技術探索

Linux —コミュニティ型開発モデルの進展と法制 I
—オープンソースソフトウェアの原点となる権利と責任の概念—

井田　昌之
青山学院大学

オープンソース，フリーソフトウェア，著作権，GPL

1. ソフトウェア開発者から見た根幹となる課題：既存ソフトウェアを利用したソフトウェア開発

オープンソースソフトウェアの開発に関して，2回にわたり，その変遷について取りあげる。単一の組織が閉鎖した環境ですべてを開発するいわゆるクリーンルーム開発ではなく，それとは全く逆の理念に基づいて開発する形態をとる場合に，どのような形で，その権利や責任を分担していくのかが2回の共通したテーマである。

簡単に言えば，「すべてを自分の手で開発するのではない」というべき状況がソフトウェア開発には付随しており，これをどう考えるかということでもある。積極的にあるいは肯定的にとらえる考え方を，ここでは大別して，コミュニティ型開発と称することとする。このような形式は，今日では広く一般的に見られるようになった。しかし，そこでの課題は整理して論じられてきたとは言い難かった。2回に分けて，歴史的な背景に触れながら論じる。

第1回では，ソフトウェア開発者の視点からオープンソースソフトウェア（OSS）の基本的な枠組みについて解説する。ソフトウェアの品質保証という観点において，「開発したプログラムのすべてに責任を持つ。自分が作成した部分でなくとも」という論理がその根底にあったことを明らかにする。フリーソフトウェアの概念の説明が中心となる。第2回では，Linuxを中心として，複雑化した今日のソフトウェア開発環境の中での開発管理の視点から，著作権を中心とする知的所有権にかかわる諸課題について扱う。「多数の人が開発するソフトウェアでの責任分担と相互依存についての対応」ということがその根底にある。

一環した課題であるコミュニティ型開発モデル，その根幹にある課題は何か？　そこから出発する必要がある。

それは，「既存ソフトウェアを利用したソフトウェア開発，これを複数の人間が共同で行う。どうやって，他人の開発した部分について，権利と責任という観点で，誰が責任を負うか？」ということである。基本的にはソフトウェアの品質維持・向上と開発効率の課題であるが，それと同時に，権利関係の課題でもある。

2. 発端：ライブラリ利用のソフトウェア開発とオープンソース

1960年代にはじまる大型計算機のTSSシステムは，今日のソフトウェア開発のありようの原型をもたらした。コンピュータプログラムは，図1にあるように，メーカなどの他者が開発した実行時ライブラリを利用して開発すると理解するのが基盤であった。

具体的な仕組みはシステムによって違いがある。ライブラリは一般性のあるソフトウェア機能モジュールの一形式でもあり，その呼び出しも，静的なソースコードレベルでの結合から，実行時の結合，更に動的な結合，ネットワークを介した通信機構を介した呼び出しへと発展し，いろいろな方式が用いられるようになった。フリーソフトウェアそしてオープンソースの課題はこれらの時代の変化に密接に関連して議論が推移する。

具体的な開発のステップの中でこれを見てみよう。図2のようになる。用意されているライブラリプログラムの中で必要な機能を呼び出す記述を加えながら，場合によっては他開発のソースを織り込みながら，開発対象であるソースコードを作っていく。それをコンパイルし，機械可読形式のプログラムを生成する。用意されている実行時ライブラリを静的にあるいは動的に結合し，アプリケーションプログラム中での呼び出し参照を解決し，全体として実行可能なソフトウェアとするのである。

当初，それらのライブラリ機能は，コンピュータメーカがハードウェアに付随して供給したものがほとんどなので，ソースコードは利用者に公開されず，実行形式のみでしか使用が許可されなかったが，それに異論をはさむケースはほとんどなかった。その動作の責任はメーカのハード

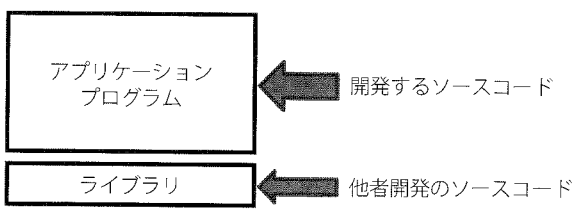

図1　ライブラリを利用したアプリケーションプログラムの開発

© 2012 The Institute of Electrical Engineers of Japan.

