

2009

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



MOVING
QUALITY
FORWARD

OCTOBER 27-28, 2009

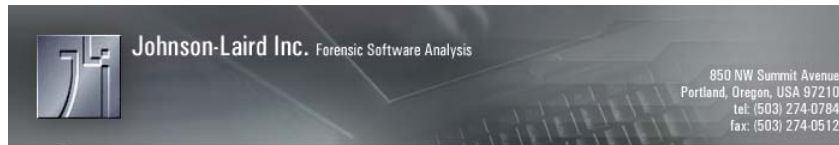
Conference Paper Excerpt
From the

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

Software Pedigree Analysis: Trust but Verify

By
Susan Courtney (susan@jli.com),
Barbara Frederiksen-Cross (barb@jli.com), and
Marc Visnick (marc@jli.com).



Abstract

As forensic software analysts, we are routinely called upon to review source code in the context of litigation or internal software audits. In our experience, it is extremely common for software developers to write software that uses or references third-party materials. These references may include source code incorporated directly into a program, source code routines that are statically linked as a part of the software, or the use of binary libraries that are dynamically referenced at the time a program is actually run. Although the inclusion of third-party materials appears prevalent in modern software development, in our experience, few companies fully document such usage and its associated licensing restrictions. This failure may expose these companies to unexpected legal liability. As a result, we conclude that quality software is defined not just by technical measurements, but also by the presence of a well-documented *pedigree* that itemizes all known use of third-party software and documents the license terms under which such materials may be used.

Quality pedigrees are established through prophylactic procedures, not reactive panic. “Best practices” include 1) carefully considered guidelines for using third-party materials, 2) thorough documentation of such use, and 3) the periodic performance of a forensic code audit, a procedure that uses a combination of automated analysis tools and visual code inspection to detect the presence of third-party material in a body of source code. Forensic code audits serve to demonstrate compliance with usage and documentation guidelines, or to identify any third-party usage which falls outside such guidelines. The combined use of these three practices creates a quality software pedigree that helps to manage the potential risks associated with use of third-party code and may avert unforeseen legal entanglements. A company using this type of program is better positioned to act quickly in the event of a lawsuit, a licensing problem, or to support efficient due diligence done in the context of a merger or acquisition.

1 INTRODUCTION

During our professional careers, we have often observed software development from a vantage point that focuses on failures which may lead to litigation. Although these failures may take many forms, one of the most common is a dispute over ownership of the intellectual property embodied by software. A substantial number of these disputes focus on whether the software in question is derived from, or makes inappropriate use of, third-party materials. These materials may include, for example, source code examples incorporated directly into the comments of a program, source code routines that are statically linked into a program, or binary libraries that are dynamically referenced when a program is actually run. We have seen many cases where, during the course of due diligence or during the process of litigation, management is surprised to learn that their software violates license terms or copyrights of a third-party. This unfortunate situation often results from a combination of unsafe software development practices and a poor understanding of copyright or license conditions associated with third-party materials. As the custodians of software quality, quality assurance engineers are uniquely qualified to help mitigate this risk by enforcing quality processes that document software origins and protect the development effort from improper use of third-party materials.

In this paper we shall introduce the concept of software pedigree analysis by first discussing software ancestry, the legal risks that are attached to the use of third-party software, and the implications of current regulatory reporting requirements. We will then present best practices for establishing a quality pedigree and the concepts comprising a forensic code audit. We conclude the paper with some observations and lessons learned from performing hundreds of forensic code audits.

2 SOFTWARE QUALITY INCLUDES SOFTWARE PEDIGREE

2.1 What is pedigree?

Ponder these questions: If you were facing a lawsuit, or if your company was in the process of being acquired, could you demonstrate that you had implemented a set of “best practices” to guide your developers in using third-party material? Could you demonstrate that you documented all known instances of third-party material in your software, and that you complied with the terms of any licenses attached to that third-party material?

If you have your own staff of software developers, you might assume that these questions are simple to answer. That assumption may prove false if your developers have relied upon or incorporated any form of third-party software during development. In the event that they have, you must consider whether it is possible to identify all such third-party code, understand associated licensing restrictions (if any), and be able to demonstrate compliance with license terms.

