Applying the AFC Test: What the AFC Test Reveals

Josh Siegel | Senior Manager & Forensic Examiner
T.J. Wolf | Senior Consultant & General Counsel

In the first installment in this series, we answered the question, What is the AFC test? In the second installment we discussed when to apply the AFC test. In this installment, we consider what the AFC test reveals.

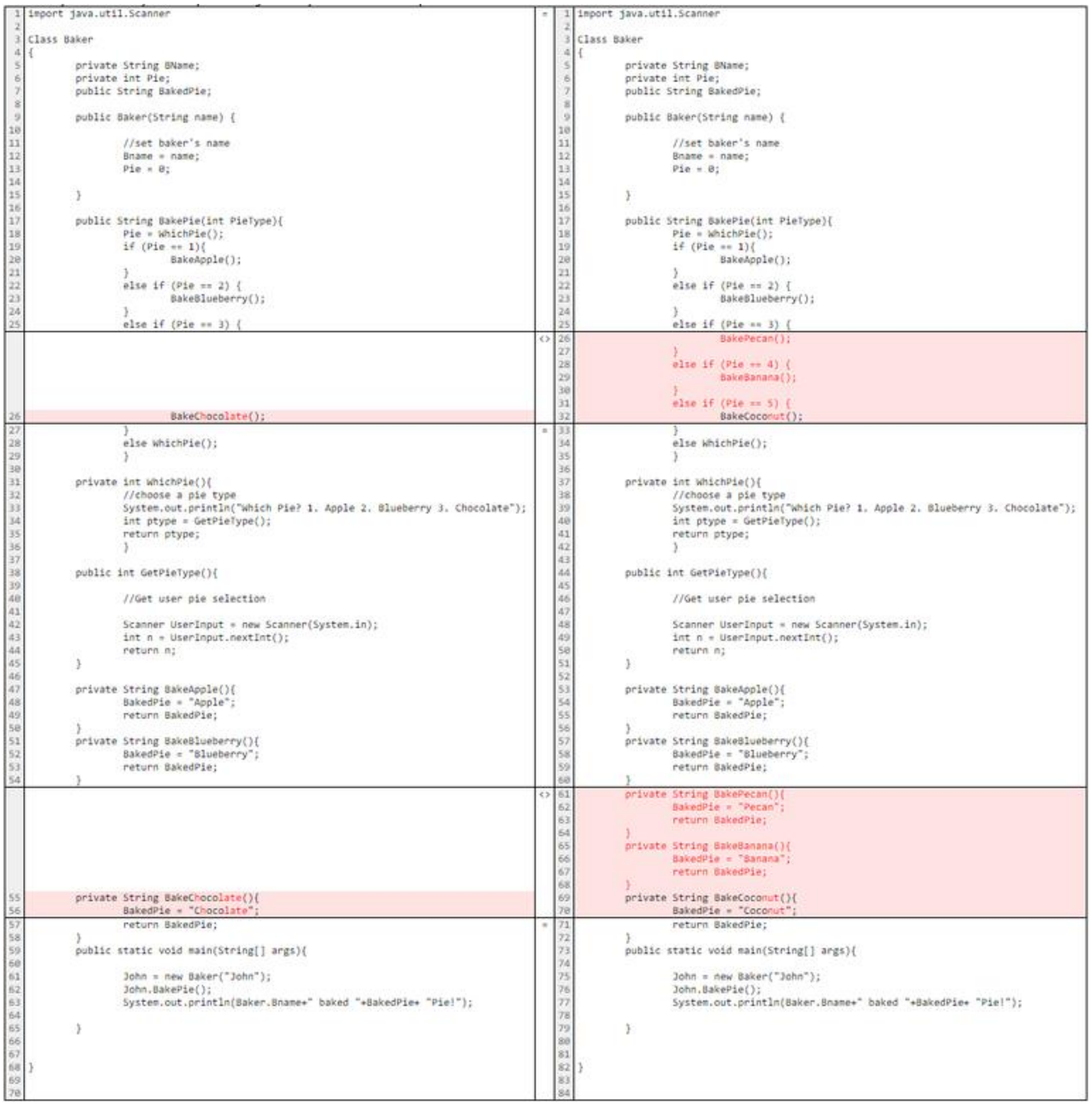
Consider the following example, which illustrates the AFC test's ability to reveal hidden similarities in the structure, sequence, and organization of two programs.

