REFACTORING OUR DOCUMENTS: HOW SOFTWARE CODE CAN INFORM LEGAL PROSE

Justin Lischak Earley [†]

"Code is law," declared Lawrence Lessig in an oft-quoted but oft-misunderstood passage published on the first day of the new millennium.[1] Indeed, there are good reasons to draw a parallel between "law coders" and "software coders."[2]

Both lawyers and software developers typically receive general direction from their clients, translating it into defined rules.[3] For example, a software developer may be told to create a new video game. The clients probably have a general sense of what goes into the game: the high-level rules, the plotline, what constitutes winning or losing, etc. But the developer cannot code a mere general sense, and it is unlikely that the clients have contemplated all the situations that might arise during gameplay. The developer must therefore think through the entire range of possibilities, and must make decisions about how to handle each scenario. The developer must then synthesize those into a set of rules that will govern gameplay. Where a rule is widely accepted (for example, gravity exists), it is reasonable for the developer to make the tactical decision to write code that causes objects in the game to fall downward, without consulting with the client. But where the rule is unique or critical (for example, whether the main character has gravity-defying superpowers of flight), the developer will likely discuss how to handle the situation as a strategic matter with the client. And finally, all the rules that the developer utilizes must be accurately transcribed into a writing that will stand entirely on its own, without the need for interpretation or human intervention, potentially lasting for years.

So also with us as real estate lawyers. A client will typically have a general sense of what goes into a real estate transaction: the property, a concept project, the economic terms, etc. But few

---

[†] I currently serve as Underwriting Innovation Director in the Corporate Underwriting Department (a/k/a Home Office Underwriting Department) of First American Title Insurance Company in Santa Ana, California. The views expressed here are my own personal views, and may not necessarily be the views of my employer. You can find me at https://jdlesq.com, a site I built using the newfound skills discussed in this piece. Many thanks to Don Patterson (https://www.djp3.net/), Paul L. Hammann, and Laura E. Schmidl for their advice and contributions to this piece.

[1] Lawrence Lessig, *Code is Law: On Liberty in Cyberspace*, HARVARD MAGAZINE, Jan. 1, 2000, *available at* https://www.harvardmagazine.com/2000/01/code-is-law-html (last visited July 14, 2020). Lessig's point was that the then-new technology of the internet should be regulated by law, and if it is not, then the code itself will *de facto* determine the boundaries of permissible online behavior – and thus the code will effectively "become" the law. Twenty years hence, Lessig's point that an unregulated internet will not "teleport us to Eden" seems eerily prescient. *See id.* That is not the subject of this paper, however.

[2] To this end, I respectfully disagree with those who propose that lawyers and programmers do fundamentally different things. While this may be the case for litigators, I see strong parallels between the activities of programmers and transactional real estate practitioners. *Contra* Jason Krause, *Does Learning to Code Make You a Better Lawyer?,* ABA JOURNAL, Sept. 1, 2016, *available at* https://www.abajournal.com/magazine/article/lawyer_learning_code_zvenyach_ohm (last visited July 14, 2020).

[3] For a view somewhat similar to my own, *see generally* Jermaine Paul Smith, *Why lawyers (should) make great developers*, MEDIUM, Nov. 6, 2018, https://medium.com/datadriveninvestor/why-lawyers-should-make-great-developers-155f73304d0 (last visited July 14, 2020). While Mr. Smith and I agree on several points, as will be seen *infra*, there are also some places where we diverge quite dramatically. *See, e.g., infra* section 2.

clients have thought through all the factual permutations that might arise in the course of a transaction or business relationship. Rather, it falls to us, as lawyers, to contemplate the range of possible fact patterns, and to synthesize those into contractual rules that will govern the transaction or relationship. Where the rule is widely accepted (for example, commercial properties are held by single-purpose LLCs), it is a reasonable tactical decision for the lawyer to adopt that rule. But if the rule is unique or critical (for example, whether a property tax payment default triggered by a global pandemic should constitute a "full recourse" event under a nonrecourse carveout guaranty covenant), the lawyer should discuss this strategy question with the client.[4] And of course, the documents that the lawyer writes must accurately transcribe the rules that have been selected, and must do so in a way that can be correctly applied years later, after memories have faded and behavioral incentives may have realigned.

