# オープンソースソフトウエアと企業経営における新たな課題

水　谷　　正　大

目次

1　はじめに
2　オープンソースソフトウエア問題の射程
2.1　法とオープンソース
2.2　e–Japan 構想
2.3　国際社会の姿勢
2.4　各国の取り組み
2.5　科学研究
2.6　日本におけるオープンソースソフトウエア利用の現状
3　代表的オープンソースソフトウエア
4　オープンソースソフトウエアの勃興
4.1　オープンソース運動の急展開
4.2　オープンソースソフトウエアの土壌
4.3　変容するソフトウエア利用—インフォウエアの登場
5　オープンソースの定義とユーザの権利
5.1　ソースコードとユーザの権利
5.2　OSI のオープンソース定義
6　オープンソースのライセンス
6.1　GPL 型ライセンス
6.2　BSD 型
6.3　MPL 型
7　オープンソースとビジネス
8　オープンソースソフトウエアの開発組織
8.1　オープンソースソフトウエアの開発
8.2　開発組織
8.3　開発支援環境
8.4　組織運営と意思決定プロセス

9 企業経営の課題
9.1 組織構造
9.2 組織の動的柔軟性
10 まとめ

## 1 はじめに

オープンソースソフトウエアとは，ソースコードを公開し，しかもユーザがそれを自由に改変し，それを再配布する権利を認めたコンピュータプログラムである。いまオープンソースソフトウエアの利用が個人や企業に急速に広がっているだけでなく，ソフトウエア開発を行っている企業が自ら開発したソフトウエアをオープンソース化する動きも活発化してきている。コンピュータソフトウエアのソースコードは，ソフトウエア開発・販売を含む情報産業においてその核心である。コンピュータシステム，広くは情報システムにおいて，それを実装したソフトウエアのソースコード群には，開発に要したすべてのアイデアや実現のためのアルゴリズムなどが余すところなく記述されている。

オープンソースソフトウエアを開発しているプロジェクト組織をオープンソースコミュニティとよぶ。このコミニュティにコミットするメンバは，学生や大学の研究者であったり，多くは企業の技術者である。大抵のオープンソースコミュニティは営利組織ではなく，このコミュニティに参加してオープンソフトウエアの開発やドキュメント作成，レビューなどに関わるメンバはそこから直接的に経済的報酬を受けることはない。したがって，オープンソースコミュニティの立場からみれば，ソフトウエアの生産は贈与経済の原理に従っていると考えることができる。オープンソースソフトウエアの開発でメンバが受ける報酬はコミュニティ内の名声や信頼であり，自分の得た知見をソフトウエアに反映し，その結果を他者が再び利用してくれたり，さらに改善してくれるというメンバ間での共通認識がある[3]。オープンソースソフトウエア開発を推進する駆動力となっている贈与経済の仕組みは，科学研究における報酬のあり方に類似している。科学者が自分の研究成果を他人に認めてもらうためには，オープンな相互協力でしか達成することができず，科学者の価値は発表した論文の内容と数，そしてその引用回数で測定される。

オープンソースソフトウエアとは対極にあって，ソースコードを公開せずソフトウエア上のアイデアをバイナリコードに隠蔽することで利潤を発生させようとするソフトウエアを，‘占有’の意をこめてプロプライエタリ（proprietary）ソフトウエアとよぶ。情報の希少性を利用して利益を生み出すことができることを保証する仕組みが著作権などの知的財産権である。2節で述べるように，情報源へのアクセスを制限する知的財産権の拡大解釈は，科学研究やオープンソースコミュニティの要求と強い対立を招くことになる。

情報システムの中核であるソフトウエアのオープンソース化とは，ソフトウエアの全てのアイデアや技術を秘密にすることで市場における独占的な価値を生み出すことにより販売価値を高め

ようとする戦略とは異なり，ソフトウエアの利用価値の源泉であるはずの多様な機能実現の工夫やノウハウの詳細をつまびらかにしてしまうことである。素朴に考えると，ソフトウエア開発企業が自社ソフトウエアのオープンソース化を行うことは，市場での企業のソフトウエアの販売価値を失い，企業利潤の獲得機会を放棄しているようにみえる。

オープンソースソフトウエア開発を行うにあたり，優れたソフトウエアアーキテクチャの設計以外にもっとも重要な点は，技術ではなく開発組織をどのように考えるかにある[1]。Linux カーネル開発の成功をもたらした開発リーダ Linus Torvalds 自身が認めているように[14, 8章]，Linus が開発コミュニティをコントロールしながら見いだした Linux 開発モデルの発明こそが大きな意味をもっている。実際，Linus 自身インタビューに次のように答えて，ソースを公開するだけでは成功は保証されないとしている[4]。

> "People think just because it is open-source, the result is going to be automatically better. Not true. You have to lead it in the right directions to succeed. Open source is not the answer to world hunger,"

オープンソースは，組織のあり方に関して，どのようにして組織を管理して意思決定を行い，組織目標を達成するかという組織経営の問題に対する 1 つのアプローチを与えている。

本論文では，オープンソースソフトウエアの今日の活況とその意義を考察し，オープンソフトウエア概念を詳しく説明し，オープンソースビジネスを考える上での出発点となるオープンソースソフトウエアのライセンス問題を概括する。そして，オープンソースビジネスの現在を紹介した上で，多くの企業がオープンソースソースビジネスにコミットし始めてきた背景を考察する。また，オープンソースの核心はその組織形態と運営法にあるとする立場から，成功したオープンソフトウエアの開発方法を紹介した後に，オープンソースの問題を経営組織の問題としてとらえ，企業経営それ自身の課題であることを論じる。

## 2 オープンソースソフトウエア問題の射程

オープンソース世界の基底には，最新のソフトウエア技術だけでなく，ネットワーク利用に結びついたソフトウエア開発のあり方，品質保証の方法，配布とその利用形態，企業ビジネスの問題など，ソフトウエアおよび情報システムに関するあらゆる姿と課題がある。本論文で明らかになることであるが，オープンソースという問題は先駆的な人間が，そして我々の社会が見いだした問題であり，しかも情報の世紀とよばれている現代から未来への社会の在りように密接につながっている。オープンソースソフトウエアが提起する課題は，以下に述べるように，ソフトウエア産業のあり方に関わる経済・社会的問題の枠には収まらない。そのざわめきは，情報技術の周辺に聞かれるだけでなく，基幹産業の変容の中に，人間や文化表現の変質の中に，そして政策遂行の戦略の中に聞こえてくる。

以下にオープンソースソフトウエアを取り巻く代表的状況を紹介し，論考を進めるに当たって

の諸問題の位置測定を可能にしておこう。

### 2.1 法とオープンソース

オープンソースソフトウエアを著作物とみた場合，6節で取り上げるライセンス問題とは別に，そのデジタル著作物という特性が従来法の規範では統治できない諸問題として立ち上がってくる[18]。権利の形に限定した場合，ソフトウエアという著作物を財産的権利としてとらえるか，あるいは人格的権利としてとらえるかという2つの立場によって大きく引き裂かれている場がオープンソースソフトウエアである。財産権と人格権は著作物の流通において互いに逆に作用するからである。著作物の財産的価値を主張するあまりに，著作権の保護期間を延長させようとする近年の圧力は巨大メディアを中心に強まるばかりである。本来は人類の創造性や文化を守るための著作権が，逆に創造性や文化をコントロールし，我々の革新を抑制する方向に向かっているという危機感が広まっている。スタンフォード大学のLawrence Lessig[9][10][11]は，著作物を社会資産とみなす眼差しをもって自分の著作物については自分でその利用方法を決めようとするCreative Commons[34]のプロジェクトを始めている。

### 2.2 e–Japan 構想

情報通信技術（IT）の活用により，世界的規模で生じている急激かつ大幅な社会経済構造の変化に適確に対応することの緊要性を背景に，日本政府は高度情報通信ネットワーク社会の形成に関する施策を迅速かつ重点的に推進するため，平成13年（2001年）1月に内閣に「高度情報通信ネットワーク社会推進戦略本部（IT戦略本部）」を設置し，世界最先端のIT国家に向けてのe–Japan重点計画を発表している[27]。

e–Japan構想では，基本政策として「縦」の重点項目を掲げ，同時に横断的な課題として「横」をつなぐように年次計画を立てている。現在は2003年7月に策定された「e–Japan戦略II」の段階にきている。その横断的課題も1つとして研究開発の推進をかかげ，「安全，安心，便利，感動」社会の実現の上で重要性の高まる技術の開発において，オープンソースソフトウエアの開発等の促進を謳っている[25]。オープンソースソフトウエアを，ソフトウエア開発産業において日本が高い国際競争力を維持していくために採用すべき方策の1つとして位置づけ，日本のソフトウエア技術者が世界水準の技術開発力を保持するためにも，オープンソースソフトウエアの利用技術および開発技術を浸透させ，オープンソースソフトウエア技術分野の人材育成が急務であるとしている[28]。

### 2.3 国際社会の姿勢

国連貿易開発会議（UNCTAD）の「電子商取引と開発レポート」（第4章，2003年）[29]では，特に強調して，開発途上国にとって，自由及びオープンソースソフトウエア（FOSS）は，現在および明日の専門家と情報技術のリーダーがスキルを獲得しその知識を急速に前進させることを

