blackduck™

The Quest for an "Open Source Genome"

A genome is the hereditary information encoded in an organism's DNA. When patterns of hereditary information in a person's genome are compared with the patterns of others, scientists can learn a great deal about that person's ancestry. For example, a recent genetic study suggests that Genghis Khan's direct patrilineal descendants constitute some 8 percent of men in a large swath of Asia—about 0.5 percent of the world's total population[1].

Like a person, each piece of source code has the analog of DNA and a unique genome containing clues to its origin and history. To identify open source code elements and determine where they originated, we need the open source equivalent to the Human Genome Project, a 13-year research project that identified and documented all of the estimated 20,000-25,000 human genes.

This paper describes how the "open source genome"[2] concept can be useful in understanding the origin and history of your code. We explore the nature of open source, why and how software developers use it, how information from an open source genome can help identify open source in your code and establish its origins, and why that is important.

The Nature of Open Source

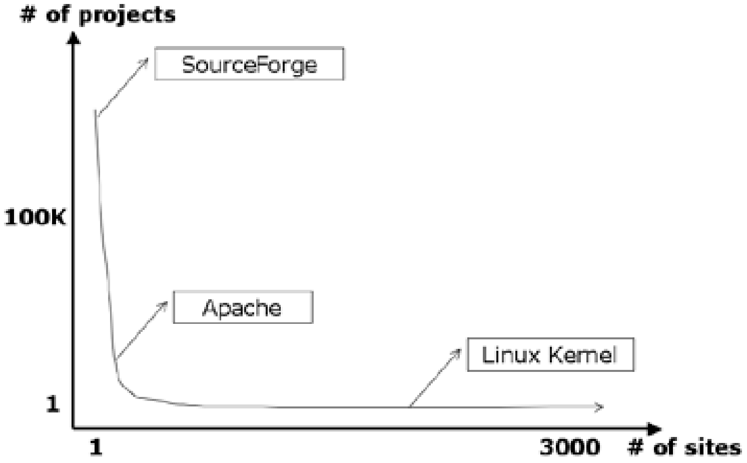
Author Bernard Golden defines open source as, "…software that has source code available to its users. It can be downloaded at will and used or modified as desired, as long as its license requirements are observed."[3]

Open source is a wonderful resource because it allows developers to reuse existing software to fill specific needs rather than write new software from scratch. An additional benefit is a Darwinian tendency that favors survival of the "fittest" open source. Open source that best meets users' and developers' needs lives on, and because the surviving code is vetted and improved by many people, it evolves to become more bug free, useful, and robust.

Open source code is developed within what are called open source projects. As of this writing there are more than 180,000 separate open source projects (although not all are active), with more spawned daily, so there is lots of open source for developers to explore and reuse.

By definition open source code is shared, so open source projects are generally hosted on the Web for public accessibility (although code was originally shared by tape and bulletin boards and is sometimes available from other sources such as books). A broad range of Web sites host open source projects. To date, Black Duck Software has identified more than 3,000 such sites. The most popular of these, SourceForge.net, hosts the lion's share of open source projects (more than 155,000 as of this writing), and adds an average of 2,400 new projects each month. Figure 1 illustrates the breadth of open source project sources.

Figure 1. Open Source Project Sources [4]



[1] Tatiana Zerjal, Yali Xue, Giorgio Bertorelle, R. Spencer Wells, Weidong Bao, Suling Zhu, Raheel Qamar, Qasim Ayub, Aisha Mohyuddin, Songbin Fu, Pu Li, Nadira Yuldasheva, Ruslan Ruzibakiev, Jiujin Xu, Qunfang Shu, Ruofu Du, Huanming Yang, Matthew E. Hurles, Elizabeth Robinson,1, Tudevdagva Gerelsaikhan, Bumbein Dashnyam, S. Qasim Mehdi, and Chris Tyler-Smith, "The Genetic Legacy of the Mongols" *The American Journal of Human Genetics* Volume 72 Number 3 (March 2003)

[2] Note that although this paper describes an open source genome, the concept applies to all code not just open source code.

