AIPLA QUARTERLY JOURNAL

VOLUME 26, NUMBER 1 WINTER 1998

EDITORIAL BOARD

AIPLA QUARTERLY JOURNAL

CONTENTS

VOLUME 26, NUMBER 1 WINTER 1998

AIPLA QUARTERLY JOURNAL

VOLUME 26, NUMBER 1 PAGE 1 WINTER 1998

ABSTRACTION, FILTRATION, AND COMPARISON IN COMPUTER SOFTWARE COPYRIGHT INFRINGEMENT: AN EXPLANATION AND UPDATE FOR THE OBJECT-ORIENTED PARADIGM

*Michael G. Smith**

* © 1998 Michael G. Smith. Mr. Smith is a Patent Examiner at the U.S. Patent and Trademark Office. Formerly, he supervised object-oriented Internet development projects. Mr. Smith received a J.D. and a B.S. in Computer Science. In addition, Mr. Smith is the Internet webmaster of the AIPLA Copyright Law Committee and Patent Law Committee websites. Mr. Smith can be contacted at mgsmith@softwarelaw.com. The author wishes to express his gratitude and indebtedness to the thoughtful and attentive critical analysis and helpful suggestions of Professor Donald Hughes, Esq., of Capital University Law School in Columbus, Ohio, without which this paper would not have been possible. The author is grateful for the review of the manuscript by the Chair of the AIPLA Copyright Law Committee, Mr. Joseph P. Ebersole, Esq.

I. INTRODUCTION

Infringement of copyright in computer software is generally proven by showing: (1) the alleged infringer had access to the infringed software; and (2) substantial similarity in the protectable expression of the alleged infringing software and the copyrighted software.[1] While literal copying of software is universally prohibited by federal courts and fairly easy to identify, determining the protection of non-literal elements has been problematic. Many tests have been devised by courts to identify substantial similarity in the non-literal elements of software, such as the iterative test, the structure, sequence, and organization (SSO) test, the "look and feel" or "total concept and feel" test, and the successive filtering test.[2] Nonetheless, a consensus has emerged in the federal district courts

[1] *See* Atari, Inc. v. North American Philips Consumer Elec. Corp., 672 F.2d 607, 614, 214 U.S.P.Q. (BNA) 33, 38-39 (7th Cir. 1982). The Eleventh Circuit has raised the degree of similarity required for copyright infringement of software user interfaces, which is distinct from the sequence, structure, and organization of a computer program, from substantial to "virtually identical." *See, e.g.*, Mitek Holdings, Inc. v. Arce Eng'g Co., Inc., 89 F.3d 1548, 1558-59, 39 U.S.P.Q.2d (BNA) 1609, 1613 (11th Cir. 1996). *See also* Apple Computer, Inc. v. Microsoft Corp., 35 F.3d 1435, 1446, 32 U.S.P.Q.2d (BNA) 1086, 1094-95 (9th Cir. 1994) (identifying a computer user interface as a compilation, and holding to a standard of "virtual identicality" in copyright infringment); Harper House, Inc. v. Thomas Nelson, Inc., 889 F.2d 197, 205, 12 U.S.P.Q.2d (BNA) 1779, 1786 (9th Cir. 1989).

[2] *See* John W.L. Ogilvie, *Defining Computer Program Parts Under Learned Hand's Abstractions Test In Software Copyright Infringement Cases*, 91 MICH. L. REV. 526, 550-60 (1992).

upon an expectation that the software is written using Procedural Analysis.

Since *Altai* was decided, another method of systems analysis and programming has arisen that rivals the popularity of Procedural Analysis and Design: Object-Oriented Analysis and Design (OOAD).[5] OOAD is a conceptualization of software that more closely resembles the way people think of the world. OOAD first identifies what things (objects) are in the domain of the problem that the software is to solve. As an object-oriented design (OOD) is defined more specifically, the attributes and functions of the objects in the problem domain are then identified.[6] The interactions of the objects are defined next.[7] Once the OOD is finished, the coding of the design in an object-oriented[8] (OO) language is completed, and the code is compiled into binary for execution on the computer.

OOAD has captured the imagination of the entire software development industry.[9] A substantial portion of software is now developed using OOAD. OOAD promises vast improvements in

---

[5] *See* BORLAND INTERNATIONAL, INC., THE WORLD OF C++ 136 (1991) ("Object-oriented programming has just begun to enter the mainstream for DOS and Windows programming."); GRADY BOOCH, OBJECT-ORIENTED DESIGN WITH APPLICATIONS PREFACE v (1991) ("Object-Oriented design is a relatively young practice."); PETER COAD & EDWARD YOURDON, OBJECT-ORIENTED ANALYSIS 7 (1991) ("Object-Oriented analysis is a relatively young method.").

[6] For example, the object "car" generally is composed of the attributes body, engine, brakes, drive train and wheels. The methods or functions are start engine, stop engine, drive forward, and apply brakes.

[7] To continue the example, in a problem domain where one object is "car" and another object is "driver," one of the functions of human could include a use of the car object where the driver invokes the method function "start engine" of the car.

[8] Compare object-based software, such as Javascript.

[9] *See* Jeff Prosise, *Much Ado About Objects*, PC MAGAZINE, Feb. 7, 1995, at 257.

possible and appropriate to abstract OO software in discrete levels. This method is appropriate because two systems designed using different methodologies can be compared on the same basis, in levels of abstraction. The most appropriate, and perhaps the only valid, taxonomy to abstract structured software is in levels of abstractions, such as John W.L. Ogilvie's levels of abstraction. Ogilvie's levels of abstraction for structured code can also be used for abstracting the SSO of OO software.[15] However, where all of the software to be compared was developed using the OO paradigm, a different taxonomy that more closely reflects the unique nature of OO software is more appropriate, an approach that abstracts software in views rather than levels.[16]

It should be noted that this paper assumes a familiarity by the reader with computer hardware and software,[17] and the constitutional and statutory basis for protecting computer software from copyright infringement.[18]

---

[15] *See infra* Part II.

[16] *See infra* Part III.

[17] *See* Donald F. McGahn II, *Copyright Infringement of Protected Computer Software: An Analytical Method to Determine Substantial Similarity*, 21 RUTGERS COMPUTER & TECH. L.J. 88, 92-95 (1995); Ogilvie, *supra* note 2, at 530-32; Judith A. Szepesi, *Maximizing Protection For Computer Software*, 12 SANTA CLARA COMPUTER & HIGH TECH. L.J. 173, 177-79 (1996); Note, *Copyright Protection of Computer Program Object Code*, 96 HARV. L. REV. 1723, 1724-26 (1983).

[18] *See* Gates Rubber Co. v. Bando Chem. Indus., Ltd. 9 F.3d 823, 829, 28 U.S.P.Q. (BNA) 1503, 1509 (10th Cir. 1993); Computer Assocs. Int'l, Inc. v. Altai, Inc. 982 F.2d 693, 702, 23 U.S.P.Q.2d (BNA) 1241, 1249 (2d Cir. 1992); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1246-47, 219 U.S.P.Q. (BNA) 113, 118-19 (3d Cir. 1983); Stern Elecs., Inc. v. Kaufman, 669 F.2d 852, 855 n.3, 213 U.S.P.Q. (BNA) 443, 445 n.3 (2d Cir. 1982); Williams Elecs., Inc. v. Arctic Int'l, Inc., 685 F.2d 870, 215 U.S.P.Q. (BNA) 405 (3d Cir. 1982); McGahn, *supra* note 17, at 96-98; Szepesi, *supra* note 17, at 184.

the idea of abstraction by starting with all of the lines of literal text and developing successively more specific abstractions for the text has been adopted by courts for other copyrighted works, including computer software.[26]

B. *Courts Struggle With The Abstraction Of Software*

In 1986, the Third Circuit Court of Appeals addressed the copyrightability of the non-literal elements of computer programs in *Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc.*[27] This case proved to be the high-water-mark for protection of the non-literal elements of computer software, as it held that the non-literal elements of computer software were *entirely* copyrightable expression.[28] The court found the only unprotectable part of computer software was the main purpose, or idea of the program, and everything else was protectable by copyright.[29] Because non-literal elements can be written in many different ways, there is no idea in the non-literal elements, just pure expression,[30] which makes the entirety of non-literal elements protectable. The *Whelan* decision was subsequently criticized for offering overly broad, patent-like protection to computer programs' expression and for stifling creativity in the software industry.[31]

The Fifth Circuit Court of Appeals took exactly the opposite approach when it declined to find *any* copyrightability in the non-literal elements of computer software where "market factors play a

[26] *See Altai*, 982 F.2d at 706-07, 23 U.S.P.Q.2d (BNA) at 1253.

[27] 797 F.2d 1222, 230 U.S.P.Q. (BNA) 481 (3d Cir. 1986).

[28] *See id.* at 1237-39, 230 U.S.P.Q. (BNA) at 491-93.

[29] *See id.* at 1239, 230 U.S.P.Q. (BNA) at 493.

[30] *See id. at 1239-40,* 230 U.S.P.Q. (BNA) at 493.

[31] *See generally* Nimmer, *supra* note 14, at 629-30.

infringing program for substantial similarity.[37] The unprotectable elements that are filtered out are idea, program elements dictated by logic and efficiency, external considerations (such as hardware standards, software standards, computer manufacturer's design standards, target industry practices, and computer industry programming standards), and elements taken from the public domain.[38] This test provided a much needed improvement in the law. The test is a recognition that the non-literal elements of computer software may have some protectable expression and some unprotectable idea. This approach avoids the danger of the "take it or leave it" approach used in *Whelan* and *Plains Cotton*.

Nimmer identified a number of abstractions that programmers use in developing software. The "general description of the function of the program"[39] is the main idea or purpose of a program. The "specific outline of the approach"[40] and a structure of the modules[41] can be thought of as "system architecture." The "data structures and algorithms" and the "source code" are other abstractions that programmers use.[42] Nimmer failed to identify abstract data types and the object code as abstractions of computer software, and did not describe the relationship between these abstractions. As later discussed, this remained for John W. Ogilvie to do.[43]

Professor Nimmer's method of determining substantial similarity between the non-literal elements of two programs was used

---

[37] *See id.* at 635.

[38] *See id.* at 640-49.

[39] *See id.* at 637.

[40] *See id.*

[41] *See id.* at 638.

[42] *See id.* at 637-38 (footnote omitted).

[43] *See infra* Part II.D.

comparison test for the non-literal elements of software.[52] The court in *Altai* followed the *Whelan* court to the extent that the non-literal elements of computer software are protectable.[53] However, the *Altai* court correctly departed from the holding in *Whelan* by finding that there may be a mix of idea and expression in the non-literal elements.[54] The court in *Altai* used Nimmer's test involving abstraction, filtration and comparison to determine what portion of the non-literal elements is protectable.[55]

Altai involved two programs, OSCAR 3.4 and OSCAR 3.5, as alleged infringers of ADAPTER.[56] The district court found that OSCAR 3.4 was substantially similar to ADAPTER because "approximately 30 percent [] of OSCAR 3.4 was copied directly from the source code of ADAPTER."[57] OSCAR 3.5 was written from a set of specifications that had been developed by reverse-engineering ADAPTER,[58] thus there was no copying of literal elements of ADAPTER.[59]

[52] *See id.*

[53] *See id.* at 702-03, 23 U.S.P.Q.2d (BNA) at 1249-50.

[54] *See id.* at 703-06, 23 U.S.P.Q.2d at 1250-53.

[55] *See id.* at 706-12, 23 U.S.P.Q.2d at 1253-58.

[56] *See id.* at 696-97, 23 U.S.P.Q.2d at 1246-48.

[57] *See* Computer Assocs. Int'l, Inc. v. Altai, Inc., 775 F. Supp. 544, 560, 20 U.S.P.Q.2d (BNA) 1641, 1651 (E.D.N.Y. 1991), *aff'd*, 982 F.2d 693 (2d Cir. 1992) (finding literal copying in OSCAR 3.4 of the source code of ADAPTER and not examining the non-literal elements). The *Altai* court used a quantitative standard for literal copying, which is the best method because the only qualitative aspects of literal source code are the irrelevant comments. Qualitative standards are much more appropriate for the non-literal elements of computer software.

[58] *See id.* at 552-53, 20 U.S.P.Q.2d (BNA) at 1644-45.

[59] *See id.* at 562, 20 U.S.P.Q.2d (BNA) at 1652.

lists and list of services were scenes a `faire, dictated by external considerations, and therefore, unprotectable.

D. *Ogilvie's Abstraction Formulation*

In 1992, John W. Ogilvie offered a refinement of the Abstractions test.[66] Ogilvie proposed that the non-literal elements of computer software be broken into the following six levels of abstraction: (1) the program's main purpose; (2) its system architecture; (3) various abstract data types; (4) various algorithms and data structures; (5) the source code; and (6) the object code.[67] This test builds on the well-recognized idea that a program can be abstracted into different parts.[68] However, Ogilvie's test identifies a strict *hierarchy* of parts of the abstraction and recognizes that system architecture and abstract data types are parts of the abstraction. This test approximates the levels of abstraction that systems analysts and programmers use in creating software in a procedural analysis.

[66] *See* Ogilvie, *supra* note 2.

[67] *See id.* at 533.

[68] *See* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 707, 23 U.S.P.Q.2d (BNA) 1241, 1253 (2d Cir. 1992) (using Nimmer's abstractions test for computer software, identifying the highest level of abstraction as the "ultimate function of the program" and the lowest level of abstraction as the "individual instructions," although apparently ignoring the data structures and algorithms as levels of abstraction that Nimmer proposed); Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1236, 230 U.S.P.Q. (BNA) 481, 490 (3d Cir. 1986) (identifying the main purpose or the 'idea' of the program as unprotectable); Apple Computer, Inc. v. Formula Int'l, Inc., 562 F. Supp. 775, 783, 218 U.S.P.Q. (BNA) 47, 52 (C.D. Cal. 1983); Ronald S. Laurie, *Comment: Use of a 'Levels of Abstraction' Analysis for Computer Programs*, 17 AIPLA Q.J. 232, 234 (1989) (identifying the "purpose" of the program as one part in an abstraction and the "literal code" as the other part in an abstraction, but not identifying any other parts to an abstraction); Nimmer, *et al.*, *supra* note 14, at 637-38 (discussing abstractions in the context of levels that vaguely foreshadow Ogilvie's levels of abstractions, starting with "a general description of the function that the program is to perform", data structures and algorithms, and source code).

architecture describes the program in terms of various *modules*"[75] which means that, when the system architecture is abstracted, the modules will be abstracted as a part of the system architecture. This makes the explicit abstraction of "modules" by *Gates Rubber* merely redundant. Furthermore, *Gates Rubber* dropped the explicit abstraction of abstract data types. However, the abstract data types must be abstracted in order to arrive at an abstraction of the data structures, which makes the abstraction of the abstract data types implicit in the abstraction of the data structures. Although the levels of abstractions of *Gates Rubber* do not explicitly follow Ogilvie's levels of abstractions, the application may not have theoretical or practical problems because the abstractions instructed by Ogilvie's test can, and will, be abstracted under the levels of abstraction of *Gates Rubber*.[76]

[75] Ogilvie, *supra* note 2, at 534 (emphasis added).

[76] The *Gates Rubber* court ruled as follows on each of the aspects of the program that the district court found was protectable: "menus" (remanded for clarification on whether this was the visual display, or the code that generates it); "constants" (found to be unprotectable facts); "sorting criteria"(remanded for clarification on whether this was the visual display, or the code that generates it); "control and data flow" (remanded for definition, and evaluation of how this relates to the idea/expression dichotomy); "the engineering calculation module and the design module" (remanded to specifically identify how these modules are expression, and perform a filtration analysis for industry standard scenes a faire practices because the record is not clear that these are two separate modules); "common errors" (remanded to determine if there was expression in the common errors); "fundamental tasks" (the court presumed that this is not the main purpose, but remanded for a better explanation of this term); "install files" (remanded for a determination of whether plaintiff held a copyright on the install files, and for a filtration of the install files). *See Gates Rubber*, 9 F.3d at 842-46, 28 U.S.P.Q.2d (BNA) at 1515-19.

orientation focuses on *entities* in the problem domain, while procedural design focuses on *events* that occur within the problem domain.

1. Object-Oriented Analysis And Design

OOAD focuses first on the objects, which are instantiated representations of real entities, and second on the architecture of the relationship of the objects. Accordingly, the process of analyzing and designing objects focuses on the abstraction of the objects and the architecture of the objects.

Examples of objects are: a mixer on a pharmaceutical production line, a checking account, or a weather system. Objects may be tangible as in a weather system, or intangible as in a checking account. What objects have in common is that they exist, they have attributes (data), and the attributes can be modified through functions known as "methods."[82] Objects are created by coding the design into a suitable source language and compiling the source into machine code.

Objects are defined in the abstract by classes. A class definition will include both the data (attributes), and the functions (methods) that each of the objects of the class have in common. Furthermore, OO software features encapsulation, inheritance, and polymorphism.[83]

[82] *See* BORLAND INTERNATIONAL, *supra* note 5, at 130.

[83] While this is the clear consensus of the community of OO analysts, designers, and programmers, at least two members of the community disagree, holding that OO software features abstract data types, which is accomplished through encapsulation, inheritance, and object identity. *See* SETRAG KHOSHAFIAN & RAZMIK ABNOUS, OBJECT ORIENTATION 9 (2d ed. 1995). At least one other author adds messaging to the list of OO features. *See* Sallie Henry & Mathew Humphrey, *Object-oriented vs. Procedural programming languages: Effectiveness in program maintenance*, J. Object-Oriented Programming, June 1993, at 41. In comparison to object-*oriented* software, object-*based* software does not include inheritance and

included in the definition of the child classes. Child classes *inherit* common qualities from their parent class.[86]

It is through inheritance that OOAD yields one of its greatest benefits: extensibility. The system can be extended by deriving new classes from existing classes and adding additional attributes and methods to the derived class as needed.

Furthermore, *polymorphism* is uniquely characteristic of OOAD. In short, polymorphism means many shapes with one interface. It allows a programmer to send messages to an object through a common interface without regard to the implementation details. More specifically, "[p]olymorphism is the ability to call an object member functions without knowing the object's exact type."[87] This is more flexible. It allows the programmer to create programs that can respond to events dynamically without having to create a large amount of conditional program code.[88] Nonetheless, even with polymorphism, OO computer languages still retain the ability to accomplish the same function with conditional code. Accordingly, an OO program may mix with OOAD and structured methodologies.

