# SEMELHANÇAS DE CODIFICAÇÃO ENTRE PROGRAMAS DE COMPUTADOR E VIOLAÇÃO DE DIREITO AUTORAL*

**Washington Umpierres de Almeida Jr.**

*Engenheiro Eletrônico com extensão universitária em Desenvolvimento de Gestão pela Fundação Dom Cabral, e Especialista em Direito e Tecnologia da Informação pela Escola Politécnica da USP. Membro da Associação Forense Internacional de Sistemas de Informação (IISFA - Itália), possui mais de 25 anos de experiência em Segurança da Informação e atua como Consultor em Segurança Digital e Perito Judicial para o Tribunal de Justiça de São Paulo e TRT da 2ª Região. E-mail: wualmeida@gmail.com.*



## 1. Introdução

Os programas de computador são obras intelectuais protegidas na Lei nº 9.610/1998 de Direito Autoral, sendo eles objeto da Lei nº 9.609/1998 que trata especificamente da proteção da Propriedade Intelectual de programa de computador.

O tratamento quanto à violação de direito autoral estabelecida pela Lei nº 9.609/1998 lida tanto com a questão da cópia não autorizada do programa de computador, também chamado na forma vulgar como pirataria de *software*, quanto com a questão do plágio, mas o termo código-fonte é apenas referenciado nos casos de transferência de tecnologia de programa de computador no parágrafo único do artigo 11.

O programa de computador é definido pela Lei nº 9.609/1998 como sendo a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza. A linguagem natural é a que, tecnicamente, recebe o nome de código-fonte, enquanto que a linguagem codificada contida em suporte físico de qualquer natureza é a mídia que contém os programas prontos para instalação no ambiente do usuário, e muitas vezes essa conotação de linguagem codificada é confundida com o código-objeto. O que a lei não especifica é que o código-objeto é gerado por um processo em que a linguagem natural é convertida em linguagem codificada, processo esse chamado de compilação. Importante ressaltar desde já que a linguagem natural sofre inúmeras alterações com a inclusão de informações em código de máquina, para que seja possível o código-objeto ser compreendido pelo processador. Em outras palavras, o código-fonte é a origem do desenvolvimento intelectual do programa de computador, enquanto o código-objeto é um formato de instruções específico, resultado de uma compilação, para este lidar com o processador que executará o programa. E a linguagem codificada contida em suporte físico de qualquer natureza compreende a mídia que contém o programa pronto para o uso.

A utilização do código-objeto para a finalidade de análise de violação de direito autoral se dá quando as empresas se utilizam do argumento que o código-fonte corre o risco de ser copiado, sendo ele então tratado como um segredo industrial. Pode também ocorrer quando a análise do código-objeto é suficiente para constatar a contrafação.

O problema maior existe quando é necessária a comparação entre dois programas. Isso porque o programa supostamente contrafator não apresenta identidade ou semelhanças nas linhas de código, sendo necessário analisar a arquitetura do programa e os outros elementos não-literais para determinar a existência de plágio. Ocorre que não existe uma definição clara das hipóteses em que um código de programa de computador, que tenha semelhanças com outro código de outro programa de computador, viole a Lei de Direito Autoral de programa de computador. Não há elementos na forma de um padrão para aferir tal semelhança. A lei apenas define, no artigo 6º, inciso III, as hipóteses em que as semelhanças não constituem ofensa ao direito de autor de programa de computador.

Existe limitada variedade de literatura disponível, principalmente de origem estrangeira, algumas referenciadas neste trabalho, que dá suporte e embasamento para definição de uma metodologia científica para tais análises, quando considerando os trabalhos de perícias judiciais. Como consequência, muitos laudos judiciais e pareceres técnicos, em sua maioria, relatam a conclusiva de violação de direito autoral com fundamentos baseados nas experiências, perspectivas de entendimento e conceitos variados que os Peritos Judiciais e Assistentes Técnicos possuem sobre o tema.

A motivação do presente trabalho, bem como o objetivo central, se dá na expectativa de preencher essa lacuna, com a pretensão de sugerir uma metodologia para tal análise, fazendo uma abordagem técnica sobre o tema, e vinculando-o, quando necessário, aos aspectos jurídicos a que estes estão relacionados.

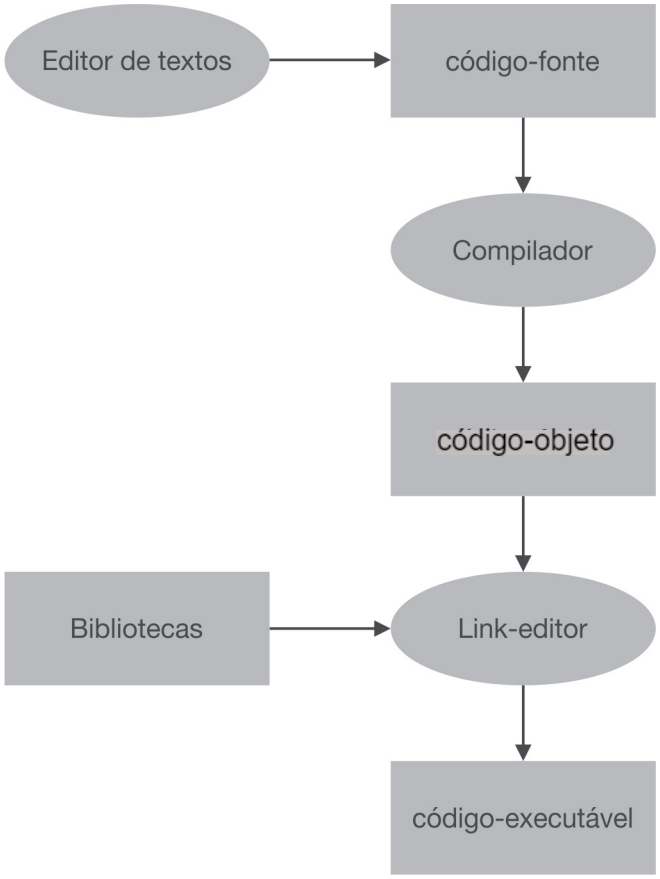
Na abordagem, pretende-se sustentar que a análise de violação de direito autoral mediante plágio deve ser realizada diretamente sobre o código-fonte do programa de computador, e não sobre o código-objeto gerado a partir deste. Logo, é fundamental compreender e definir o objeto central de nosso estudo, justificar o porquê de nossa análise a partir do código-fonte e não do código-objeto, e onde eles

* O presente trabalho resultou de pesquisa realizada no Curso de Especialização em Direito e Tecnologia da Informação da Escola Politécnica da USP, realizada sob orientação do Professor Manoel J. Pereira dos SANTOS, para a obtenção do título de Especialista.

estão localizados dentro do fluxo de interações no ciclo de desenvolvimento de um programa de computador, de forma a delimitar o escopo do estudo, exclusivamente segundo essa perspectiva.

Para justificar essa abordagem, vamos considerar um fluxo típico de desenvolvimento de um programa de computador, ilustrado na figura 1.

**Figura 1**
**Fluxo típico de desenvolvimento de um programa de computador**




*Fonte: O autor.*

Ao analisar a figura 1 sob o aspecto originalidade, logo de início temos que o código-fonte deve ser o objeto central de nosso estudo, pois o código-objeto passa por um processo de modificação, perdendo a originalidade de seu conteúdo intelectual na passagem para a fase seguinte, em que é gerado o código-objeto. Reforça o fato de que, diferentes argumentos utilizados no uso de compiladores podem gerar código-objeto diferentes. Adicionalmente, para que um estudo de análise de violação de direito autoral mediante plágio seja realizado sobre o código-objeto, é necessário garantir, por meio de uma cadeia de custódia, a integridade do código-objeto por meio da função *hash*,[1] no sentido de existir uma sequência de certificação dos dados, e esse rito simplesmente não existe no segmento pericial para análise de violação de direito autoral de programa de computador, que garantiria a integridade do código-objeto. Por essa razão, e outras que serão abordadas a seguir, a análise da violação deve ser efetuada, exclusivamente, sobre o código-fonte e não sobre o código-objeto.

Outra forte razão para não utilizar o código-objeto nas análises de violação de direito autoral de programa de computador se dá pelo fato de as instruções nele contidas, não refletirem a originalidade do código-fonte. Representam as instruções padrões da linguagem de máquina, que poderão variar seu código, dependendo do processador utilizado. O código *assembler* foi gerado para interagir com o processador, representando suas instruções lógicas e as movimentações necessárias em memória, para sua correta execução. E essas instruções são muito comuns em um código-objeto. Logo, utilizá-las na análise de violação de direito autoral de programa de computador é uma abordagem equivocada.

Outro ponto a considerar: ao analisar os arquivos de código-objeto, o leitor sequer encontrará as informações contidas nos arquivos do código-fonte, pois foram tratadas pelos compiladores e codificadas em base hexadecimal, para que pudessem ser lidas e compreendidas pelo processador.

E com relação a legislação, sabemos que o código-objeto e o código-fonte são, ambos, objeto de proteção pelo Direito de Autor. Mas se, comprovadamente, podemos constatar que o código-objeto é suficientemente contaminado de forma a não refletir a originalidade in-

1. A função *hash* é largamente utilizada no processo de duplicação forense de discos rígidos, em trabalhos de perícias judiciais, de forma a garantir a integridade dos dados. O conceito por trás do uso da função *hash* consiste na aplicação de uma função matemática sobre uma grande quantidade de dados, de forma a se obter um resultado dessa função, que será sempre um pequeno número, quando comparado à quantidade de dados a que esta função *hash* foi aplicada. A verificação do resultado da função *hash* sobre um arquivo garante a ele a identidade da versão gerada, muito útil para uma apropriada documentação em uma cadeia de custódia.

telectual do código-fonte, deveria o código-objeto estar no mesmo nível de proteção que o código-fonte?

Um entendimento mais apropriado para o artigo 1º da Lei nº 9.609/1998[2] que cita "linguagem natural ou codificada, contida em suporte físico de qualquer natureza" (BRASIL, 1998), seriam os códigos prontos para instalação e execução, contidos nas mídias que contém os programas, e não o código-objeto propriamente, como já comentamos.