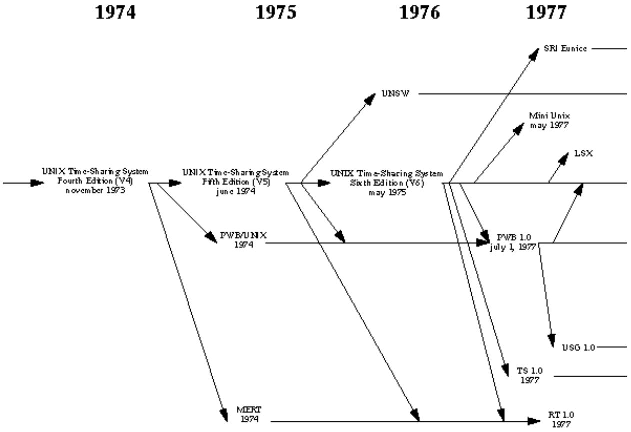
[3] Bernard Golden, *Succeeding with Open Source* (Boston: Addison Wesley, 2005), p.5.

[4] Source: Black Duck Software

Many new open source projects use code cloned from previous projects as their starting point. In these cases the new project's code begins life identical to its progenitor and is subsequently changed and augmented to satisfy the project's goals. The offspring thus becomes a new branch on an open source family tree, with each descendant carrying the software DNA from its ancestors. This software DNA can be used to identify ancestors and determine when an open source project's ancestor branched from the family tree.

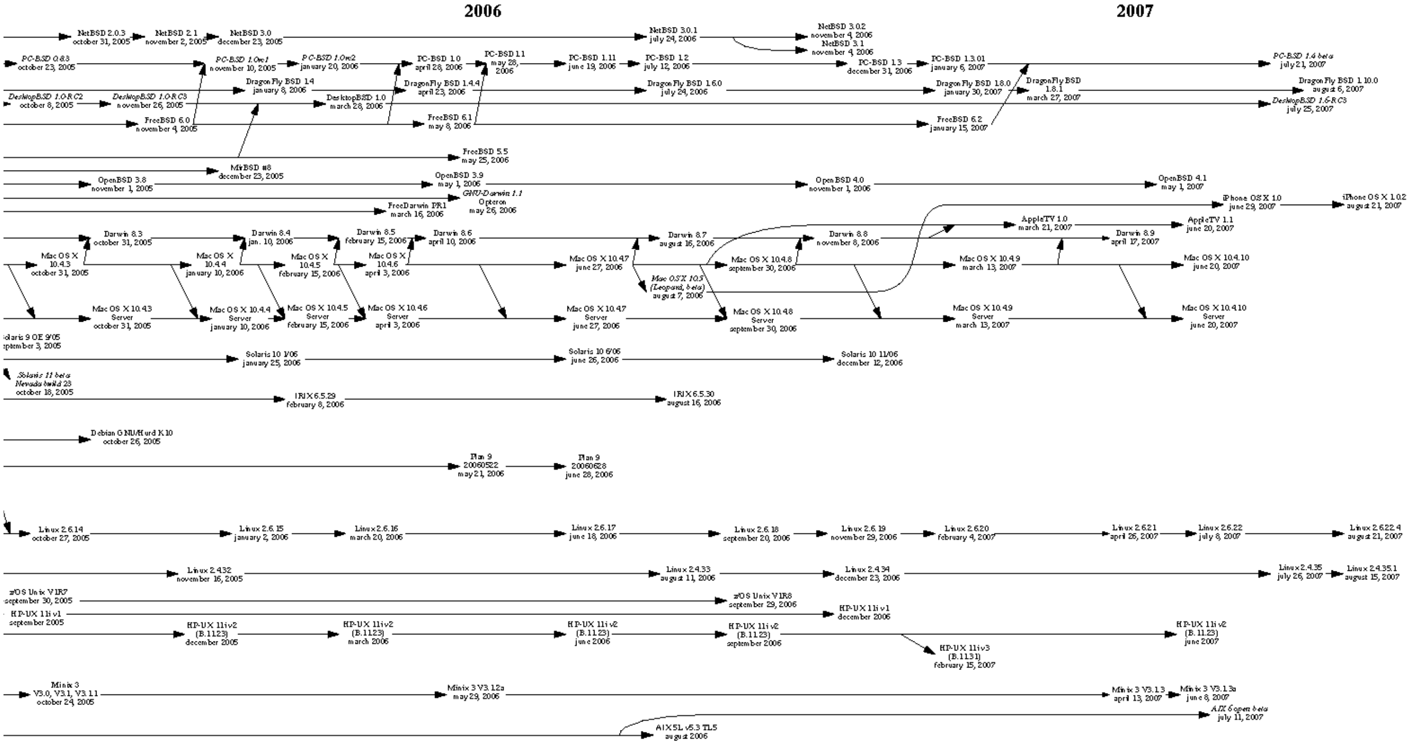
UNIX, born in 1969, provides a prime example of how complex a code family tree can become. Over nearly 40 years the UNIX family tree has branched and generated hundreds of offspring, and is now so large that space precludes us from including it in its entirety. To illustrate its growth and complexity, Figure 2 shows a snapshot of the UNIX family tree from 1974 to 1977, and Figure 3 is a snapshot from the recent past.