The process of developing software using an object-oriented design requires early definition of the object classes. The levels of abstractions systems analysts follow in designing an OO program are: (1) problem definition; (2) class development; (3) system architecture design; and (4) coding of source code and compilation into object code.[89]

---

[86] *See* WIENER & PINSON, *supra* note 80, at 2.

[87] *See* Prosise, *supra* note 9, at 260.

[88] *See* HERBERT SCHILDT, C++: THE COMPLETE REFERENCE 422 (2d ed. 1995).

[89] *See* MCGREGOR & SYKES, *supra* note 10, at 44-56.

of a class provides a modular architecture."[97] "The architecture is largely reflective of the messages sent between objects."[98]

The final step is to code the design in a source language, such as C++ or Java, and compile the source code into object code or intermediate code.

2. Procedural Design

A procedural design defines the software in terms of the behavior or function to be performed, starting with a one line description of the main purpose and finishing with numerous descriptions of the low-level functions to be performed. Designers of procedural software follow the function of the proposed software from its highest and most general description to the lowest level necessary for the target source language.[99]

---

[97] *See id.* at 54.

[98] *See id.*

[99] *See* Nimmer, *supra* note 14, at 637-38:

> A programmer starts with very general ideas of what the program is to accomplish, and moves in steps toward the ultimate goal of producing specific code that can operate the computer correctly.
>
> In practice, a programmer usually will start with a general description of the function that the program is to perform. Then, a specific outline of the approach to this problem is developed, usually by studying the needs of the end user. Next, the programmer begins to develop the outlines of the program itself, and the data structures and algorithms to be used. At this stage, flowcharts, pseudo-code, and other symbolic representations often are used to help the programmer organize the program's structure. The programmer will then break down the problem into modules or subroutines, each of which addresses a particular element of the overall programming problem, and which itself may be broken down into further modules

> until it serves no useful purpose to further subdivide a function.[102]

A diagram of the functions has the appearance of a Christmas tree or pyramid, with the main purpose at the pinnacle. This structure is strongly analogous to a thesis paper.[103]

It is clear that a procedural design "emphasize[s] the functionality of the desired end product."[104] In a procedural design, the primary focus is on the function of the software which necessarily relegates data to a secondary consideration.[105]

In a procedural design, the data are defined *within* the functional routines. A major feature of the most sophisticated procedural languages is the ability to pass pointers to data between functions, so that other functions modify the data. This ability would break encapsulation in an OOD.

The process of creating computer software using a procedural design is a process of: (1) defining the program's main purpose; (2) defining the system architecture; (3) defining abstract data types;(4) defining algorithms and data structures; (5) creating the source code; and (6) creating the object code.

The process starts with a systems analyst, who defines the program's main purpose in terms of the action or behavior, usually with a verb. For example, the main purpose of the computer program presented in *Whelan* was "efficient *management* of a dental

[102] *See* ROBERT J. VERZELLO & JOHN REUTTER III, DATA PROCESSING SYSTEMS AND CONCEPTS 307-08 (1982).

[103] In a thesis paper, the main purpose is the thesis proposal, with the main subjects subordinated to the thesis and each of the main subjects are broken down into successively more definite subjects.

[104] *See* MCGREGOR & SYKES, *supra* note 10, at 60.

[105] *See* AT&T, *supra* note 77, § 2, at 13 ("Data structures are considered as

3. The Distinction Between Object-Oriented Analysis And Design And Procedural Design

The distinction between the two approaches is that procedural design focuses on the functional aspects of the problem, while OOAD focuses on the entities:

> The point of view of an object-oriented analysis document should be different from that of a procedural analysis document. Traditional documents are functionally oriented. This point of view perceives the system as providing a set of services. The object-oriented analysis documents should give priority to describing the entities in the problem domain.[107]

A procedural design also lacks the three major features of OOAD: data encapsulation, inheritance, and polymorphism. Many procedural designs will allow other functions to modify data within a function through the use of pointers, which breaks data encapsulation. A procedural design has no inheritance of any kind because there is no definition that links the individual functions in a procedural design.

In terms of the levels of abstraction, the two design methodologies are different in two aspects: (1) in a procedural design, the system architecture is developed in the early stages of the abstraction while in an OOD the system architecture is designed in the context of the objects; and (2) in an OOD, the data structures and algorithms are abstracted in relation to a particular object while in a procedural design the data structures and algorithms are abstracted in the context of the system module of which they are a part.

[107] *See* MCGREGOR & SYKES, *supra* note 10, at 60.

> "simpler constituent problems or '*subtasks*,'" . . . . Sometimes, depending upon the complexity of its *task*, a subroutine may be broken down further into sub-subroutines.[114]

Courts' descriptions of an Abstractions test are fraught with terminology and instructions tied to procedural analysis. For example, "a court would first break down the allegedly infringed program into its constituent *structural* parts."[115] According to the *Altai* court:

> At the lowest level of abstraction, a computer program may be thought of in its entirety as a set of individual *instructions* organized into a hierarchy of modules. At a higher level of abstraction, the *instructions* in the lowest-level modules may be replaced conceptually by the *functions* of those modules. At progressively higher levels of abstraction, the *functions* of higher-level *modules* conceptually replace the implementations of those modules in terms of lower-level modules and *instructions* . . . .[116]

The *Micro Consulting* court defined the non-literal elements as "[t]he *functions* of the modules together with each module's relationships to other modules."[117] "[T]he organization of code into

---

[114] See *Altai*, 982 F.2d at 697, 23 U.S.P.Q.2d (BNA) at 1245 (emphasis added).

[115] *See id.* at 706, 23 U.S.P.Q.2d (BNA) at 1252 (emphasis added). This instruction indicates that the system architecture is at the top of the hierarchy of abstractions, which is true in Procedural Design, but not true in OOAD.

[116] *See id.* at 707, 23 U.S.P.Q.2d (BNA) at 1253 (emphasis added).

[117] Micro Consulting, Inc. v. Zubeldia, 813 F. Supp. 1514, 1529 n.15 (W.D. Okla. 1990), *aff'd*, 959 F.2d 245 (10th Cir. 1992)(emphasis added).

analysis better than purely literary works. Therefore, once the enigmatic natures of law and software have been reconciled by developing an analytical method of abstracting software in terms that judicial triers of fact understand, judicial decisions on infringement of software copyright can be made with greater confidence.

Because only expression is protected under copyright laws, idea and expression must be separated.[122] Accurately determining the expression in computer software first requires that the program be abstracted in a manner that allows the idea to be *clearly distinguishable* from the expression of the idea. A test for substantial similarity must satisfy this standard.[123] The differences between the two design methodologies, procedural and OO,[124] present problems in abstracting an OOD computer program clearly enough to distinguish idea from expression.

The legal standard for substantial similarity needs to reflect programming practices. When the court's concept of a software program is different than the programmer's concept, it is more likely that a court will not be able to correctly separate idea from expression in the software, possibly leading to an incorrect result.[125] This could

---

blocks to build new, improved, practical computer tools.") (footnote omitted).

[122] *See* H.R. REP. NO. 94-1476, at 54 (1976), *reprinted in* 1976 U.S.C.C.A.N. 5659, 5667.

[123] *See* Bateman v. Mnemonics Inc., 37 U.S.P.Q.2d (BNA) 1225, 1234 n.27 (11th Cir. 1995) ("It is essential that one keep in mind that the approaches adopted by the other circuits merely are a means to a very important end: filtering out all unprotectable material. Sometimes parties become so engrossed in disputing what "test" should apply that they lose sight of what the tests were designed to accomplish in the first place. To paraphrase a sage observer, "if you don't know where you're goin', when you get there you'll be lost.").

[124] *See supra* Part III.A.3.

[125] An example of OO C++ source code, and an explanation of how a lack of OO concepts will lead to an incorrect abstraction of OO software,

```
    // -- Add 2 for the border
    Width  += 2;
    Height += 2;

    // -- Ensure menu isn't larger than screen
    eight = (Height > LINES - 2 ? LINES - 2 : Height);

    // -- Create the Window

    WindowPtr = newwin (Height, Width, (LINES - Height) /2, (COLS - Width) / 2);
    ThePanel = new_panel (WindowPtr);
    set_menu_format (TheMenu, Height-2, 1);
    wborder (WindowPtr, 0, 0, 0, 0, 0, 0, 0, 0);
    set_menu_win (TheMenu, panel_window (ThePanel));
    set_menu_sub (TheMenu, derwin (panel_window (ThePanel),
       Height - 2, ItemWidth, 1, (Width -ItemWidth)/2));
    init_menu();
    start_child(TheMenu);
    mvwaddstr (panel_window (ThePanel), 0, ((ItemWidth -
       strlen (MenuTitle))/2) + 1, MenuTitle);
    update_panels ();
    post_menu (TheMenu);
    doupdate ();
    keypad (panel_window (ThePanel), 1);
    for(int I=0;i<NumItems;i++)
    {
       if(strncmp(match, item_name (MenuItems[i]), strlen(match))<0)
       {
          I--;
          if (i<0) I=0;
          set_current_item(TheMenu, MenuItems[i]);
          break;
       }
    }
    return HandleKeystrokes ();
};
//_____________________________________________________
#include "popmenu.h"
#include "database.h"
struct owneritem
{
    char item[18];
};

typedef struct owneritem OwnerItem;

class OwnerMenu : public PopupMenu
```

```
141          DriverMenu();
142          virtual int Show(){return TRUE;};
143          virtual int Show(char* match);
144       private:
145          DriverItem Driverlist[MAX_CHOOSE_ITEMS];
146    }
147
148    #include <string.h>
149
150    DriverMenu :: DriverMenu() : PopupMenu("Select a Driver")
151    {
152       DriverStruct Drivers[MAX_CHOOSE_ITEMS];
153       int iRecords=GetDriverList(Drivers);
154       if(iRecords)
155       {
156          char MenuString[22];
157          for(int I=0; i<iRecords; I++)
158          {
159             strncpy(Driverlist[i].item, Drivers[i].Name, 10);
160             strcat (Driverlist [I].item, "       ");
161             Driverlist[i].item[10]='\0';
162             strcat(Driverlist[i].item, " = ");
163             strcat(Driverlist[i].item, Drivers[i].ID);
164             AddItem (Driverlist[i].item, "");
165          }
166       }
167    };
168    //____________________________________________________________
```

Applying the Ogilvie Levels of Abstraction to the above OO source code yields a purpose: To "present pop-up menus," which is the main purpose of the main class. Beyond that, difficulties arise.

Lacking the concept of encapsulation, this source code would appear to be a series of disjointed function prototypes, (lines 4-6,92-94,96,141-143,145) data definitions (lines 82-87, 131-136) and functions (lines 15-79, 101-125, 150-167) with no relationship between the data and the functions, when in fact, the above source code is three classes in which data (attributes) and functions (methods) are encapsulated.

Lacking the concept of inheritance, this source code would appear to be a series of three separate classes, with no relationship between the classes. In fact, the above source code defines a parent class PopupMenu derived from class Menu (line 1), that is subclassed into children classes OwnerMenu (line 89) and DriverMenu (line 138) where the child class

Ogilvie's levels of abstraction reflect a conventional software development process. Programmers have traditionally developed software starting at the highest levels of abstraction, working down to more specific levels.[128] The first element of the design, the main purpose, is the highest level of abstraction. The last elements of the design, the source and object code, are the lowest levels of abstraction. However, levels of abstraction date back to at least 1968 and are now considered a traditional process.[129] Object-orientation allows systems analysts to take a radical departure from this traditional structured software development process. Accordingly, the abstraction of OO software can be performed in a drastically different manner, and the filtration and comparison can be modified to a lesser extent.

A. *Abstraction*

It has been said that OO is as easy as PIE, but one must spell it backwards; [E]ncapsulation, [I]nheritance and [P]olymorphism, the unique characteristics of OO software. OO software can be abstracted in three views that clearly describe its unique characteristics: generalization/specialization, association/membership, and aggregation/composition.[130] These three views can be used to abstract OO software in its entirety and not merely the architecture of the classes. Each view provides different insight into the OO software that encompass the architecture, as well as the source code that is compiled into binary code. Specific notations are necessary to describe each of the views. These class relationships can be

[128] In comparison, OO software may be designed by starting at the highest level of abstraction, and then working down through levels of abstraction, or by starting at the lowest levels of abstraction, and working up. Regardless at what level of abstraction the design process starts, OO software design is typically an iterative process, where the design is refined through a number of passes. *Id.*

[129] *See* E. Dijkstra, *The Structure of the "THE" Multiprograming System*, 11 COMM. OF THE ACM 5 (1968).

[130] Classification/instantiation is another view that has been identified but has been omitted by subsequent authors. *See* Eric Jinshuan Lee, An object-

Inheritance is implemented via generalization and specialization.[134] Classes are related by inheritance in a parent-child or parent-children hierarchical tree relationship.[135] The general, or common, characteristics of classes are described in the parent class.[136] The specialized characteristics are described in the child classes.[137] The general characteristics of the parent class are inherited by the child classes and the specialized characteristics are additional to the child class. The generalization/specialization view is characterized by "is a kind of" relationships, such as a cost accountant is a kind of accountant, or a truck is a kind of vehicle.[138] When a systems analyst documents the generalization/specialization aspects of an OO system, the uniquely OO-related concept of inheritance will also be captured

[134] *See* MARTIN FOWLER & KENDALL SCOTT, UML DISTILLED 68 (1997); RICHARD C. LEE & WILLIAM M. TEPFENHART, UML AND C++ 165 (1997). Accordingly, the generalization/specialization relationship can be documented in UML via an inheritance diagram, or an object diagram in Booch design methodology. *See* LEE & TEPFENHART, *supra*, at 35, 225. The generalization and specialization extends beyond the class relationships to the class internals. Generalization and specialization in the class internals can be documented using a diagram and notation specifically designed for that purpose. *See* Lee, *supra* note 131, at 33-34.

[135] *See supra* Part III.A.1. Implement action occurs via the mechanism of class derivation in C++ or Java. *See* LEE & TEPFENHART, *supra* note 134, at 231. Every inheritance tree has one class as the axiomatic class, from which levels of subclasses descend. *See id.* An OO system will have any number of inheritance trees. Nearly all OO systems have numerous inheritance trees. *Id.*

[136] In the context of generalization and specialization, the parent class is often referred to as the *supertype* and the child class is referred to as the *subtype*, which is consistent with the notion that a class is an abstract data *type*. *See infra* Part V.C.

[137] *See id.*

[138] To determine if two classes or objects are related via generalization/specialization, "[w]e ask, 'Is object A an object B?' 'Is object B an object A?'" *See* LEE & TEPFENHART, *supra* note 134, at 110. The allowable answers are *always*, *sometimes*, and *never*. *See id.* If the answer to both questions is *never*, the objects are not in an is_a relationship with each other. *Id.* If both answers are *always*, object A and object B are

including C++, the most commonly used OO language, and Java, an OO language that is likely to exceed C++ in popularity.[140] Moreover, these views do not describe encapsulation, inheritance, or

```
      {
        ~engine(); //destruct the engine
        ~body();      //destruct the body
      };
    private:
      char name[20];
      Engine*   engine;  //buried pointer to engine
      Body*   body;  //buried pointer to body

  };
```

Alternatively, aggregation may be implemented in C++ via embedded objects, in which the aggregated objects are not accessible directly by other objects:

```
class car
{
  public:   car()   //constructor
      {
        engine=Engine(); //instantiate the engine
        body=Body();      //instantiate the body
      };
      ~car() //destructor
      {
        ~engine(); //destruct the engine
        ~body();      //destruct the body
      };
      Void StartEngine(int I) /tell the car to start        {
              //the engine
          engine.start(I);
        }
  private:
      char name[20];
      Engine   engine;  //buried pointer to engine
      Body   body;  //buried pointer to body

};
```

[140] Java, a new OO language developed in the mid 1990s by Sun Microsystems, is foreseen by many industry observers as the natural successor to C++ because of its support for distributed architectures. Java may very well become the most popular OO language by the turn of the century.

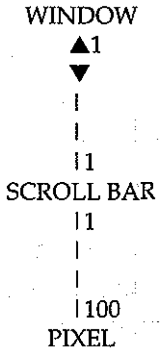
In the aggregation/composition view, classes can be described as "part of"[144] or "is composed of" relationships.[145] The relationship is similar to one in which a car is composed of a body and engine, a computer is composed of a case, motherboard and power supply, or a page is a part of a book. The distinction between the aggregation and composition is that objects that are composed into one object will not exist if the composition ceases to exist, while in aggregation, the aggregated objects may continue when the aggregation fails. For example, a car is an aggregation because the tires can exist after the car ceases; the tires may be removed at the junkyard and be fitted onto another car. A software window is a composition because the scroll bar will cease to exist when the window is destroyed. In languages that explicitly support association, support is accomplished through association classes.[146]

---

[144] *See* GRADY BOOCH, OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS 64 (1994).

[145] *See* BOOCH, *supra* note 131, at 180.

[146] Aggregation and composition is documented architecturally in UML by aggregation diagrams. *See* LEE & TEPFENHART, *supra* note 134, at 37-38. For example a class named "window" that is composed of a "scroll bar," which is in turn an aggregation of 100 "pixels" would be depicted as follows:



See id.

Aggregation and composition extends beyond the class relationships to the class internals. Aggregation and composition in the class internals can development of OO system architecture.[149] A design pattern only describes a solution to a particular design problem, it is not itself code.[150] Design patterns attempt to identify higher-level concepts, such as "producer-consumer", that recur in many software systems. Design patterns facilitate the widespread reuse of system architecture[151] because the design patterns movement seeks to identify, document, and publicly disseminate patterns for the economic benefit of society. The long-term goal of the design pattern movement is to catalog and develop software engineering handbooks of design patterns for re-use by software engineers.[152] Accordingly, OO software that is based on design patterns will have a greater degree of public domain elements.

Similarly, frameworks[153] are likely to contain public domain elements, in their entirety or in part. A framework is a collection of objects and/or classes designed for a specific application, such as bank electronic funds transfer or industrial process control. This is in contrast to design patterns which are applicable across applications and industries. Frameworks are embodied as computer program code, and thus are less abstract than design patterns. Both design patterns and frameworks are likely to contain elements that can be filtered out.

