Software Piracy Forensics: The Need for Further Developing AFC

S. Santhosh Baboo[1] and P. Vinod Bhattathiripad[2,*]

[1] Reader, P G & Research Dept. of Computer Science, D.G. Vasihanv College, Chennai, India
santhos2001@sify.com
[2] Consulting Cyber Forensic Engineer, Calicut, Kerala, India
vinodpolpaya@gmail.com

Abstract. Among all the available approaches for software piracy forensics, one existing and exceptional approach is the theoretical frame work called AFC (Abstraction-Filtering-Comparison), an accepted approach in US courts for evaluating copyright infringement claims involving computer software. Through this paper, the authors would like to approach AFC in a threefold manner: One, to discuss the nature and efficacy of AFC; two, to recount some existing observations on it, and three, to identify areas, if any, where there is scope and need for appropriate modifications to further increase the efficacy and validate the legitimacy of the AFC approach, and in particular from the view point of a researcher who believes that software intelligence offered by the automated tools for software piracy investigation needs to be supplemented with manual intelligence for making the expert report more judiciary-friendly.

Keywords: Piracy, Post-piracy modifications, Copyright infringement, Software piracy forensics, AFC, MIS forensics, Abstraction-Filtration-Comparison, Nuggets, Computer Associates vs. Altai case, Scènes a faire.

1 Introduction

In any court case, the technical expert's evidence is open to legal challenge and such challenges, irrespective of the outcome, might delay the process of litigation. Hence, it is both crucial and necessary for the expert's report to be as thorough, authentic and convincing as possible in the interest of proper justice. Also, it is important for the expert to preempt any delay by making the report as comprehensive and complete as possible. This is all the more important in software piracy litigation where a cyber forensic expert (technical expert) is the person who is often designated by the judge to compare the pirated[1] with the original[2] and prepare a report that is legally convincing and binding. Although software tools are available to help them to carry out this cyber forensic assignment, the experts need to supplement the software intelligence with

[*] Research Scholar, Bharathiar University, Coimbatore.
[1] Throughout this article, pirated means the allegedly pirated software.
[2] Throughout this article, original means the version of the software that the complainant submits to the law enforcement agency for software piracy forensics.

human intelligence, insight and common sense so as to produce and present a report as unchallengeable as possible[3].

The natural modus operandi of the expert is to compare the original and pirated software by juxtaposing the two [1]. The task of comparing two software packages arises usually when one party lodges a complaint of software piracy against another. In order to perform this task, several software tools are used to assess software piracy and these tools are based mostly on academically accepted mathematical techniques and theoretical frameworks like Discourse Analysis [2], SMAT [3], MOSS [4], IDENTIFIED [5], SCAP [5,6], and AFC [7]. All these tools are capable in their respective areas of software piracy investigation but the inherent complexity in software programming logic and underlying global commonalities in it demand a vital supplementing role for cyber forensic expert's intelligence, expertise, common sense, insight and perhaps even intuition. So, even if there are automated approaches to software comparison, there is still scope and space for a comprehensive approach (for manually comparing two software packages), which utilizes the expert's intelligence, expertise, common sense and insight. While it is true that automated comparison ensures certain level of consistency, reliability and thus, integrity of the result, manual comparison, which supplements and not replaces automated comparison, would give the expert scope to add value to the result. Such a value added report could be useful to the judge.

A manual approach of comparing two software packages needs to address comparison issues related to various parts of the software like source code, object code, databases, and fingerprints. Source code and object code comparisons need to be performed along various sub-parts (of source code, object code, databases, and fingerprints) like data structures, algorithms, system calls, errors, language, formatting, macros, comment styles, spelling, grammar, use of language features, execution paths, bugs and metrics [8]. All these areas of comparison provide enough challenge for the software piracy forensics researcher who works with the objective of formulating a methodology for manually comparing two software packages for adding value to the automated result of the same comparison.

