Similarities Between Computer Programs Codes and Copyright Infringement

by Washington Almeida

This article is the result of an extensive research work carried out at the specialization course in Law and Information Technology of the Polytechnic School of USP, classified by the WUR, the World University Ranking, among the best university in the world. This research work was conducted under the guidance of the Teacher and Doctor Manoel J. Pereira dos Santos, which resulted in obtaining the title of Specialist in Law and Information Technology with the maximum score.

The specialization course in Law and Information Technology is offered by the excellent and renowned university named Polytechnic School of USP (Escola Politécnica da USP – POLI/USP), which has the coordination by the excellent Teacher and Engineer Mr. Edson Gomi.

Acknowledgment:

My research work received a rich contribution from the extraordinary Teacher and Doctor Mr. Manoel Joaquim Pereira dos Santos, master in the subjects of copyright and intellectual property. Mr. Manoel is a professional of the best technical quality, with whom I had the privilege of having the guidance to my work. His example as person and professional, his commitment and high level of exigency, combined with his seriousness and technical competence, will always result in a product of better quality, wherever his considerations are present, and for whatever the work. With his extreme experience and

competence, he acts in the areas of Intellectual Property and Information Technology, Telecommunications and Internet, Special and Interdisciplinary Projects, Economic Law, Corporate Law and Foreign Investments, Contracts, Consumer Law, Civil and Commercial Litigation, and Labor Law.

To know more about this prestigious and respected professional Doctor Manoel J. Pereira dos Santos, visit his company's web site at the URI *www.santoslaw.com.br*.

1. Initial considerations:

The objective of this study is to evaluate until which point, technically, similarities in the code of two computer programs can be considered as copyright infringement. The theme is relevant because there is not a standard methodology of analysis in such cases where this kind of analysis is necessary, and the conclusions are always based on experiences varied and different perspectives of expert professionals around the theme.

In Brazil, computer programs are intellectual works protected by Law No. 9.610/98 of Copyright Law, and they are the object of Law No. 9.609/98, which deals specifically with the protection of the intellectual property of a computer program.

The treatment of copyright infringement established by Law No. 9.609/98 deals both with the issue of unauthorized copy of the computer program, also called in the vulgar form as software piracy, and with the question of plagiarism, but the term source-code is only referenced in cases of transfer of computer program technology in the sole paragraph of Article 11.

The computer program is defined by Law No. 9.609/98 as the expression of an organized set of instructions in natural or coded language, contained in physical support of any nature. The natural language is the one that is technically called the source-code, while the encoded language contained in physical support of any nature is the medium containing the programs ready for installation in the user's environment, and often this connotation of coded language is confused with the object-code. What the law does not specify is that the object-code is generated by a process in which the natural language is converted into coded language, in a process called compilation. It is important to emphasize that the

natural language suffers numerous changes with the inclusion of information in machine code, so that it is possible for the object-code to be understood by the processor. In other words, the source-code is the source of the intellectual development of the computer program, while the object-code is a specific instruction format, which results of a compilation to deal with the processor that will run the program. And the encoded language contained in physical media of any nature comprises the media containing the program ready for use.

The use of object-code for copyright infringement analysis occurs when companies use the argument that the source code is in danger of being copied and then treated as an industrial secret. It can also occur when object-code analysis is sufficient to detect counterfeiting.

The major problem exists when comparing two programs is necessary. This is because the program allegedly does not have identity or similarities in the lines of code, being necessary to analyze the architecture of the program and the other non-literal elements to determine the existence of plagiarism. It occurs that there is no clear definition of the assumptions in which a computer program code, which bears similarities to another code of another computer program, violates the computer program law. There are no elements in the form of a standard to gauge such similarity. The Law only defines, in art. 6, item III, the hypotheses in which the similarities do not constitute an offense against the copyright of a computer program.

There is not even literature available that supports and bases the definition of a scientific methodology for such analysis, when considering the work of judicial expertise. Therefore, many judicial reports and technical opinions, for the most part, report the conclusion of copyright infringement based on the experiences, perspectives of understanding and varied concepts that Judicial Experts and Technical Assistants have on the subject.

The motivation of the present work, as well as the central objective, is expected to fill this gap, with the intention of suggesting a methodology for such analysis, making a technical approach on the theme, and linking it, when necessary, to the aspects to which they relate.

In the approach, it is intended to maintain that the analysis of copyright infringement through plagiarism should be performed directly on the source-code of the computer program, not on the

object-code generated from it. Therefore, it is fundamental to understand and define the central object of our study, to justify the reason for our analysis from the source-code and not from the object-code, and where they are located within the flow of interactions in the development cycle of a program computer, to delimit the scope of the study, exclusively according to this perspective.

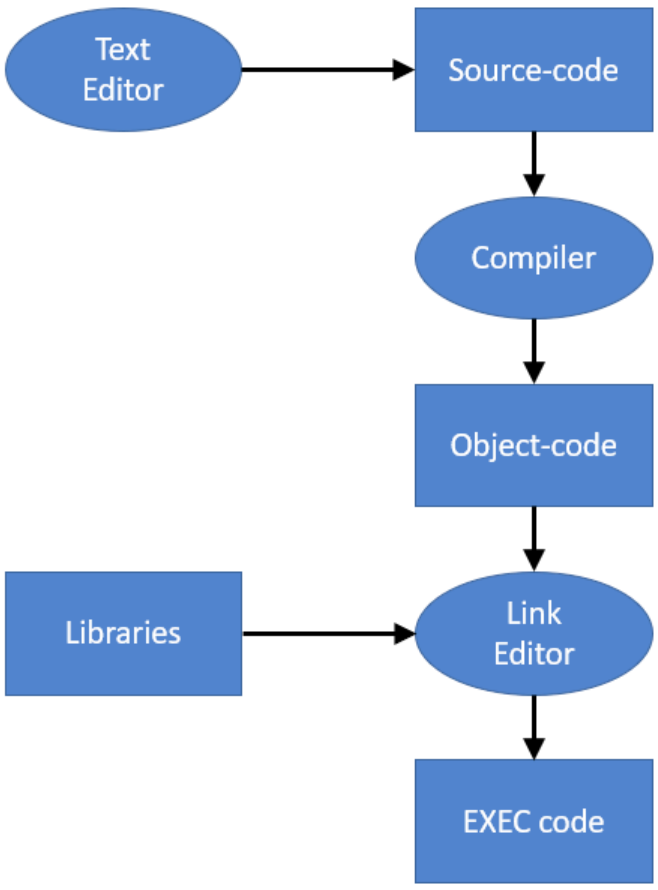

Figure 1 - Typical flow of development of a computer program, Source: The Author

To justify this approach, let us consider a typical flow of development of a computer program, shown in Figure 1.

In analyzing Figure 1 from the originality aspect, we must first have the source-code to be the central object of our study, since the object code goes through a process of modification, losing the originality of its intellectual content in the passage to the next phase, in which the object-code is generated. It reinforces the fact that different arguments used in the use of compilers can generate different object-code. In addition, for a study of copyright infringement by plagiarism to be carried out on the object code, it is necessary to ensure, through a chain of custody, the integrity of the object code through the hash function [1], in the sense of there is a certification sequence of data, and this rite simply does not exist in the expert segment for analysis of copyright infringement of a computer program, which would guarantee the integrity of the object-code. For this reason, and others that will be discussed below, the analysis of the violation should be done exclusively on the source-code and not on the object-code.

Another strong reason for not using object-code in computer program copyright infringement analysis is because the instructions contained in it do not reflect the originality of the source-code. They represent the standard machine language instructions, which may vary your code depending on the processor used. The assembler code was generated to interact with the processor, representing its logical instructions and the necessary movements in memory, for its correct execution. And these

instructions are very common in an object-code. Therefore, utilizing them in the analysis of copyright infringement of computer program is a wrong approach.