Company A creates a Java program for baking pies. This simple program is named "Bake Pie," and Company A holds a registered copyright for "Bake Pie."

Company B also creates a Java program for baking pies. This program is named "Perfect Pie." The "Perfect Pie" program was written by a former employee of Company A, and evidence indicates that the employee took a thumb drive containing the "Bake Pie" code with him after his termination. Company A sues Company B and former employee for copyright infringement, and code for both programs is produced for experts to review.

An expert must first determine whether literal copying occurred, perhaps by using text comparison software to directly compare the programs line-by-line.

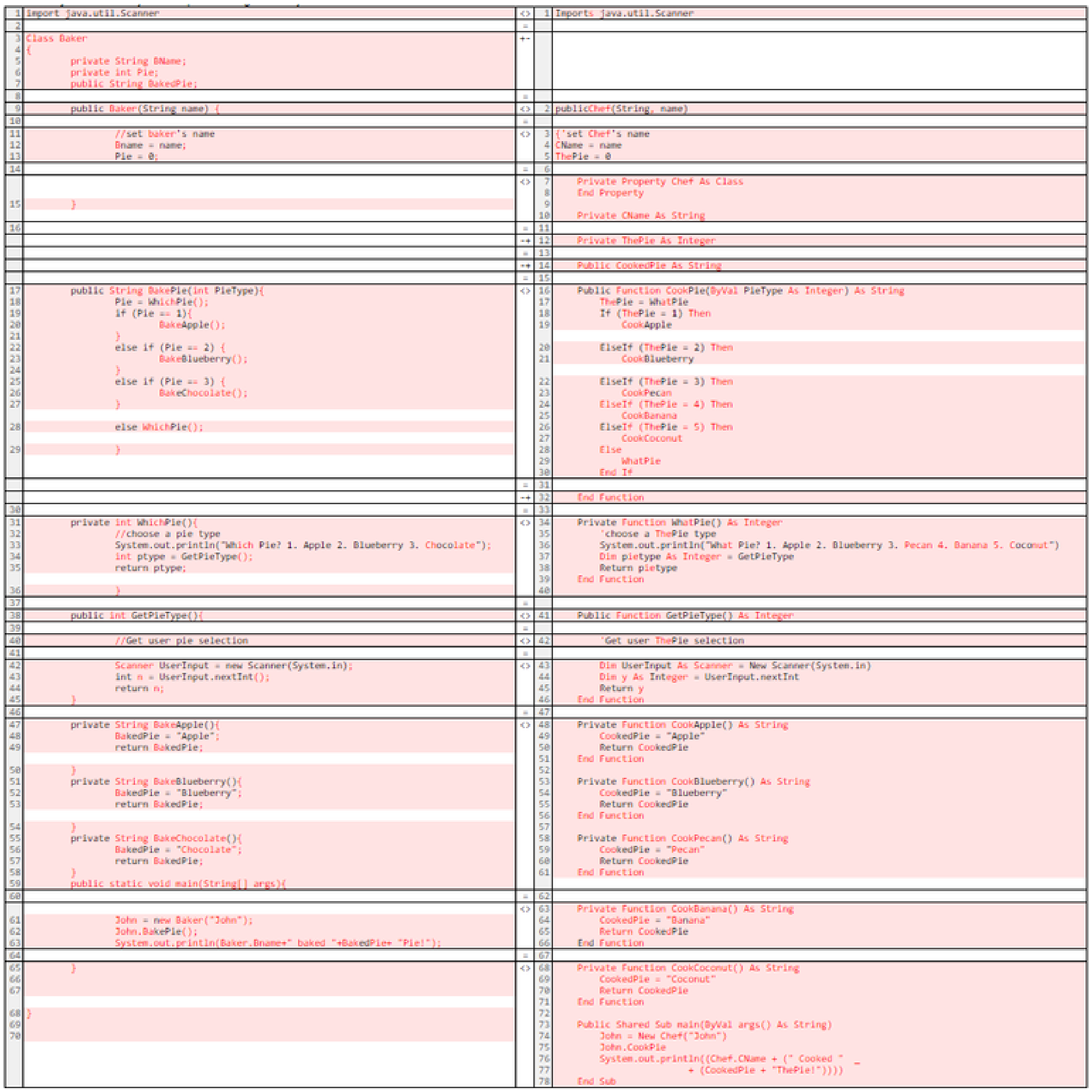
An expert must first determine whether literal copying occurred, perhaps by using text comparison software to directly compare the programs line-by-line. Were an expert to use a text comparison software, like Beyond Compare1, to examine the "Bake Pie" and "Perfect Pie" programs, the result of such a comparison might look something like the screenshot below, with "Bake Pie" on the left and "Perfect Pie" on the right. Lines in white match exactly, while lines highlighted in red do not match.



```
import java.util.Scanner

Class Baker
{
        private String BName;
        private int Pie;
        public String BakedPie;

        public Baker(String name) {

                //set baker's name
                Bname = name;
                Pie = 0;

        }

        public String BakePie(int PieType){
                Pie = WhichPie();
                if (Pie == 1){
                        BakeApple();
                }
                else if (Pie == 2) {
                        BakeBlueberry();
                }
                else if (Pie == 3) {
                        BakeChocolate();
                }
                else WhichPie();
                }

        private int WhichPie(){
                //choose a pie type
                System.out.println("Which Pie? 1. Apple 2. Blueberry 3. Chocolate");
                int ptype = GetPieType();
                return ptype;
                }

        public int GetPieType(){

                //Get user pie selection

                Scanner UserInput = new Scanner(System.in);
                int n = UserInput.nextInt();
                return n;
        }

        private String BakeApple(){
                BakedPie = "Apple";
                return BakedPie;
        }
        private String BakeBlueberry(){