---

[149] *See* Douglas C. Schmidt, et al., *guest editorial*, 39 COMM. OF THE ACM 10 (1996).

[150] *See* F. J. Budinsky, et al., *Automatic Code Generation from Design Patterns*, 35 IBM SYSTEMS JOURNAL 2 (1996).

[151] *See* Schmidt, *supra* note 149.

[152] *See id*.

[153] Frameworks dictate architectural design and facilitate design reuse by providing a collection of cooperating classes that can be customized to particular applications by subclassing new classes. *See* GAMMA, *supra* note 148, at 26-28. Frameworks differ from design patterns in that a framework is less abstract than a design pattern, a framework is a larger architectural element than a design pattern (a typical framework contains several design patterns, but a design pattern never contains several frameworks), and frameworks are more specialized in their application domain than

polymorphism, inheritance, and encapsulation can be described in Ogilvie's levels of abstraction: encapsulation can be accounted for in the various abstract data types, inheritance can be accounted for in the system architecture, and polymorphism can be accounted for in the various algorithms and data structures.

A. *Main Purpose*

Every software system has a main purpose. The main purpose of an OO system will be defined in terms that are broader and more general when the software system is larger and more encompassing. The main purpose will be defined in terms of the problem domain in which the OO system is designed to operate. An OO software system may also contain more than one executable program and, accordingly, has the potential to encompass many more functions than an OO system that has fewer executables. In comparison, software developed from traditional programming methods can be abstracted starting at each individual executable program because software is typically designed one program at a time.

For a new OO software system:

> there exists some moment in time where, in the mind of the developer, the architect, the analyst, or the end user, there springs forth an idea for some application. This idea may represent a new business venture, a new complementary product in an existing product line, or perhaps a new set of features for an existing software system.[155]

An individual object or class does not have a main purpose. By definition, classes and objects do not have a purpose because they are not designed along a functional basis. In contrast, software written in

[155] *See* BOOCH, *supra* note 144, at 250.

encapsulation.[159] Furthermore, ADTs are implemented in OO languages by encapsulating data and operations in classes.[160] Because a class encapsulates data and operations, encapsulation is contemplated by the abstraction of ADTs. Accordingly, abstraction of ADTs accounts for the OO characteristic of encapsulation.

ADT is a concept that predates the OO paradigm, but was nonetheless rarely implemented in structured programming.

[159] *See* BRUCE ECKEL, C++ INSIDE AND OUT 22 (1993). *See also* JAMES MARTIN AND JAMES J. ODELL, OBJECT-ORIENTED ANALYSIS & DESIGN 157-59 (1992):

> The *abstract data type* (ADT) extends the notion of the *user-defined type* by adding encapsulation. The ADT contains the representation and the operation of a data type. The *encapsulation* feature of the ADT not only hides the data type's implementation but provides a protective wall that shields its objects from improper use. All interface occurs through named *operations* defined within the ADT. The operations, then, provide a well-defined means for accessing the objects of a data type . . . In this way, an object can be regarded as any instance of an abstract data type.

Id. See also SETRAG KHOSHAFIAN RAZMIK ABNOUS, OBJECT ORIENTATION 3 (2d ed. 1995)("Classes implement a very fundamental concept in object orientation, namely *abstract data types.*"); David M. Papurt, *Additional Aspects of Generalization,* J. OF OBJECT-ORIENTED PROGRAMMING 32 (1996); Murali Sitaraman et al., *On the Practical Need for Abstraction Relations to Verify Abstract Data Type Representations,* 23 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 157 (1997) (Generalization corresponds to a cognitive process, and captures similarities and differences between distinct *abstract data types.*) (emphasis added); Stephen W. Liddle, <http://msm.byu.edu/students/courses/Isys540/FALL1996/princ_oo.htm> ("An ADT is a collection of data fields together with the code needed to manipulate the data.")

[160] *See* J. M. EDWARDS BRIAN HENDERSON-SELLERS, BOOK TWO OF OBJECT-ORIENTED KNOWLEDGE : THE WORKING OBJECT: OBJECT-ORIENTED SOFTWARE ENGINEERING: METHODS AND MANAGEMENT 245 (1994). Other authorities have suggested that a class is not a mere implementation of an ADT, but rather, a class and an ADT are the same concept. IAN GRAHAM, MIGRATING TO OBJECT TECHNOLOGY 352 (1995). ADTs are implemented in the non-OO languages Ada '95 and Modula 3 through the constructs "packages" and "modules" respectively. Mirada, Turing and Euclid are

and development does not accommodate polymorphism, but OO design and development does allow for polymorphism. Polymorphism is abstracted in the various algorithms.

E. *Source Code And Object Code*

The nature of the source code and the object code are not changed by the OO paradigm. The source code notation of the unique OO characteristics of encapsulation, inheritance, and polymorphism is accordingly unique and supplemental to the notation that supports structured programming. However, in the abstract, OO source code is no different from non-OO source code. Furthermore, OO object code and non-OO object code do not exhibit even the mere difference in notation that exists between OO source code and non-OO source code. Indeed, OO object code and non-OO object code are indistinguishable. Therefore, there is no difference in the abstraction of OO object code and non-OO object code. The abstraction of OO source code and OO object code is identical to the abstraction of source code suggested by Ogilvie.

F. *Levels Of Abstraction Of Individual Classes And Objects*

Claims of infringement of the SSO of OO software may either be of an entire OO system, or one or more individual objects or classes. A method of abstracting OO software needs to accommodate either type of infringement claim. Because a class or an object is a subset of an OO system, the appropriate method of abstracting a class or an object is simply using the subset of the above modified version of Ogilvie's system abstraction method that applies to the class or object.

More specifically, for purposes of abstracting an individual object or class, the first two levels of Ogilvie's abstraction, main purpose and systems architecture, have no application. The main purpose, as the highest level of abstraction, does not apply because program written in an OOD. Nonetheless, a trier of fact must have the analytical ability to separate idea from expression.

A court can separate idea from expression in computer software written in an OOD through Ogilvie's levels of abstraction. This method can be applied in a manner that accommodates the unique characteristics of an OO software system. The system architecture describes inheritance, the various abstract data types describe encapsulation, and the various algorithms and data structures describe polymorphism. Furthermore, the system architecture of an OO software system may contain a higher proportion of idea than other levels of abstraction, particularly where design patterns are used. Individual objects have no purpose or architecture to abstract, but nonetheless objects do have various abstract data types, algorithms and data structures, source code and object code to be abstracted.

Abstracting OO software in levels is applicable to inquiries into copyright infringement of OO software, and is most useful where OO software is being compared to structured software. However, where a question of copyright infringement involves only OO software, abstraction should be done in views rather than levels. The three views through which OO software can be abstracted are generalization/specialization, association/membership, and aggregation/composition.

The primary view is generalization/specialization because the unique characteristics of OO software, encapsulation, inheritance, and polymorphism, can be abstracted from that one view. In addition, most OO languages do not explicitly support the other two views. Furthermore, there are a number of graphical documentation representations that can be used to depict the abstraction, filter out the unprotected expression, compare the OO software to determine the existence or non-existence of substantial similarity in the SSO, and finally, objectively communicate the results of the analysis to a non-technical person.

AIPLA QUARTERLY JOURNAL

VOLUME 26, NUMBER 1 PAGE 55 WINTER 1998

GUIDE TO JAPANESE INTELLECTUAL PROPERTY LAW

Brian G. Strawn[*]

[*] © 1998 Brian G. Strawn. Mr. Strawn is a 1998 graduate of The George Washington University Law School. He will join the law firm of White & Case as an associate in its New York office. The author thanks Christopher P. Wells and George J. Hibbert of the Tokyo office of White

processing, the JPO will grant or reject the patent application within thirty-six months of the date on which the applicant files the request for accelerated examination.[3]

4. Restoration Procedure

The JPO now allows the restoration of patents that have lapsed due to non-payment of fees. Restoration is allowed provided that the non-payment was due to reasons beyond the patent holder's control and the patent holder files the application for restoration within six months of the expiration of the normal grace period for the late payment of fees.[4]

5. Priority Filing For TRIPS Members

Filing priority now can be claimed based on applications filed in any country that has adopted the TRIPS Agreement[5]—even if that country is not a member of the Paris Convention.[6]

6. Electronic Applications

Since 1990, the JPO has promoted a "paperless" application system.[7] According to the JPO, some ninety percent of all patent applications now are made either on-line or using floppy disks.[8] Effective April 1998, the JPO will accept electronic applications from user terminals. Further, beginning in

---

[3] *See* Japanese Patent Office, *What's New?* (visited Mar. 20, 1998) <http://www.jpo-miti.go.jp> [hereinafter *What's New?*].

[4] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 6-7.

[5] *See* Agreement on Trade-Related Aspects of Intellectual Property Rights Including Trade in Counterfeit Goods, *opened for signature* Apr. 15, 1994, 33 I.L.M. 81.

[6] Paris Convention for the Protection of Industrial Property, *opened for signature* Mar. 20, 1983, *last revised* July 14, 1967, 21 U.S.T. 1583, 828 U.N.T.S. 307; *see What's New?*, *supra* note 3.

[7] *See* Japanese Patent Office, *Introduction of Policies: Paperless System Concept* (visited March 20, 1998) <http://www.jpo-miti.go.jp>.

[8] *See id.*

3. Examinations On Request

The JPO does not automatically examine applications. The applicant or any third party must request an examination.[13]

4. Priority Filing Under Treaties

Japan observes foreign priority filing under the Paris Convention[14] and the Patent Cooperation Treaty.[15]

C. *Subject Matter*

Japanese patent law provides protection for inventions. The Patent Law defines an "invention" (hatsumei) as a "[h]ighly advanced creation of technical ideas, by which a law of nature is utilized."[16] Either products or processes may embody such inventions.[17] Provided that the statutory requirements for patentability are met, virtually any subject matter may qualify for patent protection, including genetic recombination and biotechnology.[18] Thus, even new plant or animal varieties may be patentable, provided they fall under the definition of "invention" and meet the statutory requirements.[19] Laws of nature (for example, physics or

[13] *See* Patent Law, *supra* note 11, art. 48(2).

[14] Paris Convention for the Protection of Industrial Property, *opened for signature* Mar. 20, 1983, *last revised* July 14, 1967, 21 U.S.T. 1583, 828 U.N.T.S. 307.

[15] Patent Cooperation Treaty, *done* June 19, 1970, 28 U.S.T. 7645, 1160 U.N.T.S. 231.

[16] Patent Law, *supra* note 11, art. 2(1).

[17] *See id.*

[18] *See* Masashige Ohba, *Intellectual Property Law, in* THE BUSINESS GUIDE TO JAPAN 264, 266 (Gerald Paul McAlinn ed., 1996).

[19] *See* Yoshikazu Takaishi, Outline of the Japanese Patent and Utility Model Laws 3, Jan. 21, 1997 (unpublished law school course materials, on file with the author). For example, the *Japan IP Resources* web page reports that the Tokyo High Court recently upheld a 1988 JPO patent grant for a new breed of peach. *See* Japan IP Resources, *News Update from Japanese IP*

1. Utility

To qualify for patent protection, an invention must have some practical industrial application.[27] It need not, however, have such usefulness at the time of filing.[28]

2. Novelty

Patentable inventions must be "novel."[29] Inventions meet the novelty requirement if, prior to application, they are not (i) publicly known in Japan, (ii) publicly used in Japan, or (iii) described in a publication distributed in Japan or any foreign country.[30]

Inventions do not lose their novelty if they have:

a. become publicly known as a result of experimentation, publication, or presentations made in conjunction with academic organizations designated by the Commissioner of the JPO;[31]

b. become publicly known or been exploited in Japan against the inventor's will;[32] or

c. become publicly known as a result of display at exhibitions sanctioned by the Japanese Government or local public organizations.[33]

---

[27] *See id.*

[28] *See id.*

[29] *See id.*

[30] *See id.*

[31] *See id.* art. 30.

[32] *See id.*

[33] *See id.*

1. Filing Of Application

As in most countries, the first party to file an application for a Japanese patent receives priority over subsequent applicants.[41] Thus the U.S. "first-to-invent" rule does not apply in Japan.

Foreign applicants may take advantage of the benefits of the Paris Convention and the Patent Cooperation Treaty.[42] To exercise priority filing rights under the Paris Convention, the applicant must submit the following items:

a. a statement of intent to make use of priority filing rights;

b. a document outlining

   (i) the name of the country where the application was first filed; and

   (ii) the name of the country where the application was deemed to be first filed under Paris Convention Article 4C(4); or

   (iii) the name of the country where the application was recognized as having been first filed under Paris Convention Article 4A(2);

c. a document stating the filing date; and

d. a document supporting the assertion of foreign priority filing (must be submitted within sixteen months of the foreign filing).[43]

---

[41] *See id.* art. 39(1).

[42] For a general explanation of the benefits of foreign priority filing, see generally DONALD S. CHISUM & MICHAEL A. JACOBS, UNDERSTANDING INTELLECTUAL PROPERTY LAW § 2H (1992).

[43] See generally LEC GUIDE, *supra* note 12, at 2.

his application before Gazette publication.[50] Applicants have the right to compensation from infringing parties who used the claimed invention after the initial filing date, provided that the JPO eventually grants the patent.[51]

4. Substantive Examination (On Request)

After the filing, the JPO does not examine patent applications until the applicant or a third party makes a request for examination.[52] An applicant or third party must make such a request within seven years of the initial filing date or the JPO will consider the application withdrawn.[53] After filing, amendments may be made to the application at any time until the end of a short "response period" that follows the primary substantive examination.[54] If a secondary examination is required, further amendments may be made during a similar response period following the secondary examination.[55]

5. Patent Grant

If the patent application meets all the prescribed criteria and the subject invention is deemed to qualify for patent protection, the JPO will grant a patent.[56] The JPO notifies the applicant (or his administrator) by sending a certified copy of its decision to grant.[57] The applicant must then pay registration fees within thirty days after the date that the JPO sends this

[50] *See* LEC GUIDE, *supra* note 12, at 4-5.

[51] *See* LEC GUIDE, *supra* note 12, at 21-22.

[52] *See* Patent Law, *supra* note 11, art. 48(2).

[53] *See id.* art. 48(3)(I).

[54] 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 5-6.

[55] *See id.*

[56] *See* Tsuruya, *supra* note 48.

[57] *See id.*

H. *Parallel Imports*

On July 1, 1997, the Japanese Supreme Court held that a patentee who had sold its product in a foreign market may not enforce the patent against a parallel importer in Japan, even though the patent is valid with respect to the product in the Japanese market.[69] The Court based this conclusion on the exhaustion theory.[70]

The Supreme Court limited the scope of its holding, however, by ruling that where a patent holder sells its product (for example, to distributors in its own country) on condition that the product may not be redistributed in Japan, the exhaustion theory will not apply.[71] Accordingly, a patent holder may reserve the right to enforce the patent in Japan against a parallel importer by entering into such a restrictive distribution agreement.[72]

I. *Remedies For Infringement*

The two primary civil remedies for infringement are injunctive relief and damages. The patent holder may enjoin all unauthorized uses of the patented invention.[73] To receive compensation for damages, the patent holder must prove the amount of damages that resulted from any alleged infringement.[74] Compensatory damages can be calculated in any of three

[69] *See* BBS Kraftfarzeug Technik AG v. Kabushiki Kaisha Racimex Japan, 1198 SAIBANSHO JIHŌ 8 (Sup. Ct., July 1, 1997). For an unofficial translation of this case, see *Japan IP Resources, News Update from Japanese IP Scene: Archives* (visited Mar. 20, 1998) <http://okuyama.com>.

[70] Under the exhaustion (or "first sale") theory, once a patentee makes an authorized sale of a product covered by the patent (i.e., puts the product into a stream of distribution), the patent is deemed "exhausted" and thus may not be enforced against subsequent buyers or lessees. *See generally* CHISUM & JACOBS, *supra* note 42, § 2E[3].

[71] *See generally* BBS Kraftfarzeug Technik AG v. Kabushiki Kaisha Racimex Japan, 1198 SAIBANSHO JIHŌ 8 (Sup. Ct., July 1, 1997).

[72] *Id.*

[73] *See* Patent Law, *supra* note 11, art. 100.

[74] *See id.* art. 102

A. *Recent Changes*

There have been several notable changes in the law regarding utility model rights, including the following:

1. Term

Utility models are now protected for six years from the date of filing.[81] Previously, the protection extended for ten years from the date of publication for opposition.

2. No Substantive Examination

Utility models are registered without undergoing a substantive examination by the JPO.[82]

B. *Subject Matter*

Utility model rights protect the mechanical shape or structure, as well as combinations of mechanical shape and structure, of devices.[83] As such, processes or chemical substances cannot receive protection as utility models.[84]

C. *Statutory Requirements*

The statutory requirements for utility model registration are similar to those for patents—inventive step, novelty, and utility are required.[85] However, the general standards for these elements are lower than those required for patent registration.[86]

---

[81] *See* Jitsuyō shinan hō, [Utility model law] Law No. 123 of 1959, art. 15, as amended [hereinafter Utility Model Law].

[82] *See* Ohba, *supra* note 18, at 268.

[83] *See* Utility Model Law, *supra* note 81, art. 1.

[84] *See* Ohba, *supra* note 18, at 267.

[85] *See id.* at 267.

[86] *See id.*

A. *Subject Matter*

Under Article 2 of the Design Law, a "design" is a shape, pattern, or color (or any combination of these elements) in an article, which produces an aesthetic visual impression.[94] To qualify for registration under this definition a design must (i) be embodied in a tangible article, (ii) be visible to the naked eye, and (iii) be aesthetically impressive.[95]

B. *Statutory Requirements*

The statutory requirements for the registration of design rights are creativity, novelty, and industrial applicability.[96]

1. **Creativity**

Registrable designs must display some degree of creativity. If a design easily could have been created prior to the filing of the design application by a person with ordinary skill in the art to which the design pertains, the design generally is not registrable.[97]

2. **Novelty**

Designs known in Japan or abroad, including designs described in published literature, or designs similar to other designs that are known or described in literature, are not registrable.[98]

---

[94] *See id.* art. 2.

[95] *See id.*

[96] *See id.* art. 3.

[97] *See* LEC GUIDE, *supra* note 12, at 53.

[98] Design Law, *supra* note 92, art. 3.

Japan's trademark system into compliance with internationally accepted standards.

