28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

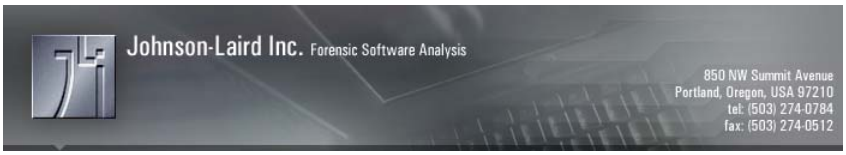
ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

QUALITY PEDIGREE PROGRAMS: OR HOW TO MITIGATE RISKS AND COVER YOUR ASSETS

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



ABSTRACT

Forensic software analysts routinely review source code in the context of litigation or internal software audits to assess whether, and, to what degree, a body of software uses or references third-party materials. These references may include source code examples incorporated directly into a program, source code routines that are statically linked as part of the program, the use of binary libraries that are dynamically referenced when a program is executed, or URL-based citations to third-party materials, such as an article on a website. While third-party materials are obviously invaluable to software development, third-party materials may introduce a variety of legal or security risks into software and expose a company to unexpected legal liability and/or negative publicity. Thus, quality software is defined not just by technical measurements, but also by the presence of a comprehensive set of policies and procedures that help mitigate these potential risks.

We believe it is essential that companies proactively establish a baseline pedigree for their existing software via a *forensic code audit*. This act authenticates a moment in time where all known third-party materials are appropriately catalogued, and risks associated with those materials are fully understood by the company's relevant business and legal stakeholders. But a one-time pedigree analysis is not sufficient. A pedigree analysis should be part of a comprehensive, *quality pedigree program,* a set of well-defined prophylactic policies and procedures surrounding the use of third-party materials. These policies and procedures should take into account the entire software lifecycle, including any customer support obligations that may remain once a program is deprecated. A company that proactively implements a quality pedigree program is better positioned to respond to customer requests, react to lawsuits or potential licensing problems, or to justify a particular valuation of their intellectual property in the context of a merger or acquisition.

BIOGRAPHY

Barbara Frederiksen-Cross is the Senior Managing Consultant for Johnson-Laird, Inc., in Portland, Oregon.

Marc Visnick is a forensic software analyst and attorney based in Portland, Oregon, and a senior consultant with Johnson-Laird, Inc.

Susan Courtney is a forensic software analyst and has worked as a consultant for Johnson-Laird, Inc.

Published at the Pacific Northwest Software Quality Conference 2010

1 INTRODUCTION

Modern source code development is rarely devoid of third-party content. The reason is simple (and is guided by the same principles that govern modern manufacturing): it doesn't make sense to reinvent the wheel for every automobile you build. Programmers are keenly attuned to this reality and routinely re-use existing source code when developing new products. Cutting and pasting code from past development projects, Open Source repositories, third-party tools, or code found on the Internet accelerates new development. The use of such reusable solutions saves time reduces debugging effort, and helps build more reliable code.

In addition to reusable solutions, a company that accepts such projects (make no mistake about it, that would be nearly all companies) inherits new responsibilities. The failure to effectively manage the use of third-party code can result in entanglements that open the door to litigation, trade secret loss, security vulnerabilities, and costly remediation efforts. The use of third-party source code becomes problematic when software developers write software that references third-party materials and fail to document such usage, fail to understand or comply with licensing restrictions, fail to consider copyright, trade secret, or patent entanglements, or fail to consider security implications.

Each of the problems listed above can occur whether the third-party code is proprietary code or Open Source code. We are all familiar with legal problems that arise when employees re-use code from past employers or consultants re-use source code that was initially written for a competitor. Disputes over copyrights or trade secrets can arise even with code developed under commercial licenses. The potential for problems expands exponentially for Open Source, due in part to the fact that it is readily available to developers without any authorization from (or visibility to) management, and in part due to the proliferation of different licensing models that may apply to Open Source. These licensing models may introduce significant legal complexities for commercial organizations, as they may contain terms that render them incompatible with commercial use or which contain terms that require making available any source code derived from code under an Open Source license.[1]

Use of third-party code (Open Source, public domain, licensed, or purchased) may also foster technical exposures such as security vulnerabilities. For example, if the security of an Open Source component is compromised, an individual who is aware of its security vulnerability may exploit the vulnerability to gain unauthorized access to the program function or to the computer system upon which it is running. Since the same open source routine may be incorporated into many different products, would-be intruders may find a wide range of targets that all stem from the same vulnerability.