                BakedPie = "Blueberry";
                return BakedPie;
        }
        private String BakeChocolate(){
                BakedPie = "Chocolate";
                return BakedPie;
        }
        public static void main(String[] args){

                John = new Baker("John");
                John.BakePie();
                System.out.println(Baker.Bname+" baked "+BakedPie+ "Pie!");

        }


}
```

```
import java.util.Scanner

Class Baker
{
        private String BName;
        private int Pie;
        public String BakedPie;

        public Baker(String name) {

                //set baker's name
                Bname = name;
                Pie = 0;

        }

        public String BakePie(int PieType){
                Pie = WhichPie();
                if (Pie == 1){
                        BakeApple();
                }
                else if (Pie == 2) {
                        BakeBlueberry();
                }
                else if (Pie == 3) {
                        BakePecan();
                }
                else if (Pie == 4) {
                        BakeBanana();
                }
                else if (Pie == 5) {
                        BakeCoconut();
                }
                else WhichPie();
                }

        private int WhichPie(){
                //choose a pie type
                System.out.println("Which Pie? 1. Apple 2. Blueberry 3. Chocolate");
                int ptype = GetPieType();
                return ptype;
                }

        public int GetPieType(){

                //Get user pie selection

                Scanner UserInput = new Scanner(System.in);
                int n = UserInput.nextInt();
                return n;
        }

        private String BakeApple(){
                BakedPie = "Apple";
                return BakedPie;
        }
        private String BakeBlueberry(){
                BakedPie = "Blueberry";
                return BakedPie;
        }
        private String BakePecan(){
                BakedPie = "Pecan";
                return BakedPie;
        }
        private String BakeBanana(){
                BakedPie = "Banana";
                return BakedPie;
        }
        private String BakeCoconut(){
                BakedPie = "Coconut";
                return BakedPie;
        }
        public static void main(String[] args){

                John = new Baker("John");
                John.BakePie();
                System.out.println(Baker.Bname+" baked "+BakedPie+ "Pie!");

        }


}
```