Apresentada a contextualização do problema, surgem naturalmente algumas perguntas originadas a partir deste contexto. São elas:

a. Quando uma análise de violação de direito autoral de um programa de computador é realizada sobre o código-objeto, qual o procedimento, ou processo, que garante a integridade desses arquivos de dados, vinculando-os aos arquivos que lhe deram origem, ou seja, o código-fonte?
b. Qual o procedimento utilizado para considerar as linhas criadas pelos compiladores, que não são objetos de criação intelectual do autor do código, uma vez que foram gerados pelos compiladores e não pelo desenvolvedor?
c. Este procedimento é acreditado por alguma entidade de qualidade de reconhecida como INMETRO, ISO, NIST ou outra entidade de referência nacional ou internacional?

Muitos trabalhos periciais são prejudicados quando estes não possuem integridade na cadeia de custódia,[3] ou seja, o objeto original do trabalho pericial é comprometido porque não é mais possível garantir sua integridade.

A mesma percepção temos quando recebemos um arquivo em código-objeto, sem que este seja acompanhado de uma documentação contendo informações que garantam a sua integridade, e as considerações quanto às linhas de programas adicionais gerados pelos compiladores, em virtude dos parâmetros utilizados.

O ponto fundamental é que demonstramos que o código-objeto não é produto de geração da Propriedade Intelectual. A origem do trabalho intelectual está em seu código-fonte. Qualquer que seja o código-objeto é um código contaminado, para efeito desse tipo de análise.

A indústria de combate ao plágio de programa de computador reforça a tese deste trabalho, desenvolvendo produtos voltados para a análise do código-fonte de programa de computador como o *CodeMatch*, o *CodeCross* e o *CodeDiff*, ferramentas integrantes do *CodeSuite*.[4]

Em todos os exemplos citados, o mais importante é observar que a indústria desenvolve ferramentas para tentar localizar plágio no código-fonte do programa de computador, e não no código-objeto, o que é condizente com a argumentação sustentada ao longo deste trabalho, que afirma que as análises de semelhanças entre códigos de programas de computador deve se dar, de fato, pelo código-fonte e não pelo código-objeto.

Artigos e publicações recentes apresentadas em convenções e conferências internacionais ao redor do globo, segmentam os trabalhos de análise de violação de direito autoral exclusivamente sobre o código-fonte. Como exemplo, podemos citar os seguintes trabalhos:

- A *State of art on source code plagiarism detection*,[5] de Mayank Agrawal e Dilip Kumar Sharma, apresentado na Segunda Conferência Internacional das Tecnologias de Computação da Próxima Geração, realizado na Índia (2016);
- *Review of source-code plagiarism detection in academia*,[6] de Matija Novak, apresentado na 39ª Convenção Internacional sobre Tecnologias de Informação e Comunicação, Electrônica e Microelectrônica, realizado na Croácia (2016);
- *Source code plagiarism detection: The Unix way*,[7] de Juraj Petrík, Daniela Chudá e Branislav Steinmüller, apresentado no 15º Simpósio Internacional de Inteligência de Máquina Aplicada e Informática, realizado na Eslováquia (2017).

Na verdade, todos os trabalhos pesquisados no repositório do IEEE, a maior organização técnica profissional do mundo para o avanço da tecnologia, se voltam para a análise do código-fonte do programa de computador e, sequer, citam a análise do código-objeto para questões relacionadas ao plágio de programas de computador.

A falta de uma metodologia padrão para os trabalhos de análise de semelhanças de códigos de programas de computador, onde este trabalho procura contribuir, é um dos principais fatores para a inexistência de uma salvaguarda especificada no inciso III do artigo 473 do Código de Processo Civil, onde o legislador declara: "III - a indicação do método utilizado, esclarecendo-o e demonstrando ser predominantemente aceito pelos especialistas da área do conhecimento da qual se originou;" (BRASIL, 2015).

Pondero que, neste aspecto, o trabalho técnico do analista forense, ou perito, deve garantir a melhor qualidade, de forma a neutralizar as possibilidades de dúvidas. Adicionalmente, todas as técnicas sugeridas no Capítulo 5, seguem as boas práticas das técnicas de aproximação sugeridas pelo instituto norte-americano NIST. Nesse sentido, a escolha por um modelo de estudos baseado em técnicas sugeridas pelo instituto NIST[8] se dá em virtude deste instituto ter a acreditação pelo INMETRO, que é uma das principais instituições de referência tecnológica e de qualidade do Brasil, razão pela qual o NIST é frequentemente referenciado nos documentos dos órgãos governamentais[9] e

2. Disponível em <http://www.planalto.gov.br/ccivil_03/leis/L9609.htm>. Acesso em 12 de junho de 2017.
3. A cadeia de custódia consiste um processo de documentar o histórico de uma evidência, com o objetivo de garantir o rastreamento dessas evidências utilizadas em processos judiciais.
4. Disponível em <https://www.safe-corp.com/products_codesuite.htm>. Acesso em 4 de abril de 2018.
5. Disponível em <http://ieeexplore.ieee.org/document/7877421/>. Acesso em 21 de maio de 2017.
6. Disponível em <http://ieeexplore.ieee.org/document/7522248>. Acesso em 21 de maio de 2017.
7. Disponível em <http://ieeexplore.ieee.org/document/7880355/>. Acesso em 21 de maio de 2017.
8. Disponível em <https://www.nsrl.nist.gov/approximate.htm>Acesso em 16 de junho de 2017.
9. Disponível em <http://dsic.planalto.gov.br/documentos/publicacoes/2_Guia_SICI.pdf.>. Acesso em 16 de junho de 2017.

jurídicos,[10] tendo sido frequentemente citado por estes, quando é desejado dar uma diretriz de referência robusta para o contexto.

Não existindo um procedimento padrão para esta atividade, e considerando as implicações técnicas apresentadas, a análise de violação de direito autoral deve ser concentrada exclusivamente sobre o código-fonte do programa de computador, que carrega consigo todas as características originais do trabalho intelectual de desenvolvimento do programa de computador.

Uma exceção se dá quando da análise de violação do Direito de Autor pelos elementos não-literais do programa de computador, que é apresentada na proposta de metodologia de análise deste trabalho.

A análise dos elementos não-literais torna-se substancialmente importante quando, sob as várias perspectivas de análise pelo código-fonte, não são encontrados elementos substanciais que evidencie a violação do Direito de Autor, sendo necessária outra abordagem de análise sugerida, sendo esta, a comparação da estrutura lógica operacional proposta no item 4.1.10 deste trabalho, pois os plagiadores também se tornaram infratores sofisticados.

## 2. Elementos conceituais

Os elementos conceituais constituem base fundamental para a compreensão do conteúdo apresentado neste trabalho. Nesse capítulo vamos introduzir alguns conceitos relacionados ao tema.

### 2.1. *Gui - Look and Feel*

Na excelente obra do Professor e Doutor Manoel J. Pereira dos SANTOS, intitulada "A Proteção Autoral de Programas de Computador", uma referência literária no trato do tema da Propriedade Intelectual, o capítulo 10 trata a Proteção da Interface de Usuário e a Teoria do *Look and Feel*.

O Doutor Manoel destaca que:

> A interface do usuário compreende basicamente os elementos que estão associados com a funcionalidade, permitindo ao usuário interagir com o programa, razão pela qual relaciona-se com o conceito de interatividade.[11]

Ele comenta que esses elementos estão relacionados às telas de vídeos dos programas, os menus, as janelas e demais recursos gráficos do programa de computador. E essas características associadas ao aspecto visual do programa de computador, ou seja, a interface do usuário, denota a expressão *Look and Feel*.

Nesse sentido, as características de *Look and Feel* estão relacionadas à concepção da interface gráfica do usuário, que abrangem as formas de como as janelas são apresentadas, as cores utilizadas, os desenhos, os formatos de letras utilizadas nesses elementos, que constituem o *Look*. Já o comportamento dos elementos gráficos, na forma dinâmica de sua operação, como o surgimento e desaparecimentos das janelas, características de abertura dos menus, comportamento dos botões de interação, constituem o *Feel*.

### 2.2. Elementos literais

O código-fonte e o código-objeto constituem os elementos literais da criação do programa de computador, desenvolvidos pelo autor sendo, portanto, objetos sujeitos a proteção pelo Direito de Autor.

Para esclarecer, a Lei nº 9.279/1996 que regula os direitos e obrigações relativos à Propriedade Industrial, estabelece que: "Art. 10. Não se considera invenção nem modelo de utilidade: [...] V - programas de computador em si;"[12] (BRASIL, 1996).

Como diretriz, o Direito de Autor protege os elementos literais, como o código-fonte e código-objeto, por 50 anos, estendendo-se a algumas exceções de elementos não-literais como a interface de usuário e a arquitetura. Por arquitetura, compreende-se a estrutura, a sequência e a organização do código do programa. Em resumo, são objetos de Direito de Autor o código-fonte, o código-objeto, a arquitetura e a interface do usuário.

10. Disponível em <https://www.trt13.jus.br/institucional/seguranca-da-informacao/documentos/cnj/diretrizes-gestao-si-2012>Acesso em 16 de junho de 2017.
11. SANTOS, M. J. P. A Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, p. 283, 2008.
12. Disponível em <http://www.planalto.gov.br/ccivil_03/leis/L9279.htm>Acesso em 31 de maio de 2017.

### 2.3. Elementos não-literais

Comumente, encontramos documentação da lógica do desenvolvimento do programa de computador por seus fluxogramas e nas especificações que surgem antes mesmo do desenvolvimento do código-fonte. Esses documentos de fluxogramas devem ser compreendidos como elementos não-literais, uma vez que são tidos como uma solução técnica desenvolvida para a resolução de um determinado problema em particular, que é utilizado como suporte para o desenvolvimento de um programa de computador. Em geral, a proteção dos elementos não-literais efetua-se por meio do sistema de patentes, mas há as exceções relacionadas à arquitetura dos programas de computador e à interface do usuário, cujas particularidades são verificadas, caso a caso, nas análises de plágio que possam constituir violação do Direito de Autor.

Apesar de o INPI não conceder patentes para programas de computador, são concedidas patentes para as invenções que sejam implementadas por um programa de computador. Logo, o regime das patentes protege os elementos não-literais por 20 anos, a partir da data do depósito da patente. Os elementos não-literais são as características técnicas da invenção, cujo direito emerge a partir do depósito da patente e posterior concessão pelo INPI.