1. "Famous" Foreign Marks

Foreign marks that are well known in Japan or abroad are now protected under Japanese trademark law without registration.[106]

2. International Classification Standards

Japan has adopted the International Classification of Goods and Services standards under the Nice Agreement.[107]

3. Collective Mark System

Procedures for the registration of collective marks have been dramatically simplified.[108]

4. Multi-Class Application

A single trademark application can cover multiple classes of goods—the "single-class rule" no longer applies.[109]

5. Registration Of 3-D Marks

Three-dimensional marks, such as packaging, may be registered as trademarks.[110]

[106] *See* JAPANESE PATENT OFFICE, REVISION OF JAPANESE TRADEMARK LAW IN 1996 at 5 (1996) [hereinafter TRADEMARK LAW REVISION].

[107] Agreement Concerning the International Classification of Goods and Services to Which Trade Marks Apply, *done* June 15, 1957, 550 U.N.T.S. 45.

[108] TRADEMARK LAW REVISION, *supra* note 106, at 5-6.

[109] *See id.* at 2.

[110] *See id.* at 5.

B. *Other Key Points*

In addition to the recent changes listed above, the following features highlight the Japanese trademark/service mark system:

1. First-To-File Rule

The first party to file an application receives trademark registration priority.[117]

2. "Intent-To-Use" Application

Applicants for Japanese trademarks may file based on intent to use; actual prior use is not required for registration.[118]

3. No "Prior Use" Challenges

Challenges to registered trademarks may not be made based on prior use.[119]

4. Parallel Imports

Under Japanese law, the parallel importation of genuine products does not constitute an infringement of trademark rights.[120]

C. *Subject Matter*

Under Japan's revised trademark law, trademarks may be comprised of characters, figures, symbols, three-dimensional objects, or combinations of these elements.[121] Composite marks using colors also qualify for

[117] *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 4.

[118] *See* LEC GUIDE, *supra* note 12, at 45.

[119] *See* Shōhyo hō, [Trademark law] Law No. 127 of 1959, art. 32 [hereinafter Trademark Law]. *See also* THE TRADEMARK LAW AND THE ENFORCEMENT LAW THEREOF, art. 32 (Fukio Nakane trans., Eibun-Horeisha) (1960).

[120] *See infra* Part V.J.

[121] *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 13.

5. marks that are misleading as to the quality of the goods or services.[128]

D. *Statutory Requirements*

The JPO judges trademarks and service marks according to a standard of "distinctiveness."[129] The following types of marks are considered non-distinctive:

1. marks that comprise the common name of the good or service;[130]

2. marks that are customarily used with the good or service;[131]

3. marks that indicate only a quality of the good or service, including place of origin or sale, price, shape, effect, use, quantity, shape or manner or time of manufacture;[132]

4. marks that indicate only a common surname or title in a manner ordinarily used;[133]

5. marks that are extremely simple or common;[134]

---

[128] *Id.*

[129] *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 5.

[130] Trademark Law, *supra* note 119, art. 3(1)(I).

[131] *Id.* art. 3(1)(ii).

[132] *Id.* art. 3(1)(iii).

[133] *Id.* art. 3(1)(iv).

[134] *Id.* art. 3(1)(v).

is not applicable in Japan, and challenges of registered marks cannot be made on the basis of prior use.[142] U.S. applicants should also note that ownership rights in Japanese trademarks and service marks are acquired through registration—they are not acquired automatically through use in trade.[143]

2. Formality Examination

Once an application has been submitted, the JPO does a preliminary formality check to ensure that the application meets the prescribed procedural requirements and to determine if the applicant has submitted all the required documents.[144] When the formality check is complete, the JPO notifies the applicant of any defects discovered in the application.[145] The applicant may make amendments to cure any such defects.[146]

3. Substantive Examination

All applications for trademark or service mark registration undergo a substantive examination by the JPO to determine the mark's distinctiveness, and confirm that the mark does not fall under any of the statutory exceptions.[147] If the JPO determines that a mark is not registrable, it issues a notice of refusal.[148] At this stage, the applicant may amend the application or file a statement indicating its reasons why the mark should

uses a similar mark." CHISUM & JACOBS, *supra* note 42, at § 5D[1].

[142] *See id.*

[143] *See* Trademark Law, *supra* note 119, art. 18.

[144] *See id.* art. 77.

[145] *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 5.

[146] *See id.*

[147] *See id.*

[148] *See id.*

period.[158] If the second installment (covering years six through ten) is not paid within five and one-half years of the initial registration, the registration is deemed canceled.[159] This cancellation is retroactive. Thus, the JPO deems that the registration was extinguished at the end of the first five-year period.[160]

6. Renewal

Registrations can be renewed after the initial ten-year term by filing a renewal request and paying related fees. An application for renewal must be submitted no earlier than six months before the end of a given term or six months after the termination date.[161] An owner no longer must submit evidence of continued use when applying for renewal.[162]

G. *Post-Grant Opposition*

In the past, the JPO published all applications that it deemed registrable to allow opposition before the actual grant of registration. The JPO has replaced this pre-grant "publication system" with a post-grant opposition system designed to expedite the registration process.[163] After the granting of a trademark or service mark, the JPO publishes details of the mark in the official Trademark Gazette.[164] Any party who objects to the registration of a trademark or service mark may file an opposition within two months of the mark's publication in the Gazette.[165] The opposition should be addressed to the Commissioner of the JPO.[166]

---

158 *See id.*

159 *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 5.

160 *See id.*

161 *See id.*

162 *See generally id.* at 8.

163 *See id.* at 6.

164 *See* Trademark Law, *supra* note 119, art. 43*bis*-43*quater*.

165 *See* TRADEMARK SYSTEM OUTLINE, *supra* note 112, at 6.

166 *See id.*

imprisonment or fines.[177] Recent revisions to the Trademark Law impose heavier fines on legal entities that are found guilty of infringement.[178]

I. *Unused Trademarks*

The 1996 revisions of the Trademark Law introduced a number of measures designed to reduce the number of unused marks. Under the revised law, any party may petition for the cancellation of a trademark.[179] Formerly such petitions were restricted to parties having an interest in the proceedings. In addition, trademark owners who have not used a mark for three years may not make last-minute uses to defend against a cancellation action.[180] If a mark has not been used for three years, the JPO will not recognize any use of the mark within three months of the date the cancellation action is filed.[181]

Finally, the new provisions allow for retroactive cancellation of unused marks. If a cancellation action succeeds, the JPO deems the cancellation effective from the filing date of the action.[182]

J. *Parallel Import Defense*

The parallel importation and domestic sale of genuine products bearing trademarks registered in Japan does not constitute an infringement of Japanese trademark rights. Accordingly, parallel importation is available as an affirmative defense in trademark infringement actions.[183]

---

[177] *See* LEC GUIDE, *supra* note 12, at 50.

[178] *See* Trademark Law, *supra* note 119, art. 82.

[179] *See* TRADEMARK LAW REVISION, *supra* note 106, at 3.

[180] *See id.* at 4.

[181] *See id.* at 3.

[182] *See id.*

[183] *See generally* Misao Toba, *Latest Developments in Japanese IP Cases*, 22 BIMONTHLY J. OF INT'L ASS'N FOR THE PROTECTION OF THE INDUS. PROP. OF JAPAN 14 (Jan. 1997) (describing Y.K. Marukatsu v. K.K. Minibox, Osaka District Court (Sept. 28, 1995)).

or where Japanese consumers associate the mark only with the senior user in Japan and not with the product's original manufacturer or source.[189]

VI. REGISTERED TRADE AND CORPORATE NAMES

Trade names and corporate names may be registered in local company registers. Locally registered names are protected under provisions of the Commercial Code[190] and under the Unfair Competition Prevention Law.[191] To qualify for local registration, a trade name cannot be the same as other trade names that are used by entities in the same line of business who have registered the name in the same city, town, or village.[192]

VII. COPYRIGHT AND RELATED RIGHTS

A. *Recent Changes*

The Japanese Copyright Law recently has undergone a number of changes and is continuing to evolve in order to bring Japan's copyright system into compliance with various international conventions. Notable changes have been made in the following areas:

1. Neighboring Rights Duration

Neighboring rights are now protected for fifty years from the first fixation, performance, or broadcast.[193]

[189] *See id.*

[190] SHŌHŌ [Commercial code] arts. 20-22.

[191] LEC GUIDE, *supra* note 12, at 9.

[192] *See id.*

[193] Chosakuken hō, [Copyright law] Law No. 48 of 1970, art. 101, as amended [hereinafter Copyright Law]. A translation of the Copyright Law can be found in COPYRIGHT RESEARCH AND INFORMATION CENTER, COPYRIGHT LAW OF JAPAN (Yukifusa Oyama et al. trans., 1997).

C. *Subject Matter*

Under Japan's Copyright Law, a copyrightable "work" is defined as "a production in which thoughts or sentiments are expressed in a creative way and which falls in the literary, scientific, artistic or musical domain."[199]

The following types of works qualify for copyright protection: literary works;[200] musical works;[201] choreographic works and pantomime;[202] works of fine art, such as paintings or sculptures;[203] architectural works;[204] maps and figurative works of a scientific nature;[205] cinematic works;[206] photographic works;[207] and computer programs.[208] Other items that qualify for protection under the Copyright Law include compilations that, by reason of the selection or arrangement of their content, constitute intellectual creations;[209] and databases that, by reason of the selection or systematic construction of information contained therein, constitute intellectual creations.[210]

Most official works of the Japanese government, including the Japanese Constitution, laws, ordinances, judgements, and administrative

---

199 *See id.* art. 2(1)(I).

200 *See id.* art. 10(1)(I).

201 *See id.* art. 10(1)(ii).

202 *See id.* art. 10(1)(iii)

203 *See id.* art. 10(1)(iv).

204 *See id.* art. 10(1)(v).

205 *See id.* art. 10(1)(vi).

206 *See id.* art. 10(1)(vii).

207 *See id.* art. 10(1)(viii)

208 *See id.* art. 10(1)(ix).

209 *See id.* art. 12.

210 *See id.* art. 12*bis*.

E. *Protection Of Foreign Works*

Foreign works receive protection under the rules and protocols of the Berne Convention, the Universal Copyright Convention, WTO conventions, and WIPO conventions.[217]

F. *Scope Of Protection*

The scope of protection of copyrights, moral rights, and neighboring rights is as follows.

1. **Author's Rights**

The original author (or assignee/licensee) of a copyrighted work has exclusive rights to reproduce the work,[218] publicly perform the work,[219] broadcast the work or transmit it by wire,[220] publicly recite the work,[221] publicly exhibit the work,[222] publicly screen or distribute a cinematic work,[223] lend copies of original works,[224] and translate or adapt a work.[225] If an individual authors the copyrighted work, the duration of copyright begins with the work's creation and continues for the life of the author plus fifty years.[226] For works of joint authorship, the copyright endures for the life of

[217] *See* COPYRIGHT RESEARCH AND INFORMATION CENTER, COPYRIGHT LAW OF JAPAN viii (Yukifusa Oyama et al. trans., 1997).

[218] *See* Copyright Law, *supra* note 193, art. 21.

[219] *See id.* art. 22.

[220] *See id.* art. 23.

[221] *See id.* art. 24.

[222] *See id.* art. 25.

[223] *See id.* art. 26.

[224] *See id.* art. 26*bis*.

[225] *See id.* art. 27.

[226] *See id.* art. 51.

performance, the right to produce phonograms, the right of broadcasting, and the right of transmitting via wire.[236] In 1997, the act of "making transmittable" was added to the right of transmission.[237] The duration of neighboring rights is fifty years from the end of the year in which the work was first fixed in a tangible medium (for phonograms), performed, broadcasted, or transmitted via wire.[238]

4. Berne Convention Rights

The duration of copyright protection for foreign works originating in Berne Convention countries is reciprocal with each member country, up to fifty years.[239]

G. *Exceptions*

Although the author is afforded "exclusive" rights under Japanese copyright law, these rights are subject to a number of exceptions. The following acts are permitted with limited exemption from copyright liability:

1. reproduction of a work for private use;[240]

2. reproduction of a work in library materials;[241]

3. quotation of a published work;[242]

---

[236] *See id.* arts. 89-100*quater*.

[237] Copyright Research and Information Center, *Copyright System in Japan* (visited Mar. 20, 1998) <http://www.japanlink.co.jp/cric/cric_e/ecsij/csij.html>.

[238] See Copyright Law, *supra* note 193, art. 101.

[239] *See id.* art. 58.

[240] *Id.* art. 30.

[241] *Id.* art. 31.

[242] *Id.* art. 32.

broadcasting or otherwise, for the purposes of reporting current events;[251]

13. reproduction of a work for use in judicial proceedings;[252]

14. ephemeral recordings of a work by broadcasters who have permission to broadcast the work;[253]

15. exploitation of a work permanently displayed outdoors in a public place;[254] and

16. reproduction required for a public exhibition of a work (for example, in the production of promotional pamphlets).[255]

The Japanese Copyright Law does not recognize the type of abstract "fair use" defense available in the United States. It limits fair use to the acts specifically enumerated above.[256]

[251] *Id.* art. 41.

[252] *Id.* art. 42.

[253] *Id.* art. 44.

[254] *Id.* art. 46. Article 46 explicitly disallows the multiplication of a sculpture, imitative reproduction of architectural works, the reproduction of a work for the purpose of locating it permanently in public places, and the reproduction of a work for the purpose of selling copies. *Id.*

[255] *Id.* art. 47.

[256] *See* Yoshikazu Takaishi, Outline of Japanese Copyright Law 7, Mar. 4, 1997 (unpublished law school course materials, on file with the author).

rights, by a person who is aware of such infringement;[263] and

3. the use of a copy of a computer program on a computer in the conduct of business, where such copy has been produced by an act of infringement (and where the user is aware of the infringement).[264]

J. *Remedies For Infringement*

Authors and registered exclusive licensees may enjoin unauthorized use of a copyrighted work.[265] Authors and registered exclusive licensees also may seek compensatory damages from infringing parties.[266] As noted above, Japanese courts do not award punitive damages, even in cases of willful infringement.[267] Article 115 of the Copyright Law allows for measures to restore the "honor" of the author in cases where infringement upon the author's moral rights has occurred.[268] Such measures include identification of the author and correction of distortion, mutilations, or modifications to the original work.[269] Finally, in addition to civil liability, infringing parties may be subject to criminal penalties consisting of imprisonment for up to three years or fines not to exceed three million yen.[270]

---

[263] *Id.* art. 113(1)(ii).

[264] *Id.* art. 113(2).

[265] *See id.* art. 112(1).

[266] *See* MINPÔ [Civil code] art. 709.

[267] *See supra* text accompanying note 76.

[268] *See* Copyright Law, *supra* note 193, art. 115.

[269] *See id.* art. 115.

[270] *See id.* art. 119. The maximum fine has recently been increased to three million yen. *See* Copyright Research and Information Center, *Copyright System in Japan: IV. Measures Against Infringement* (visited Mar. 20, 1998). <http://www.japanlink.co.jp/cric/cric_e/ecsii/csii.html>.

C. *Infringement*

Under the Unfair Competition Prevention Law, any of the following acts constitutes infringement of trade secret rights:

1. the acquisition of a trade secret by unfair means, including theft, fraud, or coercion; or the use or disclosure of a trade secret acquired by such means;[276]

2. the acquisition, use, or disclosure of a trade secret that has been unfairly acquired, including acquisition, use, or disclosure by third parties who had notice of the illegal acquisition or whose lack of notice was a result of gross negligence;[277]

3. use or disclosure of a trade secret by a third party who initially acquired the trade secret without notice of its confidential nature (bona fide acquisition), but who subsequently received notice of such confidentiality, or whose lack of notice was a result of gross negligence;[278]

4. use or disclosure of a trade secret that has been disclosed by its proprietor, for the purpose of unfair business competition or otherwise acquiring an

Holly Emrick Svetz, *Japan's New Trade Secret Law: We Asked For It — Now What Have we Got?*, 26 GEO. WASH. J. INT'L L. & ECON. 413, 427-28.

[276] Unfair Competition Prevention Law, *supra* note 124, art. 2(1)(iv).

[277] *Id.* art. 2(1)(v).

[278] *Id.* art. 2(1)(vi).

parties.[285] As noted above, Japanese courts do not award punitive damages, even in cases of willful infringement.[286]

If an infringing party acquired trade secrets through criminal means, criminal penalties may apply.[287] Additionally, disclosure of trade secrets by certain fiduciaries also may result in criminal liability.[288]

IX. COMPUTER PROGRAMS AND DATABASES

Traditionally, computer programs have received protection as confidential trade secrets under contract law.[289] Recent revisions in the Unfair Competition Prevention Law have reinforced this protection.[290] Moreover, programs and databases may be protected under Japanese copyright law.

A. *Copyright Protection Of Programs*

Under the Copyright Law, computer "program works" may be protected as "works of authorship."[291] For copyright purposes, a "program" is defined as "an expression of combined instructions given to a computer so as to make it function and obtain results."[292] In theory, only source code

[285] *Id.* art. 4.

[286] *See supra* text accompanying note 76.

[287] *See* KEIHŌ, art. 253 (embezzlement) and art. 247 (breach of confidence). Article 13 of the Unfair Competition Prevention Law provides for imprisonment of up to three years or a fine of up to three million yen. Unfair Competition Prevention Law, *supra* note 124, art. 13.

[288] *See* KEIHŌ art. 134.

[289] *See* Ohba, *supra* note 18, at 277.

[290] *See id.* at 276.

[291] *See* Copyright Law, *supra* note 193, art. 10(1)(ix). For a detailed discussion of copyright protection of software in Japan, see generally Kensuke Norichika, *Copyright Protection of Software and Related Inventions, in* JAPANESE PATENT PRACTICE, *supra* note 24, at 437.

[292] *See* Copyright Law, *supra* note 193, art. 2(1)(xbis).

software licenses or assignments against third parties.[300] Registration provides evidence of the completion date,[301] the author's identity,[302] and the program's contents.[303]

D. *Copyright Protection Of Databases*

Japanese copyright law protects databases as "compilation works."[304] For copyright purposes, databases are "collection[s] of theses, numerical figures, drawings and other pieces of information organized systematically so that these pieces of information can be searched by the aid of a computer."[305] To qualify for protection, databases must exhibit a degree of creativity in the selection and systematic arrangement of the constituent data.[306]

X. IC LAYOUT (MASK WORKS)

The Law Concerning Semiconductor Integrated Circuits protects the layout of integrated circuits in semiconductors ("mask works").[307] Upon registration, the mask work owner has exclusive rights to use the mask work

---

[300] *See* Ohba, *supra* note 18, at 276.