可能にすると報告している。そして，各国政府は，いっそう改善された貿易と発展のために，デジタルデバイドと情報通信技術とをつなぐ全ての政策との関わりにおいて，FOSS に関する政治的立場を熟考すべきであると述べている。実際，オープンソースによるプロセスは，プロプライエタリなソフトウエアと同等または優れた品質に到達できるとし，その理由として次の４つを挙げている[29, Chap4(p.102)]：

1. FOSS ではより多くの開発者が問題を見つけ，修正することができる。
2. FOSS では収益を最大化させるマーケッティングやビジネスにじゃまされることなく，開発者は修正や改良を加えてより頻繁にリリースできる。
3. プロプライエタリなソフトウエアは，法的責任を軽減するために，その利用上の責任を回避する利用許諾ライセンスによって品質を保証していない。
4. ソースコードが入手できること自体が重要なプロダクトの品質であり，ユーザは自由にソフトウエアを修理・点検することができる。

2003年12月に第１フェーズとしてジュネーブで開催された情報社会世界サミット WSIS (World Summit on the Information Society) [30]では，全ての人類に対する情報社会の基本原則の１つとして「情報と知識へのアクセス」についての諸原則を宣言した。第27項では，オープンソースソフトウエアを次のようにプロプライエタリなソフトウエアと同列に位置づけ，情報や知識へのアクセスの保証を提案している。「情報と知識へのアクセスは，プロプライエタリやオープンソースおよび自由ソフトウエアを含む異なるソフトウエアモデルが提供するすべての可能性を認識することによりいっそう促進できる。その結果，競争を加速し，ユーザによるアクセスや選択の多様性を増加させ，すべてのユーザが自分のニーズでおこる最適な解決を自身で開発できるようにすることができる。ソフトウエアへ無理なくアクセスできることは，真の情報社会の重要なコンポーネントである。」

### 2.4 各国の取り組み

いま，日本をはじめとする世界各国政府ではオープンソースソフトウエアに大きな関心を示し，その普及と導入に積極的に動き始めている[31]。政府機関で利用する情報システムを特定企業のハードウエアやソフトウエアだけで構築することに，特にプロプライエタリソフトウエアの雄である Windows を擁する世界最大のソフトウエアメーカー Microsoft をはじめ巨大コンピュータ企業を有するアメリカ製品によって，自国の情報システムが構築されてしまうことへの潜在的な警戒心をヨーロッパ諸政府は隠そうとはしていない。

一方，自国の産業の育成を課題としている途上国を多く擁しているアジア，中南米，中東，そしてアフリカにあっては，情報基盤が急速に整いつつある。こうした地域では，将来的にプロプライエタリソフトウエアの大きな市場として期待されながらも，情報インフラストラクチャの整備に際して，導入コストが安く，ソフトウエア技術さえあれば自国の要求に合った情報システムを比較的容易に構築可能なオープンソースソフトウエアの利用が広まっている。

国民の税によって運営される行政機関を含む公共サービスにおいては，その論理的な延長線上に，その全資産の国民への無料還元を含まねばならないと考えることができる。実際，この方向に沿ってイギリス放送協会（BBC）は，BBCの全放送データアーカイブをオンライン化し，営利を目的としない限り，用途や利用するデバイスを問わず，自由に資料をダウンロードできるサービスを2004年秋から始めると発表している[33]。アーカイブの配布は，パブリックアクセス権の画期的アプローチとしてL. Lessig[9][10][11]らが提唱しているCreative Commonsライセンス[34]に基づいて行われる。

### 2.5 科学研究

現代の科学研究では，研究成果を広く公開し，次の研究の発展に繋げていくという研究スタイルの原則が研究者間で合意されている。また，研究費が国家による補助を受けている場合，元来，その研究成果は税金の使途に対する説明責任として国民につまびらかにすべきものでもある。研究の現場では，以前から，ソフトウエアの導入が安価で，ソースコードを入手して目的に応じて自由に改変できるオープンソースソフトウエアが広く使われてきた。失敗の許されないミッションクリティカルな巨大研究にオープンソースを大幅に活用した最近の例として，NASAの火星探査計画における自走式探査機ローバーの制御システムがある[35]。

### 2.6 日本におけるオープンソースソフトウエア利用の現状

Linuxの成功や世界のソフトウエア企業のオープンソースソフトウエアへの投資が示しているように，ソフトウエアに関する最新技術の成果や優れた技術者集団を集積するために，オープンソースソフトウエアは有効な手段として働くという認識が世界に広まっている。企業によるオープンソースソフトウエアへの貢献や政府機関の関心が国際的に高まる中，日本企業によるオープンソースソフトウエアへの取り組みは，日本がソフトウエア開発産業において国際競争力を維持していくための大きな課題になっている。

実際，日本発のオープンソースへの取り組みは，日本語・国際化対応ソフトウエア開発が代表するような利用者＝生産者という図式が多く，オブジェクト指向スクリプト言語Ruby[44]やKAME[45]（BSDでの実装）及びUsagi[46]（Linuxでの実装）で知られるIPv 6の実装のように世界で広く利用される日本発のオープンソースソフトウエアの発信数は先進諸国としては非常に少ない[12][36]。日本発のオープンソースソフトウエアの生産が著しく少ない要因として，[12]では，オープンソース開発を担う人材自体が少ないこと，世界のオープンソースコミュニティの言語が英語であること，プロプライエタリソフトウエアが海外依存型であるように従来からの海外依存構造がオープンソースにも反映していることを挙げている。同時に，エンドユーザとしての利用や個人としてのオープンソースソフトウエアの利用や貢献とは別に，企業体によるオープンソースソフトウエアの組織的活用やオープンソースソフトウエア開発への参加を阻む障壁として，オープンソースソフトウエアの導入や維持・改良のための企業側のリテラシー水準の問題，

オープンソースソフトウエアの導入や企業化による貢献自体が本当にビジネスになるのかという疑問，そしてオープンソースソフトウエアのライセンス問題などの不安が残されているためであると指摘している。

こうした日本における現状は，2.2節で述べたe–Japan構想を実現していくためにクリアすべき課題とも重なっている。オープンソースソフトウエアのライセンスについては6節で紹介し，オープンソースソフトウエアのビジネスにおける活用モデルやその背景については7節で論ずることにする。

## 3 代表的オープンソースソフトウエア

オープンソースソフトウエアの開発プロジェクトがどの程度あるかの目安として，オープンソースソフトウエアのコミュニティを支援するOSDN (Open Source Development Network)[23]によって運営されているオープンソースソフトウエア開発のホスティングサイトSourceForge[24]のプロジェクト数がある。2004年10月で，8万9千を超える開発プロジェクトと93万人の登録ユーザがいる。日本語サイトではプロジェクト数が1200以上，登録ユーザは1万人である。これ以外にもオープンソース開発プロジェクトが多数あり，オープンソースソフトウエアの開発は世界中に広がっていることがわかる。8節で述べるように，オープンソースソフトウエアの開発は従来のソフトウエア開発の手法を突破した方法によって行われている。全てのオープンソースプロジェクトが必ずしも成功しているわけではなく，プロジェクト自体が途中で消滅してしまう場合も多数ある。そうであっても，オープンソースソフトウエアに参加する開発者は後を絶たず，この金銭的利潤の追求だけでないオープンソースソフトウエア開発のインセンティブについて考察が行われている[1][2]。

世界中で現在広く利用されているオープンソースソフトウエアの一部を表1に示した。Linuxカーネルは，1991年から開発を始めたLinus Torvaldsが率いている非常に大規模な開発チームに成長したプロジェクトによって短期間で高い完成度に達し，オープンソースソフトウエアの今日の趨勢を決定づけたUNIX互換OSである。今日，これらのオープンソースソフトウエアは，それなしではソフトウエア開発自体も成立しないほどに普及しており，特にインターネットの基幹ソフトウエアには多くの優れたオープンソースソフトウエアがある。エンドユーザにとっても，インターネットのさまざまなサービスを受けるためにオープンソースソフトウエアは不可欠になっている。実際，Paul Vixieが設計したDNSの実装であるBINDはDNSサーバとして世界のほとんど全てで利用されており，Brian Behlendorfらが設立したApache Software Foundation[53]が提供しているApacheは世界のWebサーバの7割近くを占めるに至っている[56]。また，Eric Allmanによって開発されてきたSendmailは電子メールサーバとして主要な位置を保ち続けている。

| 分 類 | 名 称 | ソースコードの入手先 |
|---|---|---|
| OS | Linux カーネル | http://www.kernel.org/ |
| | FreeBSD | http://www.freebsd.org/ |
| | Darwin | http://developer.apple.com/darwin/ |
| コンパイラ | GCC | http://gcc.gnu.org/ |
| | Jikes | http://www.ibm.com/developerworks/oss/jikes/ |
| 言語処理系 | Perl | http://www.perl.com/ |
| | Ruby | http://www.ruby-lang.org/ja/ |
| | Python | http://www.python.org |
| | PHP | http://www.php.net/ |
| エディタ | Emacs | http://www.gnu.org/software/emacs/emacs.html |
| 開発支援 | CVS | https://www.cvshome.org/ |
| | Eclipse | http://www.eclipse.org/ |
| ファイルサーバ | Samba | http://www.samba.org/ |
| インターネット | Apache | http://www.apache.org/ |
| | BIND | http://www.isc.org/ |
| | Sendmail | http://www.sendmail.org/ |
| データベース MS | MySQL | http://www.mysql.com/ |
| | PostgreSQL | http://www.postgresql.org/ |
| GUI | GNOME | http://www.gnome.org/ |
| | KDE | http://www.kde.org/ |
| | X Window System | http://www.x.org/ |
| Web ブラウザ | Mozilla | http://www.mozilla.org/ |
| | Amaya | http://www.w3c.org/Amaya/ |
| | Lynx | http://lynx.isc.org/ |
| オフィス | OpenOffice | http://www.openoffice.org/ |
| | GIMP | http://www.gimp.org/ |