As much of my work over the past few years has become "FinTech" / "InsurTech" / "PropTech" in nature, I returned to school this past year in order to sharpen my skills. By the time this article goes to print, I will be a few weeks away from completing a masters' degree in human-computer interaction design.[5] As part of my studies, I learned a little bit of coding in three basic front-end web development languages. This process was eye-opening, and drove home the similarities between software developers and transactional lawyers.

It also got me thinking about what I, as a lawyer, could learn from those who code software rather than laws. This article thus contains the software-coding ideas and techniques that I believe we, as lawyers, could all deploy in order to better "code" our statutes and contracts.

1) The simpler, the better

Plain language legal writing has been "a thing" for at least 20 years,[6] but it never seems to be "the thing." To put it simply, we lawyers like using a lot of words. There are good reasons for this cultural norm, some of which are discussed below.[7] But deep down, I think most of us would admit that much of the law code we create is more verbose than is necessary.

Software developers have learned over the course of time that simpler code is better code. Consider the following two pieces of JavaScript:

---

[4] *See, e.g.*, MODEL RULES PROF'L CONDUCT R. 1.2(a), *available at* https://www.americanbar.org/groups/professional_responsibility/publications/model_rules_of_professional_conduct/rule_1_2_scope_of_representation_allocation_of_authority_between_client_lawyer/ (last visited July 14, 2020) ("[A] lawyer shall abide by a client's decisions concerning the objectives of representation and . . . shall consult with the client as to the means by which they are to be pursued. A lawyer may take such action on behalf of the client as is impliedly authorized to carry out the representation.").

[5] Barring any unforeseen difficulties, that is. By way of background, this is my program's homepage: https://mhcid.ics.uci.edu/ (last visited July 14, 2020). If you are interested in the subject, I highly recommend it.

[6] *See generally, e.g.*, BRYAN GARNER, LEGAL WRITING IN PLAIN ENGLISH (1st ed. 2001).

[7] *See, e.g.*, sections 4 and 6, *infra*.

```
if (isGood === false || isGood === null || isGood === undefined || isGood === 0 || isGood
=== " " || isGood = NaN) {
console.log('Not good.');}
```

```
if(!isGood) {
console.log('Not good.');}
```

One need not understand what these two code blocks do or what they mean. They are materially identical in substance. Consider as a thought experiment which of these is more likely to result in transcription errors (whether merely typographical or substantive). The answer is clear: the more you must transcribe, the more likely you are to introduce errors. Software developers call those "bugs." We lawyers call them "ambiguities." Software code with bugs either does not run, or runs incorrectly. Legal code with ambiguities foments litigation.

Given a choice between two formulations that are identical in substance, choosing the simpler one is the smart move, whether coding software or laws.

2) Keep it DRY

Legal words are often terms of art with specialized meanings, and it is incumbent upon us to know those meanings. Therefore, it really can be necessary at times to string together several words with similar (but not identical) meanings into a laundry list, in order to prevent arguments about whether some minor twist was permitted or prohibited. For example, there is a real difference between "selling," "leasing," and "conveying," and so a laundry list prohibition that "Seller shall not sell, lease, or convey" to anyone but Purchaser can be necessary.

That said, unnecessarily repeating oneself is unfortunately endemic in contract-drafting. In addition to tripping over the "simpler is better" principle discussed above, unnecessary repetition creates risks to our documents, because a change made in one place may not be replicated in another, creating a mismatch that the program running the software (read: "judge") may later misinterpret. The problem with laundry lists is that you must make sure that no socks get lost in the dryer.