[301] *See* Copyright Law, *supra* note 193, art. 76*bis*.

[302] *See id*. art. 75.

[303] *See* Purogrammu no chosakubutsu ni kakawaru tōroku no tokurei ni kansuru hō [Law on exceptional provisions for the registration of program works], Law No. 65 of May 23, 1986, art. 3.

[304] Copyright Law, *supra* note 193, art. 12*bis*.

[305] *Id*. art. 2(1)(*xter*).

[306] *See id*. art. 12*bis*.

[307] *See* Handōtai shusekikairo no kairohaichi ni kansuru hōritsu [Law concerning the circuit layout of semiconductor circuits], Law No. 43 of 1985 [hereinafter Semiconductor Law].

for ten years.[308] Under the TRIPS agreement,[309] Japan also prohibits the importation, selling, or commercial distribution of protected layout designs, integrated circuits that incorporate such layout designs, and articles that incorporate the integrated circuits.[310] The holder of circuit layout rights may request enjoinment as well as the destruction of infringing circuits, items containing the infringing circuits, or items used in performing the infringing act (such as equipment used in integrated circuit production).[311] Plaintiffs may also seek compensatory damages.[312] Further, the Semiconductor Law contains penal provisions that provide for imprisonment of up to three years or a fine of up to three million yen.[313]

[308] *Id.* art. 10(1), (2).

[309] Agreement on Trade-related Aspects of Intellectual Property Rights Including Trade in Counterfeit Goods, *supra* note 5.

[310] *See id.* sec. 6.

[311] *See* Semiconductor Law, *supra* note 307, art. 22.

[312] *See id.* art. 27.

[313] *See id.* art. 51.

qualifies for copyright protection in Japan. As any unauthorized copying of object code equates with copying of the original source code, however, both types of code are protected in practice.[293] Japanese law explicitly excludes programming languages, rules related to the use of programs, and programming algorithms from copyright protection.[294]

B. *Reverse Engineering*

As a general principle, Japanese law does not permit copying programs via reverse engineering.[295] Reverse engineering is permitted, however, if it is necessary in order to make two systems interoperable.[296] This exception requires that the party engaging in reverse engineering must own a legitimate copy of the program, must not disclose the program's interface information, and must not use reverse engineering to create a *new* program.[297] In addition, Japanese law permits reverse engineering if it is accomplished without making a copy of the subject program.[298]

C. *Registration Of Programs*

In 1986, Japan's Agency for Cultural Affairs, which also oversees copyright registration, established a distinct system for the registration of computer programs. This system is administered by the Software Information Center ("SOFTIC").[299] Although it is not necessary to register programs to receive copyright protection, registration does provide the owner with a stronger evidentiary foundation when it asserts ownership of

[293] Interview with Yoshikazu Takaishi, Attorney-at-Law, in Tokyo, Japan (Mar. 4, 1997).

[294] Copyright Law, *supra* note 193, art. 10(3).

[295] Interview with Yoshikazu Takaishi, Attorney-at-Law, in Tokyo, Japan (Mar. 4, 1997).

[296] *See id.*

[297] *See id.*

[298] *See id.*

[299] *See* LEC GUIDE, *supra* note 12, at 71.

unfair benefit, or for the purpose of injuring the proprietor;[279]

5. the acquisition of a trade secret by a party who has notice that there has been an unfair disclosure of such trade secret, or whose lack of notice was a result of gross negligence;[280] and

6. use or disclosure of a trade secret by a party who has notice that there has been an unfair disclosure of such trade secret, or whose lack of notice was a result of gross negligence.[281]

D. *Remedies For Infringement*

Recent revisions of the Unfair Competition Prevention Law allow proprietors or holders of trade secrets to enjoin infringing parties from disclosing or using the secret information.[282] Japanese courts may also issue injunctions to prevent use or disclosure by third parties who acquired the secret information with notice that it initially was obtained unfairly, or who lacked notice of the infringement as a result of gross negligence.[283] Japanese courts will not enjoin third parties that make bona fide acquisitions of secret information in the ordinary course of business.[284] Proprietors or holders of trade secrets also may seek compensatory damages from infringing

[279] *Id.* art. 2(1)(vii).

[280] *Id.* art. 2(1)(viii).

[281] *Id.* art. 2(1)(x).

[282] *See id.* art. 3.

[283] *See id.* art. 2(1)(vii), (viii).

[284] *See id.* art. 11(6). The plaintiff may also request the destruction of objects used in the act of infringement and objects created through the infringement. *See id.* art. 3(2).

VIII. TRADE SECRETS AND TECHNICAL KNOW-HOW

Although a certain degree of protection for trade secrets and technical know-how has been available under contract law, tort law, copyright law, and criminal statutes, before 1990 there was no specific body of Japanese law that applied to trade secrets.[271] Recent amendments to Japan's Unfair Competition Prevention Law provide specific protection for trade secrets and technical know-how.

A. *Subject Matter*

Under Japanese law, a trade secret may include information related to manufacturing or sales methods, as well as other types of confidential technical or business "know-how."[272] Japanese courts have defined technical know-how as "a conglomerate of technical experiences, knowledge and insight required in manufacturing processes, etc., existing in a tangible or intangible form, which is kept as confidential."[273] Japanese trade secrets cases generally involve technical know-how, and the two terms "trade secret" and "know-how" often are used synonymously in the Japanese language.[274]

B. *Statutory Requirements*

To qualify for trade secret protection, the information must be useful in a commercial activity, the confidentiality of the information must be maintained, and the general public must not have knowledge of the information.[275]

[271] *See* Yoshikazu Takaishi, Outline of the Japanese Trade Secret Law 1, Jan. 21, 1997 (unpublished law school course materials, on file with the author).

[272] *See generally* LEC GUIDE, *supra* note 12, at 62-63.

[273] Deutsch Werft A.G. v. Chuetsu Waukesha, 17 KAMINSHŪ 769 (Tokyo High Ct., Sept. 5, 1966).

[274] Takaishi, *supra* note 271, at 2.

[275] *See* LEC GUIDE, *supra* note 12, at 62-63; *see also* Ohba, *supra* note 18, at 275. The secrecy requirement places a duty on the proprietor of a trade secret to take affirmative actions to protect the secrecy of the information. Japanese courts have required plaintiff companies to show that adequate security measures were taken before finding infringement. *See generally*

H. *Registration*

Although it is not required for copyright protection, registration with Japan's Agency of Cultural Affairs does provide a stronger evidentiary foundation on which to assert claims arising from copyright licenses or assignments against third parties.[257] Registration provides presumptive evidence of the author's identity, regardless of copyright ownership;[258] the date of first publication;[259] the date of the work's creation;[260] and the date of copyright registration.[261]

I. *Infringement*

Under Article 113 of the Copyright Law, the following acts constitute copyright infringement:

1. the importation into Japan, for distribution, of objects made by an act that would constitute an infringement on moral rights, copyright, the right of publication, or neighboring rights, if such objects were made in Japan at the time of such importation;[262]

2. the distribution or possession for distribution of objects made by an act infringing moral rights, copyright, the right of publication or neighboring

[257] *See* Copyright Law, *supra* note 193, art. 77.

[258] *See id.* art. 75.

[259] *See id.* art. 76.

[260] *See id.* art. 76*bis*.

[261] *See id.* art. 76.

[262] *Id.* art. 113(1)(I).

4. reproduction of a work in school textbooks;[243]

5. broadcasting of a work in educational programs;[244]

6. reproduction of a work in schools and other educational institutions for use in teaching materials;[245]

7. reproduction of a work in examination questions (only by nonprofit entities);[246]

8. reproduction of a work in braille;[247]

9. non-profit performance of a work;[248]

10. reproduction by the press of articles on current topics;[249]

11. exploitation of political speeches and speeches delivered in the course of judicial proceedings;[250]

12. reproduction of a work by means of photography, cinematography,

---

[243] *Id.* art. 33.

[244] *Id.* art. 34.

[245] *Id.* art. 35.

[246] *Id.* art. 36.

[247] *Id.* art. 37.

[248] *Id.* art. 38.

[249] *Id.* art. 39.

[250] *Id.* art. 40.

the last joint author to die plus fifty years.[227] In the case of anonymous or pseudonymous works, works authored by corporate entities, and cinematic works, the copyright continues for fifty years from the date that the work is first made public.[228]

2. Moral Rights

Authors also are provided certain inalienable "moral rights" in their copyrighted works. These rights include the right to make the work public,[229] the right to claim authorship of the work ("right of attribution"),[230] and the right to preserve the work's integrity.[231] Certain moral rights continue to exist even after the death of the author.[232] If a party offers or makes available the author's work to the public and commits an act that would infringe on the author's moral rights were the author alive, the author's family may bring an action against the infringing party.[233] Nevertheless, the Copyright Law does permit potentially infringing acts that are deemed not to contravene the will of the author.[234] In making this determination, consideration is given to the nature of the allegedly infringing act and the social context.[235]

3. Neighboring Rights

In addition to author's rights, Japan's Copyright Law also provides performers, producers of phonograms, broadcasters, and "wire diffusion organizations" with protection for neighboring rights, including the right of

[227] *See id.* art. 51.

[228] *See id.* art. 52.

[229] *See id.* art. 18.

[230] *See id.* art. 19.

[231] *See id.*

[232] *See id.* art. 60.

[233] *See id.* arts. 60, 116.

[234] *See id.* art. 60.

[235] *See id.*

notices, do not qualify for copyright protection.[211] Computer programming languages, including the rules or algorithms that comprise such languages, also do not gain protection under the Japanese Copyright Law.[212]

D. *Protected Works*

Under Japanese law, qualifying works automatically come under copyright protection — registration is optional.[213] Article 6 of the Copyright Law provides protection for the following:

1. works of Japanese nationals, including legal entities established under Japanese law and businesses having their principal offices in Japan;[214]

2. works first published in Japan, including works first published abroad and subsequently published in Japan within 30 days of the initial publication;[215] and

3. foreign works subject to protection under international copyright treaties to which Japan is a party.[216]

---

[211] *See id*. art. 13.

[212] *See id*. art. 10(3).

[213] *See* LEC GUIDE, *supra* note 12, at 36.

[214] Copyright Law, *supra* note 193, art. 6(I).

[215] *Id*. art. 6(ii).

[216] *Id*. art. 6(iii).

2. Sound Recordings

Protection of sound recordings has been extended to include those registered since 1946.[194]

3. Photographs

The activation date for copyright protection afforded to photographs has been changed to the day of the copyright holder's death.[195]

4. Right Of Transmission

The act of "making transmittable" has been added to the neighboring right of transmission.[196]

B. *Other Key Points*

Besides the above-mentioned changes, two other aspects of the Japanese copyright system are particularly noteworthy:

1. International Treaties

Foreign works are protected under the Berne Convention and other international treaties and conventions.[197]

2. Moral And Neighboring Rights

In addition to copyright protection, Japan offers protection of moral and neighboring rights.[198]

---

[194] *See* Supplementary Provisions to the Copyright Law, Law No. 112 of 1994.

[195] *See* Supplementary Provisions to the Copyright Law, Law No. 117 of 1996, art. 2.

[196] *See* Supplementary Provisions to the Copyright Law, Law No. 86 of 1997, art. 2.

[197] *See* Copyright Law, *supra* note 193, at viii.

[198] *See id.* arts. 19-20, 89-100*quater*.

Under Japanese law, gray market imports are considered to be "genuine products" when all of the following elements apply:

1. The products legally entered the original market bearing the same mark.

2. The party who uses the mark in Japan, for example, a senior user such as an exclusive distributor, is the party who originally placed the products on the market or is closely related to such party.

3. The quality of the products is substantially the same as the original.[184]

When all of the above conditions apply, the origin and warranty of the parallel products are identical to those of the original products of the senior user, and the mark's function is considered not to be adversely affected in any way.[185] Consequently, the importation and sale of genuine products is acceptable and does not constitute an infringement or unfair trade practice.[186]

This defense may not be used if the senior user has improved or added value to the products in some way, because the source and quality of the products then differs from those produced in the country of origin.[187] Additionally, it may not be used where there is no close relationship between the mark's senior user in Japan and the product's original source,[188]

[184] *Id.*

[185] *See id.*

[186] *See id.*

[187] *See id.*

[188] *See id.*

A board of JPO trial examiners examines all oppositions.[167] Where the board finds grounds for the opposition, it notifies the trademark owner of the reason for possible cancellation of the trademark registration.[168] The trademark owner then has a chance to provide a written response to the opposition.[169] If this written response does not overcome the grounds for opposition, the JPO may cancel the registration.[170] In other cases, the JPO may restrict the mark's use.[171] If the owner of a mark has lost its rights through JPO post-grant opposition proceedings, it may file an appeal with the Tokyo High Court.[172]

H. *Remedies For Infringement*

Trademark owners and exclusive licensees may enjoin unauthorized use of a confusingly similar mark that is used for articles or services in the same class.[173] Trademark owners and registered exclusive licensees may also seek compensatory damages from infringing parties.[174] As noted above,[175] Japanese courts do not award punitive damages, even in cases of willful infringement. Under Article 39 of the Trademark Law, however, infringing parties may be required to restore the business reputation of a trademark owner if it has been damaged as a result of infringement.[176] Finally, infringing parties may be subject to criminal penalties consisting of

[167] *See id.*

[168] *See id.*

[169] *See id.*

[170] *See id.*

[171] *See id.*

[172] *See id.*

[173] *See* Trademark Law, *supra* note 119, art. 36.

[174] *See id.* art. 38.

[175] *See supra* text accompanying note 76.

[176] *See* Trademark Law, *supra* note 119, art. 39.

qualify for registration.[149] If the applicant does not overcome the JPO's reasons for refusal, the JPO refuses registration.[150]

4. Notification Of "Decision Of Registration"

If the JPO examiner determines that a trademark or service mark qualifies for registration, including cases where the applicant has overcome an initial notice of refusal through an amendment or statement of opinion, the JPO will notify the applicant of its decision.[151]

5. Payment Of Registration Fees

To complete the registration process, the applicant must pay applicable registration fees to the JPO.[152] The applicant may pay in cash.[153] The JPO no longer requires the use of patent revenue stamps.[154]

Registration fees can be paid in one lump-sum payment (covering the full ten-year term of registration) or in two installments (covering the first half and second half of the ten-year term, respectively).[155] Under the installment option, the trademark owner has five years to make a determination regarding the mark's maintenance.[156] No installment charges apply if the owner pays the second installment within five years of the end of the ten-year registration term.[157] However, the JPO assesses supplementary fees if the owner pays the second installment during a six-month "grace period" that begins to toll at the end of the first five-year

[149] *See id.*

[150] *See id.*

[151] *See id.*

[152] *See* Trademark Law, *supra* note 119, arts. 40-43.

[153] *See id.* art. 40.

[154] *See id.*

[155] *See id.* art. 41*bis*.

[156] *See id.*

[157] *See id.* art. 41*bis*.

6. marks by which consumers are unable to identify the source of the product or service.[135]

Continued use of a trademark or service mark may be necessary to maintain the registration. The JPO may invalidate trademarks or service marks that have not been used for more than three years.[136]

E. *Scope Of Protection*

Owners of registered trademarks or service marks have the exclusive right to use such marks for the designated goods or services.[137] Trademark and service mark registrations are valid for ten years from the date of initial registration.[138] Registrations may be renewed for additional ten-year periods.[139]

F. *Application And Examination Procedures*

To qualify for protection under the Trademark Law, marks must be registered with the JPO by observing the following procedures.

1. Filing Of Application

As in most countries, the first party to file an application for a Japanese trademark or service mark is given priority over subsequent applicants.[140] Thus the "first-to-use"[141] principle adopted in the United States

[135] *Id.* art. 3(1)(vi).

[136] *See id.* art. 19(2).

[137] *See id.* art. 25.

[138] *See id.* art. 18-19.

[139] *See id.* art. 19(2).

[140] *See id.* art. 8(1).

[141] "A fundamental common law tenet is that trademark property rights arise from . . . actual adoption and use of a mark to distinguish . . . goods from those of others. Adoption and use creates mark ownership rights and confers a priority right against anyone who subsequently adopts and

registration.[122] Marks comprised of sounds or smells do not qualify for registration.[123] The following may not be registered as trademarks or service marks even if the mark meets the requirement of distinctiveness:

1. marks that are identical or similar to national flags, symbols of local governments or other international or domestic public organs, and other similar marks;[124]

2. marks that are identical or similar to registered marks owned by other parties;[125]

3. marks that are identical or similar to well-known or famous trademarks owned by other parties;[126]

4. marks containing a name or stage name of another person, or a famous abbreviation of such a name or stage name;[127] and

[122] *See id.*

[123] *See id.*

[124] Trademark Law, *supra* note 118, art. 4. Under recent revisions of the Trademark Law, "[p]rotection of the state coat of arms or other emblems and official seal or sign is extended to the ones of Contracting Parties of the Trademark Law Treaty." TRADEMARK LAW REVISION, *supra* note 106, at 3; *see also* Fusei kyōsō bōshi hō [Unfair competition prevention law], Law No. 47 of 1993, art. 9 [hereinafter Unfair Competition Prevention Law]. A translation of the Unfair Competition Prevention Law can be found in DOING BUSINESS IN JAPAN, Statutory Material App. 9D (Zentaro Kitagawa ed., 1997).

[125] Trademark Law, *supra* note 119, art. 4.

[126] *Id.*

[127] *Id.*

6. Six-Month Grace Period

Application for renewal of trademarks or service marks can be filed up to six months after the expiration of the registration term.[111]

7. Post-Grant Opposition

The JPO has adopted a post-grant opposition system to replace the pre-grant publication and opposition system.[112]

8. Standard Character Registration

It is not necessary to attach a trademark specimen when registering a mark consisting of standard, nonstylized characters.[113]

9. Payment Of Fees

Registration fees may be paid in two installments.[114] In addition to payment via patent revenue stamps, registration fees now may be paid in cash.[115]

10. Measures To Reduce Unused Marks

Recent revisions to the Trademark Law have introduced several measures to reduce the number of unused trademarks in Japan.[116]

---

111 *See id.* at 3.