**表1 代表的なオープンソースソフトウエア**

Netscape は，1998年1月に自社の有料プロプライエタリソフトウエアであった Web ブラウザ Netscape Communicator を無料配布し，今後そのソースコード Mozilla をオープンソースにするという発表を行い業界に衝撃を与えた。当時，大手ソフトウエア企業がその中心的ソフトウエアのソースコードを公開するという前代未聞の行為[14, 第14章]は，ネットスケープ社自身のビジネス上の思惑（自社には大きな利益を生み出さなかったといわれている）を超えて，Open Source Initiative[40]の設立によって自由ソフトウエアを包含する‘オープンソース’という言葉

を生み出す契機となり，さらには今日のオープンソースビジネスのあり方を方向付けたという大きな歴史的転回点として記憶されることになった。

最近のオープンソースソフトウエアの開発では，エンドユーザが使うアプリケーションソフトウエアにまで及んでおり，Microsoft Office 互換のオフィスツール OpenOffice や画像処理ソフトウエアの GIMP，Mozilla プロジェクトの最新の成果である Web ブラウザ FireFox とメールソフトウエア Thunderbird[62]は高度なセキュリティ機能をもち，世界中で広く利用されている。

オープンソースソフトウエアを利用する際の原則は自己責任であり，問題が発生した場合には自分で対応するかコミュニティなどから情報を得る必要がある。このオープンソースソフトウエアの‘無責任性’は，一方でソフトウエアの先進性とアイデアをテストすることができるメリットをもたらしている。

エンドユーザがどんな経験をしているかという情報収集においては，オープンソースソフトウエアの右に出るものはない。プロプライエタリーソフトウエアである Microsoft の Windows のセキュリティーホールは Microsoft だけが修正できる。ソースコードが非公開のために，ユーザー側で抜本的対策をとることができず，ユーザは自分で脆弱性をさらさないように工夫しながら Microsoft の修正パッチのリリースを待つしかない。Linux・オープンソース現象に脅威を感じたマイクロソフト社の内部戦略メモが，ハロウィーン文書として暴露された[43]。ハロウィーン文書の重要性は，オープンソース運動の発展によって一番失うものが大きい企業がオープンソース方式での開発の優位性を明確に認識していることを認めている点にある。

## 4 オープンソースソフトウエアの勃興

オープンソースソフトウエアは，企業をはじめとする多くの人にソフトウエア市場参入への道を広げている。オープンソースソフトウエアは，4.1節で述べるように急速にクローズアップされた結果，巨大な資金が流れ込んで大きなビジネス潮流の一つとなって，ビジネスとしてのさまざまな可能性が検討されるに至っている。しかし，まず検討すべきことは，何故今になってそのような動きが起こり始めたのかという点である[14, 1章]。6節でも述べるように，オープンソースソフトウエアという言葉が使われるずっと以前の1980年代から，R. Stallman による明確な意図を持った自由ソフトウエア運動が始まり大きな成果をあげていたにもかかわらず，オープンソースソフトウエア運動の隆盛にどうしてこのように時間がかかってしまい，しかも急激な展開を見せるようになったのかという問題である。

### 4.1 オープンソース運動の急展開

オープンソースとしての GNU 自由ソフトウエア[47]は1980年代中頃から開発が始まり，着実な成果を積み上げ技術者の間に広く普及していた。しかし，オープンソースソフトウエア運動の

最大の飛躍となったのは，短期間での完成度の高いLinuxの開発とその周辺ビジネスの成功である。1998年には，急激な事態の進展が立て続けて起こった。NetscapeがWebブラウザのソースコード公開を発表（1月），オープンソースという言葉を生み出しそれを普及する非営利団体Open Source Initiative (OSI)[40]が発足（2月），データベース分野の大手OracleとInfomixがLinux対応の発表（7月），これにCybaseとIBMのデータベースシステムも続き，1998年度中にLinuxが企業の基幹システムのプラットホームの1つとして認知されるに至った。また,1998年には，Red HatがintelやベンチャーキャピタルからIT出資を受け，さらに1999年3月には，Compaq，IBM，Oracleからも出資を受けるに至り，Linuxのビジネスモデルの1つが確立した。

さらに1998年末からは，企業の自社製品をオープンソース化する動きも現れた。IBMは企業戦略を大きく転回させ，オープンソースへのコミットを高め，以来多大な投資を続けている。IBMは，オープンソースのApacheをベースにして自社固有の機能を盛り込んで自社ブランドのWebサーバに組み込んだ。さらに，自社開発のJavaコンパイラJikesのオープンソース化やアプリケーション・サーバ用の開発ワークベンチEclipseのオープンソースコミュニティへの寄贈，さらにApache Software FoundationへJava用XMLパーサXercesやWebサービスSOAPのApacheSOAPなど，ソフトウエア技術のオープンソース化や寄贈だけでなく，ハードウエアビジネスでも全てのサーバー機のブランドをeServerに統合して，ユーザの選ぶプラットホームOSの選択肢としてLinuxを追加した。IBMだけでなく，HP, Dell, NEC, Fujitsu, HITACHIなど大手ハードウエアメーカもLinuxのサポートを始めた。また，Apple社は1999年3月からMac OS Xの基盤OS Darwin[68]をオープンソース化し，上位のGUIなどの部分をプロプライエタリーソフトウエアとする戦略をとった。こうした動きに合わせてオープンソース開発手法やオープンソースビジネスの可能性に関する分析や調査，紹介が盛んになされてきた[1][2][3][14][15][16][17][19]。

### 4.2 オープンソースソフトウエアの土壌

ソフトウエアは自由に流通されなければならないとR. Stallmanが主張し始めた1980年代は，ソフトウエア産業が成立し，資金力の豊富な巨大企業を頂点としたソフトウエア商業市場の成立の時代でもあった。自由ソフトウエア運動はこれらプロプライエタリソフトウエアに対するアンチテーゼとして出発したのである。パーソナルコンピュータの世界では，MicrosoftのWindowsが世界標準としてOSを支配し，ソフトウエア開発はMicrosoftが独自に定めたプロトコルとWindows APIに完全に依存しなければ開発ができない状況になっていた。この結果，ソフトウエアメーカがWindowsのネイティブソフトウエアを他のOSに移植することを困難にし，Microsoftはさらに独自のインターフェースを提供し続けることによってソフトウエア支配を維持することができた。

しかし，1990年代に本格的に普及してきたインターネット技術が，‘Microsoftの支配’の及

ばないソフトウエア世界として立ち現れてきたことがソフトウエア開発に転機をもたらした。インターネット技術は，各様のハードウエア上のさまざまなOSによって稼働しているコンピュータをつなぐために，当初からオープンな標準技術の集積として発展してきた。インターネットでは，ネットワーク全体を監視・運営している会社や団体があるわけではなく，その代わりに，インターネット接続やサービスのための標準規格を制定したり，運用に関するさまざまな諸問題を調整する国際組織がある。Internet Society (ISOC)[20]は，インターネット技術の進歩や利用に関する国際的な協力や標準化を推し進める学会として1992年に創設され，ここにはインターネット全体のアーキテクチャの方向性に関する決定と援助を話し合うためのIAB (Internet Architecture Board)[21]がある。IABの下部組織であるインターネット技術の標準化を進める作業部隊であるIETF (Internet Engineering Task Force)[22]は，その標準技術の仕様を一連の技術文書Request for Comments（RFC）として公開している。

インターネットの世界では，新しい技術をオープンソースにして議論しながら標準技術として定着させて規格化し，ソフトウエア開発では，このオープンな標準規格を厳守することが求められる。インターネット通信のための全てのソフトウエアはRFCなどで公開されているプロトコルに準拠しており，このために，異なるプラットホームへの移植は比較的容易となる。インターネットの爆発的な普及はコンピュータハードウエアの発展だけでなく，オープンな標準技術の可能性を追求した結果でもある。インターネットの成功を支えている要素はオープンソース運動にも見だすことができる。オープンソース運動とインターネット技術は互いに密接に結びついており，ベンチャーキャピタルの投資や政策によって充実したインターネットのインフラストラクチャはオープンソースの活用を劇的に促進してきた。大学などでインターネット技術を学び，オープンソースソフトウエアに親しんだ学生が今後多数輩出されてくることにより，両者が結びつくソフトウエア技術は，その科学的進歩を促しいっそうの発達をとげると期待することができる。

