Técnicas para Análise de Similaridade de Código de Software em Litígios de Propriedade Intelectual

Ana Maria Mota[1], **Denise H. Goya**[2]

[1]Escola Politécnica – Engenharia Elétrica
Universidade de São Paulo (USP), SP - Brasil

[2]Departamento de Ciência da Computação – Instituto de Matemática e Estatística
Universidade de São Paulo (USP), SP - Brasil

anamariamota_68@hotmail.com, dhgoya@ime.usp.br

Resumo. *Neste trabalho, realizamos uma breve discussão sobre conceitos de similaridade de códigos de programas de computador, apresentamos uma classificação para as várias técnicas existentes para detecção de similaridade e relacionamos algumas ferramentas que podem auxiliar no trabalho de um perito judicial, em casos de litígio envolvendo infração de propriedade intelectual de programas.*

1. Introdução

Com a expansão da internet e do comércio eletrônico, a propriedade intelectual tem sofrido forte abalo. Os litígios relacionados a infrações de direitos autorais têm crescido, e com isto a necessidade de desenvolver técnicas que auxiliam na prevenção e detecção do roubo da propriedade intelectual.

[Culwin and Lancaster 2000] afirmam que plágio é roubo de propriedade intelectual, onde se inclui a utilização de código de programa sem permissão ou referência, como sendo um dos métodos de plágio. Entende que os métodos desenvolvidos na luta contra o plágio pode ser dividido em prevenção e detecção e que é necessária a combinação de ambos. Os métodos de prevenção extrapolam os esforços de trabalho do universo científico e envolvem a sociedade como um todo na mudança de atitude. Prevenir inclui fazer com que as pessoas entendam o plágio como um ato imoral, anti-ético e, acima de tudo, criminoso.

A prevenção, portanto, recai em ações e implantação de estratégias de longo prazo, que, além de demandarem tempo, dependem também de políticas públicas que amparem a proteção da propriedade intelectual.

De outro lado, há os métodos de detecção, que apresentam resultados de curto prazo e que vem sendo usados no auxílio às questões de litígios.

Em sua maioria, as técnicas de detecção de plágio realizam a comparação de dois ou mais documentos, procurando analisar o grau de similaridade entre eles. De posse de uma medida de similaridade e de uma descrição da técnica usada para obter tal medida, um julgador poderá avaliar quem tem razão em um processo de litígio de propriedade intelectual.

Neste trabalho, apresentamos uma breve análise do conceito de similaridade e sua classificação semântica e sintática. Em seguida, são focadas as técnicas de análise de

similaridade de código de software, ou seja, técnicas que otimizam a avaliação do grau de similaridade de documentos eletrônicos, dado que eles são códigos de programas. Para cada técnica, relacionamos alguns exemplos de ferramentas que podem ser usadas por um perito que precise avaliar a possível derivação de um software.

Antes, porém, iniciamos com a apresentação de algumas questões legais no Brasil e procedimentos nos tribunais norte-americanos, que fornece algum conhecimento importante para que o perito judicial tome decisões acertadas em sua atuação.

2. Questões Legais

No Brasil, o reconhecimento da autoria de programas de computador se enquadra no regime jurídico de Direito Autoral, de acordo com a "Lei de Software" nº 9.609, de 1998, seu regulamento, Decreto nº 2.556/98, e com a "Lei de Direito Autoral" nº 9.610/98.

Para que fique assegurada a exclusividade de exploração de um programa, seu autor pode opcionalmente registra-lo junto ao Instituto Nacional da Propriedade Industrial – INPI. O reconhecimento desse registro é internacional e tem prazo de validade de 50 anos. Não é impedimento, no entanto, que um software não registrado seja objeto de ações jurídicas de violação de direitos autorais.

Os programas de computador desenvolvidos para funcionarem exclusivamente embarcados em máquinas ou equipamentos podem ser objeto de proteção via patente, segundo a Lei nº 9.279/96, quando se enquadram em Invenções Relacionadas com Programas de Computador – IRPC. Nesse caso, a abrangência é apenas nacional e por prazo máximo de 20 anos.