Any such researcher cannot stay away from recounting AFC (Abstraction-Filtering-Comparison) because of AFC's judicial acceptance as a manual approach. AFC test was primarily developed by Randall Davis of the Massachusetts Institute of Technology for evaluating copyright infringement claims involving computer software and used in the 1992 Computer Associates vs. Altai case, in the court of appeal of the 2nd federal circuit in the United States [7]. Since 1992, AFC has been recognized as a legal precedent for evaluating copyright infringement claims involving computer software in several appeal courts in the United States, including the fourth, tenth, eleventh and federal circuit courts of appeals [9,10,11]. Even though AFC is basically a manual comparison approach, there exists software tool by name SIMILE workshop, which was developed based on AFC's theoretical frame work in 2007 by the French firm, European Software Analysis Laboratory (EsaLab) with additional inputs from Institut d'´electronique et d'informatique Gaspard-Monge – IGM, University of Marne-la-Vall´ee, France [9]. EsaLab claims that SIMILE Workshop is the only software tool (available in the market) based on AFC [9]. All these confirm

[3] Our sincere gratitude to the reviewers of this paper for their valuable suggestions.

AFC's fitness both as a manual approach and as a theoretical foundation for a software tool for establishing copyright infringement claims involving computer software and so, any researcher (who works with the objective of formulating an advanced methodology for manually comparing two software packages) cannot avoid AFC.

The main objective of this paper is basically threefold[4]: One, to discuss the nature and efficacy of AFC; two, to recount some existing observations on it, and three, to identify areas, if any, where there is scope and need for appropriate modifications to further increase the efficacy and validate the legitimacy of the AFC approach, and in particular from the view point of a researcher who believes that software intelligence offered by the automated tools for software piracy investigation needs to be supplemented with manual intelligence for making the expert report more judiciary-friendly.

2 The AFC Test

The theoretical framework of AFC not only makes possible the determination of "literal" similarities between two pieces of code, but it also takes into account their functionality to identify "substantial" similarities [9]. In the AFC test, both the pirated as well as the original software are put through three stages, namely, abstraction, filtration and comparison. While other approaches (and the automated tools based on these approaches) generally compare two software packages literally, without considering globally common elements in the software, AFC, as the name indicates, first abstracts the original as well as the allegedly pirated, then filters out the globally common elements in them to zero in on two sets of comparable elements and finally compares these two sets to bring out similarities or "nuggets" [9].

The task of 'abstracting' software is not very easy to perform. In the 1992 *Computer Associates v. Altai* case [12], the abstraction was described according to the following 6 levels: (i) main purpose of the code, (ii) program structure or architecture, (iii) modules, (iv) algorithms and data structures, (v) source codes, (vi) object codes. "At the lowest level of abstraction, a computer program may be thought of in its entirety as a set of individual instructions organized into a hierarchy of modules. At a higher level of abstraction, the instructions in the lowest- level modules may be replaced conceptually by the functions of those modules. At progressively higher levels of abstraction, the functions of higher-level modules conceptually replace the implementations of those modules in terms of lower-level modules and instructions, until finally, one is left with nothing but the ultimate function of the program.... A program has structure at every level of abstraction at which it is viewed. At low levels of abstraction, a program's structure may be quite complex; at the highest level it is trivial" [12, p.13]. That means, in the absence of specific standard procedure, abstraction of hundreds of thousands (sometimes even millions of) lines of codes, presented with high degree of complexity, is a "prodigious undertaking" [13, p.4], even for a computer professional.

Just as abstraction, the task of filtering out the variety of globally shared (and hence non-protectable) expressions generally found in any software is also not an easy task. The AFC-test specifies three categories (more aptly, levels) of exclusions,

[4] Our sincere gratitude to Dr. P.B. Nayar, Lincoln University, UK, for his help in articulating our thoughts.

called doctrines [9]. Firstly, if there is only one way of effectively expressing an idea (a function), this idea and its expression tend to "merge". Since the idea itself would not be protectable, the expression of this idea would also escape from the field of the protection. Secondly, there is the "scènes a faire" doctrine which excludes from the field of protection, any code that has been made necessary by the technical environment or some external rules imposed on the programmer. Thirdly, there are those elements that are in the public domain. Thus, carrying out the task of filtration is also not very easy unless there is a standard procedure to filter out the variety of globally shared expressions generally found in any software and thus to zero in on the software elements that form the basis for the software piracy investigation.

