place of a judge:

> CS17: Yes I would always love to share the tool I created to reduce the judges work. Computer science is mean[t] to be reducing people's workload! (T10 CS+L, agree to install as advisor, agree to install as decider)

However, the two computer science participants who engaged in prolonged discussion about the law both aligned with their legal partners in refusing to use their tools to decide court cases. For instance, one legal expert spent over 20 minutes explaining the statutory language to their computer scientist partner. They offered in-depth explanations of each provision, provided hypothetical examples to clarify their partner's understanding, presented the judicial standards from case law, and discussed the merits of each judicial standard. The computer scientist did not consent to installing their tool in the local court system, independently reaching the same conclusion as their legal expert partner.

> CS12: While my software works, I am unsure about my understanding of the law. In official usage, I wouldn't feel comfortable until the software is tested. I am also aware that algorithms may be implicitly discriminatory. The legal system (presumably, I cannot cite exact statistics) disproportionately impacts Black and Muslim communities in [location redacted] specifically, and I'd prefer to have additional verification to make sure that my algorithm does not introduce additional bias. (T7 CS+L, disagree to install as decider)

The other computer scientist who refused to install their tool as a decider belonged to the one interdisciplinary team that had captured the correct structure of the statutory provisions. Their reluctance suggests that their collaboration led to a strong shared understanding of not only the tool's design but also its use in the legal system. Thus, the team that had most faithfully adhered to the law was among those hesitant to empower their tool with decision-making capabilities.

8 Discussion and Future Work

Our study examines the legal software development process to shed light on how flawed translations of the law are embedded into code. Although there is a long history of legal software producing errors and threats to legal legitimacy in high-stakes contexts, the field of computational law lacks methods for identifying how or why these issues emerge [36, 47]. Our findings demonstrate that the epistemic divide between computer science and law makes the translation of law into software fertile ground for errors, misunderstandings, and policy distortions. Furthermore, these diverging modes of reasoning make it difficult for development teams to recognize mistranslations in their tools, leading them to support the release of flawed software. These results highlight that we cannot evaluate legal software based only on the content of a law and the capabilities of code. Instead, evaluations of legal software must also consider the hands-on work of translating that law into code. The details of this translation work depend on the practitioners involved, their disciplinary logics, their development strategies, and their interactions with teammates.

8.1 Modes of Reasoning in the Production of Flawed Legal Software

The disparate modes of reasoning followed by computer scientists and lawyers led our participants to distort the law when developing software. Computer scientists relied on a textual approach to law, believing that they could represent the statute in code without understanding the legal domain. As a result, they discarded context-specific legal reasoning and built misunderstandings into their tools. Meanwhile, although scholars have pleaded for lawyers to uphold the law against the incursions of computer scientists [30], we found that legal experts also drove software to deviate from standard legal practice. As the legal mode of reasoning suggested that software requires restrictive formal specifications, legal participants jettisoned legal reasoning that did not produce crisp categories.

These modes of reasoning could explain how development practices in real-world settings lead to mistranslations. The behaviors of our participants follow closely from their disciplinary training in computer science and law. Computer scientists are taught to approach the people who interact with designs as "ordinary users" [43] and to adopt a user's point of view when defining problems [41]. In contrast, law students learn how to counsel and interview clients [4] and must pass an exam on the professional rules governing "client-lawyer relationships" before they can practice as attorneys [45]. Following their disciplines, computer scientists in our study conceived of people interacting with their tools as users and estimated lay user understandings based on their own intuitions, while legal participants related to them as clients and considered the duties owed to them.

Furthermore, many legal development contexts, including nonprofits and hackathons, enlist practitioners with limited software development experience. Truncated development cycles are typical in these contexts, giving practitioners minimal opportunities to accumulate expertise through cross-disciplinary collaboration. This lack of exposure impedes their ability to bridge the gap between the disciplines.

These tendencies suggest that the dynamics we observed in our experiment could be playing out in real-world development contexts. If that is the case, our results would help shed light on why errors and harmful outcomes persist across legal software systems. In turn, researchers and policymakers should view mistranslations as an endemic feature of legal software.

8.2 Augmenting Guidance about Translating Law into Code with Modes of Reasoning

Our findings challenge the efficacy of common recommendations for translating law into code. Some scholars assert that adding a lawyer to development teams will alleviate problems with legal software [5, 12, 36, 71]. Although we found that interdisciplinary teams were more conscious of legal complexities than computer science teams, including legal experts in development did not

prevent mistranslations. Legal training does not prepare lawyers to understand computer science and the requirements of formal language. It does not teach lawyers to produce formal models, test software, or appreciate programming capabilities. As a result, legal experts were unable to convey their legal knowledge in a clear and practical form to their computer science partners.

Although interdisciplinary team members tried to communicate their respective knowledge, their attempts at bridging the epistemic divide left out essential information. Because participants came from different disciplines, they organized their collaboration through abstractions, which took form in intermediate representations. The intermediate representations worked as boundary objects: they sought to reconcile legal and programming perspectives in a generalized form [59]. To create these intermediate representations, team members relied on their own assumptions about the information and format that their counterparts preferred. However, because their knowledge of the other person's field was limited, such assumptions hampered tool development. When computer scientists offered proof that the tool worked through demonstrations, they shielded their partners from having to understand programming code. However, this practice did not effectively surface the tool's internal logic, so legal experts could not confirm that the software was truly following the law. When legal experts compressed legal interpretation into a diagram, they excused computer scientists from having to understand the statute. However, this practice constrained the design space, so computer scientists could not consider how alternative computational approaches could have embedded legal practice. In other words, interdisciplinary collaborations may break down when boundary objects only build a narrow bridge between domains, obscuring the actual operation of law and software.

