Software IP Analysis Project

(A Two-pronged Technical Expert Exercise)

In this project, you will have a chance to apply and reinforce the knowledge that you've gained from our lectures on software-based intellectual property. As a computing student, this assignment will also give you practice at an activity that is highly important in a CS curriculum: reviewing professional grade code.

It involves evaluating the code of third-party software that is used by many professionals working for some of the world's largest companies. Our hope is that upon successful completion of this project, you will have new confidence when presented with large code bases that you are to interpret, debug, and/or extend.

The Scenario

There are two companies (A and B) involved. Company A has received a cease-desist letter from company B. The letter is non-specific and simply states that:

> *(1) company A's flagship software ("X") infringes on the intellectual property of one of company B's product ("Y");*
>
> *(2) company A must stop marketing and selling X immediately or face a lawsuit.*

Given that software X is the main revenue source for company A, option 2 (above) is not viable. Legal team A is now weighing alternate responses. One is a letter denying any infringement of Y. Another is to request a licensing agreement with company B. There are other options like redesigning and rebuilding any potentially infringing parts of software X.

The fundamental question that legal team A needs to answer is: what might any infringements stem from?

After going back and forth with their engineers for background information, the attorneys have decided to get an external perspective to minimize bias. For that reason, they have hired you (an independent technical expert) for a set of opinions about software X in relation to Y.

Your Task

Specifically, your job is to analyze the following elements of software X and Y:

1. User-interface (UI, appearance)
2. Software design (Structure of program on a file/folder level and relation between parts of program, design choices)
3. Algorithms (Process of code execution, or typical algorithms like DFS, BFS, etc. that may occur)
4. APIs (Internal and/or external)
5. Source code (Similarity in direct code, libraries used, etc.)

Note that **you can explore and interpret different ideas** amongst these main elements; the examples are given due to confusion in past semesters over sections. There may also be some overlaps between the sections (ex: there is some blurriness between, say, source code and algorithms), which is OK.

You will then generate a report that identifies potential areas of concern that may be raised in litigation. Specifically, the report should:

- **Describe any similarities and differences** that exist in the above areas
- **Assign your own risk level and rationale.** As an independent party, you should determine your own approach to quantifying or qualifying risk.
    - With that being said your analysis should be done while taking into consideration the IP mentioned in class, like copyrights, patents, etc.
    - You should also **cite any precedence from prior cases** as part of this process.

Remember that you are not supposed to be biased towards your employer-- i.e., seeking to promote the idea of little to no infringement just because they are paying you. You are not designated as a testifying expert and you are operating at the direction of counsel so your work cannot be discovered by the other side. Thus, you can and must be completely honest about your client's position.

Also, note that some aspects of the two software programs might be very different while others are not. Where necessary in your report, provide screenshots of the user-interface, diagrams of the software architecture, and other potential exhibits. These artifacts could be created by you and/or found on the internet.

The Software

While this is a fairly open-ended activity (on purpose), we have some constraints due to this being a simulation within a university course. The constraints are as follows:

1. The softwares X and Y you will be analyzing are Integrated Development Environments (IDEs) written in Java or Python, depending on what you choose.
    a. A non-trivial part of your task is finding the code bases of the IDEs you choose – just use the latest version of each IDE's code base that you can find, and link it on your final report.
    b. With the code bases of these programs being available on the internet, we cannot realistically consider trade secret theft as an option for infringement. To keep this assignment interesting, however, pretend that the source code was not downloadable, and moreover, reasonable efforts were made to maintain the source codes' secrecy. You can imagine that you were simply given a laptop containing the respective files of the two (or three) software programs to examine.
2. You will ONLY have to analyze the Debugger sections of the code bases. This includes any code relating to things like:
    a. breakpoints, stepping, error output, etc.

To elaborate on 1, for the primary language of the softwares you will analyze, we have two options: Java or Python. We encourage you to look at the Python and Java IDEs before choosing which IDEs to analyze. **Note that you only have to choose ONE IDE pair to analyze.** Meaning you can compare two Python IDEs against each other OR two Java IDEs against each other. **DO NOT JUST** analyze the one Python IDE and one Java IDE.