O depositante do registro ou patente é o único responsável pela escolha dos "Documentos de Programa" a serem anexados no pedido de registro. Tais documentos podem ser depositados em caráter sigiloso e, eventualmente, serão usados para comprovar, em juízo, a autoria do programa envolvido em processos de uso indevido.

Conforme a Resolução INPI 58/98, a documentação técnica será composta pela listagem integral, ou parcial, do programa-fonte e, ainda, memorial descritivo; especificações funcionais internas; fluxogramas e outros dados capazes de identificar e caracterizar a originalidade do programa.

No nosso entender, essa forma de documentação pode se ajustar ao critério de aceitação de provas *"Look and Feel"*, usado principalmente na década de 1980 nos tribunais dos Estados Unidos, em casos de litígio de software. Em [Richards 2002], são relatados casos em que houve reconhecimento de violações de direitos autorais de programas, ainda que tenham ocorrido traduções de uma linguagem de computador para outra. Estrutura geral dos programas e fluxogramas foram levados em conta na análise de algumas das ações julgadas.

Ainda nos Estados Unidos, o caso *"Computer Associates versus Altai"*, de 1992, criou um precedente e estabeleceu o critério *"Abstraction, Filtration, Comparison"*. Conforme descrito em [Hollaar 2002], o primeiro passo consiste em abstrair os elementos do programa de computador em diferentes níveis.

No nível mais baixo de abstração, um programa de computador pode ser entendido como um conjunto de instruções, que se organizam hierarquicamente em módulos,

descritos num outro nível de abstração. A função de cada um desses módulos e, por fim, a função final do programa, compreedem os níveis mais altos de abstração.

Na fase de filtragem, que se segue à de abstração, são separados os elementos protegidos dos não protegidos (como aqueles de domínio público). A forma como esta etapa será cumprida depende fundamentalmente de qual nível de abstração será considerado.

Na fase de comparação, os elementos protegidos são comparados para que se possa averiguar quão similar é o código litigado.

Vale notar que, nos Estados Unidos, programas de computador podem ser patenteados. Lá, o teste *"Abstraction, Filtration, Comparison"* tem sido empregado em casos em que não há uma cópia exata do programa, porém há a suspeita de derivação não autorizada.

2.1. Nossas Interpretações

Tendo em vista que, no Brasil, o software não precisa ser registrado para que esteja protegido pela Lei de Direito Autoral, documentações nos mais diferentes níveis de abstração poderiam ser apresentadas para comprovação de autoria, em casos de litígio. Incluiríamos aqui desde código binário e código-fonte até especificações do programa em linguagem formal, como UML (nos seus vários níveis de abstração), ou qualquer outra forma descritiva, ainda que informal.

Uma boa argumentação, seja de defesa ou de acusação, poderia concluir sobre a similaridade (ou não) de dois programas. Basta notar que um programa num nível relativamente alto de abstração pode ter a mesma descrição de outro, mesmo que, num nível mais baixo, diferenças importantes sejam perceptíveis.

Nos casos de litígio em que a cópia de código não for evidente, cabe ao perito judicial detectar precisamente **o que** deve ser comparado. Somente depois de concluir sobre o que será comparado, torna-se possível a seleção da ferramenta ou do método mais adequado para prosseguimento da análise. Em sistemas altamente complexos, pode ser prudente selecionar mais de um nível de abstração para realizar comparações.

3. Tipos de Similaridade

Em trabalho relativamente recente, [Koschke 2007], o autor é claro ao afirmar que os pesquisadores ainda não chegaram a um consenso para definir exatamente o significado de similaridade, nem de clone e redundância de código. Com uma definição fechada sobre o que é similaridade de código de software, poderíamos classificar corretamente as técnicas e pensar em uma avaliação comparativa do desempenho das ferramentas disponíveis.