Em resumo é objeto do depósito de patente a solução técnica para um problema técnico, que pode estar expressa nos algoritmos e outros elementos não-literais do programa de computador. Contudo, outras características não-literais são objetos de proteção pelo Direito de Autor, e a definição desses elementos não-literais constitui um grande desafio no tema da Propriedade Intelectual.

Encontramos na excelente obra do Professor e Doutor Manoel J. Pereira dos SANTOS, a proteção dos elementos não-literais, bem delimitadas, no Capítulo 7 de sua obra. O Doutor Manoel vai muito além da definição dos elementos não-literais, abrangendo aspectos do Direito de Autor na percepção do direito norte-americano, europeu e australiano.

De forma genérica, elementos não-literais seriam definidos como tudo aquilo que não está compreendido no código do programa de computador. Entretanto, o Doutor Manoel destaca a evolução da proteção dos elementos não-literais, originado do direito norte-americano, em que uma obra pode ser considerada contrafação de outra preexistente se for evidenciada a semelhança substancial com relação aos elementos protegidos da mesma.

Esse equilíbrio e cuidado na definição dos elementos não-literais tem enorme relevância no âmbito do Direito de Autor, razão pela qual o Doutor Manoel J. Pereira dos SANTOS dedicou um capítulo inteiro de sua obra para lidar com o tema. E a chave para esses casos é a interpretação da dicotomia ideia/expressão, e como isso se relaciona com os programas de computador. Não existindo atividade intelectual criativa na forma de expressão de determinada ideia, não há aplicação do Direito de Autor.

#### *2.3.1. Arquitetura*

O outro aspecto quanto aos elementos não-literais está relacionado à arquitetura do programa. De forma genérica, as literaturas[13] relacionadas ao tema definem a arquitetura como sendo a estrutura, sequência e organização do programa de computador. Essa definição, apesar de correta, é, contudo, muito vaga. O analista forense precisa conhecer bem do que se trata a arquitetura de um programa de computador, pois essa compreensão é de fundamental importância para a análise dos elementos não-literais, que exigem profunda abstração.

Conceitualmente, um programa de computador consiste de uma coleção de componentes organizados para resolver um determinado problema. Sua arquitetura consiste da organização do programa, incorporada em seus componentes, suas relações entre si e com o meio externo, e os princípios que orientam seu modo de funcionamento.

A descrição da arquitetura de um programa de computador é uma coleção dos artefatos que o compõem, sendo vários elementos considerados em sua arquitetura que, para o propósito da análise de violação de Direito de Autor, precisa ser muito bem compreendida.

A visão dessa arquitetura constitui toda a documentação que orienta o propósito para o qual o programa foi desenvolvido. Para proporcionar melhor compreensão da arquitetura precisamos subdividi-la em alguns subitens, sendo eles: arquitetura de negócios, arquitetura de dados, arquitetura de aplicação e arquitetura de tecnologia. O que é definido por esses elementos de arquitetura e o que é esperado como resultado de sua implementação no programa, ou seja, quais as entradas e saídas esperadas, e como os usuários interagem com elas?

Escrito de uma forma mais simples, quais seriam as definições dos princípios que constituem os elementos de arquitetura?

Os princípios que orientam o modo de funcionamento dos programas são as regras gerais e diretrizes, o gerenciamento dos dados, que informam e apoiam a forma como o programa se compromete a cumprir sua missão.

Nas empresas de desenvolvimento de programas mais maduras, os princípios da arquitetura são desenvolvidos pela equipe de arquitetos de sistemas em conjunto com os desenvolvedores do programa e, na maioria dos casos, também com as principais partes interessadas envolvidas no ambiente operacional. Em alguns casos, empresas clientes são convidadas a participar de um processo de melhoria do programa. É um trabalho complexo que, se bem administrado, conduz a uma consistente e clara orientação, ou forte influência, para as tomadas de decisões.

Então, com o propósito de fornecer um nível consistente e mensurável de informações relacionados à arquitetura de um programa, precisamos compreender os princípios de cada elemento de arquitetura, que orientam o modo de funcionamento do programa de computador.

13. SANTOS, M. J. P. A Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, p. 249, 2008.

*Arquitetura de negócio*

A arquitetura do negócio tem como fundamento principal maximizar o benefício para a empresa. Importante deixar claro que a empresa, nesse caso, é aquela que fará uso do programa, e não a empresa desenvolvedora do programa, pois o programa de computador foi desenvolvido para o cliente que, supõe-se beneficiar-se do programa.

As tomadas de decisões a partir da perspectiva do negócio têm maior valor a longo prazo do que as decisões tomadas em qualquer outra perspectiva. O retorno máximo do investimento requer decisões de gerenciamento de informações para aderir as prioridades de negócio da empresa. Este processo leva diretamente à escolha de, ao menos, uma dentre diferentes alternativas, todas candidatas a resolver um determinado problema.

Por exemplo, um programa de computador orientado ao gerenciamento logístico de cargas, levará em consideração fatores como custo com frete, seguro, requisitos de qualidade, entre outros aspectos, para a contratação de uma empresa de transporte de cargas. Por outro lado, uma vez contratada a empresa de transporte de cargas, o programa fará a gestão da tomada de decisão das rotas com base no menor percurso, melhor período para a viagem, histórico de menor tempo percorrido, podendo até chegar ao detalhe de análise de perfil do condutor, etc.

Por meio deste tipo de tomada de decisão do programa, o calendário de transporte é otimizado, reduzindo os custos com combustíveis, contribuindo com a diminuição da emissão de gases nocivos ao meio ambiente, e atendimento aos requisitos de qualidade para o consumidor. Perceba que, se uma empresa tem compromisso com o meio ambiente, com políticas fortes de adequação, este item não apenas contribui para uma tomada de decisão acertada, como torna a empresa ainda mais aderente às conformidades de políticas e normas ambientais, ou seja, maximizando os benefícios.

Os princípios da arquitetura de negócio mapeiam para as tomadas de decisões e requisitos de qualidade do programa que, espera-se estar bem documentado pelos arquitetos de sistemas nos requisitos de negócio, ou nos requisitos de qualidade.

*Arquitetura de dados*

A arquitetura de dados tem como principal fundamento a valoração dos dados, pois o objetivo da gestão dos dados com qualidade é auxiliar na tomada de decisões das empresas. Empresas maduras tratam os dados como um valioso ativo, pois dados precisos são fundamentais para decisões precisas e acertadas.

Os princípios envolvidos na arquitetura de dados levam em consideração que os dados, apesar de constituir um ativo da organização, devem ser compartilhados e, da mesma forma, devem estar facilmente acessíveis. Apesar de parecer simples, há uma certa complexidade envolvida neste item, pois a relação entre o valor dos dados, o compartilhamento e a acessibilidade aos dados têm perspectivas variadas sob o aspecto da segurança da informação dentro das empresas, e varia muito de empresa para empresa, pois elas possuem diferentes políticas de segurança da informação para lidar com esta questão.

Tomando com base esse cenário, muitas vezes as empresas desenvolvem procedimentos para prevenir, e corrigir, eventual produção de dados errados em seus bancos de dados, com o objetivo de melhorar os processos que possam contribuir para a produção de informações equivocadas, o que certamente contribui para influenciar de forma negativa as tomadas de decisões. A qualidade dos dados precisa ser mensurada e medidas precisam ser criadas para melhorar a qualidade dos dados de forma constante. Consequentemente, é provável que políticas e procedimentos também precisem ser desenvolvidos para lidar com a questão delicada da qualidade dos dados.

*Arquitetura de aplicação*

A arquitetura de aplicação tem como principal fundamento a premissa que os programas devem ser independentes de escolhas tecnológicas específicas e, portanto, devem operar nas variadas plataformas tecnológicas disponíveis no mercado. A independência dos programas de computador permite que eles sejam desenvolvidos, atualizados e operados da forma mais econômica possível. Este princípio exige padrões que apoiem a portabilidade para outras plataformas operacionais.

É fácil abstrair um cenário em que exista a necessidade de se criar programas para operar em diferentes plataformas tecnológicas. Considere uma

empresa como a SAP[14] que precisou desenvolver uma aplicação para seu público-alvo: as grandes corporações. A primeira percepção é que a SAP não tem controle sobre quais servidores de banco de dados o seu público utiliza. Então, ela precisou desenvolver um programa que funcionasse nas mais diversas plataformas operacionais de banco de dados. Porém, apoiada neste mesmo princípio, a empresa evoluiu quando compreendeu que as empresas que faziam uso de seu ERP precisavam também de um banco de dados. Abstraindo de outra forma, a SAP entendeu que ela era uma potencial motivadora de venda para outros *players* de banco de dados como a *Oracle*, a *Microsoft* e a IBM, por exemplo. Como resultado de um excelente trabalho de continuidade no desenvolvimento de arquitetura de aplicação, a empresa trabalhou em sua suíte que recebeu o nome de *Hana*.[15] A partir de então, a SAP tinha um ERP integrado com o seu próprio banco de dados, e os ganhos na integração com as outras bases de dados se traduziram em economia e eficiência para seus clientes que, ao licenciar seu ERP, não precisavam mais se preocupar com um banco de dados. Porém se já possuísse algum banco de dados, o processo de integração estaria facilitado.

Em outro exemplo menos complexo, esse princípio levou muitos desenvolvedores a fazer uso da tecnologia Java[16] em seus aplicativos, e os pacotes que surgiram posteriormente, semelhantes ao Java, como o Open JDK,[17] deram um alto grau de suporte à independência da plataforma para esses programas, pois eles podiam operar em plataformas distintas como Android, Windows e IOS, por exemplo.

A facilidade de uso do programa é outro princípio observado na arquitetura de aplicação, pois ela deve ser transparente para os usuários, de forma que eles possam se concentrar nas atividades principais do negócio, conhecido também como o jargão, em inglês, *core business*. Em outras palavras, a falta do entendimento deste princípio implica um ambiente menos produtivo para a empresa.