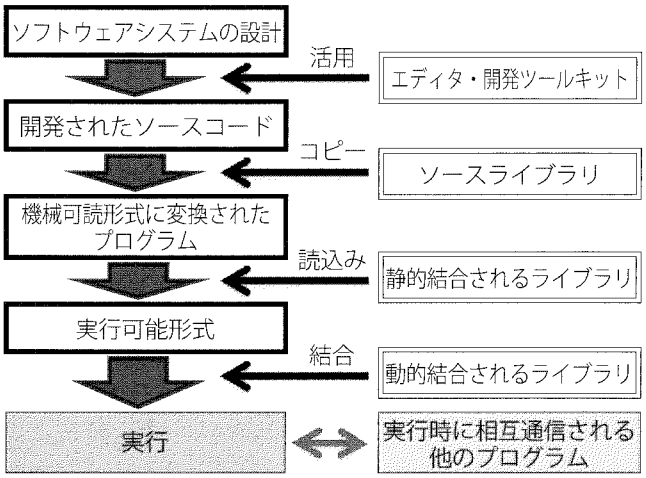

図2　他者開発のソフトウェアを利用したソフトウェア開発

ウェアの責任と一体であり，著作者の作品という理解はほとんどなかった。しかし，著作権でいう二次的著作，あるいは共同著作などとの区別に関する課題は，本来，最初からあったのである。コンピュータ利用の当初から知的所有権の問題ははじまっていたのである。この問題は，基本的には権利を保護してタッチさせないという構図だと理解されがちだが，逆に，著作権は主張するが，広く使ってほしい，という共同への志向が加わってわかりにくくなった。オープンソースソフトウェアはこの中に位置する。また，Linuxが中心的な役割を果たしているが，UNIXそしてLinuxの技術的な特徴，変遷については文献（1）に記し，Linuxでのさらなる課題についての説明は第2回に述べる。

3.「著作権」という「技術」を利用した権利保護：無方式主義の中での方式主義

ソフトウェアは，その開発の実体としてソースコードがある。したがって，著作を論じる際にはソースコードに関する課題が議論の対象となる。オープンソースソフトウェアについても同じである。

ソースコードに関する権利の保護を何によって行うかの議論があり，著作権を中核とすることとなった。著作権の保護は，1886年に成立し，その後，改訂を経ているベルヌ条約が基本となる。ベルヌ条約の文案は文豪ビクトルユーゴの発案で作成されたという。客観的な基準で判定されるべきである経済的な利害関係ということと，作品としての著作者の意志ということの両者を等しく尊重することが著作権の大枠の枠組みである。WTO加盟のベースとしてベルヌ条約の受け入れが求められ，WTO加盟を希望する途上国が，その過程としてベルヌ条約に加盟したのは，そう古い時代の話ではない。例えば，ベトナムがベルヌ条約に156番目の加盟国として加わったのは，2004年10月26日のことである。著作権の趣旨に沿った文化的な国際参加と経済的な国際参加の同期の実現といえる。

表1　ライセンス条項を含む著作権表示

1. Copyright© にはじまる著作権者の明示
2. 利用許諾条件（ライセンス条項）の記述

ベルヌ条約では，表示・登録をしなくても，著作権は著作物の創作時に発生する無方式主義と呼ばれている。このベルヌ条約をほとんどの国は認めているので，これが著作権に関する国際的な共通理解といってよい。

一方で，著作権は一定の手続きに則った処理をすることで権利として認められるという考え方がある。方式主義と呼ばれる。この考え方に沿っている条約として，万国著作権条約がある。万国著作権条約は，著作権の主張をCopyright© にはじまる記載を付することで著作権が発生する枠組みである。この方式主義に基づく考え方を記述することも著作の一部であるから，無方式主義の中にあっても，一定の方式にしたがってライセンス条項を持つ著作権を著作者の権利として守る仕組みを作れる。フリーソフトウェアあるいはオープンソースソフトウェアは，これにならって，「ソースプログラム開発者は，定められた枠組みを使って著作権の存在を宣言し，そこにライセンス条項を付記することで，その著作権者としての権利を履行する」という枠組み（**表1**）を利用して権利保護をする出発点とした。

4. GNU General Public License（GNU GPL）

1980年代にRichard Stallman氏を中心に起こったフリーソフトウェア運動はその中にあった。彼が率いるフリーソフトウェア財団（FSF）を著作権者とする表示形式が広く広められ，用いられた。その中核となる著作権表示形式とそこでの権利主張の仕組みがFSFの中核プロジェクトであるGNUでのGNU GPL（General Public License：GNU）である。あるいは単にGPLと呼ばれる。GPLは，1989年にその初版が発表された。最新のバージョンはGPL v3である。なお，GPLの本文は，必ず，原文を見てほしい。各言語訳にはどうしても訳者の受け取り方という側面とその言語の文化的な側面が入るのは否めない。

GPLは，GNUプロジェクトの成果であるGCC（GNU Compiler Collection）をはじめとしたさまざまなソフトウェアや，Linux Kernelで採用され，今日も広く用いられている。FSFは，それ以外の組織などが著作権者になって，類似した表示をすることを勧めたので，90年代後半から次第にさまざまな団体を著作権者とする表示形式が現

