プログラムの特徴認識の転換と知的財産保護

目次

1　はじめに

1-1　プログラムとは

コンピュータ・プログラム（以下プログラム）の知的財産保護においては、著作権法による保護が昭和 60 年の同法改正により明定された。

また平成 14 年特許法改正ではプログラムが物の発明として保護されることが明らかになった。

プログラムとは何か、についてはそれぞれの法律に定義規定が置かれている。

著作権法においては同法第 2 条第 1 項第 10 号の 2 で「プログラム　電子計算機を機能させて一の結果を得ることができるようにこれに対する指令を組み合わせたものとして表現したものをいう。」とされている。なお第 10 条第 3 項で「プログラム言語」「規約」「解法」については著作権法の保護が及ばないとされており、これらは同法で保護されるプログラムには該当しないこととされている。

他方、特許法においてはどうか。同法第 2 条第 4 項に「…プログラム（電子計算機に対する指令であって、一の結果を得ることができるように組み合わされたものをいう。…）」とある。著作権法との定義の違いは「表現したもの」という文言がないところであり、著作権法では保護が及ばないとされる「解法　プログラムにおける電子計算機に対する指令の組合わせの方法」（著作権法第 10 条第 2 項第 3 号）をも保護対象に含む定義と読むこともできる。

このように定義されたプログラムの有する性格、特徴については知的財産法の世界ではある程度コンセンサスが取れたうえで議論が行われているようである。しかしこのコンセンサスというものが著作権法の世界にプログラムが登場してから 20 年近くの間あまり変わっていないように思われる。その間にコンピュータは急速に進歩し、人々のコンピュータへの関わりが身近になりその変化の流れに生活そのものが飲み込まれようとしているほどなのに、である。本稿は知的財産法の世界におけるプログラムに関するコンセンサスをもう一度見つめなおし、プログラムの知的財産保護の本質を探り、今後の制度設計に一定の視座を与えることを目的とする。

1-2　プログラムの特徴

プログラムの特徴として知的財産法上よく言われることをまとめてみたい。

プログラムというのは、コンピュータに対する指令の組み合わせであり、プログラム自体が人の行為を介さず、一定の機能を果たす。だからプログラムは通常人間の理解で

きない表現で表されるし、人間が理解できなくてもかまわない。とはいえ、指令をプログラム言語（ソースコードやオブジェクトコード）で表しているところに「表現」の側面がある。

重要なことは表現よりもコンピュータに対する指令やその選択、組み合わせ、すなわち機能や技術思想であり、いかにコンピュータを作動させるかということに価値がある。そして、プログラムは機能性や効率性のみが重要となるから、その表現は自動的、機械的に、既存の常套的方法で行われる。

互換性を維持する等のためにどうしてもそのような表現にならざるを得ない、「効率性を追求するとどうしてもそのようなステップを踏まざるを得ないという[1]」ことがある。またある種のプログラムにおいては標準化ということがきわめて重要な意味を有する。

また、プログラムは一般的に技術的、機能的著作物と言われる。そこで技術的、機能的著作物の一般的特徴と捉えられているところも検討する価値があるのでまとめてみよう。

「機能あるいは用途に想定された作品については、…選択肢というのはかなりありうるとしても、合理的な当業者が選択するであろうという表現は、実質的にはかなり制約されているのが通常で[2]」ある。

また、既存の作品の修正や改良による積み重ねで発展していくという性質を有している。そして、効率性等の観点から、一定方向に進歩し、開発の結果は少数の解に収束していくという傾向が見られる。

以上の点について、なんとなくそうかな、と納得する人が多いと思う。私自身もそのように納得する者の一人であるが、ふと次のような疑問がわいてきた。「このような特徴はプログラムの著作権法改正の作業のころから同じように言われているが、プログラムの急速な進歩の中で、変化してはいないのか。」そこで、現在実際にプログラムと関わる人々に、現在のプログラムの開発動向や仕事の進め方を聞き込むことで、プログラムの特徴を改めて見直してみようというのが本稿の最初の作業になるのである。

---

1 田村善之『著作権法概説[第 2 版]』p30（2003 年・有斐閣）
2 小泉直樹「機能的著作物の創作性」著作権研究 No.28（2001）p19（2003 年）

2　プログラム開発の現状[3]

2-1　プログラム開発の過程

まず、プログラムがどのように開発されるのかについて概観したいと思う。開発に関して、「一昔前まではウォーターフォールという工程が多かった。現在は反復開発プロセス（iterative development）が主流になりつつある[4]。」これらを簡単に概観しておきたい。

2-1.1　ウォーターフォールモデル

プログラムの開発の第一段階として、その顧客の要望が何かということをまとめる。この段階を要件管理という。開発すべきソフトウエアがどのような機能を備えるかということを管理するのである。

次に、その要件を備えるように基本設計が行われる。システムがどのような動きするかを決める。どのようなセキュリティをどのように達成するかも同時に決める。

基本設計が済むと、次に詳細設計に入る。基本設計を達成するためどのようなアルゴリズムを作るかという段階である。このアルゴリズムの開発がソフトウエアの性能や速度に大きな影響を与える。

詳細設計がなされると、それを実装する作業に移る。コーディングの段階である。コーディングを終えるとそれがきちんと動作しているかをテストする段階に移る。試験を経て品質が保証された段階で出荷となる。

以上のように、要件管理、基本設計、詳細設計、実装、テスト、という流れを順番に行う制作プロセスが縦に流れる滝に捉えられ、ウォーターフォールモデルと呼ばれている[5]。

2-1.2　反復開発モデル

一昔前はこのような工程が多かったが、現在は反復開発プロセスという制作プロセスが主流になりつつある。これは各フェーズを行ったりきたりしながら改良を重ねつつ製

---

[3] この章の多くは実務家への聞き込みをもとにしている。多くの方々にヒアリングやメールを通して貴重なアドバイスを頂いた。この場を借りてお礼申し上げる。
[4] サイボウズ株式会社・山本泰宇氏<hyamamoto@cybozu.co.jp>
[5] 上記開発工程のうち、基本設計までにあたる部分を上流過程と呼び、それ以下を下流過程と呼んだりもする。

品を開発していく工程のことを言う。例えば詳細設計をしたら実装にすぐ移るのではなく、基本設計に立ち戻って再び見直し、実装をしてもそれが詳細設計にあっているかを常にフィードバックしながら次のフェーズへ進んでいくといったものである[6]。

反復開発モデルにおいて一番重要なことは「変更の管理」である。すなわちある箇所の設計や仕様を変更した場合、どこに影響が及ぶかを突き止めるということである。

2-1.3　両者の違い[7]

両者の違いは一体どこにあるか。それはウォーターフォールモデルは手戻りを考えていないという点にある。これにより、ウォーターフォールモデルでは各フェーズにおいて完璧性が求められるのである。しかし現在、ソフトウエアの開発は複雑化しており、最初から要求を完璧に満たす設計は難しく、それを実装後まで評価しないと失敗が多くなってしまう。この点を克服したのが反復開発モデルである。後述するようにオブジェクト指向といわれる現在主流になりつつあるソフトウエア開発はシステムを「モノ」単位で捉え、それらのいわばソフトウエアの部品を組み合わせて開発を進める手法をとっており、たとえ開発段階が進んだ後でも一部の部品を少し改良したいとか部品相互の組み合わせ方で実装段階から設計に一度立ち返って考えたいというような要求があったとき、問題箇所を部品単位で局所化することができることで「変更の管理」を容易にする。反復開発プロセスはこのようにオブジェクト指向と親和性を持っていると言える。

2-2　オブジェクト指向の理解

2-2.1　オブジェクト指向とは

プログラム開発の変化は「オブジェクト指向」という言葉に表されるようである[8]。これまでのように開発対象のシステムを「手続き」という単位で分割し組み合わせていたのと異なり、「オブジェクト指向とは、システムを『オブジェクト』、つまり『モノ』を単位にとらえようとする考え方[9]」である。「モノ」と捉えることにより、現実世界によ

---

[6] 分析、設計、実装のサイクルを短めに繰り返す「スパイラル・アプローチ」もこの分類に入る。

[7] http://www.atmarkit.co.jp/fjava/devs/renew_uml10/renew_uml10.html を参考にした。

[8] 「オブジェクト指向開発が根付いてきた」(「特集１　銀の弾丸？ただのお絵かき？　ＵＭＬの真実を探る」日経バイト 2002 年 8 月号 p95）またヒアリングにおいてもプログラムの技術開発動向として大きな変化はオブジェクト指向開発への移行にあるという声が多かった。

[9] 前掲注 8・日経バイト p62

り近い形でシステムを開発できるようになる。

この言葉は実に多義的であり、プログラミングにおけるオブジェクト指向と、もっと広くプログラム開発そのもののオブジェクト指向の二つに大きくは分けられる。

2-2.2　オブジェクト指向プログラミング

プログラミングパラダイムという言葉がある。「計算機を用いて問題を解く際に何に着目して問題を整理し、何をもとにプログラムを組み立てるかについての規範（paradigm）を意味している[10]。」

プログラミングパラダイムには手続き型、関数型、論理型、オブジェクト指向型などがある。手続き型とは、問題を解く手順を手続きとして表し、その手続きを（命令）にそって処理を進める方式である。関数型は、関数を基本要素としてプログラミングを行う方式で、手続き型のような計算の手順や過程の記述は不要になる。論理型は述語論理[11]の概念を基礎とし、論理式や制約条件を記述することでプログラミングを行う方式である。

オブジェクト指向は前 3 者とは次元の異なるプログラミングパラダイムである。前 3 者のプログラミングは、コンピュータの動きに沿ってどうデータを動かすかというように「手続き」を単位にして記述しなくてはならなかったのに対し、オブジェクト指向プログラミングでは、現実の世界を基準に「こういうデータをこういう手続きで動かす」というように、データ＋手続きのセット（データ構造）をあらかじめきめる、いわば、機械や部品のような「モノ」（＝オブジェクト）の単位で、その組み合わせを記述していくのである。