### 4.3 変容するソフトウエア利用—インフォウエアの登場

インターネットの普及とインターネットを利用したビジネスは，今日，コンピュータ利用に重大な局面をもたらしている。すなわち，コンピュータの利用が，仕事業務や趣味や研究のためでなく，インターネットショッピングサイトを利用するためであったり，音楽のダウンロードサービスのためであるというものに移行しはじめているという事実である。キラーアプリケーションは，もはやデスクトップ用ソフトウエアでも業務アプリケーションではなく，独立したWebサイトになってきた。Webサイトは，従来のコンピュータモデルでは処理できない仕事をコンピュータで実現していると考えることができるために，Webサイトをアプリケーションと見なすことができる。このような使われ方をインフォウエアとよぶ[14, 13章]。

インフォウエアでは，情報それ自体を取得することが最終目標ではなくなり，取得された情報は従来のソフトウエアには処理できない意志決定の拠り所として，ユーザが制御するためのイン

ターフェースとして機能している。したがって，インフォウエアの利用のためには，大きなコンピュータ資源（メモリや処理速度など）を有するパーソナルコンピュータを必要とはせず，携帯電話など軽量級のコンピュータであっても十分その目的を達成することができる。人とコンピュータとの対話を可能にする能力や，コンピュータで処理されていなかった情報を処理する能力をもっているのはソフトウエアではなくインフォウエアであり，インフォウエアの登場は相対的にソフトウエア単体の価値を減少させているということができる。

オープンソース運動におけるLinuxの成功はOS市場の金銭的価値を減じるという明白な結果をもたらし，多数のオープンソースソフトウエア開発の成功とその利用の普及は，ソフトウエアそれ自身がコンピュータビジネスにおいて価値を生み出す唯一主要な源泉ではないという今後のソフトウエアビジネスの展開を考える上で意味深い洞察を我々に与えることになったのである。

## 5 オープンソースの定義とユーザの権利

### 5.1 ソースコードとユーザの権利

Open Source Initiative (OSI)[40]は，オープンソース文化を啓蒙するためにEric S. Raymond, Bruce Perensらによって1998年2月にカリフォルニアに創設された非営利組織である。OSIは，単にソースコードが公開されるだけでなく，5.2節で述べる10箇条の条件を満たすことを求めてオープンソースソフトウエアの定義文書The Open Source Definition[41]を公開し，その定義に合致するオープンソースライセンスを承認している[42]。

OSIのオープンソースの定義は，ソフトウエアユーザのための権利憲章である。オープンソース方式による開発が可能であるのは，OSIが定義するオープンソースソフトウエアが，ユーザやソフトウエア開発者に次の権利を保証しているためである[14, Bruce Perens]：

- プログラムの複製を自由に作り，それを配布する権利
- ソフトウエアのソースコードを入手する権利
- プログラムを改良する権利

これらの権利が最初に明確に謳われたのは，1984年にR. Stallmanによって創始されたGNU Projectにおいてである[47]。6.1.1節で述べるように，R. Stallmannによって発明された巧妙なライセンスであるGPL[48]によって，GNUソフトウエアであり続けることができる。このため，GNUソフトウエアはオープンソースソフトウエアの中でもっとも純化されたソフトウエア形態である。

オープンソースとして作成・改良されたソフトウエアは，プログラムを書いた者を含むオープンソースコミュニティに還元され，何らのライセンス料を支払うことなしに，新しい市場に向けた開発をも迅速に行うことを可能にする。オープンソースソフトウエアを使って人や企業がそのソフトウエアを販売することも可能である。その結果，ソフトウエアの販売価格は安くなる。したがって，オープンソースソフトウエアでは，ソフトウエアの販売価値だけを切り出した場合，

そこから利潤を生み出すという従来のビジネスモデルは成立しない。しかし，オープンソースソフトウエアの利用者を対象にサポートサービスを提供するビジネスを行うことができ，特定の顧客やニーズに合わせてソフトウエアを改良して新たな市場を開拓することができる。ここにオープンソースのビジネスチャンスがある。7節で述べるように，オープンソースソフトウエアビジネスはいまや無視できない潮流となっているのである。

### 5.2 OSIのオープンソース定義

OSIでは，次の10箇条の条項を満たすソフトウエアライセンスをオープンソースソフトウエアと定義している[41]：

1．再配布の自由（free redistribution）。ソフトウエアは，いかなる相手に対しても，有償，無償を問わずに自由に再配布可能であること。
2．ソースコードの公開。プログラムはソースコードを含んで配布されること。あるいはその複製に要するコストとして妥当な額程度の費用で入手できること。
3．派生ソフトウエア（derived work）の再配布可能。ソフトウエアの改変が可能で，しかも変更したソースコードを同じ条件で再配布可能であること。派生ソフトウエアの作成，並びに派生ソフトウエアを元のソフトウエアと同じライセンスの下で配布することを許可しなければならない。
4．原作者のソースコードの完全性。変更をパッチファイルで配布する場合に限り，ソフトウエアの一貫性を維持するために，改変されたソースコードの配布を制限できる。
5．個人やグループに対する差別の禁止。特定の個人やグループを差別してはならない。
6．使用する分野に対する差別の禁止。特定の分野でプログラムを使うことを制限してはならない。
7．ライセンスの流通。ソフトウエアが複製され再配布される場合，ライセンスはそのまま適用され，追加的ライセンスを必要としてはならない。
8．特定製品に依存したライセンスの禁止。プログラムに付与された権利は，特定のソフトウエア配布物の一部であるということに依存してはならない。プログラムをその配布物から取り出したとしても，そのプログラム自身のライセンスの範囲内で使用あるいは配布される限り，元のソフトウエア配布物において与えられていた権利と同等の権利を保証しなければならない。
9．他のソフトウエアに干渉するライセンスの禁止。ライセンスはそのソフトウエアと共に配布される他のソフトウエアに制限を設けてはならない。例えば，ライセンスは同じ媒体で配布される他のプログラムが全てオープンソースソフトウエアであることを要求してはならない。
10．テクノロジーに中立なライセンス。ライセンスの提供は，個別のテクノロジーやインターフェースのスタイルに基づいてはならない。Web以外やクリック・ラップをサポートし

ないダウンロード方法の可能性や，非GUI環境で動作する可能性を考えること。

## 6 オープンソースのライセンス

オープンソースソフトウエアはさまざまなコミュニティ組織や企業が開発しているため，そのライセンス条件は多様で，その法的妥当性は必ずしも明確になっていない[32][12]。オープンソースソフトウエアをビジネスに利用する場合，そのソフトウエアのライセンス形態には十分に注意を払う必要がある。事実，欧州連合としてオープンソースソフトウエアの共同利用を推進する欧州委員会IDAが2002年6月に提出した報告書[32]では，オープンソースソフトウエアの核心はコードでなくライセンスにあるとし，さまざまなオープンソースソフトウエアのライセンスを詳細に分析検討して，オープンソースソフトウエアを社会で共同利用する道を検討している。

オープンソースソフトウエアに基づいて行うオープンソースソフトウエア及びプロプライエタリソフトウエアの開発は，オープンにされているソースコードを参照しながら行われる。企業など組織体が，自らの利益を維持しながらオープンソースソフトウエアを導入したり，オープンソースソフトウエアへの貢献を行おうとするにあたって，オープンソースソフトウエアのライセンスに関して次の3つの障壁がある：

- オープンソースソフトウエアのライセンスが，関係諸国の著作権法および特許法などとどのような合理的関連性を維持しているか（準拠法をどこの国としているか）。
- オープンソースソフトウエア，特にGPLに従うオープンソースソフトウエアと連動して動作するプロプライエタリソフトウエアを開発した場合，どこまでがオープンソースでどこからが自分の開発したソフトウエアであるかの境界が判然とせず，GPLの規定する派生ソフトウエアの範囲が不明確である（GPLが定義する派生物は，多くの国の著作権法が規定している派生物や2次著作物よりも広い概念とされている）。
- オープンソースソフトウエアと称するソフトウエアを使ってシステム構築を行ったときに，プロプライエタリソフトウエアが混入して著作権侵害を引き起こす可能性や，企業内業務に合わせるためにオープンソースソフトウエアを改変して社内で利用した場合，改変後のソースコードの公開を求められる可能性があり得る。

オープンソースソフトウエアのライセンス形態は，以下で述べるようにGPL型，MPL型，BSD型に大別することができ，この順に自由度が増し，プロプライエタリソフトウエアへの転用が可能となっている。

### 6.1 GPL型ライセンス

GNU GPL (General Public License)[48]は，全てのユーザにGNUソフトウエアを再配布し変更する自由を与えるというGNU自由ソフトウエア宣言[47]を実施するために，R. Stallmanによって考案された巧妙なライセンス契約である。GPLでは，GPL対象ソースコードと他のソー

スコードを組み合わせて1つのソフトウエアとしたとき，そのソースコードについてもGPL条件が波及し，GPLに従わねばならない。GPLに従うソフトウエアとして，GNU C/C++コンパイラ，GNU Emacsエディタ，GNU Cライブラリなど現在のソフトウエア開発には不可欠なUNIX互換のツール群をはじめ，Linuxカーネル（LinuxカーネルとGNUツール群と組み合わせたOS一式をGNU/Linuxともいう），画像処理アプリケーションGIMP，GUI環境のGNOMEなどがある。

