Register | Login Search

Intellectual Property Today™

RFC Express™ New IP Lawsuits

Home | Issues | News | Classified | Jobs | Reports | Poster | Subscribe | Links | Contact | RFC Express

Measuring Changes in Software IP

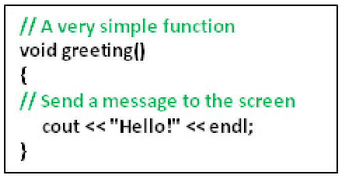
By Nik Baer and Bob Zeidman of Zeidman Consulting

As software developers we have often asked ourselves, “What is our software worth? ” As expert witnesses in IP litigation, we have often been asked the same question of our client’s software. The factors that go into determining software value are numerous and varied, including the amount of effort involved in developing the code, amount of time debugging the code, the complexity of the code, the degree of expertise required, the selling price of the finished program, and the size of the market for the final product. Quantifying the source code itself is only part of the valuation process, and there are many good methods to use.

Sometimes, though, it can be useful to measure not the absolute value of software, but the amount of change in the software and the intellectual property it represents. In a recent tax case, the taxes owed by a software company rested on the value of a current version of software with respect to the value of the initial version of the software created some years back. While financial analysts argued about the value of that initial software and the IP that the code embodied, we devised a means to measure the changes in the software from one version to another. The measure is called Changing Lines of Code or “CLOC.” An estimated valuation of the changes was then calculated by combining the CLOC measurements with the monetary value of the initial IP.

Traditional Methods

The source code of a program consists of lines of human-readable statements that tell the computer what to do and nonfunctional comments that are used to document what the code is doing. An example of computer source code showing functional lines in black and nonfunctional comments in green is given in Figure 1.



```
// A very simple function
void greeting()
{
// Send a message to the screen
   cout << "Hello!" << endl;
}
```

Figure 1: Sample source code

Traditionally, the evolution of software systems has been measured by counting the lines of code. The “source lines of code” (SLOC) measurement simply counts the number of non-blank lines of source code (“LOC”) in a program including functional and nonfunctional code. Mostly, the SLOC measures can compare two completely different programs and give a very rough comparison of the relative efforts involved. For example, if one program has ten times the SLOC as another program, we can reasonably conclude that the first program is more complex and required more people and/or time to develop.

However, these measures are much too rough to provide insight into the changes in code between subsequent versions of a single software project.

How Source Code Evolves

In order to understand the CLOC method and its benefit, it is necessary to first understand how the source code of a software project evolves. Figure 2 shows how the LOC and files of a software project evolve from an original version to a subsequent version. Some of the original files may get removed. Other files may continue without any change to the LOC inside. There may also be files that continue from the original version but do have changes inside. The modified continuing files are files that continue from the original version but are composed of a mix of unchanged LOC, changed LOC, and new LOC. Finally, there may be new files that are completely composed of new LOC.

- Blogs
- Executive Search
- Expert Witness
- File Wrappers
- For Sale
- IP Law Firms
- Jobs
- Office Space
- Patent and TM Searches
- Patent Drafting
- Patent Enforcement
- Patent Translations
- Referral Services
- Sequence Searching/Listing
- Situations Wanted