オブジェクト指向プログラミングの大きな特徴は、プログラムを部品化できるところにあると言える。これまでのプログラミングでは、新たなプログラムを作るときには、たとえ以前に同じような機能のサブルーチンがあっても、一から作っていく必要があったのに対し、オブジェクト指向プログラミングでは、以前作ったプログラムを部品として再び使うことができる。そうすると今まで、プログラムの一部を改変しようとすると、全て書き直さなくてはならなかったのが、あたかも車のタイヤ交換のように、改変したい部分だけを交換できるようになるのである。

---

[10] 河村一樹『図解雑学　コンピュータ科学の基礎』p104（2003 年・ナツメ社）
[11] 命題があらわす事象の関係や性質に注目した論理。変数の具体的な値を与えることでその真偽性が定まるもの。例えば「ソクラテスは人間である」の「人間である」という述語を関数（H）とすると H(s),s=ソクラテスとあらわせる。（前掲注 10・河村 p64）

両パラダイムの違いは次のように例えられる。

「ソフトウェアの世界でエレベータに乗るということを考えた場合、いったいどのようなことになるのでしょうか。従来のソフトウェアの世界では、まずエレベータのサブルーチンが初期化されます。初期化に従って 1 階にいるとか 10 階にいるとかが決まり、サブルーチンはその状態に従わなければなりません。つまり、エレベータそのものの動きを使う人が知っていなければ、ソフトウェアの世界ではエレベータを使うことができないのです。また、既存のサブルーチンを寄せ集めて何かをするということもできず、何かをしたいときは新しくサブルーチンを作って初期化して使わなければならないのです。…つまり、エレベータに乗りたいときエレベータの設計・製造から起動方法に関する記述まですべてを行わなければ、エレベータに乗ることはできませんでした。

しかし、本来の目的はただエレベータに乗りたいわけですから、私たちの世界のように、エレベータに関する設計や製造などは考えずに、誰かが作っておいてくれているもの（エレベータそのもの：筆者注）を利用すればそれを呼ぶだけでエレベータに乗ることはできるはずです。

…ソフトウェアについても、もっと人間に近いレベルで考えたほうが便利です。そのアプローチのひとつがオブジェクト指向なのです。これは、いつも人間が物を見ている見方と適合するものなのです[12]。」

オブジェクト指向プログラミングにおいては、メインのソースコードは個々のオブジェクトを起動する指令を記述することになる。それぞれの部品でどのような処理を行うか、コーディングをするかは必ずしもその本人の感知するところではない。（後述のカプセル化も参照）

オブジェクト指向プログラミングは、昭和 60 年の著作権法改正のころ、すでに存在していた。しかし、オブジェクト指向型のプログラムは従来のプログラムよりも沢山の処理を実行するため、昔のコンピュータでは処理が遅くて使えなかったと言われている。しかし、コンピュータの飛躍的な進歩により、この問題が解消されると先述のオブジェクト指向プログラミングの特徴が認められるようになってくるのである。

2-2.3　オブジェクト指向プログラミングの手法

オブジェクト指向における実装では、(1)まずクラスを定義して、(2)そのクラスのオブジェクトを生成し、(3)そのオブジェクトに対してメソッドを実行する、という３段階の

[12] http://www.s34.co.jp/cpptechdoc/article/oo/index.html#s3

手順を踏む[13]。

クラスとはオブジェクトに共通したデータと手続きをテンプレート化したものをさす。クラスは抽象的な部品、オブジェクトは具体化された部品とでも捉えられる。例えばエンジンがクラスであれば HONDAV12 エンジンはオブジェクトと言えよう。

プログラムはメインのプログラムとクラスのプログラムの組み合わせとなる。メインのプログラムにおいては、プログラムの全体的な作業の流れを示し、それぞれのクラスにおいてはデータと手続きが記され、そこからオブジェクトを作成し、他のオブジェクトと相互に作用しながら、結果を取り出す。それぞれのオブジェクトは機能ごとに独立している。

この点手続き型においては、メインのプログラムは同じだが、機能を果たすために手続きの一連の流れを記述する。それぞれの手続きは、一連の流れの中にコミットしており、独立できないため、一部の改変は全部の確認を要することになる。

手続き型プログラミングとの違いに関する実例は例えば、http://www.microsoft.com/japan/msdn/net/books/vbnet_uml/chapter1.asp 等でわかりやすく示されている。

2-2.4　オブジェクト指向開発

前章で見たように、顧客のニーズが複雑になり、反復開発プロセスが重視されるようになると、このオブジェクト指向プログラミングはそれに整合的であることが分かる[14]。なぜなら、修正の繰り返しや「変更の管理」が容易だからである。こうしてオブジェクト指向プログラミングが発展を遂げるのと同時に、ソフトウエア開発の上流過程からオブジェクト指向にしようという動きが当然のように出てくる。例えば 1990 年の OOPSLA(Object-Oriented Programming, Systems, Languages, and Applications)というオブジェクト指向技術に関する国際会議において、上流工程におけるオブジェクト指向分析、設計にたいする肯定的な結論が出されている[15]。その後、分析の手順や UML のような表記法が提唱され発展した。

---

[13] http://www.microsoft.com/japan/msdn/net/books/vbnet_uml/chapter1.asp

[14] 反復開発プロセスにおいて最も重要になる「変更の管理」を容易にするため、オブジェクト指向は問題箇所の局所化という意味で貢献するという意味で整合的であるが、両者が直接的な関係にあるというわけではない。「変更の管理」自体は、そのためのシステムやツールを通じて実現されるものである。

[15] 青山幹雄他『オブジェクト指向に強くなる－ソフトウエア開発の必須技術』（2003 年・技術評論社）

2-2.5　オブジェクト指向のキーワード

以上のような「オブジェクト指向」の技術的特徴をいくつかのキーワードで理解しておきたい。

カプセル化

抽象化

継承

カプセル化

カプセル化とは、データとそれを処理する手続きをまとめてプログラムの単位とすることである。これがオブジェクトとなる。

オブジェクト指向ではデータを属性と呼び、手続きをメソッドと呼ぶ。カプセル化の仕組みにおいては、外部から、オブジェクト内のデータへ直接アクセスすることを避けるようにできる。メソッドを通してのみアクセスできるようにして、データを隠蔽できるのである。また、メソッドのなかでも外部に公開する情報は必要最小限に抑え、他のプログラムとの独立性を高めることができる。

カプセル化により､プログラムのインターフェースとオブジェクト内の実装を分離し、プログラムの外にはインターフェースのみ公開できるようになった。これにより、オブジェクトを利用する人はその内部の実装を知らなくてもプログラミングに利用できるのである。

抽象化

オブジェクト指向は、プログラムを「モノ」単位で見ていこうという考え方だが、単位の分け方が次の問題となる。このときの基本的な考え方は、対象世界にどれだけ違うものが存在するかということである。同じ種類のものはできるだけ少ない種類にまとめることができると分類が簡潔になるように、同じ種類のオブジェクトをまとめて分類することが必要である。同じ種類のものを集めると、全てに共通する部分はもちろんあるものの、それぞれ異なる部分もある。電話機の例で言えば、抽象的には電話機は電話番号を含むものとして共通であるが、個々に電話番号をもつと言った具合である。この抽象的な共通部分を表すものを「クラス」と呼び、そのクラスに属しつつ個々に異なる部分をオブジェクトと呼ぶ。

継承

クラスの概念が理解できると、この共通部分をそのまま利用して、新たな機能を追加

できるようにし、効率よく開発が進められるようになる。このようにクラスを元に、共通部分を引き継いで新たなクラスを作り出す過程を「継承」という。この考え方に基づくと、クラスは階層的な構造を作り、頂点には共通部分を抽象的に表したもの（抽象クラス）がくることになる。このクラスでは、インターフェースや実装の雛形だけが示され、ソフトウエアの構造を規定することができる。この抽象クラスを元に継承を通して実行可能なクラスが生成されていくのである。

2-2.6　オブジェクト指向による再利用可能性[16]

ソフトウエアに対する要求と複雑度は飛躍的に増加し、開発期間の短縮もあり、ソフトウエア開発のリスクとコストは高まっている。そしてソフトウエアの再利用は必要不可欠になっている。オブジェクト指向の継承やカプセル化といった特徴はソフトウエアの再利用の可能性を増大させた。ところが、オブジェクト指向をただ導入しただけで再利用性が向上するわけではない。そこで、技術開発においてもソフトウエアの部品化と再利用を実現するための仕組みや研究開発が進められた。この中からコンポーネント指向開発とよばれるような部品化、再利用技術等が実用化されている。

コンポーネントとは、ある機能を実現するために、複数のオブジェクトから構成されたものである[17]。このようなコンポーネントを用いて、より効率的にソフトウエアの作成を行おうというのがコンポーネント指向開発である。

コンポーネント指向開発の特色はソースコードを見ずにそのままオブジェクトコードを再利用できるということである。ブラックボックス型再利用とも言われる。もちろん外部との接続、他のコンポーネントとの組み合わせが必要であるので、インターフェース情報を知る仕組みは備えている。

このようなブラックボックス型再利用においてはコンパイルすらする必要もなく、コスト的にも非常に有益であり、再利用の究極形とも言える。ブラックボックス型再利用では、コンポーネントとして提供されたクラスやその集合を継承したり上書きしたりせずそのまま使うことができる。コンポーネントは互いに独立性の高い部品であり、利用

---

[16] 前掲注 15・青山 p94

[17] コンポーネントという言葉の定義自体は多義的であり、およそ「再利用可能なもの」という意味で広く使われる。手続き型プログラミング言語における関数などのモジュールやそのまとまりをさす場合もあり、オブジェクト指向においても、アーキテクチャ、フレームワークにまで拡大して意味する場合もあるが、ここで指摘しているコンポーネントはブラックボックス型の再利用が可能なソフトウエア部品体系をさしているため、区別して考える必要がある。ここではコンポーネント指向開発の説明の便宜上本文のような定義としておきたい。

者はこれを組み合わせて自分の要求に沿ったソフトウエアを作ることができる。

なお本稿で「部品」という言葉を使うことが多いが、これはコンポーネントも個々のオブジェクトもさしうる用語として用いている。意味としては「オブジェクト指向で作られたソフトウエアを構成する機能単位に分割された要素」とでも捉えられよう。