Programmers recognized this problem, and a basic tenet of software development is "Don't Repeat Yourself" ("DRY"). Good code is DRY code.[8] Bad code is WET code (that's "Write Everything Twice," "We Enjoy Typing," or at worst, "Waste Everyone's Time").[9] If you must code something complicated, smart coders know that you should code it once, and then give it a

[8] *See, e.g.*, Reading 3: Code Review, http://web.mit.edu/6.005/www/sp15/classes/03-code-review/ (last visited July 14, 2020) ("Duplicated code is a risk to safety. If you have identical or very similar code in two places, then the fundamental risk is that there's a bug in both copies, and some maintainer fixes the bug in one place but not the other.")

[9] *See, e.g.*, Don't repeat yourself, https://en.wikipedia.org/wiki/Don%27t_repeat_yourself (last visited July 14, 2020).

name so that it can be later invoked without having to re-transcribe the whole block. In coding software, those are called "functions." In coding law, those are called "definitions."

3) Beware of killer commas

Punctuation kills. Software, that is.[10] I lost count of how many hours I spent staring at a computer screen, desperately trying to figure out why my code was throwing off errors. In nearly every instance, it was a typographical issue causing my bug. My most common typographical misstep was one involving punctuation – commas and semicolons and curly brackets serve crucial roles in JavaScript, and a missing or misplaced punctuation item can cause hours of work to sputter on the runway.

As the "Oxford comma case"[11] illustrates, punctuation matters in legal code as well. Small characters that can have an outsize effect on meaning merit special attention. That includes not just the placement of commas, but also crucial-but-tiny words such as "and" vs. "or."

4) Stand on the shoulders of (good) precedent

As lawyers, we've all got examples of documents that we reach for when a fact pattern calls for it. Need a special warranty deed? Get it from the X deal. Need a loan agreement? The Y deal is a great one to start with. "Borrowing" from prior work is a well-established tradition in real estate law: it's inefficient to start from scratch every time, and doing so increases the likelihood of forgetting or mis-transcribing important points.

Software coders have a similar approach, although here is a place where we lawyers could take a page out of their book. Software coders utilize "libraries" of code that have been made freely available as "open source" for anyone to use, without a license. Two of the most common open-source code libraries used in web development are "Bootstrap"[12] and "jQuery,"[13] which power many websites on the internet today. These libraries are maintained by volunteers, who keep them up to date and current with the advance of technology. Working on these libraries as a volunteer is a respected, socially-useful form of giving back to the web development profession.

Pro bono work and advancing the legal profession are moral obligations of lawyers. There are many ways to give back, including things that aren't traditional "representation of clients": teaching, speaking, writing, etc.[14] Today, the practice of real estate law has no equivalent of Bootstrap or jQuery. Perhaps it should. Perhaps we, as lawyers, should find a way to better our

---

[10] That said, there is also this paradigm example: https://www.amazon.com/Lets-Eat-Grandpa-Punctuation-Rocks/dp/B008VIJWBQ (last visited July 14, 2020).
[11] *O'Connor v. Oakhurst Dairy*, 851 F.3d 69, 70 (1st Cir. 2017) ("For want of a comma, we have this case.").
[12] Bootstrap – The most popular HTML, CSS, and JS library in the world, https://getbootstrap.com/ (last visited July 14, 2020).
[13] jQuery, https://jquery.com/ (last visited July 14, 2020).
[14] MODEL R. PROF'L CONDUCT R. 6.1(b)(3), *available at* https://www.americanbar.org/groups/professional_responsibility/publications/model_rules_of_professional_conduct/rule_6_1_voluntary_pro_bono_publico_service/ (last visited July 14, 2020) ("[T]he lawyer should . . . participat[e] in activities for improving the law, the legal system or the legal profession.").

profession by offering more open-source, party-neutral, model forms of common real estate transactional instruments that our fellow lawyers could utilize as a starting point. I fully recognize the challenges of creating such documents,[15] but they do already exist in certain places and for certain circumstances.[16] The Uniform Law Commission has done much good work in the area of "statutory code." I submit that our practice could be much advanced by some group[17] creating model, open source "contract code."[18]

5) Scan for vulnerabilities

The internet is dangerous. The news headlines are full of nefarious acts by brigands who hack and harm websites, just because they can. Because technology is constantly advancing, software code that was safe yesterday may not be safe today. Software coders have learned that vulnerability scans are a part of their lives, and that software code must be periodically updated from time to time to close vulnerabilities created by emerging threats.[19]