Our findings also challenge the advice that precise, determinate laws can be translated into code. This recommendation assumes that perfect translation between law and software is possible, provided that a suitable law is selected for translation [40, 61]. Our results suggest that this principle is insufficient to guide the production of legal software. Both computer science and interdisciplinary teams consistently made errors when translating determinate elements of the law. Furthermore, computer scientists did not distinguish at all between determinate and indeterminate elements of the law. Since laws do not come pre-labeled as determinate or indeterminate, it is unclear how developers would draw this line in practice. In fact, it may not be possible to draw a neat distinction between determinate and indeterminate law at all. Prior work has shown that significant discretion is involved in translating laws framed as determinate, such as speed limits, which can lead to diverging code representations [57]. We conclude that the translatability of law depends not just on the formal structure of the law, but also on the training, practices, and communication styles of the people involved in constructing the translation. As a result, it is impossible to identify a priori whether the translation of any particular law into code will be accurate and reliable.

8.3 Future Work to Reorient Legal Software Systems

Given the difficulty of translating law into code, we encourage policymakers and developers to approach legal software with skepticism. When contemplating legal software, stakeholders should not imagine a perfectly implemented system, but a realistic system that contains mistranslations. When considering whether to install a new legal software system, policymakers and legal access advocates ought to weigh the consequences of mistranslations against the benefits of automation. If legal software is adopted, proactive interventions should contend with the fallibility of these systems. However, reducing such failures requires upstream intervention.

Because we find that disciplinary training strongly shapes the development of legal software, our study highlights the importance of pedagogical interventions. First, computer science training should abandon its messaging that frames programming as a modern "superpower" [27] for building anything imaginable. This attitude elides the fact that studying computer science does not provide

professionalized knowledge in other realms, giving computer scientists unwarranted confidence in their creations. Second, courses meant to bridge the divide between computer science and law should address the distinct modes of reasoning employed by each discipline. For instance, courses aimed at computer scientists could orient them toward understanding the people who interact with their software as clients rather than generic users. Similarly, courses aimed at lawyers could help them understand the capabilities of code and the principles of software evaluation. Third, future research should assess whether cross-training programs improve how practitioners in each field approach the translation process, as we observed members of the most successful interdisciplinary team in our study working to understand one another's domains.

Another direction for future work is to investigate how indeterminate law becomes altered during the translation process. In our study, when teams attempted to concretize the ambiguous standard of undue hardship, they consistently adopted less generous standards than those used by judges in real cases. In principle, codifying legal ambiguity could result in a ruleset that is over-inclusive or under-inclusive relative to existing standards. However, it is worth investigating whether the translation process involves a systematic tilt toward adopting harsher, more restrictive rules. Importantly, ambiguity is not careless under-specification in the legal system: it allows for dialogue, discretion, and considerations of individual circumstances that are hard to specify ex ante. It is possible that resolving ambiguity within software consistently cuts out moments of potential leniency, resulting in worse outcomes for people. Future studies could explore whether lawyers limit relief when representing ambiguous language across different legal contexts.

The limitations of our study suggest several directions for future research on legal software development. Importantly, we did not examine the full range of legal development settings and tasks. First, the teams in our study worked together for only a few hours. Although all teams finished the task in the allotted time, it is possible that their modes of reasoning and collaboration styles would shift during a longer project. Second, we did not scaffold or enforce best practices for software development. Though the teams all engaged in some form of testing, we did not require them to write unit tests or perform requirements analysis. Third, our experimental design treated the inclusion of a lawyer as a binary variable: either fully absent or fully present. Other work has suggested that lawyers often have a present-but-peripheral role in legal software development, rather than being co-located with developers throughout the development process [71]. More work is needed to see whether the duration, timing, and frequency of collaboration between computer scientists and lawyers alters the logics of team members or the artifacts they produce.

Finally, there is substantial potential for future work on legal software at CSCW. Most technical research on computational law has applied a formalist lens. Though formal methods are important for validating and testing legal software, they do not explain how these systems are built and embedded in society. The CSCW community could bring a sociotechnical focus to legal software applications and development. Many concerns of CSCW and legal software development overlap, such as supporting collaboration (including remote collaboration with distributed teams), bridging disciplines, investigating impacts of organizational structure (spanning professionalized and informal work settings), and evaluating systems. Future work at CSCW could consider new regimes for holistic evaluation, such as end-to-end audits that follow how software shuttles people through the legal system.

The processes we observe highlight the stubborn challenges impeding promised gains in accessibility, efficiency, and consistency from legal software. Translating law into code is hard and can easily stray from established legal practice. To avert dangerous consequences for the public, we must recognize and bridge the disciplinary divide between computer science and law that afflicts the development of legal software.

Acknowledgments

Thank you to Keith Porcaro, attendees of the ProLaLa 2022 and the Feasibility of Implementing Automatically Processable Law workshops, students enrolled in Public Policy 810 in Fall 2023 at the University of Michigan, Jen Triplett, the members of the CompHCI lab, and the reviewers for valuable suggestions on how to improve this manuscript. We are grateful to Alexander Miller, Jennifer Jiyoung Huseby, Divya Ramesh, Alice Liu, Sam Mikell, Matthew Bilik, and Nina Cahill for their outstanding research assistance.