2-2.7　オブジェクト指向の普及度

最近オブジェクト指向プログラミング言語であるＪａｖａやＣ＋＋が普及し、オブジェクト指向分析、設計のモデル言語であるＵＭＬが広まってきてはいるものの、実際の開発の現場では必ずしもオブジェクト指向というものが完全に浸透しているというわけではない。

理由は、オブジェクト指向で開発していなかった既存のシステムを一から作り直さなくてはいけないこと(システム移行費用がかかってしまうこと)、従来の技術者にとって、オブジェクト指向という発想法が難しいこと、うまくシステムを部品化できないこと[18]、などがあげられている。

オブジェクト指向というのは技術そのものではなく、発想法にすぎないため、やはり、最後は開発者の力量次第ということになるのである。よってそのような発想法、それを体現可能なプログラミング言語や分析・設計言語が普及したとしても、上記のような特徴を使いこなし、効率的で品質の良いソフトウエアを作るのはなかなか困難なようである。

しかし、着実にオブジェクト指向を活かした開発を進めていこうという企業も多数出てきており、今後も、オブジェクト指向のプログラム製品は増えていくものと思われる。

2-3　ネットワーク環境の整備

ネットワーク環境の整備がプログラミングにも大きな影響を与えている。

すなわち、ネットワーク環境の整備により、ソフトウエア開発において全ての構成要素がひとつのところに集まっている必要がなくなるということである[19]。プログラム開

---

[18] 自分の実現したい機能をどのように部品化し、どのように抽象化すればより効率的かということを考えること、すなわちオブジェクトやコンポーネントの設定の仕方（その抽象化の仕方やサイズ）が難しいと言われている。

[19] 例えば、Windows のパソコンにはマイコンピュータとマイネットワークがあり、自分のパソコンの中にあるファイルも他のコンピュータの中にあるファイルやフォルダも同じように自由にアクセスすることができる。これをソフトウエア開発に当てはめて考えてみると本文のようなことが言えるのである。

発においても､ハードウエアの構成を知らなくても開発できるプログラムが多く存在し、ネットワークを通してひとつの機能を有するプログラムができあがる場合も多い。今後もハードウエアを意識しない方向に進んでいくであろうことが予想される。

オブジェクト指向プログラミングの文脈でこれを捉えると、全ての部品がひとつのコンピュータ内に存在する必要がなく、どのようにハードウエアがプログラムと関わるかということは重要性があまりないということになる。

このことを先述の特徴とあわせて考えてみると、オブジェクト指向プログラミングのプログラムは、手続き型プログラミングとは異なり、部品ごとに自分で好きなものを取ってこられる。よって他人Ａさんのコンピュータ上のプログラムを利用しても、他人Ｂさんのコンピュータ上のプログラムを利用してもよい。しかもその中身のコードは知る必要もなく、しっかり機能してくれればそれで良いのである。

2-4　2 のまとめ

ソフトウエア開発の複雑化に応じて、現在の開発プロセスは分析、設計、実装等の各段階を反復しながら進めていくモデルに変わってきている。このプロセスにおいて重要なのはいわゆる「変更の管理」ということであった。オブジェクト指向はそのプロセスに問題の局所化という形で貢献しうるものであった。

オブジェクト指向プログラミングに変わったことにより、まずプログラムの構造が変わった。手続きを指令する言葉の羅列から、どの部品をどう組み合わせればよいかという指令の仕方に変わった。このことにより、ある機能を果たすにはどうすればよいかという観点から、どのように全体のシステムをより小さな部品に区分けしたらよいかを考え、適切な部品（その中身の手続きは必ずしも自分でプログラミングする必要はない）を探していくという開発手法が可能になった。部品の組み合わせという概念はプログラムに限らず、分析、設計段階にも広がっている。ソフトウエア開発は全体としてオブジェクト指向が主流になりつつある[20]。

このような開発プロセス、プログラミングの変化によって、プログラムの再利用可能性が高まり、必ずしも全てを一人でプログラミングしなくてもよくなった。極端な話をすれば、あるソフトウエアを開発する場合、部品の呼び出しの順序を記述したメインのプログラムを最低限作れば、その中で利用する部品はその機能さえ知っていれば、全て他人が作ったものでもかまわなくなったのである。しかもその部品の中身のコードがどうなっているかも分からなくてもよくなっている。この点が今までのプログラミングと

[20] この点について、2-2.7 で述べたように、普及への障害も多く、異論もある。

はまったく異なるところであり、著作物としての表現に当たるか否か、他のどのような類型に当たるのかという考慮が必要になってくる。

3　プログラムの特徴再考

以上のようにプログラム技術のパラダイム転換とも言えることが起きているのであるが､著作権法でプログラムを論じるときのプログラムの捉え方はあまり変化していない。変化させる必要がないとしても、それを主張する文献も見当たらないので、この点について検証することは有用である。また、特徴の中でも、抽象的過ぎてよく分からないことが多いこともある。それも含めて理解を深められればと思う。

「はじめに」の部分で見たことを参考に以下の７点についてそれぞれもう一度検証してみよう。①指令をプログラムのコードで表しているところに表現の要素があるということ、②機能のほうが大事ということ、③表現は自動的、機械的に既存の常套的方法で行われるということ、④標準化のためや互換性を維持するためにどうしても踏まざるを得ない表現があるということ、⑤合理的な当業者が選択するであろうという表現はかなり制限されるということ、⑥既存の作品の修正や改良の積み重ねで発展していくということ、⑦一定方向に収束していく傾向にあるということ、①から④はプログラム特有とも言える特徴、⑤から⑦は技術的著作物の一般論として言われることである。⑤から⑦に関しては、プログラムも技術のひとつであるから、この一般論に当てはめるとどうかという観点から検討する。

①表現の要素があること、②機能の方が大事ということ

オブジェクト指向プログラミングにおけるプログラムも言語に変わりはなく、そのソースコードは表現の要素を当然有する。しかし手続き型プログラミングとは表現のあり方がまったく異なる。

手続き型プログラミングが、アルゴリズムの達成のための指令を直接記述しているのに対して､オブジェクト指向プログラミングは､オブジェクトの組み合わせを指令する。言い換えると、手続き型はプログラムの機能の手順を一から記述しているのに対して、オブジェクト指向はある機能を果たすために必要な機能の組み合わせ方を記述していると言える。

ただし、注意しなければならないのは、プログラムはコンピュータに対する指令であり、興味があるのは、いかにコンピュータを利用して課題を解決するかということなのである。したがって当然表現よりも機能の方が相対的には大事ということになる。

③表現は自動的、機械的に既存の常套的方法で行われるということ

コーディングの内容は確かにそのプログラムの設計が終わった段階である程度決まっているものではあるが、それが果たして機械的、自動的に行われるものかは必ずしもそ

うではないように思われる。

この点につき、コーディングについて選択肢が無数にあり、バグが生じないように相当な労力が費やされるとしても、その選択は機械的に行われ、“ 額に汗 ”が必要なだけにすぎないということになるという意見もある。

しかし指令の選択や組み合わせが決まっても表現において何らかの工夫が必要である点で機械的、自動的ということはできないのではないか。このように考えれば、このコーディングの作業は “ 額に汗 ” という言葉で片付けられないように思われる。

この考え方の違いは表現の幅を考える際にも違いとなって現れてくるとも思われる。（5-2 参照）

④標準化のためや互換性を維持するためにどうしても踏まざるを得ない表現があるということ

インターフェースの部分など、標準化や互換性維持のためどうしても踏まざるを得ないステップ、表現があると言われるが、この点に関しては著作権法第 10 条第 3 項第 2 号の「規約」に当たり、そもそも著作権の保護対象外であるとも言える。しかし、保護対象外とはされないとしても、互換性を維持するために表現が制限されてしまうのは当然のことなのかもしれない。

オブジェクト指向プログラミングにおいても同様のことが言えると思われる。

ただし、表現として捉えるレベルが変わっているため、組み合わせる対象は互換性が達成されていることが前提となっており、インターフェース部分の表現の幅がないことがプログラムの表現の幅に直接に影響するとは必ずしも言えない。オブジェクト指向プログラムの部品においてインターフェースが共通化されていれば、中身の表現はまったくのフリーだからである。

⑤合理的な当業者が選択するであろうという表現はかなり制限されるということ

技術的、機能的著作物は、無意味な改変等を除くとかなり表現の選択肢は限られてくると言われている。

特に手続き指向型プログラミングでは全ての手続きが目に見える状態で示され、蓄積されていき、当業者が、表現の多様な選択肢の中、比較的到達しやすい表現というものが生じる可能性がある。

この手続きのレベルのプログラムはオブジェクト指向プログラミングにおいては、ほとんどがメインのプログラムのプログラマにとっては隠蔽されたものとなっており、蓄積されるものではない上に、プログラマが主体的に表現するというようなものでもない。

オブジェクト指向プログラミングの関心事は、個々の指令の組み合わせよりも、個々

の指令を組み合わせてできた部品の組み合わせにあると言える。そしてその組み合わせは多様にあると考えられる。既存の技術を見れば、同じものは作りやすいという意味で到達しやすいという面はあるかもしれない。しかし、そのことが表現の多様な選択肢を狭めるということにはならないと思われる。

⑥既存の作品の修正や改良の積み重ねで発展していくということ

しばしば、著作権法の教科書等では機能的著作物は、既存の著作物の修正や改良をすることで発展していくということが言われている。

既存のプログラムに新たに指令を付け加えるという形で積み増すと捉えがちだが、かつての手続き型プログラミングにおいては、プログラムの指令の付け加えは全てのプログラムの見直しを意味していた[21]ことからすると、結果的に表現が類似するところはあるものの、全体として一からプログラミングすると評価できよう。

他方、オブジェクト指向プログラミングにおいては、部品ごとの組み換えが可能であり、より積み重ね的要素は多いと言える。ただ、部品ごとの組み換えが容易にできることにより、表現の幅が狭まるということはないであろう。

⑦一定方向に収束していく傾向にあるということ

「およそ技術というものは、より優れた効果、効用をもつかどうかということでその価値を判断される。そして技術は客観的であり、誰が追試しても同じものができ、それを基礎としてより優れた技術の開発にしのぎが削られる。開発の結果、行き着くところは、誰が開発してもひとつあるいは少数の解に収束するという性質を有している[22]。」と言われる。

