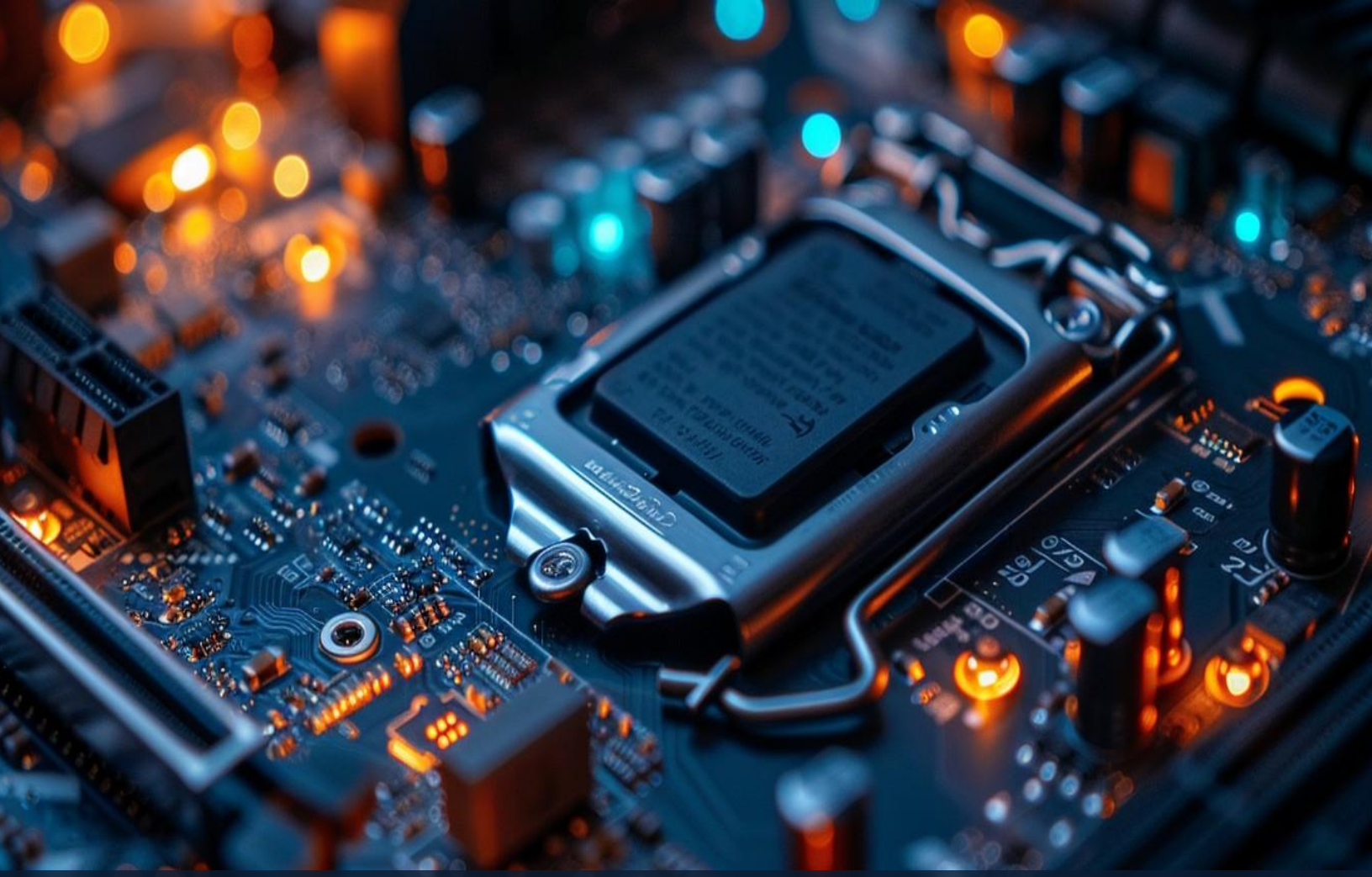


The Basics of Software Development for Attorneys

Key concepts for high-stakes commercial disputes involving software and software development

April 2025
v1.0

DisputeSoft®
Software Dispute Experts

The Basics of Software Development for Attorneys

In today's world, software is a central component of many modern business operations, covering everything from supply chain management to online marketing. Businesses often invest substantial amounts of time and money building or implementing complex software systems in seeking a competitive edge for their operations.