Most of the programs' lines match exactly, indicating that literal copying occurred. A situation in which literal copying occurred may not require an expert to perform the AFC test. But perhaps, instead of creating "Perfect Pie" in Java, the language used in "Bake Pie," Company B creates it in the Visual Basic .NET language. The below screenshot illustrates what happens when an expert compares the two programs using Beyond Compare.



```
import java.util.Scanner

Class Baker
{
        private String BName;
        private int Pie;
        public String BakedPie;

        public Baker(String name) {

                //set baker's name
                Bname = name;
                Pie = 0;

        }


        public String BakePie(int PieType){
                Pie = WhichPie();
                if (Pie == 1){
                        BakeApple();
                }
                else if (Pie == 2) {
                        BakeBlueberry();
                }
                else if (Pie == 3) {
                        BakeChocolate();
                }

                else WhichPie();

                }


        private int WhichPie(){
                //choose a pie type
                System.out.println("Which Pie? 1. Apple 2. Blueberry 3. Chocolate");
                int ptype = GetPieType();
                return ptype;

                }

        public int GetPieType(){

                //Get user pie selection

                Scanner UserInput = new Scanner(System.in);
                int n = UserInput.nextInt();
                return n;
        }

        private String BakeApple(){
                BakedPie = "Apple";
                return BakedPie;

        }
        private String BakeBlueberry(){
                BakedPie = "Blueberry";
                return BakedPie;

        }
        private String BakeChocolate(){
                BakedPie = "Chocolate";
                return BakedPie;
        }
        public static void main(String[] args){


                John = new Baker("John");
                John.BakePie();
                System.out.println(Baker.Bname+" baked "+BakedPie+ "Pie!");

        }


}
```