技術イコール収斂というこの考え方は、個性の多様性を保護する著作権法とは相容れないという考え方にもつながれば、このようなものも著作物で保護することになったのだから、著作権法も産業保護の法律としてシフトする必要があるという考え方にもつながる。

どちらの意見をとるにしてもここで一度考えておかねばならないのは、ここで言われる「技術」にプログラムが含まれると仮定すると、プログラムにおいても本当に収斂する方向にあると言えるのかということである。

プログラムの技術の収斂とはまさに、ある機能を果たすプログラムを作ろうとすると、それがより改良されていくほど、その効率性、機能性を達成するためのコーディングが

---

[21] この点を「部品化」による問題解決の局所化で解決したのがオブジェクト指向プログラミングであった。

[22] 中山信弘『マルチメディアと著作権』p42（岩波新書・1996 年）

一様化されてくるということと思われる。(であるからこそ効率性、機能性を考慮した上での表現の幅が問題になるのである。) この点につき技術の現場の方の声を聞いてみた。

ヒアリングにおいて「ある機能を果たすプログラムを制作するとき、コーディングの仕方は無数にあるのか、効率性等を考えた場合はどうか？」という質問に対し、「目的に対する手段がいくらでもあるのと同じように色々ある。」という答えを頂いた。「オブジェクト指向開発において、オブジェクトの組み合わせ方、手順を決めてしまえば実装はほぼ１対１対応のコーディングになるのでしょうか？ 」という質問に関しても「１対ｎです。」また、「ひとつの機能を果たすにはひとつの記述だけではない」等の声も聞かれる。他方でアプリケーションのプログラムではバリエーションが相当あるが、OS のプログラムに関しては OS として機能するためのさまざまな制約があり、相対的に表現の幅が限られる[23]という声も聞かれる。

また、ある機能、しかも同程度の機能性を有するプログラムのソースコードの表現方法にバリエーションがあるということの例は例えば、サイボウズ事件[24]において問題になった GUI（グラフィック・ユーザー・インターフェース）のソースコードの違いからも現れている。まさに機能的な部分である GUI の表現は顧客が混同するほど被疑侵害品と似ていたが、ソースコードはほとんど異なっていたと言う。そして、GUI を見れば、同一の GUI を有するソフトウエアを作るのは簡単だと言われているのである。

少なくとも、技術者の間で、機能の高まりが必ずしもプログラムの表現の幅に影響を与えるというようには考えられていないと思われる[25]。既存の特徴認識とは異なる実態がここには現れているように思われる。

以上のように検証してくると次のようなプログラムの特徴認識が必要であるように思われる。

プログラムにおいてはその機能が重要であるが、表現の要素を有する。表現と比べるとその機能の重要性が相対的に高い。手続き型プログラミングにおいては、指令の手順を一から記述していたというところから表現的要素を捉えやすかったが、オブジェクト指向プログラミングによるプログラムは、部品の組み合わせ方という形で表現がなされ、

---

[23] かといって全体の表現を見たらやはりまったく同じになるということはない。表現の幅がなくなるというわけではない。

[24] 東京地判平成 14 年 9 月 5 日　最高裁ホームページ

[25] 表現の幅という言葉を使うとき、記述の同等性をどこまで見るかという問題は常に付きまとう。ソフトウエアのコードは何千行、何万行にも及び、まったく同じコードになることはないからである。記述の同等性の判断は機能の同等性にも影響するため、プログラムの「表現の幅」という意味の詳細な検討が今後必要になろう。(関連→5,7)

部品の中身の表現についてはこだわらないというように、これまで以上にプログラムの機能的要素を鮮明にした。

技術思想の積み重ねがあり、効率性、機能性の向上が目指されるとしても、プログラムの表現の多様性は必ずしも収斂する方向にはない。

4　オブジェクト指向プログラミングによるプログラムの性質決定

前章で見たようにプログラムの特徴を明確にした上で、知的財産法上、プログラムをどのように捉えればよいか、性質決定をしてみたい。ここでは、著作物としての捉え方、設計図としての捉え方、編集著作物としての捉え方、機械としての捉え方を考えてみたい。

オブジェクト指向プログラミングによるプログラムといってもそのあり方は多様でありひとまとめにして語るのは危険ではあるが、ここでは特に指定しない限り、オブジェクト指向分析、設計を経て作成されたオブジェクト指向プログラミングによるメインのプログラムということにする。

メインのプログラムとは、色々と捉え方はあると思うが、これまでの手続き型プログラミングによるプログラムとの違いをはっきりと表すために、そのプログラムが果たそうとする機能を実現するためにどのような部品（カプセル化されて中身を隠蔽された形のオブジェクト）をどのように組み合わせるかということを指令するプログラムと考える[26]。

4-1　著作物

これまでの手続き型プログラミングにおけるプログラムは、コンピュータが実行すべき手続きを指令することを表現するものであり、ある機能を果たそうとするときにどのような指令の組み合わせをすればよいかが記述されていた。そしてその部分がソースコードという文字の形で現れるところが言語著作物に近いために著作権法での保護が決まったと言える[27]。

この点はオブジェクト指向プログラミングにおけるプログラムも同様にソースコードを持っているので、言語著作物類似の著作物として捉えることができる。

ただオブジェクト指向プログラミングにおいては、カプセル化という特徴からも現れているとおり、プログラマは部品の使い方（どう機能するか）さえ知っていれば、部品がどういう手続きで動くか、中身のコードがどうなっているのかを知らなくてもプログラミングが可能である。部品を組み合わせてひとつのプログラムを作り上げていく作業

---

[26] このメインのプログラムがオブジェクトの相互作用を指令し、ひとつの機能を果たす。（2-2.3 参照）

[27] 前掲注 22・中山（マルチ）p54 も「昭和 60（1985）年の著作権法改正により、プログラムが新たに著作物の例示に加えられた（10 条 1 項 9 号）。立法者は、単に従来から存在する言語著作物に類似したものとして、新たにプログラムを付け加えただけであ」るとしている。

は、手続き型プログラミングのようにコーディングによりいかに効率よくコンピュータを動かせるかという問題関心と比べて、プログラムの表現（コーディング）に関する著作者の関心の質が異なるように思われる。

これまではプログラマのアイデアの直接的表現としてコーディングは位置づけられていた。ソースコードを自ら書き、それをコンパイルしたものでコンピュータを動かしていたからだ。ところが、オブジェクト指向においては、上述のとおり、個々の機能を果たす部品のコードは隠されており、必ずしもそれを知らなくても開発が可能であり、全ての部品を一から制作する必要がない。むしろ、既存の部品を再利用し、その組み合わせをしてソフトウエアを制作する。つまり、プログラマに、いかにうまくオブジェクトの組み合わせをしてもらうかというところを重視するため、出来上がったプログラムの全体としてのコードは、そのプログラマの独自の創作物としてはあまり意味がない[28]。ブラックボックス型再利用ということを考えてみるとその点はより明確に理解できると思われる。

オブジェクト指向プログラミングにおけるプログラムでは、プログラマの創作物として現れる表現はシステム動かすための部品の組み合わせ等なのである。

このようにオブジェクト指向プログラミングは手続き型プログラミングと体系がまったく異なるため、その表現するものが異なっていることに注意する必要がある。そしてアイデアと表現の関係や創作性に、手続き型プログラミングのプログラムとは異なる考慮を要することになる。（5 参照）

4-2　設計図

設計図は著作物として認められている。著作権法第 10 条第 1 項第 6 号の学術的な性質を有する図面、その他の図形の著作物に該当するとされている。ただし、「設計図は、そのための基本的訓練を受けたものであれば、誰でも理解できる共通のルールに従って表現されているのが通常であり、その表現方法そのものに独創性を見出す余地はな」い（スモーキングスタンド事件[29]）とされるようにその保護範囲は非常に狭い上、設計図から読み取れる具体的な機械やそのデザイン、機能は「設計図との関係で言えば表現の対象である思想又はアイデア」（同判例）である。

---

[28] もちろんそれぞれの部品（オブジェクト）はメソッド（手続き、指令の組み合わせ）を含んでいるため、これらは著作物性が認められうる。このようなオブジェクトの制作には旧来のコーディングと同じような観点からの考慮が可能である。

[29] 東京地判平成 9 年 4 月 25 日判時 1605 号 136 ページ

設計図の表現をどのように捉えるかは争いがあり、大きく分けて、表現内容にまで踏み込まずに捉える捉え方[30]と、設計図の表現の内容の構造や形状をも表現と捉え、その創作性を判断するという捉え方[31]の二つがある。

これは、アイデアと表現の境目の難しさを示している。

この点につきオブジェクト指向プログラミングに基づくプログラムは、部品を使う順番や部品を使う、使わないということの判定をさせる手順を規定する必要がある。部品の組み合わせの記述ということで､設計図に近い役割を果たしていると言える｡しかし、ある機能を果たすための機械をつくるために書かれるという性質を持つ設計図とは異なり、この書かれたプログラムそのものが機能をも果たすのである。すなわち、設計図はその後それを見て人が機械を作るのだが、プログラムは設計図のような記述そのものが機能を果たすというまさに機械と同じような役割を持っている[32]。

むしろ実装前の詳細設計の段階で作られる、UML 等によるモデル図を設計図と捉え、実装は機械そのものを制作する段階と捉える方が自然のように思われる。よって、表現という構成から著作物としての検討は可能であるにしても、設計図と捉えることはできなそうである。

4-3　編集著作物

編集著作物は素材の選択・配列に創作性があれば、著作物として保護されるのが一般的なルールである（著作権法第 12 条)。

これに関しては、その解釈に争いがある。素材の選択・配列はどのような形態の著作物であっても創作性の判断の要素になるものであり、どのような著作物も創作性のある表現が保護されることに変わりはないという説（確認規定説）と素材の選択・配列は創作性の判断対象であると同時に保護対象でもあると捉え、創作性にある程度高度なレベルを要求しつつ編集体系等、従来アイデアに属すると考えられてきたところ（「アイデア

---

30 前掲注 1・田村 p23 参照

31 「設計図の具体的な表現というのは、単なる表層的な表現だけでなく、形状や寸法や色彩等を伴った具体的な表現であると考えるべきでしょう。」（中山信弘「創作性についての基本的考え方」著作権研究 No28(2000),p9（2003 年））