Another point to consider: when analyzing the object-code files, the reader will not even find the information contained in the source-code files as they were handled by the compilers and encoded on a hexadecimal basis so that they could be read and understood by the processor.

And regarding legislation, we know that the object-code and the source-code are both subject to copyright protection. But if we can prove that the object-code is sufficiently contaminated so as not to reflect the intellectual originality of the source-code, should the object-code be on the same level of protection as the source code?

A more appropriate understanding of Art. 1 of Law 9.609/98[2] that cites "*natural or coded language, contained in any physical media nature*" (BRASIL, 1998), would be the ready-to-install codes and codes contained in the media containing programs, not the object-code itself, as I have already commented. After presenting the contextualization of the problem, some questions arise naturally from this context. They are:

a. When a copyright infringement analysis of a computer program is performed on the object-code, what procedure, or process, guarantees the integrity of these data files, linking them to the files that gave rise to it, that is, the source-code?

b. What is the procedure used to consider the lines created by the compilers, which are not intellectual authoring objects of the code author since they were generated by the compilers and not by the developer?

c. Is this procedure accredited by any recognized quality entity such as INMETRO, ISO, NIST or other national or international reference entity?

Many forensics works are damaged when they do not have integrity in the chain of custody [3], that is, the original object of the expert's work is compromised because it is no longer possible to guarantee its integrity.

The same perception is given when we receive a file in object-code, without it being accompanied by documentation containing information that guarantees its integrity, as well as the considerations regarding the additional program lines generated by the compilers, due to the parameters used.

The fundamental point is that we show that the object-code is not a product of the generation of intellectual property. The origin of intellectual work is in its source-code. Whatever is in the object-code is a contaminated code for the purpose of this type of analysis.

The industry of combating the plagiarism of computer programs reinforces the thesis of this work, developing products directed to the analysis of the source-code of computer programs, like CodeMatch, CodeCross and CodeDiff.

In all the cited examples, the most important thing is to observe that the industry develops tools to try to locate plagiarism in the source-code of the computer program, and not in the object-code, which is consistent with the sustained argumentation throughout this work, which states that analysis of similarities between computer program codes should in fact be based on the source-code and not on the object-code.

Recent articles and publications presented at international conventions and conferences around the globe target the work of analyzing copyright infringement exclusively on the source-code. As an example, we can cite the following works:

- A State of art on source code plagiarism detection [4], by Mayank Agrawal and Dilip Kumar Sharma, presented at the Second International Conference on Next-Generation Computing Technologies held in India (2016);
- Review of source-code plagiarism detection in academia [5], by Matija Novak, presented at the 39th International Convention on Information and Communication Technologies, Electronics and Microelectronics, held in Croatia (2016);
- Source code plagiarism detection: The Unix way [6], by Juraj Petrík, Daniela Chudá and Branislav Steinmüller, presented at the 15th International Symposium on Machine and Computer Intelligence, held in Slovakia (2017).

In fact, all the works surveyed in the IEEE repository, the world's largest professional technical organization for the advancement of technology, turn to computer program source-code analysis and even cite object-code analysis for issues related to plagiarism of computer programs.

The lack of a standard methodology for the work of analyzing similarities of codes of computer programs, where this work seeks to contribute, is one of the main factors for the inexistence of a safeguard specified in item III of article 473 of the Code of Civil Procedure, where the legislator states: "*III - the indication of the method used, clarifying it and demonstrating that it is predominantly accepted by the specialists in the area of knowledge from which it originated*" (BRAZIL, 2015).

In this respect, the technical work of the forensics analyst, or expert, must guarantee the best quality, to neutralize the possibilities of doubts. In addition, all the techniques suggested in this article follow the best practices of approach techniques suggested by the NIST institute. In this sense, the choice for a model of studies based on techniques suggested by the NIST institute is because this institute is accredited by INMETRO, which is one of the main institutions of technological and quality reference in Brazil, which is why NIST is often referred to in government and legal documents, and has often been cited by them, when it is desired to provide a robust reference guideline for the context.

In the absence of a standard procedure for this activity, and considering the technical implications presented, the copyright infringement analysis should be concentrated exclusively on the source-code of the computer program, which carries with it all the original characteristics of the intellectual development work of the computer program.

An exception occurs during the analysis of copyright infringement by the non-literal elements of the computer program, which is presented in the proposed methodology of analysis of this work.

The analysis of the non-literal elements becomes substantially important when, under the various perspectives of analysis by the source-code, no substantial elements are found that evidence the copyright infringement and another suggested analysis approach is necessary, the comparison of the operational logical structure proposed in item 4.2.10 of this work, since the plagiarists also became sophisticated offenders.

2. Conceptual Elements

2.1 Introduction

The conceptual elements are the fundamental basis for understanding the content presented in this work. In this chapter, we will introduce some concepts related to the theme.

2.2 GUI – Look and Feel

In the excellent work of Professor and Doctor Manoel Joaquim Pereira dos Santos, titled "A Proteção Autoral de Programas de Computador" (The Autoral Protection of Computer Programs), a literary reference in the treatment of Intellectual Property, chapter 10 deals with User Interface protection and the theory of "Look and Feel".

Dr. Manoel points out that:

The user interface basically comprises the elements that are associated with the functionality, allowing the user to interact with the program, which is why it relates to the concept of interactivity. [7] (SANTOS, 2008, p. 283).

He comments that these elements are related to the video screens of programs, menus, windows and other graphic features of the computer program. And these characteristics associated with the visual aspect of the computer program, that is, the user interface, denotes the expression "Look and Feel".

In this sense, the characteristics of "Look and Feel" are related to the design of the graphical user interface, which cover the ways in which windows are presented, the colors used, the designs, and the letter formats used in these elements, which constitute the "Look". The behavior of the graphic elements, in the dynamic form of their operation, such as the emergence and disappearance of the windows, characteristics of opening the menus, and the behavior of the interaction buttons, constitute the "Feel".

2.3 Literal Elements

The source-code and the object-code constitute the literal elements of the creation of the computer program, developed by the author and, therefore, are objects subject to copyright protection.

To clarify, Law No. 9.279, which regulates the rights and obligations related to industrial property, establishes that: "Art. 10. It is not considered an invention or a utility model: [...] V - computer programs," [8] (BRAZIL, 1996).

As a guideline, copyright protects literal elements, such as source-code and object-code, for 50 years, extending to some exceptions for non-literal elements such as the user interface and architecture. By architecture, the structure, sequence, and organization of the program code are understood. In summary, the source-code, the object-code, the architecture, and the user interface are copyright objects.

2.4 Non-Literal Elements

Commonly, we find documentation of the logic of the development of the computer program by its flowcharts and in the specifications that arise even before the development of the source-code. These flowchart documents should be understood as non-literal elements, since they are considered as a technical solution developed for solving a particular problem, which is used as support for the development of a computer program. In general, the protection regime for non-literal elements is determined by patent filing, but there are exceptions related to the architecture of computer programs and the user interface, whose peculiarities are verified, on a case-by-case basis, in plagiarism analysis, which may constitute copyright infringement.

Although the INPI does not grant patents for computer programs, patents are granted for inventions that are implemented by a computer program. Therefore, the patent regime protects the non-literal elements for 20 years, from the date of filing the patent. The non-literal elements are the technical characteristics of the invention, whose right emerges from the filing of the patent and subsequent grant by the INPI.

In summary, the patent application is the technical solution to a technical problem, which may be expressed in the algorithms and other non-literal elements of the computer program. However, other non-literal features are objects of copyright protection, and the definition of these non-literal elements constitutes a major challenge in the subject of intellectual property.