### 6.1.1 GPLのコピーレフト

自由ソフトウエアの普及を実現するために，R. Stallmanは著作権上の概念「コピーライト」を逆手にとってコピーレフト（Copyleft）の概念を生み出した。GPLは，そのコピーレフトの概念をライセンス契約として明文化したものに他ならない。ソフトウエアに対するユーザの権利を与えるために，仮にそのソフトウエアの作成において自動的に付与される著作権をあえて放棄してソースコードを公開した場合，そのソースコードを改変してプロプライエタリソフトウエアとしてソースコードを公開しない商用ソフトウエアに転用され，結果としてユーザのソフトウエアに対する自由な権利を奪ってしまう可能性を排除できない。そこでR. Stallmanは，ソフトウエアの著作権を放棄せずにあえて保有したうえで，その配布条件として，まずソースコードおよびそれから派生するいかなるソフトウエアに対しても使用・改変・再配布する権利を与える，そして，これを再配布する人にもこの再配布条件を変更しないことを条件に，配布される人にもそれをコピーし変更を加える自由を与えなければならないという論理を考え出し，これをコピーレフトとよんだのである。コピーレフトは，著作物の占有化を目的とする権利ではなく，それを自由にしておく権利を保護するための著作権上の仕組みである。

### 6.1.2 GPLの伝搬

ソフトウエアの原作者によってひとたびGPLが宣言されると，そのソフトウエアはどんなに改変されて配布されようともGPLであり続けねばならないというGPLの伝搬性が発生する。このGPLの伝搬性によって，自由ソフトウエアが将来的にプロプライエタリソフトウエアに‘なる’ことをあらかじめ排除することができ，全てのユーザがソフトウエアに対する自由な権利が保証されるという点が，GPLに従うソフトウエアライセンスの本質である。

ライセンス法的には，GPLの第5条にある「コピーレフト」の規定および，その派生ソフトウエア（a “work based on the Program”）の部分が重要である。GPL対象のソフトウエアと他のソフトウエアとを混合またはリンクさせて1つのソフトウエアにすると，全体がGPLの対象となる。たとえば，プロプライエタリソフトウエアを開発する企業がGPLソフトウエア周辺で開発を行う場合，GPLのいう派生物であれば，企業秘密にしたい部分を含むソフトウエア全体にGPLが適用され，ソースコードの公開が求められることになる。GPLの「伝搬性の及ぶ範囲」をどのように解釈するべきかが，企業などでGPLソフトウエアを導入する場合の障壁とな

り得る[32][12]。どこまでが派生物かという境界はGPLでは不明確であり，ライセンス上の問題が発生することになるからである。

### 6.2 BSD型

BSD型ライセンスは，BSD（Berkley Software Distribution）系UNIXで使用されていたライセンス形態で，再配布に際して，ソフトウエアの著作権表示と再配布条件の表示，および無保証・免責宣言を行うことのみを条件とする非常に制限の緩やかなオープンソースライセンスである。このため，BSD型ライセンスはソフトウエアベンダがオープンソースビジネスに参入する壁を低くしている。BSD型のライセンスには，Free BSD Copyright[52]，MIT License[51]，X 11 License[50]，Apache Software License[54]などがある。

BSD型ライセンスでは，著作権表示と再配布条件および無保証・免責宣言さえすれば，BSD型ライセンスのコードを他のソフトウエアに組み込み，さらに組み込み後のソースコード（派生ソフトウエア）を非公開にすることができる。このために，7節で述べるように，特にプロプライエタリソフトウエア開発企業にとっては，BSD型ライセンスのソースコードをベースとしてビジネスを展開することができる。GPLのように厳しい利用条件がないため，オープンソースソフトウエアの普及という観点からみれば，BSD型で開発したソフトウエアはクローズドソフトウエアとなってしまう危険性がある。

### 6.3 MPL型

NetscapeのWebブラウザのソースコード体系はMozillaとよばれている。Netscapeが1998年にWebブラウザのソースコードを公開するにあたり，慎重に検討されたライセンス方式に基づいてできあがったものがMozilla Public License（MPL）[58]である。MPLでは，準拠法，裁判所轄や特許への対応などについても明示されており，GPLにあるような法的な曖昧さは解決されている。

MPLは，初期開発者（原作者）と寄与者との間で異なる権利をもっているコピーレフト的なライセンスであるが，派生物に同じMPLライセンスを適用する義務はソースコードだけに制限されている。寄与者が修正した場合，初期開発者やそのコミュニティにソースコードを公表しなければならないが，寄与者はそのバイナリコードをプロプライエタリを含むどんな形のライセンスであっても再配布することができる。これによってエンドユーザがバイナリ版を複製／配布する権利を制限しながら，一方で，開発者（寄与者）の開発しコンパイルした新たなバージョンを配布する権利を守るというライセンスである[32, p.43]。

MPLはオープンソース開発とビジネスとの間の興味深い妥協を提供している。その結果，MPL型のライセンスは，元来プロプライエタリソフトウエアを開発してきた企業による自社ソフトウエアのオープンライセンス化に都合がよい。MPL型ライセンスには，たとえば，Mac OS Xの非GUI基本レイヤーであるDarwin[68]に適用されるオープンソースライセンスであるAp-

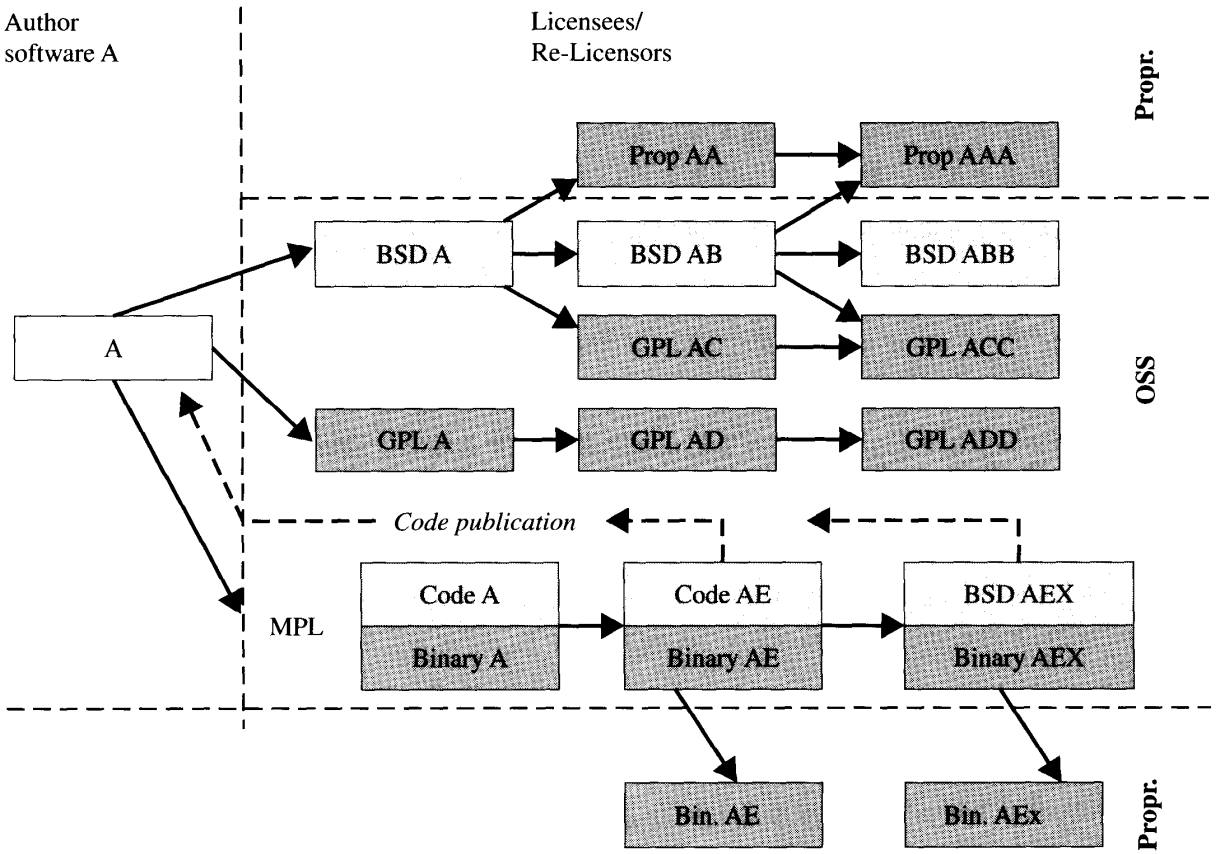


**図1 ライセンスの生成（[32, Figure 2 (p. 10)]）**

ple Public Source License[67]や，IBM の IBM Public License[69]などがある。

しかしながら，MPL ではコピーレフト効果が GPL とは異なっているため，原作者が決心しない限り MPL コードは途中の派生物から GPL にはなり得ない。さらに，MPL ライセンスでは原作者にコードのある部分を他のライセンス形態にして，1つのソフトウエアパッケージを多重のライセンス契約のもとでリリースすることを認めている。このため，MPL ライセンスは GPL とは両立しない[63, MPLの項]。

図1（[32, p. 10]から引用）は，これら3つのライセンス型で，オリジナルソースコードを改変して派生ソフトウエアが生みだされていく場合に，ライセンスがどのように変化し得るかを示している。BSD 型では，ソフトウエア A がその原作者によって BSD ライセンス化された場合，それは別のベンダによってプロプライエタリソフトウエアとしてソフトウエア AA，AAA などになったり，同時に，初期バージョン A の BSD ライセンス版が AB に改変され，同じ BSD 型ライセンスのもとで再配布されて自由に AB，ABB などの派生物となっていったり，別に GPL として派生物 AC，ACC などになっていくことができる。