References

[1] Abdi Aidid and Benjamin Alarie. 2023. *The Legal Singularity: How Artificial Intelligence Can Make Law Radically Better*. University of Toronto Press.

[2] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine Bias. *ProPublica* (2016). Retrieved September 23, 2024 from https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[3] Azer Bestavros, Stacey Dogan, Paul Ohm, and Andrew Sellars. 2022. *Bridging the Computer Science–Law Divide*. Technical Report. Georgetown Law, Institute for Technology Law & Policy.

[4] David A. Binder and Paul Bergman. 2003. Taking Lawyering Skills Training Seriously. *Clinical Law Review* 10 (2003), 191–220. https://doi.org/10.2139/ssrn.470903

[5] Anna Bobkowska and Magdalena Kowalska. 2010. On Efficient Collaboration Between Lawyers and Software Engineers When Transforming Legal Regulations to Law-related Requirements. In *2010 2nd International Conference on Information Technology*. IEEE, 105–109.

[6] Valerie Braithwaite. 2020. Beyond the Bubble that is Robodebt: How Governments that Lose Integrity Threaten Democracy. *Australian Journal of Social Issues* 55, 3 (2020), 242–259. https://doi.org/10.1002/ajs4.122

[7] Travis D. Breaux and Thomas Norton. 2022. Legal Accountability as Software Quality: A U.S. Data Processing Perspective. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*. IEEE, 101–113. https://doi.org/10.1109/RE54965.2022.00016

[8] Noah Bunnell. 2020. Remedying Public-Sector Algorithmic Harms: The Case for Local and State Regulation via Independent Agency. *Columbia Journal of Law & Social Problems* 54 (2020), 261–303.

[9] Ryan Calo and Danielle Keats Citron. 2021. The Automated Administrative State: A Crisis of Legitimacy. *Emory Law Journal* 70 (2021), 797–845.

[10] Terry Carney. 2019. Robo-Debt Illegality: The Seven Veils of Failed Guarantees of the Rule of Law? *Alternative Law Journal* 44, 1 (2019), 4–10. https://doi.org/10.1177/1037969X18815913

[11] Christian Chessman. 2017. A "Source" of Error: Computer Code, Criminal Defendants, and the Constitution. *California Law Review* (2017), 179–228. https://doi.org/10.2139/ssrn.2707101

[12] Danielle Keats Citron. 2008. Technological Due Process. *Washington University Law Review* 85, 6 (2008), 1249–1313.

[13] Jennifer Cobbe. 2020. Legal Singularity and the Reflexivity of Law. In *Is Law Computable? Critical Perspectives on Law and Artificial Intelligence*. Ed. Simon Deakin and Christopher Markou, Hart Publishing, 107–134. https://doi.org/10.2139/ssrn.3858474

[14] Mark Cohen. 2020. The FT Innovative Lawyers-Global Legal Hackathon: Meeting The Challenge Of Law In The Digital Age. *Forbes* (2020). Retrieved September 23, 2024 from https://www.forbes.com/sites/markcohen1/2020/05/26/the-ft-innovative-lawyers-global-legal-hackathon-meeting-the-challenge-of-law-in-the-digital-age/

[15] Colin Lecher. 2018. What Happens When an Algorithm Cuts Your Health Care. *The Verge* (2018). Retrieved September 23, 2024 from https://www.theverge.com/2018/3/21/17144260/healthcare-medicaid-algorithm-arkansas-cerebral-palsy

[16] Joseph Dainow. 1966. The Civil Law and the Common Law: Some Points of Comparison. *The American Journal of Comparative Law* 15, 3 (1966), 419–435. https://doi.org/10.2307/838275

[17] David Freeman Engstrom, Daniel E. Ho, Catherine M Sharkey, and Mariano-Florentino Cuéllar. 2020. Government by Algorithm: Artificial Intelligence in Federal Administrative Agencies. *NYU School of Law, Public Law Research Paper* 20-54 (2020). https://doi.org/10.2139/ssrn.3551505

[18] Nel Escher and Nikola Banovic. 2020. Exposing Error in Poverty Management Technology: A Method for Auditing Government Benefits Screening Tools. In *Proceedings of the ACM on Human-Computer Interaction (CSCW1, Vol. 4)*. Association for Computing Machinery, New York, NY, 1–20. https://doi.org/10.1145/3392874

[19] Virginia Eubanks. 2018. *Automating Inequality: How High-Tech Tools Profile, Police, and Punish the Poor*. St. Martin's Press.

[20] Matthew S. Farina. 2021. Schoolbooks and Shackles: The Undue Hardship Standard and Treatment of Student Debt at Bankruptcy. *Boston College Law Review* 62, 5 (2021), 1620–1664.

