Ensaio Iurídico. Patos de Minas: UNIPAM, (1):143-158, 2010 | ISSN 2179-1945

Doutrina ideia-expressão e o *software*: o estudo comparado da *merger doctrine, misappropriation doctrine* e teoria da escolha nas cortes norte-americanas, australianas e francesas

Idea-expression doctrine and the software: a comparative study of the merger doctrine, misappropriation doctrine and the theory of choices in North-American, Australian and French courts

Leonardo Machado Pontes
Faculdade de Direito Milton Campos, bolsista pesquisador pela FAPEMIG (Fundação de Amparo à Pesquisa do Estado Minas Gerais). e-mail: lm.pontes@hotmail.com

Resumo: O artigo tem por objeto o estudo de várias teorias aplicadas pelos tribunais australianos, norte-americanos e franceses para decidir os casos de violação aos *softwares*. O art. 6, inc. III, da Lei 9609/98 (Lei de *Softwares*) incorporou noções fundamentais de algumas dessas teorias. O artigo é um guia prático para a resolução de conflitos entre *softwares*.
Palavras-chave: Doutrina ideia/expressão. *Merger doctrine*. *Misappropriation doctrine*. Teoria francesa da escolha. *Software*.

Abstract: This article intends to study some theories applied by the Australian, North American and French courts to decide the cases of software infringement. Art. 6, section III, of Law 9.609/98 (Law of Softwares) incorporated basic fundamental knowledge of some of these theories. The article is a practical guide for the conflict resolution between softwares.
Keywords: Dichotomy idea/expression. Merger doctrine. Misappropriation doctrine. French theory of choice. Software.

1. Introdução

Segundo a lei norte-americana (LAURIE, 1989, p. 232), para que autor de uma ação de *copyright* (direito de autor) obtenha êxito na alegação de *infringement*[1] ou viola-

[1] "Nos Estados Unidos existem dois tipos de violação ao *copyright*: o *primary infringement* e o *secondary infringement*. O *primary infringement* trata de uma violação direta ao direito de reprodução e pode envolver também o direito de comunicação ao público (...); violam-se os direitos inerentes ao titular do *copyright* que apenas ele poderia exercer. Intenção inocente, boa-fé, cópia inconsciente, ou ignorância da existência do *copyright* não são bastantes para motivar a defesa. O *secondary infringement* diz respeito à importação e/ou distribuição comercial de cópias infringidas" (PONTES, 2009, p. 7).

ção de seu programa de computador, deve provar que o réu se apropriou (incorporou em seu próprio programa de computador) mais do que a quantidade *de minimis* da expressão protegida do autor. Em outras palavras, deve provar que existe similaridade substancial no nível da expressão e não no nível da ideia entre os *softwares*.

Isso porque, segundo a doutrina ideia/expressão, o direito de autor não protege ideias, mas apenas as expressões dessas ideias que tenham o mínimo de originalidade e sobre as quais o autor tenha exercido sua autonomia. Como consectário lógico, essa premissa nos revela que todas as obras, inclusive *softwares*, têm elementos protegidos (expressão) e elementos não-protegidos (ideias).

O programa de computador é sujeito à proteção do direito de autor (HILTON, 1991, p. 271), quando seu código-objeto (*object code*),[2] código-fonte (*source code*)[3] ou código executável (*executable code*)[4] é fixado, não importa se em papel, em disco ou em um circuito integrado de *chip*. A proteção, segundo Hilton (1991, p. 272), estende-se a todos as outras formas do programa que tenham os mesmos elementos expressivos, como traduções em outras linguagens de programação ou em versões feitas para rodar em outros computadores.

O *software* é protegido pelo *copyright* e em alguns países por patentes, constituindo um sistema misto de proteção; outras vezes, por um sistema unitário baseado apenas no direito de autor. O *software*, para se adequar ao regime do *copyright* ou do direito de autor, deve respeitar o princípio da dicotomia ideia/expressão. Todavia, a análise da dicotomia ideia/expressão em programas de computador, como veremos, revela-se o mais das vezes problemática. O que seria ideia e o que seria expressão em um *software*?

Essa dificuldade surge porque os programas de computador têm uma natureza híbrida (SAMUELS, 1989, p. 368); implementam, ao mesmo tempo, palavras e símbolos para controlar um processo. A dificuldade, como vermos, é o fato de existirem elementos literários nos *softwares* que apenas transmitem informações ou que têm uma funcionalidade intrínseca (HILTON, 1991, p. 279).

Em face de sua natureza híbrida, duas correntes jurídicas surgiram para justificar uma visão mais protecionista e uma visão menos protecionista ou liberal, ambas com argumentos no sistema jurídico, que podem ser extraídas da seguinte maneira: a)

[2] Passo intermediário na produção do código executável para o código-fonte.
[3] Escrito em linguagem convencional humana e em símbolos.
[4] Escrito em linguagem binária lida por máquinas.

o *copyright* ou direito de autor protege mais do que o código literal (leia-se código objeto, código executável e código-fonte) do programa; b) o *copyright* ou direito de autor protege apenas o código literal (leia-se código objeto, código executável e código-fonte) do programa.