Legal code is similar, but unlike software coders, we law coders cannot rely on automated systems to scan for vulnerabilities on our behalf. Instead, we must keep up with emerging developments, new cases, and new statutes, making determinations as to whether our documents can withstand the tests new challenges may pose. Here again is good reason to consider model transactional documents as a pro bono effort for the good of the profession. Bootstrap and jQuery are constantly updated to eliminate emerging threats and to offer software coders a safe and sound model from which to start and customize their projects. Could not the same thing benefit the practice of real estate law?

---

[15] Some of the likely challenges and objections include: the fact that software code is the same everywhere, but laws differ across the 50 states; the fact that transactions differ by parties and their goals; the fact that stock legal documents tend to become "pro" one party and "con" another depending on who drafts them; the fact that stock documents can be misused by the untrained; etc. In my view, these challenges (and others) can be overcome.

[16] Model legal opinions are one such place. *See, e.g.*, *Inclusive Real Estate Secured Transaction Opinion*, AMERICAN BAR ASSOCIATION & AMERICAN COLLEGE OF REAL ESTATE LAWYERS, Feb. 2, 1999, *available at* https://cdn.ymaws.com/acrel.site-ym.com/resource/resmgr/Docs/Inclusive_Real_Estate_Secure.pdf (last visited July 14, 2020); *Amended and Restated Report on Legal Opinions to Third Parties in Georgia Secured Real Estate Transactions*, GEORGIA REAL PROPERTY LAW SECTION, Mar. 17, 2009 (on file with author).

[17] It should be noted that the eminent Prof. Pat Randolph's website "Dirt" (ably run today by the great Prof. Dale Whitman) has long been a collector of forms. *See, e.g.*, Forms from Friends of Dirt, http://dirt.umkc.edu/files.htm#abaforms (last visited July 14, 2020); *see also* Dirt Archives – A web site and list-serv for real estate law professionals, https://whitmandale.wixsite.com/dirt/forms (site under construction) (last visited July 14, 2020). What I propose here is a more robust, open-source-oriented next step to those laudable efforts, going beyond just collecting forms, and instead actively curating and updating them as an "open source pro bono" effort to the profession.

[18] I recognize that there are any number of reasons (especially perhaps economic reasons) why we as lawyers have not yet done what I suggest. Of course, many of those same reasons apply to software. Bootstrap was invented at Twitter in 2010. *See* About – Bootstrap v4.5, https://getbootstrap.com/docs/4.5/about/overview/ (last visited July 14, 2020). Rather than keeping the invention to themselves and attempting to monetize it, Twitter offered it to the world as open-source software, and have reaped huge reputational rewards for doing so. Could not some group in our profession do the same?

[19] *See, e.g.*, Vulnerability Scanning Tools – Veracode, https://www.veracode.com/security/vulnerability-scanning-tools (last visited July 14, 2020).

6) "Whenever you draft what happens, draft what happens if it doesn't happen."