[21] Katherine Fink. 2018. Opening the Government's Black Boxes: Freedom of Information and Algorithmic Accountability. *Information, Communication & Society* 21, 10 (2018), 1453–1471. https://doi.org/10.1080/1369118X.2017.1330418
[22] Kenneth R Fleischmann. 2006. Boundary Objects with Agency: A Method for Studying the Design–Use Interface. *The Information Society* 22, 2 (2006), 77–87. https://doi.org/10.1080/01972240600567188
[23] Michael Genesereth. 2015. Computational Law: The Cop in the Backseat. *White Paper, CodeX: The Center for Legal Informatics Stanford University* (2015).
[24] Daniel Goldsworthy. 2019. Dworkin's Dream: Towards a Singularity of Law. *Alternative Law Journal* 44, 4 (2019), 286–290. https://doi.org/10.1177/1037969X19875825
[25] Ben Green. 2022. The Flaws of Policies Requiring Human Oversight of Government Algorithms. *Computer Law & Security Review* 45 (2022), 105681. https://doi.org/10.1016/j.clsr.2022.105681
[26] Ben Green and Salomé Viljoen. 2020. Algorithmic Realism: Expanding the Boundaries of Algorithmic Thought. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. ACM, Barcelona Spain, 19–31. https://doi.org/10.1145/3351095.3372840
[27] Alexandria Hansen, Jim Gribble, Amber Moran, Eric Hansen, and Danielle Harlow. 2021. Making Computer Science Accessible. *Science and Children* 58, 5 (2021), 80–86.
[28] HelloPrenup. 2023. *Online Prenuptial Agreements - Affordable Prenups.* Retrieved September 23, 2024 from https://helloprenup.com/
[29] Mireille Hildebrandt. 2020. Code-Driven Law: Freezing the Future and Scaling the Past. In *Is Law Computable? Critical Perspectives on Law and Artificial Intelligence*. Ed. Simon Deakin and Christopher Markou, Hart Publishing, 67–84. https://doi.org/10.2139/ssrn.3522079
[30] Mireille Hildebrandt. 2023. Grounding Computational 'Law' in Legal Education and Professional Legal Training. In *Research Handbook on Law and Technology*. Ed. Bartosz Brożek, Olia Kanevskaia, and Przemysław Pałka, Elgar Publishing, 99–127. https://doi.org/10.4337/9781803921327.00014
[31] Dominique Hogan-Doran. 2017. Computer Says "No": Automation, Algorithms and Artificial Intelligence in Government Decision-Making. *Judicial Review: Selected Conference Papers: Journal of the Judicial Commission of New South Wales* 13, 3 (2017), 345–382.
[32] Immi. 2023. *Do You Know Your Path to Legal Immigration Status?* Retrieved September 23, 2024 from https://www.immi.org/en
[33] Jason Iuliano. 2012. An Empirical Assessment of Student Loan Discharges and the Undue Hardship Standard. *The American Bankruptcy Law Journal* 86, 3 (2012), 495–526. https://doi.org/10.2139/ssrn.1894445
[34] Tech For Justice. 2017. *Tech For Justice Hackathon+ Veterans - Project Gallery.* Retrieved September 23, 2024 from https://tech-for-justice-veterans.devpost.com/project-gallery
[35] David Kennedy and William W. Fisher. 2007. *The Canon of American Legal Thought.* Princeton University Press. https://doi.org/10.2307/j.ctv39x51k
[36] Marc Lauritsen and Quinten Steenhuis. 2019. Substantive Legal Software Quality: A Gathering Storm?. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law (ICAIL '19)*. Association for Computing Machinery, New York, NY, 52–62. https://doi.org/10.1145/3322640.3326706
[37] Brian Leiter. 1999. Positivism, Formalism, Realism. *Columbia Law Review* 99 (1999), 1138–1164. https://doi.org/10.2307/1123484
[38] Victor Li. 2017. Chatbot to Help Victims of Military Sex Trauma Wins Tech for Justice Hackathon Event. *ABA Journal* (2017). Retrieved September 23, 2024 from https://www.abajournal.com/news/article/chatbot_to_help_victims_of_military_sex_trauma_wins_tech_for_justice_hackat
[39] Thomas Lodato and Carl DiSalvo. 2018. Institutional Constraints: the Forms and Limits of Participatory Design in the Public Realm. In *Proceedings of the 15th Participatory Design Conference: Full Papers-Volume 1*. 1–12. https://doi.org/10.1145/3210586.3210595
[40] Nathaniel Love and Michael Genesereth. 2005. Computational Law. In *Proceedings of the 10th International Conference on Artificial Intelligence and Law (ICAIL '05)*. Association for Computing Machinery, New York, NY, 205–209. https://doi.org/10.1145/1165485.1165517
[41] Steve McConnell. 2004. *Code Complete.* Pearson Education.
[42] Mona Nikidehaghani, Jane Andrew, and Corinne Cortese. 2022. Algorithmic Accountability: Robodebt and the Making of Welfare Cheats. *Accounting, Auditing & Accountability Journal* 36, 2 (2022), 677–711. https://doi.org/10.1108/AAAJ-02-2022-5666
[43] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition.* Basic Books.
[44] Law Help NY. 2023. *I am Serving Active Duty and Facing Eviction. What Can I Do?* Retrieved September 23, 2024 from https://www.lawhelpny.org/resource/letter-to-landlord-eviction
[45] House of Delegates. 2020. Model Rules of Professional Conduct. American Bar Association. Retrieved September 23, 2024 from https://www.americanbar.org/groups/professional_responsibility/publications/model_rules_of_