Infallibility of AFC. The infallibility of the AFC approach has not stayed unquestioned. One of the best observations on AFC is the amicus brief [13] submitted to the United States court of appeals for the second circuit. This amicus brief was filed in 1996 by Mark M. Arkin, Counsel on behalf of five computer scientists, Roy Campbell (University of Illinois), Lee A. Hollaar (University of Utah), Randall Davis (the inventor of AFC), Gerald J. Sussman, and Hal Abelson, (all three of Massachusetts Institute of Technology), who believed that there was uncertainty among courts in how to implement the AFC for evaluating copyright infringement claims involving computer software. This legal document describes the technical complexity and difficulty of performing the AFC test "from the standpoint of computer science", which these five computer scientists have experienced in their role as experts. The amicus brief observes that "Performing AFC test in the context of litigation is a challenging task, in no small measure because of the technical difficulties that arise. These technical difficulties stem from (1) the sheer magnitude of the task of analyzing programs that routinely consist of hundreds of thousands of lines of computer code, (2) the lack of any fixed or agreed-upon set of levels of abstraction by which to describe a program, (3) the interaction of legal doctrines (such as merger, scenes a faire, and public domain) with the technical constraints of the computer industry and (4) the rapid evolution of these doctrines in the areas of computer software" [13, p. 4]. Lee A. Hollaar, in a later work [14], has further emphasized the supplementary role of the expert's insight and commonsense in the interpretation of the AFC. Our principal contention here is that this need should be looked at more carefully by upgrading its status from supplementarity to essentiality. The expert's input is also vital in the further development and modification of AFC.

The amicus brief further observes that the filtration process also requires substantial technical expertise both to draw a line between idea and expression and to determine whether the expression in a body of code is necessarily incidental to the idea being expressed (merger). To make matters worse, most of the abstractions used to describe the software may not qualify for final comparison (that means, most of them get filtered out in the filtration stage) as most software, especially those for commercial purpose, are written using well-known ideas, designs and principles, because the known (well-tested) design principles are globally believed to be most reliable [13]. Even then, the amicus brief suggests that "despite the difficulty of the task, creating a set of abstractions and answering the technical questions raised in the filtration are still sensible and well-founded undertakings" [13, p. 6] and the task of abstraction should be necessarily carried out based on the control structure, data structures, data flow, information architecture and the textual organization of the code. Finally, this amicus brief suggests legal and technical procedures for applying the AFC test by courts in copyright claims involving software. While the suggested legal procedure may well be comprehensive and practical, the suggested technical procedure seems to be not capable enough to reduce the difficulty of abstraction and filtration and this difficulty remains as a limitation of AFC, even now. Even with this limitation, AFC continued to be an acceptable approach for US courts for comparing the original with the pirated and had been used in several subsequent suits on copyright infringement and trade secret misappropriation involving software [9].

Another observation (on AFC) worth mentioning is the one by honorable judge John M. Walker, Jr., through his article "Protectable 'Nuggets': Drawing the line between idea and expression in computer program copyright protection" [7]. Through this article, the judge explains the difficulties in drawing the line between the idea and expression in software. By observing that the group of three doctrines (listed above) "is not exclusive" and that "one might expect in future cases to see filtration occurring with respect to other program elements, including those listed in § 102(b) of the copyright act, such as procedures, process and methods of operation", the judge fears that "even under an expanded list of elements excluded by the filtration step, there could still be many "nuggets" of protectable expression left in a program" [7, p.83] and because of this, the filtration stage is a battleground where the attorney who is able to argue successfully that his client's program, after filtration, contains many elements of protected expression should be all the more likely to succeed on a claim of infringement. These observations of the judge also point to the limitation of the filtration stage.

Baboo and Bhattathiripad [1][15] explore an interesting point of how to deal with the increasing use of design patterns and programming patterns in general software piracy forensics. This is applicable to AFC too. While this presumably comes under some of the "doctrine" exceptions, to date it does not seem to have been explicitly examined. This is further discussed below, in the context of the limitations of AFC.

Firstly, AFC does not recognize or uphold the need to identify and filter out post-piracy modifications in original as well as in the allegedly pirated. There is a serious risk of the expert's opinion being challenged in the court if post-piracy modifications are not properly identified and filtered out of both original as well as allegedly pirated, before comparing these two [1]. During the post-piracy life span of the software, the original and pirated software are expected to grow functionally almost in the same direction but with stylistically different patterns. The possibility of different patterns of growth is a very valuable and useful dimension of study for the software piracy forensic expert. A proper manual investigation into the potential post-piracy modifications in the source codes, embedded images and finger prints, the database procedures and the database schemas of the pirated will contribute substantially to the reliability of cyber forensic investigation [1].

Secondly, AFC does not properly explain, how the increased-complexity of analyzing a program that is written by one programmer and updated at various instances later, by others programmers, is dealt with. As far as program logic is concerned, most programs are very likely to have the thumb impressions of many programmers due to the frequent turnover of employees in the software industry and these multiple thumb impressions, compatibly identified in both the 'original' and the 'pirated' sources, can prove to be vital in establishing piracy.