れてきた。また，ライセンス条項もさまざまに派生していった。

GPLから派生したライセンスは主に，1991年にまとめられたGPL v2からの派生である。GPL v3は2005年ころに文案の作成に着手され，2007年にできた。また，主にライブラリを想定して，自身はGPLと等しい概念に基づくフリーソフトウェアだがそれを利用するアプリケーションプログラムへの著作権の非連続を認める条項を含むライセンスが，LGPL（Lesser GPL）の名称で1999年にその初版が作られた。このGPLとLGPLがのちに大きな違いを形づくることになる。第2回の主要なテーマとして扱う。

1990年代から出現した多くのオープンソースライセンスについて，http://www.gnu.org/licenses/license-list.htmlにFSFからの解説がある。それらのライセンスの違いを細かく指摘するのは本稿の目的ではない。むしろ，それらに至る流れと変遷の根底にあるものを紹介することで，開発者の視点では何を問題にしようとしたかを紹介する。結果であるライセンスの字句を理解しやすくすると思うからである。

5. GPLが本来うたってきたもの：開発者の責任と権利

GPL発表の当初，FSFでは，Emacsと呼ばれるテキストエディタの利用が盛んであった。GNU Emacsである。筆者はそのいきさつと内容をまとめて「Emacs解剖学」というシリーズ記事を今はもうなくなったbit誌（共立出版）に連載した。Emacsでの開発イメージはGPLに大きく影響した。Emacsはインタプリタシステムである。この構造では，利用者が使いやすくするために行ったコード追加を他の人が共有する・利用するという文化がきわめて自然に存在していた。今日では，ウェブで多用されるスクリプト言語によるプログラム記述などにもつながるソフトウェアの考え方である。GNU Emacsは，図3に示す方式をとっている。Emacs Lispによるプログラム記述で（ライブラリ）機能を追加したり，変更したりできるようにされた。Emacsは単にエディタとしてだけでなく，メールの読み書きからはじまって，表計算や科学計算のためのパッケージまで誰かによって作られ，共有され，機能拡張されていった。また，Emacsインタプリタ自身のソースコードも公開されて，多数の人がバグの指摘や改良に協力した。

GNU Emacsは，すべてRichard Stallman氏が直接そのソースコードを管理した。すべての改良・機能拡張・バグの修正は，送られてきたソースコードを彼が十分に理解し，更に，彼の書き方となじむまで吟味されたのちでなければ組み入れられなかった。これによって彼はGNU Emacsというインタプリタの全体の品質に責任を持った。すべてを背負うという覚悟のもとに動いていた。大変な手間と責任を一人で背負った。利用者から送られたバグフィックスコードは全体としては必ずしも最適・適正ではない。それに手を入れていいかといちいち確認する必要があった。それをスムーズにするために，しかし，他人の著作に勝手に手を入れないようにするために，まとまった機能単位に関して著作権の仕組みの利用を思いついた。これがGPLのアイデアの発端である。

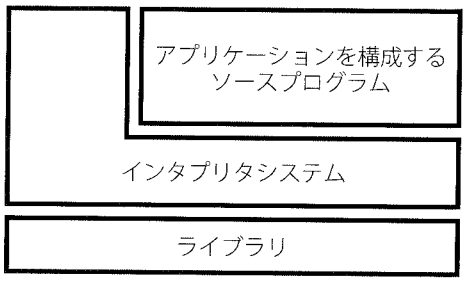

図3　Emacsインタプリタシステムの構造

表2　四つの自由

1. いかなる目的に対しても実行する自由
2. 改変・改造の自由
3. 複製の再配布の自由
4. 改良版の再配布の自由

このころ，80年代の後半，Cコンパイラの隆盛をみることとなった。GNUでもCコンパイラの開発をすることになった。さまざまな状況に対応できる，すなわち，プラットフォーム独立になるように，コンパイラはアセンブラへの入力を出力し，それを対象となる機械の機械語へ変換するという二段階方式がとられた。他の言語にも対応ができるし，さまざまなコンピュータへの移植がしやすくなる。独立したコンパイラを別々に作るより便利になるはずだ。GCCの誕生である。GCCで利用するライブラリは，また別に開発・維持することで更に開発効率を上げようとした。Glibcの誕生である。

多数の人がこれらにかかわることで，ソフトウェアの開発環境を共通化し，相互に助け合えるようにした。ソフトウェアにはバグがつきものだ，それをみんなで拾って直して行こう。それにはソースコードをアクセスできないと困る。

『原著作者の権利は尊重するが，利用者（である別の開発者）が自分で直せる，改良できるものは，それを自分でやる。それは皆の利益になるはずだ。それを自分のソースコードでも宣言しよう。』

これにつきる。Richard Stallman氏がフリーソフトウェア運動を進めた動機は，これがさまたげられたことに対するある種の怒りからである。実行時に利用する（メーカ供給の）ライブラリの中味についても私が直せるなら直したい。ソースコードを見せてほしい，という主張である。