*Arquitetura de tecnologia*

A arquitetura de tecnologia tem como principal fundamento suportar as necessidades de alterações dos programas de computador ao longo de seu ciclo de vida, em resposta às necessidades do negócio que vivenciam constantes requerimentos de mudanças. O objetivo deste princípio é proporcionar um ambiente operacional no sentido de manter a empresa focada nos negócios e não na tecnologia. A arquitetura tecnológica desenvolvida deve proporcionar esse ambiente.

Uma necessidade de caráter comercial da empresa pode requerer uma melhoria técnica ou o desenvolvimento de subsistemas do programa de computador que está em operação. Por exemplo, o advento do SPED[18] Contábil trouxe uma necessidade de adequação dos programas de computador para todas as empresas do Brasil. As empresas que possuíam um ambiente tecnológico suportado por este princípio não tiveram grandes dificuldades em promover as alterações necessárias para a adequação de seus programas, mantendo o foco em seus negócios. Por outro lado, houve inúmeros casos de empresas que não conseguiram adequar seus programas para o SPED Contábil. Como consequência precisaram demandar grande esforço para cumprir com as exigências da Secretaria da Fazenda, desviando o foco do negócio para o cumprimento de uma exigência legal.

Voltando novamente ao exemplo da SAP, para uma empresa que possuía uma licença do seu ERP, ela precisou apenas desenvolver uma interface para o SPED Contábil, com as especificações desenvolvidas pela SEFAZ.[19]

### 2.4. Plataforma operacional

Nos dias atuais, o termo plataforma é utilizado de forma genérica para especificar elementos nos mais variados segmentos da indústria da tecnologia da informação. Por exemplo, para se fazer a distinção entre as linguagens de programação, são utilizadas expressões como plataforma Java, plataforma C, etc.

Da mesma forma, para subentender a especificação de suporte para os variados sistemas operacionais em que um programa de computador pode ser executado, é comum utilizar termos como plataforma *Window*s, plataforma *Linux*, plataforma *Mac*, etc., que remete para o sistema operacional em que o programa pode ser executado. O termo é também utilizado para diferenciar as especificações de processadores, ao utilizar jargões como plataforma *Intel*, plataforma *Motorola*, plataforma *Arm*, etc.

Neste trabalho, o termo plataforma refere-se ao sistema operacional em que um programa de computador pode ser executado. Isso porque no contexto dos estudos deste trabalho existe a abordagem de plágio de código de programa de computador que pode ser utilizado em diferentes sistemas operacionais, razão pela qual é utilizado o termo plataforma operacional.

## 3. Tipos de semelhanças

Um artigo publicado pelo DROPS[20] sob o título *Similarity in Programs* oferece uma abordagem conceitual sobre a similaridade de programas, explorando os aspectos sintático e semântico.

14. SAP, acrônimo de *System Analysis and Program*, é uma empresa de origem alemã, criadora de *softwares* de gestão para empresas. Disponível em <https://www.sap.com>. Acesso em 4 de abril de 2018.
15. Disponível em <https://news.sap.com/brazil/2015/09/09/mas-o-que-e-sap-hana/>. Acesso em 4 de abril de 2018.
16. Java é uma linguagem de programação multiplataforma desenvolvida pela empresa Sun Microsystems. Disponível em <https://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em 4 de abril de 2018.
17. O Open JDK, acrônimo de *Java Development Kit*, é um kit de desenvolvimento Java de código aberto para a edição *Standard* (Java SE), que recebeu apoio das empresas Oracle e IBM para a criação desta comunidade. Disponível em <http://openjdk.java.net/>. Acesso em 4 de abril de 2018.
18. SPED, acrônimo de Sistema Público de Escrituração Digital, foi instituído pelo Decreto nº 6.022, de 22 de janeiro de 2007, parte do Programa de Aceleração do Crescimento do Governo Federal (PAC 2007-2010). Disponível em <http://sped.rfb.gov.br/>. Acesso em 4 de abril de 2018.
19. SEFAZ, acrônimo de Secretaria de Estado da Fazenda, é um órgão vinculado ao Ministério da Fazenda responsável pelo controle das receitas e das despesas de cada um dos Estados e do Distrito Federal. Disponível em <https://portal.fazenda.sp.gov.br/>. Acesso em 4 de abril de 2018.
20. Disponível em <http://drops.dagstuhl.de/opus/volltexte/2007/968/>. Acesso em 1º de maio de 2017.

O artigo refere-se ao aspecto sintático como sendo a forma de representar o código-fonte, normalmente uma sequência de caracteres formando uma estrutura de texto mais complexa. Nesta abordagem, os pesquisadores observaram que a similaridade na representação admite a semelhança entre códigos em diferentes níveis de abstração, que correspondem aos diferentes modos de se observar os programas, como por exemplo, as similaridades entre declarações em programas, blocos, classes e arquiteturas.

E no aspecto semântico, o artigo sustenta que os programas podem ser considerados similares quando a principal semelhança está em seu comportamento, que pode ser definido pelas funções implementadas pelo programa. Entretanto, o consenso dos pesquisadores foi de que existem poucas formas de se fazer a comparação semântica, mas, em um sentido mais amplo, é possível apontar dois tipos de representação que podem ser utilizadas neste tipo de comparação: a operacional e a lógica.

A semelhança operacional detalha os passos de execução de um programa, enquanto a semelhança lógica estabelece o significado de cada linha de programação, suas variáveis e sua lógica. Essa é uma outra forte razão para que a comparação seja realizada sobre o código-fonte e não o código-objeto, pois o significado de cada linha pressupõe o entendimento da programação, as declarações realizadas no código-fonte e a lógica associada ao código, pois já vimos anteriormente que o código-objeto é um formato de saída padrão da linguagem para ser tratada e executada pelo processador.

A doutrina também tem contribuído no sentido de reforçar a necessidade de análise pelo código-fonte. Na excelente obra que recebeu o título *Software Evolution*, os autores belgas Tom Mens e Serge Demeyer destacam o consenso na comunidade internacional quanto à importância de se detectar os clones, que são as semelhanças entre os códigos. Este é um equívoco comum observado em muitas literaturas, que referenciam o clone como sendo plágio. Primeiramente é importante deixar claro que clone não é plágio. Uma definição muito mais apropriada é a do Doutor Manoel J. Pereira dos SANTOS, que trata das diferenças entre clones, plágio e cópia servil no Capítulo 5 de sua extraordinária obra "A Proteção Autoral de Programas de Computador", conceituando e caracterizando os clones dessa forma:

> Um clone pode ser definido como um programa que é funcionalmente equivalente, porém distinto, de outro programa de computador. Na terminologia técnica, contudo, o clone não apenas executa substancialmente as mesmas funções, como imita a aparência e o funcionamento da criação anterior.[21]

E um exemplo clássico da definição do Doutor Manoel é o editor de textos *Microsoft Word*, do Pacote *Office*, e um de seus clones, o editor de textos *Writer*, do pacote *LibreOffice*. O *Writer* é um clone do *Word*, e ele não é plágio do *Word*.

É necessário ter muito cuidado na interpretação apropriada da palavra clone nas literaturas, pois muitas vezes o termo quer referenciar o plágio de código de programa de computador.

Importante salientar que, em todas as condições abordadas na obra *Software Evolution*, de 347 páginas, destaca-se o fato de todo estudo e técnicas mencionadas, serem aplicado sobre o código-fonte, em nenhum momento sobre o código-objeto, como sustentamos no presente trabalho.

### 3.1. Hipóteses de semelhanças

A Lei nº 9.609/1998 prevê as hipóteses de semelhanças de programa de computador no artigo 6°, onde o legislador define:

> Art. 6º Não constituem ofensa aos direitos do titular de programa de computador:
> [...]
> III - a ocorrência de semelhança de programa a outro, preexistente, quando se der por força das características funcionais de sua aplicação, da observância de preceitos normativos e técnicos, ou de limitação de forma alternativa para a sua expressão;
> IV - a integração de um programa, mantendo-se suas características essenciais, a um sistema aplicativo ou operacional, tecnicamente indispensável às necessidades do usuário, desde que para o uso exclusivo de quem a promoveu.[22] (BRASIL, 1998).

As hipóteses de semelhanças admissíveis são aquelas linhas de código-fonte de programas que podemos identificar como sendo mandatórias, sem a qual os programas não podem ser compilados ou, se forem, produzirão erros. Exemplo: a declaração *#include <stdio.h>* em um programa escrito em linguagem C.

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

22. Disponível em <http://www.planalto.gov.br/ccivil_03/leis/L9609.htm>. Acesso em 6 de junho de 2017.

Devido a essa obrigatoriedade, a semelhança para funções como essa são admissíveis quanto a declaração *#include <stdio.h>* estiver presente nos programas.

Outra hipótese está relacionada ao endentamento das linhas de código-fonte. A técnica de endentamento é utilizada para organizar a lógica dos programas, por exemplo, quando da abertura e fechamento de chaves na linguagem de programação C. Observe a linha de programa abaixo, extraído do programa prog1.c.

```
int main() {
```

Valendo-se da técnica de endentamento, o código poderia ser escrito na forma apresentada a seguir:

```
int main()
{
```

A chave isolada na linha imediatamente posterior é uma forma de isolar o código, organizando-o de forma endentada, para melhor separação, técnica que proporciona melhor compreensão do código pelo programador.

Ao encontrar técnicas de endentamento como esta, é provável encontrar muitas linhas idênticas de códigos distintos. Essa condição também é admissível quando da constatação de semelhanças encontradas em código-fonte.

Outro aspecto importante quanto às hipóteses de semelhanças está relacionado às linhas e espaços vazios. Embora a incidência de linhas e espaços vazios serem, na maioria dos casos, admissíveis, a similaridade dessa semelhança ao longo de dois códigos sob análise pode mapear para um cenário de plágio.

Existem profissionais que produzem uma espécie de assinatura digital em seus códigos. Por exemplo, há códigos-fonte em que o autor insere sempre dois espaços em vez de um espaço, entre os argumentos do código. Ao analisar dois códigos com essa característica, um analista forense experiente certamente perceberá essa sutil semelhança. Esse aspecto terá grande importância para o leitor na metodologia de análise de semelhanças entre códigos-fonte de programas de computador, proposta neste trabalho e apresentada a seguir.

