SO GOOGLE BEAT ORACLE AT SCOTUS: WHAT DOES THAT MEAN FOR MY SOFTWARE IP STRATEGY?

So Google Beat Oracle at SCOTUS: What Does That Mean for My Software IP Strategy

On April 5th, the U.S. Supreme Court handed Google a victory in its prolonged dispute with Oracle. The dispute relates to Google's use of Oracle's Java Standard Edition (SE) application programming interface (API) in its Android platform. Much has already been written about the merits of the decision, but the larger question is what to do about it? Below are practical takeaways for software developers and technology companies.

1. **The decision did <u>not</u> eliminate U.S. copyright protection for software.**

   U.S. copyright protection for software is explicitly provided for by statute, with 17 U.S.C. § 101 addressing software in terms of a computer program. A computer program is defined as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result." Subject to certain limitations, original works of authorship that are fixed in a tangible medium of expression, including software, remain eligible for copyright. This decision avoids the question of copyrightability. It instead addresses whether Google's copying was fair use. As described in further detail below, the Court found that Google's copying of specific portions of the Java SE API was fair use, thereby allowing Google to avoid what otherwise might be considered copyright infringement.

2. **The copyright-limiting, fair-use aspects of the decision specifically relate to how functions are called in software libraries and APIs. Other aspects of software copyrights are mostly unimpacted.**

   The Java SE API provided a programmer-facing interface for use by software developers. Accordingly, the Java SE API provided an organized way to call a set of reusable software functions. In its analysis, the Court separated the Java SE API into *declaring code* and *implementing code*. As generally treated by the Court, the *declaring code* is the organized way in which the software functions are called or invoked. The *implementing code*, in contrast, is the code that performs the software functions when they are called. Google had copied large portions of the Java SE API's *declaring code* due to widespread developer familiarity with the code. However, Google had written its own *implementing code*. After a fact-intensive analysis using four statutory factors, the Court concluded that Google's copying of the Java SE API's *declaring code* was fair use and thus did not constitute copyright infringement.

Significantly, the Court makes it clear that *implementing code* likely would have been treated differently in its fact-intensive analysis. In other words, if Google had copied the *implementing code* of the Java SE API, it is unlikely that such copying would have been considered fair use. Other types of software code, whether source or object code, should be similarly unimpacted by the decision as long as such code is not analogous to *declaring code* in a programmer-facing interface.

3. **Copyrights are still an important part of a comprehensive software intellectual property (IP) strategy.**

   As noted above, the Court's decision is largely limited in scope to the copying of *declaring code* in a programmer-facing interface. Since software copyrights have broader applicability than *declaring code*, copyrights can still provide protection against, for example, literal copying. Additionally, the Court's fair-use determination was, in part, dependent upon case-specific facts. Therefore, there may exist facts under which the copying of *declaring code* would not be considered fair use.

4. **A comprehensive software IP strategy should diversify beyond copyrights. Consider and choose the right combination of IP tools.**

   Different types of intellectual property are appropriate for protecting different aspects of software. For example:

   - Design patents can protect graphical user interfaces, for example, subject to novelty, non-obviousness, and other statutory requirements.
   - Utility patents can protect software functionality, for example, subject to novelty, non-obviousness, and other statutory requirements.
   - Trademarks can protect an identifier of a source of software-related goods or services.
   - Trade-secret protection may be appropriate for information that has commercial value, so long as reasonable steps are taken to keep the information confidential.
   - Agreements may be used to protect confidentiality and to control, for example, how software is licensed and used.

5. **An existing best practice remains important. Lessen the opportunity for software code to be copied. Limit disclosure and access.**

   Protect software code from disclosure to the extent feasible, for example, via nondisclosure or other agreements or by not disclosing at all. In recognition that copyright protection for *declaring code* in programmer-facing interfaces may be thin, be strategic about the functions that are included in such interfaces. Consider not including, or hiding in internal interfaces, strategic arrangements of functions that do not need to be exposed. If competitor copying of *declaring code* is of particular concern, consider other creative ways copying might be deterred or made ineffective.

6. **Remember that software licenses, including open-source licenses, are based upon an underlying copyright. Consider this decision as you craft your licensing strategy.**

   Whichever side of a license you may be on, consider the software code at issue. For example, as a prospective licensor, you may want to contractually limit a licensee's ability to copy and use *declaring code*. As a prospective licensee, if *declaring code* is involved, you may want to use the principles of the decision to negotiate a lower licensing cost.

7. **This decision is <u>not</u> a blanket license to freely copy software code.**

   As noted above, software code remains copyrightable. Software code can also implicate numerous other types of IP. Any decision to copy software code introduces substantial risk and requires appropriate due diligence.