Richard Stallman氏は，ソフトウェアに関する四つの自由の概念（**表2**）をまとめた。1985年にGNU Manifesto宣言を出し，www.gnu.orgを利用してその概念を展開した。開発者である著作権者がその成果であるソースコードに対してとなえる理念である点から見てほしい。どのような目的でも使えるように，直してもいい，コピーして配ってもいい，改良ができたらそれを再配布してもいい，ということである。1番目の点は，たとえ軍用であったとしても利用してもいいよ，何に使うかまでは著作権者は責任は持たない，といったニュアンスまで含んでいた。

四つの自由に基づき，そのソースコードを自由利用してよいとした。それは本節で述べてきたような文脈からであった。皮肉なもので，一度文章になり，多くの人が知ることになると，それが独り歩きする。文の解釈とその適用にさまざまな人の思惑と主張がからんでくるようになる。開発者の視点ではなく，ソフトウェアの利用者の視点で，使えるものは自由に利用しよう，あるいは，平等にすべて公開しよう，あるいは法原理的な視点で，公開されないのはよくない，といった概念の人達が入ってきて，状況は複雑になった。FSFも，切り離す部分は切り離すが，できるだけいろいろなものもその中に含められるようにしようとする。大きな議論を呼んだのは，4番目の改良版の再配布の自由についてである。その改良版を作った人・組織の著作権主張はどのようになったらいいのか？　この点が今後の枝分かれの原点となる。

さまざまな個別の課題への相談が外部からFSFに持ち込まれた。自分の権利は守りたいが広く使ってもほしい，ということから，他者が開発したソフトウェアを利用したいが利用しても権利関係は大丈夫か，といったことまでいろいろであった。こうした実務上の法的な相談に対して対応する部門も1990年代に入ってFSFの中に作られた。いずれもFSFでの視点は，ソフトウェアソースコード開発者がどのように他者の著作権を尊重しながら，自分の開発をするかというものだった。FSFの運営にかかわる費用はそうした相談の費用からも出すということが模索された。GPLライセンスにすることにメリットを感じるハードウェアメーカなどが資金あるいは人材を出して，GPLに基づくGNUソフトウェアの開発が進められるという側面もあった。GPL理念の普及活動にRichard Stallman氏自身は次第に専念することとなった。

FSFでのフリーソフトウェアの根本の思想は，経済的な価値観についての課題ではなく，開発者の立場にたって，自分以外の人の作品である部分もどうやってチェックできるかという課題である。これがそのまま工業産品であるソフトウェアについて導入されたので，一世を風靡（ふうび）する概念となったと同時に，商業的なソフトウェアに対する挑戦のように受け取られたことは不幸な時代のめぐりあわせだったと個人的には思っている。インターネットの波がそれにも関係した。インターネットを使えば，誰でも情報の発信者になれるし，地球上にあるさまざまな情報にアクセスできる。多くの場合，無料で，ということが前提だった。

フリーソフトウェア財団を中核とするフリーソフトウェア活動はさまざまな枝分かれを生じた。多くは，使われたソフトウェアの再配布をめぐって，また，特化したハードウェアインタフェースにからんだソフトウェアの公開に関してである。

フリーソフトウェア財団では，Unixではなく，独自のOSカーネルを持とうとしてGNU Hurdプロジェクトの立ち上げを試みた。一方，オランダのタネンバウム教授の著書（文献（2））中で公開された教育用のminixにルートを持つLinuxが90年代になり開発されるようになった。Linux Kernelの開発者であるライナストーバルズ氏は多数の人による改良が全体の利益になるように，フリーソフトウェア化を志し，Linux KernelはGPLになった。一方，Linux kernelはひとつの著作というより，さまざまな用途に利用できる汎用のオペレーティングシステムとして使われることを目指していたので，自身の著作権を守ることにはゆるぎはないが，その上で実行されるアプリケーションソフトウェアについてはさまざまな権利形態を持つソフトウェアが実行できるように意識されていった。GPL非互換のアプリケーション実行や機能付加に対応できる仕組みが工夫された。

文　献

（1）井田昌之：「UNIX OSの技術変遷」，電学誌，Vol.125, No.2, pp98-101（2005）
（2）A. S. Tanenbaum : Operating Systems, Design and Implementation, Prentice Hall（1987）

井田　昌之

いだ・まさゆき

1951年生。青山学院大学大学院国際マネジメント研究科教授。1980年代末からしばらくFSFのVice Presidentを務める。Common LispおよびJavaの言語仕様に関する研究等を通して，FSF，MIT人工知能研究所と長年の交流があった。インターネット上のグローバルな情報システムについて研究。工学博士。