Even if the third-party code is free of security vulnerabilities, its use may still incur security risks due to license terms that require public availability for the source code of its derivative product. Publication of source code gives would-be intruders an excellent opportunity to search out security vulnerabilities that may exist in the product. It also gives competitors the opportunity to study all aspects of the program's operation in intimate detail, including any trade secrets the program may embody.

Even if license terms do not require making source code available, the security ramifications of other license requirements, such as attribution, should be carefully examined. The knowledge that particular software includes as constituent parts specific Open Source or third-party components necessarily reveals some aspects of its internal operation, with the result that the particular software is rendered more vulnerable to tampering or reverse engineering.

2 MANAGING THE USE OF THIRD-PARTY CODE

The problems that can stem from third-party code entanglements get worse over time. As comingled code evolves, the fact of the comingling is easy to document, but the actual origin of any particular code

[1] For more information about the various types of Open Source licensees *see* http://www.gnu.org/licenses/license-list.html

component becomes harder to ascertain. Since modified code may be re-used and spread to other products, it becomes harder to identify the scope of the original code's use, and harder to contain or correct any damage or risk. The costs associated with tracking the true origins of third-party code will also increase over time, since the evidence of the code's origin may be further eroded by subsequent modifications. Further, the original third-party product may be superseded by subsequent releases, rendering code matching exercises more difficult.

Management of Open Source and third-party content is best accomplished through prophylactic procedures, not reactive panic. Companies should proactively establish a *quality pedigree program* that introduces a series of "best practices" which include:

1) carefully considered guidelines for using third-party materials,
2) a clearly defined review and approval process,
3) thorough documentation of third-party code use,
4) the periodic performance of code audits to detect the presence of third-party material in a body of source code.

2.1 The Core Team

The skills required to effectively manage third-party source code dictate the need for a core compliance team that includes participants from both legal and technical backgrounds, including input from individuals responsible for business development strategy. Management's first step to bring third-party code usage under continuous control is to charter this team.

The compliance team should be led by a single compliance officer who assumes overall responsibility and final authority for the review and approval for use of third-party code (including Open Source). Business development managers and other stakeholders help the team prioritize products for analysis, and identify critical timeframes relative to new product rollouts or system implementation dates.

One or more participants with a legal background review license terms and assess legal risk factors relating to compliance and remediation efforts. Legal team members may also assess the effect of third-party code use on internal and external contracts, user licenses, and intellectual property matters relating to trade secrets, copyrights, and patentability.

One or more team members with a technical background review candidate third-party code, and assist in determining the scope and nature of existing third-party code usage. Technical team members also help assess risks related to Open Source security and whether its proposed use is appropriate; *i.e.* determine if the terms of license restrictions for commercial use or restrictions requires the publication of code that embodies trade secrets or other proprietary technology.

2.2 Proactive Code Management

Proactive management of Open Source and third-party content should include a published policy that defines where and how Open Source and third-party content may be used. This policy defines the criteria that distinguish between approved and unapproved licenses and code origins. The management policy should have a clearly identify scope with respect to software developed by in-house developers, independent contractors, software vendors, and development partners. Where necessary, the management team should update contract language to addresses code developed by business partners, contractors, consultants, and outside developers.

Software pedigree documentation, which identifies the origin and use of all third-party components, should be formalized and produced as an artifact of all new development projects. These records should be maintained and updated throughout the lifecycle of the software to account for any changes to third-party content. Think of these records as insurance or as a metaphorical *get-out-of-jail-free* card that guards against expensive litigation.

The policies governing third-party code management should include an approval process for the evaluation of third-party material, its associated documentation identifies the code origin and license restrictions on its use. The policy governing the review should clearly document the review submission process, including the expected turnaround for approval, and the communication mechanism that will be used to notify the applicant about the result of their request.

Approval from the review process will optimally be obtained prior to implementing new third-party code. It is crucial to define a review process that is sensitive to your company's business needs, development model, and risk tolerance. The review process itself will typically include identification and assessment of the proposed usage, applicable license terms, legal considerations relating to licensing and Intellectual Property ("IP") ownership, and a review of any known security considerations with respect to the software or its use.