Given its complex and valuable nature, software can be and often is at the center of high-stakes commercial disputes. When such disputes arise, attorneys need to understand numerous technical aspects that might be at issue, including how the software was developed, which kinds of tools were used, and which kinds of systems of record might be available that would shed light on the dispute. This paper provides an overview of these topics and provides context for attorneys handling such matters.

What is Software?

Software or a *computer program* is a set of instructions that tells a computer how to perform different tasks. Software can be anything, from applications like Microsoft Word that are operated directly by users, to behind-the-scenes software like print drivers that send data to your printer.

In most instances, software exists in machine-readable *binary* format. That is, the software is composed of a set of zeros and ones representing instructions to the computer that would be difficult for a human to understand. To understand the software, it is usually necessary to turn to the *source code* used to create the software in the first place.

What is Source Code?

Source code is a human-readable set of instructions for a computer. Source code may be written in one or more *programming languages*, which are structured languages used for creating computer programs (shown in the figure below).

Figure 1: Example Source Code

```
28      // Bulk operations run against the entire set
29      if ( raw ) {
30          fn.call( elems, value );
31          fn = null;
32
33      // ...except when executing function values
34      } else {
35          bulk = fn;
36          fn = function( elem, _key, value ) {
37              return bulk.call( jQuery( elem ), value );
38          };
39      }
```

Shown above is an excerpt of source code written in the JavaScript programming language.

Source: *jQuery JavaScript Library*, https://github.com/jquery/jquery

How is Software Developed?

As an analogy, consider how a construction company builds a new house for a buyer. The process of building the house will be done in several steps, including (1) starting with determining which features should be in the house, (2) creating blueprints, (3) building the designed house, (4) inspecting the house to make sure it is up to code, and (5) turning the keys over to the buyer.

At its heart, *software development* follows a similar set of processes to create working software. In general, key software development processes typically include the following:

- *Requirements elaboration*, in which the specific *requirements* for the software are established, such as specific functions the software must have, constraints, etc. ("determining which features should be in the house");
- *Design*, in which plans are made for how components of the software will be built and integrated together ("creating blueprints");
- *Construction*, in which source code is written in accordance with the planned designs ("building the designed house");
- *Testing*, in which the software is evaluated and subjected to quality control efforts to identify and fix any potential defects ("inspecting the house"); and
- *Implementation*, in which the software is put into *production*, or operational usage by the customer, known also as reaching "go-live" ("turning the keys over to the buyer").

There are different methodologies in the software industry for how these software development processes are carried out. For example, in a *waterfall* approach, these processes are carried out once and in sequential order. Returning to the homebuilding analogy, in a waterfall approach, the construction company would gather all the requirements first, then make all the blueprints, then build the entire house, etc.

In contrast, in an *iterative* approach (*e.g.*, the *Agile* or *Scrum* methodology), the software development processes are carried out multiple times to build the software piece-by-piece. Returning to the homebuilding analogy, in an iterative approach, the construction company might gather requirements for only the kitchen first, then design, build, and inspect the kitchen, etc., before repeating these processes again for each next room to be built, one at a time.

The specific methodology and processes used by a software development company and the manner in which it implements these processes may vary based on the kind of project. For example, if a development firm is creating completely customized software for a client from scratch, it will likely apply all of these processes to build the software from the ground up to meet the client's needs.