GPL 型では，全てのソフトウエア A の再配布は，（同じものや，改変されたもの，ソフトウエアの一部として含まれていても）派生物 AC／ACC や AD／ADD は同じく GPL でなければならない。GPL では，どんなソフトウエアのバージョンであっても将来的にプロプライエタリソフトウエアにはるようなことはなく，公共的であることを保証する。

一方，MPL 型では，ソースコードとバイナリとは連動しない。ソースコードはオープンであり続け，その原作者には全ての改変 AE，AEX などは通知される。ところが，バイナリはプロプライエタリソフトウエアとして配布することができるが，単なるエンドユーザが複製して再配布することは制限できる。開発者は，改変したいときソースコードにアクセスし，コンパイルして，新しいオリジナルバージョンとして再配布することができる。

## 7 オープンソースとビジネス

オープンソフトウエアという仕組みをビジネスで活用する場合，現在，大別すると以下の4つの方向性がある。ビジネス戦略としてのオープンソースの活用については，ここではその詳細は述べないが，すでにいくつかの分析も進んでいる[3][14,11章][12][29]。いずれの場合も，オープンソースソフトウエアのさなざまなライセンス問題を正確に認識しておくべきことはいうまでもない。

**オープンソースソースの利用** 企業が自社の情報システムの構築や業務遂行のためにオープンソースソフトウエアを利用するという方向がある。このためには，企業自身がオープンソースソースを独力で導入し自社で維持管理できるリテラシーを備えている必要がある。

**配布とインテグレーションビジネス** Red Hat が成功したビジネスモデルで，オープンソースソフトウエアそれ自身を取り扱うディストリビューションビジネスや，さらに発展したシステムインテグレーション・ソリューションビジネスがある。ユーザにとって価値あるソフトウエアを選択し，利用しやすいようにパッケージ化して無償または有償で配布し，その品質やサポートを有償で保証するのである。また，インテグレーションやソルーションビジネスにおいては，必要に応じて非オープンソースソフトウエアや個別に開発したソフトウエア組み合わせてユーザの要求を満たす情報システムを提供し，技術支援や利用支援を行う。こうしたサポートビジネスの品質を保証するために，Red Hat の RHCE や LPI Linux[70]のような認定制度を設けて技術者の育成を支援する仕組みも広まっている。

**オープンソースソフトウエアの付加価値ビジネス** 既存のオープンソースソフトウエアをベースにして，付加価値を付けて商用ソフトウエアとして販売するアプローチがある。Apache コミュニティの特徴はベースとなるソースコードの共有にある。Apache ライセンスでは，Apache のソフトウエアを独自に拡張した新しいソフトウエアを開発した場合，それをプロプライエタリソフトウエアとすることができる。ベンダはこの方式で追加開発したモジュールをライセンス販売するのである。IBM は自社の Web サーバ IBM HTTP Server を Apache を元にして機能強化し，製品 WebSphere の一部としている。こうすることによって，Apache ユーザはそのままコンテンツを移行させることができ，付加価値を持つ差分機能の使い方だけを学べばよく全体を学び直す必要はない。

**自社ソフトウエアの寄贈とオープンソース化**　さらに踏み込んで，自社が開発してきたプロプライエタリソフトウエアをオープンソースソース化し，オープンソースソースコミュニティ支援に強くコミットする方向性がある。4.1節で述べたように，IBM は自社の統合開発環境 (IDE) Eclipse をオープンソースコミュニティに寄贈した。IDE のオープンソースソース化によって開発ツールは標準統合され，ベンダはこの共通基盤の上に品質のよいモジュールを開発して，自社のソフトウエアに組み込むことができるようになった。コミュニティは Eclipse を多数のプラットホーム OS に移植しため，共通の環境操作に慣れたエンジニアを増やすことにつながり，その結果として優れたエンジニアが育成される。結果的に，コミュニティ全体と寄贈した企業はその恩恵を被ることになるという効用の大きな循環の可能性がでてくるのである。

IBM に関していえば，自社のサーバハードウエア群すべてで Linux の稼働を保証し，全てのプラットホームで同じソリューションを提供するというミドルウエア戦略をとっている。さらに，IBM は自社の研究実験段階の先進ソフトウエアを公開する仕組み alphaWorks[71] と Developer Works[72] を持っている。必ずしも収益を生まないかもしれない開発中のソフトウエア開発をバイナリ公開やオープンソースとすることで，研究者や学生，オープンソースコミュニティとの間で生まれるフィードバックを，企業のソフトウエア開発のインセンティブとする長期的な戦略を立てている。

企業によるオープンソースコミュニティとの関係維持や積極的なコミットの背景には，Netscape が1998年に自社の Web ブラウザのソースコードを公開したときのように，オープンソースモデルを企業の生き残りのチャンスと捉えて当面の自社の利益を後回しにしてでも，オープンソース化によって得られる技術革新を最優先にすることで技術開発における優位性を確保しようとした経済・ビジネス戦略とは別に，もう２つの傾向を窺うことができる。

その１つは，企業投資や利益回収を，企業と市場との間での直接的な二者関係からではなく，ソフトウエア産業の全体を取り巻く社会をなしている異なる複数コミュニティにおける知的財産と人的資源，およびユーザとの循環プロセスから，最終的な利益を生み出そうとする利益獲得モデルの見直しを指摘することができる。もう一つの傾向として，企業利益の確保の方法が変化したため，市場だけでなく社会貢献としての企業姿勢の可能性を模索し始めたことが挙げられる。オープンソースソースビジネスの成功を通して，自社の企業価値が再利用可能な社会のインフラストラクチャであり得ることに企業自らが気づいたのである。

オープンソースソフトウエア運動の歴史はソフトウエアの社会資源化の確認の歴史でもあった。オープンソースソースコミュニティによるソフトウエア開発とそれを支える企業活動が続くことにより，オープンソースソフトウエアは社会の天然資源となっていくことになる。

# 8 オープンソースソフトウエアの開発組織

ソフトウエア開発の歴史において，成功するソフトウエアプロジェクトの要点は技術でなく組織にあるという認識はひろく拡まっており，オープンソースソフトウエアにおいてもその事情は変わらない。参加者の自由意志に基づいて成立しているオープンソースコミュニティでは，9節で論ずるように，プロジェクト組織上の課題はさらに先鋭化された形で経営学上の問題を提示している。

オープンソースコミュニティでは，非営利目的のために自由意志で参加しているメンバに対して金銭的インセンティブを与えたり組織買収などによってプロジェクト運営上の問題の解決を強制することはできず，あくまでもコミュニティメンバの賛同を得るような形でコミュニティ自身による自発的な解決が求められる。にもかかわらず，成功したオープンソースプロジェクトは，従来の開発形態よりもソフトウエア開発能力において迅速に，しかも品質の高いソフトを作ることができたのである。ここにオープンソース組織体を経営の問題として考える意義がある。オープンソースという方法から学ぶことで，組織運営のあり方や，情報をどのように管理し，同時にメンバの自習性を尊重しながら，どのようにして組織が抱えている問題を解決していくのかといった一連の課題にアプローチできるからである。

オープンソースソフトウエアの開発スタイルについては，E. S. Raymond の考察「伽藍とバザール」がよく知られている[1]。Linux 開発の成功は，オープンソースソフトウエアの開発において史上まれなその開発規模において特別な地位にある。Linux プロジェクトは，リーダの Linus Torvalds が中心となって世界中の万単位の開発者と数百万のユーザが携わったとされており（正確な数は把握し切れていない），現在でも数万から数十万の開発者を擁するコミュニティで開発が続けられている。

自由ソフトウエアの開発者であった E. S. Raymond は，この型破りな方法による Linux の成功に驚き，自分が開発していたオープンソースソフトウエアである Fetchmail をその開発スタイルに変更して，Linux スタイルの開発モデルを実地に検証してみせた報告が「伽藍とバザール」である。伽藍モデルとは，1人か2，3人の少数のコア開発者で開発を進め，一定の品質に達したと判断した時点で新しいバージョンをリリースするという従来採用されていたオープンソース開発モデルである。これに対して，開発中のバージョンを品質にかかわらずに次々とリリースし，なるべく多くの人を開発にかかわらせようとするスタイルがバザールモデルであり，E. S. Raymond は Linux 開発がまさにこのバザールモデルの成功例であるとし，バザールモデルの伽藍モデルに対する優位性を証明しようとしたのである。

## 8.1 オープンソースソフトウエアの開発

ソフトウエア開発では，ソフトウエア工学の観点からは，次の工程が含まれる[14, 7章]：

1. 要求分析
2. システム設計
3. 詳細設計
4. 実装
5. 結合テスト
6. 総合テストと実地検証
7. 保守の工程

現在のオープンソースソフトウエアの開発は，インターネットを介して分散したボランティアベースのコミュニティメンバによって行われている。オープンソースソフトウエアは要求仕様がある程度明確になってから開発される傾向にあるため，実際の開発は下流工程が主なタスクとなる。コミュニティから玉石混淆の多数のフィードバック情報が戻されてくるため，開発プロセスに管理上のリスクを回避する仕組みを取り入れる必要がある。