Por ora, o que podemos fazer é nos basear em pesquisas como o de [Walenstein et al. 2007] que delinearam um caminho. Segundo eles, antes de classificar similaridade o desafio é responder o que é, como medir e como escolher um instrumento de medição da similaridade. Após esta reflexão, os autores concluem que há dois tipos de similaridade, listadas nas próximas duas subseções, de acordo com o método a ser utilizado para comparar os itens. Nós adotaremos essas classificações, para chegarmos à descrição das ferramentas, adiante.

3.1. Semelhança Representacional – Sintática

A semelhança representacional refere-se ao programa como sendo uma seqüência de caracteres formando uma estrutura de texto complexa. A similaridade pode ser definida em termos de suas formas, propriedades ou características desta representação. Pode-se detectar a existência de similaridade em diferentes níveis de abstração, que correspondem às diferentes visões de programa:

- similaridade entre sentenças de programa;
- em blocos de programas;
- no nível de classes;
- no nível de unidades de programa;
- ou em nível arquitetural do programa.

3.2. Semelhança Comportamental – Semântica

Os programas podem ser considerados similares, quando a principal semelhança está em sua semântica, isto é, em seu comportamento, que pode ser definido pelas funções que estes programas implementam. A dificuldade está em definir medidas de similaridade semântica. Por exemplo, ao avaliar o contexto de clonagem, questiona-se bastante, se semelhanças existentes em parte dos códigos (blocos) podem ser considerados clones ou se é necessário que a semelhança esteja representada em blocos bem definidos. No entanto, analisando a similaridade semântica num sentido mais amplo, é possível apontar dois tipos de representação que pode ser utilizado neste tipo de comparação.

Semelhança Funcional. Dois programas podem ser considerados análogos se implementarem uma função similar. Ou seja, a similaridade funcional pode ser descrita como sendo a semelhança entre as entradas relacionadas com suas saídas.

Semelhança de Execução. Neste tipo de semelhança, será observada a seqüência de execução das instruções do programa, por exemplo em *bytecode* Java ou código *Assembly*. Nestes casos será necessário encontrar uma correspondência entre as instruções dos programas executáveis.

Avaliando o comportamento, a semântica do programa pode ser de um dos tipos abaixo, dentre outros mais, onde cada um tem sua forma específica e podem ser similares, não precisamente o mesmo:

- denotacional, onde o efeito gerado pelo programa interessa mais que a forma como foi produzido;
- operacional, onde se detalha os passos de execução de um programa;
- lógica, onde se estabelece o significado de cada sentença, suas variáveis e sua lógica;

3.3. Algumas Armadilhas

Podemos citar exemplos extremos que indicam o quão é difícil se chegar a uma resposta correta com relação à similaridade ou não de dois códigos.

Na Figura 1, encontra-se um exemplo de código que exibe dois trechos que são praticamente idênticos no nível textual (semelhança sintática). No entanto, no momento da execução, eles podem apresentar comportamento completamente diferentes, dependendo de qual é o tipo do objeto instanciado. As linhas 4 a 6, por exemplo, podem tratar

o resultado de uma consulta a um banco de dados qualquer; as linhas 10 a 12 poderiam tratar recursos de uma rede de computadores, cujos recursos fazem parte de uma coleção arbitrária. Além dessa diferença comportamental, a variável *sum* é global no primeiro bloco, enquanto que no segundo é de escopo local, o que remete a outra diferença significativa no momento da execução.

Figura 1. Semelhança textual com comportamento absolutamente distinto

```
1  int sum = 0 ;
2
3  void foo (Iterator iter){
4    for ( item = first (iter) ; has more (iter) ; item = next (iter) ) {
5      sum = sum + value ( item ) ;
6    }
7  }
8  int bar (Iterator iter){
9    int sum = 0 ;
10   for ( item = first (iter) ; has more (iter) ; item = next (iter) ) {
11     sum = sum + value ( item ) ;
12   }
13 }
```

Por outro lado, há casos em que trechos de código com pouca semelhança textual podem reproduzir exatamente o mesmo comportamento. Na Figura 2, o código do lado esquerdo é semanticamente idêntico ao do lado direito. Um bom programador que tivesse a intenção de camuflar uma cópia saberia gerar um código a partir do outro. Nesse exemplo, os nomes das variáveis foram mantidos; caso tivessem sido alterados, seria ainda mais improvável a detecção da derivação.