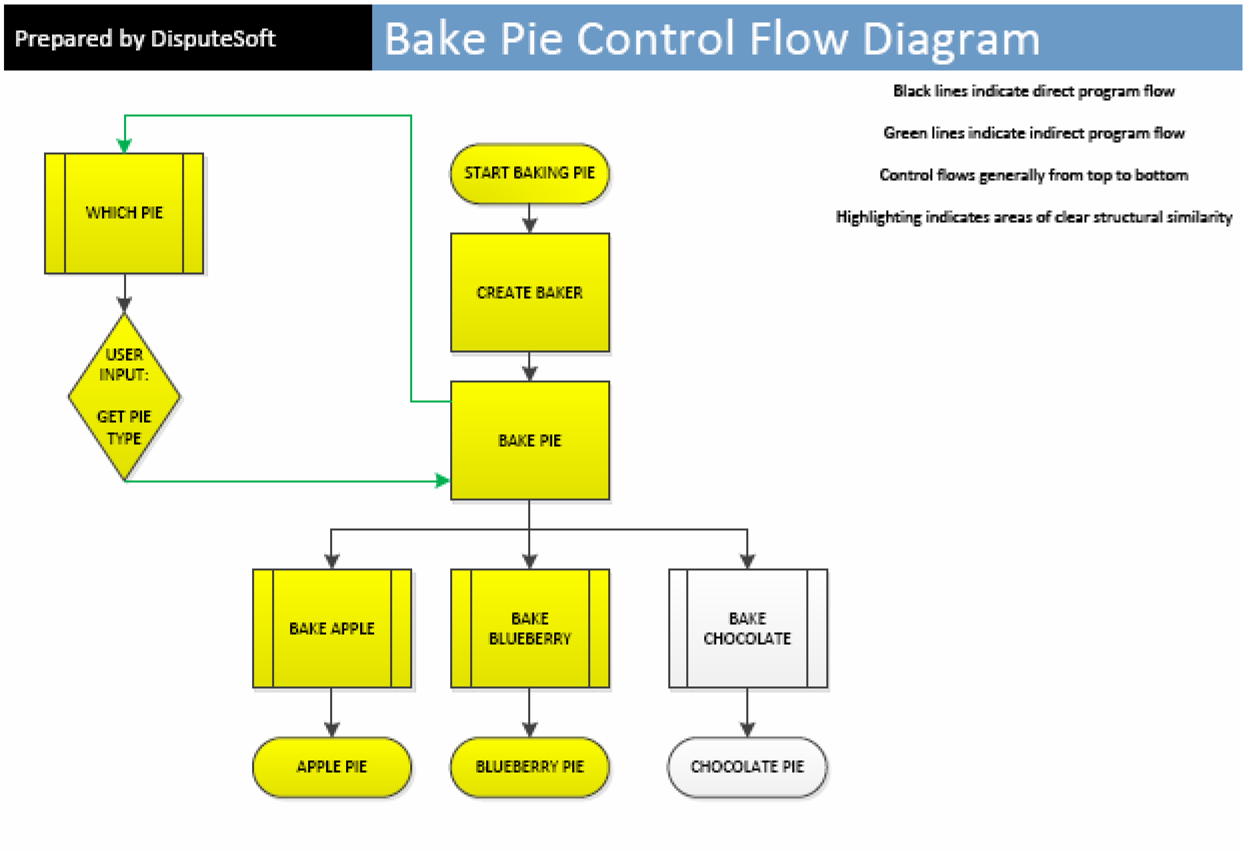
```
Imports java.util.Scanner


publicChef(String, name)

{'set Chef's name
CName = name
ThePie = 0

    Private Property Chef As Class
    End Property

    Private CName As String

    Private ThePie As Integer

    Public CookedPie As String

    Public Function CookPie(ByVal PieType As Integer) As String
        ThePie = WhatPie
        If (ThePie = 1) Then
            CookApple

        ElseIf (ThePie = 2) Then
            CookBlueberry

        ElseIf (ThePie = 3) Then
            CookPecan
        ElseIf (ThePie = 4) Then
            CookBanana
        ElseIf (ThePie = 5) Then
            CookCoconut
        Else
            WhatPie
        End If

    End Function

    Private Function WhatPie() As Integer
        'choose a ThePie type
        System.out.println("What Pie? 1. Apple 2. Blueberry 3. Pecan 4. Banana 5. Coconut")
        Dim pietype As Integer = GetPieType
        Return pietype
    End Function


    Public Function GetPieType() As Integer

        'Get user ThePie selection

        Dim UserInput As Scanner = New Scanner(System.in)
        Dim y As Integer = UserInput.nextInt
        Return y
    End Function

    Private Function CookApple() As String
        CookedPie = "Apple"
        Return CookedPie
    End Function

    Private Function CookBlueberry() As String
        CookedPie = "Blueberry"
        Return CookedPie
    End Function

    Private Function CookPecan() As String
        CookedPie = "Pecan"
        Return CookedPie
    End Function

    Private Function CookBanana() As String
        CookedPie = "Banana"
        Return CookedPie
    End Function

    Private Function CookCoconut() As String
        CookedPie = "Coconut"
        Return CookedPie
    End Function

    Public Shared Sub main(ByVal args() As String)
        John = New Chef("John")
        John.CookPie
        System.out.println((Chef.CName + (" Cooked " _
                        + (CookedPie + "ThePie!"))))
    End Sub
```