A vertente mais protecionista busca o amparo na legislação para fomentar sociedades empresárias e programadores, ao proteger juridicamente outros elementos que excedem o código literal de um programa, uma vez que o investimento de capital se tornaria vulnerável se estes outros elementos não fossem protegidos. Essa vertente, como veremos em nosso estudo, utiliza argumentos da *idea/expression doctrine*, *merger doctrine*, *misappropriation doctrine* e o princípio mais genérico do *mixing labor* de John Locke.

A vertente menos protecionista ou liberal busca o amparo na legislação para permitir que os elementos funcionais dos *softwares* sejam objeto de cópia, uma vez que extrapolam os limites do código literal, de maneira a permitir o avanço e o estudo tecnológico pelo intercâmbio de elementos não protegidos. Essa vertente se utiliza dos argumentos da *idea/expression doctrine* e *merger doctrine*.

Veremos que toda a polêmica que surge no *software* é representada por essas duas vertentes, na medida em que os casos que envolvem os *softwares* surgem quando um programador cria e vende seu programa, cujo desempenho, aparência e atuação se assemelham a outro *software* existente (BIXBY, 1981, p. 31).

2. Breve histórico

A opção pela proteção do programa de computador pelo *copyright* foi uma escolha oportuna nos anos setenta, já que o *software*, como trabalho literário em sua forma de código-fonte, necessitava de proteção contra a pirataria. Uma vez que o *copyright* protege o trabalho contra sua cópia (KARJALA, 1994, p. 975), tratou-se de uma escolha razoável para a proteção do programa de computador ao mesmo tempo em que Fornécia proteção transacional automática sob a égide das convenções internacionais.

Todavia, nos anos noventa, iniciou-se um debate mais acirrado para determinar se tudo aquilo que extrapolasse o código literal na tecnologia computacional deveria ser objeto ou não de proteção pelo *copyright*. Em 1992, nos Estados Unidos, começaram a surgir os primeiros casos que discutiram essas questões, como *Computer Ass,. Int'l, Inc. v. Altai, Inc.*; *Apple Computer, Inc. v. Microsoft Corp.*; e *Lotus Dev. Corp. v. Borland Int'l, Inc* (KARJALA, 1994, p. 975).

3. Funcionalidade versus literalidade

O que torna a discussão acirrada em torno dos *softwares* são seus elementos funcionais. "Um trabalho é funcional quando ele executa uma tarefa utilitária que não seja para informar, entreter ou portar uma aparência para seres humanos" (KARJALA, 1994, p. 977. Tradução nossa). Enquanto, de um lado, a patente tem por função proteger trabalhos criativos funcionais, o *copyright* ou direito de autor tem por função proteger trabalhos criativos não-funcionais, ainda que o *copyright* tenha ao longo dos anos expandido seu campo jurídico para áreas que antes, teoricamente, pertenciam exclusivamente à propriedade industrial, como o *design* e a patente.

Os programas de computador, os protocolos de comunicação, o *hardware* para o *software*, as interfaces de *software* para outro *software*, e as *interfaces* dos usuários são, segundo Karjala (1994, p. 983), intrinsecamente funcionais; permitem que as máquinas realizem determinadas funções ao abrir as portas e as janelas que os programas usam para se interagir. Portanto, os programas de computador têm propósitos utilitários que não são para entreter ou informar seres humanos. Se os elementos dos programas de computador são mais funcionais do que não-funcionais, por que não protegê-los por meio do sistema de patentes?

Na realidade, a maioria dos programas de computador é desenvolvida com base em princípios de programação já conhecidos para a resolução de problemas específicos, que não atingiriam o requisito de novidade absoluta para lograr a proteção por meio da patente (KARJALA, 1994, p. 984). Uma vez que, por meio do código-fonte, a cópia desses programas é relativamente fácil de obter, houve a necessidade de se pensar novas formas de proteção para os investidores e programadores. Dessa forma, para proteger contra as cópias dos códigos foi eleito o sistema de *copyright*, já que esses códigos não atingiriam os requisitos de proteção no sistema de patentes.

A proteção ao código impede, pois, a cópia para a venda ou uso simultâneo, o que vai de encontro aos padrões de equilíbrio. O código literal deve ser protegido (código-fonte, código-objeto, código executável). A grande questão é, então, definir o quê, dentro do espectro do que não seja o código literal, também seria protegido pelo *copyright*.

4. As cortes australianas

As cortes australianas desenvolveram certos parâmetros e técnicas para tentar equacionar melhor essa questão.