技術探索

Linux —コミュニティ型開発モデルの進展と法制 II

—オープンソースの利用に伴う著作権課題の技術変遷：Linux が歩んだ道—

井田　昌之
青山学院大学

オープンソース，Linux，著作権，コミュニティ型開発

1．Linux でのライセンスの混在：Linux kernel を中心とする各種ソフトウェアのパッケージ

さて，先月号の第 1 回（文献(1)）では，GNU GPL を中心とするフリーソフトウェアそしてオープンソースソフトウェアでの著作権による権利主張と保護，そして開発したソースプログラムに対する実行時の責任に対する，ソースコード開発者の意識に言及した。企業活動からすれば，そこで生産するソースコードは知的財産そのものであり，GPL のソフトウェアを利用すると，自社の開発ソフトウェアも公開しなければならない。ということが強調され，本来的に複数の異なる開発者が作ったプログラムの，実行時の責任の分担をどうするかという視点はあまり注視されなかった観がある。ソフトウェアにはバグ（設計・制作の誤り・不十分からくる誤った動作を引き起こすプログラム部分）の存在が宿命であり，ソフトウェアの品質の管理は開発管理の中でも大きなウェイトを占める。同時に，自社開発の部分は自社で責任を持つ，そうでない部分はなんらかの約束によって他者が責任を持つ。そうやって，協業は成立するという考え方も他方のごく自然な論理である。

前回（文献(1)）での文脈を理解してもらえれば，こうした点から，「四つの自由」に付随して，GPL を利用したソフトウェアは GPL でなければならないという FSF の主張が含まれることの意味を読者には理解してもらえたと思う。GPL に従うソースプログラムを改良した場合，そのソースプログラムは GPL でなければ全体の理念が整合しない，という共通認識を FSF は求めた。一方，自分のところだけ自分で責任を持つという立場では，GPL に従うソフトウェアの実行形式を利用した場合にどう扱うか，ということがこの後の議論の中心となる。

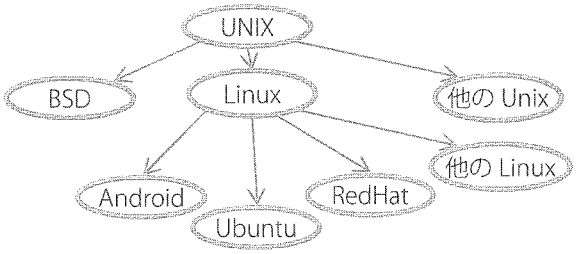

図 1　UNIX から Linux へ

Linux と単に称される場合，Linux Kernel を基盤として，それに，いろいろなパッケージを組み合わせて構成された配布パッケージ（ディストリビューション）を指す。GNU GPL に従うものだけをまとめた Ubuntu Linux，商用を意図し，場合によっては GPL ライセンスに基づかないものもそのパッケージ中に含めることを想定した RedHat Linux などが代表的なディストリビューションの例である。スマートフォンの OS として広く用いられている Android も Linux kernel を基盤にしている。図 1 に示す。なお，Apple 社の Mac OS X は，BSD Unix をベースとした自社 OS である。iPhone などのスマートフォン OS である iOS も同様に成り立っている。

Linux Kernel は，GPL に基づくことを選択した。前回の GNU Emacs に関して Richard Stallman 氏が直面した課題と同じ意識があったのだと筆者は理解している。すなわち，どうやって，利用者からの修正コードを取り込むかという基本的な問題意識である。自分の開発でない部分に問題が出ても，全体だけを利用している利用者からは主開発者にだけ文句がくる。権利関係がはっきりしないので直せない，いじれない，そういうことにはなりたくないだろう。かといって閉じた世界を作って，その中ですべてを扱うのはそもそも開発目的にあわない。

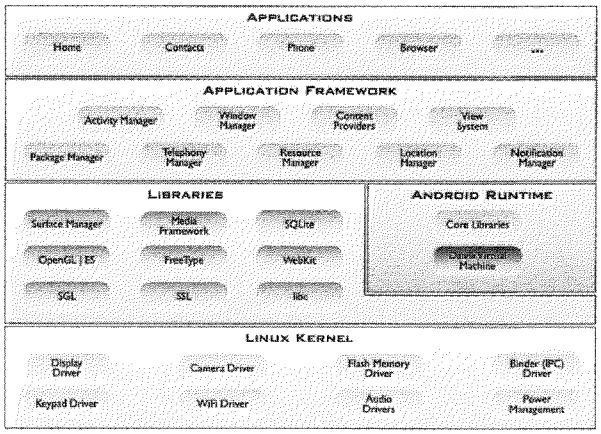

図 2　Android のシステムアーキテクチャ
(http://developer.android.com/guide/basics/what-is-android.html よりそのまま)

© 2012 The Institute of Electrical Engineers of Japan.