32 前掲注 22・中山（マルチ）p54「一見すると、プログラムと機械の設計図とは同質のようにも見える」が「設計図は人の行為を介して機械となり、その機械が一定の機能を果たしている」他方プログラムは「プログラム自体が人の行為を介することなく、直ちに一定の機能を果たしている。」すなわち「設計図は人に対する指令であるのに対して、プログラムはコンピュータという機械に対する指令であるという点において異なっている。」

とも表現とも付かないグレイ・ゾーン[33]」）を保護しうるという説（創設規定説[34]）との対立がある。後者の立場においては、まず素材を、編集者が何に注目して編集したかを基準に認定し、利用者の予測可能性、独占権による創作のインセンティブと他者の活動領域の確保の比較衡量をしたうえで、表現とすべきレベルとアイデアとすべきレベルを捉え、表現とすべきレベルの素材の選択・配列の創作性をもって、その素材の選択・配列そのものを「表現」として保護しようというものである。

前者の説においては編集著作物の素材の選択・配列は創作性の判断対象のひとつ[35]であって、保護されるのはそれが現れた表現の部分のみである。表現に現れたアイデア部分を創作性の判断に用いつつ、保護範囲は表現に限定するとも言えようか。

後者の説においては従来の考え方ではアイデアに近いとされた部分に保護を及ぼすという効果をもたらすと言える[36][37]。

オブジェクト指向プログラミングは部品を選択・配列して作り上げるプログラムである。メインのプログラムは部品を素材と捉えると編集著作物と捉えることができそうである。

前者の説に従えば、部品の選択・配列が創作性の判断対象のひとつとなりえるということになる。後者の説に従うと、部品の選択・配列がまさに創作性の判断対象であり、保護対象にもなるということになる。

4-4　機械

プログラムは、コンピュータに対して指令をし、一定の機能を果たすものである。一定の機能を果たすという意味において、そしてそれがプログラムにとって最も重要であるという点において普通の機械と共通項を有している。それは製作過程において、分析、設計、実装というような言葉使いにも現れていると言える。さらにオブジェクト指向プログラミングによるプログラムは手続き型プログラミングによるプログラムにも増して機械と捉えやすい要素があると言える。

---

[33] 横山久芳「編集著作物に関する基礎的考察－職業別電話帳は果たして著作物なのか？－」コピライト 475 号 p5（2000 年）
[34] 前掲注 33・横山 p6、　潮海久雄[判批]ジュリスト 1111 号 p234（1997 年）
[35] 前掲注 1・田村 p85
[36] もちろんこの説においても表現から看取できる範囲でしか素材としては認定しないので、前者の説と大きくかけ離れた結果を生むものではない。
[37] この議論で用いられる「アイデア」という言葉にも微妙な意味の違いがあるように思われる。特に保護対象として唱えられる「従来の考え方ではアイデアに属するとされた部分」というときの「アイデア」は厳密な意味でのアイデア・表現二分論上の「アイデア」とは異なる意味で用いられているように思われる。

手続き型プログラミングによるプログラムはその機能を果たす点において機械的であるが、ソースコードの表現は、機能の手順が仔細に書かれているという形をとっており、ある機械の一連の動作手順を表現したものと捉えることができるため、著作権法上の表現と多少の親和性がある。

他方、オブジェクト指向プログラミングによるプログラムのソースコードは部品の組み合わせが記述されており、表現だけを見ると、機械の設計図のように見えるが、それそのものが機能を果たすという性質を加味して考えると、部品を組み合わせて作った結果物としての機械の外形そのものを表しているように思われる。

さらにブラックボックス化されていて、機械の外形は見えるけれども、それぞれの部品の中身が見えないという点も類似している。

このことをプラモデルの車に例えてみよう。プラモデルの車のモーターの中身はその機能さえ分かっていれば中身がどうなっているかは問わない。であるから、プログラムはちょうど車の外観のように捉えられる。まさに機械を作っているのである。手続き型プログラミングとはここが大きく違う。手続き型プログラミングにおいてはプラモデルの車がどういう仕組みで動くべきかを逐一表記しなくてはならないのである。

オブジェクト指向プログラミングはオブジェクト指向という言葉どおり、オブジェクト＝「モノ」を単位としてプログラムを捉えるというコンセプトが底流にあるので、より機械という有体物に似た感覚で、プログラミングを行うことができる。また開発においても、部品を組み合わせて機能を果たすものを作り上げる点、部品ごとに取り替えたり、改良できたりする点においては、機械の開発と似ていると思われる[38]。

以上のようにオブジェクト指向プログラミングにおけるプログラムは著作物、編集著作物、機械と３つの捉え方が可能である。以下、この違いを念頭に置きつつ、前２者においては著作権法による保護のあり方について、後者では特許法による保護のあり方について考察を加えたいと思う。

[38] ただしたとえ部品の組み合わせという点において機械の開発と似ているとしても、時間的な流れを規定しなくてはならない点において異なっている。

5　著作権法としての保護の検討

プログラムは以上のように、著作物、編集著作物、機械と３つの捉え方ができる。そのうちまず前 2 者の捉え方をした場合、どのような問題があるか。

著作権法においては、前提としてアイデアとは区別された表現が保護されることになる。そしてその表現が創作的であることが要求される。

そこでアイデアと表現の区分けをどう見るか、次に創作性をどのように捉えるかが問題となる。以下順次検討することにしよう。

5-1　アイデアと表現の二分論

プログラムの著作権法上の保護は、「指令の組み合わせの表現」（同法第 2 条第 1 項第 10 号の 2）に与えられる。プログラムにおいては指令の組み合わせが表現に直接現れるため、その表現が指令の組み合わせというアイデアに近いものではないかという問題提起がこれまでずっとなされてきた[39]。オブジェクト指向プログラミングにおいても、その表現が著作権法上の表現に当たるのかという問題が生じるだろう。

オブジェクト指向プログラミングにおけるプログラムの表現を捉えようとすると二通りの捉え方がある。すなわち、部品の選択・配列を示すための表現（メインのプログラムのみに着目）[40]と、部品の中身のコードも含めた全体の表現である。

後者の捉え方では、著作者の関与しないことが多い部品の中身の表現もその人の表現と捉えられるということになる。ここではオブジェクト指向プログラミングにおける著作者の著作と捉えやすい前者の捉え方[41]での表現を中心に検討し、必要に応じて後者の捉え方を紹介するという形にする。

5-1.1　著作物として捉えた場合

著作物として捉えた場合、一方で手続き型プログラミングや、オブジェクト指向の部品のプログラムにおいては、指令の選択・配列の幅を表現の幅として捉えながら、オブ

---

[39] 例えば斉藤博『著作権法』p91（2000 年・有斐閣），

[40] メインのプログラムのみに着目し部品の選択・配列を考えた場合、具体的にどういう部品を使うか（部品の中身）をも含めた意味で部品の選択・配列を考えると、後者の捉え方と近似してくる。ここでは、ある機能を果たす部品でその中身は問わないという意味での部品の選択・配列と捉えるとより両者の区別がつきやすくなるであろう。

[41] 4（p20、前書き部）と同じ捉え方である。

ジェクト指向プログラミングのメインのプログラムでは、それよりもより機能に近い、すなわちアイデアに近い、部品の選択・配列の幅を表現の幅と捉えている。

手続き型プログラミングにおいては、プログラムにある機能を想定し、それを実現するための指令の組み合わせを考える。指令の選択・配列そのものはアイデアであるとして保護されていないという前提がある[42]。そして、指令の選択・配列が決まってから、コーディングという具体的表現の過程を経てはじめてアイデアが表現として具現化される。

オブジェクト指向プログラミングにおいては、全体としての機能を実現するためにまず細かい機能の選択・配列を行い、それぞれの機能を果たす部品の選択・配列をあらわすメインのプログラムを作る。それぞれの部品の中に手続きの要素が含まれ、指令の組み合わせが表現されることになる。メインのプログラムにおける部品の選択・配列は「機能を果たす指令の組み合わせ」の組み合わせ、ということになる[43]。どのように機能を組み合わせて要求される機能を作り出すかということがそこに表現される。

このように選択・配列の対象の大きさの違いはあるものの、プログラムとしてコードの形で現れるということから、どちらもそのコードを表現ということはできる。

ただし、オブジェクト指向プログラミングでは、そのメインのプログラムの表現に具現化されるアイデアがより機能そのものに近くなったという意味で、今までのプログラムとは異なる[44]と言え、それが表現・アイデア二分論との関わりで著作権法上問題となってくるのである。表現とアイデアの境目がより分かりにくくなり、これまで以上に、

---

[42] この指令の選択・配列がアイデアかどうかというところで意見も分かれようが、少なくとも特許法におけるプログラムの定義が「プログラム（電子計算機に対する指令であって、一の結果を得ることができるように組み合わされたものをいう。…）」（同法第 2 条第 4 項）とされていることから、指令の選択・配列そのものは特許法の保護対象とする技術的思想と言うことができると考えられる。もちろん技術的思想イコール著作権法上のアイデアかという点については反対する人もいるかもしれないが、プログラムの指令の選択・配列自体はアイデアに近い存在であることは間違いないと思われる。

[43] コンポーネント指向開発ということも紹介したが、複数のオブジェクトの組み合わせがブラックボックス化されて中身の見えないひとつの部品として提供されることもある。よって、部品の大きさは一様ではない。その部品の大きさを選択するのもプログラム制作のひとつの過程である。

[44] それぞれのプログラミング手法による表現とその選択の幅の違いは以下のように例えられる。手続き型プログラミングによるプログラムは機能を果たすプロセスを逐一説明しており、その説明の文章の書き方が表現の幅と言えた。オブジェクト指向プログラミングにおけるメインのプログラムでは、機能を果たすには何と何をどう組み合わせればできるかが表現され、その幅は機能を果たすには何を選びどう組み合わせられるかということで決まる。プロセスの説明は機能の実現までを具体的に説明しているが、部品の組み合わせは部品の中身の仕組み等が説明されない分、その機能をより簡潔に述べているだけというように思われる。より「抽象的」な記述とでも言えようか。