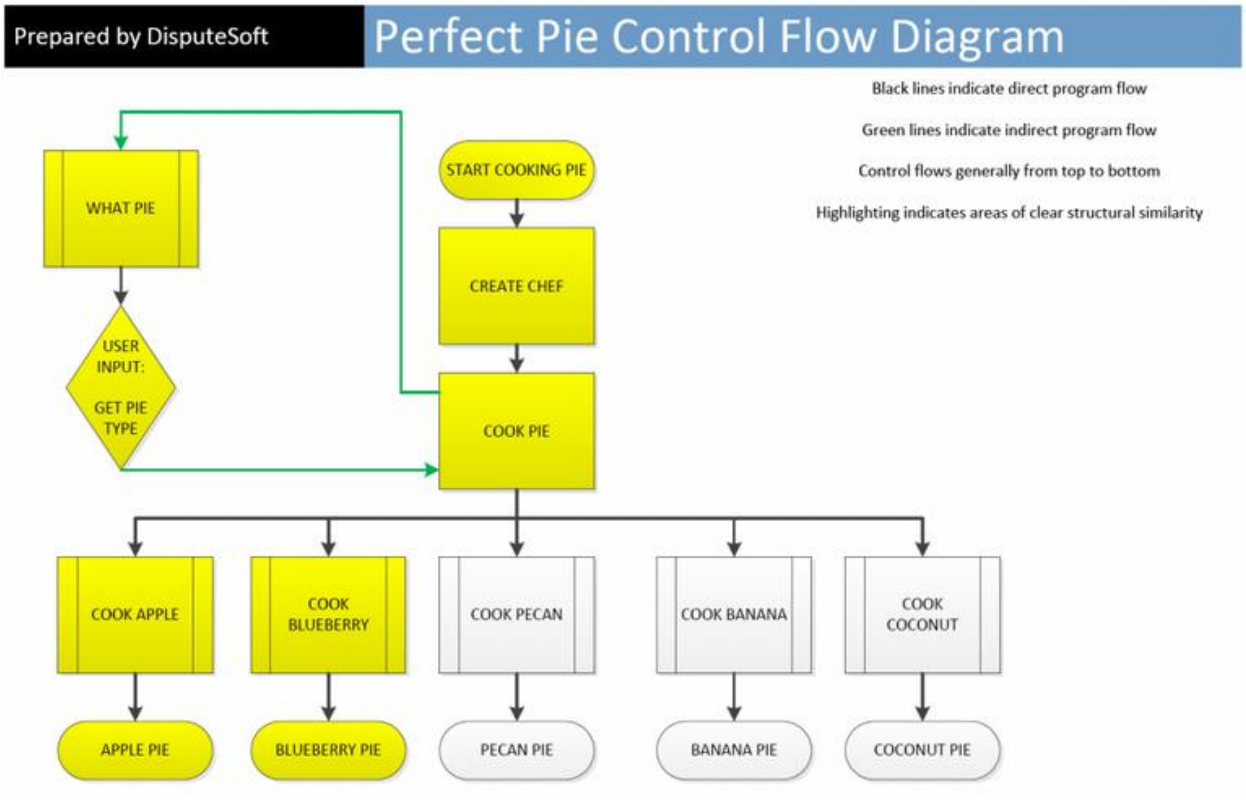
One cannot immediately determine whether one program was copied from the other by merely looking at the literal code, as no text matches exactly, and the programs share only blank lines. This is a direct result of "Perfect Pie" being written in an entirely different programming language than "Bake Pie." However, these two programs still may be similar in other ways.2

While the text comparison software indicates that "Pefect Pie" did not copy source code directly from "Bake Pie," the two programs' variable names, method names, and program sequence are suspiciously similar, raising the question of whether non-literal elements of "Perfect Pie" were copied from “Bake Pie.” How might an expert determine whether “Perfect Pie” exhibits similarity to “Bake Pie” with regard to structure, sequence, and organization?