We find in the excellent work of Professor and Doctor Manoel Joaquim Pereira dos Santos, the protection of non-literal elements, well delimited, in Chapter 7 of his work. Dr. Manoel goes far beyond the definition of non-literal elements, covering aspects of copyright in the perception of American, European and Australian law.

Generally, non-literal elements would be defined as anything that is not understood in the code of the computer program. However, Dr. Manoel emphasizes the evolution of the protection of non-literal elements, originating in US law, in which a work can be considered a counterfeit of a pre-existing one if the substantial similarity with respect to the protected elements of the same is evidenced.

This balance and care in the definition of non-literal elements has enormous relevance in the scope of copyright, which is why Dr. Manoel Joaquim Pereira dos Santos dedicated an entire chapter of his work to deal with the subject. And the key to such cases is the interpretation of the idea/expression dichotomy, and how this relates to computer programs. If there is no creative intellectual activity in the form of expression of a certain idea, there is no copyright application.

2.4.1 Architecture

The other aspect regarding non-literal elements is related to the program architecture. Generally, the literature related to the theme defines the architecture as the structure, sequence and organization of the computer program. This definition, however correct, is however very vague. The forensic analyst needs to know what the architecture of a computer program is all about, since that understanding is of fundamental importance for the analysis of the non-literal elements that require profound abstraction.

Conceptually, a computer program consists of a collection of components organized to solve a given problem. Its architecture consists of the organization of the program, incorporated in its components, its relations with each other and with the external environment, and the principles that guide its way of functioning.

The description of the architecture of a computer program is a collection of the artifacts that compose it, being several elements considered in its architecture that, for the copyright infringement analysis, must be well understood.

The vision of this architecture constitutes all the documentation that guides the purpose for which the program was developed. To provide a better understanding of the architecture, we need to subdivide it into some sub-items, such as: business architecture, data architecture, application architecture, and technology architecture. What is defined by these architectural elements and what is expected as a result of their implementation in the program, i.e., what are the expected inputs and outputs, and how do users interact with them?

Written in a simpler way, what would be the definitions of the principles that constitute the elements of architecture?

The principles guiding the way programs work are the general rules and guidelines, data management, which inform and support the way the program commits itself to fulfill its mission.

In more mature program development companies, the principles of architecture are developed by the team of system architects in conjunction with program developers and, in most cases, with key stakeholders involved in the operating environment. In some cases, client companies are invited to participate in a program improvement process. It is complex work that, if well managed, leads to a consistent and clear orientation, or strong influence, for decision making.

So, to provide a consistent and measurable level of information related to the architecture of a program, we need to understand the principles of each architectural element that guide the way the computer program works.

2.4.1.1 Business Architecture

The architecture of the business is based on maximizing the benefit to the company. It is important to make it clear that the company, in this case, is the one that will make use of the program, and not the company developer of the program, because the computer program was developed for the client who are supposed to benefit from the program.

Decision-making from the business perspective has greater long-term value than decisions taken from any other perspective. The maximum return on investment requires information management decisions to adhere to the company's business priorities. This process leads directly to the choice of at least one of many different alternatives, all candidates to solve a certain problem.

For example, a computer program oriented to cargo logistics management will consider factors such as freight cost, insurance, quality requirements, among other things, for hiring a freight carrier. On the other hand, once the freight company is hired, the program will manage the decision making of the routes based on the shortest route, the best period for the trip, the history of the shortest time traveled, driver profile, etc.

Through this type of program decision-making, the transportation schedule is optimized, reducing fuel costs, contributing to the reduction of the emission of harmful gases to the environment, and meeting the quality requirements for the consumer. Note that if a company is committed to the environment, with strong adaptation policies, this item not only contributes to sound decision making, but also makes the company even more compliant with environmental policies and standards, that is, maximizing the benefits.

The principles of business architecture map to the decision making and quality requirements of the program, which are expected to be well documented by system architects in business requirements, or in quality requirements.

2.4.1.2 Data Architecture

The data architecture has as its main basis the valuation of the data, since the objective of data management with quality is to assist in the decision making of the companies. Mature companies treat data as a valuable asset because accurate data is critical to accurate decisions.

The principles involved in data architecture consider that data, while constituting an asset of the organization, must be shared and, likewise, be easily accessible. Although it seems simple, there is a certain complexity involved in this item, since the relationship between data value, sharing and accessibility to data has varied perspectives on the information security aspect within companies, and varies greatly from company to company, because they have different information security policies to deal with this issue.

Based on this scenario, companies often develop procedures to prevent, and also correct, the eventual production of wrong data in their databases, in order to improve processes that may contribute to the production of misinformation, which will certainly contribute to a negative influence on decision-

making. Data quality needs to be measured and measures need to be created to consistently improve data quality. Consequently, it is likely that policies and procedures also need to be developed to address the sensitive issue of data quality.

2.4.1.3 Application Architecture

The application architecture is based on the premise that the programs must be independent of specific technological choices and, therefore, must operate in the various technological platforms available in the market. The independence of computer programs allows them to be developed, updated and operated in the most economical way possible. This principle requires standards that support portability for other operating platforms.

It is easy to abstract a scenario in which there is a need to create programs to operate in different technological platforms. Consider a company like SAP that needed to develop an application for its target audience: large corporations. The first realization is that SAP has no control over which database servers your audience uses. So, the company needed to develop a program that would run on a variety of database operating platforms. However, based on this same principle, the company evolved when it realized that the companies that made use of its ERP also needed a database. In other words, SAP understood that it was a potential sales motivator for other database players such as Oracle, Microsoft and IBM, for example. As a result of an excellent work of continuity in the development of application architecture, the company worked in its suite that was named Hana. From then on, SAP had an ERP integrated with its own database, and gains in integration with other databases translated into savings and efficiency for its customers who, in licensing their ERP, no longer needed to worry about a database. But if you already had a database, the integration process would be easier.

In another less complex example, this principle led many developers to make use of Java technology in their applications, and later Java-like packages such as Open JDK provided a high degree of platform independence support for these programs, since they could operate on different platforms like Android, Windows and IOS, for example.

The ease of use of the program is another principle observed in the application architecture as it should be transparent to users so that they can focus on the core business activities, also known as the "core

business". In other words, the lack of understanding of this principle implies a less productive environment for the company.

2.4.1.4 Technology Architecture

The technology architecture has as main foundation to support the needs of changes of the programs of computer throughout its life cycle, in response to the needs of the business that experience constant requirements of changes. The purpose of this principle is to provide an operating environment to keep the company focused on business and not on technology. The technological architecture developed should provide this environment.

A commercial need of the company may require a technical improvement or development of subsystems of the computer program that is in operation. For example, the advent of Accounting SPED brought a need to adapt computer programs to all companies in Brazil. Companies that had a technological environment supported by this principle did not have great difficulties in promoting the necessary changes to the adequacy of their programs, keeping the focus on their business. On the other hand, there were numerous cases of companies that were unable to adapt their programs to SPED Accounting. As a result, they needed to make great efforts to comply with the requirements of the Treasury Department, shifting the focus of the business to comply with a legal requirement.

Going back to the example of SAP, for a company that had a license for its ERP, it only had to develop an interface for SPED Accounting, with the specifications developed by SEFAZ.

2.5 Operational Platform

Nowadays, the term platform is used in a generic way to specify elements in the most varied segments of the information technology industry. For example, to distinguish between programming languages, expressions such as Java platform, C platform, etc. are used.