## 4. Metodologia de análise proposta

Durante minha palestra no Seminário de Combate Preventivo e Repressivo aos Crimes Cibernéticos, a convite do Departamento de Segurança da FIESP,[23] evento promovido pelo Ministério Público, a FIESP e a FEBRABAN,[24] na data de 21 de junho de 2017, uma das questões que surgiram do público foi: Qual a maturidade da análise e investigação forense digital no Brasil?

Minha resposta foi dada no sentido de explicar que o Brasil enfrenta, de fato, grande dificuldade nos casos de investigação forense, justificada principalmente pela falta de uma metodologia padrão para os trabalhos de análise forense no País.

Nessa mesma linha, a falta de uma metodologia para os trabalhos de análise de semelhanças de código de programa de computador contribui para um cenário onde variadas perspectivas são observadas, por diferentes ângulos, de diferentes profissionais com experiências variadas sobre o tema, tendo como consequência, diferentes métodos de abordagem, resultando em trabalhos de avaliação com diferentes conteúdos e procedimentos e, consequentemente, resultados distintos. A sugestão da metodologia proposta, a seguir, tem por objetivo preencher essa lacuna e contribuir para o Judiciário, que tem a difícil tarefa de julgar cada caso.

As técnicas apresentadas neste capítulo, são baseadas nos estudos das técnicas de aproximação sugeridas pelo instituto NIST,[25] em virtude deste instituto ter reconhecimento mundial e, também, ter a acreditação pelo INMETRO, que é uma das principais instituições de referência tecnológica e de qualidade do Brasil, razão pela qual o NIST é frequentemente referenciado nos documentos dos órgãos governamentais[26] e jurídicos,[27] tendo sido frequentemente citado por estes, quando é desejado dar uma diretriz de referência robusta para o contexto.

São também resultados dos estudos da análise técnica conhecida como *abstraction-filtration-comparison test*, que se refere a um teste criado judicialmente nos Estados Unidos, e usado para determinar se existem semelhanças substanciais entre os elementos não-literais de dois ou mais programas.

### 4.1. Metodologia de análise

O processo da metodologia de análise proposta possui uma sequência de fases, que parte de uma análise mais simples, avançando para fases mais complexas e de difícil percepção, requerendo maior esforço intelectual dos analistas forenses à medida que avança nas fases seguintes desta metodologia de análise.

Então, como resultado quase que natural da abordagem contida neste trabalho, o primeiro passo da metodologia proposta consiste em analisar os elementos literais do código-fonte de programa de computador, numa primeira fase de análise e, em uma segunda fase, analisar os elementos não-literais.

A análise dos elementos não-literais é fundamentalmente importante, principalmente quando levamos em consideração a existência de

23. FIESP é o acrônimo de Federação das Indústrias do Estado de São Paulo. Disponível em <http://www.fiesp.com.br/agenda/seminario-de-combate-preventivo-e-repressivo-aos-crimes-ciberneticos/>. Acesso em 4 de abril de 2018.
24. FEBRABAN é o acrônimo de FEderação BRAsileira dos BANcos. Disponível em <https://portal.febraban.org.br/>. Acesso em 4 de abril de 2018.
25. Disponível em <https://www.nsrl.nist.gov/approximate.htm>. Acesso em 16 de junho de 2017.
26. Disponível em <http://dsic.planalto.gov.br/documentos/publicacoes/2_Guia_SICI.pdf.>. Acesso em 16 de junho de 2017.
27. Disponível em <https://www.trt13.jus.br/institucional/seguranca-da informacao/documentos/cnj/diretrizes-gestao-si-2012>. Acesso em 16 de junho de 2017.

plágio em plataformas operacionais distintas, em que a semelhança poderá ser observada no aspecto comportamental do programa de computador, e não pelo código-fonte, podendo este ter sido objeto de um processo de recodificação de código de programa de computador para utilização em outra plataforma operacional.

Adicionalmente, há que considerar também o cenário em que é utilizada outra linguagem de programação, com o objetivo de ofuscar ainda mais a semelhança. A seguir são apresentadas as dez fases de comparação propostas neste trabalho.

#### *4.1.1. Fase 1: Leitura e comparação geral dos códigos*

A primeira fase da metodologia de análise de semelhança proposta constitui de uma comparação inicial através da completa leitura dos dois arquivos de código-fonte a serem analisados.

Há duas motivações em colocar os dois arquivos de código-fonte lado a lado. A primeira é pelo fato de as evidências iniciais fortes já aparecerem logo nessa primeira fase, como comentários repetitivos, marcas d'água, *strings* proprietárias, entre outros sinais fortes que evidenciam o plágio já nesta fase inicial de análise, como os espaços duplos que citamos ao final do Capítulo 3, que funcionam como uma espécie de marca d'água do autor.

Por vezes, ao analisar um código-fonte de programas escrito em C++ ou *Python*, de minha autoria, um analista forense questionaria o que seriam as *strings* WA1, WA2, WA3, e assim por diante. Esta é uma codificação de meu controle para identificar como os blocos foram ordenados em meus programas. Um bloco dentro de outro bloco teria a *string* WA4.1, ou seja, um sub-bloco 1 dentro do bloco 4. Essa prática constitui o que eu chamo de marcas d'água, que identificam minha autoria nos controles lógicos de meus programas. Programadores experientes, frequentemente criam marcas d'água para este propósito.

Outra motivação de colocá-los lado a lado se dá em observar todo o processo de transformação que ocorrerá, na medida que as técnicas de análise posteriores sejam aplicadas. Muitos arquivos de código-fonte parecem distintos num primeiro olhar, entretanto, quando aplicadas algumas técnicas de análise, eles começam a realçar as semelhanças não observadas num primeiro momento. Após a completa leitura de ambos os códigos, é executada a fase 2, que consiste na extração dos metadados.

#### *4.1.2. Fase 2: Extração dos metadados*

A extração dos metadados tem por objetivo verificar a existência de fragmentos de informações da origem do autor do código-fonte do programa de computador.

Nessa segunda fase, a extração dos metadados é aplicada não apenas sobre o código-fonte do programa de computador, como também sobre o código-objeto, arquivos de documentação, arquivos de lógica e demais arquivos existentes da solução implementada.

As informações contidas nos metadados podem mapear para uma origem diferente da fonte que reivindica a autoria intelectual do programa de computador e, por esse motivo, deve ser efetuada a extração dos metadados de todo e qualquer arquivo vinculado à solução implementada por derivação de código-fonte de programa de computador.

Como exemplo, a extração dos metadados do arquivo deste trabalho, pela ferramenta *exiftool*,[28] traz os seguintes resultados:

```
┌─[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
```

28. *Exiftool* é uma interface de linha de comando utilizada para ler e escrever meta-informação em uma variedade de tipos de arquivos. Os metadados são lidos a partir de arquivos de origem e impressos em forma legível para a console, também conhecido por terminal ou *prompt* de comando.

| | |
|---|---|
| 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) |

As informações de metadados coletadas trazem algumas informações importantes de cada arquivo, valiosas para a interpretação de sua autoria. Essas informações não estão no texto do corpo do documento. É possível verificar, por exemplo, o número de revisões que o documento sofreu desde a sua criação, no exemplo, 556 revisões. Há a data da última impressão deste arquivo, que se deu no ano de 2011, porém ele tem data de criação no ano de 2012. Como isso é possível?

Ao compilar essas informações, considerando também a possibilidade de que esse arquivo possa ter sido transportado de um dispositivo para outro, podemos perceber que, apesar do registro do autor reportar como sendo de Washington Almeida, é pouco provável que ele seja o criador do arquivo. De fato, este modelo de documento foi cedido para o autor deste trabalho, Washington Almeida, pela Escola Politécnica da USP.

Uma outra informação valiosa que os metadados trazem são as versões dos programas de computador. Seria muita coincidência que duas empresas reivindicando a autoria de um determinado programa, tivessem os mesmos critérios de documentar o formato da versão como, por exemplo, v.1.00a.

Normalmente, esses critérios de documentar a versão, estão ligados a uma norma interna da empresa definindo os padrões para nomeação de documentos. Em determinadas situações, a ausência dessas normas internas em uma das empresas, associadas à forma como os programas foram confeccionados, constituem fortes indícios de violação.

#### *4.1.3. Fase 3: Remoção dos comentários*

Um padrão de ação relativamente comum observado na violação do Direito de Autor de programa de computador ocorre quando o violador remove as linhas de comentários, na tentativa de ofuscar o código-fonte original, o que altera significativamente parte do código-fonte do programa de computador num primeiro olhar.

Esta fase visa deixar ambos os arquivos do código-fonte livres da poluição dos comentários. A remoção dos comentários contribuirá para realçar as semelhanças mais aparentes, ainda que o código-fonte tenha sido alterado. Removidos todos os comentários, segue a fase da comparação sintática no item seguinte.

#### *4.1.4. Fase 4: Reordenação do código-fonte*

O quarto passo da metodologia de análise de semelhança proposta consiste em reordenar o código-fonte, logo após a remoção dos comentários.

Considere as seguintes sequências de códigos abaixo:

```
Código-A                Código-B
int main ()             int
{                       main()
printf ("Teste")        printf (
return 0;               "Teste");
}                       return
                        0;
                        }
```

Note que ao reordenar um dos códigos acima, percebemos que eles são exatamente iguais. Nas linguagens de programação que usam a notação dos delimitadores "{" e "}" para separar as declarações e os blocos de código, como a linguagem C, por exemplo, as restrições no uso de espaços em branco são mínimas. Da mesma forma, o endentamento do código-fonte pode ser propositadamente alterado para conotar a expressão de um código-fonte diferente. No exemplo dos códigos acima, alterações sintáticas foram realizadas de forma a reestruturar o código-fonte para este parecer estruturalmente diferente. Num primeiro momento eles estariam, ainda mais diferentes se houvesse a existência dos comentários, com o objetivo de ofuscar o código, razão pela qual os comentários são removidos na fase anterior.

A reordenação do código visa identificar a similaridade estrutural de ambos os conjuntos de arquivos do código-fonte. A similaridade estrutural é realçada à medida que os elementos do código são reordenados. Após reordenar os arquivos do código-fonte, segue a comparação sintática.