In this example, assume that the employee in question had access to the original protected work of Company A. This fact allows us to analyze the two programs for evidence of non-literal copying. One method of conducting such an analysis is to apply the AFC test. For the purposes of this example, we abstract each program into simple modules, with a module for each function. Assume as well for this example that after applying various filters (e.g., legal doctrines, industry-specific requirements, etc.), we have concluded that nothing needs to be filtered out of the programs, as they contain only protectable code. Now that the abstraction and filtration steps are complete, we can compare the programs; one way to do this is by examining the programs’ “control flow” through the modules.

The way a program gets from start to finish is referred to as its control flow. A computer program that needs to accomplish a specific task can do so using a variety of different control flow structures, and a programmer’s decision to design a program’s control flow one way instead of another way may arguably be protectable expression. Examining control flow can reveal similarities in two programs’ structure, sequence and organization that may have remained hidden from other analyses. Below are two control flow diagrams, one for “Bake Pie” and one for “Perfect Pie,” with similarities in structure, sequence, and organization highlighted in yellow.


Prepared by DisputeSoft
Bake Pie Control Flow Diagram
Black lines indicate direct program flow
Green lines indicate indirect program flow
Control flows generally from top to bottom
Highlighting indicates areas of clear structural similarity
WHICH PIE
USER INPUT: GET PIE TYPE
START BAKING PIE
CREATE BAKER
BAKE PIE
BAKE APPLE
BAKE BLUEBERRY
BAKE CHOCOLATE
APPLE PIE
BLUEBERRY PIE
CHOCOLATE PIE


Prepared by DisputeSoft
Perfect Pie Control Flow Diagram
Black lines indicate direct program flow
Green lines indicate indirect program flow
Control flows generally from top to bottom
Highlighting indicates areas of clear structural similarity
WHAT PIE
USER INPUT: GET PIE TYPE
START COOKING PIE
CREATE CHEF
COOK PIE
COOK APPLE
COOK BLUEBERRY
COOK PECAN
COOK BANANA
COOK COCONUT
APPLE PIE
BLUEBERRY PIE
PECAN PIE
BANANA PIE
COCONUT PIE

Although the “Perfect Pie” program contains three pies (Pecan, Banana, and Coconut) that “Bake Pie” does not, and the “Bake Pie” program contains one pie (Blueberry) that “Perfect Pie” does not, the diagrams show that the structure, sequence, and organization of “Perfect Pie” is very similar to that of “Bake Pie.” Both start in the same way, generate a person to create the pie, create the pie by getting user input, call the function to create the pie, and generate a pie as the end result. Such similarity suggests that non-literal elements of “Perfect Pie” were copied from “Bake Pie.” Had “Perfect Pie” accomplished the same task as “Bake Pie” by ordering the steps of its control flow differently or by employing completely different procedures, the non-literal elements of the two programs may have been different enough to conclude that protected expression was not copied from the other.

The AFC test can reveal material similarities between programs that may have otherwise remained hidden. Experts use the process of abstracting programs into modules, filtering out non-protectable elements, and comparing the control flow of the programs to come to an opinion regarding whether two programs that differ literally are still substantially similar. This is especially necessary when a party suspects that protected code was copied and translated into a different programming language or otherwise used as the basis for creating a competing program.