Assim, sempre quando uma ação for proposta, o autor deve provar: a) que sua obra como um todo é original e, portanto, protegida; e b) que o réu se locupletou de parcela substancial de sua obra. Para provar o segundo requisito, vale dizer, o da cópia de parcela substantiva da obra, as Cortes exigem: b') a existência de um grau objetivo de similaridade entre a obra protegida e a alegação do ilícito e; b'') a existência de uma necessária conexão causal (ROTHNIE, 1998, p. 60). Se os requisitos forem suficientemente demonstrados, haverá uma presunção de contrafação.

Acreditamos que esses requisitos poderiam ser adotados por nossos tribunais, em uma melhor tentativa de elucidar a controvérsia, com a ressalva, todavia, de que não cabe ao autor provar que *criou a obra*. Basta ter seu nome indicado na obra, na qualidade de autor, o que cria uma presunção *iuris tantum* de autoria, que deve ser desconstituída pela réu, nos termos da Convenção de Berna e da própria Lei de regên-cia.[5]

Todavia, a questão mais complicada, na visão de Rothnie (1998, p. 67), diz respeito à dicotomia ideia/expressão no que toca a programas de computador. A criação ou armazenamento de materiais em formatos digitais ou eletromagnéticos que envolvem a conversão de obras em sequências de 0 a 1, armazenadas em alguma forma de memória computacional mediante impulsos elétricos.

Nos programas de computador existem quatro estágios em sua concepção (ROTHNIE, 1998, p. 67):

Primeiro estágio: no plano mais elevado, existe a ideia ou função do programa.
Segundo estágio: em um plano mais rebaixado, existe uma lista de características de um programa de computador.
Terceiro estágio: em um nível menos abstrato, as características de um programa de computador são descritas sob uma especificação funcional e elaboradas com maiores especificações técnicas. O terceiro estágio é considerado como trabalho literário e pode envolver labor artístico.
Quarto estágio: alguns programas complexos podem consistir em número de módulos ou em programas em si mesmos. A relação dos módulos entre si, e as funções de cada programa individual, podem ter sido trabalhadas graficamente em pseudocódigos.

[5] O art. 15/1, da Convenção de Berna (Dec.75.699/75), prescreve o princípio fundamental de que o criador da obra é aquele em nome do qual a obra é divulgada, cabendo a terceiros provar o contrário. Pode-se extrair dessa regra, igualmente, o *ius conventiones* – o direito convencional –, que estabelece critérios mínimos de proteção ao autor do Estado signatário da Convenção. Isso quer dizer que a legislação interna do Estado deve respeitar esse critério mínimo. A legislação infraconstitucional praticamente copiou o art. 15/1, da Convenção nos dispositivos dos arts. 12 e 13, da Lei 9.610/98. Nesse sentido, se no software conste o nome do autor como o autor do programa há presunção *iuris tantum* de autoria.

A dicotomia ideia/expressão pode aparecer em qualquer desses estágios, inclusive entre estágios.

As cortes australianas têm julgado no sentido de que, em relação ao primeiro estágio, o simples fato de um ou mais programas terem a mesma função não é suficiente para caracterizar a contrafação (ROTHNIE, 1998, p. 68).

Em relação ao segundo estágio, as decisões australianas não são uníssonas. A cópia de inúmeros elementos das características de um programa de computador pode caracterizar contrafação, desde que envolva cópia de considerável detalhamento ou quantidade de elementos, definidos pela reprodução sequencial de pseudocódigos do código-objeto ou código-fonte, por sua vez, gerados em outra sequência de documento, que pode ser protegido como compilação. Essa tem sido a interpretação majoritária, dada também pelos tribunais americanos, aplicando uma definição mais larga de proteção, que se estende também à lógica, estrutura, *design* e a desempenho do programa de computador (SAMUELS, 1998, p. 368).

Em relação ao terceiro estágio, a apropriação substancial do código-fonte viola o direito de autor que subsiste naquela fonte, salvo se for demonstrado pelo réu que a similitude decorre não do direito de autor que subsiste no código fonte, mas de outros fatores, como: a) derivação de *bits* similares de uma terceira parte; b) derivação do domínio público; c) similaridades advindas apenas do próprio estilo do programador; d) similaridades que derivam de uma função necessária, vale dizer, quem quer que fizesse aquele trabalhado necessariamente faria daquela forma (ROTHNIE, 1998, p. 71). Veremos, mais à frente, que o item "d", acima listado, é conhecido, nos Estados Unidos, como *merger doctrine* ou *plurality of expressions test* e tem grande relevância prática na análise de programas de computador.

5. As cortes americanas

Nos Estados Unidos, vários julgados referentes à dicotomia ideia/expressão, comcernentes à análise de programas de computador, geram polêmica, dado a complexidade técnica da matéria.

No caso *Apple v. Microsoft*, a comunidade jurídica foi obrigada a se deparar com questões complexas até mesmo para os mais estudiosos da matéria, como: a) quando um programa de computador é substancialmente similar a outro?; b) quando um programa de computador constitui uma derivação de outro?; c) o *copyright* protege os grá-

ficos gerados nas telas do computador, pelo programa de computador?; d) os gráficos gerados na tela de um computador pelo *software* podem ser independentemente protegidos enquanto compilação de dados ou enquanto uma obra audiovisual? (BIXBY, 1981, p. 32).