If the origin of your software is called into question, can you produce records that prove when your code was designed and developed? Do those records identify all of the third-party code your developers used during the development process? Have you preserved a copy of each third-party license, and do your developers understand any restrictions placed on the use or re-use of third-party material? If you cannot authoritatively answer these kinds of questions about

your software, then you have failed to establish a pedigree sufficient to document your software's ancestry, and you may be putting yourself in legal jeopardy (or in an acquisition scenario, setting up your company or product for a reduced valuation).

2.2 Why should your company care?

While the tendency of developers to share and reuse computer source code is not new, the Internet and the proliferation of software distributed under open source licenses have made such sharing both easier and, from the legal perspective, more complex. In the authors' experience, it is common for developers to reference or incorporate third-party source code, giving little or no consideration to the potential legal consequences flowing from such use. In many cases, no formal record is retained to document the origin, use, or license terms associated with that code.

The failure to preserve such records can create a minefield of potential risks, legal and otherwise, possibly affecting both the software's valuation and your company's reputation.

Much of this risk comes from license terms that are attached to the use of third-party software. In particular, software that is distributed as *free*¹ or *open source*² software may be subject to a wide range of different use restrictions, whether under a *copyleft*-type³ license like the GNU General Public License (GPL) or the Mozilla Public License, or under *attribution*-type licenses such as the BSD license.⁴ Different licensing models place various conditions on both the current and *future* use of open source software. Any software that is derived from software under an open source license *inherits* these conditions; in turn, the licensing conditions are passed on to all subsequent generations of software derived from that software.

Open source software's potentially complex paternity suggests that it may pose a higher risk for certain types of legal exposure. Open source software is usually developed collaboratively via contributions from many different developers, and there is no sure way to guarantee that the contributors actually own the rights to material they are contributing. Further,

¹ Free is a matter of liberty, not price. The Free Software Foundation (FSF) offers the following definition of free software: "Free software is software that comes with permission for anyone to use, copy, and distribute, either verbatim or with modifications, either gratis or for a fee. In particular, this means that source code must be available."

²"Open source software (OSS) is defined as computer software for which the source code and certain other rights normally reserved for copyright holders are provided under a software license that meets the Open source Definition or that is in the public domain. This permits users to use, change, and improve the software, and to redistribute it in modified or unmodified forms." (http://en.wikipedia.org/wiki/Open_source_software visited June 25, 2009)

³ Copyleft is a play on the word copyright; it describes the practice of using copyright law to remove restrictions on distributing copies and modified versions of a work for others and requiring that the same freedoms be preserved in modified versions. A primary restriction of most copyleft licenses is that the user who incorporates copyleft materials must make available the source code of any derivative work they produce, under the same terms as the parent material's license.

⁴ There are now many distinct forms of open source licenses. The specific licenses mentioned in this paper are all available on the Internet. For example, the GNU General Public license (GPL) can be found at <http://www.opensource.org/licenses/gpl-license.html>. The GNU Library Public License (LGPL) can be found at <http://www.opensource.org/licenses/lgpl-license.html>. The BSD license can be found at <http://www.opensource.org/licenses/bsd-license.html>. The Mozilla Public License can be found at <http://www.mozilla.org/MPL/>. The Q Public License can be found at <http://www.opensource.org/licenses/qtpl.php>. The Apple Public Source License (APSL) can be found at <http://www.opensource.apple.com/license/apsl>. (All sites visited June 26, 2009.)

because of the dynamic nature of open source development, contributions that infringe a third-party copyright or patent are easily propagated and may contaminate multiple discrete software products before the infringement comes to light.

Code published under an open source license is not the only third-party code that your developers may have used without a full understanding of the license restrictions associated with that code. In some cases, developers maintain extensive collections of the programs they have written over the course of their careers and re-use handy bits of code over and over again when confronted with similar problems to solve. Ponder the ramifications of such re-use in the context of a consultant or employee who has worked in the past for one of your competitors. Here, the developer's decision to re-use portions of his prior work product may lead to copyright, patent, or trade secret litigation, even if you are unaware of the use of this material. In the authors' experience, this is an all-too-common scenario. The key to avoiding these pitfalls is a combination of education and internal control processes.

2.3 Managing the risk

Developers must be educated about the potential pitfalls of third-party code use.⁵ They should be given clear guidelines with respect to the requirements for recording the provenance and license terms of any code they contemplate using. A code review process should not only evaluate code quality but should also determine whether license restrictions are acceptable and consistent with corporate policies.