Figura 2. Códigos semanticamente idênticos, com diferenças sintáticas significativas

```
while ((x = pi[t-1]) != '(' && x != '+' && x != '-') {
  posf[j++] = x;
  --t;
}
```

```
while (1) {
  x = pi[t-1];
  if (x == '(' || x == '+' || x == '-')
    break;
  --t;
  posf[j++] = x;
}
```

Os exemplos acima se posicionam em dois extremos que uma ferramenta de detecção automática teria dificuldades em ser bem sucedida. Nesses casos, a análise humana e a boa experiência do analisador tornam-se imprescindíveis.

4. Medidas de Similaridade

Admitindo-se diferentes graus de similaridade, tornam-se necessários modelos e formas de medição diferentes para similaridades sintáticas e semânticas.

4.1. Semelhança Representacional – Sintática

Similaridade Textual. Determina o quanto duas seqüências de caracteres(strings) são semelhantes. Como a distância de Levenshtein que é dada pelo número mínimo de operações necessárias para transformar uma string em outra.

Similaridade por Métricas. Esta métrica utiliza consultas de similaridades que tem por objetivo procurar por objetos em um conjunto que segundo um critério de similaridade, sejam mas parecidos ou mais distintos.

Similaridade Baseada em Características. Esta similaridade pode ser medida observando a quantidade de correspondência entre as características existentes em dois programas, de suas lista de elementos e propriedades. Neste caso, podemos incluir várias medidas de similaridade, no entanto é importante estabelecer o que estamos comparando para gerarmos a base de comparação entre diferenças e semelhanças.

Informação Compartilhada. Os programas são considerados documentos-texto e então são aplicados métodos da Teoria da Informação de Shannon, onde a idéia essencial é a de considerar dois programas como mensagens de texto. Se os programas são independentes, então a quantidade de informação transmitida em suas concatenações será proporcional ao tamanho total de suas concatenações. Se um dos programas for idêntico, então a cópia extra adicionará apenas um bit de informação a mensagem na concatenação. A similaridade é medida pela quantidade de informação que os programas compartilham.

4.2. Semelhança Comportamental – Semântica

Como já foi referido anteriormente, há um consenso quanto à dificuldade de encontrar medidas de similaridade semântica. Algumas idéias para medi-la são:

Semelhança na Curva de Execução. Traços de execução podem ser representados por uma curva, onde se avalia o comportamento de um programa em um determinado tempo e espaço. As curvas resultantes são comparadas para avaliar o nível de similaridade existente.

Semelhança na relação de Entradas e Saídas. Neste teste verifica-se a relação do conteúdo de entrada com o de saída, sendo que os dois programas geram o mesmo resultado.

Distância Semântica. Mede-se o custo de mutação de um programa para o outro. Aplicando o conceito de diferenças de Levenshtein no espaço semântico, observam-se as operações de mutações e seus custos.

Distância na Equivalência da Abstração. Verifica o nível de abstração que um programa precisa alcançar antes que dois programas se tornem idênticos. Removem-se detalhes não essenciais, para se chegar a essência. Esta técnica se aplica à fase de abstração do teste *"Abstraction, Filtration, Comparison"*, citado na Seção 2.

Similaridade pelo Grafo de Dependências do Programa. A maioria das técnicas já citadas dependem que a ordem textual seja preservada até um certo ponto. Quando programadores tentam camuflar cópias reordenando trechos, sem comprometer o funcionamento, podem ser bem sucedidos em não serem detectados por ferramentas. Um grafo de dependências do programa cria uma representação das dependências dos dados e controle entre as sentenças.

5. Ferramentas para Detecção e Medição de Similaridade

Passamos a descrever algumas ferramentas que medem a similaridade de programas, seguindo a classificação apresentada na Seção 4.

5.1. Ferramentas para Detecção de Semelhança Representacional – Sintática

Nesta classe, encontramos maior número de opções. Vamos relacionar algumas delas.