Algumas dessas questões já vinham se travando na justiça, em outros julgados.

O precedente mais proeminente e mais polêmico é o caso *Whelan, Inc. v. Jaslow Dental Laboratory*. Neste caso, foi decido que tudo aquilo que não seja o propósito ou a função do programa, visto externamente, é protegido porque expressão (LAURIE, 1989, p. 233). Em outras palavras, foi dada proteção pelo *copyright* a estrutura, sequência e organização do programa de computador, mesmo que não se tratasse de código literal.

O mesmo raciocínio foi aplicado posteriormente ao caso *Broderbund Sofware Inc. v. Unison World*, dando proteção do *copyright* as telas geradas pelo *menu* do *software*, que tinham sido substancialmente copiadas no *software* da ré, com algumas pequenas e insignificantes alterações (BIXBY, 1981, p. 41).

No caso que se seguiu a esse, *Digital Communications Associates, Inc. v. Softklone Distributing Corp*, a Corte teve outra postura reflexiva, ao entender que o *status* da tela de um programa de computador era protegido como expressão literária e não obra audiovisual, além de poder ser protegido como compilação de dados, já que arranjado de uma forma peculiar e original.

Como nos faz saber Laurie (1989, p. 233), embora toda a fundamentação teórica seja baseada no *copyright*, as Cortes julgam, na maioria dos casos, com base na doutrina da *misappropriation* (apropriação ilegal da propriedade alheia com a finalidade de beneficiar a si mesmo ou a outrem), traçando argumentos racionais de concorrência desleal.

Em uma visão mais geral, o que a jurisprudência demonstra é que se o réu buscou se apropriar do *mixing labor* do autor (do trabalho que o mesmo cunhou na obra), sem empregar qualquer esforço criativo que demonstrasse que estivesse comprometido com as práticas honestas do tráfico comercial, mas ao contrário, queria apenas se beneficiar do lucro rápido e fácil advindo da cópia de elementos substanciais de outro *software*, a tendência dos tribunais é condenar dentro do escopo maior da *misappropriation*, ainda que apresentem argumentos da doutrina *ideia/expressão*.

O bom senso dos julgadores está diretamente relacionado às condições causais que permitiram que a ré criasse o seu *software*, isto é, se suspeitas ou não. Nesse sentido, em 1987, o *Tribunal de grande Instance,* de Paris, julgou que o *software* do réu era uma violação ao *software* do autor porque "ele não detinha a capacidade técnica e fi-

nanceira, dentro de um ano, para produzir um produto original de *software* (SHUSTER, 1992, p. 49. Tradução nossa).

No mesmo sentido, no Reino Unido, um *software* é original se não for copiado de uma fonte existente e se no *software* há graus substanciais de habilidade, julgamento e trabalho (SIMON, 2006, p. 698).

Portanto, a solução mais lógica e fácil é tentar resolver a questão primeiramente no âmbito da concorrência desleal, do segredo industrial, da confidencialidade e de princípios traçados para criar uma relação fiduciária entre as partes, como a boa-fé e a confiança, somente para, se ainda for infrutífera, restar a tentativa de buscar a aproximação da dicotomia ideia/expressão, que é definitivamente mais complicada e elaborada e reclama por prova pericial.

6. Merger doctrine *e teoria da escolha francesa*

Como visto há pouco, a metodologia de análise dos níveis de abstração do programa de computador é a melhor forma de se verificar se houve ou não violação ao direito de autor, uma vez que toda concepção do programa de computador passa pela estrutura chamada de *top down* – isto é, os programas de computador são criados seguindo os níveis descritos (KARJALA, 1994, p. 977).

A grande dificuldade que subjaz é traçar a linha divisória que separa nitidamente a *ideia* da *expressão* no programa de computador. Nesse sentido, os doutrinadores norte-americanos e alguns juízes têm se reportado à *merger doctrine*, na tentativa de elucidar a questão.

A *merger doctrine* é uma doutrina que nega proteção ao *copyright*, mesmo que seja caracterizado como expressão protegida, nos casos em que uma ideia somente pode ser expressa de uma maneira ou por apenas algumas maneiras. Se uma ideia só pode ser expressa de uma forma ou por algumas formas, diz-se que a ideia se fundiu a expressão, que houve um *merger*. É, assim, uma diminuição à proteção, uma vez que se uma forma específica de programação somente pode ser descrita de algumas maneiras, portanto, dar proteção a estas algumas maneiras prejudicaria todos os outros *softwares* (KARJALA, 1998, p. 521).