Additionally, companies should adopt a clear set of procedures that prohibit the use of materials which may have been produced for past employers, including source code, proprietary documents, and any other confidential information. Further, companies should provide developers with specific guidelines about *outbound* code, including contributions to open source initiatives, trade publications, and any other situation where a developer might contribute company-owned software or materials to a third party.

A company should communicate these guidelines and procedures to new developers and contractors during the indoctrination process and then periodically thereafter. A company should also ensure that during exit interviews developers and contractors are asked to turn over all company source code, and instructed that all code developed during their now-ending tenure is the intellectual property of the company and that it must not be used elsewhere.⁶

Companies may want to consider performing periodic software audits to make certain that corporate guidelines with respect to third-party materials are being followed. In many situations these audits can be incorporated into existing quality assurance processes. Audits by an outside party may be required in the context of litigation or acquisitions. In the event that undocumented (or unacceptable) materials are found, due diligence may require additional research and potential remedial action.⁷

⁵ This also applies to commercially licensed third-party material. We have observed numerous instances in which a body of code being purchased by a company contains materials under a third-party commercial license that is not readily transferable to the purchasing company. We have also observed instances where a programmer used a demo version of a commercially licensed product, without then obtaining a paid-up license for the product.

⁶ This restriction assumes that your employment or engagement agreements actively assert ownership of your employees' and consultants' work product.

⁷ Remediation can take a number of forms, such as removal of problematic material; complying with the terms of a third-party license (for example, providing appropriate attribution); or code refactoring (rewriting), possibly in the context of an independent software development project ("software cleanroom").

2.4 Source Code and SOX

Many companies, especially technology companies, categorize their source code as intellectual property (“IP”) on their financial reports. In 2002, in response to financial reporting abuses by major corporations, the Sarbanes-Oxley Act (“SOX”) was enacted by Congress of the United States of America. It placed into law new rules governing corporate accounting for public companies and enacted tougher ethical standards, including criminal penalties, for corporate officers for SOX violations. Intellectual property is not directly mentioned in SOX, but “part of the general requirements that come out of it are that companies are expected to monitor and disclose IP events, including the relationship between their intellectual property position and their financial performance.”⁸

For many companies, IP is a crucial part of its mission. However, the risks associated with IP are increasingly significant for all public companies, not just public technology companies. SOX is applicable to *any* assets that have a material impact on the financial condition of a public company, including material third-party IP encumbrances which may affect the ability of the company to conduct business.

SOX also requires corporate directors to certify that the company complies with all laws and contracts, including software licenses. Violating software license terms therefore places corporate executives at risk for personal liability. In this context, open source can be particularly problematic, since developers have access to open source materials outside normal software acquisition processes and may therefore incorporate open source into a company’s software without any review of its license terms.⁹

In light of SOX legislation, public companies are well served by establishing policies and procedures to bring structure and consistency to IP management. This includes identifying and assessing the materiality of IP assets. Further, companies should educate employees on the importance of IP assets, the procedures used for IP controls, and the importance of reporting to management material events related to IP assets.

3 HOW TO ESTABLISH PEDIGREE FOR A CODE BASE

3.1 Software Pedigree Analysis

Software pedigree is best established through use of systematic procedures and policies. “Best practices” include 1) carefully considered guidelines for using third-party materials, 2) thorough documentation of such use, and 3) the periodic performance of a *forensic code audit*. Forensic code audits help to ensure compliance with the corporate usage and documentation guidelines and to identify any unanticipated third-party use. The combined use of these three practices establishes a quality software pedigree. The maintenance of your software’s pedigree helps to manage the risk of unforeseen legal entanglements. A company using this type of

⁸ Quote from Leo Longauer in Barraclough, Emma “What Sarbanes-Oxley means for you” Managing Intellectual Property, 1 May 2008, available at <http://www.managingip.com/Article/1933503/What-Sarbanes-Oxley-means-for-you.html> [visited 6/25/2009].

⁹ “Taming The Open Source Monster” Open Source, 1 June 2006, available at <http://www.risk.net/public/showPage.html?page=331247> [visited 7/23/2009].

process is better positioned to react quickly in the event of a lawsuit, a licensing problem, or to support efficient due diligence in the context of a merger, acquisition, or divestiture scenario.¹⁰

Software development “best practices” should include an approval process for the evaluation of third-party materials, as well as the review of supporting documentation regarding the origin of and license restrictions on those materials, before the materials are approved for use. A copy of the actual license text should be a required document for the review process and appropriately preserved for future reference.