IDE Option 1: Java

Your options for software X and Y are the following IDEs written in Java. We have supplied three potential IDEs to compare. **Pick just two** for your evaluation. It doesn't really matter which is X and which is Y since you must avoid being subjective.

a. Eclipse (by IBM)
b. Netbeans (by Oracle)
c. IntelliJ (by JetBrains)

IDE Option 2: Python

Your options for software X and Y are the following IDEs written in Python. We have supplied three potential IDEs to compare. **Pick just two** for your evaluation. It doesn't really matter which is X and which is Y since you must avoid being subjective.

a. IDLE
b. Eric
c. Spyder

Extra Credit Options

A maximum of 20 additional points is possible. Your EC options are as follows:

- Choose one of the following additional tasks to do:
  - Compare the first 2 IDEs to the 3rd IDE of the **same** language
  - Compare the first 2 IDEs to one of the IDEs in the **other** language
- For either language, EC may be awarded for exceptional effort or rigor of analysis.

Example Combinations

- Bob chooses to analyze IDLE and Eric. For EC, he compares the UI of IDLE and Eric against the UI of Eclipse.
- Annie chooses to analyze Eclipse and Netbeans. For EC, she compares multiple elements of Eclipse and Netbeans against IntelliJ.
- Throckmorton chooses to analyze Eclipse and Netbeans. For EC, he compares the algorithms used in Eclipse and Netbeans against those in IDLE.
- Sam chooses to analyze Eclipse and Intellij. For EC, Sam compares multiple elements of Eclipse and IntelliJ against IDLE.
- etc.

The TLDR/Summary

You need to analyze the debugging portions of two IDEs (either 2 Python or 2 Java) on the following elements:

1. User-interface (UI, appearance)
2. Software design (Structure of program on a file/folder level and relation between parts of program, design choices)
3. Algorithms (Process of code execution, or typical algorithms like DFS, BFS, etc. that may occur)
4. APIs (Internal and/or external)
5. Source code (Similarity in direct code, libraries used, etc.)

For each of the above categories, you need to assign your own risk level and rationale. When possible/relevant, please tie in cases from class into your analysis.

The code base choices are as follows (choose from a) or b)):

a. Java (choose 2 to compare)
   - Eclipse
   - Netbeans
   - IntelliJ

b. Python (choose 2 to compare)

- IDLE
- Eric
- Spyder

You can get EC as follows:

- Choose one of the following additional tasks to do:
  - Compare the first 2 IDEs to the 3rd IDE of the **same** language
  - Compare the first 2 IDEs to one of the IDEs in the **other** language
- For either language, EC may be awarded for exceptional effort or rigor of analysis.

General Tips

- READ THROUGH THE WHOLE ASSIGNMENT BEFORE STARTING!
- Please do not **just** analyze UI. This is not sufficient; you should look at and compare code also.
- Remember to focus on the debugging section of the code! While you are able to go into other sections if you really want (perhaps for EC), we will primarily be looking at the debugging sections.
- Make sure your report is **readable to a non-technical audience**. You could even ask a non-technical friend to read parts and see if it makes sense.
  - You can be more technical in certain parts, but generally, your writing should be understandable to people without a technical degree, e.g., most attorneys.
- Feel free to brush up on relevant debugging/programming language prior to starting the project. It may help you feel less lost.
- Try and incorporate relevant case law when you can – this will also give you more of a basis for when you assign your **risk levels**.
- Try and explain your work. It is not sufficient to just say "There were no similarities." If this is the case, what were some of the differences?
- Adding PICTURES and figures will likely be helpful. This may be somewhat more technical in nature.
- Make your sections clear.

FAQ

- Does it matter which company is Company A or Company B/which company I was hired by?
    - No. Those details are mostly given to compare with real life.
- Is there a length requirement/limit? How in-depth should my sections be?
    - The assignment is intended to be open-ended. With that being said, we want a good-faith, thorough attempt at an analysis. However, don't feel compelled to write us a 30-page book. There is no real length limit/requirement. We are mostly looking for genuine efforts at analyzing the code for similarities/differences and applying relevant legal cases to support your decisions when possible.
- Do I need to look at a particular portion of the code?
    - Yes, in the sense that you should focus on debugging portions. But you are allowed to look in whatever folders/subfolders you want to find these.
- I'm lost, please help me.
    - Don't panic – while students have had difficulties in the beginning, almost everyone figures it out in the end. If you are truly lost, here are some suggestions for ways to start:
        - Familiarize yourself with the code bases. Find the source code for the debugging aspects.
        - Do background research on debugging – this may help you understand what to look for.
        - Look over precedent software cases to inform the aspects you may be looking for.

Further Help

Feel free to post on Ed if you need help or guidance. We've run a version of this exercise for multiple semesters with undergraduates and have incrementally refined things based on such questions. As in the Acquisitions Paper, please make any requests for feedback as specific as possible. Best of luck!