Figure 2. UNIX Family Tree 1974 to 1977[5]



As you can see, projects with shared development and source code constantly evolve, grow, spawn offspring—and in some case die. Some open source projects thrive, while others fail the Darwinian fitness test and pass into oblivion.

Figure 3. UNIX Family Tree 2006 to Early 2007 [6]



[5] http://www.levenez.com/unix/history.html#03

[6] Ibid.

The evolutionary nature of open source means that the genome for any open source project evolves over time. To be truly useful, therefore, an open source genome project is a perpetual work in progress that uses a learning approach to gather information about new open source projects, version changes within existing projects, and "missing link" projects that have died but whose software DNA lives on within other projects.

How Developers Use Open Source

To help developers locate relevant code within the terabytes of open source available on the Internet, a host of open source-specific search sites have emerged (see Figure 4). By making it easier to find the right code, these search sites enable developers to reuse code they might not otherwise find. Because open source is easy to find and reuse, code from many sources is often embedded into a single software product. Each time a developer incorporates open source into a product, the software DNA for that code (even for small code snippets) is incorporated as well.

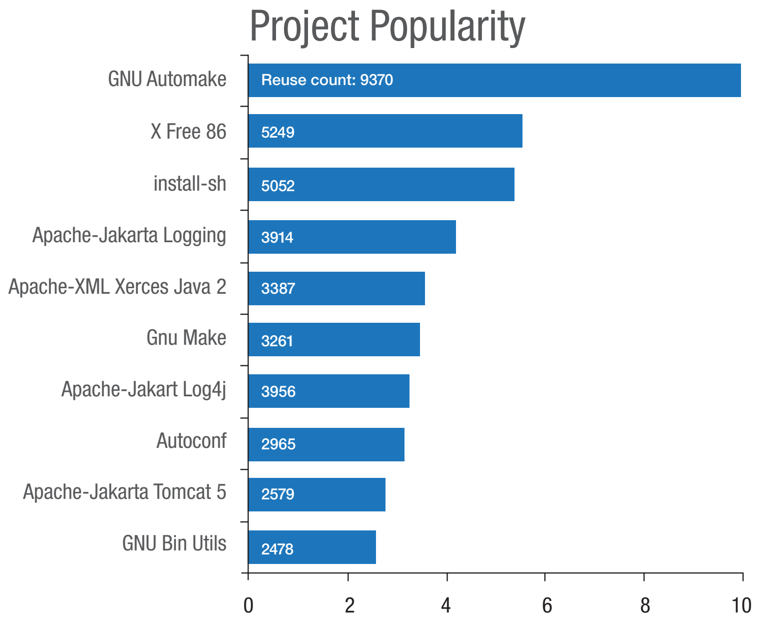
Figure 4. Open Source Search Sites

Apache.org	Koders
CodeProject	Krugle
CodeZoo	Ohloh
csourcesearch.net	OSDir.com
Enterprise Open Source Directory	Oslookup.org
freshmeat	Savannah
Google Code Search	SourceForge.net

Code from some open source projects tends to be more popular than others. Figure 5 shows the 10 open source projects whose code is most reused in other projects. This analysis done by Black Duck Software shows that code from GNU Automake is most reused, followed by X Free 86 and install-sh. Some might say GNU Automake is the Genghis Kahn of the open source world. The more generally useful code is, the more likely its software DNA will propagate.