Software pedigree documentation should be formalized and produced as an artifact of new development and updated during the on-going maintenance process. Until it is a common practice, it may be advisable to perform periodic compliance audits for this documentation. Developers (and managers) must be instructed to keep clear records related to the use of third-party code. These records should include a clear identification of the third-party code, including its specific version (if available), where it came from, how it is used, and what license terms apply to it. Additionally, the records should document which components incorporate the code and which components interact with it. Consider maintaining a special repository for approved third-party code and associated licenses and documentation, in order to preserve a clear record of the original third-party code base.

One point cannot be overstated: developers must be explicitly instructed to never reference or incorporate any code from their private libraries that was developed while working for past employers. Consider adding this policy to your company’s internal code review process, and provide periodic reminders about this policy to employees and contractors involved in software development.

3.2 Forensic Code Audits

Forensic code auditing looks for the presence of legally problematic material in a body of source code. In and of itself, a forensic code audit will not remediate problematic source code but serves as an effective means to detect problems, which can then be remediated in an appropriate manner. Forensic code auditing provides a proactive, objective review of source code to detect possible licensing problems. Essentially, a forensic code audit uses a combination of automated tools and visual inspection to determine if third-party materials have been incorporated into proprietary code. A variety of techniques may be used to identify:

- 1) literally-similar or near-literally similar incorporation; third-party code has been incorporated with no or little textual modification into code;
- 2) obfuscated incorporation; third-party code has been incorporated with textual modifications (e.g., variable and parameter name spoofing, function reorganization, etc.), but the textual narrative can be abstracted, compared, and shown to be similar;
- 3) textually dissimilar but logically similar incorporation; third-party code has been substantially rewritten on a textual level, but the original third-party logical flow remains intact.

¹⁰ Appendix One describes a software pedigree analysis done in the context of mergers and acquisitions. This paper focuses on audits done in the context of internal quality processes.

A forensic code audit has two purposes: first, to determine whether third-party material is present in a code base (a technical exercise); and second, to determine the ultimate origin and license associated with those third-party materials (both a technical and a legal question). Be warned: determining ultimate origin and license conditions may present unexpected challenges. Sometimes the code may have an obvious third-party attribution but may come from a long-extinct source, i.e. a defunct “pre-Web” company who never posted any of its software or documentation assets on the Internet. Sometimes the code may be expressly distributed under a choice of licenses with no guidance as to what license applies in the immediate situation. In other cases, multiple licenses may apply due to the collaborative nature of the code’s development.

3.2.1 Scope the Audit

One of the first orders of business in performing a forensic code audit is to determine the scope of the audit. Scope decisions will define both the breadth (which applications) and the depth (which types of analysis) are appropriate to the business needs. A simple audit may comprise only a *surface scan* of a code base looking for obvious indicators of third-party usage (e.g. to discover whether there are third-party copyright or license notices in the code). Deeper audits may incorporate more sophisticated tools to identify more subtle evidence of copying. This scope determination must be made on a case-by-case basis, consistent with the circumstances triggering the audit, available time, risk tolerance, and cost factors.

3.2.2 Types of Analysis

Once the scope of the audit is established, the comparison can begin. Several types of comparison techniques may be used together during the analysis, including textual analysis, code comparisons, code scanning tools¹¹ (which usually involve some form of *digital fingerprinting*), and analysis of program logic.

Textual analysis examines a single body of code to determine whether it contains indications of third-party code in the form of copyright notices, license notices, or similar textual indicia.

Code comparisons match one body of code against another to identify areas of literal or near-literal similarity. Comparisons may be based on examining full lines of code, partial lines of code, significant internal names and literals, or some form of tokenized code representation.

Digital fingerprinting seeks to find identical files or sections of files by comparing algorithmically derived fingerprints of files or sections of files. The fingerprint usually takes the

¹¹ E.g. Palamida™ (<http://www.palamida.com/>), OSRM (<http://www.osriskmanagement.com/>), Black Duck™ (<http://www.blackducksoftware.com/>). All these companies have tools that compare a body of source code against a repository of code fingerprints generated from a variety of publicly available software projects (e.g. projects hosted on SourceForge), to minimally look for either in-file *snippets* of unattributed source code taken from third-party sources, or *digest matches* demonstrating that an entire file was taken from a third-party source. These tools are only as good as their search algorithms and fingerprint database and will not catch contamination from non-public sources (unless such materials are expressly incorporated into a custom fingerprint repository per customer request). In our experience, these tools must be accompanied by human analysis to weed out false positives and catch false negatives and, above all, to judge the contextual relevance of a *hit* on third-party materials.

form of a hash value.¹² (Appendix Two illustrates a simple use of MD5SUM hashes.) A reasonable analysis tool can compare the fingerprint of file A (or sections of file A) to a set of fingerprints as generated from file B and sections thereof (and file C, and so on).

