Negotiating Intellectual Property Rights in Software-Related Contracts

By Chris Falkowski

Fast Facts:

Defining software deliverables in terms of component parts allows the intellectual property rights in each component to be addressed independently and distinctly from each other.

Software can involve a variety of distinct types of intellectual property rights including copyrights, trade secrets, utility patents, design patents, and trade dress.

Ownership of property is really just a bundle of rights that are associated with ownership; parties can often get what they want without actually owning an asset.

Contract negotiations can involve a lot of tradeoffs. Issues such as intellectual property (IP) rights can easily become intractable because of a false perception that there is no potential middle ground. By contrast, provisions involving financial values, periods of time, etc., tend to be far easier to resolve because there is a readily apparent ability to "split the baby." If one party is at $100,000 and another is at $150,000, it takes little imagination to propose a potential compromise at $125,000. The ability of the parties to formulate middle-ground positions as potential compromises can be critical to resolving the issues in a negotiation.

Issues that are subject to a split-the-baby approach are considerably easier to resolve than those not perceived in that manner. Unfortunately, IP rights in software contracts are typically thought of as being in the intractable category. IP rights play an increasingly important role in the modern economy. Negotiations involving IP rights can be difficult because both vendors and clients have objectively valid arguments for protecting their core interests in the transaction.

By compartmentalizing the code components and IP rights pertaining to those code components, both parties can facilitate reasonable agreements while avoiding long and intractable negotiations that can potentially have a negative effect on the relationship of the parties going forward.

Dividing Software into Component Parts Makes Reaching Win-Win Agreements Easier

When Ross Perot founded Electronic Data Systems in 1962, the world of software technology looked a lot different than it does today. Back then, information technology (IT) was composed of large centralized mainframe computers running large centralized computer programs. Modern software is designed to be the exact opposite, with highly modular software that can be distributed across diverse and vastly dispersed networks to be accessed by a range of client devices including smartphones, tablet computers, and desktop computers. The modularity of software brings many advantages with it, including reusability. Just as lawyers are likely to begin the process of drafting a legal document by finding an existing document to start from, software engineers often begin the process of creating software by finding a somewhat similar (or at least architecturally suitable) starting point from a library of past work. Reusability brings with it quicker results, enhanced reliability, and cost savings—important factors in virtually any project attempting to use resources efficiently.

The very structure of modularized modern software technology makes it easier to formulate viable compromise positions by distinguishing between components in the software deliverable.

Code Formats—Source Code vs. Object Code

One potential distinction frequently used to parse out IP rights is the contrast between source code (the format in which humans create a software application) and object code (the format in which a computer runs the program after the source code is compiled). Differentiating between source code and object code can make it easier for parties to reach agreements because in most contexts the client is primarily concerned with object code and the vendor is primarily concerned with source code.

IT Architecture—Ancestor Components vs. Application Components

In addition to highly modular programming languages, modern IT approaches to software development also use hierarchical relationships between code components. Most modern IT projects are based on programming languages and software infrastructures that are generally classified as being "object-oriented" to at least some degree. Object-oriented software uses classes of "objects" as reusable building blocks for software development. One class of objects can "inherit" the attributes and functions of another class of objects. Object-oriented design can involve a complex web of relationships between object classes that are numerous generations deep.

In developing software for a client, the vendor is likely to use preexisting generic ancestor object classes as a foundation for building client-specific application objects. The difference between ancestor code and application code is another potential basis for distinguishing code components.

Components Outside the Control of Either Party—Third-Party Components

The trends of complexity and modularization make it increasingly probable that a software deliverable will include at least some components originating from neither the vendor nor the client. Third-party software is sometimes directly licensed by the client from a third-party licensor but is nevertheless a discussion point in the negotiations because such software can be a necessary prerequisite for the software deliverables to function properly. In other instances, the vendor sublicenses the software to the client. Many IT projects will involve some open source components[1] that raise their own unique issues.