Similaridade Textual. As ferramentas mais antigas que encontramos se enquadram nesta classificação. O Duploc de [Ducasse et al. 1999] é um exemplo. Linhas inteiras são comparadas; para melhorar o desempenho, é usada uma função de hash sobre os textos das linhas.

Em [Baker and Manber 1998], a popular ferramenta *diff* do Unix e outras duas desenvolvidas pela autora (*siff* e *dup*, que parametrizam o código de programa

e calcula diferenças) são usadas para deduzir similaridades em códigos-fonte em Java, a partir de bytecodes.

Em [Zeidman 2004], há uma breve descrição das ferramentas Plague, Yap e JPlag, que usam a estratégia:

1. remover comentários, identificadores (como nomes de variáveis e de funções), e espaços em branco;
2. substituição de sentenças do código por seqüências de tokens;
3. comparação das seqüências de tokens para encontrar similaridade.

A empresa comercial de Zeidman vende um pacote chamado CodeSuite, do qual faz parte o CodeMatch, que permite a comparação de nomes de identificadores e de comentários, pois esses podem conter indícios importantes de cópia de código. Também possibilita a detecção de linhas reordenadas (que alguns programadores aplicam na tentativa de dificultar a detecção por ferrarmentas de automatização). Segundo o autor, seu programa tem sido usado por peritos em casos de litígio de propriedade intelectual. Cópia de demonstração pode ser obtida em http://www.safe-corp.biz/products_codesuite.htm.

A ferramenta JPlag possui serviço gratuito na Web, que permite que usuários enviem arquivos para serem comparados, a partir do endereço http://www.ipd.uka.de:2222/. Nessa página, há referência de que o JPlag também tem sido usado por peritos judiciais.

Em [Koschke 2007], são descritas e comparadas algumas ferramentas baseadas em comparação de tokens, como o *dup* de [Baker and Manber 1998] e CCFinder, de [Kamiya et al. 2002]. Uma técnica denominada AST (*abstract syntax tree*) também é revisada no mesmo trabalho. Trata-se de uma melhoria da técnica de token e um exemplo de ferramenta é o CloneDr.

Similaridade por Métricas. A ferramenta CLAN, de [Merlo 2007], é um exemplo nessa classe. Ela foi desenvolvida para detectar plágio entre trabalhos de alunos, em projetos escritos em C e C++. A técnica consiste em extrair métricas e fazer uma análise de similaridade espectral. São sete as métricas usadas no CLAN:

1. número de chamadas de funções
2. número de variáveis locais usadas ou definidas
3. número de variáveis não-locais usadas ou definidas
4. número de parâmetros
5. número de sentenças
6. número de desvios
7. número de loops

De acordo com o autor, os resultados obtidos com o CLAN foram suficientes para desencorajar a prática de plágio pelos alunos, dentro do curso onde ele é professor. Em [Verco and Wise 1996], há uma abordagem semelhante pela contagem de atributos dos programas e inclui um comparativo de outras ferramentas como Accuse, Yap3 e o sistema de Faidhi-Robson.

Similaridade Baseada em Características. [Schleimer et al. 2003] apresentam uma classe de algoritmos para calcular a "impressão digital" de documentos. Uma das aplicações resultantes é o sistema Moss de detecção de plágio de programas, incluindo um serviço on-line em http://theory.stanford.edu/∼aiken/moss/. O Moss trata várias linguagens de programação, como C, C++, Java, C#, Python, Visual Basic, Javascript, Fortran, Pascal, Perl, Matlab, entre outras. Os autores usam o

conceito de *window* de tamanho w, como w hashes consecutivos de k-gramas em um documento (ou seja, evoluíram o conceito de k-gram).
[Lukashenko et al. 2007] apresentam uma visão geral de métodos e ferramentas de detecção de plágio, em especial de plágio de textos. Porém tratam também do Moss e do JPlag, que se focam em detecção de similaridade em códigos de programas. Nesse trabalho, os autores já notaram que ambos (Moss e JPlag) não tratam bem comparações intra-arquivos. Na pesquisa de [Roy and Cordy 2007], a mesma observação é feita. JPlag e Moss apresentam desempenho semelhante em detecção e em precisão, mas não são adequados para comparação entre vários pares de programas. Em termos de desempenho, tornam-se impraticáveis para detecção em sistemas muito grandes.