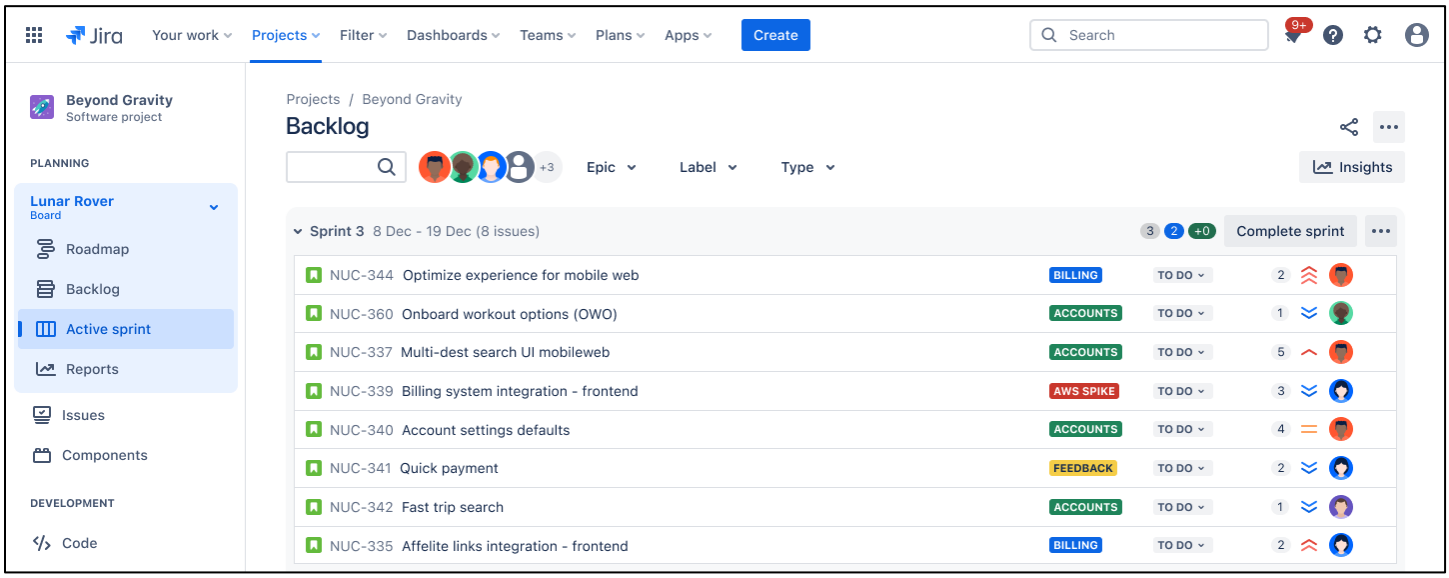
In contrast, if a development firm is implementing *commercial-off-the-shelf* or *COTS* software, some of these processes may not be necessary. For example, if a client has already determined that certain software available on the market meets its needs, it could collaborate with a development firm solely to implement the COTS system in production as-is, eliminating the need for any further requirements gathering.

Software Development Systems of Record

When a software development firm embarks on a software development project, it typically uses various systems of record to track and manage its work on the project. Each system has distinct sets of data that may be relevant in a software dispute.

One important system of record often used by software development firms is a *software project management system*. These systems are often COTS products such as Atlassian Jira (shown in the figure below), Microsoft Azure DevOps, HP Application Lifecycle Management, or IBM DOORS. However, in some situations, these systems may be custom-built applications created by the development firm for its own internal usage.

Figure 2: Example Screen Display from Atlassian's "Jira" Software Project Management System



Shown above is an example screen display from Atlassian's "Jira" software, showing individual tasks, at a summary level, in a project's ongoing backlog of planned work. Each task can be expanded to show additional information about the nature of ongoing work to complete the task.

Source: *Atlassian, "Product Backlog Template" Webpage, https://www.atlassian.com/software/jira/templates/product-backlog-template*

Software project management systems typically store various kinds of records about the development of the software. For example, some systems track work done to meet specific requirements, including information about who worked on each requirement, when the work was performed, issues encountered, etc. Similarly, some systems track detailed testing data, such as specific test plans, when tests were run, the results of those tests, whether defects were detected, whether or when defects were fixed, etc.

Another important system of record typically used in modern software development is a *source code repository*. A source code repository is a type of database that stores a set of source code files and tracks changes made to those files over time. Information tracked in the repository typically includes what changes were made to the files over time, who made the changes, and summary information describing the nature of those changes. Typical source code repository systems include Git (shown in the figure below), Mercurial, Subversion, and Perforce.