Given the likelihood of third-party components being part of the deliverable provided to the client, generic statements about ownership of the deliverable as a whole are almost certain to be incorrect. It is outside the aggregate power of the vendor and client to determine the ownership of a third-party component.

Bringing up the issue of third-party components can be a helpful way to introduce the need to compartmentalize the deliverable into component parts. No matter how aggressive the other party is being in a negotiation, third-party components are often a reality that cannot be avoided or denied.

Code Components—The Different Viewpoints of Vendors and Clients

In a modern software development contract, it is to be expected that the result will include a variety of code component categories. These categories can be formulated from the position of the vendor or the client.

Vendor's View—The Deliverables Are Based on Vendor's IT Toolkit

Any IT vendor worth doing business with is going to possess a fairly large and valuable library of code components developed before the relationship with the client began. These components represent an ongoing investment by the vendor in its business. In many respects, the ongoing expertise of a software vendor is embodied in a toolkit of software components that it builds on from project to project. From the vendor's perspective, code components can be characterized as follows (in order from most important to least important):

(a) Components developed by the vendor outside the scope of the relationship
   i. Pre-existing components
   ii. Components not otherwise developed pursuant to the agreement

(b) Vendor components constituting improvements or derivative works to (a)

(c) New components designed using the expertise of the vendor that do not embody the client's detailed instructions or confidential information

(d) New components designed in accordance with specific instructions by the client or that otherwise embody the client's confidential information

Any demand by a client that limits the vendor's rights to the components identified in (a) or (b) will be extremely difficult for the vendor to accept. Many vendors will want at a minimum to

Any IT vendor worth doing business with is going to possess a fairly large and valuable library of code components developed before the relationship with the client began.

preserve the right to create derivative or alternative versions of (c) components for their future customers. In contrast, a vendor demanding ownership of (d) components should be prepared for a deserved rejection.

Client's View—The Deliverables Are Based on Client's Confidential Information

Any sizable IT project that is tailored to address the ways in which a client conducts business will probably involve the client's confidential information. Many IT projects require clients to open themselves up to vendors in a way that is unusual outside the context of law and accounting firms. From the client's perspective, software deliverables produced pursuant to an IT project embody the confidential information and business practices of the client. Code components can be characterized as follows (in order from most important to least important):

(a) Components specifically developed for the core purposes of the project
  - i. Components embodying the client's confidential information
  - ii. Components designed on the basis of the client's business practices
  - iii. Components designed on the basis of the client's specific instructions

(b) Components constituting improvements or derivative works to (a)

(c) Infrastructure components that would need to be modified to change the components in (a)

(d) Infrastructure components needed to run the components in (a) and (b)

Any demand by a vendor that exposes (a) components to third parties is going to be difficult for a client to accept, even if there is a binding nondisclosure agreement in place that would prevent the disclosure of confidential information as part of such a transaction. The closer a component is to the business domain of the problem being solved by the vendor, the more likely the client will want to own the component or limit the vendor's rights.

Clients are typically most concerned with the business-domain aspects of an IT project and vendors are typically most concerned with the IT infrastructure aspects. This distinction mirrors the contrast between ancestor components and application components discussed previously.

Software Can Involve a Variety of Intellectual Property Rights

If a purported asset does not fall within at least one specific category of intellectual property, then it is part of the public domain and can be freely used by anyone. There is no such thing as generic or general IP. If a software application or some other type of deliverable provided by IT professionals involves intellectual property, it is because what is produced embodies one or more specific types of intellectual property.

The types of intellectual property that can be relevant to software-related deliverables are typically copyrights, trade secrets, patents, and trade dress.[2] Each of these rights is potentially distinct from the other rights.

Copyrights

Copyright law can protect creative expression but not ideas, facts, functions, or structure.[3] In the context of software, distinct copyrights can exist with respect to source code, object code, and the look and feel of software as it is used by humans.