Like the open source Web site chart shown in Figure 1, the popularity of open source projects exhibits a

Figure 5. 10 Open Source Projects Most Reused within Other Open Source Projects (as of August, 2007) [7]



[7] Source: Black Duck Software, August 2007

similarly long tail. Some 50 open source projects are the most used projects in the open source world, while thousands of open source projects are comparatively unpopular. But even less popular project code is reused on occasion, so information about it must be included in an open source genome project.

Why Care About What Is in Your Code?

Over 80 percent of IT organizations reuse software components[8], and if yours is like most organizations, chances are good that you have at least some open source within your code. In a recent survey of software developers[9], 77 percent said they used open source libraries, 60 percent used snippets of open source project code, and 50 percent used Linux.

But why should you care what open source is in your code? Because understanding the ancestry and attributes of open source in your code can help you assess and improve the code's stability and security, assure you are complying with export requirements for encryption, and determine the applicable license.

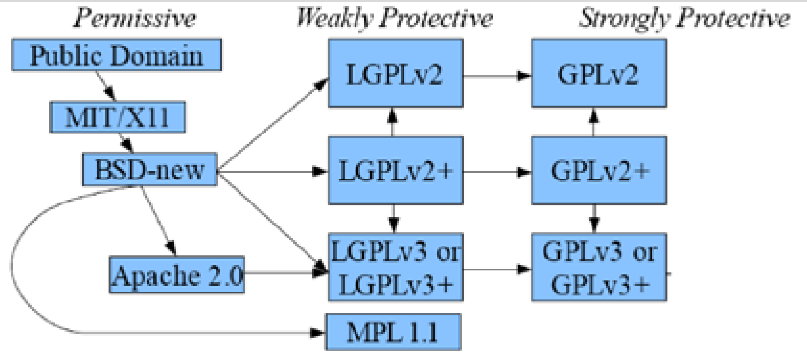
To assess a piece of code's stability it is helpful to know if it is from a reputable source, if it has known incompatibilities with other code, and if it is supported. Should code contain security vulnerabilities, it is useful to identify those vulnerabilities and find out where to patch them. It is also helpful to know when software upgrades are available, and where to find troubleshooting information should problems arise.

Identifying encryption components in your software helps help you ensure that your products comply with export laws. Knowledge of the encryption algorithms can enable you to reduce your business risks, reach distant markets faster, and open new international markets.

When code is reused, multiple licenses may come into play, with some license types imposing more significant obligations than others. This attribute is similar to dominant genetic traits trumping recessive traits in humans, e.g. if a gene for brown eyes is inherited from one parent and a gene for blue eyes is inherited from the other parent, brown eyes dominate. Even though the gene for blue eyes is still present, the gene for brown eyes will control. Knowing which licenses apply to the open source reused in your software product helps you understand the obligations associated with your code.

Figure 6 shows how licenses range from permissive licenses with few restrictions to strongly protective licenses with more demanding restrictions. When multiple licenses are in play, weakly protective licenses such as LGPLv2 dominate permissive ones like BSD, and strongly protective licenses such as GPLv2 dominate both permissive and weakly protective ones.

Figure 6. Free Libre/Open Source Software (FLOSS) License Examples [10]