Informação Compartilhada. Em [Chen et al. 2004], há a descrição de outra técnica de detecção de plágio; um sistema Web chamado SID, tal qual o Moss e JPlag, permite a comparação online de códigos suspeitos de cópia, no endereço http://genome.uwaterloo.ca/SID/. A técnica dos autores é bastante diferente das demais, pois se baseia a teoria da informação de Shannon, para criar uma métrica baseada na complexidade de Kolmogorov, que normalmente é usada em outros contextos de recuperação de informação. Nesse mesmo artigo, são comparados os desempenhos do Moss e JPlag contra o SID.

5.2. Ferramentas para Detecção de Semelhança Comportamental – Semântica

Pouco já foi implementado nesta categoria e consideramos que é uma área de pesquisa interessante a explorar.

Semelhança na Curva de Execução. Esta parece ser uma técnica útil para avaliar se um código é o resultado da ofuscação de um original. Quer dizer, se dois programas apresentam praticamente mesmo comportamento (um foi derivado de outro e o código foi maquiado para tentar ofuscar a cópia), as curvas serão semelhantes. Não encontramos implementação dessa classe, para detecção de similaridade. Em [Walenstein et al. 2007] são citados trabalhos de outras áreas de aplicação, que usam curvas de execução e que poderiam servir de ponto de partida para averiguar plágio.

Semelhança na relação de Entradas e Saídas. Não encontramos ferramentas.

Distância Semântica. Também não encontramos ferramentas.

Abstraction Equivalence Distance. Também não encontramos ferramentas.

Similaridade pelo Grafo de Dependências do Programa. Clones de programas podem ser identificados por subgrafos isomorfos, detectados por algoritmos de aproximação. Em [Krinke 2001], o Duplix é um exemplo de ferramenta deste tipo; seu desempenho é comparado com o de outros em [Koschke 2007].

5.3. Outras Ferramentas

Encontramos outras duas ferramentas que podem ser de interesse na área forense, mas que não puderam ser enquadradas nas classificações anteriores.

Em [Wong et al. 2001] é apresentada uma metodologia que visa identificar a entidade que criou um determinado programa.

Já em [Lee and Hwang 2007] é apresentado um sistema preliminar, para reconhecer quando uma infração de direitos autorais está sendo cometida.

Pela nossa compreensão, essas técnicas devem ser, de alguma forma, adotadas antes da finalização do código que se deseja proteger, como forma de prevenir o plágio, ou caso ele ocorra, o perito tenha condições de afirmar se houve ou não infração de propriedade intelectual.

5.4. Sobre as Comparações de Desempenho das Ferramentas

Como havíamos comentado, os trabalhos de [Koschke 2007] e [Chen et al. 2004], por exemplo, dentre outros, incluem testes comparativos entre algumas das ferramentas citadas. O primeiro dos dois, coloca lado a lado ferramentas de tipos diferentes, com o intuito de comparar tecnologias distintas.

Não reproduzimos os resultados aqui e orientamos o leitor para que consulte os artigos originais, pois há muitas variáveis que comprometem os resultados, que poderiam até ser outros, caso houvesse mudanças nas aplicações dos testes. Os critérios de comparação, a massa de dados para os testes e até mesmo a definição sobre o que é ou não similar influenciam bastante os resultados.

5.5. Comparações Mais Gerais

Embora tenhamos que ser criteriosos com relação à comparação de desempenho das ferramentas, conforme dito na Seção 5.4, alguns comentários gerais podem ser colocados.

A grande maioria das ferramentas levantadas são resultado de experimentos acadêmicos. Uma parcela significativa tinha como objetivo geral ajudar professores na detecção de plágio entre trabalhos de programação de seus alunos (como Clan, Moss, SID, Yap, entre outros).