112 *See* JAPANESE PATENT OFFICE, OUTLINE OF JAPANESE TRADEMARK SYSTEM: GUIDE TO TRADEMARK LAW REVISED IN 1996, at 6 (1996) [hereinafter TRADEMARK SYSTEM OUTLINE].

113 *See id.* at 11.

114 *See id.* at 5.

115 *See* TRADEMARK LAW REVISION, *supra* note 106, at 6.

116 *See infra* Part V.I.

3. Industrial Applicability

Designs must be incorporated in articles that can be mass-produced.[99] Accordingly, "one-off" designs do not qualify for registration.[100]

C. *Scope Of Protection*

Registered designs are protected for fifteen years from the date of registration.[101] The registered owner or registered licensee of a design has the right to its exclusive commercial use, and may enjoin any unauthorized use of both the registered design and confusingly similar designs.[102] The owner or licensee may also seek compensatory damages from infringing parties.[103]

D. *Application And Examination Procedures*

The procedures for design registration generally are similar to those that apply to patent registration.[104] The most significant difference is that an applicant for design registration may request that the design be kept secret (unpublished) for up to three years from the date of registration.[105]

V. TRADEMARK/SERVICE MARK RIGHTS

A. *Recent Changes*

Japanese trademark law recently has undergone a number of important revisions designed to expedite trademark registration and bring

---

[99] *See* LEC GUIDE, *supra* note 12, at 53.

[100] Interview with Yoshikazu Takaishi, Attorney-at-Law, in Tokyo, Japan (Jan. 21, 1997).

[101] Design Law, *supra* note 92, art. 21.

[102] *See id*. art. 23.

[103] *See id*. art. 39.

[104] *See generally supra* Section II.F.

[105] *See* Design Law, *supra* note 92, art. 14(1).

D. *Scope Of Protection*

In contrast to patents, which are protected for twenty years, utility models registered after 1993 are protected for six years from the date of filing.[87] Utility models registered before 1993 are protected for ten years from the date of publication for opposition or fifteen years from the initial filing date, whichever occurs first.[88]

E. *Application And Examination Procedures*

The procedures for utility model registration are similar to those that apply to patent registration.[89] The most significant difference is that the JPO does not perform a substantive examination of utility model applications.[90] An applicant must provide a written evaluation of the technology to be registered; the JPO examiner makes a validity determination based on this evaluation.[91]

IV. DESIGN RIGHTS

Under the Design Law, visually aesthetic designs that have commercial application may be registered with the JPO.[92] Applicants for design registration may request that the JPO protect the design's secrecy for up to three years from the date of registration.[93]

[87] *See* Utility Model Law, *supra* note 81, art. 15.

[88] *See* Ohba, *supra* note 18, at 268.

[89] *See generally supra* Section II.F.

[90] *See* Ohba, *supra* note 18, at 268.

[91] *See* Japanese Patent Office, *FAQ: Q4* (visited March 30, 1998) <http://www.jpo-miti.go.jp>.

[92] Ishō hō, [Design law] Law No. 125 of 1959, arts. 2-3, as amended [hereinafter Design Law].

[93] *See id.* art. 14(1).

ways: lost profits, actual profit gained by the infringing party, or reasonable compensation (equivalent to unearned royalties).[75] Japanese courts do not award punitive damages, even for cases of willful infringement.[76] Further, due to limitations on discovery in Japan, it is often difficult to ascertain actual profits earned by infringing parties.[77]

In addition to a possible injunction and damage award, Article 106 of the Patent Law allows for measures to restore the business reputation of a patent holder where it has been damaged as a result of infringement.[78] Such measures typically include public apologies by the infringing party.[79] Finally, infringing parties may be subject to criminal penalties consisting of imprisonment up to five years or fines not to exceed 500,000 yen.[80]

III. UTILITY MODEL RIGHTS

The JPO maintains a separate registration system for utility models. The procedures and rights under this system are similar to the patent system, with some exceptions.

[75] *See* Ohba, *supra* note 18, at 267.

[76] *See* Nobutoshi Yamanouchi & Samuel J. Cohen, *Understanding the Incidence of Litigation in Japan: A Structural Analysis*, 25 INT'L LAW 443, 452 (1991). Also, in a 1997 holding the Japanese Supreme Court refused to enforce punitive damages awarded by a foreign court. Northcon v. Mansei Kogyou, 1199 SAIBANSHO JIHŌ 3 (Sup. Ct., July 11, 1997).

[77] *See generally* Mark A. Behrens & Daniel H. Raddock, *Japan's New Product Liability Law: The Citadel Of Strict Liability Falls, But Access To Recovery Is Limited By Formidable Barriers*, 16 U. PA. J. INT'L. BUS. L. 669, 706-08 (1995).

[78] *See* Patent Law, *supra* note 11, art. 106.

[79] *See generally* Yamanouchi & Cohen, *supra* note 76, at 452.

[80] *See* Patent Law, *supra* note 11, art. 196. For a detailed discussion of available remedies, see generally Kazuko Matsuo, *Remedies Against Infringement and Possible Defenses*, *in* JAPANESE PATENT PRACTICE, *supra* note 24, at 369.

notice.[58] Patent rights take effect upon the payment of annual fees for the first three years of registration.[59]

G. *Post-Grant Opposition And Invalidity Actions*

Under recent revisions to the Patent Law, any party may raise post-grant opposition within six months of public notice of the granting of the patent.[60] The JPO has primary jurisdiction over post-grant opposition actions, which are adjudicated by a tribunal of appeal examiners.[61] Examiners may, on their own initiative, examine reasons for revocation other than those raised by the opposing party.[62] Proper grounds for opposition include allegations that the claimed invention is not novel, that the claimed invention does not involve an inventive step, or that the description of the claimed invention is inadequate.[63]

Patent owners who lose their patent rights as a result of opposition actions may appeal to the Tokyo High Court.[64] The Supreme Court hears subsequent appeals.[65] Opposing parties may not appeal to the High Court.[66] Interested parties may bring separate invalidity actions through the judicial system.[67] Judicial invalidity actions brought by interested parties may be initiated at any time after the granting of the patent.[68]

---

[58] *See* Patent Law, *supra* note 11, art. 66.

[59] *See id.*

[60] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 4-5.

[61] *See id.*

[62] *See id.*

[63] *See id.*

[64] *See id.*

[65] *See* Ohba, *supra* note 18, at 279.

[66] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 5.

[67] *See id.*

[68] *See id.*

Initially, an applicant may file applications and supporting documents in either Japanese or English.[44] An applicant filing the initial application in English must submit a Japanese translation of all documents no more than two months after the initial filing.[45] In all cases, the JPO will examine the patent application based on the Japanese-language version.[46] The filing date, however, is based on the initial filing, regardless of which language is used.[47]

2. Formality Examination

Once an application is submitted, the JPO does a preliminary "formality check" to ensure that the application meets the prescribed formality requirements, and to determine whether the applicant has submitted all of the required documents. When the formality check is complete, the JPO notifies the applicant of any defects discovered in the application. The applicant may make amendments if necessary to cure such defects.[48]

3. Pre-Grant Publication

All applications for Japanese patents are "laid open" through publication in the Patent Gazette eighteen months from the date of filing (or eighteen months from a foreign applicant's Paris Convention foreign priority filing date, whichever is earlier).[49] Due to Japan's deferred examination system, most patent applications are laid open before examination by the JPO. To avoid this pre-grant public disclosure, the applicant must withdraw

[44] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 3-4.

[45] *See id.*

[46] *See id.* at 2.

[47] *See generally id.* at 3.

[48] For a detailed discussion of patent application and examination procedures, see Yuji Tsuruya, *The Japanese Patent System* (visited Mar. 20, 1998) <http://www.st.rim.or.jp/~try/patents.html>.

[49] *See id.*

3. Inventive Step

Patentable inventions must incorporate some kind of "inventive step."[34] Under Japanese law, this means that the invention could not "have been easily inferred before the filing of the patent application on the basis of prior art by a person with ordinary skills in the art to which the invention pertains."[35] To qualify for Japanese patent protection, the invention must incorporate a substantial technical advancement.[36] U.S. applicants should note that the "inventive step" requirement is somewhat stricter than the "nonobvious" standard applied by the United States Patent and Trademark Office.[37]

E. *Scope Of Protection*

Registered patents provide monopoly rights to use inventions commercially, or to license inventions for commercial use.[38] Patent rights expire twenty years after the date of filing.[39] Extensions of up to five years may be granted in cases where compliance with government approval procedures delayed exploitation of the patented invention by two or more years.[40]

F. *Application And Examination Procedures*

To qualify for patent protection, inventions must be registered with the JPO through the following procedures:

[34] *See id.* art. 29.

[35] *See id.*

[36] *See generally* Umeda, *supra* note 24. *See also* LEC GUIDE, *supra* note 12, at 3.

[37] LEC GUIDE, *supra* note 12, at 3. For a detailed discussion of the statutory requirements for patent registration, see generally Umeda, *supra* note 24, at 19.

[38] *See* Patent Law, *supra* note 11, arts. 2(3), 68.

[39] *See id.* art. 67.

[40] *See id.* art. 67(3).

chemistry); mathematical formulas; and pure, abstract ideas are considered to exist independently in nature, and thus only may be discovered — not invented.[20] Accordingly, they may not be the subjects of a Japanese patent.[21] Further, certain inventions that otherwise might incorporate technical advancements may not qualify for patent protection if they contravene public order, morals, or health.[22] Additionally, in order to comply with Article 27 of the TRIPS Agreement, Japan now allows patenting of inventions that are manufactured by the transformation of atoms.[23]

Computer programs standing alone are not patentable subject matter because they consist solely of mathematical algorithms.[24] If a program is incorporated as an integrated step in a process or as a component in a physical structure, however, it may qualify for patent protection.[25]

D. *Statutory Requirements*

The statutory requirements for patent registration are utility, novelty and inventive step.[26]

Scene (visited Mar. 20, 1998) <http://www.okuyama.com>.

[20] *See* Yoshikazu Takaishi, *supra* note 19.

[21] [21] *See id.*

[22] *See* Patent Law, *supra* note 11, art. 39(1).

[23] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 1.

[24] *See* Yoshikazu Tani, *Special Problems of Certain Technologies: Computer Software-related Inventions, in* AMERICAN INTELLECTUAL PROPERTY LAW ASSOCIATION, JAPANESE PATENT PRACTICE, PROSECUTION, LICENSING, LITIGATION 113-29 (1994) [hereinafter JAPANESE PATENT PRACTICE]; *see also* Akihiko Umeda, *Fundamental Principles of Japanese Patent Law: Requirements for Patentability, in* JAPANESE PATENT PRACTICE 27.

[25] *See generally* Tani, *supra* note 24.

[26] Patent Law, *supra* note 11, art. 29.

January 1999, the JPO will accept on-line applications between the hours of 9:00 a.m. and 10:00 p.m.[9]

7. Power Of Attorney Requirements

Effective April 1998, applicants using an administrator to handle their filing procedures must submit a power of attorney only for major amendments, abandonment or withdrawal of an application, requests for trial, or applications for registration by a new administrator appointed after patent rights have been established.[10]

B. *Other Key Points*

In addition to the aforementioned recent changes in policy and procedure, the following features highlight the Japanese patent system:

1. First-To-File Rule

The first party to file an application for registration receives patent priority.[11]

2. Public Disclosure

Patent applications are "laid open" to the public eighteen months after filing.[12]

---

[9] *See What's New?*, *supra* note 3. Presently, the JPO will accept on-line applications between the hours of 9:00 a.m. and 8:00 p.m.

[10] *See id.*

[11] Tokkyō hō, [Patent law] Law No. 121 of 1959, art. 39(1), as amended [hereinafter Patent Law].

[12] *See* A GUIDE TO JAPANESE INTELLECTUAL PROPERTY LAW 4-5 (Peter A. del Vecchio ed., 1994) [hereinafter LEC GUIDE].

I. INTRODUCTION

Japanese law provides extensive protection of intellectual property rights, including patents and utility models, trademarks and service marks, copyrights, trade names, and industrial designs. As part of an ongoing effort to harmonize its intellectual property laws with internationally recognized standards, expedite registration procedures, and provide greater protection for intellectual property, Japan has made a number of important revisions in relevant laws in recent years. This article highlights these changes and summarizes the fundamentals of intellectual property protection under current Japanese law.

II. PATENT RIGHTS

A. *Recent Changes*

Japanese patent law recently has undergone a number of revisions to expedite registration and bring Japan's patent system into compliance with internationally accepted standards. Notable changes have been made in several areas.

1. Application In English

An applicant may initially file its application in English.[1]

2. Post-Grant Opposition

To expedite the application and examination process, Japanese law has been amended to allow post-grant opposition to patents.[2]

3. Accelerated Processing

The Japanese Patent Office ("JPO") will carry out patent processing on an accelerated basis if the applicant submits a Search Report issued by a foreign or regional patent office. If an application is accepted for accelerated

[1] *See* JAPANESE PATENT OFFICE, REVISION OF THE JAPANESE INDUSTRIAL PROPERTY LAWS IN 1994 at 3 (1995) [hereinafter 1994 INDUSTRIAL PROPERTY LAWS REVISION].

[2] *See* 1994 INDUSTRIAL PROPERTY LAWS REVISION, *supra* note 1, at 3.

there is no main purpose of an object or class. The second level, system architecture, does not apply because there is no system architecture of an object or class because it is the hierarchical relation of classes and objects that compose an OO system. Where there is no OO system, there is no system architecture. Only the last four levels of abstraction are applicable to the individual classes or objects: (3) various abstract data types; (4) various algorithms and data structures; (5) the source code; and (6) the object code.

VI. Conclusion

Industry is quickly adopting OO software design and development techniques. The majority of innovative software products which are more likely to be contested because of their greater economic value are developed using OO methodology. Existing copyright law is expected to provide protection.[165] The purpose of copyright law is to serve society's interests by "creat[ing] the most efficient and productive balance between protection (incentive) and dissemination of information, to promote learning, culture and development."[166] Software copyright law can only satisfy this criteria by protecting expression in original works of authorship. However, before the expression can be protected, it must be accurately separated from the idea.

Ogilvie's method of abstracting computer programs is generally inadequate for OO software because the method assumes a procedural design in which idea and expression are embodied differently than in an OOD. The assumption of a procedural design in Ogilvie's levels of abstractions means that the system architecture, data structure, and algorithms of OO computer software may not be understood in the correct context. This misunderstanding may confuse the separation of idea from its expression for a computer

[165] *See* Robert D. Sprague, *Multimedia: The Convergence of New Technologies and Traditional Copyright Issues,* 71 DENV. U. L. REV. 635, 669 (1994).

[166] Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1235, 230 U.S.P.Q. (BNA) 481, 490 (3d Cir. 1986).

Implementation of ADTs was limited primarily to stacks and queues. ADTs were rarely used in structured programming because the notation of structured languages was not well suited to implementation of the concept. The wide commercial availability of OO languages in the late 1980s facilitated the common use of ADTs.

The notation of OO languages makes encapsulation much easier to accomplish. With the advent of OO languages, all data in an OO design can be easily encapsulated with the associated methods. One of the reasons OO languages are attractive is that there is a simple mapping from an ADT to its implementation as a class. Nonetheless, some OO software applications are developed without full encapsulation, resulting in a "break" in encapsulation. Most OO languages allow encapsulation to be broken, such as through the use of the "public" notation on a data attribute in C++.[161] Therefore, even OO software may not fully realize ADTs.

D. *Various Algorithms And Data Structures*

Polymorphism can be accounted for in the abstraction of the various algorithms. An algorithm is a precise specification of a method.[162] In polymorphism, methods along paths of inheritance are overridden and redefined.[163] Thus, it is in the various algorithms that polymorphism is exhibited.[164] Traditional structured software design

---

[161] *See* SCHILDT, *supra* note 88, at 384.

[162] *See* FRANK M. CARRANO, DATA ABSTRACTION AND PROBLEM SOLVING WITH C++: WALLS AND MIRRORS 4 (1995).

[163] *See* LEE & TEPFENHART, *supra* note 134, at 231.

[164] Some authors argue that polymorphism is exhibited in the system architecture because inheritance is necessary to exhibit polymorphism, and inheritance is accounted for in the system architecture. *See* e.g. Lee, *supra*, note 130, at 108, 235. This argument is an appealing alternative to accounting polymorphism in the various algorithms. Polymorphism cannot be fully accounted until the algorithms, the precise specification of the methods, are abstracted. The algorithms are abstracted in the fourth level, and that is where polymorphism is exhibited, albeit, dependant upon the forgoing system architecture. The cliche that OO is as easy as PIE, but it must be spelled backwards, see *supra* Part IV.A., is also

the traditional structured method is designed purely on a functional basis.[156]

B. *System Architecture*

When an OO systems analyst abstracts an existing OO software system, the analyst will start by analyzing the architecture of class relationships. The class relationships define the system architecture. Thus, an OO systems analyst will abstract Ogilvie's second level of abstraction, the system architecture, from the class relationships. The class relationships are intimately bound by inheritance. Inheritance is abstracted in OO software in the abstraction of the architecture. The class relationships, i.e., the system architecture, of an OO system cannot be properly identified without an accounting of inheritance.

Below the level of main purpose and system architecture lie the internals of the classes.

C. *Various Abstract Data Types*

Encapsulation is already accounted for in the abstraction of ADTs. Abstract Data Types (ADTs) consist of data types and operations on the data types:[157] "[t]he ADT encapsulates a data type in the sense that the definition of the type and all operations on that type can be localized to one section of the program."[158] In other words, an ADT consists of encapsulated data and methods. Encapsulation is inherent in an ADT. Indeed, some authorities even suggest that abstract data typing is synonymous with

[156] For example, the most popular structured programming language on personal computers and in client/server environments is C, of which the most basic building block of the language is a "function". The first function that is executed when the program is invoked is even named "main," which is the main function of the program. *See* GAMMA, *supra* note 148, at 26-28.

[157] *See* Ogilvie, *supra* note 2, at 535.

[158] *See* ALFRED V. AHO, ET AL., DATA STRUCTURES AND ALGORITHMS 11 (1983).

Furthermore, a substantial portion of the class relationship will be filtered out as idea. In some cases, the entirety of the class relationship will be filtered out as an idea. In particular, where a class hierarchy models real world objects, such as in industrial control or simulation of natural phenomena, a greater proportion of the class relationship will reflect reality and ideas. There will be less protectable expression as a result.