De igual modo, a *Cour de Cassation*, no caso *Société Babolat Maillot Witt v. Pachot*, criou o teste conhecido como *teoria da escolha*, para determinar a originalidade ou não em um *software*. A *teoria da escolha* nada mais é do que a *merger doctrine* norte-

americana, com o diferencial de que a originalidade é baseada na escolha que detém o programador de expressar sua personalidade. Assim, para os franceses, se houver diferentes maneiras pelas quais um programador pode expressar sua criatividade naquele caso em específico, a forma de expressão escolhida carrega sua expressão original. Em outras palavras, para ser original uma expressão de *software* ela deve transcender os meros obstáculos da lógica da programação (SHUSTER, 1992, p. 57).

Nosso legislador teve essa precaução, ao estabelecer no art. 6.º, inc. III, da Lei 9.609/98, que não constitui violação ao programa de computador *a ocorrência de semelhança de programa a outro, preexistente, quando a)* ***se der por força de características funcionais de sua aplicação****; b) da observância de preceitos normativos e técnicos, ou c)* ***de limitação de forma alternativa para a sua expressão****.*

As hipóteses (**a**) e (**c**) do art. 6.º incorporaram a ideia central da *merger doctrine/teoria da escolha*, enquanto a hipótese (**b**) trouxe a preocupação da doutrina *ideia/ expressão* do art. 8º, inc. I, da Lei 9.610/98 (Lei do Direito de Autor): *não são objeto de proteção como direitos autorais (...) ideias, procedimentos, sistemas, conceitos matemáticos...*

O que nos interessa em relação a isso é o argumento de que se existem várias possibilidades de se expressar algo por meio da tecnologia computacional; então, qualquer uma dessas formas específicas se torna expressões protegidas. Em outras palavras, o que é protegido no *software* é a ausência de um *merger* (KARJALA, 1988, p. 521). Todavia, basear toda a análise apenas na *merger doctrine* para separar a ideia da expressão se mostra inviável, por carecer de outros elementos. Se a análise fosse conduzida apenas pela *merger*, várias iniquidades poderiam surgir.

A abordagem lógica que sugere Karjala (1998, p. 522) é proteger por meio do *copyright* o código literal do programa de computador ou traduções mecânicas ou eletrônicas sem qualquer criatividade ou esforço. Isso parece óbvio. A violação, nesses termos, deve ser prontamente objeto de procedência para uma ação condenatória. Estabelece o *Copyright Act,* de 1976, emendado em 1980, que a cópia do código-fonte e do código-objeto de um programa constitui violação ao *copyright.*

Nesse mesmo sentido, os tribunais franceses, por analogia, descreveram programas de computador como trabalhos literários e chegaram à conclusão de que o *software* original se constitui como *oeuvre de l'esprit* (SHUSTER, 1992, p. 45).

Pela análise, igualmente, do art. 6º de nossa lei, acima retratado, a violação do código-fonte, código-objeto e código-executável é contrária à lei.

A controvérsia, todavia, é saber se a estrutura de um *software*, sua sequência,

organização (SSO) e os elementos de interface deveriam ou não ser protegidos pelo *copyright*, justamente a polêmica gerada pelo caso *Whelan, Inc. v. Jaslow Dental Laboratory*. Sequência, organização e estrutura são conhecidas pela expressão popularizada pela mídia e pelos tribunais de *total concept and feel* ou simplesmente de *look and feel*. Isso significa o teste feito pelos juízes em seu nível mais basilar, algo do tipo "olhe e sinta cada aspecto particular do programa" (BIXBY, 1981, p. 34).

É cediço que a forma em que se arranjam os módulos internos de um *software* pode aumentar sua eficiência em um mercado competitivo, e falta de proteção poderia prejudicar o programador ou a sociedade empreendedora.

Ao revés, o excesso de proteção poderia inibir o avanço tecnológico, já que os *softwares* se retroalimentam da tecnologia pretérita com frequência. Nesse caso, os módulos internos deveriam ser interpretados apenas como elementos funcionais, engenharia, método, e não como criações, mas apenas infra-estrutura (KARJALA, 1988, p. 525). Aqui é definitivamente onde a *merger doctrine* exerce o papel fundamental, segundo a visão da maioria dos julgados e de alguns doutrinadores americanos:

a) – Se for demonstrado que a forma em que foi arranjado o módulo interno só poderia ser arranjada daquela maneira ou mediante limitadas formas, deve ser julgado que não houve violação, pois se trata de elemento funcional.

b) – Se for demonstrado que o módulo poderia ser arranjado em diferentes formas que não diminuíssem a eficiência do programa, e que mesmo assim o programa foi arquitetado utilizando quantidades substantivas de outro módulo, haverá violação, pois o elemento funcional já não mais prepondera e foi copiada a "expressão".

7. Os três testes do juiz Leonard Hand

Além da *merger doctrine*, mediante a contribuição do juiz Leonard Hand, os tribunais americanos desenvolveram três diretrizes para guiar a condução da doutrina ideia/expressão, com aplicação, inclusive, para os *softwares*: a) o teste de abstrações (*abstractions test*); b) o teste da pluralidade de expressões (*plurality of expressions test*) e; c) distinções funcionais/expressivas e essenciais/não-essenciais (*funcional/expressive and essential/non-essential distinctions*) (SHUSTER, 1992, p. 922)

Esses três testes foram aplicados no caso *Lotus Development Corp. v. Paperback Software International*, como passaremos a estudar.