保護範囲が狭まるという形で影響が現れてくるかもしれない。

5-1.2　編集著作物として捉えた場合

編集著作物として捉えた場合はどうか。編集著作物の捉え方については争いがあることは先述した。それぞれの説に従って検討してみよう。

編集著作物を著作物と同様に扱えばよいという見解（確認規定説）にたつと以下のようになるだろう。この説によると、およそ、あらゆる著作物は素材の選択・配列ということを行っており、編集著作物においては、全体の表現を捉え、その素材の選択・配列が創作性の判断の大きな要素となることを確認しているということになる。

この説に従ってオブジェクト指向プログラミングによるプログラムの保護を考えると、部品の中身を含めたコードを表現と捉えることになろう。メインのプログラムのみで編集著作物の表現と捉えることはないように思われる[45]。そして侵害と疑われる製品について、部品の選択・配列が同じ場合でも、部品の中身も含めた全体の表現をみて侵害を判断することになろう。使っている部品の中身が異なれば、（たとえメインのプログラムのソースコードは類似していても）具体的表現たる全体のソースコードは異なってくるため、その表現の相違により、「再生と言いがたい[46]」と認定され、著作権侵害にならない可能性もあろう[47]。

この場合、ソースコード全体を著作物と捉えた場合と結論は極めて近くなる。編集著作物は、著作物のうち素材の選択・配列に特に着目したものというだけのこととなり、むしろ編集著作物と呼ぶか、著作物と呼ぶかの違いにすぎないということになる[48]。

他方、編集著作物の類型を別個特別の類型（創設規定説）と捉えると以下のようになる。

素材は編集者が何に注目して編集したかを基準に認定される[49]。素材の選択・配列そのものが表現から看取されれば、「表現」とされるという考え方となる。

これをオブジェクト指向のプログラムに当てはめてみると、メインのプログラムにおいて部品の選択・配列ということが表現から看取されれば、それを「表現」と扱うことができる。少なくとも、どのような部品に細分化し、配列したかということは表現から

---

45 メインのプログラムの表現のみを捉える場合、著作物として捉えることになろう。
46 前掲注 1・田村 p85
47 メインのプログラムが同じであれば、翻案となりうる余地はあるかもしれない。
48 前掲注 39・斉藤 p98（注３）参照。また、前掲注 1・田村 p23 も著作物と編集著作物を区別するための「限界線を引くことが困難である」としている。
49 前掲注 33・横山 p6

看取できると言えなくもないため、部品の選択・配列を「表現」と捉えることも可能であろう。この時、部品の組み合わせ方（メインのプログラム）は同じだが、部品の中身が違うというプログラムに対して編集著作権侵害を主張できるかは前者の説よりも積極的に認められやすくなると思われる[50][51]。

5-1.3　5-1 のまとめ

手続き型プログラミングにおいては、指令の組み合わせというプログラム特有の書き方を一人のプログラマが書き記していくという作業に著作物の表現と見やすい要素があったが、オブジェクト指向プログラミングにおけるメインのプログラムは、その表現を著作物と捉えることが可能であるが、機能の組み合わせを表現し、組み合わされる機能（部品）の中身は他人にゆだねられるという点がいかにも編集的であり、プログラマの書き記すことがアイデア（部品の組み合わせ）の提示という形をとるという点においてこれまで以上に表現の保護とアイデアの保護が混同されやすくなっていると指摘できる。

また編集著作物として捉えた場合、説によって表現をどのように捉えるかが変わり、またそれぞれの説の解釈によっても変わってきうる。その点については本稿の目的から外れてくる部分も出てくると思われるので、紹介程度にとどめておきたい。

5-2　創作性（表現の幅）

著作物として保護されるには創作性がなければならないと言われている。

---

[50] ただし「表現から看取される」という言葉を厳格に捉えると、部品の中身のコードを踏まえたうえでの選択・配列が「表現から看取される」ということになり、確認規定説と結論は変わらなくなるということも考えられるのではないか。

[51] しかし注意しなければならないのは、部品の選択・配列そのものはプログラムの機能そのものであるということである。この説の結論はややもするとアイデア保護ということになりかねない。部品の選択・配列を「表現」と呼び、保護範囲を創作の最も価値のあるところに近づけようとすること自体は、より実効的な著作権法の役割を考える上で理解できないでもないが、アイデア保護が国際協定（TRIPs 協定 9 条 2 項等）によっても違反となっているという現実も決して軽視できない。

また、この説ではアイデアに少しにじり寄った部分に保護を与える分、創作性の基準を高く設定すると言う。オブジェクト指向のプログラムにおいて当てはめてみると、プログラムの機能的部分にある程度拡大して「表現」とし、創作性の基準を高くするということになる。これでは特許法と保護対象、保護範囲の両面で肉薄することになる。表現保護法としての著作権法と技術思想保護法の特許法が保護対象と保護範囲を同じくすることの妥当性を再検討する必要があると共に、そこまで著作権法がカバーする必要があるのかということも検討する必要があろう。

プログラム著作物の創作性においては、表現の幅がしばしば問題となる[52]。表現というと、著作権法において何を表現とするかがしばしば問題となるが、ここでは仮にソースコードを表現としてみよう。ところが、このように限定したとしても、手続き型プログラミングにおける表現とオブジェクト指向プログラミングにおける表現は原理そのものが大きく異なる。そこで別々に検討しておく必要がある。

5-2.1　手続き型プログラミングにおける表現

手続き型プログラミングにおいては、ある課題解決のための全ての作業が表現される必要がある。Ｃ言語やＣＯＢＯＬなどの手続き型プログラミング言語が用いられる。従来から著作権法においても問題となっており、研究もこの考え方をベースにして行われているように思われる。プログラムの著作権が問題となったシステムサイエンス事件[53]もＣＯＢＯＬのソースコードが問題となった事案であった。システムサイエンス事件においては被告人の「プログラムの表現は指令の組合わせであるから、たとえ個々の制御コード、出力番地及び出力指令がハードウエアに規制されその処理が同一であっても、その指令の組み合わせが同一になる必然性はない」という主張に対し、「あるプログラムがプログラム著作物の著作権を侵害するものと判断し得るためには、プログラム著作物の指令の組合わせに創作性を認めうる部分があり、かつ後に作成されたプログラムの指令の組合わせがプログラム著作物の創作性を認め得る部分に類似していることが必要であるのは当然であるが…プログラムはこれを表現する記号が極めて限定され、その体系（文法）も厳格であるから、電子計算機を機能させてより効果的に一の結果を得ることを企図すれば、指令の組合わせが必然的に類似することを免れない部分が少なくない」上に機能のほとんどをハードウエアがこなし、本件で問題になったプログラムが相当する作業が限定されていること、そのなかでのルーチンは「ハードウエアに規制されるので本来的に同様の組合わせにならざるを得ないこと」「極めて一般的な指令の組み合わせを採用していること」から「創作性を認めることは困難である」とされた。

そして「プログラムにおける『処理の流れ』自体は、アルゴリズム、すなわち著作権法第 10 条第 3 項第 3 号に規定されている『解法』であって著作物としての保護を受けない部分であるから、プログラムの創作性とは無関係である。」としている。

このシステムサイエンス事件判決は 1989 年の判決であるが、この考え方が現在もプログラム著作物に対する理解の前提になっていると言える。この判決の是非を問うこと、

---

[52] 例えば、前掲注 2・小泉 p16、金井重彦『著作権の基礎知識マルチメディア時代のコンピュータ・プログラム』p21（ぎょうせい・1998 年）
[53] 東京高決平成元年 6 月 20 日判時 1322 号 138 ページ

ここから派生する解釈論についてはその後とても多くの議論がなされているのでここで特別に扱うことはしない。むしろ私の視点はこのようなプログラム著作物とはまったく異なる体系のプログラムが主流になりつつあり、プログラム著作物を捉えるときの前提を見つめなおす必要があるというところにある。

5-2.2　オブジェクト指向プログラミングにおける表現

オブジェクト指向プログラミングでは、ソフトウエアを開発するプログラマは、どの部品を利用するかという順番を記述する。具体的な処理の手続きはそのメインのプログラムとは別の部分に書かれる。すなわち、メインのプログラムと部品としてのプログラムでは記述のされ方が異なると言えよう[54]。

メインのプログラムは部品の組み合わせを実装したものであるが、その実装（表現）は機械的に行われるものではなく、効率性等を勘案しても多様であり、表現の幅は十分にある。（3 参照）

また、部品のプログラムも創作物の一部であるとすれば、そこに含まれるコードも含めひとつのプログラムと捉えることもできる。この場合、部品の使われ方は同じでも、まったく同じ部品を使うということはないであろうから、そこにも表現の幅は生まれてくると言える。

しかしその場合も、先述のようにネットワーク化によりプログラムの部品がひとつのまとまりを呈さなくなった場合、それを表現の一部に加えるのか、また、部品は他人の著作物であることも多いであろうから、その中身の表現の幅をも根拠にして著作権を認めることは適当かどうかという疑問は残る。

この点につき、プログラムを編集著作物と捉えると、表現の幅という意味では、プログラマが直接コミットした部品の選択・配列の幅によって創作性を認定することができる[55]。

5-2.3　5-2 のまとめ

---

[54] もちろん部品は必ずしも手続き型プログラミングのように関数の羅列というわけではなく、ここでもオブジェクト指向プログラミングが可能である。オブジェクト指向はオブジェクトの利用の多重構造となっているからである。より具体的な処理の手順が描かれていると考えればよいのではなかろうか。

[55] ただし、確認規定説が全体の表現を捉えた上での素材の選択・配列を考慮していることと、創設規定説は編集者が注目したものを素材と捉えるということとの違いが、創作性の判断対象の違いに現れてくる可能性がある。(5-1.2 参照)

このようにプログラムのコーディングに関して、コーディングの体系そのものが変化している。これは単なる言語の変化にとどまらない意味を持っている。

というのも以前からプログラムは技術的著作物、機能的著作物と言われていたが、コードの全体に対してプログラマが関わっていた。ところがオブジェクト指向プログラミングにおいては、出来上がったプログラムのコードの中にその著作者が関与しないコードが内蔵されている。著作者たりうるプログラマは部品の選択・配列を決定し、メインのプログラムを作成するが、そのとき、カプセル化により隠蔽された部品の中身のコードにはまったく関与することがないからである。

