Learning How To Code Can Unleash New Potential In Lawyers

By **George Zalepa** (March 22, 2021)

George Zalepa

Programming may have once been considered an arcane endeavor, limited to universities and the military before ultimately occupying Silicon Valley with campuses of well-educated (and competitive) programmers.

Now, society's increasing reliance on technology has resulted in the democratization of programming information. Further, the availability of tools and resources, fueled in part by the open-source and free software movements, has reduced financial or institutional barriers to entry.

Lawyers are aware of the increase in client demands and the increase in competition among legal organizations capable of meeting those demands. As a result, legal organizations are increasingly facing challenges in providing top-tier legal advice at competitive prices.

While traditional approaches to reducing margins have so far proven successful, lawyers who are well-versed in technology and, in some scenarios can actively develop software, may contribute to a legal organization's profitability while increasing the quality of their work. Additionally, adopting a technology-centric practice may be implemented incrementally to avoid culture shock between the legal and technology professions.

This article provides a broad overview of how legal organizations may begin to build a technology-centric practice in incremental steps. This building initially can start with overcoming preconceptions and encouraging individual exploration of programming. Those who move forward will start to organize an internal team of programmers, and ultimately separate entities, to support development.

Lawyers Are Well-Suited for Coding

Steve Jobs once said that everyone "should learn to program a computer because it teaches you how to think." While aimed at a general audience, this sentiment is especially salient for lawyers. Many lawyers are already suited for software development because programming uniquely dovetails with how lawyers already think.

Consider the basic features of a contract, such as definitions ("Buyer refers to"), conditions ("If the Payment is less than the Purchase Price"), and operative clauses (e.g., shipping, etc.) that result in actions by the parties.

Now consider a small software application that includes variable definitions (buyer = "Acme Co."), conditional expressions (if payment < price), and functions (function shipGoods). While the syntax differs, the organization and logic applied to a contract are remarkably similar to many of the core features of programming languages and, by implication, software development as a whole.

The largest stumbling block for lawyers can be the preconceived notion that programming is hard and unattainable without formal and extensive training. However, parallels such as the example above illustrate that, conceptually, many concepts involved in programming may already be ingrained from legal education and practice.

Indeed, this training is a significant leg up compared to the average new developer. Moreover, lawyers are problem solvers, which can be a critical feature of a programmer and one that cannot easily be taught.

How to Start

Every lawyer can begin incorporating aspects of software development in their day-to-day practice with little to no changes in their existing tools or workflow. The primary need for this step is a curious mind, one that asks, "How can this be simpler, faster?"

For example, custom auto-correct entries in Microsoft Word can be used to shorten longer, repeated statements into a few memorable characters, similar to "snippets" used in many code editors.

Understanding the basics of regular expressions (similar to wildcard searches) can be used to search for intricate patterns, avoiding a word-by-word review of a document or website. A suitably formatted regular expression can match words having the same root but different suffixes (e.g., /deliver(ies | y | ed | ing | s)*/, matching deliver, delivers, delivered, delivery and delivering).

The ability to inspect the underlying code of a webpage when text is not able to be copied and pasted can potentially avoid manual retyping of the page to paralegals or other support staff.

These examples are purposefully small. An auto-correct entry may only save a few seconds (although some may save nearly half a minute). However, implementing multiple shortcuts throughout a daily workflow can result in a noticeable increase in speed and efficiency.

For example, a piece of canned text is not subject to arbitrary typos. This combination of speed and accuracy may improve an attorney's workflow and requires little to no change in the tools used on a daily basis.

More importantly, these techniques — and similar ones — do not involve programming and thus are more approachable than jumping directly into programming. They are, however, the types of improvements to workflow that have direct analogs to everyday programming and thus help stimulate an appetite for a deeper dive.

The Next Step: Automate the Boring Stuff

To move beyond built-in functionalities of everyday software, at least a cursory understanding of code is useful. Still, however, coding can be done on tools most attorneys already have: Microsoft Office, web browsers and web-based programming environments such as CodePen.io.

An initial step may be to simply gain the skill to find solutions, rather than build such solutions.