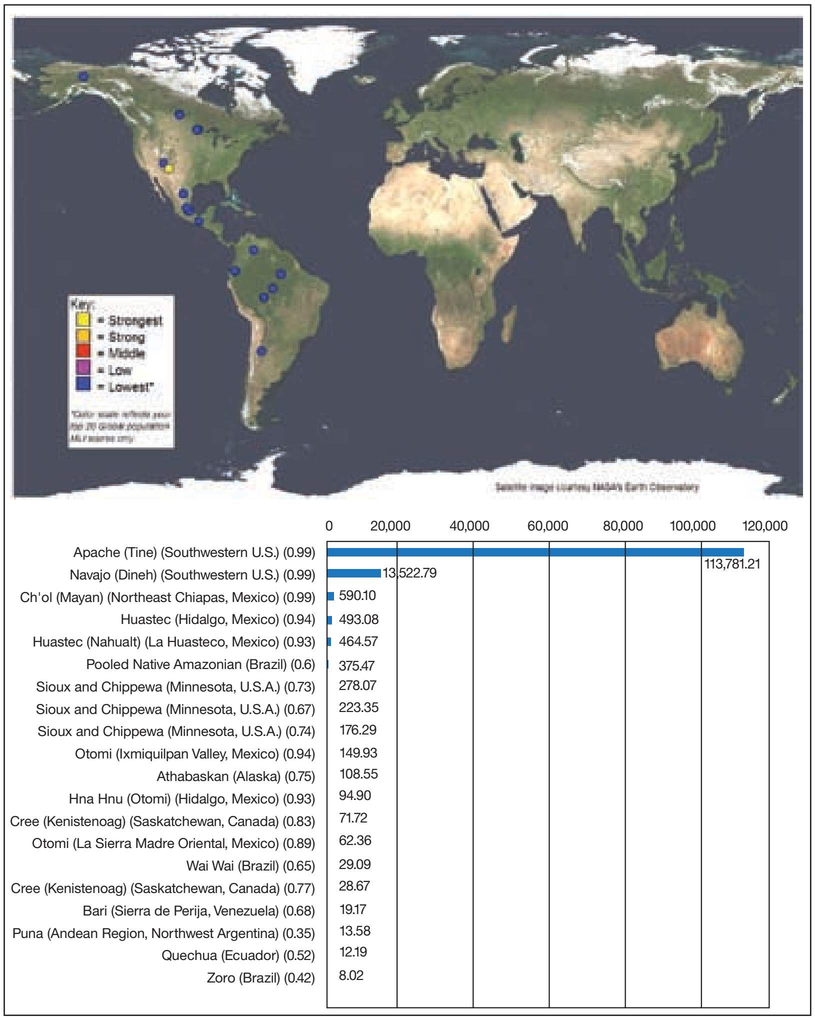


Determining Your Code's Ancestry

Determining your code's ancestry is similar to determining your own ancestry. If complete records exist, you can determine your ancestry from official documents, though even an "official" ancestry may be inaccurate. DNA analysis, such as the one in Figure 7 that shows a sample individual who can trace his ancestry back to the Apache Nation, provides a more dependable account of ancestral links.

[8] Source: Gibson Marketing Group, June 2007
[9] Source: Black Duck Software, 2006
[10] David A. Wheeler, The Free-Libre /Open Source Software (FLOSS) License Slide, July 2007. Available at http://www.dwheeler.com/essays/floss-license-slide.pdf

Figure 7. Sample Human DNA Analysis [11]



Human DNA analysis relies on a vast database containing genetic information about indigenous populations all over the globe, against which your DNA is compared. A software DNA analysis requires an analogous database for vast amounts of open source.

Establishing your code's ancestry can be challenging because, as Figure 5 shows, code from some open source projects is reused and even cloned in its entirety thousands of times creating a potentially huge set of potential software DNA matches. Given pervasive open source reuse and a proliferation of code versions, similar software DNA sequences sometimes exist in tens of thousands of places.

A comprehensive software DNA database helps you determine two important ancestral attributes. First, you can determine where your code left its family tree. And second, you can identify its origins.

To identify where your code left its family tree, you need to look for the largest overlap between your software DNA sequences and those within the software DNA database. A complete sequence overlap clearly marks the place where the code left the open source universe to be incorporated into your code "island". But complete overlap may not be found, in which case the analysis needs to look at the degree of overlap. The higher the degree of overlap between software DNA sequences in the database, the higher the certainty of locating the jump-off point. The concept of code leaving the open source universe to be incorporated into a code "island" is analogous to early Polynesians leaving South East Asia to live on remote Pacific islands and proceeding down a genetic path distinct from the population they left behind.