De acordo com o *teste das abstrações*, a ideia e expressão podem ser separadas ao

se descrever o contínuo da expressão. No final do contínuo, as ideias são descritas de uma maneira geral, e, no seu início, de uma maneira expressa. O julgador deve criar, assim, uma escala de abstrações, medida caso a caso (SHUSTER, 1992, p. 923).

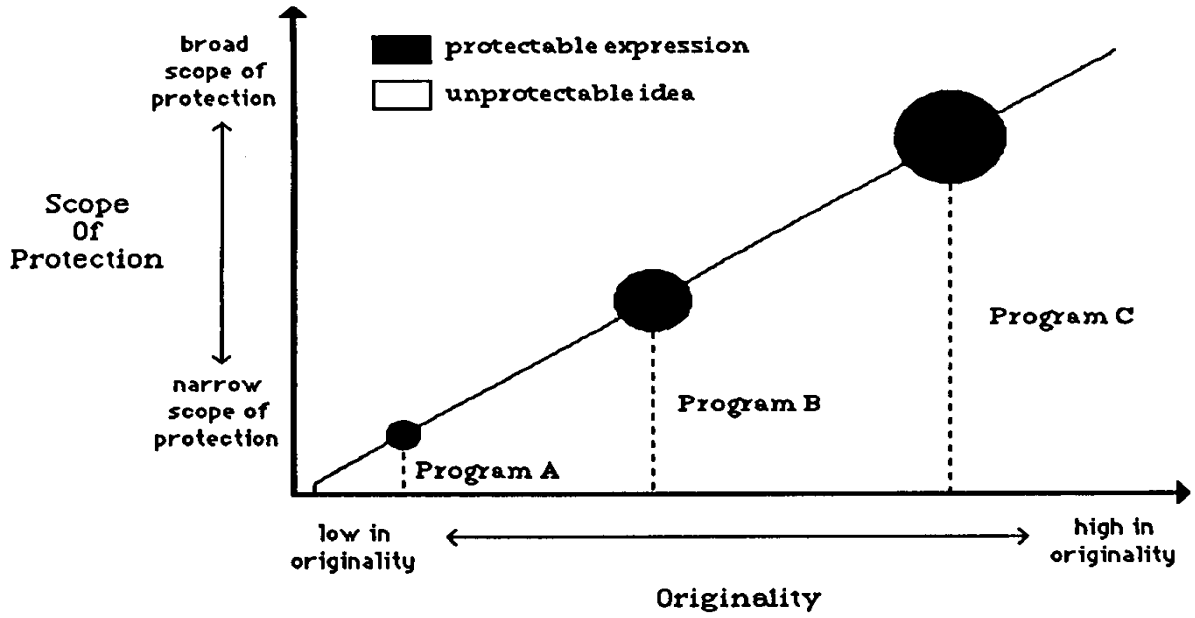
Em *Lotus Corp*., ao se aplicar o referido teste, os julgadores chegaram à conclusão de que programas como *VisiCalcs*, *1-2-3, Multiplan*, *SuperCalc4 e Excel*, embo-ra diferentes em sua estrutura e aparência, eram maneiras diversificadas de se expres-sar um *spreedsheet* eletrônico. Eles não sugeriram que as diferentes maneiras de se expressar o *spreedsheet* seriam protegidas pelo *copyright*, mas antes que outros elemen-tos, mais específicos, também poderiam não receber proteção, como o formato em L rotatório que era apresentado na tela do *spreedsheet* (SHUSTER, 1992, p. 923).

Para determinar se formato em L era ou não protegido pelo *copyright*, os julgadores se valeram do teste da *pluralidade de expressões*. Segundo esse teste, quanto maior for o espectro de maneiras em que determinada coisa pode ser expressa, maior será a probabilidade de essa coisa ser protegida pelo *copyrigh*t.

Os julgadores, chegando à conclusão de que o formato em L giratório da tela poderia ser expresso em apenas algumas maneiras, aplicaram a *merger doctrine* e constataram ausência de violação. Aplicaram o mesmo teste para determinar se a cópia do comando (/) para abrir o *menu* do programa do autor seria ou não violação. Haven-do poucas formas de se expressar o referido comando, o tribunal também aplicou a *merger doctrine* (SHUSTER, 1992, p. 923).

Todavia, ao aplicarem o mesmo teste para determinar se a linha do comando do *menu* do programa, descrita sob os caracteres *Command: BCDEFGIMPRSTVW*, seria ou não violação, afastaram a *merger doctrine* porque essa mesma linha poderia ser descrita de infinitas formas. Ao final, embora ambos os *softwares* tivessem elementos de *merger*, o tribunal julgou que a estrutura *Lotus 1-2-3* era em seu todo protegida, porque "original e sem qualquer obviedade" (SHUSTER, 1992, p. 924).

