Understanding and Efficiently Resolving Software Development Disputes

By Jonathan B. Frank

Since this article addresses a litigator's approach to resolving software development disputes, it makes sense to start with some background. With apologies to those who may find Wikipedia a superficial (or worse) source of information, custom software is "software that is developed specifically for some specific organization or other user. As such, it can be contrasted with the use of out-of-the-box software packages developed for the mass market, such as commercial off-the-shelf software, or existing free software." The primary advantage is that "it can accommodate that customer's particular preferences and expectations, which may not be the case for commercial off-the-shelf software." Industries that use custom software include construction, health care, education, and retail.

With this background, it should not surprise us to learn that Wikipedia contains this heading: "Major project overruns and failures." The answer to the question "What could possibly go wrong?" is "A lot." To help put this in a more familiar context, let's start with a more common subject matter for litigation: construction. Anyone who has litigated a construction case knows how painfully detailed and trade-specific the factual disputes can be.

Software development litigation is like construction litigation on steroids. The central five issues are the equivalent of:

- "Who told you to move that retaining wall over there?"
- "I didn't think that roof was going to be so expensive."
- "You told me that air-conditioning unit would keep my whole building cool."
- "The foundation is sinking, and my walls are cracking."
- "Why aren't you done already?"

In other words:

- Conflicting directions;
- Confusion about cost and specifications (perhaps the result of changes without change orders);
- Misplaced reliance on the expertise of the performing party;
- Project-threatening quality failures; and
- Unclear milestones and measurement of performance.

As in a construction case, the significant facts will be bitterly contested (luckily, the law will not be). No doubt each side's project management team has met multiple times, pointing fingers but not solving problems. Given the magnitude of the problems, senior management has likely been involved. Not-so-friendly threats have been exchanged. So, the lawyers get called.

What can we do? Even understanding the analogy to construction cases, the first challenge is obvious: There is no wall, roof, air-conditioning unit, or drywall to look at. And while it may be easy to describe and inspect an unfinished building, it may be nearly impossible to "see" whether a software development project has failed or is incomplete. Luckily, clients who have already spent time fighting about the facts should be able to explain them, even to an attorney!

This process, though, highlights the major difficulty we will have in representing our clients. Imagine presenting a software development dispute to a judge, jury, or arbitrator. Think about how much foundational knowledge you would need to gather, then provide to the fact finder, before even addressing the disputed facts (unfortunately, you won't find a handy Wikipedia article). And that only sets the stage for the dueling narratives and expert testimony, likely focused on scrutinizing and analyzing a series of microdecisions, often with cascading impact.

Without being overly pessimistic, there is very little chance that a third-party decision would be reliable and therefore acceptable to the parties. Put simply, the legal system is ill-equipped to deal with a software development dispute. The American Arbitration Association reports that in recent years, the number of technology cases submitted to the AAA has doubled, demonstrating that parties are increasingly opting for more-qualified decision-makers. But that statistic doesn't tell the whole story. Even with the assignment of sophisticated arbitrators, between 50% and 70% of the technology cases scheduled for AAA arbitration settled before the award.

With that in mind, counsel's overriding goal from the outset should be to find an efficient resolution. How? Let's go back to more familiar ground. It turns out that resolving a software development dispute, like a construction case, has a certain cadence and structure. I have followed this path to settle every one of these cases I have handled:

- Understand the "blueprint" and "specs" to identify objective and subjective deliverables. As with a construction project, this will depend on the time and effort the parties dedicated to drafting the contract documents in the first place. Unfortunately, the parties, in the spirit of cooperation, often decide to "deal with that after we get the framing done" or to "build a really nice bathroom."
- Compare performance to the objective criteria by bringing in subject-matter experts from both sides, along with decision-makers, as soon as possible. This may be the most counterintuitive step to most litigators, since it involves complete transparency and dialogue, as well as the flexibility to sit in a room with IT people, hopefully understand them, and let them try to problem-solve, likely over multiple sessions. This isn't as hard as it may seem, though. Remember, problem-solving is usually part of their job description.
- Fairly allocate responsibility for problematic decision-making and quality issues. You may find that, as in a construction project, implicit understandings never became explicit; salespeople got ahead of the designers and engineers; project managers passed the project along without a smooth transition; individual workers may not have been as skilled as advertised. While there has no doubt been considerable finger-pointing, rarely is only one side truly to blame.
- Determine if additional work can solve the problem. Certainly, the relationship has been damaged, but there may be value in repairing it, often with some change of personnel. Beware of the urge to "change horses midstream," since, as in a construction project, bringing in a new IT contractor to "patch a few walls" or "redo the paint" could actually be more disruptive, risky, and expensive than continuing with the old one.
- Develop a structure for financial resolution. The performing party likely has unpaid bills, believing that the contracting party just does not want to pay for the value it has received. The contracting party likely has claims to recover money already paid and money that needs to be spent on repairs/reconstruction, believing that the performing party overpromised and underdelivered. Both sides have serious holes in their budgets and the hope that the litigation system can fill those holes. As with most litigation, however, there should be a way to conduct a thorough, balanced analysis and find "rough justice" that takes into account the time/risk/aggravation/cost of litigation, avoiding the possibility that the budget holes just get deeper and deeper.
- Create a structure to problem-solve through completion of the project. The stakes are generally too high for both sides to run the risk that the solution could be derailed.

Although the subject matter of software development litigation may be complicated, the structure of the dispute and its resolution is not. Following this path will keep software development disputes out of a litigation system that is ill-equipped to resolve them to the satisfaction of either party.

Jonathan B. Frank and his wife, Jan, founded Frank & Frank Law in 2019 to provide practical, efficient solutions for business and real estate disputes. Jonathan is an honors graduate of Stanford University and the University of Michigan Law School. He has authored numerous articles on litigation practice, was recently co-chair of the OCBA's Business Court and Counsel Committee, and is currently a member of the Oakland County Bar Association board of directors. He is also a trained mediator and a neutral arbitrator with the American Arbitration Association. Find more information at frankandfranklaw.com.