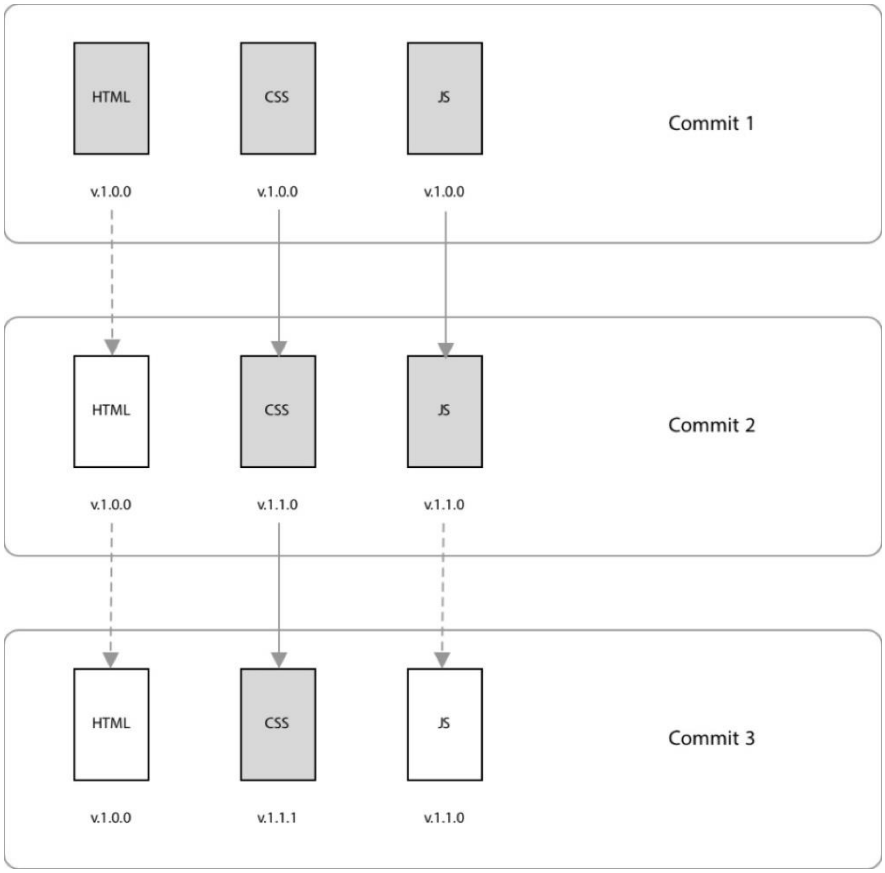
When I was a newly-minted lawyer, a fellow of this College taught me the lesson quoted above. I've carried his words with me ever since.[20] In coding software, every "if" statement generally needs a "then" statement. But it also generally needs an "else if" statement. Otherwise, the code may simply hang, leaving the program spinning in a circle with nowhere to go.[21]

Legal code can suffer from the same bug. If the contract states that landlord must send tenant a notice of default, what happens if the notice of default does not arrive, or if tenant refuses to accept it? In all types of coding, something should happen, even if nothing happens.