上記の開発工程プロセスでは，ある工程の作業が実質的に完了しない限り，次の工程には進めない。開発すべきシステムがモジュール化されたアーキテクチャを持つ場合には，依存される側のモジュール設計を依存する側の設計を待たずに開発を先行させることができる。成功しているオープンソースソフトウエアのモジュールは，分散した組織で並行した開発を可能とするために，独立性が高いように巧妙に設計されている。Linux プロジェクトのリーダ Linus Torvalds も計画の早い段階でこの重要性に気づき，成功するオープンソースソフトウエアは優れたソフトウエア設計が不可欠であることを強調している[14, 8章]。

また，ソフトウエア開発で欠かせない要素の1つがレビューで，設計品質やそれを実現するための工程やソースコードを評価し，改善点を提示し次の段階に進めるかどうかを確認するための組織的活動は開発において中心的役割をもつ。オープンソースソフトウエアは，そのプロジェクトが成功している場合には，最高水準のテストを受けており非常に高い品質を誇っている[14, 7章]。ソースコードが公開されているために，多数のプログラマがシステムをピアレビューしてバグを探し出し，致命的な問題点が報告され開発者はソースコードを修正することができる。このため，例えば DNS サーバや Web サーバなどのセキュリティクリティカルなソフトウエアでは，インハウスでなくむしろオープンソースで開発することにより標準化技術が追求され，十分なピアレビューを経ることができるオープンソースソフトウエアの方が安全で信頼できるとする考えがある[3]。ただし，オープンソースであることだけではセキュリティは確保できない[38][39]。オープンソースでは悪意をもつ開発者によるトロイの木馬の混入を阻止できず，またセキュリティホールが報告されないかもしれないからである。この意味で，プロジェクトの進め方については今後も検討を進めていく必要はあるが，オープンソースであるが故に重大な欠如が未然に避けられてきたことも事実である。

オープンソースソフトウエアの開発では次のような特徴がある[36][26, 3.4.4節]：

- 高速インクリメンタル開発による超短期リリース更新

- ソフトウエアアーキテクチャが可能としている並行開発
- 開発者がインターネットに分散した分散開発組織
- 開発者およびユーザからなるコミュニティからの直接的なフィードバック
- 独立したピアレビュー
- 高い技術力と動機をを持つ開発者の存在
- ユーザの積極的で高度なコミット

これらの特徴を持つ成功するオープンソースソース開発は，以下に述べるように，開発組織，開発支援環境と意志決定機構を含む組織運営が協調しあって達成されている[36][14, 11章]。

成功するオープンソースプロジェクトでは，目的としているソフトウエアのアーキテクチャ自体が独立性の高いモジュールに分割設計されていることに加え，コミュニティに対するソースコードの公開の仕方と開発を進める上でのプロジェクトマネジメントの成功が決定的役割を果たしている。以下，オープンソースプロジェクト組織の概要を[36]に従って簡単に紹介する。

### 8.2 開発組織

オープンソースソフトウエア開発の組織上の最大の特徴は，開発組織とそれを取り巻くコミュニティの存在にある。オープンソースソフトウエア利用の構造が拡大し，その開発スタイルが研究されるようになって，オープンソースを取り巻く組織環境は，初期の開発者＝利用者という構図から機能別に分化してきている[12]。

図2に示すように，単にソフトウエアを利用するだけの一般ユーザとは別に，オープンソースコミュニティは，オープンソースソフトウエアを生産するコア技術者集団である開発組織，その

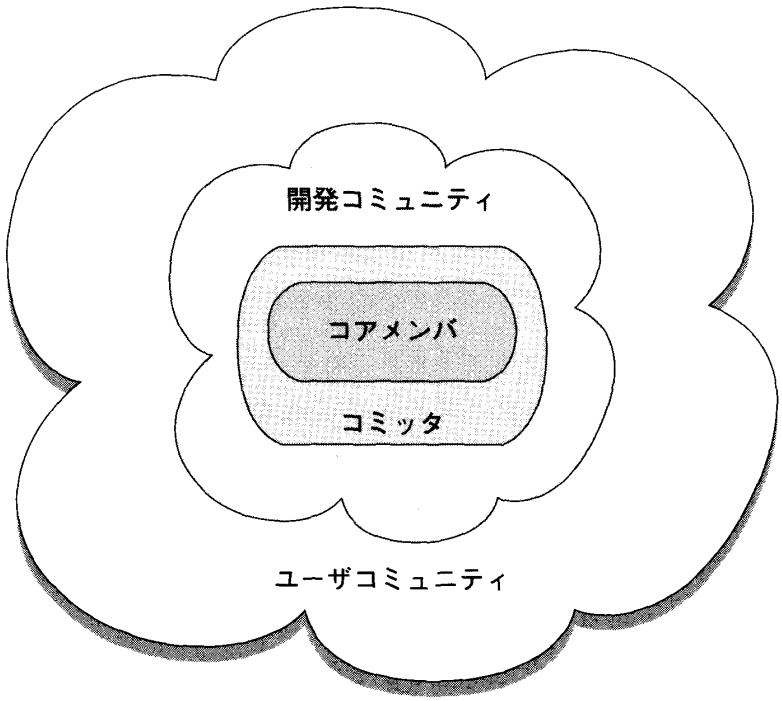


**図2 オープンソースソフトウエアのコミュニティ構造**

開発組織の周りに臨時に開発に参加する技術者群からなる開発コミュニティがある。開発組織は，オープンソースプロジェクトというボランタリーな組織ではありながら，この組織が本来持つ分散性とソフトウエア開発を成功に導くための組織力とを両立させ，しかもコミュニティからの寄与とフィードバック対応によってもたらされる首尾一貫性に対するリスクを回避するために，組織編成上の工夫が必要となる。このため，開発組織は核となるリーダのもと，開発への貢献に応じた階層的組織構成をとるのが通例である。ただし，オープンソースプロジェクトでは企業内組織のように深い階層レベルを持たない２～３層の浅い階層と，トップダウンとは異なるリーダシップによって運営される。階層のレベル数が少ないほど，組織は堅固になる。このため，Linux の Linus Torvalds や Apache の Brian Behlendorf のような優れたリーダーの存在が，オープンソースソース開発での成功を大きく左右する。

さらに開発組織の周辺部には，比較的高い利用技術をもったパワーユーザからなるユーザコミュニティがあり，バグレポートや改良の結果報告，ドキュメント作成など，ソフトウエア開発を底辺から支えている。ユーザコミュニティのメンバは，開発への寄与者という意味で，コントリビュータともよぶ。優れたユーザコミュニティを得たオープンソースソフトウエアでは，コミュニティ全体がテスト環境となることができ，開発者に正のフィードバックをもたらし，オープンソースソフトウエアの高い品質を生み出す大きな役割を担っている。また，ユーザコミュニティは新たな開発者を生みだす土壌でもあり，オープンソースソフトウエアの普及にとって大きな駆動力となっている。

開発組織の規模が大きくなると，リーダ１人でプロジェクト全体の開発運営を行うことが困難になってくる。このため，開発のある段階から集団指導体制に移行する場合が一般的である[37]。Linux では，開発の早い段階から貢献度の高いコントリビュータをメインテナ（Maintainer）として開発の一部を任せて，Linus Torvalds 自身は開発リーダとして全体の舵取りとリリース管理を行っている。また，Apache プロジェクトのように，開発の一部を任せるだけでなく，全体の舵取りやリリース管理などについても開発者と同じ権限を与えられた開発組織の最上位に位置する少数チームをコアメンバまたはコアチーム（Core team）とする場合もある。さらに，次の8.3節の開発支援環境で述べるように，サーバ・クライアント方式でソースコードを公開するソース・文書管理システムの機能向上は，開発チームの運営にも影響を与えてきた。この機能向上が，公開しているソースコードの変更権限だけが与えられたコミッタ（Committer）とよばれるコントリビュータを生みだし，開発コミュニティの活性化と拡大に寄与している。コミッタ方式は開発メンバへの参加を奨励し，コントリビュータの開発チームへの一員としての自覚を促す役目を担っている。

### 8.3　開発支援環境

オープンソースプロジェクトを世界中に散らばったコミュニティ組織で開発を進めるためには，そのための開発環境が整備される必要がある。コミュニティ間の通信手段として，電子メールや

Webなどに加えて，個人またはグループからのバグの受付と，受け付けたレポートの管理検索を支援するバグ追跡システムがある[59]。バグ追跡システムを使うと，ユーザはソフトウエアの未解決バグを効率的に追跡することができ，受付済みのバグ報告と重複しないようにレポートを開発者にフィードバックすることができる。

オープンソースプロジェクトにおける中核的環境は，ソースコードを管理するシステムである。オープンソースプロジェクトでは，多数のメンバが自由にソースコードを参照してバグフィックスをしたり改良したソースコードが集積されることになる。このため，これらの派生ソースコードの依存関係を管理し，開発者が開発しているソースコードを逐次コミュニティに公開していくための情報管理システムはオープンソースプロジェクトが備えるべき本質的機構である。ソースコードの公開方法と特定メンバへの公開ソースの変更権限の与え方や，ソースへの変更手段をどのように組織化し管理するかというポリシーと方式の選択はプロジェクトの運営を大きく左右する。