professional_conduct
[46] Colorado Department of Health Care Policy & Financing. 2024. *Care and Case Management System.* Retrieved September 23, 2024 from https://hcpf.colorado.gov/care-case-management-system
[47] The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems. 2019. Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems. *IEEE* (2019).
[48] Frank Pasquale. 2016. Automating the Professions: Utopian Pipe Dream or Dystopian Nightmare. *Los Angeles Review of Books* (2016).
[49] Frank Pasquale. 2019. A Rule of Persons, Not Machines: The Limits of Legal Automation. *The George Washington Law Review* 87 (2019), 1–55.
[50] Jeanne Frazier Price. 2013. Wagging, Not Barking: Statutory Definitions. *Cleveland State Law Review* 60, 4 (2013), 999–1055.
[51] In re Long. 2003. *322 F.3d 549* (2003).
[52] Tapani Rinta-Kahila, Ida Someh, Nicole Gillespie, Marta Indulska, and Shirley Gregor. 2023. Managing Unintended Consequences of Algorithmic Decision-Making: The Case of Robodebt. *Journal of Information Technology Teaching Cases* (2023), 165–171. https://doi.org/10.1177/20438869231165538
[53] Andrea Roth. 2016. Trial by Machine. *Georgetown Law Journal* 104, 5 (2016), 1245–1305.
[54] Andrea L Roth. 2017. Machine Testimony. *Yale Law Journal* 126, 1 (2017), 1973–2053.
[55] Pierre Schlag. 1985. Rules and Standards. *UCLA Law Review* 33 (1985), 379–430.
[56] Suzanne J. Schmitz and Alice M. Noble-Allgire. 2011. Reinvigorating the 1L Curriculum: Sequenced Writing Across the Curriculum Assignments as the Foundation for Producing Practice-Ready Law Graduates. *Southern Illinois University Law Journal* 36 (2011), 287–315. https://doi.org/10.2139/ssrn.1879618
[57] Lisa A. Shay, Woodrow Hartzog, John Nelson, and Gregory Conti. 2016. Do Robots Dream of Electric Laws? An Experiment in the Law as Algorithm. In *Robot Law*. Edward Elgar Publishing, 274–305. https://doi.org/10.4337/9781783476732.00020
[58] Gilien Silsby. 2018. Global Legal Hackathon: USC Law Prof. Challenges Hackers to Tackle 'Problems Worth Solving'. *USC Gould School of Law* (2018). Retrieved September 23, 2024 from https://gould.usc.edu/news/global-legal-hackathon-usc-law-prof-challenges-hackers-to-tackle-problems-worth-solving/
[59] Susan Leigh Star and James R Griesemer. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science* 19, 3 (1989), 387–420. https://doi.org/10.1177/030631289019003001
[60] Esther Surden. 2017. At CourtHack 2017, Hackers Present Solutions For Real-World Justice System Problems. *New Jersey Tech Weekly* (2017). Retrieved September 23, 2024 from https://njtechweekly.com/at-courthack-2017-hackers-present-solutions-for-real-world-justice-system-problems/
[61] Harry Surden. 2010. The Variable Determinacy Thesis. *Columbia Science and Technology Law Review* 12, 1 (2010). https://doi.org/10.7916/stlr.v12i0.3949
[62] Richard Susskind. 2008. *The End of Lawyers? Rethinking the Nature of Legal Services.* Oxford University Press.
[63] Iddo Tavory and Stefan Timmermans. 2014. *Abductive Analysis: Theorizing Qualitative Research.* University of Chicago Press.
[64] Intuit TurboTax. 2023. *File Taxes Online, Tax Filing Made Easy.* Retrieved September 23, 2024 from https://turbotax.intuit.com/
[65] Thomas v. Dep't of Educ. (In re Thomas). 2019. *931 F.3d 449* (2019).
[66] Brunner v. New York State Higher Educ. Serv. 1987. *831 F.2d 395* (1987).
[67] Ivory v. United States (In re Ivory). 2001. *269 B.R. 890* (2001).
[68] Erkson v. U.S. Dept. of Ed. 2018. *582 B.R. 542* (2018).
[69] Smith v. U.S. Dept. of Ed. 2018. *582 B.R. 556* (2018).
[70] Sarah Valentine. 2019. Impoverished Algorithms: Misguided Governments, Flawed Technologies, and Social Control. *Fordham Urban Law Journal* 46 (2019), 364–427.
[71] Ari Ezra Waldman. 2018. Designing Without Privacy. *Houston Law Review* 55 (2018), 659–727.
[72] Nyu Wang and Michael Yuan Tian. 2022. 'Intelligent Justice': AI Implementations in China's Legal Systems. In *Artificial Intelligence and Its Discontents: Critiques from the Social Sciences and Humanities*. Springer, 197–222. https://doi.org/10.1007/978-3-030-88615-8_10
[73] Ran Wang. 2020. Legal Technology in Contemporary USA and China. *Computer Law & Security Review* 39 (2020). https://doi.org/10.1016/j.clsr.2020.105459
[74] Quicken Willmaker. 2024. *Estate Planning You Can Trust.* Retrieved September 23, 2024 from https://www.willmaker.com/
[75] Stephanie Wykstra. 2020. Government's Use of Algorithm Serves Up False Fraud Charges. *Undark* (2020). Retrieved September 23, 2024 from https://undark.org/2020/06/01/michigan-unemployment-fraud-algorithm/