Similarly, to understand the support specification for the various operating systems on which a computer program can run, it is common to use terms such as Windows platform, Linux platform, Mac platform, etc., which refers to the operating system in which the program can be run. The term is also used to differentiate processor specifications by using jargon such as Intel platform, Motorola platform, Arm platform, etc.

In this work, the term platform refers to the operating system in which a computer program can be run. This is because in the context of the studies of this work there is the plagiarism approach of computer program code that can be used in different operating systems, which is why the term operational platform is used.

3. Types of Similarities

3.1 INTRODUCTION

An article published by DROPS [9] under the title "Similarity in Programs" offers a conceptual approach on the similarity of programs, exploring the syntactic and semantic aspects.

The article refers to the syntactic aspect as being the form of representing the source-code, usually a sequence of characters forming a more complex text structure. In this approach, the researchers observed that similarity in representation allows the similarity between codes at different levels of abstraction, which correspond to different modes of observing programs, such as the similarities between statements in programs, blocks, classes and architectures.

And in the semantic aspect, the article maintains that the programs can be considered similar when the main similarity is in their behavior, that can be defined by the functions implemented by the program. However, the consensus of the researchers was that there are few ways to make the semantic comparison, but in a broader sense it is possible to point out two types of representation that can be used in this type of comparison: operational and logic.

Operational similarity details the steps of running a program, while logical similarity establishes the meaning of each line of programming, its variables, and its logic. This is another strong reason for the comparison to be made over the source-code and not the object-code, since the meaning of each line assumes the understanding of the programming, the declarations made in the source-code and the logic associated with the code, since we have previously seen that the object-code is a standard output format of the language to be handled and executed by the processor.

The doctrine has also contributed to reinforce the need for source-code analysis. In the excellent work titled Evolution Software, the Belgian authors Tom Mens and Serge Demeyer highlight the consensus in the international community on the importance of detecting clones, which are similarities between codes. This is a common misconception observed in many literatures, which refer to the clone as plagiarism. First of all, it is important to make it clear that cloning is not plagiarism. A much more appropriate definition is that of Dr. Manoel Joaquim Pereira dos Santos, who deals with the differences between clones, plagiarism and servile copy in Chapter 5 of his extraordinary work "A Proteção Autoral dos Programas de Computador" (The Autoral Protection of Computer Programs), conceptualizing and characterizing the clones in this way:

A clone can be defined as a program that is functionally equivalent but distinct from another computer program. In technical terminology, however, the clone not only performs substantially the same functions, but imitates the appearance and operation of the previous creation. [10] (SANTOS, 2008, p. 348).

And a classic example of Dr. Manoel's definition is the Microsoft Word text editor, Office Package, and one of its clones, the Writer text editor, of the LibreOffice package. Writer is a clone of Word, and it is not Word plagiarism.

One needs to be very careful in properly interpreting the word ‘clone’ in literature, as the term often refers to the plagiarism of computer program code.

It is important to point out that in all the conditions covered in the work "Software Evolution" of 347 pages, it is worth mentioning the fact that all studies and techniques mentioned are applied to the source-code, at no time on the object-code, as we maintain in this work.

3.2 Hypotheses of Similarities

Law 9.609/98 foresees the hypotheses of similarities of computer program in Art. 6°, where the legislator defines:

> *Art. 6° Do not constitute an offense against the rights of the holder of a computer program:*

[...]

III - the occurrence of similarity of program to another, pre-existent, when due to the functional characteristics of its application, observance of normative and technical precepts, or limitation alternatively for its expression;

IV - the integration of a program, maintaining its essential characteristics, to an application or operational system, technically indispensable to the needs of the user, provided that for the exclusive use of who promoted it. [11] (BRAZIL, 1998).

The assumptions of permissible similarities are those program source lines that we can identify as being mandatory, without which programs cannot be compiled or, if they are, will produce errors. Example: the statement **`#include <stdio.h>`** in a program written in C language.

Because of this requirement, similarity to functions like this is permissible if the statement **`#include <stdio.h>`** is present in the programs.

Another hypothesis is related to the indentation of the source-code lines. The indentation technique is used to organize program logic, for example, when opening and closing keys in the C programming language. Note the program line below, extracted from a C program.

```
int main() {
```

Using the indentation technique, the code could be written as follows:

```
int main()
{
```

The isolated key in the immediately following line is a way of isolating the code, arranging it in an indented form, for better separation, a technique that provides a better understanding of the code by the programmer.

When encountering indenting techniques like this, it is likely to find many identical lines of distinct codes. This condition is also permissible when finding similarities found in source-code.

Another important aspect regarding the likelihood of similarities is related to the lines and empty spaces. Although the incidence of lines and voids are in most cases admissible, the similarity of this similarity across two codes under analysis can map to a scenario of plagiarism.

There are professionals who produce a kind of digital signature in their codes. For example, there are source-codes where the author always inserts two spaces, instead of a single space, between the arguments of the code. By analyzing two codes with this characteristic, an experienced forensic analyst will surely notice this subtle resemblance. This aspect will be of great importance to the reader in the methodology of analysis of similarities between computer program source-codes, proposed in this work and presented below.

4. Proposed Analysis Methodology

4.1 Considerations

During my lecture at the Seminar on Preventive and Repressive Cyber Crimes, accepting the invitation of the FIESP Security Department, an event promoted by the Public Prosecutor's Office, FIESP and FEBRABAN, on June 21, 2017, one of the issues that arose from the public was the maturity of digital forensic analysis and investigation in Brazil.

My answer was to explain that Brazil faces, in fact, great difficulty in cases of forensics investigation, justified mainly by the lack of a standard methodology for the forensics analysis work in the country.

In this same line, the lack of a methodology for the work of analyzing similarities of computer program code contributes to a scenario where different perspectives are observed, from different angles, of different professionals with varied experiences on the subject, different methods of approach, resulting in evaluation work with different contents and procedures and, consequently, different results. The suggestion of the methodology proposed below is intended to fill this gap and contribute to the judiciary, which has the difficult task of judging each case.

The techniques presented in this chapter are based on studies of the approximation techniques suggested by the NIST [12] institute, since this institute has worldwide recognition and is also

accredited by INMETRO, which is one of Brazil's leading institutions of technological and quality reference, which is why NIST is often referenced in documents of government agencies [13] and legal [14], and has often been cited by them when it is desired to provide a robust reference guideline for the context.

They are also the results of the technical analysis studies known as "abstraction-filtration-comparison test", which refers to a judicially created test in the United States, and used to determine if there are substantial similarities between the non-literal elements of two or more programs.

4.2 Analysis Methodology

The process of the proposed analysis methodology has a sequence of phases, starting from a simpler analysis, advancing to more complex and difficult-to-perceive phases, requiring greater intellectual effort from forensic analysts as it advances in the next phases of this analysis methodology.

Then, as a natural result of the approach contained in this paper, the first step in the proposed methodology is to analyze the literal elements of the computer program source-code in a first phase of analysis and, in a second phase, non-literal elements.

The analysis of the non-literal elements is fundamentally important, especially when we consider the existence of plagiarism in different operating platforms, in which the similarity can be observed in the behavioral aspect of the computer program, and not by the source-code, which may have been the subject of a process of recoding computer program code for use in another operating platform.

In addition, we must also consider the scenario in which another programming language is used, with the aim of further blurring the similarity. The following are the ten comparison phases proposed in this methodology.

4.2.1 Phase 1: Reading and general comparison of the codes

The first phase of the proposed similarity analysis methodology constitutes an initial comparison through the complete reading of the two source code files to be analyzed.

There are two motivations in putting the two source-code files side by side. The first is that strong initial evidence already appears in this first phase, such as repetitive comments, watermarks, proprietary

strings, among other strong signs that evidence plagiarism in this initial phase of analysis, such as the double spaces we quoted earlier, which function as a kind of watermark of the author.