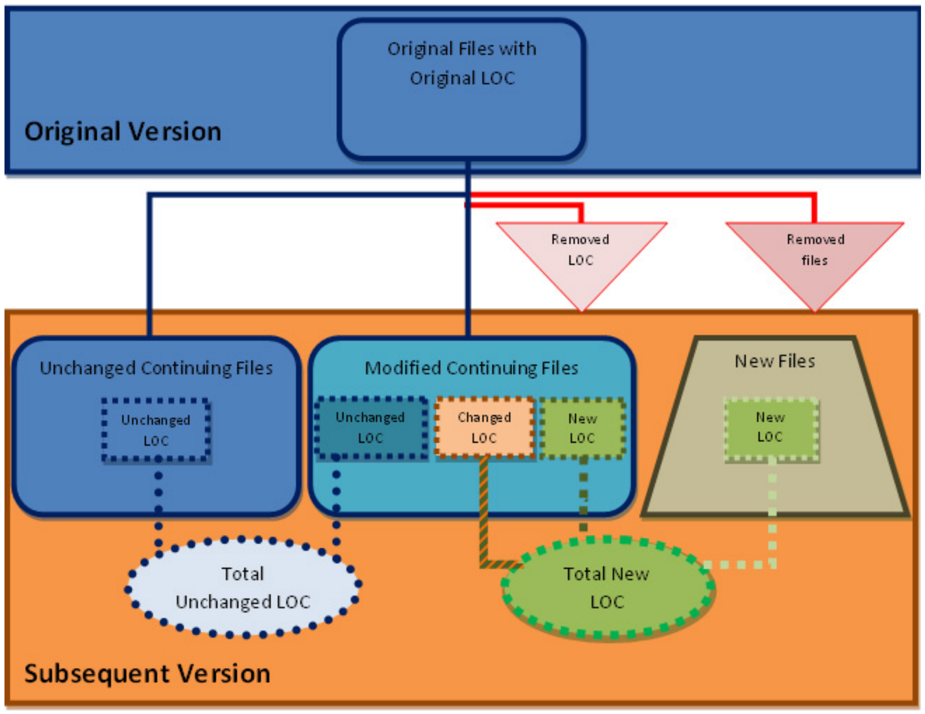


Figure 2: The evolution of files and LOC in software projects

In lay terms, source code evolves much as a book or an article evolves. During each revision lines are added, edited, and deleted. This comparison illustrates why simply counting SLOC is not accurate enough to measure software evolution. One can reasonably conclude that a 400 page novel with 100,000 sentences takes more effort than a two page article. But what about the evolution of that novel? Perhaps the first draft was 399 pages and 99,750 sentences long. The SLOC method would measure a .25% change between the first and final draft. This might seem reasonable at first, but what about all of the lines that were changed or removed? Perhaps entire chapters consisting of 60,000 sentences were changed or removed and other chapters of 10,250 sentences were added during the editing of the novel. Those edits represent a significant amount of change and effort on the part of the author, and at .25% a simple SLOC comparison grossly underestimates this change.

Changing Lines of Code Measure (CLOC)

The CLOC method was designed to measure the intricate changes involved in the evolution of a software project from one version to another. The CLOC method tallies the number of lines of code that have been added, changed or remain unchanged, just as one might count the number of sentences of a novel that have been added, changed, or remain unchanged. These measurements are then analyzed to express the change in software in terms of either growth of new code or decay of the original code.

Returning to the book comparison, after all the editing the change in terms of growth of new lines would be a much more reasonable 70% according to the CLOC method.

Measurements

The CLOC method combines CodeSuite® tools from Software Analysis & Forensic Engineering Corporation (S.A.F.E.) and a specially developed spreadsheet to evaluate the evolution of software. CodeSuite® is a patented collection of software analysis tools that includes BitMatch®, CodeCross™, CodeDiff®, CodeMatch®, SourceDetective®, FileCount™, and FileIsolate™. The CLOC method uses the CodeDiff and FileCount tools.

FileCount is used to count the number of files and non-blank lines in the software project's directory tree. The source code involved with a software project is often stored in files of specific program file types. Therefore, it is important to only count the files, and non-blank lines from the specific program file types. FileCount can be set up to only examine files of selected program source code file types.

CodeDiff is used to compare files from the original version to subsequent versions of a software project. In the CLOC method the CodeDiff comparison is limited to only comparing files with the same name. Typically file names are not changed from version to version, and a movement of source code between files or a file name change represents work being performed. CodeDiff is set up to only examine the same specific program source code files types as FileCount.

Using CodeSuite a distribution report is then created automatically. The distribution report is a spreadsheet containing statistical information about the files and LOC that were analyzed. The data from the CodeDiff distribution report is combined with the FileCount data in a CLOC spreadsheet to generate the software evolution results. An example of a CLOC spreadsheet that has been populated with data from an analysis of the popular Mozilla Firefox web browser is shown in Table 1 below. The calculated data elements are shown in bold.

Data from Firefox browser	**Version .1**	**Version 1.0**	**Version 2.0**	**Version 3.0**
Total Files ("TFn")	10,302	10,320	11,042	9,429
New Files ("NF")	0	175	1,455	2,399
Total LOC ("TLn")	3,169,530	3,180,268	3,570,712	3,288,983
Total New LOC ("TNL")	0	40,922	843,683	1,530,918
Total Continuing Files				