A Screenshots of Legal Tools

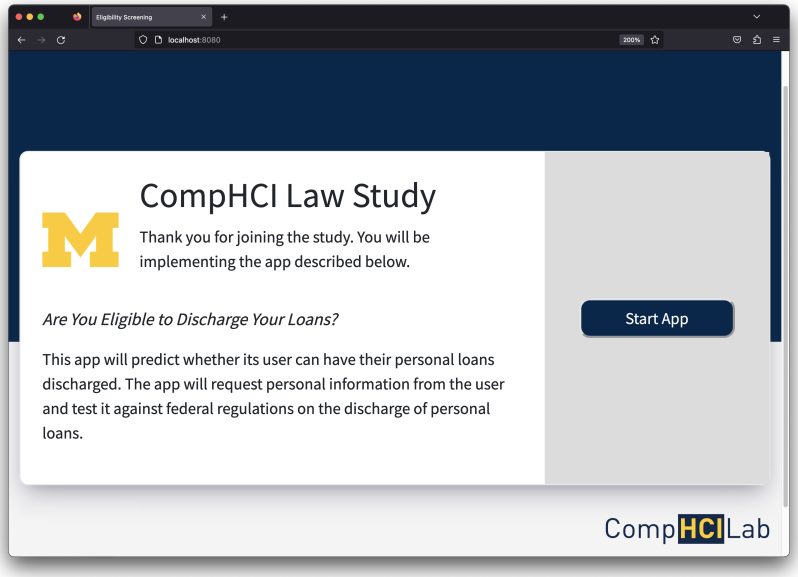


Fig. A.1. Index page of starter code.

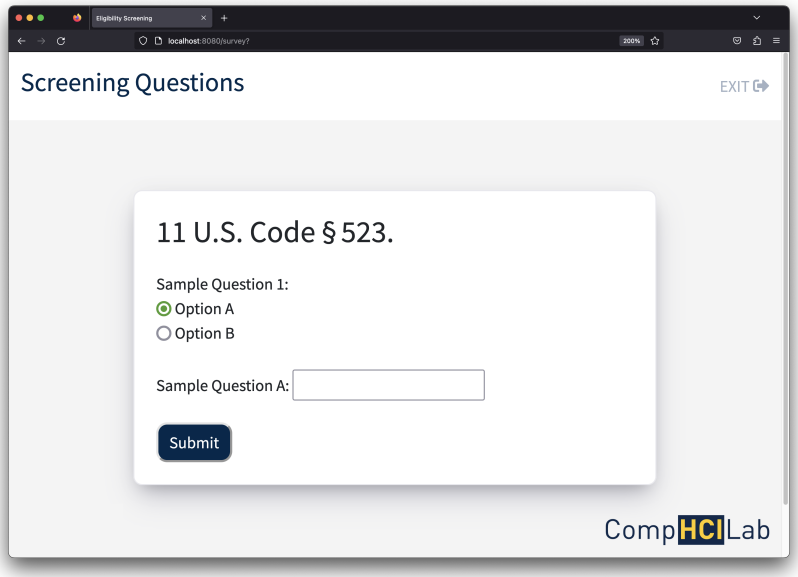


Fig. A.2. Screening questions page of starter code.

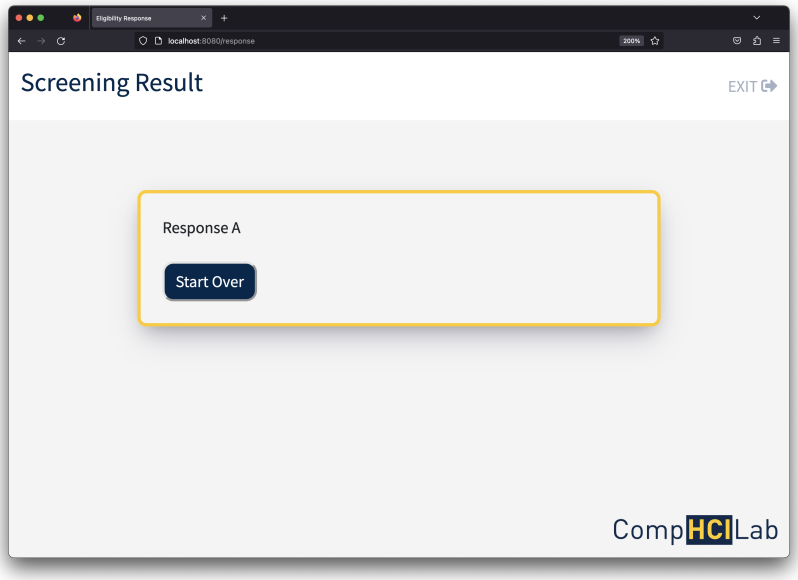


Fig. A.3. Results page of starter code.

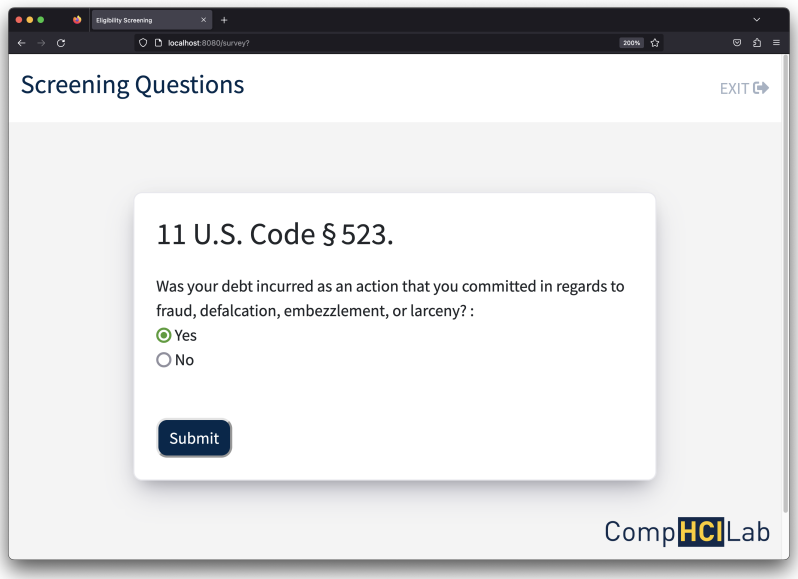


Fig. A.4. Screening questions page of Trial 1 (CS).