In all cases, the more obfuscated the code, the more challenging the problem becomes for automated analysis. For example, simply changing the variable names in code copied from a third party may be sufficient to fool some automated comparison tools. Likewise, a minor amount of intra-function reorganization and/or name obfuscation may be enough to fool these tools. Automated tools can certainly facilitate the review process by highlighting potentially significant areas for review, but ultimately there is no substitute for direct visual inspection of two bodies of potentially matching source code.

The analysis of program logic and architecture is a more manually intensive and time-consuming process than the techniques above. Though automated analysis can still play a role in these situations, this scenario must rely upon visual inspection and the attendant experience of the examiner. While there are a variety of tools that permit an examiner to “walk through” a program in static fashion (visually examine the source code) and in dynamic fashion (watching the behavior of a program during its execution), these tools are usually directed to one body of code at a time and do not directly identify logical similarity. To date, that assessment is still best made by a human examiner.

3.2.3 Cautions

Given the enormous and ever-expanding universe of potential third-party material, no forensic software audit can provide a 100% guarantee that all possible third-party issues have been discovered in a code base. This is especially true regarding code that is derivative of, but nonetheless textually dissimilar to, third-party code.

In our experience, third-party materials are found not just in the obvious source files or binary redistributables, but also in a variety of potentially unexpected locations. For example:

- Container files such as Zip, RAR, Java JAR, MSI and CAB files: These types of *container* files store material in an internal format that can not be inspected for third-party materials directly without first performing an expansion or extraction of the files’ contents using an appropriate tool. The authors have encountered numerous instances where a seemingly innocuous container file turned out to contain unexpected third-party subcomponents, including third-party source code files under copyleft or proprietary licenses.
- Archives within archives: In some instances, unexpected third-party subcomponents may be buried in “archives within archives.” It is essential that before analyzing any materials set, multiple recursive passes are made through the materials to decompress *all* archives embodied within the materials.
- Font files: Many font files are, in fact, distributed under copyleft licenses.
- SQL files and databases: We have found various instances of third-party routines embedded into otherwise innocuous SQL files or saved in databases as stored procedures and queries.
- Program documentation: third-party material may be found in program documentation.

¹² MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value and is commonly used to check the integrity of files.

- Generated code: Some files may have been generated by a third-party tool, e.g. a GNU bison-generated parser file.¹³ These files may be subject to unexpected license terms.
- Referential citations to third-party materials under a potentially problematic license: We have observed a number of instances wherein a developer includes a comment in a source file citing to a third-party location as “the inspiration for” or the basis of the source code related to that comment. While in most cases we have determined that these instances are in fact benign referential citations, in some instances the citation was to material under a copyleft license, raising potential taint concerns (and/or possible publicity concerns).

This list is by no means exhaustive. The critical point is that a forensic code audit must take the *totality* of the materials into account.

4 EXPERIENCES

Over the course of several hundred forensic code audits, we have observed the following:

- Nearly every forensic code audit performed by the authors of this paper has turned up evidence of undocumented third-party material in the subject code base.
- A company’s disclosures regarding third-party materials have almost always proved incomplete, suggesting that many, if not most, companies have yet to implement effective pedigree tracking practices. While representations and warranties are certainly useful, our guiding axiom is “trust but verify.”
- Education is crucial. The authors have seen multi-million dollar litigation launched in response to the actions of a single uninformed developer.
- A proactive audit can help identify potential problems. In acquisition scenarios, the authors have seen deals terminated because of the unexpected scope of third-party materials in a seller’s code base, usually in contrast to the seller’s disclosures.
- Automated analysis tools are useful, but the human brain is ultimately the best judge of contextual relevance.