Josh Siegel

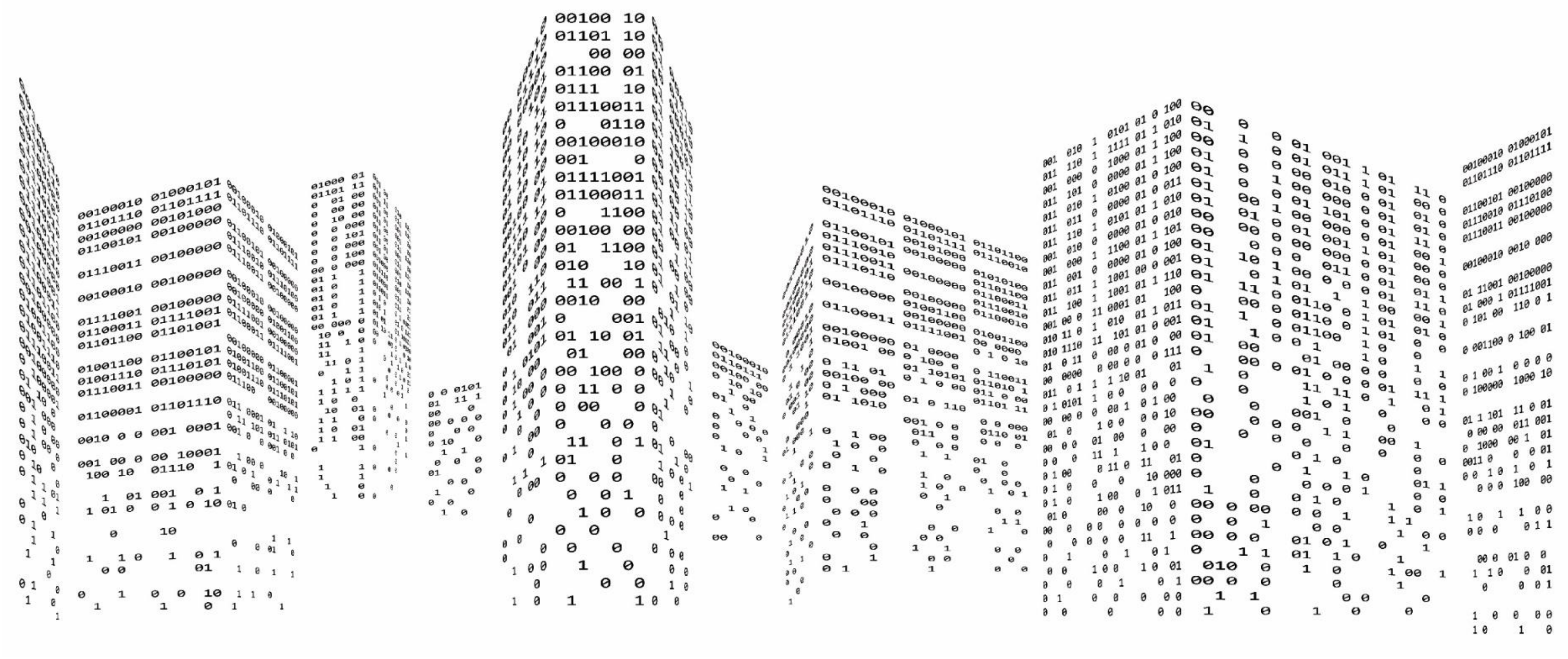
Senior Consultant & Forensic Examiner

Josh Siegel has substantial experience analyzing copyright, patent, and trade secret claims related software and information technology. Josh performs functional testing, analyzes defect systems and metadata, examines source code in intellectual property disputes, acquires and analyzes data in digital forensics, and finally integrates that data into written reports and testimony.

T.J. Wolf

Senior Consultant & General Counsel

Since joining DisputeSoft in 2016, T.J. Wolf has consulted for clients on a variety of software related matters, including breach of contract disputes, software implementation failure matters, and intellectual property matters involving allegations of copyright infringement and trade secret misappropriation. By researching and analyzing documentation to produce content and support for expert reports, T.J. has become deeply involved in analyzing the root causes of many IT failure cases and assessing misappropriation in IP matters.



If you are an attorney in need of an intellectual property expert, we invite you to consider DisputeSoft.

Contact Information

Jeff Parmet, Managing Partner

301.251.6182

jparmet@disputesoft.com

12505 Park Potomac Ave. | Suite 475 | Potomac, MD | 20854