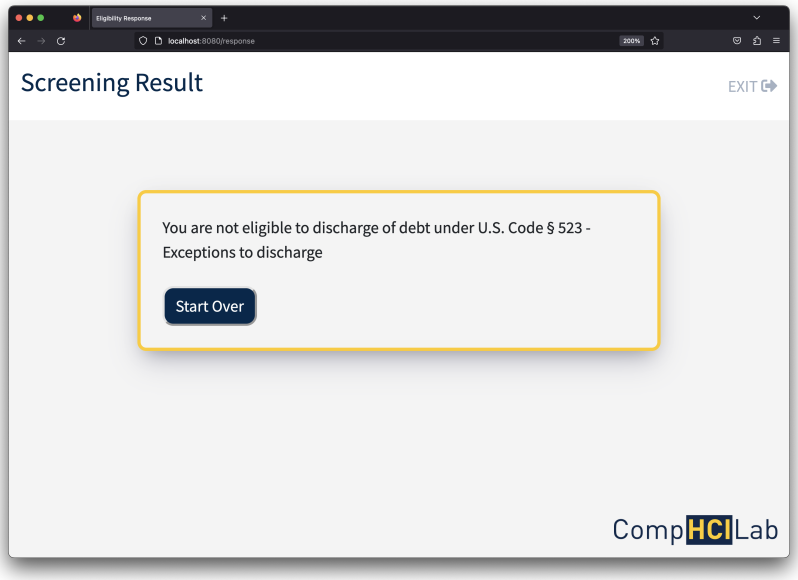


Fig. A.5. Results page of Trial 1 (CS).

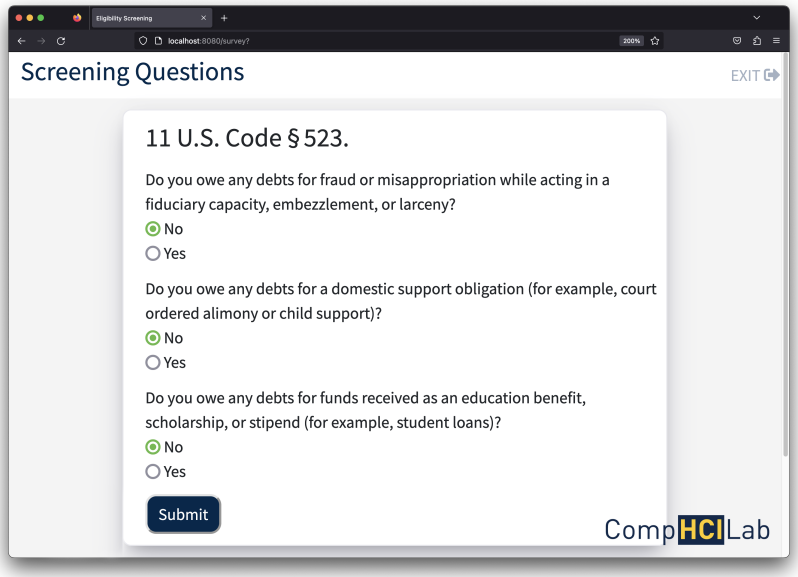


Fig. A.6. Screening questions page of Trial 3 (CS+L).

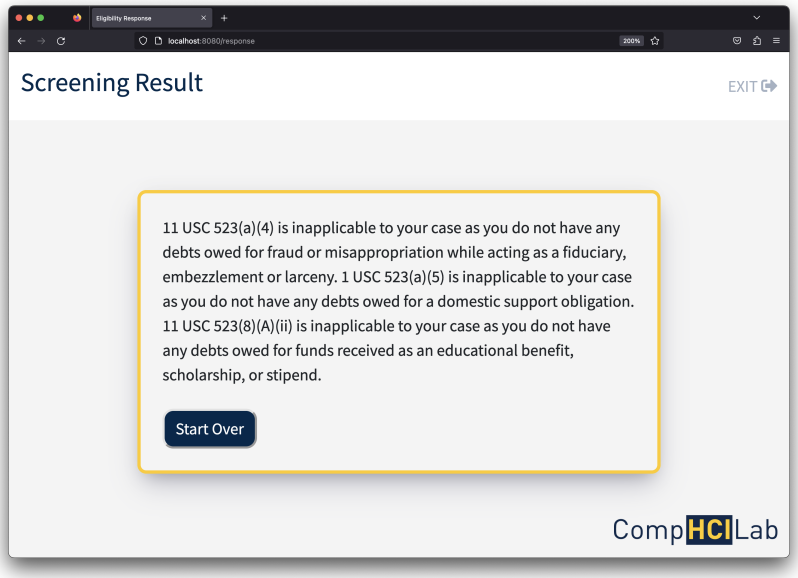


Fig. A.7. Results page of Trial 3 (CS+L).