To facilitate the approval review, engineering and software development teams should provide advanced notice to the review team about the proposed third-party code use. The notice should identify the software, its origin, the proposed scope of its use (*e.g.,* experimental, internal use only, development tool, distributed with product, etc.). To expedite review, a copy of the actual license text is attached to the document for the review process. The review document and the license should be preserved for future reference.

Although this process may sound cumbersome, streamlining is easily achieved by the use of black lists, white lists, and grey lists. These lists will grow over time and represent classifications of licenses and source origins into one of three categories:

- The *black list* identifies materials that must never be used (e.g. materials developed for previous employers, materials whose source cannot be determined, and materials subject to unacceptable license restrictions);
- The *white list* identifies pre-approved materials (e.g. license terms and origin have already been accepted, and scope of proposed new use is unlimited or precisely defined);
- The *grey list* identifies materials that are candidates for use, but require approval to assess license terms in the context of a specific proposed use.

The approval review process includes maintaining a special repository for approved third-party code and associated licenses as well as documentation to preserve a clear record of both the original third-party code base and the license under which it was obtained.

Once code has been approved from the white or grey lists, it is also important to maintain records of where, in the larger context of a company's software, that particular code is actually used, because the scope of use may grow over time. The usage records should provide information at the program or product level that identifies the specific version of third-party code used, when it was implemented, the scope or context of its use, *e.g.* internal vs. external, etc. Additionally, the records should document which specific components incorporate the code and/or which components interact with it. Further, if at all, possible individual source code files should incorporate a comment block that documents any restrictions that might apply to use of the code within a file.

Taken together, the repository of approved code and the records related to its use form the basis for technical due diligence reporting that may be required in the context of mergers, acquisitions, and divestitures. These records will also be extremely helpful in the event of downstream problems that might arise with respect to reliability, security vulnerabilities, licensing issues, patent disputes, or other legal issues related to intellectual property or product liability.

Once third-party code management policies are in place, employees, contractors, and consultants must be educated about their obligation to protect company assets by using said policies. This includes training all developers (and their managers) in the actual mechanics of the review and approval process.

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.

In the authors' experience, it is extremely helpful to implement some form of regular, periodic audits or other monitoring process to identify potential problems that may arise during early implementation of the new policies. The policy should also include exception procedures to address existing code or other special circumstances.

3 LEGACY CODE

The development of third-party code management policies is directed toward preventing future problems, but what about legacy code? Open Source and third-party code bases have been available to developers for many years. Despite this, only 22% of companies surveyed reported their organizations have explicit management policies/procedures in place.[2]

The results of this survey suggest that many companies should begin their source code management program with an audit of their existing code base. The code audit, in combination with carefully considered guidelines for using third-party materials, and thorough documentation of such use, completes the foundation of a proactive source code management program. The combined use of these three practices establishes a permanent record ("pedigree") documenting the origin of your company's software.

The existence of software pedigrees helps to manage the risk of unforeseen legal entanglements. A company using a third-party code management 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. A proactive management plan for third-party source code can help protect intellectual property, and it can also help identify software that includes components known to have security vulnerabilities.

Since software may be modified in response to changing business needs, any proactive plan must include processes that apply to ongoing development and maintenance, as well as the initial code development.

3.1 CODE AUDIT

As noted above, the first step toward managing third-party code use is often an audit of the existing code base. The code audit has two primary goals: 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). The audit seeks to answer very basic questions: What third-party code are you using? Where did the third-party code come from? What license(s) pertain to its use?

[2] Based on a survey conducted by Black Duck Software, Inc. at the SD West conference (March 11, 2009). SD West is an international technical conference for software developers. Black Duck Software, Inc. provides services and software to help companies manage their use of Open Source and Internet-downloadable source code.

The code audit itself does not remediate problem source code. Rather, it serves as an effective means to detect potential problems, providing useful information about the nature and scope of third-party code entanglements, and identifying applicable licenses.

One of the first orders of business in performing a 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. It is important to note that no software audit can provide a 100% guarantee that *all* possible third-party issues will be discovered in a code base. This is especially true regarding code that is derivative of, but nonetheless textually dissimilar to, third-party code. 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 subtle evidence of copying. The scope of the audit is determined on a case-by-case basis, and is consistent with the circumstances triggering the audit, available time, risk tolerance, and cost factors.