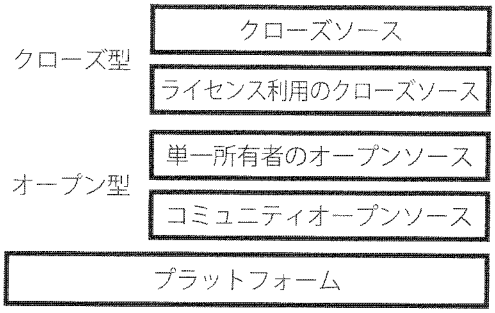

図3　ライセンスの混在したソフトウェア構成
（文献(2)図1をそのまま引用）

Androidは図2に示すアーキテクチャを持っている。Androidを構成する各機能は，それぞれに維持開発母体を伴って分担されている。その意図と管理形態に応じて，ライセンス条項もそれぞれ異なるものがとられている。

これらの結果，オープンソースソフトウェアを含むソフトウェアシステムの構造は図3のように一般化できる。そのソフトウェア製品のソースコードを公開していないクローズ型，すべてをオープンソースとするオープン型の部分とに分けられよう。それらのソフトウェア機能モジュールがプラットフォーム（Linuxであれば，Linux Kernel）上に乗っている。更に，コミュニティオープンソースとして，コミュニティメンバが構成する機関が全体の著作権者となる形態が多く見られるようになった。図3中のオープン型には，著作権者が本来複数であるか，単一組織であるかという観点が発生した。

2. 排他的私権である著作権とソフトウェアの開発との整合の模索

さて，混在したライセンスを持つソフトウェアが一体として利用される状況について論じるために，「既存ソフトウェアを利用した新規開発の著作権」というテーマに話を戻してみる。

単体で利用が完結するケース，例えば組み込み機器での制御ロジックをつかさどるソフトウェアと，OSのように，基盤ソフトウェアとして多くのソフトウェアがその上で実行されてほしいと願うケースでは著作権者の意志は異なってくる。Linuxは基本的に後者の代表例である。また，今日では，独立したソフトウェアモジュールが独立したままで，利用者から見れば一つの統合された機能に見えるようなサービスとして機能供給されるようになった。

いずれにしても，多くの開発者・企業の立場は，既存のシステムの上に「別のソフトウェア（サービス）」を作る立場である。そうした企業は，重要度が増してきた知的所有権戦略の中で，オープンソースにどう関係するかを決める必要が出てくる。ビジネスモデルの選択の問題である。また，商標権や特許権などとの組み合わせでの権利主張と自社の利益の確保ということがソフトウェアビジネスの中核の話題としてクローズアップされるようになった。オープン型は，付随するサービス，知的財産活用方法をビジネス対象とするといってよいだろう。一部をオープンソースとして公開している併用型も存在する。「フリー」というキーワードを持つ戦術を内在させるビジネス手法も広く論じられるようになった。

オープンソースの利用を伴いながら，自社のビジネスゴールを達成する方法論については，文献(2)あるいは文献(3)などに論文例を見ることができる。特に文献(2)はこの構成をもつソフトウェアに関して，関連する機関・企業のビジネスゴールとして次の3類型を提示し，その上で，ブランド戦略の重要性を意識し，論じている。

（1）開発コストの低減。ライセンス供与クローズソースおよびオープン型構成要素を採用し，クローズソース開発部分を最少化することで開発コストを下げようとする戦略。

（2）顧客露出の増加。単一所有のオープンソースビジネス機関・企業の戦略，その部分が利用されれば，迅速に低コストでソフトウェア製品が出荷できるという戦略ゴールを持ち，積極的に利用を促進することで自身のビジネス目標に役立てる戦略。

（3）競争の最少化。オープン型構成要素の採用により，不要な競争優位性への注力をさけることができる。また，コミュニティオープンソースの採用は，安定した地盤を形成できると考える戦略。

一方，ソフトウェア開発者側も，オープンあるいはフリーを自己の戦略として利用する人たちも表れてきた。例えば，無料で使えるオープンソースソフトウェアを公開することで，自分の実力を示し，それによって仕事の受注能力の証明にする考え方などである。現実的な戦略がそれぞれとられて，オープンソースは定着していっている。