#### *4.1.5. Fase 5: Comparação sintática*

O quinto passo da metodologia de análise de semelhança proposta consiste em fazer a comparação sintática entre os arquivos do código-fonte considerados semelhantes. A semelhança sintática consiste em analisar os dois conjuntos de códigos-fonte, colocando-os lado a lado e, particioná-los em vários blocos, sendo que apenas as linhas de código do mesmo bloco são comparadas.

Nesta fase é realizada a comparação das declarações que ocorrem em cada bloco, como declarações de variáveis, de funções, de classes, e de constantes. Em casos de códigos de programas extremamente longos, o perito pode utilizar a técnica do uso da função *hash* para *strings* repetitivas, de forma a simplificar o processo para o analista forense, porém a utilização dessa técnica deve ser muito bem docu-

mentada, principalmente levando-se em consideração que a leitura do material produzido será realizada por pessoal não técnico da área do Direito, como advogados e juízes.

Nessa fase, a análise das semelhanças requer maior esforço intelectual do analista forense ou perito, além de sua excelente capacidade de abstração, pois é provável que blocos contendo as declarações de variáveis, de funções, de classes, e de constantes tenham sido completamente modificadas com o objetivo de fazer parecer uma criação única, quando na realidade se constata o aproveitamento disfarçado, ofuscado, mascarado dos elementos da criação preexistente.

Elencadas as semelhanças sintáticas, passamos para a fase seguinte, que consiste em fazer a análise das dependências.

### *4.1.6. Fase 6: Análise das dependências*

O sexto passo consiste em analisar as dependências das variáveis definidas nos códigos. A forma mais simples de analisar essas dependências consiste na utilização de grafos simples, que tem por objetivo evidenciar as semelhanças nas dependências dos dados, assim como de controles presentes nos códigos-fonte dos programas de computador.

Embora o violador tente se valer de técnicas de ofuscação de código, utilizando nomes de variáveis diferentes, a análise de grafos identificará a lógica utilizada pelas diferentes variáveis encontradas nos códigos analisados.

Consideremos os seguintes códigos abaixo, que executam, ambos, o cálculo fatorial de um número inteiro:

- Código 1:

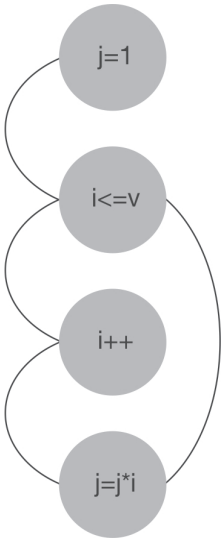
```
int i, j=1;
for (i=1; i<=VALUE; i++)
j=j * i
```

- Código 2:

```
int fatorial (int n) {
if (n == 0)
return 1;
else
return n * fatorial (n – 1)
}
```

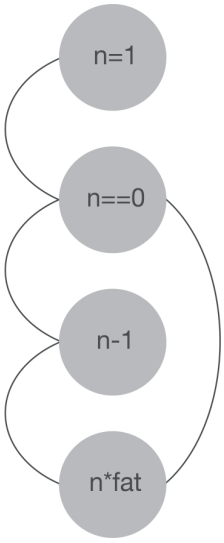
Em um primeiro olhar para os dois códigos, observamos que eles parecem ser muito diferentes. Vamos então analisá-los, considerando suas dependências, visualizando-os em um gráfico de grafos.

**Figura 2**
**Grafo do código 1**




*Fonte: O autor.*

**Figura 3**
**Grafo do código 2**




*Fonte: O autor.*

Observando os grafos de ambos os códigos, percebemos que eles são exatamente iguais, apenas executam o cálculo fatorial na ordem inversa um do outro, porém, revelando a similaridade de suas dependências quanto às variáveis "i" e "n".

A dependência de controle é identificada, no código-1, pela expressão "i<=VALUE", enquanto que a dependência de controle identificada no código-2 é representada pela expressão "n==0". Da mesma forma, a dependência de dados é identificada, no código-1, pela expressão "j=j*i", enquanto que a dependência de dados identificada no código-2 é representada pela expressão "n*fatorial".

Note que nesta fase, a análise já requer um nível de percepção e de abstração mais apurado do analista forense, pois a evidência pode não estar tão perceptível em uma análise inicial. Identificadas as semelhanças pela análise das dependências, passamos para a próxima fase, que consiste na análise do fluxo dos dados.

#### *4.1.7. Fase 7: Análise do fluxo dos dados*

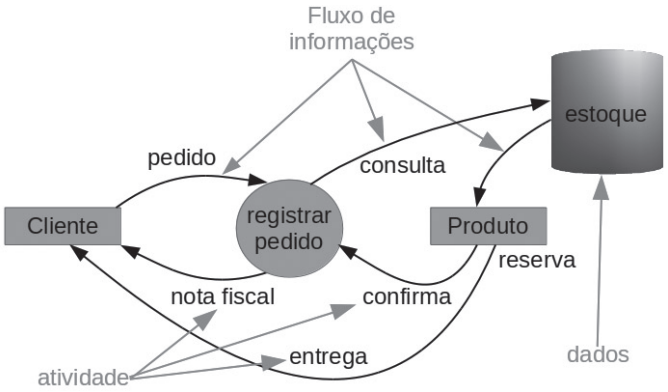
O sétimo passo consiste em analisar os fluxos de dados do código-fonte, considerando também os fluxos de procedimentos ou funções que são invocados dentro desses fragmentos.

A utilização de um diagrama de fluxo de dados realça, graficamente, a representação que descreve o problema a ser resolvido. Essa técnica visa evidenciar as semelhanças sob duas perspectivas: a semântica e a estrutural. Sob o aspecto semântico, visa identificar como o comportamento do programa foi modelado, e sob o aspecto estrutural, visa identificar como foi modelada a estrutura dos dados processados pelo programa.

As componentes utilizadas na análise do fluxo de dados são os processos e o fluxo. O processo transforma as entradas em saídas, e o fluxo propriamente dito, identifica a movimentação dos dados.

A figura 4 abaixo é um exemplo em como o fluxo das informações podem ser encadeadas para que se tenha melhor compreensão do problema a ser resolvido, no caso do exemplo apresentado na figura, realça o fluxo dos passos simplificados para a emissão de uma nota fiscal.

**Figura 4**
**Exemplo de diagrama para análise de fluxo dos dados**




*Fonte: O autor.*

Os dados são transformados em cada passo antes de mover para o próximo estágio. Esse procedimento ajuda o analista forense a compreender o que acontece com os dados durante a execução do programa. Fluxos semelhantes implicam semelhanças semânticas de códigos.

A fase seguinte considera as hipóteses legais de semelhanças.

#### *4.1.8. Fase 8: Remoção das hipóteses de semelhanças legais*

O oitavo passo da metodologia de análise de semelhança proposta consiste em separar as semelhanças admissíveis das semelhanças não admissíveis. Implica avaliar as hipóteses de semelhanças enquadradas no inciso III do artigo 6º da Lei nº 9.609/1998, onde o legislador declara serem legalmente admissíveis:

> III - a ocorrência de semelhança de programa a outro, preexistente, quando se der por força das características funcionais de sua aplicação, da observância de preceitos normativos e técnicos, ou de limitação de forma alternativa para a sua expressão;[29] (BRASIL, 1998).

Um exemplo clássico dessa fase é a análise de código-fonte de programas voltados para o SPED, Sistema Público de Escrituração Digital, que deve, obrigatoriamente, obedecer às resoluções do Conselho Federal de Contabilidade.[30]

29. Disponível em <http://www.planalto.gov.br/ccivil_03/leis/L9609.htm>. Acesso em 12 de junho de 2017.

30. Disponível em <http://cfc.org.br/legislacao/>. Acesso em 4 de abril de 2018.

Programas desenvolvidos para o SPED Contábil, SPED Fiscal, SPED do PIS e COFINS, Modelo-3, E-Social,[31] entre outros, como programas voltados para a emissão de notas fiscais em estabelecimentos comerciais, que obedecem às notas técnicas do SEFAZ, também se enquadram nas hipóteses de semelhanças legais e devem ser removidas nessa fase do processo de análise.

### *4.1.9. Fase 9: Análise da semelhança semântica*

A análise da semelhança semântica é fundamentalmente importante para os casos em que um programa tenha sido recodificado em outra linguagem de programação para, por exemplo, sua utilização em outra plataforma operacional. Esse cenário é mais comumente observado quando são utilizadas técnicas de engenharia reversa pelo violador, para se obter os binários do código que ele deseja plagiar.

A semelhança semântica diz respeito ao comportamento dos programas observados. Nessa fase, uma parte do código-fonte de um programa A é semanticamente semelhante a outra parte do código-fonte de um programa B, se, e somente se, o programa B conter a funcionalidade do programa A.

A estrutura e a sintaxe do código-fonte não são alvos de análise nessa fase. O que define a similaridade é a semântica envolvida no processamento computacional, ou seja, o resultado funcional das operações. Por exemplo, o *loop* definido por "*while* (a > b)", pode ser alterado para "*while* (b < a)" ou "*while* (x > y)", ou ainda "*while* (y < x)", este último bem mais sensível na percepção porque ocorre a inversão lógica do comando *while* junto com a alteração das suas variáveis.

Uma vez que a separação por blocos foi realizada na fase 5, esses detalhes aparecem de forma mais realçadas para o analista forense. Outro perfil de similaridade, que demanda maior expertise nessa fase, é a substituição de funções por fragmentos de código similares e vice-versa.

Consideremos os dois fragmentos de código-fonte utizados no item 4.1.6, representados novamente abaixo, cujos mesmos estão escritos em linguagem de programação C++.

- Código 1:

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

O fragmento de código acima executa o cálculo fatorial de um número. Como visto no item 4.1.6, esse código pode ser reescrito pelo seguinte código abaixo.

- Código 2:

```
int fatorial (int n) {
if (n == 0)
return 1;
else
return n * fatorial (n – 1)
}
```

Sob o aspecto semântico, os códigos apresentados acima são exatamente iguais, pois a similaridade observada está definida na semântica envolvida no processamento computacional de ambos, ou seja, os códigos fazem exatamente as mesmas coisas.