O Moss é um exemplo que virou produto comercial, chamado MossPlus. Tal qual o CodeSuite, é usado em tribunais.

Jplag e SID possuem serviço gratuito on-line via Internet e pode ser usado por qualquer pessoa; dentre esses dois, ao menos o Jplag também tem sido usado para o propósito de auxiliar na resolução de casos de litígio envolvendo autoria de código de software.

Note que essas quatro ferramentas (Moss, CodeSuite, Jplag e SID) implementam técnicas de detecção de similaridade sintática, ou seja, partem do código-fonte ou executável para análises.

O Moss é o que trata maior número de linguagens de programação, incluindo Assembly. O CodeSuite também trabalha com arquivos binários (código executável). O SID e Jplag reconhecem apenas Java, C, C++ e C#.

Os autores do SID relatam que a técnica deles gera menos resultados falso-positivos. O CodeMatch, no contra-ponto, parece gerar muita informação ruidosa, isto é, indica semelhanças pouco relevantes, mas que, em alguns casos, podem ser determinantes para indicar cópia ou derivação não autorizada (como comentários comprometedores esquecidos na cópia).

De uma forma geral, as técnicas que comparam similaridade sintática devem ser mais velozes que as semânticas. A técnica semântica de comparação pelo grafo de dependências do programa, por exemplo, possivelmente é uma das mais lentas, dada a natureza do problema de decidir isomorfismos de grafos ser NP-completo. Mesmo com

algoritmos de aproximação, a técnica deve ser empregada apenas sobre pequena quantidade de arquivos.

Todas as ferramentas citadas nesta subseção aceitam como entrada uma certa quantidade de arquivos e os compara aos pares. Como saída, é dado um índice de similaridade entre cada par de programa comparado (ou são apontados os pares mais relevantes). Isso ajuda bastante o perito, para passar um filtro sobre um grande sistema (composto de muitos arquivos) e chamar a atenção para alguns pares de arquivos específicos. Em [Roy and Cordy 2007], há referência de que o Jplag e o Moss (que fazem esse pareamento) apresentam complexidade $O(n^3)$, o que pode ser pouco praticável para a comparação de sistemas imensos. Não encontramos trabalhos relatando resultado semelhante, envolvendo o CodeSuite e o SID.

6. Conclusões

Ao analisar as ferramentas disponíveis para a detecção de plágio, baseadas em critérios estatísticos ou semânticos, podemos concluir que todos os métodos têm por objetivo vencer a luta contra os plágios. Muitas são as ferramentas criadas para a detecção de plágio, tendo cada uma suas métricas e atributos relacionados à sua aplicação.

Verificamos também, que o universo de conceitos sobre o tema similaridade é bastante vasto, e a grande dificuldade está em definir tipo de similaridade. Assim cada tipo de ferramenta diferencia-se pelo método utilizado para a comparação dos itens, e qual semelhança pretende identificar.

Nesta questão técnica há um longo caminho a ser percorrido pela ciência da computação para a melhoria da análise dos tipos e métodos de similaridade. Independente disto, devemos considerar que a análise humana é essencial neste universo, por ser capaz de utilizar métodos estatísticos ou semânticos. [Lukashenko et al. 2007] afirmam "... pelo menos por ora, nada pode substituir completamente o olhar atento dos seres humanos".

Aí está o grande desafio do perito, juntar as ferramentas técnicas e a habilidade humana, para gerar provas claras e argumentações precisas para instrumentar os julgadores na avaliação de um processo de litígio, atendendo a característica da legislação de cada país.

Antes de iniciar um trabalho relacionado a litígio de propriedade intelectual, envolvendo código de software, o perito deve fazer análises adequadas (de abstração e filtragem) para decidir *o que* será comparado. Em seguida, deve-se classificar o objeto de comparação quanto a seu tipo e sub-tipo (semântico ou sintático, e respectivas subclassificações). Somente então, é possível escolher técnicas e ferramentas adequadas. Em alguns casos, mais de uma ferramenta pode ser usada, mas a análise dos resultados deve ser feita sempre em função de:

- o que a ferramenta compara ou avalia;
- como é feita a comparação;
- como as configurações da ferramenta influenciam os resultados;
- taxa de detecção e precisão dessa detecção;
- índice de falsos-positivos, caso seja oferecido.