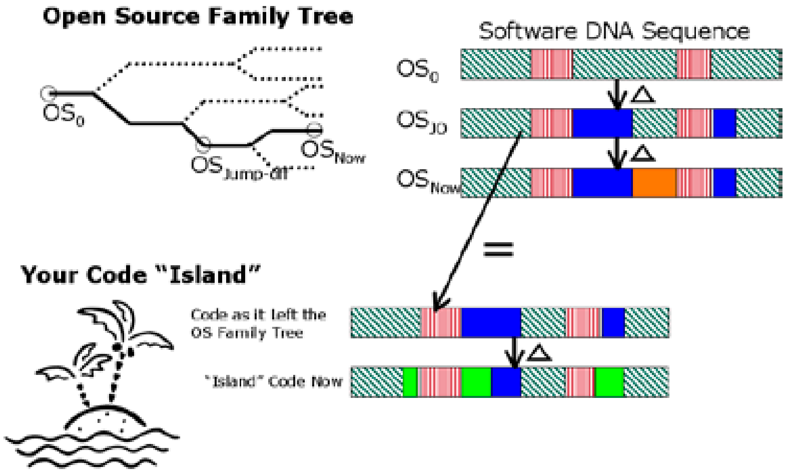
Conversely, because open source project code is reused so much (as Figure 5 shows), complete sequence overlap may occur in many places within the software DNA database. When popular code such as GNU Automake is reused without alteration, an "open source geneticist" must analyze instances of identical DNA sequences and mark the "authoritative" version within the database.

To identify your code's origins it is vital to know how code changes over time, as Figure 8 illustrates. Each code generation incorporates new changes, so your code's ancestors (e.g. OS0 in Figure 8) will not contain the complete set of the changes that your code contained when it left the family tree because those changes came later in the code's evolution. Likewise, code that continued to be part of the open source family tree after your code left to inhabit your code island (e.g. OSNow) will have changes that your code will not include. It is important to note that once

[11] http://www.dnatribes.com/sample-results/dnatribes-sample-apache.pdf

embedded into your product, your developers are likely to change the code to best meet your needs, thus your code will continue to evolve (e.g. “Island” Code Now).

Figure 8. Example of Code Evolution



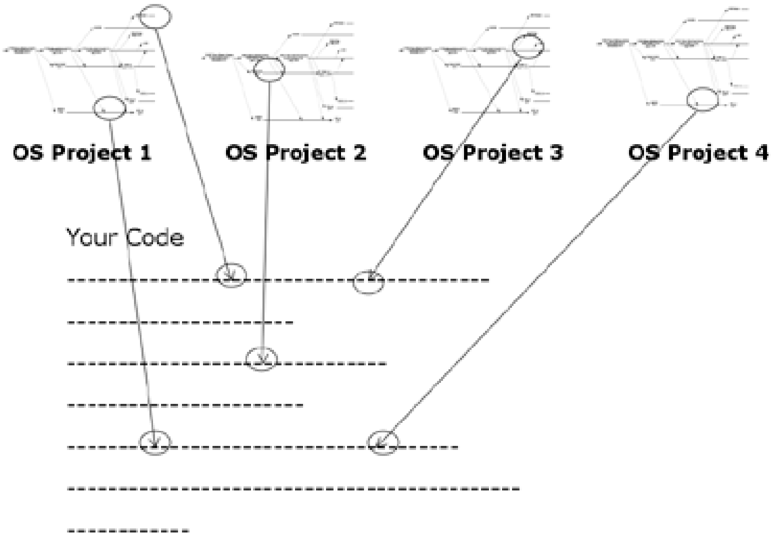
To add to the inherent complexity of software DNA analysis, chances are that open source from a wide variety of sources has wended its way onto your code island. Figure 9 illustrates how software DNA from many sources may now rub shoulders on your code island. For this reason you cannot determine origins based on simple measures such as the size and frequency of the DNA sequences found because such a statistical approach cannot yield sufficiently precise results.

For precise results it is important to identify non-overlapping software DNA sequences and highlight the superset of software DNA sequence matches found. Using our Figure 8 example, the superset of software DNA sequence matches are represented by the patterns and colors the “island” code now contains that match the patterns and colors of the open source code. Such sequence matching is best accomplished using a set theory-based methodology (think of the Venn diagrams you learned about in junior high), rather than statistically-based methodology. An example of a set theory-based approach is the “Precision” technology employed by Black Duck Software.