Nessa fase da análise é fundamental que o analista forense tenha vasto conhecimento e entendimento dos aspectos relacionados à linguagem de programação utilizada, e também excelente capacidade de abstração das características semânticas, quando analisar os fragmentos dos códigos.

### *4.1.10 Fase 10: Comparação da estrutura lógica operacional*

A comparação da estrutura lógica operacional constitui, talvez, a fase de análise mais difícil para o analista forense, dentro do processo da metodologia proposta neste trabalho.

31. O Decreto-lei nº 6.022 que instituiu o Sistema Público de Escrituração Digital - SPED, trouxe a obrigatoriedade de transformar todas as informações contidas nos livros contábeis e fiscais tradicionais em arquivos digitais. Esses arquivos precisam obedecer a um formato padronizado e predefinido e servem para informatizar a relação entre empresas e a Receita Federal ou a Secretaria da Fazenda - SEFAZ, uma vez que as informações são transmitidas para esses órgãos, alimentando um banco de dados padronizado. As obrigações do SPED foram agrupadas em cinco grandes projetos: Nota Fiscal Eletrônica - NF-e; SPED Fiscal, via Escrituração Fiscal Digital - EFD; SPED Contábil, via Escrituração Contábil Digital - ECD; SPED Folha, via Escrituração Fiscal Digital Social - EFD Social; e a Escrituração Fiscal Digital para o PIS e COFINS (EFD PIS/COFINS). O livro Registro de Controle da Produção e do Estoque, conhecido como Modelo-3, destina-se à escrituração dos documentos fiscais e dos documentos de uso interno da empresa, correspondentes às entradas e às saídas, à produção, bem como às quantidades referentes aos estoques de mercadorias, obrigatório para os estabelecimentos industriais ou a eles equiparados pela legislação federal e para os estabelecimentos atacadistas, integrando o sistema SPED, e denominado "Bloco K". O "Bloco K" é uma das partes de informação do SPED Fiscal para os tributos ICMS e IPI, que se constitui no livro eletrônico de Registro de Controle da Produção e do Estoque. Disponível em <https://www.fazenda.sp.gov.br/sped/downloads/GUIA%20PR%C3%81TICO%20DA%20EFD%20-%20Vers%C3%A3o%202.0.12.pdf>. Acesso em 4 de abril de 2018.

A complexidade se resume no fato de não haver evidências substanciais que caracterizem o plágio de programa de computador nos exames realizados nas fases anteriores, e porque o analista forense deve constatar as eventuais evidências através de um processo de abstração.

Neste processo de abstração, o analista forense deve exercer uma atividade intelectual, levando para sua reflexão o elemento da análise, e isolado-o dos fatores comuns que estão relacionados a ele, como por exemplo as fases de análises anteriores.

Em particular, o trabalho de comparação desta fase consiste em determinar se os elementos não-literais de um programa de computador foram copiados, o que caracteriza o plágio. Entretanto, diferentemente das fases anteriores em que a análise se baseia, fundamentalmente, sobre os aspectos relacionados com o código-fonte do programa de computador, o trabalho de análise desta fase leva em consideração uma possível cópia dos elementos não-literais para outra plataforma operacional.

Essa última fase de comparação levou em consideração os estudos do instituto NIST, particularmente do seu modelo de aproximação definido no documento de nome *NIST Special Publication 800-168*,[32] e também o estudo dos elementos não-literais sob duas perspectivas: a arquitetura e o teste de abstração-filtragem-comparação.

A primeira perspectiva é relacionada com a arquitetura que, como visto detalhadamente no item 2.3.1, constitui a estrutura, sequência e organização do programa, e tem por objetivo estabelecer uma divisão entre a dicotomia ideia/expressão de modo a se certificar que a forma de expressão do programa preexistente não tenha sido copiado. Dito de outra forma, a proteção de direitos autorais é dada apenas à forma em que as ideias são expressas, e não as próprias ideias. Com o objetivo oferecer melhor entendimento, dividimos a arquitetura, no item 2.3.1, em quatro famílias, sendo elas a arquitetura de negócios, arquitetura de dados, arquitetura de aplicação e arquitetura de tecnologia.

Quanto à arquitetura de negócios, o analista forense deve concentrar seus esforços nas características das empresas desenvolvedoras, seus critérios para as definições de negócios, e compreender como o programa supostamente infrator trata das questões relacionadas à arquitetura de negócios.

Em relação à arquitetura de dados, os critérios de análise do analista forense devem se basear nos requerimentos para a especificação da modelagem dos dados, e muitas vezes esses elementos não são encontrados na documentação dos sistemas, à exceção de empresas desenvolvedoras muito maduras em seus sistemas de gestão, que possuem procedimentos para documentação de programas de computador, mesmo nesta fase de detalhamento da arquitetura.

Já com relação à arquitetura de aplicação, o analista forense deve elevar o nível de abstração neste item, avaliando o programa da empresa supostamente contrafator, no sentido de comprovar seu compromisso com os dois princípios da arquitetura de aplicação: a portabilidade tecnológica e a facilidade de uso.

E quanto à arquitetura de tecnologia, o analista forense deve avaliar se os critérios de análise se basearam na maturidade do programa supostamente contrafator, em relação à sua capacidade de resposta às variadas necessidades do negócio, que exige capacidade de rápida adaptação do programa de computador.

A segunda perspectiva é o estudo da análise sob o aspecto do teste de abstração-filtragem-comparação. Este teste, em particular, teve origem em 1992 durante a apuração de um caso envolvendo a *Computer Associates International Inc.* e a *Altai Inc.*,[33] que consiste em separar, de maneira metódica e organizada, um programa de computador de acordo com seus diferentes níveis de generalidade, ou diferentes níveis de abstração. Desde então essa análise tem sido amplamente adotada pelos tribunais dos Estados Unidos e também reconhecido por tribunais ao redor do globo.

O Doutor Manoel J. Pereira dos SANTOS referencia este teste, em sua obra,[34] como "Teste da abstração-filtragem-comparação", que consiste de um sofisticado processo contendo três passos para determinar a similaridade substancial dos elementos não-literais de um programa de computador. De forma bem resumida, cada parte separada é examinada para filtrar os elementos do programa que não são protegidos. Então, remove-se os elementos que estão desprotegidos, e os elementos protegidos remanescentes são comparados com o programa supostamente infrator, para determinar se os elementos substanciais do programa original foram objetos de plágio dos seus elementos não-literais.

A aplicação do teste de abstração-filtragem-comparação é complexa porque a comparação da estrutura lógica operacional não é descrita em nenhuma forma de linguagem de programação ou métodos de desenvolvimento de programas de computador, o que seria familiar aos programadores e cientistas da computação, que seriam os especialistas técnicos mais indicados para ajudar os tribunais nesses casos.

Logo, os especialistas forenses qualificados devem desempenhar um papel importante nesse processo de abstração, justamente em virtude deste nível de análise não compreender as tarefas técnicas convencionais de análise de determinada linguagem de programação ou de processos de desenvolvimento de programas.

Outro fator que aumenta a complexidade dessa fase se deve ao tamanho do código-fonte da maioria dos programas comerciais, pois o esforço envolvido na abstração e filtragem de praticamente qualquer programa é muito grande, e produziria um enorme número de variações. Dessa forma, o melhor método de análise é aquele em que o especialista forense limita a parcela do programa que o autor alega ter sido copiado.

Porém, devido à deficiência técnica dos analistas forenses para a avaliação dessa fase específica, implica dificuldades para alcançar

32. Disponível em <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-168.pdf>. Acesso em 13 de julho de 2017.

33. Disponível em <https://cyber.harvard.edu/people/tfisher/IP/1992%20Altai.pdf>. Acesso em 4 de abril de 2018.

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

resultados objetivos, pois há muitos detalhes envolvidos neste processo de análise. A abstração pode resultar na comparação de centenas de aspectos diferentes de um programa de computador, se o analista forense não for cuidadoso.

Então, a aplicação mais apropriada da comparação, nesta fase, deve considerar o refinamento da tarefa, com o objetivo de reduzir a complexidade técnica.

É sobre essa redução que trata as diretivas do instituto NIST e, dessa forma, com base nos estudos mencionados, o entendimento é que os elementos envolvidos na comparação da estrutura lógica operacional devem ser filtrados com base nos seguintes critérios: eficiência, fatores externos, convenção da linguagem utilizada, processo e domínio público.

a. Eficiência;
   A funcionalidade de determinada forma de expressão diz respeito à eficiência do programa. Sua existência dentro do programa se deu por requisitos de eficiência. A forma de executar determinada função é expressa por razões de eficiência.

b. Fatores externos:
   A forma de expressão do elemento foi criada por requisito externo, sendo necessário a especificação de um formato de dados para interoperar com outro programa. Por exemplo, um programa ERP pode ter o requisito externo de ter que exportar um arquivo de um de seus módulos, para interfacear com outro programa externo, como o SPED Contábil, cuja especificação de seus blocos de dados é determinada pela Secretaria da Fazenda, ou seja, um fator externo.

c. Convenção da linguagem utilizada:
   É o cenário encontrado onde a expressão do elemento foi criada devido a maneira convencional de escrever algo na linguagem de programação específica daquela solução. Por exemplo, bibliotecas presentes nas linguagens para expressar formas de sua estrutura dinâmica, na máquina em que o programa é executado.

d. Processo:
   O elemento, neste nível particular de abstração, é um processo, e não uma forma de expressão de criação que seja objeto de proteção. A identificação de processos visa eliminar uma substancial parcela das abstrações que são objetos de processos funcionais dentro de uma organização, frequentemente refletido nos programas de computador.

e. Domínio público:
   A identificação do elemento, nesse caso, não é objeto de criação, mas uma incorporação de domínio público. Por exemplo, elementos de arquitetura que exercem ações sobre o mecanismo de gerência de QoS,[35] que são capazes de tomar decisões dinâmicas a respeito da taxa de utilização dos recursos e de carga do sistema, são implementações de domínio público, cuja especificação está definida na RFC 2212.[36]

Então, remove-se os elementos não-literais encontrados pela filtragem desses cinco elementos, e os elementos não-literais remanescentes serão objeto da análise da estrutura lógica operacional.

Importante ressaltar que qualquer proteção de elementos criados pela eficiência, ou por fatores externos, ou processos, deve ser objeto de proteção por patentes, se esse for o caso, e não objeto de proteção pelo Direito de Autor.