A aproximação intentada, segundo Shuster, reflete assim o que as Cortes chamam de *estilo de criação* do programador. Se o programador usar para expressar uma ideia uma expressão *original e sem obviedade*, e se esta ideia pode ser expressa de inúmeras outras formas, sua obra será protegida. Ao contrário, se uma ideia somente for passível de ser expressa limitadamente, haverá *merger* e, consecutivamente, falta de originalidade e proteção. Essa parece ser também a orientação da Corte de Cassação, como visto. O seguinte gráfico (HILTON, 1991, p. 284) explicita esse raciocínio:



Quanto maior a originalidade, maior a proteção e maior será o perímetro circular concedido à expressão. Nesse mesmo sentido, o aspecto de funcionalidade que seja imbricado ao aspecto literário do programa pode ser igualmente trazido para o escopo da *merger doctrine*. Em outras palavras, se a funcionalidade do programa neces-sita de determinada sequência ou módulo para funcionar apropriadamente, as Cortes não dão proteção a esses aspectos e não consideram violação (SHUSTER, 1992, p. 925). Essas posições, todavia, são tomadas caso a caso, analisando a zona de interconexão entre o 2.º e o 3.º estágios, havendo possibilidade de proteção. Esse é o teste conhecido como *distinções funcionais/expressivas e essenciais/não-essenciais*.

No caso *Arnstein v. Porter*, as Cortes desenvolveram, outrossim, mais um teste para determinar se havia similaridade substancial entre os *softwares* objeto de litígio. Esse teste é composto por um componente extrínseco e um componente intrínseco. O componente extrínseco é baseado na análise pericial, enquanto o componente intrínseco é baseado na visão leiga dos julgadores. No final, ao combinarem os componentes intrínsecos e extrínsecos, os julgadores são capazes de decidir com maior grau de certeza se há ou não similaridade substancial (BIXBY, 1981, p. 35).

8. A abordagem mais liberal

Até agora, vimos a interpretação dada aos *softwares* que extrapolam os limites do código literal. Vejamos agora a interpretação menos conservadora.

Na opinião de Karjala (1994, p. 998), a opção de se adotar a *meger doctrine* para análise do 2.º estágio é equivocada, porque independentemente do processo e sequência que possa existir para se obter determinado resultado, não importa quais outros sistemas ou métodos possam existir, sistemas e métodos não são protegidos pelo *copyright*. Pouco importa, para o *copyright*, a forma como os módulos são arranjados, se mais eficientes ou ineficientes, porque simplesmente são se enquadram como *expressão literária*.

Essa foi a mesma conclusão a que chegou o *Office of Technology Assessment*, em estudo publicado, ao dizer que a concepção mais larga adotada pelos tribunais, protegendo outros elementos que ultrapassam o código literal, é inadequada porque o *copyright* não protege sistemas (SAMUELS, 1989, p. 368).

Nesse sentido, houve certo avanço em alguns julgados, como *Gates Rubber Co.*, distanciando-se do precedente do caso *Whelan, Inc. v. Jaslow Dental Laboratory*, em que se decidiu que não são protegidos:

> ideias, processos, fatos, informação de domínio público, material que merge, incluindo os standards de hardware e especificações mecânicas, os standards do software e requerimentos de compatibilidade, os standards do design manufaturado de computadores, e as práticas da indústria de computadores (KARJALA, 1994, p. 987. Tradução nossa).

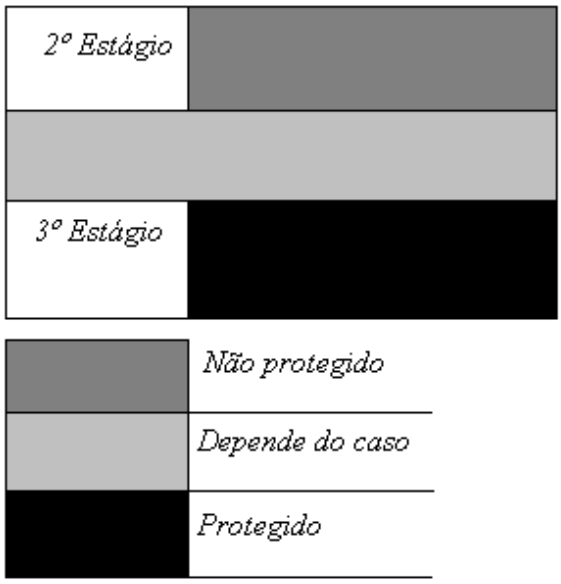
Também, no caso *Computer Assoc.*, foi decidido que não são protegidos:

> Elementos necessariamente incidentais à função de um programa, incluindo elementos dedicados a eficiência; elementos requeridos por fatores externos, incluindo elementos ditados por especificações mecânicas, compatibilidade de requerimentos; standards de design manufaturado do computador; largas práticas de programação aceitas e elementos de domínio público. (KARJALA, 1994, p. 987. Tradução nossa).