Figure 3: Example Data from Git Source Code Repository

Shown above is an example of data typically stored in a "Git" source code repository. Each "commit" represents a distinct set of changes to files in the repository, and contains information such as:

(1) ***Commit hash:*** *an alphanumeric code uniquely identifying the commit;*
(2) ***Tags****: optional labels provided by a developer (typically marking a specific software version);*
(3) ***Author:*** *the developer who created the changes;*
(4) ***Date****: when the changes were made; and*
(5) ***Commit message****: the developer's description of the changes (indented in the example above).*

Source: *jQuery JavaScript Library*, https://github.com/jquery/jquery

Software may be composed of source code stored in one or multiple repositories used together. These source code repositories may be stored and maintained in cloud-based services such as GitHub, Bitbucket, and Apache Subversion, or on a company's corporate computer systems.

As another example of an important system of record, many development firms may use *document* or *content management systems* for their software development projects. Such systems may be COTS products such as Atlassian Confluence or Microsoft SharePoint, or may be simply various files stored on network attached storage devices (*i.e.*, corporate file systems). Many kinds of project files may be stored on such systems, including design documents, meeting notes, project schedules, testing results, system deployment records, etc.

As yet another example of an important system of record, development firms may have distinct *environments*, or sets of computer systems, for a particular project. Distinct environments are often used to test the software at various stages of development. For instance, one environment might be used for unstable, in-progress development work, while another might be used for final tests of stabilized software before it is put into production usage. Each set of computer systems may store data such as various technical records and log files that may be relevant to a dispute.

These environments may exist as *physical machines*, which are usually specialized hardware such as rack-mounted servers installed in a data center (as opposed to desktop computer systems), or as *virtual machines*, which are software-based computers that act like physical computers.

Producing data from physical machines can be as simple as copying off relevant data onto dedicated hard disks, or as complex as performing specialized forensic processing to create a replica of each physical system. Producing virtual machines in their entirety is almost always relatively simple, since all the files composing an entire virtual computer system are typically saved within a few files on a host computer system.

Specific Considerations for Attorneys

When handling a software dispute, it is often a good idea to start by understanding which software development life cycle processes were carried out, and which systems of record were used to track work during each phase of the software's development. This approach can help an attorney to understand what kinds of records might be available, either to request during discovery or to respond to an opposing party's production requests.

However, understanding all the processes and tools used during a software project may not be a trivial affair. For example, a project may have been carried out using a single development methodology for the entirety of the project, or the methodology may have changed during the project for any number of reasons, resulting in a more complex picture of software development events. Engaging an experienced expert early on may be necessary to investigate and understand such complexities.

Similarly, a project may have been managed using one set of tools or project management software for the entirety of the project, or there may have been a switch to an entirely different set of tools mid-way through the project, thereby creating multiple systems of record over time. Once the used systems of record are identified and understood, production can commence from whichever systems of record are relevant to the dispute.

The manner in which systems of record are produced is frequently an important consideration. Several approaches are typically used to produce records from such systems:

- *Exports*, in which a system's built-in functionality is used to create spreadsheets or documents containing requested records;

- *System backups*, which contain a replica of all data extant in a system of record;

- *Direct access*, in which users can be provided direct access to a system of record (typically on a read-only basis); and

- *Native format*, in which files are produced in their original format as used during the regular course of business operations.

The appropriate production approach may vary based on what kind of data is available in a system of record. As an example, consider Amazon CloudWatch, a cloud-based service for recording system logs made by operational software. For this system, export logs may be sufficient as long as the files contain all relevant recorded information.

However, exports are seldom, if ever, sufficient when producing source code repositories. Many analyses, such as determining the exact nature of changes over time to a set of source code files, can only be done on a source code repository when provided direct access to the native files. Conferring with an experienced expert is often necessary to determine which production method will be the most helpful in addressing a dispute.

The appropriate production approach may also vary based on cost considerations. For most of these approaches, technical personnel for a party should be able to produce relevant records without substantial time or expense. For example, creating a database backup for a system of record is generally a straightforward task. More involved tasks, such as creating full forensic replicas of entire computer systems, may require specialized personnel or substantive effort. Such considerations are typically specific to the facts of a dispute.