5 CONCLUSION

Based on our experiences, we anticipate that many companies will, in their not-too-distant future, face some of the issues discussed in this paper. The growth of the open source movement and open source projects help support our premise, as more and more companies leverage open source materials to produce commercial applications. Regarding that growth, Amit Deshpande and Dirk Riehle (SAP Research, SAP Labs LLC) said:

¹³ Bison is a parser generator that takes a grammar description, and converts that description into a C-language program to parse that grammar. See <http://www.gnu.org/software/bison/> [visited 6/25/2009]. Bison-generated parser files are expressly distributed under the terms of the GPL but with an exception that states: “As a special exception, you may create a larger work that contains part or all of the Bison parser skeleton and distribute that work under terms of your choice, so long as that work isn't itself a parser generator using the skeleton or a modified version thereof as a parser skeleton.”

The significance of open source has been continuously increasing over time. [...] [T]he number of new open source projects, and the total number of open source projects are growing at an exponential rate. The total amount of source code and the total number of projects double about every 14 months.¹⁴

Additionally, in April 2008 Sun UK's chief open-source officer, Simon Phipps, stated that “[i]n 2005 we came to the conclusion that this was the future for the software industry.[...] We see the software industry switching over from its current delivery model to adopt far greater focus on open source.” He further stated that “most chief information officers simply don't have a policy for open source.”¹⁵

Given these and other considerations discussed in this paper, we believe that the importance of well-documented software pedigrees and carefully-considered usage guidelines cannot be overstated. Companies using quality pedigree programs will be better positioned to deal with licensing questions, respond to merger and acquisition scenarios, or act quickly in the event of a lawsuit.

¹⁴ Proceedings of the Fourth Conference on Open Source Systems (OSS 2008). Springer Verlag, 2008. *Also available* at <http://dirkriehle.com/publications/2008/the-total-growth-of-open-source/> [visited 7/16/2009].

¹⁵ <http://www.zdnetasia.com/insight/software/0,39044822,62040810,00.htm> [visited 7/17/2009].

6 APPENDIX ONE -- SOFTWARE DUE DILIGENCE

A software pedigree analysis done in the context of mergers and acquisitions is often referred to as Software Due Diligence (“SDD”). Like a software pedigree analysis, it is a multi-step process that is used to determine the true origin of third-party material and to identify the applicable licenses associated with that material. The steps of the process include the scope determination, the preparation and delivery of the materials, and the preparation of a budget for the effort. Ideally, SDD involves independent analysts and possibly outside counsel since the buyer should not see the seller’s actual source materials, only a summation of findings based on that material. If the transaction should happen to fall through, an independent analysis framework helps avoid litigation related to misappropriation claims.

SDD is not always an easy assessment. There may be multiple sources for particular materials, with conflicting licenses and ultimately ambiguous origin. Often there is a single source for the code, but there may be a choice of licenses associated with the code. It may be unclear under which license the code is distributed.

SDD analysis is usually focused on literal expression in the code, i.e. obvious third-party attribution notices, suspect words or phrases, textually-similar code and file-level similarity. While non-literal analysis is possible, it is usually labor-intensive and therefore time-consuming and costly.

A SDD effort must be scoped and sized with input from all of the relevant players. Deliverables generally include the performance of the analysis, a report on the findings and, if necessary, a remediation review. The scope of a SDD is usually decided by the factors of both time and budget. A detailed review is preferred but not always possible. Time and/or cost pressures may limit analysis to obvious third-party materials. The scope is thus a factor of risk tolerance. An abbreviated assessment may be useful, but be riskier in the long run.

The third-party source code universe is vast and still largely uncharted. SDD cannot provide a 100% guarantee that all third-party code, derivative or otherwise, has been identified in a body of code since that is both a difficult and complicated task.¹⁶

In the end, most problems identified in the report findings are remediable. Some problems are better resolved prior to the close of the sale (e.g. issues related to continuing support obligations for acquired company’s software) while other can be resolved post-close (e.g. the disposition of non-critical source code). When core issues related to software are not easily resolved, it may be time for a reality check. How much liability do you want to buy?

¹⁶ Further, patents are the elephant in the room; i.e. even if the pedigree of the software is well established, the software could still infringe one or more patents.