Trade Secrets

Trade secrets can cover any information that derives economic value from not being known so long as it is subject to reasonable efforts to maintain its secrecy.[4] Trade secrets are the only form of

intellectual property that can cover raw data. It would not be unusual for software prepared by the vendor for a client to include some trade secrets of both the vendor and client.

Utility Patents

A utility patent can protect the functionality or structure of an invention.[5] Utility patents are what most people think of when they think of patents. There are many different utility patents that relate to software and software-enabled systems. Unlike copyrights, trade secrets, and trade dress, which can exist without any governmental application or registration process, obtaining a patent requires submitting an application to the United States Patent and Trademark Office.

Design Patents

A design patent protects the aesthetic attributes of a product.[6] The visual appearance of a software application can be the subject matter of design patents. In the context of software, design patent protection can overlap with trade dress and look-and-feel protection under copyright law.

Trade Dress

Trade dress is a variant of trademark law in which the appearance of a product and not a mark serves as a source identifier.[7] Trade dress in the context of software typically relates to the look and feel of a user interface (i.e., screen) and can overlap significantly with protection of a copyright or design patent. Unlike copyrights or design patents, trade dress protection is based on a perceived relationship between the appearance and source of a product.

Ownership is Really Just a Bundle of Rights

"Ownership" of an asset is really just a bundle of rights associated with the asset. The various rights related to ownership can be parsed and compartmentalized in the same way that the components of a software deliverable can be parsed and compartmentalized. Ownership of an asset is traditionally connected with the rights to use the asset, sell the asset, earn income from the asset, and enforce property rights in the asset. When it comes to negotiations involving intellectual property rights, it is often helpful to look past the label of ownership and instead address the specific concerns of the other party.

Joint Ownership

One approach that can be used to address issues over who holds title to an asset is to have both parties hold title jointly. Joint ownership can be implemented in a structured way through some type of joint ownership agreement or the parties can simply co-own an asset without any specific contractual obligations to each other. Unstructured joint ownership can make the enforcement of IP rights against third parties more difficult. It is also worth noting that jointly owned copyrights include a duty among the co-owners to account for income (i.e., to split the proceeds) derived from the asset, while no such obligation exists with respect to patents.

Sole Ownership by One Party, But the Other Party Gets a License

Since title of an asset is really just a bundle of rights, it is often possible for both parties in a transaction to get what they truly want even if one of the parties has title and the other party merely obtains a license. A perpetual royalty-free license to use, distribute, modify, and use and distribute modifications is from most operational perspectives equivalent to ownership. A list of potential options is provided below:

- License for internal use only
- License to use (no internal use restriction)
- License to distribute
- License to modify
- License to do one or more of the above

Time limitations, territory limitations, royalty provisions, market segment restrictions, and other related provisions can be used to modify any of the options listed above.

Conclusion

Contract negotiations can be challenging for clients and attorneys alike. The complexity of IT coupled with the complexity of the different types of IP rights can prove to be significant obstacles on the path toward reaching an agreement. However, those same complexities can also provide a way to create the type of middle ground necessary for negotiating a win-win transaction. ■

Chris Falkowski is a solo practitioner at Falkowski PLLC focusing primarily on intellectual property and information technology matters. He is a former chairperson of the SBM Information Technology Law Section and a former editor of the Michigan Computer Lawyer. *Before attending law school, he worked as a software developer. Mr. Falkowski is licensed to practice before the USPTO.*

FOOTNOTES

1. See Falkowski, *Open source licensing: An innovation in contract drafting,* 84 Mich B J 30–35 (May 2005), available at <http://www.michbar.org/journal/pdf/pdf4article864.pdf> (accessed July 18, 2012).
2. Falkowski, *Protecting software: The case for software patents,* 86 Mich B J 24–28 (June 2007), available at <http://www.michbar.org/journal/pdf/pdf4article1164.pdf> (accessed July 18, 2012).
3. See 17 USC 102.
4. MCL 445.1902(d).
5. See 35 USC 101 *et seq.*
6. See 35 USC 171 through 173.
7. See 15 USC 1125(a).