Typically, the code audit uses a combination of automated tools and visual inspection to determine if third-party materials have been incorporated into proprietary code. Depending on circumstances, a variety of tools and techniques may be used. The software tools used for auditing range from simple search tools used to identify indicia of third-party origin, to sophisticated tools that compare one body of code against another. Cases that involve non-literal or structural analysis may require the use of specialized tools and techniques that render visible the architectural structure and program control flow.

One very common technique used for software audits is textual searching. Textual search strategies can reveal indicia of third-party origin, such as copyright or license references and annotations, that identify code which was "adapted from", based on", or "stolen from"[3] some third-party source. Although extremely useful, the results from textual searching depend on the premise that there are relevant comments embedded in the source code, and that the list of terms used for searching is sufficiently broad to identify all third-party code. For this reason, textual searching may not be sufficient for a thorough audit, especially if the body of existing code is very large. It can, however, be extremely valuable to help form a preliminary assessment about the number and nature of third-party products entangled in the source code.

The results of textual searching are often refined and supplemented by using code matching tools which serve to identify literally-similar or near-literally similar code use, *i.e.* cases where third-party code has been incorporated with no or little textual modification into code. Some code matching tools can perform an automated comparison of entire code bodies against extensive repositories that contain source code from many different publicly available sources.[4] Because such tools actually compare code, rather than merely searching textual content, they can catch third-party code use that might otherwise remain undetected.

Even with the use of very sophisticated tools, human analysis is still required to weed out false positives, catch false negatives, and, above all, to judge the contextual relevance of a hit on third-party materials. Human judgment is also required in assessing the degree to which third-party code may present a risk to security, patentability, trade secrecy, or control of intellectual property.

---

[3] Most programming languages allow programmers to annotate the source code with non-functional comments. Such comments often contain information about the purpose, origin, and operation of the code. In the author's experience, it is not uncommon to see the phrase "stolen from" in comments that reference the origins of coding routines adapted from third-party sources.

[4] E.g. Palamida™ (http://www.palamida.com/), OSRM (http://www.osriskmanagement.com/), and 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. It is important to note that these tools will not catch contamination from non-public sources (unless such materials are expressly incorporated into a custom fingerprint repository per customer request). Because they do not fully analyze context, these tools may also generate false positives. In the author's experience, these tools must be accompanied by human analysis.

A thorough search for third-party materials should also extend beyond source code libraries to include executables, documentation, and other types of files. In the authors' 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 cannot 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[5] or proprietary licenses.
- Archives within archives: In some instances, unexpected third-party sub-components 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.
- Databases and stored queries: 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 such as code samples may be found in program documentation.
- Generated code: Files generated by a third-party tool, such as the GNU bison-generated parser files[6] 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.

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

Regardless of the tools used, the product of a code audit should be a detailed record documenting every instance of third-party code, where it was found, what purpose it serves, and whether the code is used internally or distributed as a part of some product. These records should be sufficiently detailed and identify every file that includes third-party material. Where possible, the records should also preserve any information gleaned about the code's origin, owner, copyright, license terms, and the specific third-party product or project from which the code was derived. This file-level metadata will prove invaluable in the event of litigation.

3.2 LICENSE REVIEW

Once third-party code inclusions have been identified, the next step is to identify the code origin and the applicable license conditions. In many cases, comments within the source code itself will identify the specific governing license or the software product and owner. Armed with this information, licenses may be located via simple Internet searches, and, with luck, their complete text may be downloadable. In other instances, the search for source code origins and the associated license text may present unexpected

[5] Copyleft is a general term for license restrictions that require that users who use, modify, or extend a particular Open Source program must make available the source code of the modified work.

[6] 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 GNU General Public License ("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."

challenges. Sometimes the source code may have an obvious third-party attribution, but may come from a company that does not appear to have posted any of its software or documentation on the Internet. This is not necessarily a contradiction, as it may reveal that an outsourced developer is recycling code written for prior clients. In other cases, the software may reference companies (or individuals) whose assets have transferred as a result of mergers and acquisitions, or products that are no longer publicly available.

Your own company's acquisition of source code assets may also prove problematic, if the software pedigrees lack details at the program-specific level. This is especially true if the original copyrights were not supplemented or if the acquired code includes unexpected third-party content.