Mas nesse cenário, consideramos que um programa está baseado em outro programa que copiou recursos não-literais do programa preexistente, e a comparação da estrutura lógica operacional deve ser utilizada para determinar se muito dos elementos não-literais foram copiados, resultando em violação de direitos autorais.

A correta aplicação desta comparação deve levar em consideração que as abstrações devem ser muito específicas e precisas, do contrário é muito fácil o especialista forense se perder no processo de análise desta fase, mesmo sendo um analista forense ou perito experiente.

Da mesma forma, as abstrações devem ser acompanhadas de convenções gráficas consistentes para todas as exposições que o analista forense abstrair, pois como são abstrações, a ideia dessa abstração precisa estar muita clara para qualquer profissional que venha a avaliar as evidências descobertas nesta última fase de análise.

Uma vez identificada a ideia que deu origem à forma de expressão, avalia-se a proteção de direitos autorais sobre essa forma de expres-

35. QoS é o acrônimo do termo proveniente do inglês *Quality of Service*, que se refere à qualidade dos serviços de dados, vídeo e voz, particularmente relacionados à capacidade de ajuste automático da banda, de forma garantir a qualidade dos serviços de comunicação.
36. Disponível em <https://tools.ietf.org/html/rfc2212>. Acesso em 13 de julho de 2017.

são, ou seja, avalia-se como está expressa a estrutura da ideia dentro da lógica operacional do programa de computador avaliado, o que poderá evidenciar a violação do Direito do Autor.

## 5. Jurisprudência

Os tribunais do Brasil e do mundo têm firmado entendimento quando ao conteúdo apresentado neste trabalho, sobre os aspectos de proteção no contexto do código-fonte, a arquitetura e as formas de expressão dos programas de computador.

Os casos que seguem têm relação direta com os estudos e o conteúdo apresentado neste trabalho.

### 5.1. Caso *Mandic* vs. *Intervale* - Brasil[37]

A Intervale criou um *website* praticamente idêntico ao da *Mandic*, fornecendo ao usuário, inclusive, o mesmo sistema oferecido pela *Mandic*. A *Intervale* não comprovou possuir programa, o que evidenciava a utilização de propriedade alheia.

O Tribunal de Justiça do Estado de São Paulo julgou a Apelação Cível nº 122.616-4/6 em 18/06/2012, e a decisão do Desembargador Relator Ênio Santarelli Zuliani levou em consideração os elementos que constituem a arquitetura, como a estrutura, sequência e organização, destacando que: "Verdade é que se constatou a ocorrência de cópia dos itens "estrutura", "sequência" e "organização" do programa fonte [...]" (SÃO PAULO, 2012).

### 5.2. Caso *Data Access* vs. *Powerflex Services* - Austrália[38]

O *Data Access* é o editor do pacote *Dataflex*. A *Powerflex* criou o pacote da *Powerflex* para complementar o pacote da *Dataflex*. Porém, para tornar o *Powerflex* compatível com o *Dataflex* a empresa usou certas macros e palavras reservadas utilizadas pela *Dataflex*. No entanto, o código de serviço para as macros e as palavras reservadas eram diferentes. Também foi de entendimento pelas partes que o código-objeto usado pelos dois programas para os comandos e as palavras reservadas não eram, necessariamente, o mesmo, conforme argumentamos ao longo deste trabalho.

A *Data Access* afirmou que a *Powerflex* tinha violado os direitos autorais nas palavras reservadas, algumas macros e uma tabela de compressão de dados. A corte compreendeu que a tabela de compressão de dados era objeto de proteção por direitos autorais, pois independente do programa de computador, a tabela era uma forma de expressão original.

### 5.3. Caso *Mitel* vs. *Iqtel* - Estados Unidos[39]

A *Iqtel* começou a fabricar o controlador de chamadas IQ200+ e elaborou um conjunto de instruções de códigos de comando para ativar os recursos do seu controlador de chamadas. Embora os controladores de chamadas *Iqtel* e *Mitel* forneçam muitos dos mesmos recursos, para identificar os recursos do seu controlador, os registros selecionados da *Iqtel* eram diferentes dos registros da *Mitel*. A *Iqtel* usou "descrições" e "valores" idênticos, onde as funções do IQ200+ foram as mesmas utilizadas pelo controlador *Smart-1* da *Mitel*.

Para produzir um controlador de chamadas compatível com o controlador da *Mitel*, o *Iqtel* copiou os códigos de comando da *Mitel* em três aspectos importantes. Primeiro, a *Iqtel* programou o IQ200+ para aceitar códigos de comando da *Mitel* e traduzi-los para o código de comando *Iqtel* correspondente. A *Iqtel* então nomeou este recurso com o nome *Mitel Translation Mode*.

Assim, o apelo que nos interessa nesse caso exigia avaliar se a *Mitel* poderia comprovar que os códigos de comando copiados pela *Iqtel* estavam protegidos por direitos autorais. O tribunal concluiu que os códigos de comando da *Mitel* não possuíam expressões protegidas porque a expressão nos códigos de comando não possuía a originalidade necessária.

## 6. Conclusão

O desenvolvimento do presente trabalho possibilitou uma análise de como uma metodologia pode orientar os especialistas forenses, e a justiça, nos casos das análises de semelhanças entre códigos de programas de computador, com resultados mais assertivos. A metodologia proposta permite que profissionais do meio jurídico, como advogados e juízes, mapeiem uma direção quando envolvidos neste campo de análise, de forma a obter resultados mais consistentes nos processos envolvendo violação do Direito de Autor de programas de computador.

De um modo geral, os profissionais do meio jurídico envolvidos com este tipo de análise, em sua grande maioria operadores do Direito, possuem pouco conhecimento das linguagens de programação que são utilizadas no desenvolvimento de um programa de computador, e uma metodologia estruturada traz contribuição valiosa no sentido de auxiliá-los na difícil tarefa de lidar com este assunto de elevada complexidade técnica. Nesse sentido, o desenvolvimento de novas formas de análise, um método que funcione como um guia, pode contribuir não apenas com os profissionais da esfera jurídica, como também os especialistas forenses, que podem se valer de um modelo padrão para a análise de semelhanças entre códigos de programas de computador.

Esse ponto torna-se muito relevante em virtude dos profissionais da esfera jurídica não terem a obrigação do entendimento sobre questões técnicas relacionadas com programação e com temas técnicos específicos da área da tecnologia da informação, contudo, são eles que carregam a grande responsabilidade de julgar cada caso.

37. Disponível em <https://viniciustini.jusbrasil.com.br/artigos/121943976/copy-paste-de-websites-violacao-ao-direito-do-autor>. Acesso em 14 de julho de 2017.
38. Disponível em <http://www.austlii.edu.au/au/journals/SydLawRw/1998/12.html>. Acesso em 14 de julho de 2017.
39. Disponível em <http://digital-law-online.info/cases/44PQ2D1172.htm>. Acesso em 14 de julho de 2017.

Nos processos que transitam nos tribunais de justiça, é evidenciado que os especialistas forenses não trabalham sob uma metodologia única nas análises de semelhanças entre códigos de programas de computador, o que sustenta a justificativa deste trabalho de pesquisa, no esforço pela busca por um método padronizado que torne o trabalho técnico, e também complexo, mais objetivo, contribuindo para que os resultados sejam alcançados com qualidade e eficiência.

Nesse sentido, a utilização do método apresentado neste trabalho colabora para a sugestão de um padrão que permita os especialistas forenses realizarem seus trabalhos de forma unificada. Além disso, diminui o tempo de análise de semelhanças, à medida que o especialista forense torna seu trabalho metódico e repetitivo, contribuindo dessa forma para a celeridade nos julgamentos.

Para dar esta contribuição, o presente trabalho chega à conclusão desta tese, sustentado por dois pilares fundamentais: o primeiro, que a análise de semelhanças de códigos de programas de computador deve ser realizada sobre o código-fonte e jamais sobre o código-objeto. O segundo, que há a necessidade de uma metodologia de análise de semelhanças de códigos de programas de computador que seja uma orientação única para os profissionais que atuam nos casos envolvendo a análise de semelhanças de códigos de programas de computador.

Por meio da aplicação da metodologia apresentada neste trabalho, utilizando o processo de classificação, comparação, análise e síntese, o especialista forense poderá extrair o entendimento necessário que o auxiliará alcançar a verdade dos fatos, nos casos de violação de Direito de Autor de programas de computador.

## Referências bibliográficas


DUNTERMANN, J. *Assembly Language Step by Step: Programming with Linux. 4.* Indianapolis: Wiley Publishing, 1265 p, 2009.

HIDE, R. *The Art of Assembly Language.* 2. Indianapolis: No Startch Press, 1443 p, 2010.

KOSCHKE, R. *Survey of Research on Software Clones.* In *Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, july,* 2006.

MOTA, A. M.; GOYA, D. H. Técnicas para Análise de Similaridade de Código de *Software* em Litígios de Propriedade Intelectual. São Paulo: Universidade de São Paulo - USP.

SANTOS, M. J. P. A Proteção Autoral de Programas de Computador. Rio de Janeiro: Lumen Juris, 2008.

TAIRAS, R.; GRAY, J.; BAXTER, I. *Visualization of Clone Detection Results.* In *Proc. OOPSLA workshop on Eclipse technology exchange,* 50–54 [33, 34], 2006.

WALENSTEIN, A. *et al. Similarity in Programs.* In KOSCHKE, R.; MERLO, E. and WALENSTEIN, A., *editors, Duplication, Redundancy, and Similarity in Software, number* 06301 *in Dagstuhl Seminar Proceedings*. Dagstuhl, Germany. *Internationales Begegnungs- und Forschungszentrum für Informatik - IBFI,* Schloss Dagstuhl, Germany. Disponível em <http://drops.dagstuhl.de/opus/volltexte/2007/968>. Acesso em 1 de maio de 2017.

ZEIDMAN, B. *Detecting Source-Code Plagiarism – tools and algorithms for finding plagiarism in source code.* Dr. Dobb's Journal, july, 2004. Disponível em <http://www.ddj.com/architect/184405734>. Acesso em 1 de maio de 2017.