Por fim, vale observar que as ferramentas apenas auxiliam o trabalho do perito. Este deve necessariamente fazer também uma avaliação manual sobre os resultados, comparando diretamente os trechos do sistema delatados pelas ferramentas. E, é claro, o perito deve ter conhecimento de sistemas de software nos seus vários níveis de implementação. Isto é, não basta que ele seja especialista no nível de código-fonte ou objeto, mas deve também dominar os níveis arquiteturais e de modelagem do sistema.

Referências

Baker, B. S. and Manber, U. (1998). Deducing similarities in java sources from bytecodes. In *USENIX Technical Conference*, pages 179–190.

Chen, X., Francia, B., Li, M., Mckinnon, B., and Seker, A. (2004). Shared information and program plagiarism detection. *IEEE Trans. Inform. Th*, 50:1545–1551.

Culwin, F. and Lancaster, T. (2000). A review of electronic services for plagiarism detection in student submissions. In *LTSN-ICS 1st Annual Conference*.

Ducasse, S., Rieger, M., and Demeyer, S. (1999). A language independent approach for detecting duplicated code. In *Proceedings ICSM99 (International Conference on Software Maintenance*, pages 109–118. IEEE.

Hollaar, L. A. (2002). Legal protection of digital information. Chapter 2: Copyright of Computer Programs. Disponível em: http://digital-law-online.info/lpdi1.0/treatise22.html. Acesso em: outubro de 2008.

Kamiya, T., Kusumoto, S., and Inoue, K. (2002). Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28(7):654–670.

Koschke, R. (2007). Survey of research on software clones. In *Duplication, Redundancy, and Similarity in Software'06*, volume 06301 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

Krinke, J. (2001). Identifying similar code with program dependence graphs. In *WCRE '01: Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, page 301, Washington, DC, USA. IEEE Computer Society.

Lee, W. and Hwang, C. (2007). A forensic computing system using a digital right management technique. *Fuzzy Systems and Knowledge Discovery, Fourth International Conference on*, 3:258–262.

Lukashenko, R., Graudina, V., and Grundspenkis, J. (2007). Computer-based plagiarism detection methods and tools: an overview. In *CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies*, pages 1–6, New York, NY, USA. ACM.

Merlo, E. (2007). Detection of plagiarism in university projects using metrics-based spectral similarity. 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/2007/986. Acesso em: dezembro de 2008.

Richards, J. (2002). Copyright protection for computer software in the united states. 3.2 The ”Look and Feel”Test. Disponível em: http://www.ladas.com/Patents/Computer/SoftwareAndCopyright/Softwa05.html. Acesso em: outubro de 2008.

Roy, C. K. and Cordy, J. R. (2007). A survey on software clone detection research. Disponível em: http://www.cs.queensu.ca/TechReports/Reports/2007-541.pdf. Acesso em: dezembro de 2008.

Schleimer, S., Wilkerson, D. S., and Aiken, A. (2003). Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, New York, NY, USA. ACM.

Verco, K. L. and Wise, M. J. (1996). Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. In *Proc. of 1st Australian Conference on Computer Science Education*, pages 86–95. ACM.

Walenstein, A., El-Ramly, M., Cordy, J. R., Evans, W. S., Mahdavi, K., Pizka, M., Ramalingam, G., and von Gudenberg, J. W. (2007). 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: outubro de 2008.

Wong, J. L., Kirovski, D., and Potkonjak, M. (2001). Computational forensic techniques for intellectual property protection. In *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*, pages 66–80, London, UK. Springer-Verlag.

Zeidman, B. (2004). Detecting source-code plagiarism – tools and algorithms for finding plagiarism in source code. *Dr. Dobb's Journal*, july. Disponível em: http://www.ddj.com/architect/184405734. Acesso em: dezembro de 2008.