For example, management of email into designated folders is a common maintenance function. Usually, this act involves dragging items to the appropriate folder or using the move function of an email client. However, custom scripts can be developed to perform a move to a specific folder and, if desired, perform other operations.

Searching online to find an example of a script that will perform the desired task may be

useful. Knowing how to add this script to an email platform and create toolbar buttons to trigger the scripts can result in a one-click move for any number of destinations. Some lawyers may go deeper and modify found scripts to perform other actions, for e.g., marking the item as read and adding a category.

The above example is illustrative of an initial foray into programming. Many resources already exist and many solutions have already been found. Similar to the first step, understanding how to apply these solutions and then, when necessary, modify them, provides a deeper integration of software in everyday practice and further increases productivity.

The Middle Distance

For most lawyers, the first two steps alone will provide significant productivity increases, and many or most may stop there. For some, a desire to go beyond existing tools and build actual software may be the next logical step.

Certainly, this step requires more time and resource investment to learn the skills — e.g., programming languages, toolkits, developer tools, design theory, etc. — needed to develop software. Resources, primarily in the form of software tools, will likely also need to be installed to facilitate such development.

Fortunately, there are some free or low-cost options to learn how to develop software.

For the budget-conscious, nearly all programming languages, toolkits and developer tools provide their own free tutorials. Third-party sites likewise provide a plethora of online courses, many of which are free to audit. For those willing to invest the time and money, full-fledged boot camps provide ample training to build most internal tools.

Armed with the knowledge of how to build software, many legal problems can be segmented into purely legal questions and those portions that can be automated or improved via software.

For example, tasks that involve repetitive actions — e.g., downloading files, creating directory structures, organizing files, etc. — can generally be automated with a few lines of code, potentially saving time and reducing the likelihood of error.

As another example, projects that require collaboration may benefit from a web application that can handle simultaneous access and editing to work product. In other words, every shared spreadsheet is a potential web application.

Creating a web application to solve a task may seem extreme, but modern open-source projects make creating the skeleton of these applications easier. Indeed, in 2005, Ruby on Rails was originally touted via a 15-minute video where a developer created a blogging website from scratch. Django, in a similar manner, is the self-proclaimed "web framework for perfectionists with deadlines."

Moving beyond personal improvement may also lead to requests from others. A lawyer who develops software to share with a team could receive requests to change the software. The easiest way to demonstrate the value of software is to demonstrate it in action. Such an act may inspire others to follow suit and, possibly, result in a professional software development team within a legal organization.

Legal Organizations as Tech Incubators

Every legal organization handles problems, yet few legal organizations handle truly unique problems. Thus, a solution by one legal organization could likely be used by other legal organizations. When legal organizations develop their own technical solutions, they have created a commodity that can be licensed to other entities.

Common advice to founders-to-be is to first find a problem and then find a solution, as attempting the reverse frequently results in a product with few customers. Legal organizations are in the unique position of being bombarded with problems at all times. Thus, such organizations are constantly presented with opportunities to commoditize software solutions.

A developer, or team of developers, may develop a quick solution to help handle a legal question. This solution may then be tweaked, adjusted and rewritten to handle similar questions, potentially from other clients. Ultimately, the solution will evolve into a general-purpose solution that could be used for similar questions from any client.

At this stage, legal organizations can then spin off the solution into its own business. At the birth of such a business, a clear problem has been identified, a product has been developed and its benefits demonstrated with real data. Few new businesses can rely on such a leg up and legal organizations would be wise to capitalize on this.

Most importantly, most software developed by legal organizations will be developed by lawyers for lawyers. Thus, the software will be developed with near-total domain knowledge of the underlying problem.

The software will inherently be subject to so-called dogfooding: a legal organization's use of its own product, as a way of testing and eventually helping to monetize it. Such an approach, when executed faithfully, could result in best-in-class products having high appeal to all needing the solution.

Conclusion

Not all legal organizations will become tech incubators. Not all lawyers will become expert programmers. The approaches suggested above do not require this. However, as younger attorneys raised with technology enter the workforce, the potential for innovative solutions being developed in legal organizations is higher than ever. The onus is on all of us to realize that potential.

George David Zalepa is of counsel at Greenberg Traurig LLP.