7 APPENDIX TWO -- MD5SUM EXAMPLE

The simple example below illustrates how the values computed for md5hash sums relate to the file contents and not to other considerations such as file name and date created. The original file “bar (original file).txt” was copied into three different files. It was edited once in the file named “copy of bar (text edited after copy).txt” times; three of the four differently named files have identical md5sum hashes. The edited file has a different *fingerprint* i.e. it was changed and its md5sum hash reflects that change.

```
C:\WINDOWS\system32\cmd.exe
T:\My Documents\PNWSQC paper\md5sum example>dir *.txt
Volume in drive T has no label.
Volume Serial Number is CC96-776A

Directory of T:\My Documents\PNWSQC paper\md5sum example
07/16/2009  12:34 PM                11 bar (original file).txt
07/16/2009  01:07 PM                11 Copy of bar (text edited after copy).txt
07/16/2009  12:34 PM                11 Copy of foo.txt
07/16/2009  12:34 PM                11 foo (copy bar file but rename it) .txt
               4 File(s)                44 bytes
               0 Dir(s) 17,693,736,960 bytes free

T:\My Documents\PNWSQC paper\md5sum example>md5sum *.txt
5eb63bbbe01eeed093cb22bb8f5acdc3 *bar (original file).txt
b10a8db164e0754105b7a99be72e3fe5 *Copy of bar (text edited after copy).txt
5eb63bbbe01eeed093cb22bb8f5acdc3 *Copy of foo.txt
5eb63bbbe01eeed093cb22bb8f5acdc3 *foo (copy bar file but rename it) .txt

T:\My Documents\PNWSQC paper\md5sum example>type *.txt

bar (original file).txt

hello world
Copy of bar (text edited after copy).txt

Hello World
Copy of foo.txt

hello world
foo (copy bar file but rename it) .txt

hello world
T:\My Documents\PNWSQC paper\md5sum example>
```

8 APPENDIX THREE -- PEDIGREE CHECK LIST

- What third-party code is in use?
- Which version?
- Where did it come from?
- When was it procured?
- What license(s) apply to the third-party code?
- Which version of the license?
- Is a copy of the license on file?
- What is the scope of intended use for the third-party code?
 - Is the proposed use internal, or will it be incorporated into product(s) distributed outside your organization?
 - Will the third-party code be used in the context of “software as a service” (client / server scenario)?
 - Will the third-party code be “pushed” to a user, such as JavaScript files to a user’s browser?
 - If distributed outside your organization, is the distribution under a commercial license or an open source license?
 - How will you comply with license requirements such as attribution notices or availability of your modified source code?
- Where and how is the third-party code used in your software?
- Are all license compliance requirements related to the third party code met?
- What processes are in place to track compliance issues?
- Who is responsible for compliance?
- Who is responsible for code review?
- What directives have developers been given with respect to this code?
- Have developers affirmed they followed directives?
- What processes ensure developers follow the directives?
- Once third party code has been used, what procedures govern its re-use for subsequent development projects?
- Are your developers aware of the specific procedures?
- Are there controls over third party code use? (Example: developers may be required to “check out” third party source from a centralized repository of approved code)
- Do you plan to perform periodic compliance audits?
- If so, who will perform the audit?
- How will the audit results be documented?
- If follow-up is required post-audit, who will be responsible for the follow-up?
- What steps have you taken to ensure that your developers and consultants do not incorporate code written for past employers into your products?

BIOGRAPHY

Susan Courtney is a forensic software analyst and has worked as a consultant for Johnson-Laird, Inc. She has done forensic software analysis for patent and copyright cases, performed electronic evidence analysis, and assisted with data preservation and discovery. Ms. Courtney is also the owner of SLC Software LLC, a software consulting firm that provides a variety of software-related services including business systems analysis, quality assurance and project management.

Barbara Frederiksen-Cross is the Senior Managing Consultant for Johnson- Laird, Inc., in Portland, Oregon. Barbara is a forensic software analyst specializing in the analysis of computer-based evidence for copyright, patent, and trade secret litigation. She is also an expert in computer software design and development, the recovery, preservation, and analysis of computer-based evidence, and computer systems' capacity issues. Mrs. Frederiksen-Cross was appointed as Court Data System Advisor to the Honorable Marvin J. Garbis, in the U.S. District Court for the District of Maryland in December 2000.

Marc Visnick is a forensic software analyst and attorney based in Portland, Oregon, and a senior consultant with Johnson-Laird, Inc. He specializes in forensic software analysis for patent, copyright and trade secret litigation, as well as software due diligence, independent development project design and supervision, and electronic evidence preservation, recovery and analysis. Mr. Visnick is past Chair of the Oregon State Bar Computer and Internet Law Section.