3. LGPLが許諾するもの：権利の非伝搬性

GPLによるソフトウェアとそうでないソフトウェアの間をどう橋渡しするかという問題が出てくる。Lesser General Public License（LGPL）はこれへのFSFの回答である。GPL以後のさまざまな状況への妥協といってもよい。GPL v2に対応したLGPL v2，そしてGPL v3に対応したLGPL v3の発表によって多くの課題はFSFとしてはひとまずの区切りをつけた。2007年くらいのことである。図4を見てほしい。

http://www.gnu.org/licenses/lgpl-java.html に掲載され

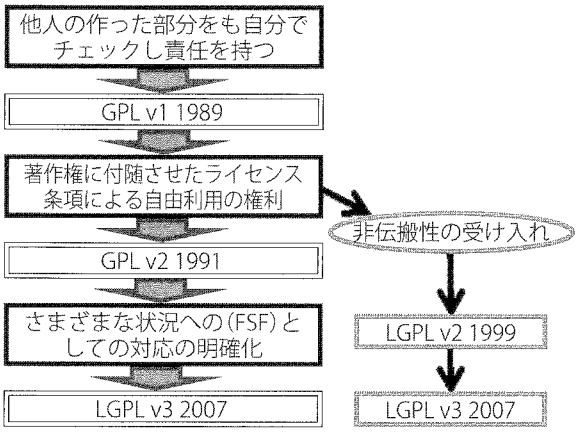

図4　GPL と LGPL

ている David Turner による文章が，FSF としての LGPL の意味を明確にする見解である。（なお，この中でも work という言葉が「生産物」といった物理的な存在を規定する言葉で訳されていることが多いが，ここでは「働き」とする。翻訳の問題もある。ニュアンスの取り方もある。）

この見解には，基本的な論点が表れている。それらは重要なので枠をつけて記す。以下の訳ならびに箇条書きによる要約は本稿のために筆者が行ったものである。

1. 一体性：動的リンクによってアプリケーションソフトウェアをライブラリにリンクすることは，両者が一体となって，一つの働き（work）を形成している，とみなす。
2. 伝搬性：GPL は，すべての派生する働き（derivative works）は，全体としても GPL でライセンスされることを要求している。これを hereditary.（伝搬性あるいは遺伝性）と呼ぶ。したがって，GPL によるライブラリをアプリケーションがリンクするのであれば，そのアプリケーションも GPL でライセンスされなければならない。
3. LGPL の非伝搬性：一方，LGPL でライセンスされているライブラリは，クローズ型のアプリケーションでリンクしてよい。

その上で，この LGPL の非伝搬性について，Java を例にして確認している。次にまとめた記述がその中にある。『Java アプリケーションは import 機能によって，ライブラリを参照する。アプリケーションがコンパイルされると，リンクが作られ，この結果，一つの働きを派生する。したがって，ライブラリの著作権者は，この派生された働きの配布をなんらかの形で authorize しなければならないことになるとも考えられる。LGPL は，この配布を許可する。LGPL に従うライブラリを import する Java アプリケーションは，その利用者が，ライブラリを修正したり，その修正をデバッグするために，アプリケーションプログラムの部分をリバースエンジニアリングすることを許可するライセンスを持っていなければならない。このことは，アプリケーションのソースコードやアプリケーション内部の詳細を供給する必要があると言っているのではない。利用者がライブラリを改造すると，ライブラリとアプリケーションのインタフェースが壊れることもありうる。その場合，アプリケーションとライブラリが働かなくなる。しかし，それは心配無用だ。そのことをした人の責任である。』

4. ライブラリを GPL によるアプリケーションと一緒に配布するなら，ライブラリのソースコードも配布しなければならない。しかし，利用者がそのライブラリを自分で入手することを要求するのであれば，ライブラリのソースコードをアプリケーションが供給する必要はない。
5. LGPL の観点で，Java と C の違いは，Java は継承を供給するオブジェクト指向言語であることだ。LGPL は継承に関して何の規定ももたない。というのは，何の規定も不要だからだ。継承は，伝統的なリンクによるものと全く同一の形で派生する働き（derivative works）を生み出すからだ。そして，LGPL は，通常の関数呼び出しを許可しているのと同様に，このような派生する働きを許可している。

すなわち，FSF としては，動的なリンクであっても呼び出していることは，一体的な働きをなしているから，ちゃんとその意図をはっきりさせないといけないと言っている。GPL なのか LGPL なのか，クローズされているソフトウェアなのかということである。そして，GPL に基づくソフトウェアを実行時に呼び出している場合，それは一体的な働きをしているのだから，呼び出しているソフトウェアもソース公開をするべきだとした。GPL がソースの公開を要求していて，かつその公開要求は伝搬するからである。しかし，伝統的なリンクによるものや継承によるものについて，呼び出しによって著作権問題を生じさせないことをはっきりしたい場合は LGPL でそれら被利用のソフトウェアのライセンスを規定すれば，その被利用ソフトウェアは基本的なフリーソフトウェアとしての要件を満たしているが，同時に，フリーソフトウェアでないソフトウェアからも呼び出しができることがはっきりするとした。例えば，glibc（GNU C ライブラリ）は LGPL であるから，それを使って基本機能を利用する場合，それらの基本機能はどんな種類のソフトウェアからであっても使えるとした。

まとめると，オープンソースライセンスには，伝搬性のあるものと非伝搬のものとがあると理解するべきである。前者の代表はGNU GPLである。後者の代表例はLGPLである。Apacheライセンスは後者に入り，Androidのライセンスはこの系統になる。