Sometimes, when analyzing a source-code program written in C ++ or Python, of my own, a forensics analyst would question what would be the WA1, WA2, WA3, and so on strings. This is a coding of my control to identify how the blocks were ordered in my programs. A block within another block would have the string WA4.1, that is, a sub-block 1 within block 4. This practice constitutes what I call watermarks, which identifies my authorship in the logical controls of my programs. Experienced programmers often create watermarks for this purpose.

Another motivation to put them side by side is to observe the entire process of transformation that will occur as the subsequent analysis techniques are applied. Many source-code files seem distinct at first glance, however, when some analytical techniques are applied, they begin to highlight the unseen similarities at first. After a complete reading of both codes, phase 2 is performed, which consists of extracting the metadata.

4.2.2 Phase 2: Extraction of Metadata

The extraction of the metadata has the objective of verifying the existence of fragments of information of the origin from the author of the computer program's source-code.

In this second phase, the extraction of the metadata is applied not only to the source-code of the computer program, but also to the object-code, documentation files, logical files and other existing files of the implemented solution.

The information contained in the metadata may map to a source other than the source claiming the intellectual property of the computer program, and for that reason, the metadata must be extracted from any file linked to the solution implemented by derivation of a computer program’s source-code.

As an example, the extraction of the metadata from the archive of this work, by the tool exiftool [15], brings the following results:

```
┌─[wash@parrot]─[~/Documents/Poli-USP/Monografia]

└──╼ $exiftool Monografia-WashingtonAlmeida.doc

ExifTool Version Number: 10.56
```

```
File Name                       : Monografia-WashingtonAlmeida.doc
Directory                       : .
File Size                       : 904 kB
File Modification Date/Time     : 2017:06:30 20:58:53-03:00
File Access Date/Time           : 2017:06:30 16:20:10-03:00
File Inode Change Date/Time     : 2017:06:30 20:58:53-03:00
File Permissions                : rw-r--r--
File Type                       : DOC
File Type Extension             : doc
MIME Type                       : application/msword
Comp Obj User Type Len          : 24
Comp Obj User Type              : Microsoft Word-Dokument
Title                           : Monografia - Washington Almeida
Subject                         : SEMELHANÇAS DE CODIFICAÇÃO ENTRE PROGRAMAS DE COMPUTADOR
E VIOLAÇÃO DE DIREITO AUTORAL
Author                          : Washington Almeida
Keywords                        : programa de computador, código-fonte, código-objeto, direito
autoral, similaridade, infração
Comments                        : Trabalho de Conclusão de Curso da Especialização em Direito e
Tecnologia da Informação da Escola Politéncica da USP
Template                        : Normal.dotm
Revision Number                 : 556
Total Edit Time                 : 3.5 days
Last Printed                    : 2011:04:06 04:15:00
Create Date                     : 2012:02:09 15:59:00
Modify Date                     : 2017:06:30 23:58:52
Code Page                       : Unicode (UTF-8)
```

The metadata information collected brings some important information from each file, valuable for the interpretation of its authorship. This information is not in the body text of the document. You can check, for example, the number of revisions that the document has undergone since its inception, in the example, 556 revisions. There is the date of the last print of this file, which occurred in the year 2011, however it has a creation date in the year 2012. How is this possible?

When compiling this information, considering also the possibility that this file may have been transported from one device to another, we may realize that, despite the author's record reporting as being from Washington Almeida, it is unlikely that he was the creator of the file. In fact, this document model was assigned to the author of this paper, Washington Almeida, by the Polytechnic School of USP.

Another piece of valuable information that metadata brings is the versions of computer programs. It would be very coincidental that two companies claiming the authorship of a given program had the same criteria of documenting the format of the version as, for example, v.1.00a.

Typically, these criteria for documenting the version are linked to an internal company standard setting the standards for document naming. In certain situations, the absence of these internal rules in one of the companies, associated to the way the programs were made, are strong indications of violation.

4.2.3 Phase 3: Removal of Comments

A relatively common action pattern observed in Computer Program Copyright infringement occurs when the violator removes the comment lines to overshadow the original source-code, which significantly alters part of the source-code of the computer program at first glance.

This phase aims to leave both source-code files pollution-free from comments. Removing comments will help to highlight the most apparent similarities, even if the source-code has been changed. After removing all comments, follow the phase of the syntax comparison in the next item.

4.2.4 Phase 4: Reordering the Source-Code

The fourth step of the proposed similarity analysis methodology is to reorder the source-code as soon as comments are removed.

Consider the following code sequences below:

```
Code-A                          Code-B

int main ()                         int

{                               main()

printf ("Teste")                    printf (

return 0;                       "Teste");

}                               return

                                0;

                                }
```

Note that by rearranging one of the above codes, we realize that they are exactly the same. In programming languages that use the "{" and "}" delimiter notation to separate declarations and code blocks, such as C language, for example, the restrictions on the use of whitespace are minimal. Likewise, the indentation of the source-code can be purposely altered to connote the expression of a different source-code. In the example of the above codes, syntactic changes were made to restructure the source-code for this structurally different opinion. At first, they would be even more different from the existence of comments, in order to obfuscate the code, which is the reason why comments are removed in the previous phase.

The reordering of the code aims to identify the structural similarity of both sets of source-code files. Structural similarity is enhanced as the elements of the code are rearranged. After reordering the source-code files, the syntax comparison follows.

4.2.5 Phase 5: Syntactic Comparison

The fifth step of the proposed similarity analysis methodology is to make the syntactic comparison between the source-code files considered similar. The syntactic similarity consists of analyzing the two sets of source-codes, placing them side by side and partitioning them into several blocks, with only the lines of code of the same block being compared.

In this phase, the statements that occur in each block are compared, such as variable declarations, functions, classes, and constants. In cases of extremely long program codes, the expert can use the

technique of using the hash function for repetitive strings, in order to simplify the process for the forensics analyst, but the use of this technique must be very well documented, especially taking into account that the reading of the material produced will be carried out by non-technical personnel in the area of law, such as lawyers and judges.

In this phase, the analysis of similarities requires greater intellectual effort from the forensics analyst, in addition to his excellent abstraction ability, since it is likely that blocks containing the variable, function, class, and constant declarations have been completely modified with the aim to make it appear as a unique creation, when in reality, one sees the disguised, obscured, masked exploitation of the elements of the pre-existing creation.

Once the syntactic similarities are highlighted, we move on to the next phase, which consists of analyzing the dependencies.

4.2.6 Phase 6: Analysis of Dependencies

The sixth step is to analyze the dependencies of the variables defined in the codes. The simplest way to analyze these dependencies is the use of simple graphs, which aim to show similarities in data dependencies, as well as controls present in the source-codes of computer programs.

Although the violator tries to use code obfuscation techniques, using different variable names, the graph analysis will identify the logic used by the different variables found in the analyzed codes.

Consider the following codes below, which both perform the factorial calculation of an integer:

Code 1:

```
int i, j=1;
for (i=1; i<=VALUE; i++
j=j * i
```

Code 2:

```
int factorial (int n) {
if (n == 0)
return 1;
else
return n * factorial (n - 1)
}
```

At a first look at the two codes, we note that they appear to be very different. Let us then analyze them, considering their dependencies, visualizing them in a graph of graphs.