すると、オブジェクト指向では、創作性の判断において表現の幅の有無を判断する際、著作者たるプログラマの直接関与する部品の選択・配列を示すための表現（メインのプログラムの表現）の幅を判断することもあれば、部品の中身のコードを含めた全体の表現の選択の幅を判断することもある（関連→5-1）。どちらを表現と捉えるにせよ、技術的に優劣をつけられない同等の効率性と機能性を持つプログラムにおいても表現の幅が十分に見出せることがプログラムの特徴認識から確認された。

ただ後者の表現の捉え方において、部品はそれぞれが再利用をしあう部分であり、メインのプログラムのプログラマが直接感知しない場合が多い。メインのプログラマにとっては、部品の選択としてしかその部分を捉えていない。

オブジェクト指向プログラミングは部品化により、多数の人間が関わることを容易にし、部品の再利用を促進し、メインのプログラムのプログラマが全てを一から作り出すことをしなくても良いところが利点であるから、このメインのプログラマの著作者としての側面は部品の中身のコードではなく、どういう部品を設定し、どの部品を選択しどう並べて、それをどう実装するかというところに現れる。

常にそのことを念頭において創作性の判断をする必要があるように思われる。そのためには 5-1 で検討したような、何を表現とするかという議論が大きく影響してくることになろう。

5-3　5 のまとめ

オブジェクト指向プログラミングによるプログラムにおいても、表現・アイデア二分論、創作性の両方をリンクさせて考える必要がある。結果として、著作権としての保護範囲は狭いものと思われるが、少なくとも、機能的著作物だからといって必然的に表現の幅が狭くなるということはなく、実装における創作性の発現の余地は大いにありうる。そのときの表現の幅は、部品の選択・配列の表現の幅に現れる場合もあれば、どのような部品を使ったかによる部品の中のソースコードを含めた全体のソースコードの表現と

しても現れる。

ただし、著作者が一体何を作ったのかということに立ち返って考えると、やはり前者の捉え方がより重要になってくるだろう。

また、上記のような選択肢の中から、潜在的であれ、選択をしていることには著作者の人格の流出としての個性の発揮という側面があるだろう[56]。この点については 7 で触れようと思う。

[56] 前掲注 31・中山（創作性）p4 はこの個性といわゆる伝統的著作物における人格の現れとしての個性は「同床異夢」と指摘する。ただし、伝統的著作物においてもプログラムにおいても表現の時においてその著作者が潜在的に選択や配列をして表現していることには変わりなく、その部分には最低限の人格の流出があると見ても良いのではないか。その流出の度合いがプログラムにおいては少ないというだけのことである。

6　特許法の保護における問題

プログラムは審査基準の改訂や平成 14 年の特許法改正を経て、物の発明として保護される。

本稿においては、明らかにされたプログラムの特徴が特許法の保護にあうものかどうか、どのような問題点が浮かび上がるかに集中して論じたいと思う。

以下のようなことを論じることになろう。①保護が求められる部分と保護対象の関係②他の物の発明との関係性③ネットワーク化による保護対象設定上の問題と順次検討していきたい。

6-1　保護が求められる部分と保護対象の関係

著作権法におけるプログラム保護においては、著作権法がプログラムのコード、すなわち表面的な表現に着目した保護が前提となっていたため、いかにして機能の部分にまで保護を拡張し、実効性を高めようかというスタンスで議論が行われていた。

特許法においては定義規定からも分かるとおり、保護すべきプログラムは「指令の組み合わせ」であり、機能そのものが保護対象になりうる。

プログラムにおいて最も保護の要請が強いのはその機能である。

ソフトウエアの開発においては顧客のニーズを分析し、それに応じたソフトウエアを設計し、製品化する。顧客ニーズの分析は、そのソフトウエアの機能となって現れるのだが、この分析作業には相当のコストと時間がかかる。いま、Ａ社のあるソフトウエアが売れたとしよう。競合他社は当然類似の機能を持つ商品を発売してくるであろう。なぜなら、その機能が顧客のニーズに最もあっているからだ。Ａ社のソフトウエアの機能を見れば、それと同じ機能を持つソフトウエアを作ることはたやすいことであるようだ[57]。つまり、競合他社はプログラムのコードにアクセスしなくても同様の機能をもつプログラムを制作することが可能であるということだ。競合他社が大手になればなるほど、そのような再生の容易性は高まる。すると、Ａ社にとっては二つの意味で痛手である。まず、競合製品が出てきてしまい売上に影響が出ること。もうひとつは、競合他社は顧客ニーズの分析やひいては機能を果たすための設計等がＡ社の製品をみて分かるため、そのコストが削減できてしまうということである。

この損害に対しては、保護の要請が高いことは容易に想像できるが、著作権法はなん

[57] サイボウズ株式会社山本氏へのヒアリングにおいても、GUI を見れば、独自のコードで同じ機能を果たす GUI を容易に作成することができるという話を聞いている。

の保護もしない。なぜなら後者のような部分を参考にしても、これはアイデアに過ぎず、著作権法の関知するところではないからである。

この点にこそ特許法の出る幕がある。A社は機能、及びそれを果たすための設計を盗まれた。この機能こそ特許法の保護範囲である。

6-2　他の「物の発明」との関係

プログラムは有体物としての形態を持たないという点で、他の「物の発明」とは大きく異なる。また、これまでの手続き型プログラミングにおいては、何かの機能を果たすための手順がコードに書かれており、機械というよりも、機械の構造を説明する文章と捉える方がしっくりしていたように思われる。プログラムが審査基準で保護対象と認められるに至った過程を見ても、「自然法則利用性」との関係上、ハードウエア資源の利用、記録媒体に収めたプログラムなど、有体物と関連させた形で特許を認めていた経緯からも、プログラムはコンピュータに対する命令の流れを示した単なる文章として捉えられていたように思われる[58]。それが最近解釈によって機械と同様の物として捉えるようになってきた。

オブジェクト指向プログラミングにおいては、システムを「モノ」を単位に捉え、ひとつのソフトウエアをあたかも部品の組み合わせのように捉えることができる。これはまさに機械が部品の組み合わせでできているのと同じような印象で捉えられる。有体、無体という大きな違いはあるものの、構造そのものに観念上の違いはなくなっているのである。

よって今までより、より「物の発明」として観念しやすくなっていると言える。このイメージの違いはプログラム特許の利用可能性に心理的に大きなプラス要因になるように思われる。

6-3　ネットワーク化による保護対象設定上の問題

オブジェクト指向プログラミングにより、システムを「モノ」単位で捉えられるようになり、プログラムは部品の組み合わせとして実現するようになった。それぞれの部品

---

[58] 例えば、河野登夫「特許法によるソフトウエアの保護」パテント Vol.37 No.3p89（1984 年）で、「特許法が『ソフトウエア』の保護を歓迎しない理由」として「審査の困難性」を挙げ、外国においてプログラムそれ自体を保護しない特許法が存在することについて、その理由を「読解が困難」（傍点は筆者挿入）であることに求めている点からも分かるだろう。

は、その機能のみが明らかになっており、ソフトウエアを作る人は必ずしも、その中身の手続きがどのようになっているかを知らなくてもよい。よって、目的に合致する機能を果たすものであって、要求したときに動作してくれれば、どこにどのような形で存在していてもかまわないのである。通常の機械とは異なり、部品はひとつの場所に物理的に組み合わされている必要がないのである。したがって、ネットワーク化が飛躍的に進歩した現在、ソフトウエアのプログラムはひとまとまりにならず、メインのプログラムはＡのコンピュータにあるが、そのプログラムが呼び出す部品はＢのコンピュータにあるということも大いに生じうる。もちろん部品もＡのコンピュータにある場合もあるが、これらのバリエーションはハードウエアの利用の仕方が異なる。クレームを書く際に両方をカバーしようとすると、ハードウエアの利用のあり方について抽象的に書かざるをえず、自然法則利用性の要件にかかり、拒絶理由にかかる可能性があり、他方、ハードウエアの利用形態ごとに請求項を並べると、審査請求費用が高くつく上に、バリエーションの見落としが生じる可能性もある。発明の最も大事なところとは言えないハードウエアの利用形態が発明の保護範囲に影響を与えてしまうという問題が生じているのである。

機械のようにひとつの所に部品が集まってある機能を果たすという有体物ならば当然の前提が、同じ「物の発明」たるプログラムには通用しないのである。プログラムの一体性の問題とも言えよう[59]。

[59] このプログラムとしての一体性の問題は特許発明としての一体性の問題とともに、著作物としての一体性の問題としても存在しうる。

7　著作権法、特許法の保護の併存－著作権法の拡大？

以上のように、著作権法、特許法ともに問題点を抱えながらも、保護の根拠とすることは可能であるように思われる。

そして両者の保護は併存することができる[60]。著作権法はあくまで表現に着目した保護を行うし、特許法はアイデアに着目した保護をしているからどちらに特化するという必要もない[61]。

プログラムにおいて、実際に保護の要請がどこにあるのかということに注目してみよう。保護の要請があるのは「海賊盤と優れた技術思想である[62]。」海賊盤すなわち、デッドコピーは表現に着目していると言いやすく、技術思想とはプログラムが要求された機能をどのように果たすかというアイデアそのものである。単純に見れば前者を著作権法、後者を特許法で保護するという結論になろう。

ところが、優れた技術思想が盗まれた場合、それは異なるコードを持ったプログラムとして現れた段階で侵害に気づくことが多いため、また享受できる機能が同じであることから、こちらも何とかプログラムの表現保護としての著作権法に頼ろうという考え方に陥りやすい。そしてデッドコピー、技術思想、両者ともに著作権法での保護を何とかできないかということになってしまう。

昨今の著作権法の議論において、機能的著作物の効果的な保護を企図して、従来の考え方ではアイデアとされていた部分においても規範的に「表現」と捉えて、その部分に踏み込んで保護していこうという考え方が現れている[63]。

