Andrew Schulman
Software Litigation Consulting (San Francisco CA)
http://www.softwarelitigationconsulting.com
undoc@sonic.net
Dec. 17, 2013; slightly revised Jan. 29, 2014

Paper for ASU-Arkfeld eDiscovery & Digital Evidence Conference

Computer Software Source Code and eDiscovery

While eDiscovery focuses largely on *data* stored in or generated by computers, there is an additional area whose handling is becoming an essential eDiscovery skill: *code*, that is, the software which computers run in order to create and process data. This article quickly compares and contrasts source code (the most readable type of software evidence) with eDiscovery generally, noting sample cases the reader may consult for more details.

The Sedona Conference's glossary defines software as "Any set of coded instructions (programs) stored on computer-readable media that tells a computer what to do." Software is a form of text, written by humans. It is in some ways not much different from Word documents or emails, except that software text conforms to certain rules (programming languages), and that computers interpret this text as instructions.

What is source code?

All software-based products, devices, and services are built from files containing *source code*, which is text directly written by computer programmers. This is generally transformed into a different type of file, called *object code,* which is what directly instructs the computer in its operation. The relationship between source code and a software product or service (used by consumers or within an enterprise) is somewhat analogous to the relationship between a blueprint and a building. However, there are two important differences between blueprints and source code.

First, while humans consult blueprints in order to construct buildings, software products/services are created *directly* from source code. The source code kept inside a company thus has a direct relationship to its publicly-visible products. Contrast the more indirect relationship of an organization's other internal documents to its public or external behavior.

Second, because so much of modern business, government, and other activity is based on software, source code has wide applicability in litigation. Everything in the modern world which uses software was built in part from source code. Source code embodies de facto policies of organizations (see the book *Code and Other Laws of Cyberspace* by Lawrence Lessig).

While many software products and services are widely accessible (how many million copies of Windows or Facebook or the iPhone iOS operating system exist on the planet right now?), their underlying source code is tightly held (apart from the important exception of so-called "open source") as a proprietary so-called "crown jewel." Litigation in which software is at issue generally focuses on the closely-held source code, rather than on more widely-available object code, in part because the source code is more readily understood by a larger number of people (somewhat shockingly, even many computer programmers have little idea of how to reverse engineering object code).

When is source code relevant and necessary in litigation?

Source code may be relevant in litigation to explain the electronic data which has been produced (see "code book" or "data dictionary" cases); or as "discovery about discovery" (source code may embody a company's de facto policy for document preservation, or logging of otherwise-ephemeral data).

Source code also often has an *intrinsic* relevance, as in a software patent case or other intellectual property litigation involving software copyright or trade secrets. Source code's direct relevance is not limited to IP cases. Source code may also be relevant in antitrust (for example, the Microsoft cases), criminal law (DUI arrestees demanding to review Breathalyzer source code), medical malpractice or products liability (software used in medical devices), securities law (see recent headlines regarding Bernie Madoff's computer programmers), environmental law (models used by the EPA), and even election law (voters demanding audits of the software used in voting machines).

Having noted source code as a location of relevant information in litigation, however, relevance is not the same as necessity. Source code requires an expert or consultant for its interpretation, and the consequent cost of using source code likely demands a greater showing of necessity beyond mere relevance (see e.g. Generac v. Kohler, emphasizing alternates to source code, such as deposing the programmer; courts will particularly favor this when the source code belongs to a third party; and see OpenTV v. Liberate, applying the Zubulake cost/benefit factors to establish which party has the burden to extract relevant source code from a larger collection).

How does source code differ from other electronic discovery?

In many ways, source code shares the characteristics of other electronically stored information (ESI): volume, dispersal, searchability, and so on. But it differs in key ways, briefly described below.

Centralization and dispersal: Source-code productions in discovery are frequently incomplete, with little bad intent. Compared to other ESI, however, source code generally is more centralized, with less good reason for "missing" files (see spoliation below).

Searchability: Source code generally appears in plain-text files, and so is searchable by a wider variety of tools than e.g. a Word document. However, knowing what to search for is a different matter: relevant keywords often appear in source code as part of a single "CamelCase" word (e.g. "GetEmployeePolicyLocation", when looking for "employee policy"), or single punctuated unit (e.g., "employee_policies.get_location"). Most eDiscovery software is ill-suited to quickly finding relevant terms inside other terms. If there were one area of source code where e-discovery and IT personnel could become more familiar, it would be the different method of searching.

Review process: In part because source code seems less searchable by non-programmers, it is often produced during discovery in an unusual way: neither a straight production, nor an on-site inspection, but something in between, a source code "review," similar to a "quick peek" procedure. Source code will be copied onto a secure computer, the other side's expert or examiner will review that copy under a stringent protective order (PO), and will request specific extracts to be printed and Bates stamped (see e.g. early software patent case, Rates Tech. v. Elcotel). The source-code computer will generally be detached from the internet, but must provide the reviewing party with reasonable searching and source-navigation tools (see Big Baboon v. Dell). Such tools include SciTool's "Understand" and

Microsoft Visual Studio (see GeoTag v. Frontier Comm.). There will often be fights over the quantity of source code to be printed (see EPL v. Apple; Digital Reg of Texas v. Adobe).

Verification & spoliation: When source code corresponds to an externally-available product, the source-code production can be compared with this external product. Few other types of ESI have such external verifiability. This not only serves authentication, but also spoliation detection. Source code seems to go missing in many cases. While this is sometimes egregious spoliation (e.g. Keithley v. Homestore.com), it may also reflect the way that the crown jewels have been dispersed (e.g. Windows Media Player source code absent from central "tree" in Microsoft antitrust litigation).

Versions: While many types of ESI have multiple versions, versioning is crucial to source code. Source code is generally kept in a "version control" repository (e.g., Perforce, Git, SVN). The other side will often vaguely ask for "the code" without a clear idea of what code they mean: which specific product/service, version, for which platform. While overbroad requests are common (see e.g. Symantec v. McAfee, seeking all source code for 3.5 years), when there is a publicly-accessible product or service, the requestor could often be quite specific. Sometimes source code for long-defunct, unreleased, or future products is requested (see BigBand v. Imagine).

Intermingling and third parties: Third-party source code may be intermingled in the producing party's repository (see Robotic Parking v. Hoboken, where third party intervenor sought protective order). Conversely, some of the source code relevant to a claim or defense may be held by a third party, who must then be subpoenaed. Discovery rules will be enforced tightly for third parties, even if otherwise not (see Realtime Data v. MetroPCS).

Form & manner of production: In addition to preservation of all available datestamps, it is important that source code be preserved as ordinarily kept. So-called "comments" (descriptions written by programmers) should be not "redacted" (see MSC v. Altair). Whether to produce via a source-control system such as Perforce, or as plain-text files, should be negotiated.

Burden of explanation: The source-code owner sometimes has a burden, besides producing its source code in discovery, to also provide some explanation, "roadmap" or hand-holding to the requesting party (see Leader Tech v. Facebook; LaserDynamics v. Asus). This issue also emerges when a party produces source code rather than directly answer interrogatories (see FRCP 33(d)). Conversely, it is a rare requestor who will rest on the producing party's own explanations (see "trust but verify" in Fleming v. Escort).

To summarize, because software is a form of text, it can in some ways be treated as any other electronic discoverable documents such as emails or spreadsheets. It is another area for litigants to mine for relevant information. However, computer code has several unusual features which make its use in eDiscovery a somewhat distinct skill.