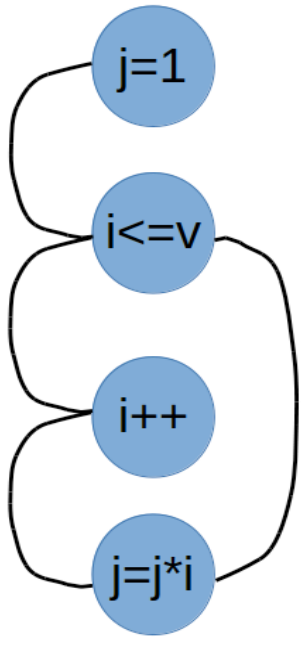

Figure 2 - Code-1 Graph, Source: The Author

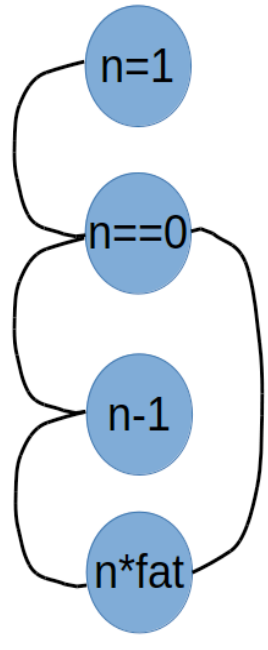

Figure 3 - Code-2 Graph, Source: The Author

By observing the graphs of both codes, we realize that they are exactly the same, they only perform the factorial calculation in the inverse order of each other, but revealing the similarity of their dependencies for the variables "i" and "n".

Control dependency is identified in code 1 by the expression "i <= VALUE", while the control dependency identified in code 2 is represented by the expression "n == 0". Likewise, data dependence is identified in the code 1 by the expression "j = j * i", while the data dependence identified in code 2 is represented by the expression "n * factorial".

Note that at this stage, the analysis already requires a more accurate level of perception and abstraction from the forensics analyst, since the evidence may not be so noticeable in an initial analysis. Identifying the similarities by the dependency analysis, we move to the next phase, which consists of the data flow analysis.

4.2.7 Phase 7: Data Flow Analysis

The seventh step is to analyze the data flows of the source code, also considering the flows of procedures or functions that are invoked within these fragments.

The use of a data flow diagram graphically highlights the representation that describes the problem to be solved. This technique aims to highlight similarities from two perspectives: semantic and structural. In the semantic aspect, it aims to identify how the behavior of the program was modeled, and under the structural aspect, it aims to identify how the structure of the data processed by the program was modeled.

The components used in the analysis of the data flow are the processes and the flow. The process transforms the inputs into outputs, and the flow itself, identifies the movement of the data.

Figure 4 below is an example of how the information flow can be chained so that a better understanding of the problem to be solved can be understood. In the case of the example shown in the figure, it highlights the flow of the simplified steps for issuing an invoice.

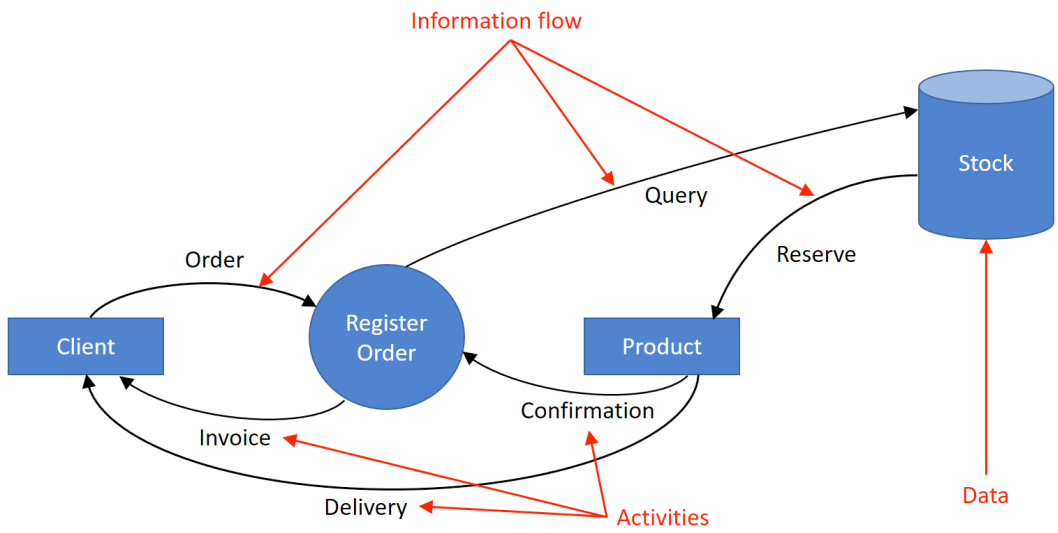

Figure 4 - Sample diagram for data flow analysis, Source: The Author

The data is transformed into each step before moving to the next stage. This procedure helps the forensics analyst understand what happens to the data during the execution of the program. Similar flows imply semantic similarities of codes.

The next phase considers the legal hypotheses of similarities.

4.2.8 Phase 8: Removal of Hypotheses of Legal Similarities

The eighth step of the proposed similarity analysis methodology is to separate the permissible similarities from the non-permissible similarities. It implies evaluating the hypotheses of similarities

included in the inc. III of Art. 6 of Law 9.609/98, where the legislator declares that they are legally admissible:

> *III - the occurrence of similarity of program to another, pre-existent, when due to the functional characteristics of its application, observance of normative and technical precepts, or limitation alternatively for its expression; [16] (BRAZIL, 1998).*

A classic example of this phase is the analysis of source-code of programs directed to the SPED, Public System of Digital Bookkeeping, that must obey the resolutions of the Federal Counsel of Accounting.

Programs developed for SPED Accounting, SPED Fiscal, SPED of PIS and COFINS, Model-3, E-Social, among others, such as programs aimed at issuing invoices in commercial establishments, which obey SEFAZ technical notes, also fit into the hypotheses of legal similarities and should be removed at this stage of the analysis process.

4.2.9 Phase 9: Analysis of Semantic Similarity

The semantic similarity analysis is fundamentally important for cases where a program has been re-coded in another programming language for, for example, its use in another operating platform. This scenario is most commonly observed when reverse engineering techniques are used by the violator to obtain the binaries of the code he wants to plagiarize.

The semantic similarity refers to the behavior of the observed programs. In this phase, a part of the source-code of a program A is semantically similar to another part of the source-code of a program B if, and only if, program B contains the functionality of program A.

The structure and syntax of the source-code are not the targets of analysis at this stage. What defines the similarity is the semantics involved in computational processing, that is, the functional result of the operations. For example, the loop defined by "while (a > b)" can be changed to "while (b < a)", while "while (x > y)" is much more sensitive in the perception, because the logical inversion of the while command occurs along with the change of its variables.

Since block separation was performed in phase 5, these details appear more prominently for the forensics analyst. Another similarity profile, which demands greater expertise in this phase, is the replacement of functions by similar code fragments and vice versa.

Let us consider the two fragments of source-code used in section 4.2.6, represented again below, which are written in C ++ programming language.

Code 1:

```
int i, j=1;

for (i=1; i<=VALUE; i++)

j=j * i
```

The code fragment above performs the factorial calculation of a number. As seen in 4.2.6, this code can be rewritten by the following code below.

Code 2:

```
int factorial (int n) {

if (n == 0)

return 1;

else

return n * factorial (n – 1)

}
```

In the semantic aspect, the codes presented above are exactly the same, because the observed similarity is defined in the semantics involved in the computational processing of both, that is, the codes do exactly the same things.

In this phase of the analysis it is fundamental that the forensics analyst has vast knowledge and understanding of the aspects related to the programming language used, and also excellent ability to abstract the semantic characteristics when analyzing the fragments of the codes.

4.2.10 Phase 10: Comparison of the Operational Logic Framework

The comparison of the logical operational structure is perhaps the most difficult phase of analysis for the forensics analyst, within the process of the methodology proposed in this work.