(in Subsequent Version) ("TCF")	10,302	10,145	9,587	7,030
Total LOC in Continuing Files in Subsequent Version) ("TLCF")	3,169,530	3,148,460	3,125,785	2,288,903
New LOC in Continuing Files, Not in Original Files ("NLCF")	0	9,114	398,756	530,838
Continuing LOC, (Present in the Original Files and in the Continuing Files) ("CL")	3,169,530	3,139,346	2,727,029	1,758,065
Modified Continuing Files ("MCF")	0	281	8,577	6,543
Unchanged Continuing Files ("UCF")	10,302	9,864	1,010	487
Growth(n): Total New LOC(n) / Total Original LOC(0)	0%	1%	27%	48%
File Decay(n): Total Continuing Files(n) / Total Files(n)	100%	98%	87%	75%
Line Decay(n): Continuing LOC(n) / Total LOC(n)	100%	99%	76%	53%
Unchanged File Decay(n): Unchanged Continuing Files(n) / Total Files (n)	100%	96%	9%	5%
SLOC Growth(n): (TL(n) - TL(0)) / TL(0)	0%	0%	13%	4%

Table 1: Firefox browser CLOC analysis spreadsheet

The software evolution results are expressed by four different data elements as enumerated below.

Growth

The growth of each version is the ratio of total new LOC ("TNL") to the total LOC ("TL") in the original version. The TNL is the number of LOC that do not exist in the original version, including lines that have changed along with completely new lines in either continuing files or completely new files. The TNL is represented by the green oval in Figure 1 above. The ratio of TNL per version divided by the original total LOC is expressed as a percentage.

Growth(n) = **TNLn** / *TL0*

LOC Decay

It can also be important to know how the original code and the IP it represents has decayed as a software project evolves. The LOC Decay is the ratio of total continuing LOC ("CL") to the total LOC in each subsequent version. The total continuing LOC is the count of LOC that are present in the original version as well as in the files that have continued into the subsequent version and is show as the blue oval in Figure 1 above.

Line Decay(n) = **CLn** / *TLn*

File Decay

The decay can also be represented in terms of either the file decay or the unchanged file decay. The file decay is ratio of original files that are still remaining ("TCF") to the total number of files in the subsequent version ("TF"). The unchanged file decay is the ratio of the continuing files that are unchanged in the subsequent version ("UCF"), to the total number of files in the subsequent version.

File Decay(n) = *TCFn / TFn*

Unchanged File Decay(n) = **UCF**n/ *TFn*

Measured Results

The graphical representation of the traditional SLOC and new CLOC measurements of the Mozilla Firefox software project in Figure 3 shows the superiority of the CLOC method to the traditional SLOC method. It is safe to assume that the Firefox project is continually evolving. The CLOC measurement correctly shows a steady growth of the Firefox software project. In contrast the SLOC measurements rise and fall. The SLOC method only measures the total LOC of each version instead of the dynamically changing total new LOC that the CLOC method measures, so it does not accurately portray the growth and evolution of work being done to this popular open-source project.

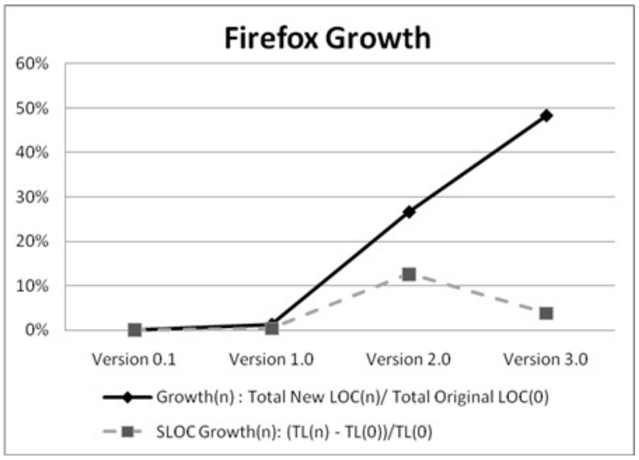


Figure 3: CLOC growth vs. SLOC growth of Mozilla Firefox software

Conclusion

The CLOC method was devised to provide an accurate measurement of the change in a software project from one version to another, which can then in turn be used as an accurate base for various valuations. Since its creation the CLOC method has been used in a large tax case to help calculate the value of the source code through different versions of software projects, and has been presented at an academic conference. By focusing on the dynamic changes between subsequent versions of a software project CLOC is able to accurately measure the growth of the source code as well as the decay of the original version of code relative to the increasing subsequent code. These measurements of the change in code and the IP it embodies can be combined with other variables to provide accurate valuations.

About the authors

Nikolaus Baer is a research engineer at Zeidman Consulting. He has written articles and given presentations about software trade secret theft and how to analyze source code. He holds a bachelors degree in computer engineering from UC Santa Barbara, where he attended on a Regents Scholarship.

Bob Zeidman is the founder and president of Zeidman Consulting. He is the author of several engineering textbooks and holds five patents. He holds a Master's degree from Stanford University and two Bachelor's degrees from Cornell University.