これらの意見は、論者によって保護対象や創作性の判断の仕方が異なるが、いずれも、表現保護法としての限界線があることを自覚している。ところが、「表現」自体が規範的

---

60 例えば、小笠原昭夫「コンピュータ・プログラムの法的保護－著作権法による保護と特許法による保護」判例タイムズ No.1097p53(2002 年)、板倉集一「コンピュータ・プログラムの法的保護－特許法と著作権法の役割－」日本工業所有権法学会年報 23 号（1999）p15（2000 年)、豊田正雄「ソフトウエア特許とプログラム著作権」パテント Vol.45 No.7p25（1992 年）

61 前掲注 60・板倉 p19

62 サイボウズ株式会社・中根弓佳氏

63 例えばオートくん事件（大阪地判平成 14 年 7 月 25 日最高裁ホームページ）において、プログラム著作物の複製について「被告ソフトウエアは、本件ソフトウエアとは構造が著しく異なり、本件ソフトウエアに設けられている機能の多くを有しておらず、プログラムの具体的な表現と言えるコードにも類似する部分がないから、構造、機能、表現のいずれについてもプログラムとしての同一性があるとは認められない。したがって、被告ソフトウエアは、本件ソフトウエアを複製又は翻案したものとはいえない。」と判示している。この事案は結局侵害としなかったが、表現のみならず、機能の同一性について表現と同等に判断しているところが気になる。

に判断されるということがその限界線を不安定にしてしまう。

これを今回のプログラムの保護の要請と照らし合わせてみると、デッドコピーは表現に着目していると言いやすく、技術思想とはプログラムが要求された機能をどのように果たすかというアイデアそのものであるから、これらの両方を著作権法で保護しようというのは無理であるように思われる。ところが、「表現」を規範的に捉えることで効果的な保護を企図すると結論は変わってくるかもしれない。表現保護の限界線も規範的に変えられるものだからだ。表現・アイデア二分論の危うさをここに認めることができるのである。その危うさの中で著作権法の体系を保持しようとすると、その最後の砦はもはや「表現」という言葉そのものなのかもしれない。

また、創作性の議論においては、創作性の要件から、人格の流出としての個性という要素[64]を排し、表現の幅のみで判断しようとする説が出てきている[65]。プログラム等の機能的著作物が人格の流出物であるとは言えないと考えられるということを根拠に、人格の流出としての個性を軸として創作性を判断できるものとできないものが著作権法の保護対象に入ってきてしまったため、その両者の保護要件を統一的に把握するための便法として出てきた説である。

表現の幅のみを考えるという創作性の判断基準をすべての著作物に持ち込むことは、個性があるゆえに他人に模倣させてはならない著作物を決める基準として創作性があったということから、個性の有無は関係なく、ただ模倣させたほうがいい（模倣せざるをえない）著作物を保護対象からはずす基準として創作性を見るという方向にシフトしたとも考えられる。

模倣させないとしていたのは、情報の豊富化のための手段だったと考えると、プログラムの混入により、情報の豊富化に対する手段を 180 度転換したようにも思われる。

この説は著作権法による保護対象をこれまで以上に拡大しようというような意図はない。著作権法の目的たる情報の豊富化を実現する方法を変えただけである。

しかし、その結果として、著作者人格権や著作権法とはそもそも何かという問題が生じてくることが指摘されている[66][67]。

---

[64] 前掲注 39・斉藤 p71「創作は独自の精神作業である。その成果物が著作物である。したがって、著作物の要件としての創作性は極めて単純に著作者の個性、独自性と解してよい。」
前掲注 52・金井 p17 では創作性を「独創性」と「創造性」にわけ、それぞれ著作者の独自創作、著作者の「創造的個性」が現れていることを要求する。

[65] 前掲注 31・中山（創作性）p6

[66] 前掲注 31・中山（創作性）p11

[67] これまで人格の流出としての個性を創作性の要件としていたことが、著作者人格権の強い著作権法の体系と適合していたが、この説によるとその体系を根底か

表現の幅があるなかで、潜在的であれ、何らかの選択・配列をし、プログラムとして表現したのであれば、そのコードに関してはその度合いは低いものの人格の流出としての個性が現れているといってもかまわないと思われる[68]。少なくともデッドコピーまでの範囲であれば個性発現の結果としての著作物として捉えうるであろう。創作性の考え方をドラスティックに変えることなくユーザーニーズは満たせるのではなかろうか。

以上のように、プログラムにおいて求められる保護は海賊盤防止と優れた技術思想であるとすれば、デッドコピーということについて著作権法が担当すればよいのである。そのためには従来の著作権法の体系を維持しつつぎりぎりの解釈論で保護が可能である。これ以上に極端にアイデアに踏み込んだ保護を企図してみたり創作性の概念から個性的要素を排除したり、著作者人格権を見直したりする必要はないように思われる。この解釈論を超えるような保護がプログラムに必要であれば、もはやそれは著作権法ではどうしようもできないものなのである。

ただ、技術思想の保護を特許法のみに限ることは、進歩性の判断が厳しいため十分な保護が期待できないということが一般的に言われている。しかし、進歩性の判断こそ、技術動向に応じて容易想到性も変化しうることから、大いに政策的な判断が介することになる。進歩性の判断が厳しいというのは、それだけ技術思想のパブリックドメイン化の利益を大きく見ているという結果なのである。よってプログラムの保護を特別に弱くしている意図はない。さらにそれを補完するように著作権法の保護を技術思想にまである程度拡大して実効性のある保護を目指すというのは、この政策判断と矛盾する[69]

オブジェクト指向の特徴が、表現の幅があることと、プログラムの機械的な要素をより色濃くしたことにあるとすれば、その表現の部分の保護と、機械的な部分の保護の住み分けがより直感的にできるようになるのではないか。

ら覆すことになるということである。しかし、創作性における人格の流出としての個性があることが、著作者人格権の前提となると言えるのか疑問の残るところもあり、今後の議論を待ちたい。

[68] 前掲注 56 参照

[69] かつて議論されたプログラム権法は、プログラム開発の経済的利益の保護に重点をおいてプログラム開発を促進するという目的で作られ、特許法との補完関係を目指していた。ここでは「製品としてのプログラムの流通及び利用の促進」という観点が強く現れる。そして「プログラムそれ自体に限って保護客体とする姿勢を貫くべきである」（牛久健司・森田俊雄「特許法とプログラム権法（仮称）」パテント Vol.37No.3p84(1984 年)）と言われていた。プログラム権法が今の著作権法が目指すべき方向性を示しているように見えてならない。

[70] 例えば、小笠原昭夫「コンピュータ・プログラムの法的保護－著作権法による保護と特許法による保護」判例タイムズ No.1097p53(2002 年)、板倉集一「コンピュータ・プログラムの法的保護－特許法と著作権法の役割－」日本工業所有権法学会年報 23 号（1999）p15（2000 年)、豊田正雄「ソフトウエア特許とプログラム著作権」パテント Vol.45 No.7p25（1992 年）等

8　おわりに

ソフトウエア開発の世界はまだまだ発展途上の段階にはあるものの、確実に進歩し、プログラミングパラダイムやソフトウエア開発技術の転換が起きている。

その技術転換はプログラムが機械に近い存在であることをより浮き彫りにした。そして、プログラミングという作業の性質の変化は、ソフトウエア開発における表現的要素をこれまで以上に希薄にした。また、他方でコーディングの際の表現の幅が広く存在することを認識させた。

新たな特徴認識のもと、プログラムを改めて捉えなおすと、著作物（編集著作物も含む）とも機械とも捉えることができることがわかった。

昭和 60 年著作権法改正以降、技術的著作物の適正な保護を何とか図るために様々な努力が行われてきた。ところが、プログラムの技術の方が、著作権法から離れていっている現実がある。コードという表面的な表現とそれが機械的な要素を持つことがはっきり分離して把握できるようになった。その技術変化に気づかぬまま、ただただ著作権における保護の実効性を高めようとするのは危険である。

技術とユーザーニーズをしっかりと把握し、著作権法、特許法、それぞれのできることとできないことをしっかりと見直すことでプログラムの実効的な保護は可能であるように思われる。

技術を知ることと、法律の体系を知ることは車の両輪のように密接に関わっている。本稿において、技術を知ることで、知的財産保護の対象がより明確になり、法律の体系の検討をすることができたということもこの一例と言えよう。

プログラムに限らず技術の進歩はめまぐるしい。法律はその進歩に追いつけないまで

---

71 前掲注・板倉 p19

72 *横山先生はこちらの点に特に注目して規範的に表現とアイデアを分岐し、創作性については、表現の幅ではなく、「他者が同一の結果に到達する蓋然性」が低ければ創作性を認めやすくなるという基準で判断している。ここでは著作者が独自に創作したことも判断の一要素として考慮されるため、完全に個性的要素を排除しているわけではないようである。*

73 前掲注・中山シンポ p11

74 サイボウズ株式会社・中根弓佳氏

75 かつて議論されたプログラム権法は、プログラム開発の経済的利益の保護に重点をおいてプログラム開発を促進するという目的で作られ、特許法との補完関係を目指していた。ここでは「製品としてのプログラムの流通及び利用の促進」という観点が強く現れる。そして「プログラムそれ自体に限って保護客体とする姿勢を貫くべきである」（牛久健司・森田俊雄「特許法とプログラム権法（仮称）」パテント Vol.37No.3p84(1984 年)）といわれていた。プログラム権法が今の著作権法が目指すべき~~最低限の~~方向性を示しているように見えてならない。

76 前掲注 48 参照

も、追随する努力が必要である。法律を論ずるにはその社会を正しく認識しなければならない。技術の法律を論ずるには技術を正しく認識する必要がある。そのためには法律の枠を超えた学際的な視点、そして専門分野の枠を超えた協力が必要になる。本稿が微力ながらこの視点の重要性を認識させるきっかけになれば幸いである。

以上

---

[i] ~~*横山『著作権法における創作性概念の再構成－個性的世界から可能的世界へ－』（未刊行）P114*~~

[ii] サイボウズ株式会社においては、実装、試験、評価のプロセスについて反復開発モデルを導入している。

[iii] 横山

[iv] *前掲注・横山論 p114*

[v] 後者の捉え方は部品のコードの違いに着目している点において、部品の選択と言う要素を多分に含む。後者のようにそこに強く注目した捉え方をした場合、そのプログラムを編集著作物として捉える考え方に近くなっていくように思われる。

[vi] だから、プログラム自体は「技術的思想」と捉えられていなかったといえる。