Although a laudable goal, it is infeasible to document all the code iterations that ever lived in the open source universe. There are a many reasons for this. Some open source was published before the Web existed and is unavailable or inaccessible. Even many UNIX tools do not have historical versions available. In addition, Web sites can be removed, code can be removed from Web sites for a variety of reasons, books can go out of print, and a company may change an open source project into a proprietary product causing open source project history to be lost.

Thus, there will be missing links or even missing family trees for certain software DNA sequences. But a system that builds on a learning approach that allows users to feed information into the database gets as close as practical to the ultimate goal of a complete database.

Figure 9. Sample Open Source Origins



A well conducted software DNA analysis should provide contextual information to help you make decisions about the open source elements within your software. Because no database is perfect, there will be ambiguous cases where you lack sufficient data to know precisely where code left the open source universe to inhabit your code island. In such cases,

you may want to call in a “software geneticist” to help you with such tasks as determining which software licenses are likely to apply.

Black Duck and the Open Source Genome Concept

Black Duck Software applies the open source genome project concept to code analysis. Black Duck’s KnowledgeBase contains software DNA sequence information for more than 180,000 open source projects. The Black Duck KnowledgeBase builds on automated information gathering augmented by the specialized knowledge of our experienced development staff.

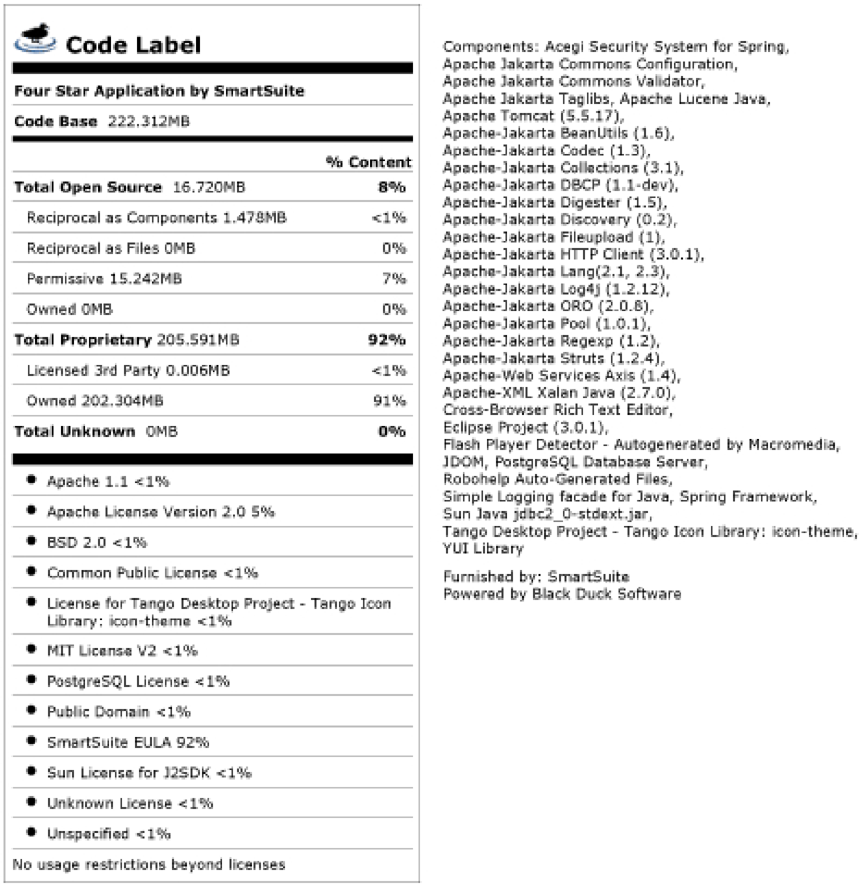
The unprecedented volume of software DNA information amassed in the KnoweldgeBase combined with Black Duck’s “Precision” set theory methodology produces a previously unattainable degree of software matching precision. In addition to software DNA analysis, the KnowledgeBase provides decision support, intellectual property management support, and detailed information about which software licenses dominate within your code.