Ao se proteger apenas o código-fonte, o código-objeto e o código executável, os demais elementos funcionais, que maximizam a eficiência dos programas, desde que não sejam protegidos por patente, permitem o intercâmbio de estudo entre programadores, reduzindo os impactos de um retrocesso tecnológico. Assim, protege-se o código do programa contra cópias, transformações ou traduções banais e sem criatividade, direta ou indiretamente.

Aplicando-se, então, o teste da doutrina ideia/expressão, a linha tênue que se-

para a ideia da expressão nos programas de computador está justamente na interposição entre o 2.º estágio para o 3.º estágio:



Essa proteção é conhecida como *thin protection* – proteção fina – e consiste na proteção concedida apenas ao terceiro estágio e a alguns aspectos da zona intermediária entre ambos.

Resta saber com mais detalhes o que compõe o 2.º estágio, porque já sabemos que o 3.º estágio é composto pelo código literal. Nesse sentido, para a corrente menos protecionista, resta saber se as chamadas *interfaces*, que são janelas e portas pelas quais os usuários e o hardware e software fazem uso do programa, são protegidas pelo *copyright*, pois vimos que as *interfaces*, para a corrente protecionista, podem receber proteção.

Ao se aplicar o teste da dicotomia ideia/expressão, para a corrente menos protecionista, as *interfaces* do programa em si não seriam protegidas. As chamadas *nonuser interfaces* também não atendem aos critérios de proteção, vistas como uma combinação de símbolos ou códigos, nem mesmo atendem à proteção dispensada à compilação de dados (KARJALA, 1994, p. 980). Todavia, algumas *interfaces* com características apresentadas na tela podem ser protegidas como aspectos de trabalho audiovisual. Resta a dúvida, ainda não equacionada, se as *interfaces* de usuários, que têm graus de funcionalidade, deveriam ou não ser protegidas pelo *copyright*.

Segundo nossa visão, para decidir se as chamadas *user interfaces* deveriam ser protegidas, é aqui, com mais rigor, que deveria ser aplicada as regras já vistas, traçadas

pelos tribunais americanos, isto é, para se verificar a zona de interconexão entre o 2.º e o 3.º estágios.

9. Conclusão

Entre as duas correntes estudadas, vimos que ambas concordam que o 1º estágio do *software* não é protegido e que o 3.º estágio é protegido. A divergência travada se refere à proteção dispensada ao 2.º estágio.

Na Austrália, nos Estados Unidos e na França, se for demonstrado que a outra parte se utilizou substantivamente da estrutura do *software*, das *interfaces* de usuários, e havendo ausência de *merger*, a tendência das Cortes (embora haja exceções) é julgar no sentido de violação ao *copyright*, porém, fornecendo também argumentos da doutrina da *misappropriation*. Assim, segundo a visão que atualmente prevalece, o segundo estágio tem elementos que são protegidos pelo *copyright*, além das interconexões entre o 2.º e 3.º estágios.

10. Bibliografia

BIXBY, Michael B. Synthesis and originality in computer screen displays and user interfaces: the "look and feel cases", *Willamette Law Review*, Oregon, v. 27, pp. 31-50, 1981.

HILTON, Willian E. Quantifying originality: a logical analysis for determining substantial similarity in computer software copyright infringement actions. *IDEA – The Journal of Law and Technology*, v. 31, pp. 269-296, 1991.

KARJALA, Dennis S. Copyright protection of computer documents, reverse engineering, and professor Miller. *University of Dayton Law Review*, Ohio, v. 19, pp. 975-1019, 1994.

KARJALA, Dennis S. Copyright protection of computer program structure. *Brooklyn Law Review*, New York, v. 64, pp. 519-543, 1998.

LAURIE, Ronald S. Comment: use of levels of abstraction analysis for computer programs. *AIPLA Q.J.* Washington, v. 17, pp. 232-236, 1989.

PONTES, Leonardo Machado. *Fair use doctrine* e as obras e não publicadas. *Jornal das Faculdades Milton Campos*, v. XV, n.º 127, 2009.

ROTHNIE. Warwick A. Idea and expression in a digital world. *University of Tasmania Law School, Journal of law and information science*, Hobart, v. 9, n. 1, pp. 59-76, 1998.

SAMUELS, Edward. The Idea-Expression Dichotomy in Copyright Law. *Tennessee Law Review*. Tennessee, v. 56, p. 322-462, 1989.

SHUSTER, Todd H. Legal protection of computer software under American and French law. *RDAI/IBLJ*, London, n. 8, pp. 915-944, 1992.

SHUSTER, Todd H. Originality in computer programs and expert systems: discerning the limits of protection under copyright laws of France and United Stats. *The Transnational Lawyer*, US, v. 5, pp. 3-96, 1992.

SIMON, Ilanah. South Africa Supreme Court rules on copyright in software and computer-generated works. *Journal of Intellectual Property & Practice*, London, v. 1, n. 11, pp. 696-699, 2006.