Thirdly, AFC does not offer a standard and judiciary-friendly format for presenting the result of comparison. Any judiciary system with a sense of responsibility and accountability would like to ensure that they be properly informed in matters that are outside their area of expertise [16, p.20]. This makes it incumbent on computer experts not only to provide the judicial system with such expertise but also to create general awareness for the proper utilization of such expertise. In actual terms, what the court might expect from a technical expert would be an informative report that is self-explanatory, jargon-free and non-esoteric in format. Thus, the format of expert's report becomes a matter of prime importance.

Further, it is also not clear how AFC deals with the problems of identifying programming errors and blunders in the original and of establishing their presence verbatim in the allegedly pirated. A programming error found in well-tested and implemented software can be a variable or a code segment or a field in a database table, which causes or produces wrong result during the execution of the program. A programming blunder found in well tested and implemented software can be a variable or a code segment or a field in a database table, which is hardly used or executed in the context of the application or the user's functionality but unlike an error, it will be harmless [15]. While trying to establish a variable name or a code segment as a programming blunder, the expert needs to confirm that it is (i) absent elsewhere in the original, (ii) present in the allegedly pirated exactly in the same context as it was found in the original and (iii) absent elsewhere in the allegedly pirated. Identifying programming errors and blunders in the original and establishing their presence verbatim in the allegedly pirated may additionally require expert's common sense, intuition and expertise.

One of the biggest limitations of almost all these theoretical frameworks (including AFC) is their inability to view software piracy investigation as a cyclic process. All the existing theoretical frameworks consider software piracy investigation as a linear sequential process. For instance, if an error in abstracting the software was found in the filtration stage of AFC, the need to return to the abstraction stage for necessary correction and the need for a possible modification of all the subsequent abstractions and filtrations are not clearly explained in any of these tools.

Not only the AFC test but also all the existing and established approaches and products would fail when the scope of investigation of the software piracy suit is expanded beyond comparing two software packages. While software tools, including SIMILE workshop, can yield results in software comparison, there are certain areas where only manual comparison can yield convincing result. For instance, if sundry complainant-specific data is found in the seized computer system of the alleged pirate, only manual investigation can bring this out as tangible evidence. [17]. One specific manifestation of this could be the existence of files related to the complainant's clientele or a back up copy of the live database of a client of the complainant found in the pirate's computer system. Such rare instances occur when the piracy was done by performing a disk copy (of the complainant's computer) that resulted in pirating not only the source code and data base related files but also other confidential data of the complainant [17]. The existence of such material in the seized memory is strong evidence of piracy as these are materials irrelevant for the functioning of the software and hence should not have been there at all. Another manifestation of this is a case where a UK-based software firm alleged that one of its (former) employees had appropriated a software product of his employers and was marketing it as his own even while he was still employed in the company, it was found that the alleged culprit had already created and maintained a website to market the pirated while still employed by this firm. This proof from the DNS server was enough to initiate a case in the court. None of the software piracy forensic tools has capacity to unearth such supporting evidence. This also shows that software intelligence needs to be supplemented with manual intelligence.

Where do we go from here? So, a step by step and less-complex manual procedure to compare two software packages is highly necessary for the technical experts to supplement software intelligence offered by the automated tools. Such a manual approach can be created either by further developing some existing and established procedure or by formulating a new procedure, which would cover the above mentioned limitations of abstraction and filtration. To suit the requirements of such an approach, AFC approach needs to and can be further modified.

In spite of all the advantages of AFC, there are two areas in AFC that require further explanation / modification for enhancing the effectiveness of AFC and they are abstraction and filtration. One can give clarity to the abstraction stage and thereby add value to it by further listing out various forensic investigation areas in the software namely functional area, reports generated, screens generated, menu structure, data structure, object code strings, documents, just to name a few (See also 18, pp. 317-25, and 19). Additionally, the abstraction stage also requires clear explanation as to how the abstraction process needs to be implemented in each area. The filtration stage, on the other hand, can be improved by providing an enhanced list of specific elements that are to be filtered out. The AFC list does provide a good guideline, but one needs to specifically bear in mind some of the weaknesses of the prevailing lists (for instance, absence of reference to post-piracy modifications). Above all these, the existing sequential process of AFC needs to be modified to incorporate the idea of cyclicality of the process so as to enable the expert to go back and correct errors at any previous stage and properly also attend to consequential changes in the subsequent stages in the software piracy investigation process. Finally, the addition of a totally new stage that will stress the need for presenting the results of the whole investigation in a judiciary-friendly, less-technical, jargon-less, and non-esoteric format that explain similarity, preferably in numerical terms is strongly advocated.