C. *Comparison*

A systems analyst will typically use a software utility such as Rational Rose to draw a graphical depiction of the class relationships.[154] The source code is the input to the reverse-engineering tool. From this graphical depiction, it is quite easy to compare other class inheritance trees to identify similarities in the class relationships. And perhaps most importantly, these diagrams are vastly easier to understand for non-technical judges and juries than many of the common documentation techniques.

V. APPLYING OGILVIE'S LEVELS OF ABSTRACTION TO OBJECT-ORIENTED SOFTWARE

For the purposes of comparing OO software to procedural software, Ogilvie's level of abstraction can be modified for the abstraction of OO software. Ogilvie's abstraction of OO software does not reflect the realities of OOAD, and therefore must be modified accordingly. While Ogilvie's levels of abstraction do not specifically account for these characteristics, Ogilvie's levels can be understood in light of the OO paradigm.

The six levels of abstraction of OO software are the program's main purpose, the program's system architecture, various abstract data types, various algorithms and data structures, the source code, and the object code. The unique characteristics of OO software,

[154] Rational Rose supports the Unified Modeling Language (UML) standard of graphical documentation. *UML Modeling Language, Standard Software Notation: Resource Center* (last modified June 15, 1998) <http://www.rational.com/uml/index.shtml>.

If Ogilvie's idea of levels of abstraction are at all directly applicable to the abstraction of OO software, it is within the abstraction of the views. For example, in generalization/specialization and aggregation/composition:

> When dealing with . . . "generalization and specialization" . . . hierarchies . . ., we often speak of *levels of abstraction*, a concept first described by Dijkstra. In terms of its 'is a' hierarchy, a high-level abstraction is generalized, and a low-level abstraction is specialized. Therefore, we say that a *Flower* class is at a higher level of abstraction then a *Plant* class. In terms of the 'part of' hierarchy, a class is at higher level of abstraction than any of the classes that make up its implementation. Thus the class *Garden* is at a higher level of abstraction than the type *Plant*, upon which it builds.[147]

Accordingly, it is not improper to abstract OO software in terms of levels of abstraction as Ogilvie has proposed, but it is improper to abstract the levels within the views.

B. *Filtration*

The second step in the process of determining copyright infringement of OO software is filtration of unprotectable elements. Unprotected elements are ideas, elements dictated by logic and efficiency, elements dictated by external considerations such as hardware and software standards, and elements from the public domain.

"Design patterns" and "frameworks" should be examined closely as candidates for filtration as public domain elements. Design patterns are a "general arrangement of . . . classes."[148] used in the

[147] *See* BOOCH, *supra* note 144, at 65. Ogilvie derived many of his ideas on levels of abstraction from Dijkstra. *See id.*

[148] *See* ERICH GAMMA, ET AL., DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE 3 (1995).

polymorphism, and thus further reduce the significance of association/membership and aggregation/composition.[141]

The association/membership view involves classes that use other classes. This relationship is akin to situations where a patent law firm uses a patent search firm, a person uses a bank account, a club has members, or a husband has a wife. The association allows objects of one class to request services of objects of another class.[142] The relationship can be subordinate or peer. In languages that explicitly support association, support is accomplished through association classes.[143]

[141] There is considerable disagreement among OO experts on the definitions of association, membership, aggregation, and composition. Some experts define some aspects as variations of the other aspects, and thus do not recognize three distinct types of aspects. The bounty of ideas on how these aspects of OO software relate to each other is illustrated by the discussion posted by Object Currents Journal at <http://www.sigs.com/publications/docs/oc/9608/oc9608.d.dialog.html> and by Rational Corp. at <http://www.rational.com/HyperMail/otug/1737.html>. At these sites, definitions are suggested that attempt to minimize the disagreements and find a happy middle ground.

[142] *See* LEE & TEPFENHART, *supra* note 134, at 143.

[143] Association is documented architecturally in UML by association diagrams. *See id.* at 180. Following the above example of an association class named "employment" between the classes "employer" and "employee," would be depicted as follows:

```
EMPLOYEE ---------------------- EMPLOYER
                  |
                  |
                  |
             EMPLOYMENT
             (Attributes)
              (Methods)
```

Qualified associations are documented in UML through Qualifed Association Diagrams. *Id.* Association and membership extends beyond the class relationships to the class internals. Association and membership in the class internals can be documented using a diagram and notation specifically designed for that purpose. *Id.*

through an examination of the generalization and specialization view of the classes.

Generalization is abstraction of classes. Therefore, in general, association/membership and aggregation/composition views are less important views than generalization/specialization because they are not explicitly supported in many OO languages,[139]

[139] *See* LEE & TEPFENHART, *supra* note 134, at 249. However, C++ and Java implicitly support association and membership through buried pointers, in which one class has as an attribute a pointer to an associated class. This is demonstrated in the following C++ class in which an employee is able to access a service of its employer:

```
class employee :: person
{
  public:
    employee(int I, char s);//constructor
    employee *getEmployer() {return employer;}
    void setEmployer(employer *e) {employer=e;}
    ~employee() {};          //destructor
  private:
    int age;
    char name;
    employee *employer;

};
```

In terms of cardinality, associations can be implemented a 'one-to-one', 'one-to-many,' and' many-to-many,' associations. *Id.* at 250-51, 257; FOWLER & SCOTT, *supra* note 135, at 58. The above example implements a 'one-to-one' cardinality, where each person is associated with one employer. To implement 'one-to-many' cardinality, an array of pointers to person objects would be used in the 'many' class.

Furthermore, C++ and Java implicitly support aggregation through buried pointers, in which one class has as attributes, pointers to an aggregation of class objects. This is shown in the following C++ class for a car where other objects may access the aggregated object directly:

```
class car
{
  public:  car()  //constructor
    {
      engine=Engine(); //instantiate the engine
      body=Body();     //instantiate the body
    };
    ~car() //destructor
```

documented in one of the many OO design notations[131] and the class internals can be documented in terms of the three views.

Regardless of which of the three views are documented first, the first step in abstraction of the SSO of OO software is documentation of the classes.[132] Documentation is required because the classes need to be identified and their scope established before the relationship between the classes and the class internals can be analyzed. *Encapsulation,* one of three unique characteristics of OO software,[133] will be inherently abstracted during this step because a class is an encapsulation of data and methods. After the classes have been documented the OO software can be analyzed in any order of the three views: generalization/specialization, association/membership, and aggregation/composition.

[131] The most popular notation is Unified Modeling Language (UML), but there are many other methods, such as Booch, Rumbaugh, Jacobson, (the latter three of which were combined to yield UML), Coad, Odell, Coad, Shlaer/Mellor, or Views (the architecture can be abstracted in five views, a logical view, a process view, a development view, a physical view and scenarios). *See* Philippe Krutchen, *The 4+1 View Model of Architecture,* 12 IEEE SOFTWARE 42-50 (1995). Scenarios are also known as use-cases. *See* GRADY BOOCH, THE BEST OF BOOCH 113 (Ed Eykholt, ed. 1996); *see also* IVAR JACOBSON ET AL., OBJECT-ORIENTED SOFTWARE ENGINEERING: A USE CASE DRIVEN APPROACH (1994).

[132] This is necessary as a first step because all structure, sequence, and organization in OO software are based on the class concept. Documentation of the classes will be done in UML via class diagrams as follows:

```
 ______________________________________
|                                      |
|             Class Name               |
|  __________________________________  |
| attribute:data_type                  |
| attribute:data_type=init_value       |
| ...                                  |
| ____________________________________ |
| operation                            |
| operation (arg_list):result_type     |
| ____________________________________ |
```

[133] The other two unique characteristics are inheritance and polymorphism. *See supra* Part II.A.1.

have the effect of making software engineers excessively cautious in their disposition toward innovation which could lead to less useful programs, and in turn to lowered productivity[126] and a less successful software industry. "Software copyright law should reflect the engineering realities of programming, just as the Uniform Commercial Code reflects actual mercantile practice and real property law reflects pragmatic aspects of land ownership."[127]

IV. A New Taxonomy For Abstraction, Filtration, And Comparison Of Object Oriented Software

Given the inability of the Ogilvie test to account for encapsulation, inheritance, and polymorphism, it is clear that, for purposes of analyzing copyright infringement, the OO software should be abstracted in an entirely different taxonomy than Ogilvie's levels of abstraction. The suggested taxonomy more accurately reflects the SSO of OO software.

---

through inheritance.

Lacking the concept of polymorphism, the following source code:

```
169     PopupMenu* pPopup_Menu;
170     pMenu=new OwnerMenu;
171     int result =pPopup_Menu->Show();
```

at line 171 would appear to perform the instructions in the Show method in PopupMenu (lines 28-79), when in fact, line 171 will perform Show method in OwnerMenu (line 125) which is a method that has a drastically different result than the Show method in PopupMenu (lines 28-79).

The above example illustrates how the lack of OO concepts will lead to an incorrect understanding of OO software. This understanding, in turn would naturally lead to total confusion in the separation of expression and idea.

[126] In contrast, several well publicized studies have shown no productivity gain from computer automation.

[127] *See* Ogilvie, *supra* note 2, at 533-34.

```
      OwnerMenu();
      virtual int Show(char* match);
      virtual int Show();
   private:
      OwnerItem Ownerlist[MAX_CHOOSE_ITEMS];
};

#include <string.h>

OwnerMenu :: OwnerMenu() : PopupMenu("Select an Owner")
{

   OwnerStruct Owners[MAX_CHOOSE_ITEMS];
   int iRecords=GetOwnerList(Owners);
   if(iRecords)
   {
      char MenuString[22];
      for(int I=0; i<iRecords; I++)
      {
         strncpy(Ownerlist[i].item, Owners[i].Name, 10);
         strcat (Ownerlist [I].item, "       ");
         Ownerlist[i].item[10]='\0';
         strcat(Ownerlist[i].item, " = ");
         strcat(Ownerlist[i].item, Owners[i].ID);
         AddItem (Ownerlist[i].item, "");
      }
   }
};
int OwnerMenu :: Show(char* match)
{
   PopupMenu::Show(match);
   return ChosenIndex ();
};
int OwnerMenu :: Show(){return TRUE;}

//______________________________________________
#include "popmenu.h"
#include "database.h"

struct driveritem
{
   char item[18];
};

typedef struct driveritem DriverItem;

class DriverMenu : public PopupMenu
{
   public:
```

```
// -- PopupMenu class
// --
#include "menu.h"

class PopupMenu : public Menu
{
  public:
    PopupMenu (const char *Title);
    ~PopupMenu ();
    virtual int Show (char* match);
        virtual int Show();

      protected:
    };
    #include <string.h>
    #define QUIT (MAX_COMMAND + 1)
    #define ITEM_CHOSEN (MAX_COMMAND + 2)

    PopupMenu::PopupMenu (const char *Title) : Menu(Title)
    {
    };

    PopupMenu::~PopupMenu ()
    {
    };

    int PopupMenu::Show ()
    {
      return 0;
    };

    int PopupMenu::Show (char* match)
    {
      int Height, ItemWidth, Width;
      Chosen_Index = -1;
      TheMenu = new_menu (MenuItems);

      // -- Change the mark string
      set_menu_mark (TheMenu, "*");

      // -- Store the size
      scale_menu (TheMenu, &Height, &ItemWidth);

      // -- Check the Title
      Width = ItemWidth;
      if (strlen (MenuTitle) > ItemWidth)
      {
        Width = strlen (MenuTitle);
      };
```

modules relating to particular *functions* is a common programming technique."[118]

C. *The Goal Of The Test Is To Filter-Out Idea*

Because software enjoys the unusual benefit of being protected by both patent law and copyright law, it is particularly important to filter patentable idea from expression in software copyright analysis. This goal is not easy to achieve because law and computer science, two incompatible schools of thought, must find a common language.[119] Triers of fact typically are not schooled in computer science, and computer scientists typically are not schooled in law. The arcane nature of the two subjects compounds the challenge of reconciling the two subjects.

Furthermore, separating idea from expression is difficult even under the best of circumstances. Judge Learned Hand, an experienced trier of copyright infringement cases, observed toward the end of his career: "Obviously, no principle can be stated as to when an imitator has gone beyond copying the 'idea,' and has borrowed its 'expression.' Decisions must therefore inevitably be *ad hoc*."[120] Fortunately, the engineered nature of computer programs[121] lends itself to objective

[118] *See* Softel, Inc. v. Dragon Med. And Scientific Communications, Inc., 1992 WL 168190 at *15 (S.D.N.Y. 1992)(emphasis added), *vacated*, 118 F.3d 955 (2d Cir. 1997), *cert. denied*, 118 S. Ct. 1300 (1998).

[119] *See Micro Consulting*, 813 F. Supp. at 1526 ("[T]he task [of separating idea from expression] is even more daunting in the field of computer software.").

[120] *See* Peter Pan Fabrics, Inc. v. Martin Weiner Corp., 274 F.2d 487, 489, 124 U.S.P.Q. (BNA) 154, 155 (2d Cir. 1960).

[121] *See* Richard A. Beutel, *Software Engineering Practices and the Idea/Expression Dichotomy: Can Structured Design Methodologies Define the Scope of Software Copyright?*, 32 JURIMETRICS J. 1, 7 (1991) ("Programming is probably most strongly akin to engineering. It is concerned with making structures, machines, products, systems, and processes useful to mankind. In the computer field, the term "software engineering" is being increasingly used to indicate that software development must be primarily concerned with constructing useful tools. Such development requires using techniques, methodologies, and previously constructed building

B. *Assumption Of Procedural Design In Ogilvie's Abstractions Test*

Ogilvie's discussion of the development process and abstractions is replete with terminology grounded in procedural design.[108] Ogilvie states that "[t]wo programs may perform the same *functions* despite differences in their source code."[109] He describes the method of designing a program as "top-down,"[110] which in the language of traditional computer science indicates a procedural design.[111] The process of designing a program is summarized by Ogilvie as follows: "A program begins as a purpose or desired function, which programmers expand into a preliminary design. Programmers make this design increasingly specific by *splitting large tasks into smaller ones* and defining the interaction of these tasks."[112] Ogilvie even uses source written in C language, which is designed in a procedural analysis, to exemplify a computer source code.[113]

Ogilvie's test is grounded in procedural design methodology as is the Abstractions test that courts have been using. The *Altai* court described the process of computer program design as follows:

> The first step in this procedure is to identify a program's ultimate *function* or purpose. An example of such an ultimate purpose might be the creation and maintenance of a business ledger. Once this goal has been achieved, a programmer breaks down or "decomposes" the program's ultimate *function* into

[108] *See* Ogilvie, *supra* note 2.

[109] *Id.* at 531 (emphasis added).

[110] *See id.* at 532.

[111] *See* JOHN MOTIL, PROGRAMMING PRINCIPLES: AN INTRODUCTION 2-1 (1984) ("Top-down Design . . . is the process of creating algorithms in stages, by successively refining them into smaller sub-algorithms.").

[112] Ogilvie, *supra* note 2, at 532 (emphasis added).

[113] *See id.* at 542 n.74.

laboratory."[106] Often, this main purpose will be given to the systems analyst when the project is assigned.

Then the system architecture will be defined, likewise, in terms of the function that each portion of the system is to perform. Examples of the system architecture of a dental office management program are accounting, patient files, charting, and patient communication. The function of each of these major portions of the system will be defined sufficiently to define the interaction of the modules.

The next step is to define of the abstract data types for each of the modules in the system architecture. All of the data types will be defined in the abstract, that is on paper, without any corresponding computer code.

The fourth step is to define the actual data and algorithms associated with the data. The heart of this step is identification of the data, their role within each module, the circumstances under which the data will be modified, and how the data will be modified. These definitions are carried out to such a specific degree that computer code can be written directly from the definitions of the data structures and algorithms.

The last steps are creation of the source code from the definitions of the data structures and the algorithms, and compilation of the source code into object code. Creation of source code in an OOD differs from creation of source code in a procedural design only in that a language that is specifically suited to the design methodology is used. Creation of object code is not different between the two design methodologies.

[106] Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1236 n.28, 230 U.S.P.Q. (BNA) 481, 490 n.28 (3d Cir. 1986) (emphasis added).

Procedural design focuses on the function to be performed. First, the main purpose is defined. The main purpose will be a one sentence description of a computer program, just as in an OO design.[100] Then, each of the major functions are in turn broken down into minor functions. The process of breaking functions down into a series of smaller functions is performed iteratively until the function is defined in terms that have a one-to-one correspondence with statements in the computer source language in which the program will be coded.[101]

A procedural design can be documented by use of a "function diagram," which:

> breaks down a system into its components, starting at the top with the broadest description of the system and continuing in a *top-down* fashion to more detailed delineations of the subsystems. The diagram starts with a single box that identifies the major function of the system. The next level down subdivides the upper level into roughly equivalent major functions. Each successive level further subdivides the functions from which it extends. This subdividing process continues

---

> and subroutines. Finally, the programmer writes specific source code to perform the function of each module or subroutine, as well as to coordinate the interaction between modules or subroutines. (Footnotes omitted.)

Id.

[100] D.L. Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules*, COMM. OF THE ACM, Dec. 1972, at 2.

[101] *See* ALAN W. BIERMAN, GREAT IDEAS IN COMPUTER SCIENCE 110 (1990). The process of "top-down" design is to "[d]ecompose the problem into simpler subtasks and then repeatedly decompose those subtasks until at the lowest level each remaining subtask is easy to comprehend. The solution to the whole problem is the assembly of all the subtask solutions obtained in the decomposition." *Id.*

The problem definition occurs at the beginning of the development process.[90] The problem is often identified and defined by users, people who are not computer specialists.[91] At other times, the problem is identified jointly by users and computer specialists as they work together to identify the scope of the project, balancing the costs and benefits of the development process.

Class development follows the definition of the main purpose. This includes definition of the data abstractions, algorithms, and data structures.[92] Data abstractions and data structures are part of a class because the data is encapsulated within the classes.[93] The algorithms are part of the class because the methods (functions) are defined within the structure of the class.[94]

System architecture is abstracted after the program's main purpose and classes have been defined.[95] "The architecture of an object-oriented system is determined by it's component objects. . . . This architecture also reflects the problem domain since it consists mainly of objects from the problem domain."[96] "The abstract nature

[90] *See* BOOCH, *supra* note 5, at 190.

[91] *See* MCGREGOR & SYKES, *supra* note 10, at 60.