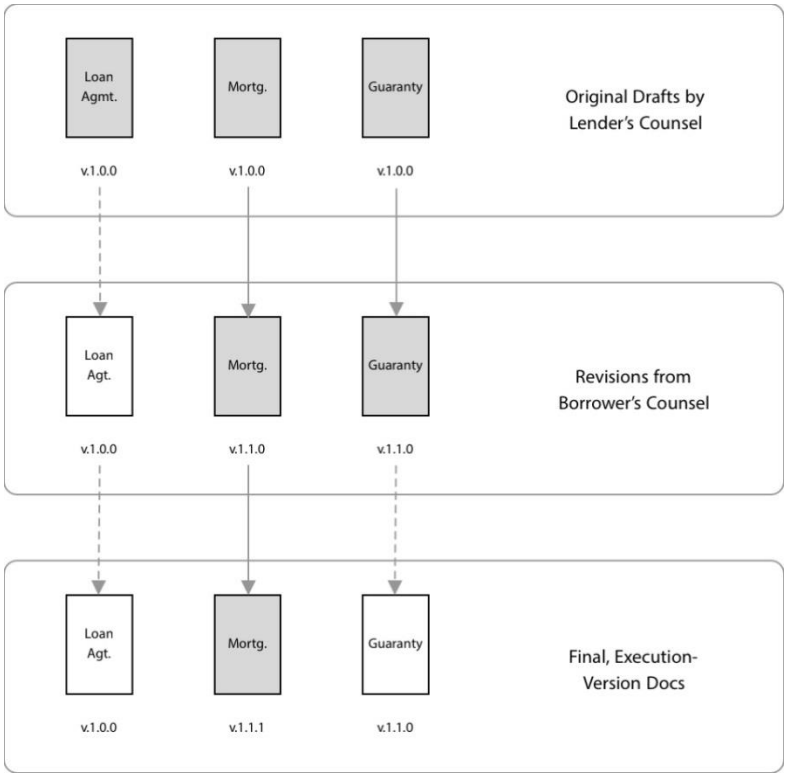
7) Source code management

In my journey through learning basic web development, I found nothing more perspective-altering than the concept of source-code management, commonly done through a program called "Git." The basic premise is this: Suppose you have a website comprised of three files: HTML, CSS, and JavaScript. They must all work together as a package for the website to function. You realize that you'd like to update the JavaScript file with some new code, creating a new version of it, but you want to keep working with the original versions of your HTML and CSS files. Git knows how to group your package of files together in something it calls a "commit," enabling you to always know exactly what package you have deployed, and when you deployed it. Think of it this way:

[20] Many thanks, Rob! https://us.eversheds-sutherland.com/People/R-Robinson-Plowden (last visited July 14, 2020).
[21] There are situations in both a software coding sense and a law-coding sense where an "if / then" statement may not need an "else if" statement. But as a general matter, the rule of thumb holds.



The parallels to real estate transactions are apparent. Consider the example above, but replace "HTML," "CSS," and "JavaScript" with "Loan Agreement," "Mortgage," and "Guaranty." Each document in a negotiation goes through various versions. Documents are packaged together as groups that all must work together, with each package being a "commit." The commit that actually "runs" the transaction occurs when execution versions of all the transactional documents are agreed-to by all parties. Thus:



Git also allows you to "roll back" to a prior commit, and it even creates automatic redlines of each version of a file, enabling you to trace changes across file history seamlessly! Imagine the logistical pain that could be saved by a "Git for real estate lawyers." Our kind has struggled with document management systems since the rise of law-office computing, and no magic bullet has ever been found. Perhaps we have simply been looking in the wrong place.

I am not the first person to realize this potential use case.[22] Unfortunately, creating "Git for dirt lawyers" is much harder than it seems, because Git assumes that it is ingesting code.[23] It cannot as yet work its magic on the Microsoft Word documents that anchor our workflow.[24] Further, understanding Git's commands takes some training and technical know-how. And so, Git remains forbidden fruit in our realm for the time being, awaiting the person or company who can create a user-friendly "Git for dirt lawyers." A product that solves this problem could make such a person or company fabulously wealthy.[25]

---

[22] For one view, *see* Xavier Beauchamp-Tremblay, *GitHub Workflow and Legal Drafting*, SLAW: CANADA'S ONLINE LEGAL MAGAZINE, Apr. 21, 2017, *available at* http://www.slaw.ca/2017/04/21/github-workflow-and-legal-drafting/ (last visited July 14, 2020).

[23] *See, e.g.*, Martin Fenner, Using Microsoft Word with git, Aug. 25, 2014, http://blog.martinfenner.org/2014/08/25/using-microsoft-word-with-git/ (last visited July 14, 2020) ("This approach is not ideal, as git was written with source code in text format in mind and for example doesn't understand what has changed between two versions of a Word document.").

[24] It is worth noting that Microsoft recently purchased "GitHub," which is the largest and most popular online implementation of Git. *See* Official Microsoft Blog, Microsoft completes GitHub acquisition, Oct. 26, 2018 https://blogs.microsoft.com/blog/2018/10/26/microsoft-completes-github-acquisition/ (last visited July 14, 2020). Perhaps Microsoft has this idea on their product roadmap?

[25] At least one former lawyer who "turned programmer" appears to be working on this problem. *See Should lawyers learn to code? Arguments for and against*, Posting of jandinter to Y Combinator,

*

In closing, I submit that the work of real estate transactional lawyers and software developers shows curious parallels. There is much we can learn from each other, and in my experience, learning coding skills offered philosophical insights, rather than just the ability to "automate things." If you've ever had an urge to learn a little bit of software code, I can attest that it's a journey worth taking.

https://news.ycombinator.com/item?id=20635659 (last visited July 14, 2020) ("[I] now am building a 'GitHub for Word' . . . check out my project Jules . . . ."); *see also* Jules – Collaborate with amazing ease, https://julesdocs.com/ (last visited July 14, 2020).