It is common for Open Source code to be expressly distributed under a choice of licenses, and found during the audit in a context that offers no guidance as to what license applies. In other cases, multiple licenses may apply due to the collaborative nature of the code's development.

The product of the license review should be a detailed record that includes the full text of each license, and where and when it was obtained. As described in greater detail below, it will be important to maintain some enduring record that establishes which license(s) correlate to which programs or code portions. This reference is essential because the compliance evaluation must be done on a program-by-program basis, since software licenses may specify different terms for commercial and non-commercial usage.

3.3 Determining Compliance Status

Once third-party code and any associated license(s) have been identified, assessment of compliance status and identification of related technical issues may proceed. This process has two primary components: a legal assessment of the current compliance status and a technical assessment of factors related to the code and its use.

The focus of the legal assessment is a review of license terms governing each third-party source code use to determine its compliance requirements, and a review of current practices to determine the current status with respect to those requirements. In addition to reviewing accounting records for third-party software licenses, a compliance assessment will likely require review and revision of the following:

1) end user license agreements or product packaging (to assess attribution compliance);

2) product revision and software download practices (to determine if required notices are shipped with each version of the product); and

3) web or phone based download support (to review the process whereby GPL-based source code can be requested or delivered.)

The technical assessment provides program-specific information that must be considered during the legal assessment. For example, the same license may contain separate compliance requirements for commercial and non-commercial usage, thereby necessitating a technical evaluation to determine the context in which the third-party code is used. Additionally, the technical assessment should address whether code based on third-party software would pose security risks, or compromise trade secrets, if license compliance required publication of the source code.

Original (unmodified) copies of any Open Source or third-party code should be preserved. If remediation requires the removal of the third-party code, a comparison of the existing code to the unmodified third-party base can expedite development of a roadmap that identifies the components which must be replaced. Conversely, if the third-party code remains a permanent part of your product, the original base code may be extremely helpful in the context of patent, trade secret, or copyright disputes. In the case of Open Source, the complete license text, as well as, any documentation relating to origin and use should be maintained in the same repository used for the base code. This practice can also be followed for proprietary third-party source, assuming that source code is available under the terms of the license agreement.

Careful record keeping during the review process is essential, and will provide a foundation that minimizes the burden of future reviews. At a minimum, the record keeping should identify where compliance has already been achieved and the status of any potential compliance issues that require further research or action. Depending on the depth of the review, the review records may also be used to flag issues relating to intellectual property protection, Sarbanes Oxley[7] compliance, or similar categories of concern.

Generally speaking, the compliance recordkeeping is program-centric or product-centric rather than license centric, since the same piece of third-party code may be re-used multiple times in different contexts. For example, third-party code may be both used internally and as part of a distributed product, and is, potentially, subject to different license terms, depending on the context of its use.

3.4 REMEDIATION

The specific steps taken to address compliance related to third-party code use, *i.e.* remediation, must consider both the needs of the business and the terms of the license. Where remediation is necessary, it will generally fall into one of the following categories:

1) Modify existing documents and processes to comply with the license terms;

2) Remove or replace the third-party code;

3) Try to negotiate for more acceptable terms.[8]

As with any business decision, factors of cost and risk will influence the decision of whether and how to remediate the issues that arise from third-party code use. Legal factors, such as license terms that prohibit all commercial use, may drive the decision process. In other cases, additional technical or legal fact-finding may be required to support the decision making process. Technical factors, such as the importance of the third-party code, the cost to replace it, the degree to which the code has been modified for integration, and resource or time constraints, may limit available remediation strategies.

Unfortunately, there is no general rule to guide retrospective remediation– every license and code use will need to be addressed on a case-by-case basis. As each case is decided, a record of the specific actions taken for remediation, as well as the factors considered, should be preserved.

4 EXPERIENCES

Distilled from several hundred forensic code audits performed in response to litigation matters, technical due diligence for mergers and acquisitions, and internal code audits to proactively identify third-party code use, the following observations are offered:

- Nearly every forensic code audit performed by the authors of this paper has revealed evidence of undocumented third-party material in the subject code base.
- Automated analysis tools, while useful, are not substitutes for human analysis; the human brain is ultimately the best judge of contextual relevance.
- A company's disclosures regarding third-party materials almost always prove incomplete. This suggests that many, if not most, companies have yet to both understand fully this vulnerability and to implement effective pedigree tracking practices. While representations and warranties are certainly useful, our guiding axiom is "trust but verify."