The software DNA information Black Duck supplies can help you identify original versus open source code within your software, manage increasingly complex software licensing obligations, identify cryptographic elements in your source code to ensure export regulation compliance, and help you identify third party code within your software. In the case of mergers or acquisitions, Black Duck’s software DNA analysis helps determine the genetic makeup of the potential acquisition software, which in turn helps assess the value and potential risk associated with the acquisition.

Black Duck’s software DNA analysis can help you trace your code’s ancestry to the open source projects from which the code originated. Figure 10 shows a summary of one such analysis. Black Duck’s reporting capabilities provide you with as much detailed information as you need in easily digestible form, and Black Duck provides decision support to help you reach the most accurate conclusions.

Once you determine where within the open source gene pool code originated, Black Duck also helps you track its history. If needed, Black Duck’s “open source geneticists” are also available to help you.

Figure 10. Sample Open Source Analysis [12]



Code Label

Four Star Application by SmartSuite

Code Base 222.312MB

	% Content
Total Open Source 16.720MB	**8%**
Reciprocal as Components 1.478MB	<1%
Reciprocal as Files 0MB	0%
Permissive 15.242MB	7%
Owned 0MB	0%
Total Proprietary 205.591MB	**92%**
Licensed 3rd Party 0.006MB	<1%
Owned 202.304MB	91%
Total Unknown 0MB	**0%**

- Apache 1.1 <1%
- Apache License Version 2.0 5%
- BSD 2.0 <1%
- Common Public License <1%
- License for Tango Desktop Project - Tango Icon Library: icon-theme <1%
- MIT License V2 <1%
- PostgreSQL License <1%
- Public Domain <1%
- SmartSuite EULA 92%
- Sun License for J2SDK <1%
- Unknown License <1%
- Unspecified <1%

No usage restrictions beyond licenses

Components: Acegi Security System for Spring, Apache Jakarta Commons Configuration, Apache Jakarta Commons Validator, Apache Jakarta Taglibs, Apache Lucene Java, Apache Tomcat (5.5.17), Apache-Jakarta BeanUtils (1.6), Apache-Jakarta Codec (1.3), Apache-Jakarta Collections (3.1), Apache-Jakarta DBCP (1.1-dev), Apache-Jakarta Digester (1.5), Apache-Jakarta Discovery (0.2), Apache-Jakarta Fileupload (1), Apache-Jakarta HTTP Client (3.0.1), Apache-Jakarta Lang(2.1, 2.3), Apache-Jakarta Log4j (1.2.12), Apache-Jakarta ORO (2.0.8), Apache-Jakarta Pool (1.0.1), Apache-Jakarta Regexp (1.2), Apache-Jakarta Struts (1.2.4), Apache-Web Services Axis (1.4), Apache-XML Xalan Java (2.7.0), Cross-Browser Rich Text Editor, Eclipse Project (3.0.1), Flash Player Detector - Autogenerated by Macromedia, JDOM, PostgreSQL Database Server, Robohelp Auto-Generated Files, Simple Logging facade for Java, Spring Framework, Sun Java jdbc2_0-stdext.jar, Tango Desktop Project - Tango Icon Library: icon-theme, YUI Library

Furnished by: SmartSuite
Powered by Black Duck Software

[12] Source: Black Duck Software

Conclusions

Although documenting all open source projects is a tall order, implementing the open source genome concept can be enormously helpful in establishing your code's ancestry. For success, such an open source genome must encompass the depth and breadth of open source projects over time, and must also include "missing link" open source projects that may have died leaving their software DNA behind in other projects or in books.

For a number of years, Black Duck Software has embraced the open source genome project concept, and has documented software genomes for hundreds of thousands of open source projects. Such an effort is never finished, but Black Duck believes that its approach and progress enable it to provide the best information about open source in your code and also about what that information means for your organization.

blackduck®

Contact

To learn more, please contact:
sales@blackducksoftware.com
or call +1 781.891.5100

Additional information is available at Black Duck's web site:
www.blackducksoftware.com