4. 今日のソリューションとコミュニティ型の開発

かつてフリーソフトウェアのビジネスが成立するかどうかが大きな議論となった。ソースコードの販売はGPLでも許されているが，無料で入手できるものをただ単に有料で入手する人はいないから，必然的にも付加価値をつけることでビジネスにすることが挑戦された。企業情報システムに対するサービス機能が中心である。利用方法に関するコンサルテーション，あるいはその企業向けに特化した機能の追加などであろう。差別化をはかろうという考えも出てくる。これはビジネスとしては自然な方向であるが，開発者の意図していたGPLの基本精神からは異なるものとなっていった。

また，産業用の応用を主に視野に入れて，Open Source Initiative（OSI，www.opensource.org）をはじめ，フリーソフトウェア財団とは一定の距離を置いて活動するグループが複数できるようになった。オープンソースソフトウェアの多くは，LinuxあるいはUNIXと関連したものが多かったので，一般的な印象では，OSS = Linuxと認識されるケースが増えた。推進団体は，特定のソフトウェアを中核に置くものや，フリーソフトウェアを使ったビジネスの円滑な推進のための団体などが残っていった。

個別の企業でも，コンプライアンスあるいは企業戦略の観点からも，一定のホームページで必要部分のソース公開をする例（文献（4））もある。また，さまざまなブログや個人サイトなどで，各自の視点からオープンソースライセンスとの付き合い方を述べているWebページも増えた。Googleなどで検索してみるとよい。ソース公開のタイミングもコミュニティ外からは見えにくい場合もでてきている。たとえば，Androidは，3.1などでは公開されなかったようだが，4.0では再びソースが公開されている。（文献(5)）

更に，先進国によるものと考えられてきた先進的ソフトウェアの開発と地球規模での普及の中で，新興国・途上国からの発信が大きく寄与する時代になっていることに留意したい。開発コストが低いといって，先進国が開発そのものを他国あるいは途上国へアウトソースするとそれを行った国あるいは企業は空洞化し，開発の競争力を失う可能性を十分に意識する必要がある。同時に，ソース共有をベースにしたライセンスであれば，アウトソースしても，その成果を共有できる。グローバルソーシングモデルである。

ソフトウェアの開発と言うこと自身の意味付けもこの間で変化している。従前のプログラミングの時代は終わったという論調はあちこちでみることができる。新しい機能とすぐれた視点を持つ，個別の単独ソフトウェアアプリケーションを作ることは，依然として重要な働きを持っているが，同時に，既存の機能を組み合わせて，求める機能を構築すること，あるいは，分散した機能が共同することで全体として一つの機能を形成すること，つまりサービスとしての一体機能の供給に，ソフトウェア開発の主点が移っているという側面がある。この場合でも，さまざまな機器に組み込まれたソフトウェアは，その機器のハードウェアの信頼度と同等の信頼度が求められ，比較すると低頻度の利用を前提としたソフトウェアとは品質評価において異なる基準が求められている。また，ハード機器の知的所有権あるいはトレードシークレットの維持という点では，ソース公開は両刃の剣となろう。

基本となるソフトウェアの開発手法という観点では，伝統的な，ソースコードを作成し，それをコンパイルし，一つの実行可能形式のプログラムとしてまとめた上で供給するというモデルとは異なった形式の利用形態が広く見られるようになっている。特にインターネットあるいはネットワーク環境下で動作するソフトウェアは，単一のソフトウェアとして機能させる場合以上に，少なくともサーバ・クライアントモデル，あるいは更に分散モデルで構築されるようになっている。ソースプログラムの著作権をベースとした知的所有権戦略は，時代の区切りをつけつつあると考えている。

〈謝辞〉編修委員会の意向がなければ，筆者の身近な経験としてのこれらの話が文字になることはなかった。機会を与えていただいた電気学会に深謝する。

文　献

（1）井田昌之：「Linux-コミュニティ型開発モデルの進展と法制Ⅰ」，電学誌，Vol.132, No.4, pp.229-232（2012）
（2）D. Riehle：Controlling and Steering Open Source Projects, Computer, IEEE CS, pp.93-96（2011-7）
（3）駒木亮伯他：「製品開発へのオープンソース利用の実状」，情報処理，Vol.52, No.8, pp.982-989（2011）
（4）http://www.fmworld.net/product/phone/sp/android/develop/
（5）http://source.android.com/source/downloading.html

井田　昌之
いだ・まさゆき
1951年生。青山学院大学大学院国際マネジメント研究科教授。1980年代末からしばらくFSFのVice Presidentを務める。Common LispおよびJavaの言語仕様に関する研究等を通して，FSF，MIT人工知能研究所と長年の交流があった。インターネット上のグローバルな情報システムについて研究。工学博士。