[7] The legislation came into force in 2002 and introduced major changes to the regulation of financial practice and corporate governance. For more information *see* http://www.soxlaw.com/index.htm

[8] This form of remediation may be successful if the negotiation occurs before the code is actually used. Its likelihood of success diminishes once the third-party code is incorporated into live software.

- 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.
- Proactive audits can also have big payoffs in the context of mergers and acquisitions. The enduring value of your IP assets depends on proper identification, disclosure and management of third-party content; moreover, sophisticated buyers may force last minute discounts when the seller can not comply with technical due diligence disclosures.
- The validity and enforceability of copyright registrations depend upon appropriate identification of pre-existing works, including Open Source and third-party components.

5 CONCLUSION

Successful, profitable, and cost-effective management of Open Source and third-party content is best accomplished through prophylactic procedures, not reactive panic. Almost all of the pedigree problems of software can be managed, and the pitfalls described in this paper avoided, with proactive care. "Best practices" include 1) carefully considered guidelines for using third-party materials, 2) a clearly defined review and approval process, 3) thorough documentation of third part code use, and 4) the periodic performance of a code audit to detect the presence of third-party material in a body of source code. The use of periodic audits serves 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 four 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 in the context of a merger or acquisition.

6 APPENDIX ONE – CODE MANAGEMENT CHECK LIST

This checklist is designed to help foster dialog and awareness between legal and technical participants in the code management process. While not exhaustive, this list may be helpful to identify opportunities to strengthen your current code management processes.

- What third-party code is in use today?
- Can the specific owner and software version be identified?
- Where did it come from?
- When was it obtained?
- How long has it been in use?
- What license(s) apply to the third-party code?
- Which version of the license?
- Is a copy of the license on file?
- Does the license permit commercial use?
- Does the license contain copyleft terms?
- Does the license have any clauses that raise patent issues? (Licensing, indemnification, existing infringement suits, etc.)
- What is the intended usage scope 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?
    - If the third-party code is closely integrated with your own development, how do you tell who owns what?
    - What are the license terms that govern copyright ownership?
    - If there are multiple owners, do they jointly own everything?
    - If not, what are the rules that govern ownership?
    - Can you apply these rules to specifically identify what you own?
        - If not, you need to clarify the rules
        - If so, generate a list of the specific programs (“manifest”) for which you claim copyright. Generate a separate manifest for jointly owned materials. Generate a third manifest for the parts you do not own.
- Has the third-party code been reviewed to identify whether it is subject to known security vulnerabilities?
- Are all license compliance requirements related to the third-party code met?
    - For developers?
    - For distribution to end customers?
- What processes are in place to track compliance issues?
- Who is responsible for compliance?
- Who is responsible for code review?
- Are the guidelines used in the code review process clearly documented?
- What directives have developers been given with respect to the use of third-party code?
- Have developers affirmed they understand and follow these directives?
- What processes ensure developers are following the directives?
- Are there published guidelines for developers who contribute to Open Source initiatives?

- 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 these controls properly address re-use of code in future development?
  - Do guidelines clearly distinguish between code that can be used internally and code that can be redistributed?
- Do you plan to perform periodic compliance audits?
  - If so, who will perform the audit?
  - What process will be used?
  - How will the audit results be documented?
- If an audit reveals the need for remedial action, do you have a plan?
  - Who will decide what action is appropriate?
  - Are there findings that might require you to stop shipping product?
  - What criteria will be used to make this determination?
  - If so, what notification procedures will be used, both internally and externally?
  - Does the process differ if there is a licensing problem vs. security vulnerability?
  - What will you do if you discover a “back door” in your code?
  - Who will perform the required actions, which may require involvement of both legal and technical participants?
  - Who pays for the remediation?
- 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?
- Have you communicated your code management standards to your consultants, contractors, and software development partners?
- Does your business plan include success based on longer trajectory of time or shorter? In either case, if code management is not a priority is there a budgeted ‘war chest’ for the costs of emergency and expedited management? (In many ways, code management is akin to changing the oil in your car. You can avoid it, but the avoidance will be costly.)