References

1. Santhosh Baboo, S., Vinod Bhattathiripad, P. (under editorial review): Software Piracy Forensics: Impact and Implications of post-piracy modifications. Digital Investigation - The International Journal of Digital Forensics & Incident Response (2010)
2. van der Ejik, P.: Comparative Discourse Analysis of Parallel texts, eprint arXiv:cmp-lg/9407022. Digital Equipment Corporation, Ratelaar 38, 3434 EW, Nieuwegein, The Netherlands, CMP-lg/ 9407022 (1994)
3. Yamamoto, T., Matsushita, M., Kamiya, T., Inoue, K.: Measuring Similarity of Large Software Systems Based on Source Code Correspondence. IEEE Transactions on Software Engineering, XX, Y (2004)

4. Lancaster, T., Culwin, F.: A Comparison of Source Code Plagiarism Detection Engines. Computer Science Education (2004), `http://www.informaworld.com/`
5. Li, C.-T.: Handbook of Research on Computational Forensics, Digital Crime, and Investigation: Methods and Solutions. In: Information Science Reference, ch. XX (2010), `http://www.info-sci-ref.com`
6. Frantzeskou, G., Stamatatos, E., Gritzalis, S., Chaski, C. E., Howald, B. S.: Identifying Authorship by Byte-Level N-Grams: The Source Code Author Profile (SCAP) Method. International Journal of Digital Evidence 6(1) (2007)
7. United States Court of Appeals Judge John Walker, Protectable Nuggests: Drawing the Line Between Idea and Expression in computer Program Copyright Protection, 44. Journal of the Copyright Society of USA 44(79) (1996)
8. Spafford, E.H., Weeber, S.A.: Software forensics: Can we track the code back to its authors? Purdue Technical Report CSD–TR 92–010, SERC Technical Report SERC–TR 110–P, Department of Computer Sciences, Purdue University (1992)
9. European Software Analysis Laboratory, The SIMILE Workshop: Automating the detection of counterfeit software (2007), `http://www.esalab.com`
10. Raysman, R., Brown, P.: Copyright Infringement of computer software and the Altai test. New York Law Journal 235(89) (2006)
11. United States District Court of Massachusetts, Memorandum and Order, Civil Action number 07-12157 PBS, Real View LLC. Vs. 20-20 Technologies, p. 2 (2010)
12. United States Court of Appeals, Second Circuit, Computer Associates International, Inc. v. Altai, Inc., 982 F.2d 693; 1992 U.S. App. LEXIS 33369; 119 A.L.R. Fed. 741; 92 Cal. Daily, Op. Service 10213, January 9, 1992, Argued, December 17, 1992, Filed (1992)
13. United States Court of Appeals, Second Circuit, Corrected Amicus Brief filed by Mark M. Arkin, Counsel of 5 computer scientists regarding software copyright and trade secret cases – Observation of Abstraction, Filtration and Comparison test, on appeal from the United States district court for the southern district of New York, on the suit Harbour Software Inc. Vs Applied Systems Inc., 97-7197L (1997)
14. Hollar, L.A.: Legal Protection Of Digital Information. BNA Books (2002)
15. Santhosh Baboo, S., Vinod Bhattathiripad, P.: Software Piracy Forensics: Exploiting Nonautomated and Judiciary-Friendly Technique'. Journal of Digital Forensic Practice 2(4), 175–182 (2009)
16. United States Court of Appeals, Second Circuit, Brief of Amici Curiae of 17 technical experts, in the case Universal City Studios, Inc., et al. Vs. Eric Corley, A/K/A Emmanuel Goldstein and 2600 enterprises (2001)
17. Vinod Bhattathiripad, P.: Judiciary-friendly computer forensics, Kerala Law Times, Part 13 & Index, June 29, p. 54 (2009)
18. Davis, R.: The nature of software and its consequences for establishing and evaluating piracy. Software Law Journal 5(2), 317–325 (1992)
19. Kremen, S.H.: Presentation of Technical Evidence by Experts in Computer Related Intellectual Property Litigation. Computer Forensics Online 2(1) (1998)