The complexity is summarized in the fact that there is no substantial evidence to characterize computer program plagiarism in the examinations carried out in the previous phases, and also because the forensics analyst must verify the possible evidences through an abstraction process.

In this process of abstraction, the forensics analyst must exercise an intellectual activity, leading to his reflection the element of analysis, and isolated it from the common factors that are related to it, as, for example, the phases of previous analysis.

In particular, the work of comparing this phase consists in determining whether the non-literal elements of a computer program have been copied, which characterizes the plagiarism. However, unlike the previous phases in which the analysis is based mainly on aspects related to the source-code of the computer program, the analysis work of this phase takes into account a possible copy of the non-literal elements to another operational platform.

This last phase of comparison considered the studies of the NIST institute, particularly its approach model defined in the document NIST Special Publication 800-168 [17], and also the study of non-literal elements from two perspectives: architecture and the abstraction-filtration-comparison test.

The first perspective is related to the architecture which, as seen in detail in section 2, constitutes the structure, sequence and organization of the program, and aims to establish a division between the idea/expression dichotomy in order to make sure that the form pre-existing program has not been copied. Put in another way, copyright protection is given only to the way ideas are expressed, not to ideas themselves. With the objective of offering a better understanding, we divided the architecture in section 2 into four families, being the business architecture, data architecture, application architecture and technology architecture.

As for business architecture, the forensics analyst should focus his efforts on the characteristics of the developer companies, their criteria for business definitions, and understand how the allegedly offending program addresses issues related to business architecture.

In relation to the data architecture, the forensic analyst's analysis criteria must be based on the requirements for specifying the data modeling, and often, these elements are not found in the system’s

documentation, except for very mature developer companies in their management systems, which have procedures for documentation of computer programs, even in this phase of detailing the architecture.

Regarding application architecture, the forensic analyst should raise the level of abstraction in this item, evaluating the supposed counterfactor's program, so that it proves its commitment to the two principles of application architecture: technological portability and ease of use.

As for technology architecture, the forensics analyst must assess whether the criteria for analysis were based on the maturity of the allegedly counterfactual program in relation to its ability to respond to the varied needs of the business, which requires the ability to rapidly adapt the computer program.

The second perspective is the study of analysis under the aspect of the abstraction-filtration-comparison test. This test originated in 1992 during the investigation of a case involving Computer Associates International Inc. and Altai Inc., which consists of orderly and organized separation of a computer program according to its different levels of generality, or different levels of abstraction. Since then, this analysis has been widely adopted by the courts of the United States and also recognized by courts around the globe.

Doctor Manoel Joaquim Pereira dos Santos refers to this test in his book [18], as "Abstraction-filtration-comparison test", which consists of a sophisticated process containing three steps to determine the substantial similarity of the non-literal elements of a computer program. Briefly, each separate part is examined to filter out unprotected program elements. Then the elements that are unprotected are removed, and the remaining protected elements are compared with the allegedly infringing program, to determine if the substantial elements of the original program were plagiarized from their non-literal elements.

The application of the abstraction-filtration-comparison test is complex because the comparison of the operational logical structure is not described in any form of programming language or computer program development methods that would be familiar to programmers and computer scientists, who would be the most appropriate technical experts to assist the courts in such cases.

Qualified forensics experts should, therefore, play an important role in this process of abstraction, precisely because this level of analysis does not understand the conventional technical tasks of analyzing a given programming language or program development processes.

Another factor that increases the complexity of this phase is due to the size of the source-code of most commercial programs, since the effort involved in abstraction and filtering of almost any program is very large and would produce a huge number of variations. Thus, the best method of analysis is one in which the forensics expert limits the portion of the program that the author claims to have been copied.

However, due to the technical deficiency of the forensics analysts for the evaluation of this specific phase, it implies difficulties to reach objective results, since there are many details involved in this process of analysis. Abstraction can result in the comparison of hundreds of different aspects of a computer program if the forensics analyst is not careful.

So, the most appropriate application of the comparison, at this stage, should consider the refinement of the task, in order to reduce the technical complexity.

It is on this reduction that it treats the directives of the NIST institute and, based on the mentioned studies, the understanding is that the elements involved in the comparison of the logical operational structure should be filtered based on the following criteria: efficiency, external factors, convention of the language used, the process and the public domain.

a) Efficiency;

The functionality of a particular form of expression relates to the program's efficiency. Its existence within the program was due to efficiency requirements. The way to perform a given function is expressed for reasons of efficiency.

b) External factors;

The form of expression of the element was created by external requirement, being necessary for the specification of a format of data to interoperate with another program. For example, an ERP program may have the external requirement of having to export a file from one of its modules to interface with another external program, such as SPED Accounting, whose specification of its blocks of data is determined by the Treasury Department, or is an external factor.

c) Convention of the language used;

It is the scenario found where the expression of the element was created due to the conventional way of writing something in the specific programming language of that solution. For example, libraries

present in the languages to express forms of their dynamic structure on the machine in which the program is executed.

d) Process;

The element, at this particular level of abstraction, is a process, not a form of creative expression, that is the object of protection. Process identification aims to eliminate a substantial portion of abstractions that are objects of functional processes within an organization, often reflected in computer programs.

e) Public domain.

The identification of the element, in this case, is not an object of creation, but an incorporation of public domain. For example, architectural elements that perform actions on the QoS management mechanism, which are able to make dynamic decisions about the resource utilization rate and system load, are public domain implementations whose specification is defined in the RFC 2212 [19].

Then, the non-literal elements found by the filtering of these five elements are removed, and the remaining non-literal elements will be the object of the logical operational structure analysis.

It is important to emphasize that any protection of elements created by efficiency, or by external factors, or processes, must be protected by patents, if this is the case, and not subject to copyright protection.

But in this scenario, we consider that a program is based on another program that copied non-literal resources from the pre-existing program, and the comparison of the logical operational structure should be used to determine whether much of the non-literal elements were copied, resulting in copyright infringement.

The correct application of this comparison should take into account that the abstractions should be very specific and precise, otherwise, it is very easy for the forensics expert to get lost in the process of analyzing this phase, even being an experienced forensics analyst.

In the same way, abstractions must be accompanied by consistent graphic conventions for all expositions that the forensics analyst abstracts, since as abstractions the idea of this abstraction must

be very clear to any professional who comes to evaluate the evidence discovered in this last phase of analysis.

Once the idea that gave rise to the form of expression has been identified, the protection of copyright is evaluated on this form of expression, that is, it is evaluated how the structure of the idea is expressed within the operational logic of the evaluated computer program, which may show the violation of Copyright Law.

5. Jurisprudences

5.1 Considerations

The courts of Brazil and the world have established understanding of the content presented in this work, on the aspects of protection in the context of the source-code, the architecture and the forms of expression of computer programs.

The cases that follow have direct relations with the studies and the content presented in this work.

5.2 Case Mandic vs Intervale – Brazil [20]

Intervale has created a web-site almost identical to Mandic, providing the user with the same system offered by Mandic. Intervale did not prove to have a program, which evidenced the use of third party property.

The Court of Justice of the State of São Paulo judged the Civil Appeal No. 122.616-4/6 on 06/18/2012, and the decision of the Appeal Judge Ênio Santarelli Zuliani took into consideration the elements that make up the architecture, such as the structure, sequence and organization, highlighting that: "It is true that was found a copy of the items " structure "," sequence "and" organization "of the source-code program [...] (SÃO PAULO, 2012).

5.3 Case Data Access vs Powerflex Services – Australia [21]

Data Access is the Dataflex package editor. Powerflex has created the Powerflex package to complement the Dataflex package. However, to make Powerflex compatible with Dataflex the company used certain macros and reserved words used by Dataflex. However, the service code for the macros

and the reserved words were different. It was also understood by the parties that the object-code used by the two programs for commands and reserved words was not necessarily the same, as we have argued throughout this work.