B Trial Duration

Table B.1. Duration in hours and minutes that each team worked on each trial component.[10]

Trial	Team Type	Warm-up	Main Study Task	Total
T1	CS	1:31	1:28	2:59
T2	CS	0:59	1:03	2:03
T3	CS+L	0:58	2:37	3:35
T4	CS	1:10	1:20	2:30
T5	CS+CS	0:37	3:47	4:25
T6	CS+L	1:42	3:14	4:26
T7	CS+L	0:44	3:37	3:42
T8	CS+L	0:31	1:47	2:19
T9	CS	1:34	1:22	2:57
T10	CS+L	2:10	1:31	3:41
T11	CS	0:33	1:19	1:53
T12	CS+L	1:02	3:03	3:41
T13	CS+CS	1:12	1:31	2:44
T14	CS+CS	0:51	1:20	2:14
T15	CS+CS	1:04	1:10	2:15

Table B.2. Average duration in hours and minutes spent on each trial component for each team type.

Team Type	Warm-up	Main Study Task	Total
CS	1:09	1:28	2:28
CS+CS	0:56	1:57	2:54
CS+L	1:11	2:44	3:34

[10]Warm-up and main study task durations may not sum up to the total duration, as some legal experts started the main task while their partners were finishing the warm-up.

C Diagram-Based Intermediate Representation

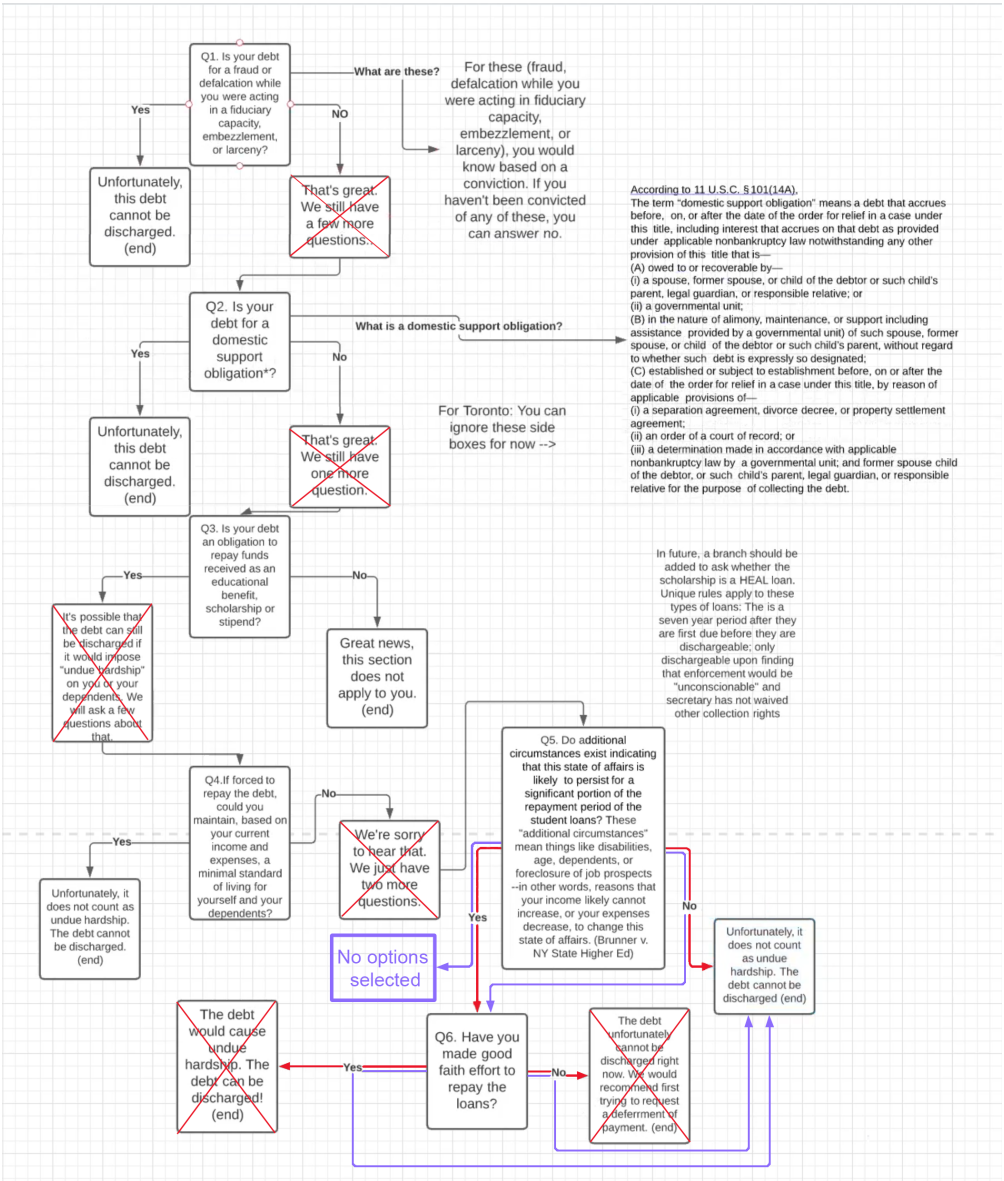


Fig. C.1. Flowchart constructed by legal expert, overlaid with colored annotations. Red represents messages and edges removed by computer scientist in final tool; purple represents messages and edges added by computer scientist. These annotations demonstrate how final tools could deviate from a legal expert's instructions, indicating that the translation between intermediate representations and software code is a potential site of error. (T12 CS+L)

Received July 2023; revised January 2024; accepted March 2024