Attorneys should particularly consider the form in which source code is produced. In many cases, source code is directly exchanged between the parties in encrypted format, while in other cases, source code is only made available for inspection on a review computer in opposing counsel's office.

The manner in which source code is produced may create constraints on how it can be analyzed. For example, a copyright infringement matter may require side-by-side comparisons of two parties' source code files to investigate and assess allegations of potential copying. Such an investigation can be aided by programmatic comparisons tools if both parties' source code files can be exchanged or otherwise made available on a single computer system, as opposed to making each set of source code files available for inspection only on isolated review computers. Specific situations may vary, and are best considered with the assistance of an experienced expert in such matters.

Conclusions

Software development covers a variety of technical tasks that are typically documented in numerous systems of record. Attorneys handling software dispute matters can benefit from an understanding of these concepts and systems of record in considering how to address such disputes.

However, while attorneys can benefit from such an understanding, it should not be considered a substitute for in-depth technical expertise. In any software dispute, attorneys should typically plan to engage an expert who can help address the dispute's technical complexities and can provide analyses and conclusions that can be useful to successfully resolving the dispute.

About DisputeSoft

Founded in 2003, DisputeSoft specializes in providing expert analysis and expert witness services in software project failure disputes and in intellectual property disputes involving software.

Based in the Washington, D.C. metropolitan area, DisputeSoft's experts have been engaged in matters in numerous national and international jurisdictions, including various U.S. state and federal courts, the Court of Federal Claims, the International Trade Commission, and the Patent Trial and Appeal Board, as well as various domestic and international arbitration panels, and in matters in Canada, Australia, Europe, and Asia.

DisputeSoft's experience covers a wide breadth of software failure disputes, particularly those with claims related to late, inadequate, or incomplete delivery and support, as well as extensive code comparisons and analyses of software and systems at issue in copyright infringement, trade secret misappropriation, and patent infringement matters.

DisputeSoft's clients include Big Law firms, boutique practices, and sole practitioners, who represent corporate, government, and individual clients across numerous industries and subject matter domains.

DisputeSoft provides expertise in the following service areas:

- Software Project Failure
- Software Copyright Infringement
- Software Patent Disputes
- Computer Forensic Analysis
- Trade Secrets Analysis
- State & Federal Government Services
- Software Project Management
- Source Code Audits
- Data Analytics
- Data Privacy, Protection, and Security
- Software Project Schedule Analysis
- Software Labor and Cost Analysis
- Internet Misconduct
- Source Code Investigations
- Pre-Litigation Services
- Project Recovery
- Electronic Systems Discovery
- Surety Investigations

About the Authors

Nick Ferrara, EnCE

Mr. Ferrara has approximately 20 years' experience in the information technology field, including consulting work, IT management, staffing, and systems administration. At DisputeSoft, Mr. Ferrara has been an integral part of more than 100 matters, spanning numerous commercial industries and all of DisputeSoft's core practice areas.

Mr. Ferrara has been engaged as an expert in state, national, and international jurisdictions and has testified both in U.S. federal court and before the American Arbitration Association. Mr. Ferrara is also a certified computer forensic examiner, holding the EnCase Certified Examiner (EnCE) certification.

Aparna V. Kaliappan

Ms. Kaliappan has approximately 10 years' experience in the information technology field, including substantial experience in providing technical analysis in software disputes. At DisputeSoft, Ms. Kaliappan has worked on approximately 50 matters involving copyright, trade secrets, data analytics, software project failure, software patents, and computer forensics in various commercial industries.

Ms. Kaliappan has extensive experience in data analytics, including graduate-level work in quantitative data visualization. Ms. Kaliappan also holds multiple Oracle and Microsoft database certifications.

DisputeSoft®
Software Dispute Experts

6116 Executive Boulevard, Suite 330
North Bethesda, MD 20852

Phone: (301) 246-3150
Fax: (240) 465-4442
info@disputesoft.com