[92] *See* DARREL INCE, OBJECT-ORIENTED SOFTWARE ENGINEERING WITH C++ 24 (1991).

[93] *See id.*

[94] *See id.*

[95] *See* BOOCH, *supra* note 5, at 190; COAD & YOURDON, *supra* note 5, at 198-201.

[96] *See* MCGREGOR & SYKES, *supra* note 10, at 53.

Encapsulation is one of the unique aspects of OOAD. The attributes (data) of each class are *encapsulated* with the methods (functions).[84] Encapsulation means that the data of an object cannot be modified by other objects. Rather, other objects invoke methods of the object to modify the object's attributes. The methods of each object can modify it's own attributes, but no object can modify the attributes of another object without an explicit exception that is noted in the definition of the attribute.[85]

Encapsulation is exhibited in common everyday life. For example, a clock encapsulates time. To change the time, we use the methods defined for the alarm clock, such as a "set time" button. The time is the encapsulated data, and the "set time" button is the method.

Inheritance is also unique to OOAD. Typically, two related classes share attributes and methods in common. Rather than defining each of the classes separately, which would make the common definitions duplicative, each class will be defined as a child of the parent class of which it is a member. For example, the Saturn SL1 and SL2 are two kinds (classes) of cars. Most of the attributes and functions they perform are shared in common. Rather than defining each of these kinds of cars completely from scratch, a parent class is first defined, Sedan, that contains all of the commonalities. Then, the two child classes, SL1 and SL2, are defined as child classes of Sedan, and the unique attributes and functions of each child class are

polymorphism, but does include encapsulation.

[84] *See* WIENER & PINSON, *supra* note 80, at 4.

[85] There is an exception: almost every OO programming language offers extensions that allow encapsulation to be broken. However, breaking encapsulation is a poor idea for two reasons. *Id.* It increases the chance of "bugs" in the software, and it destroys the object-orientation of the software. *Id.* Furthermore, most OO programming languages, such as C++, allow structured source code to be compiled, so the fact that an OO compiler was used to compile the source code, does not mean the source code is OO. *See id.*

III. PROBLEMS USING OGILVIE'S ABSTRACTIONS TEST FOR OBJECT-ORIENTED SOFTWARE

In the history of software development, there were three major paradigms[77] of analysis[78]. The first paradigm occurred in the 1950s when "callable routines" were written that could be used by programs from multiple points within the programs. The second paradigm occurred in the 1970s when Procedural Analysis was developed and became dominant.[79] The third paradigm is OOAD, which was developed in the 1980s, and is currently gaining widespread acceptance.[80]

Ogilvie's Abstractions test for substantial similarity was specifically tailored for procedural design and will not properly separate idea from expression in OO software.

A. *Differences Between Object-Oriented Analysis And Design And Procedural Design*

OOAD and procedural design are completely different methods of designing software; the methods cannot be used interchangeably during development of a software program.[81] Object-

[77] *See* AT&T, OBJECT-ORIENTED SYSTEM DESIGN § 2, at 5 (1992) ("A software development paradigm is a technique for deriving the structure of a software system.").

[78] Currently, there are six paradigms: procedural, functional, relational, object-oriented, access-oriented, and process-oriented. *See id.* § 2, at 9.

[79] *See, e.g.*, MCGREGOR & SYKES, *supra* note 10.

[80] *See* RICHARD S. WIENER & LEWIS J. PINSON, AN INTRODUCTION IO OBJECT-ORIENTED PROGRAMMING AND C++ 1 (1988); COAD & YOURDON, *supra* note 5, at 188.

[81] Both methods can be used interchangeably where an object-oriented program invokes a procedurally oriented program, i.e., a method in an object that was written in C++ that invokes a function in a dynamic link library that was written in C.

Moreover, the process of development[69] that computer scientists follow is a process that parallels Ogilvie's levels of abstraction, which allows a judicial trier of fact to examine the software at each stage of development. This modified test has been used by the Tenth Circuit in *Gates Rubber Co. v. Bando Chemical Industries, Ltd.*[70]

In *Gates Rubber*, the court ostensibly used Ogilvie's levels of abstraction[71] when it identified those levels as: (i) the main purpose, (ii) the program structure or architecture, (iii) modules, (iv) algorithms and data structures, (v) source code, and (vi) object code.[72] The third level, "modules," is different than Ogilvie's third level, "various abstract data types." These two abstractions of computer software are distinctly different. A module is a major subdivision of a program that is commonly compiled into a separate file.[73] Abstract data types are definitions of data structures for which no memory is allocated.[74] It is impossible to conceive of the "modules" of *Gates Rubber* as synonymous with the "abstract data types" of Ogilvie's thesis.

Nonetheless, the Tenth Circuit's formulation of the levels of abstraction may have no practical ill effects for courts that use *Gates Rubber*'s levels of abstraction, and it did not present any practical problem in application. In Ogilvie's levels of abstractions "[t]he *system*

[69] The process of development is typically referred to as "life cycle" by software developers.

[70] 9 F.3d 823, 834, 28 U.S.P.Q.2d (BNA) 1503, 1508 (10th Cir. 1993).

[71] *See* Ogilvie, *supra* note 2, at 533.

[72] *Gates Rubber*, 9 F.3d at 834-35, 28 U.S.P.Q.2d (BNA) at 1508-09 (10th Cir. 1993).

[73] *See* ROBERT J. VERZELLO & JOHN REUTTER III, DATA PROCESSING SYSTEMS AND CONCEPTS 287 (1982).

[74] Abstract data types are used in the allocation of memory during execution of the program by "data structures," which is included in Ogilvie's fourth level of abstraction.

The *Altai* court examined four non-literal elements of 3.5: parameter lists, macros, list of services, and organizational charts.[60] The court found that most of the similarities in the parameter lists and macros were unprotectable as elements either in the public domain or dictated by the functional demands of the program.[61] The *de minimis* amount of protected similarity in parameter lists and macros did not rise to the level of substantial similarity.[62] The court also found the list of services to be unprotectable because it "was dictated by the nature of other programs with which it was designed to interact" which presumably means that it was, in terms of Nimmer's test, dictated by external considerations.[63] The organizational charts were also found to be unprotectable because they were "so simple and obvious to anyone exposed to the operation of the program,"[64] a rationale which could fit a number of categories of unprotectable expression: elements dictated by logic and efficiency, elements dictated by the external considerations of software standards, or computer industry programming practices. The *Altai* court found no substantial similarity in the non-literal elements of the allegedly infringing OSCAR 3.5 programs because any substantial similarities were filtered out as unprotectable idea.[65]

The court in *Altai* reached the correct result and applied Nimmer's test correctly for OSCAR 3.5. The mere fact that OSCAR 3.5 was written from reverse-engineered specifications makes any substantial similarity improbable. While the *Altai* opinion fails to clearly define "macros", the court correctly found that the parameter

[60] *Altai*, 982 F.2d at 714-15, 23 U.S.P.Q.2d (BNA) at 1259-60.

[61] *See id.*

[62] *See id.*

[63] *See id.* at 715, 23 U.S.P.Q.2d (BNA) at 1260.

[64] *Id.* (quoting *Computer Assocs.*, 775 F. Supp. at 562).

[65] *See id.* at 714, 23 U.S.P.Q.2d (BNA) at 1259.

by a court in 1990.[44] *Micro Consulting, Inc. v. Pedro Zubeldia and Medical Electronic Data Exchange, Inc.* involved two personal computer programs that automated medical insurance claims processing.[45] At the highest level of the non-literal elements, the "overall function is accomplished in part by receiving and storing claims from health care providers, identifying, validating and editing those claims and then transmitting the claims to insurance carriers."[46] The court defined these elements as ideas.[47]

The *Micro Consulting* court found the non-literal elements at lower levels of abstraction, the claim identification and claim validation functions, to have protectable expression.[48] Yet, one of the programs, which scrolls, synchronizes, and stops at the first error, was less than substantially similar to another program that paginates and is able to detect multiple errors.[49] The court also found that both programs used back-end processors and used Kermit file transfer protocols, but that such use was in the public domain, and thus was filtered out as unprotectable.[50]

The Second Circuit's opinion in *Altai*[51] marked the beginning of the widespread adoption of the abstraction, filtration, and

[44] *See* Micro Consulting, Inc. v. Zubeldia, 813 F. Supp. 1514, 1529 (W.D. Okla. 1990), *aff'd*, 959 F.2d 245 (10th Cir. 1992).

[45] *See id.* at 1518.

[46] *See id.* at 1529.

[47] *See id.*

[48] *See id.*

[49] *See id.*

[50] *See id.* at 1530.

[51] Computer Ass'ns Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 23 U.S.P.Q.2d (BNA) 1241 (2d Cir. 1992).

significant role in determining the sequence."[32] In *Plains Cotton Cooperative Ass'n of Lubbock, Texas v. Goodpasture Computer Service, Inc.*, the court ruled that where uncopyrightable ideas, which are rooted in the customs and practices of the industry for which the program was written, have a significant effect on the non-literal elements, *all* of the non-literal elements are uncopyrightable.[33]

The courts in both *Whelan* and *Plains Cotton* recognized that the non-literal elements of computer software are a dimension of computer software separate from the literal text of the code and the screen displays.[34] However, both cases exemplify "a take it or leave it" approach that defines the non-literal elements as either completely protectable or completely unprotectable. Neither case recognizes that the truth may be somewhere in between: that some portions of the non-literal elements are protectable expression and some are unprotectable idea.

C. *Courts' Adoption Of Nimmer's Concept*

In 1988, Professor Nimmer proposed a test for substantial similarity of non-literal elements.[35] Professor Nimmer's test first abstracts the non-literal elements of the allegedly infringed program, starting at the literal source code level, into successively higher levels.[36] Then it filters out the unprotectable elements, and finally it compares the remaining protectable expression to the alleged

[32] 807 F.2d 1256, 1262, 1 U.S.P.Q.2d (BNA) 1635, 1640 (5th Cir. 1987).

[33] *Id.*

[34] *See id.*, Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1244-45, 230 U.S.P.Q. (BNA) 481, 497 (3d Cir. 1986) (recognizing that similarity in screen displays may be evidence of similarity in the underlying code).

[35] *See* Nimmer, *supra* note 14, at 625-26.

[36] *See id.* at 636-38.

II. Development Of The Abstractions Test

The Abstractions test was first formulated by Judge Learned Hand,[19] and courts subsequently struggled with its application to software.[20] Later, abstraction of software was further refined by Professor Nimmer[21] and John W. Ogilvie.[22]

A. *The Origin—Judge Learned Hand*

The Abstractions test originated with Judge Learned Hand in 1930.[23] The test described a method of separating the idea from the expression of a literary work by starting with the literal text and identifying the "patterns of increasing generality" successively until the entire work could be described in a single statement.[24] Judge Hand's rationale was that a copyright "cannot be limited literally to the text, else a plagiarist would escape by immaterial variations."[25] While Judge Hand's method of abstractions was devised for a play,

[19] *See infra* Part II.A.

[20] *See infra* Part II.B.

[21] *See infra* Part II.C.

[22] *See infra* Part II.D.

[23] *See* Nichols v. Universal Pictures Corp., 45 F.2d 119, 7 U.S.P.Q. (BNA) 84 (2d Cir. 1930). *See also* Engineering Dynamics, Inc. v. Structural Software, Inc., 26 F.3d 1335, 1343, 31 U.S.P.Q.2d (BNA) 1641, 1646 (5th Cir. 1994); Gates Rubber Co. v. Bando Chem. Indus., Ltd. 9 F.3d 823, 834, 28 U.S.P.Q.2d (BNA) 1503, 1509 (10th Cir. 1993); Autoskill v. Nat'l Educ. Support Sys., Inc., 994 F.2d 1476, 1488, 26 U.S.P.Q.2d (BNA) 1828, 1836 (10th Cir. 1993); Productivity Software Int'l, Inc. v. Healthcare Techs., 37 U.S.P.Q.2d (BNA) 1036, 1039 (S.D.N.Y. 1995); Mitek Holdings, Inc. v. Arce Eng'g Co., Inc., 864 F. Supp. 1568, 1578, 34 U.S.P.Q.2d (BNA) 1417, 1422 (S.D. Fla. 1994); Micro Consulting, Inc. v. Zubeldia, 813 F. Supp. 1514, 1526 (W.D. Okla. 1990), *aff'd*, 959 F.2d 245 (10th Cir. 1992).

[24] *See Nichols*, 45 F.2d at 121, 7 U.S.P.Q. (BNA) at 86.

[25] *See id.* at 121, 7 U.S.P.Q. (BNA) at 86.

software development, such as reusability, easier maintenance, and lower complexity.[10] OOAD is likely to become the dominant method of analysis and design in software development after the turn of the century, if not by the end of the 1990s.[11]

The *Altai* court considered the possibility that software development technology could quickly make its formulation of the abstractions test obsolete, and stated, "we are cognizant that computer technology is a dynamic field which can quickly outpace judicial decisionmaking."[12] The court instructed inferior courts to update the test to new software technologies that may arise "in cases where the technology in question does not allow for a literal application of the procedure we outline below, [and cautioned that its] opinion should not be read to foreclose the district courts of our circuit from utilizing a modified version."[13] The test used to determine substantial similarity in software is crucial because the other element of infringement, access, is typically conceded.[14]

The need to modify the abstractions test has arisen because a different method of abstraction is needed for software designed with an object-orientation. Where OO software needs to be compared to structured software to determine copyright infringement, it is both

---

[10] *See* JOHN D. MCGREGOR & DAVID A. SYKES, OBJECT-ORIENTED SOFTWARE DEVELOPMENT: ENGINEERING SOFTWARE FOR REUSE 24-25 (1992).

[11] *See* Kristine M. Morrison, *Warming Up to Objects*, DEC PROFESSIONAL, July 1995, at 20.

[12] *Altai*, 982 F.2d at 706, 23 U.S.P.Q.2d (BNA) at 1252.

[13] *Id.*

[14] *See* Ogilvie, *supra* note 2, at 526-27; Computer Assocs. Int'l, Inc. v. Altai, Inc., 775 F.Supp. 544, 558, 20 U.S.P.Q.2d (BNA) 1641, 1649 (E.D.N.Y. 1991) ("In most cases the 'substantial similarity' inquiry presents the heart of a copyright infringement case."); David Nimmer, et al., *A Structured Approach to Analyzing the Substantial Similarity of Computer Software in Copyright Infringement Cases*, 20 ARIZ. ST. L.J. 625, 626-27 (1988) [hereinafter Nimmer].

that the abstractions test is the most appropriate test.[3] It seems likely that this test will prevail through the 1990's.

In 1992, the Second Circuit was the first federal circuit court of appeals to utilize an abstractions test for substantial similarity in the non-literal elements of software. The court, in *Computer Associations International, Inc. v. Altai, Inc.*,[4] devised a test whose language is rooted in a software development technology that was dominant at that time: Procedural Analysis. The abstractions test for software is premised

[3] *See* Bateman v. Mnemonics, Inc., 79 F.3d 1532, 38 U.S.P.Q.2d (BNA) 1225 (11th Cir. 1996)(remanding to district court in part to reconsider software copyright infringement applying the abstractions test to both literal and non-literal elements of software); Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 34 U.S.P.Q.2d (BNA) 1014 (1st Cir. 1995) (declining to apply the abstractions test because only literal copying was at issue, but nonetheless endorsing the test); Engineering Dynamics, Inc. v. Structural Software, Inc., 26 F.3d 1335, 31 U.S.P.Q.2d (BNA) 1641, (5th Cir. 1994) (adopting the abstractions test to analyze scope of copyright protection for user interface, input formats, and output reports); Kepner-Tregoe, Inc. v. Leadership Software, Inc., 12 F.3d 527, 29 U.S.P.Q.2d (BNA) 1747 (5th Cir. 1994)(declining to apply the abstractions test because substantial similarity was stipulated, but nonetheless endorsing the test); Comprehensive Techs. Int'l, Inc. v. Software Artisans, Inc., 3 F.3d 730, 736, 28 U.S.P.Q.2d (BNA) 1031, 1034-35 (4th Cir. 1993); Autoskill Inc. v. Nat'l Educ. Support Sys., Inc., 994 F.2d 1476, 1487, 26 U.S.P.Q.2d (BNA) 1828, 1839-40 (10th Cir. 1993); Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 28 U.S.P.Q.2d (BNA) 1503 (10th Cir. 1993) (adopting Ogilvie's six levels of declining abstractions); Mitel, Inc. v. Iqtel, Inc., 896 F. Supp. 1050 (D. Colo. 1995); Control Data Sys. v. Infoware, Inc., 903 F. Supp. 1316 (D. Minn. 1995); Productivity Software Int'l, Inc. v. Healthcare Techs., 37 U.S.P.Q.2d (BNA) 1036 (S.D.N.Y. 1995); DSC Communications Corp. v. DGI Techs., 898 F.Supp. 1183, 1190 (N.D. Tex. 1995); Triad Sys. Corp. v. Southeastern Express Co., 31 U.S.P.Q.2d (BNA) 1239, 1248 (N.D. Cal. 1994); Mitek Holdings, Inc. v. Arce Eng'g Co., Inc., 864 F. Supp. 1568, 34 U.S.P.Q.2d (BNA) 1417, (S.D. Fla. 1994); Cognotec Servs. v. Morgan Guar. of N.Y., 862 F. Supp. 45 (S.D.N.Y. 1994); Atari Games Corp. v. Nintendo of Am., Inc., 30 U.S.P.Q.2d (BNA) 1401 (N.D. Cal. 1993); Computermax, Inc. v. UCR, Inc., 804 F.Supp. 337, 351-56, 26 U.S.P.Q.2d (BNA) 1001, 1012-16 (M.D. Ga. 1992); Softel, Inc. v. Dragon Med. and Scientific Communications, Inc., 1992 WL 168190, at *22-25 (S.D.N.Y. 1992) (adopting the abstraction, filtration, and comparison method).

[4] 982 F.2d 693, 23 U.S.P.Q.2d (BNA) 1641 (2d Cir. 1992).

AIPLA QUARTERLY JOURNAL

VOLUME 26, NUMBER 1 WINTER 1998

PUBLICATION STAFF

AIPLA student members interested in participating on the Publication Staff should direct inquiries, in writing, to AIPLA Quarterly Journal, 2001 Jefferson Davis Highway, Suite 203, Arlington, VA 22202-3694. Members are selected annually for 2-year terms beginning in August.

American Intellectual Property Law Association