Data Access stated that Powerflex had violated copyright in reserved words, some macros, and a data compression table. The court understood that the data compression table was subject to copyright protection, because regardless of the computer program, the table was an original form of expression.

5.4 Case Mitel vs Iqtel – United States [22]

Iqtel began manufacturing the IQ200+ call controller and produced a set of command code instructions to activate the capabilities of its call controller. Although Iqtel and Mitel call controllers provide many of the same features, to identify the capabilities of your controller, Iqtel's selected records differed from Mitel's records. Iqtel used identical "descriptions" and "values", where the functions of the IQ200+ were the same as those used by the Mitel Smart-1 controller.

To produce a call controller compatible with the Mitel controller, Iqtel has copied Mitel's command codes in three important ways. First, Iqtel programmed the IQ200+ to accept Mitel command codes and translate them into the corresponding Iqtel command code. Iqtel then named this resource with the name "Mitel Translation Mode".

Thus, the appeal that interests us in this case required assessing whether Mitel could prove that the command codes copied by Iqtel were protected by copyright. The court found that Mitel's command codes did not have protected expressions because the expression in the command codes did not have the required originality.

6. Summary

The development of the present work allowed an analysis of how a methodology can guide the forensics specialists, and the justice, in the cases of the similarities analysis between codes of computer programs, with more assertive results. The proposed methodology allows legal professionals, such as

lawyers and judges, to map a direction when involved in this field of analysis, to obtain more consistent results in lawsuits involving copyright infringement of computer programs.

In general, legal professionals involved in this type of analysis, mostly legal operators, have little knowledge of the programming languages that are used in the development of computer programs, and a structured methodology makes a valuable contribution to assist them in the difficult task of dealing with this highly complex technical subject. In this sense, the development of new forms of analysis, a guiding method, can contribute not only to legal professionals, but also to forensics experts, who can use a standard model to analyze similarities between computer program codes.

This point is very relevant because legal professionals do not have the obligation to understand technical issues related to programming and technical topics specific to the field of information technology, but they are who bear the great responsibility of judging each case.

In court proceedings, it is evidenced that forensics specialists do not work under a single methodology in the analysis of similarities between computer program codes, which supports the justification of this research work, in the effort to search for a standardized method which makes technical work, and complex, more objective, contributing to the results achieved with quality and efficiency.

In this sense, the use of the method presented in this paper contributes to the suggestion of a standard that allows forensics specialists to carry out their work in a unified way. In addition, it reduces the time of analysis of similarities, as the forensic expert makes his work methodical and repetitive, thus contributing to celerity in judgments.

To make this contribution, the present work strikes at the conclusion of this thesis, supported by two fundamental pillars: first, that the analysis of similarities of computer programs codes must be performed on the source-code and never on the object-code. Second, there is a need for a methodology of analysis of similarities of computer program codes that is a unique orientation for professionals who deal in cases involving the analysis of similarities of codes of computer programs.

Through the application of the methodology presented in this work, using the process of classification, comparison, analysis and synthesis, the forensics expert can extract the necessary understanding that

will help him to reach the truth of the facts, in the cases of copyright infringement of computer programs.

About the author: Washington U. de Almeida Jr.

Washington U. de Almeida Jr. is a member of the International Information Systems Forensics Association (IISFA - Italy). He holds academic degrees in Electrical Engineering and Information Technology, with university extension in Management Development Programme from Fundação Dom Cabral, and Specialist in Law and Information Technology by the Polytechnic School of USP – Poli/USP, with more than 25 years of experience, familiar with digital forensic procedures that comprises digital forensics investigations phases as collection, examination, analysis and reporting. His excellent technical background has been acquired through consistent support in cases involving the social media environment, instant messaging, droppers, ransomware, copyright infringements, e-mails system, HR systems, databases, data theft, bank fraud, computer hacking, and Internet applications, among others. Cyber security professional also works with sophisticated systems invasion testing, helping companies to improve the security of their assets. In the assistance of the Justice, he is qualified by the "Tribunal de Justiça de São Paulo" and "Tribunal Regional do Trabalho da 2ª Região" to act as digital forensics expert appointed by the judge. He also acts as Consultant in Digital Security and Digital Forensic Expert and is author of technical publications for the Polish magazines Hakin9, Pentest and eForensics, specializing in cybernetic security, penetration testing and digital forensics respectively.

References:

[1] The hash function is widely used in the process of forensic duplication of hard disks, in court expertise, to ensure the integrity of the data. The concept behind the use of the hash function is to apply a mathematical function on a large amount of data in order to obtain a result of this function, which will always be a small number when compared to the amount of data that this function hash has been applied. Checking the result of the hash function on a file guarantees it the identity of the generated version, very useful for proper documentation in a chain of custody.

[2] Available at: <*http://www.planalto.gov.br/ccivil_03/leis/L9609.htm*> Access: 12 Jun 2017.

[3] The chain of custody consists of a process of documenting the history of an evidence, in order to guarantee the tracking of the evidence used in legal proceedings.

[4] Available at: <*http://ieeexplore.ieee.org/document/7877421/*> Access: 21 May 2017.

[5] Available at: <*http://ieeexplore.ieee.org/document/7522248* > Access: 21 May 2017.

[6] Available at: <*http://ieeexplore.ieee.org/document/7880355/*> Access: 21 May 2017.

[7] SANTOS, M. J. P. A Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, 2008. p. 283.

[8] Available at: <*http://www.planalto.gov.br/ccivil_03/leis/L9279.htm*> Access: 31 May 2017.

[9] Available at: <*http://drops.dagstuhl.de/opus/volltexte/2007/968/*> Access: 01 May 2017.

[10] SANTOS, M. J. P. A. Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, 2008. p. 348.

[11] Available at: <*http://www.planalto.gov.br/ccivil_03/leis/L9609.htm*> Access: 06 Jun 2017.

[12] Available at: <*https://www.nsrl.nist.gov/approximate.htm*> Access: 16 Jun 2017.

[13] Available at: <*http://dsic.planalto.gov.br/documentos/publicacoes/2_Guia_SICI.pdf*> Access: 16 Jun 2017.

[14] Available at: <*https://www.trt13.jus.br/institucional/seguranca-da_informacao/documentos/cnj/diretrizes-gestao-si-2012*> Access: 16 Jun 2017.

[15] Exiftool is a command-line interface used to read and write metadata in a variety of file types. The metadata is read from source files and printed in readable form to the console, also known as terminal or command prompt.

[16] Available at: <*http://www.planalto.gov.br/ccivil_03/leis/L9609.htm*> Access: 12 Jun 2017.

[17] Available at: <*http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-168.pdf*> Access: 13 Jul 2017.

[18] SANTOS, M. J. P. A Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, 2008. p. 365.

[19] Available at: <*https://tools.ietf.org/html/rfc2212*> Access: 13 Jul 2017.

[20] Available at: <*https://viniciustini.jusbrasil.com.br/artigos/121943976/copy-paste-de-websites-violacao-ao-direito-do-autor*> Access: 14 Jul 2017.

[21] Available at: <*http://www.austlii.edu.au/au/journals/SydLawRw/1998/12.html*> Access: 14 Jul 2017.

[22] Available at: <*http://digital-law-online.info/cases/44PQ2D1172.htm*> Access: 14 Jul 2017.

Images in text:

Figure 1 – Typical flow of development of a computer program;

Figure 2 – Code-1 Graph;

Figure 3 – Code-1 Graph;

Figure 4 – Sample diagram for data flow analysis;