オープンソースソフトウエア開発においてE. S. Raymondが命名したバザールモデルが登場して以来，従来の伽藍モデルの開発から大きく変化した点は，ユーザがバグフィックスや改良のために利用するソースコードが，開発者が一定品質に到達したと判断した安定したリリース版ではなく，開発者がいま手にしている開発途中またはそれに近接したソースコードになったことである[36]。

伽藍モデルでは，開発者が公開に相応しいと判断したソースコードをユーザに公開していたため，次のリリース版の公開までに要する開発期間は短くはない。このため，ユーザには実際の開発状況がわからず，バグや改良点を見いだしたとしてもすでに他人や開発者が気づいて手当てしているのかどうかを判断することができず，発見したユーザが結果的には報告しないことがある。あるいは，開発者の意図する方向とは異なる方針でパッチが当てられたソフトウエアが開発者のソースとは別に広まり，ソフトウエア開発が分岐してしまう場合も出てくる。バザール方式を有名にしたLinus Torvaldsが始めたLinuxカーネルの開発では，開発中のソースコードを手元には長くとどめず，次々とリリースする開発方法をとった。このため，ユーザはLinusの手元にあるソースに近いコードを随時参照することができ，その結果，ユーザは進んでバグレポートを寄せることができ，開発が途中で分岐することもなくLinusが開発方針を決定しながら短期間で高い完成度に達することができた。この高速インクリメンタル開発はソフトウエア開発にあり方に新しい考え方を提供したのである。

Frederick Brooksは，ソフトウエア工学の立場から，ソフトウエア開発の作業はメンバ間のコミュニケーションの比重が大きく，メンバが$N$倍に増えると作業量は$N$倍になるが，一方で複雑さとバグの発生する頻度が$N^2$乗倍になり，メンバが増えても生産性は向上せず，むしろ生産にかかる時間は増大するという，今日Brooksの法則とよばれていることを見いだした[13]。Linuxカーネルの開発では非常に多数のメンバが開発に参加し，Linusの適切なリーダシップのもとでプロジェクトは短期間で大きな成果をあげ，Brooksの法則が破られているように見える。

E. S. Raymondは，オープンソースソフトウエアのバグの除去は，ソースコードを公開することによって迅速かつ効率的に発見されるピアレビューによるものであると説明している[1]。したがって，Brooksの法則は依然として正しく，成功したオープンソースプロジェクトが，多数メンバによるピアレビューなどのフィードバックによってバグ発見の劇的向上をもたらすネットワーク開発組織の仕組みを発見したと考えることができる。

オープンソースソースソフトウエアの開発において生じるソースコードや文書の変更を管理するシステムとしてBitKeeper[61]やCVS（Concurrent Version System）[60]がある。BitKeeperはLinuxカーネルの開発に，CVSはFreeBSDやApacheおよびオープンソースソフトウエア開発のホスティングサービスのSourceForge[24]などで広く利用されている。CVSは1995年のCVS1.5からサーバ・クライアント機能が導入され，ソースコードの公開方式に大きな影響を与えた（2004年10月現在でCVS1.12が最新）。この機能を使った匿名CVSサーバを用意して，開発者が使っているソースコードのCVSリポジトリを公開し，ユーザはCVSクライアントを使ってリポジトリから最新のソースを入手して検討したり，開発者がコアメンバやメインテナにリポジトリの変更権限を与えることによって，インターネットを介した分散共同開発が可能となっている。また，開発者は，コントリビュータにリポジトリの変更権限だけを与えたてコミッタという役割を開発コミュニティに加えることによって，きめ細かい多様な開発スタイルが可能となった。

### 8.4 組織運営と意思決定プロセス

オープンソースソフトウエアの開発チームの人数が増えてきた場合，ソフトウエア開発の運営方針の決定やプロジェクトの方向性を左右する大きなソースコード変更などをコミュニティとしてどのように合意し決定するかは，組織運営上の大きな課題となってくる。

Linuxカーネルの場合，巨大な開発コミュニティからのフィードバックをもとに意見を交換しコミュニティとのバランスを取りながら，リーダであるLinusが最終的な方針を決定するという「やさしい独裁者モデル」[2]の意思決定システムをもつ。一方，Apacheプロジェクトでは，「やさしい独裁者」方式をとらず，主要な案件についての意志決定は原則としてプロジェクト運営に責任をもつメンバによる投票Binding voteによって行われる[55]。重要な変更（Product Changes）の承認には，反対が1つもなく3票以上の賛成で成立する合意承認（Consensus approval）が必要である。また，リリース管理（Release Plan）関する承認には，賛成が反対よりも多く，賛成票が3つ以上含まれている場合に成立する過半数承認（Majority approval）が必要となる。これらの場合であっても，なるべくメンバ間での議論を踏まえて合意形成する傾向がある。また，PythonやPerl6の開発では，プロジェクト参加者による議論を重ねたうえで合意形成をするために，大きな変更を加えようとする場合は，その変更について説明した文書を事前公開することを求めている[65][66]。

オープンソースソースソフトウエアの開発では，単にバザールモデルでソースコードを次々に

公開しながら開発を進めるだけではプロジェクトは成功しない。コミュニティにおいて，これらの意思決定システムが支持されることが，オープンソースプロジェクト組織が有効に機能するためのキーポイントである。

## 9 企業経営の課題

成功したオープンソースプロジェクトは何故成功したのかという分析は，すこぶる経営学的課題である。組織が効果的であるためには，組織のメンバがどのような組織のために仕事をしているのかを理解していることと，最終的な決定を下せるリーダーの存在が必要である。1節で述べたように，オープンソースプロジェクトを組織体として結合させている求心力として，企業体と雇用関係にある社員や金銭的動機とは異なるインセンティブが働いている。オープンソースソースプロジェクトに参加しているメンバは，その意識においても高い動機を維持しているのである。オープンソースプロジェクトでは，ソフトウエア開発者以外の一般の技術者やユーザが，誰もが何らかの役割をもって開発に参加することができるオープンな組織性と，プロジェクトの価値を共有するある種の一体感が参加者メンバに共有されている。

以下，オープンソースソフトウエア開発にみられる開放的なメンバコミュニティを職階による序列組織と比較しながら，問題解決指向のネットワーク型企業組織の姿を考えてみる。メンバ間の評判，責任，そして相互信頼に基づい活動するネットワーク型組織モデルは，組織および構成メンバのもつ無形資産と専門知識を素早く適用して問題を解決して組織目的を達成するための企業組織のあり方に1つの解を与えていることがわかる。

### 9.1 組織構造

階層型組織とは，図3のように，従来型企業に見られる職階によるヒエラルキーに従った組織で，組織内の各メンバには階層関係（半順序関係）とそれに応じた職務が割り当てられている。グラフ理論的には，階層型組織は閉路をもたない木構造であり，任意のメンバ間を結ぶパスは唯1つだけ存在する。組織リーダを木構造のルートにおいた場合，階層組織の深さはルートから下位リンクを持たない端点までのパス数で表すことができる。

ネットワーク型組織とは，図4のように，順序関係をもたない一般のグラフ構造をもつ組織で，しかも，オープンソースプロジェクト組織がそうであるように，メンバ間のつながりが単一の上下関係にはよらず，組織機能ごとにグループ化される多重の階層性からなるメンバ間の関係性がある。

実際の企業組織は，階層型構造とネットワーク型構造との間でバランスをとったハイブリット構造の組織形態をとるように工夫されているが，オープンソースプロジェクトのネットワーク組織は組織の動的対応能力において企業組織とは異なる。以下で見るように，成功するオープンソースソースプロジェクトにおける組織運営は，たびたび指摘してきたように，その動的組織形

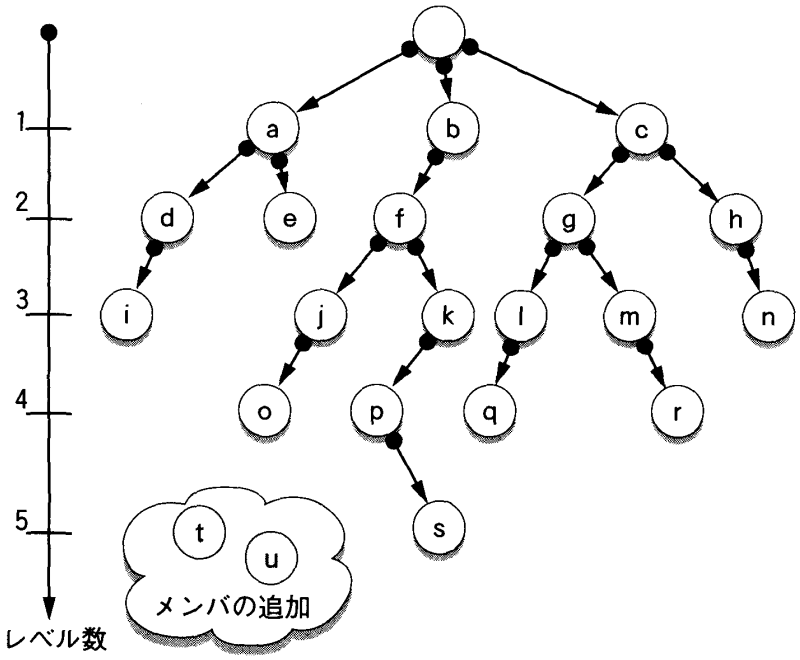


**図3　階層構造をもつ組織**

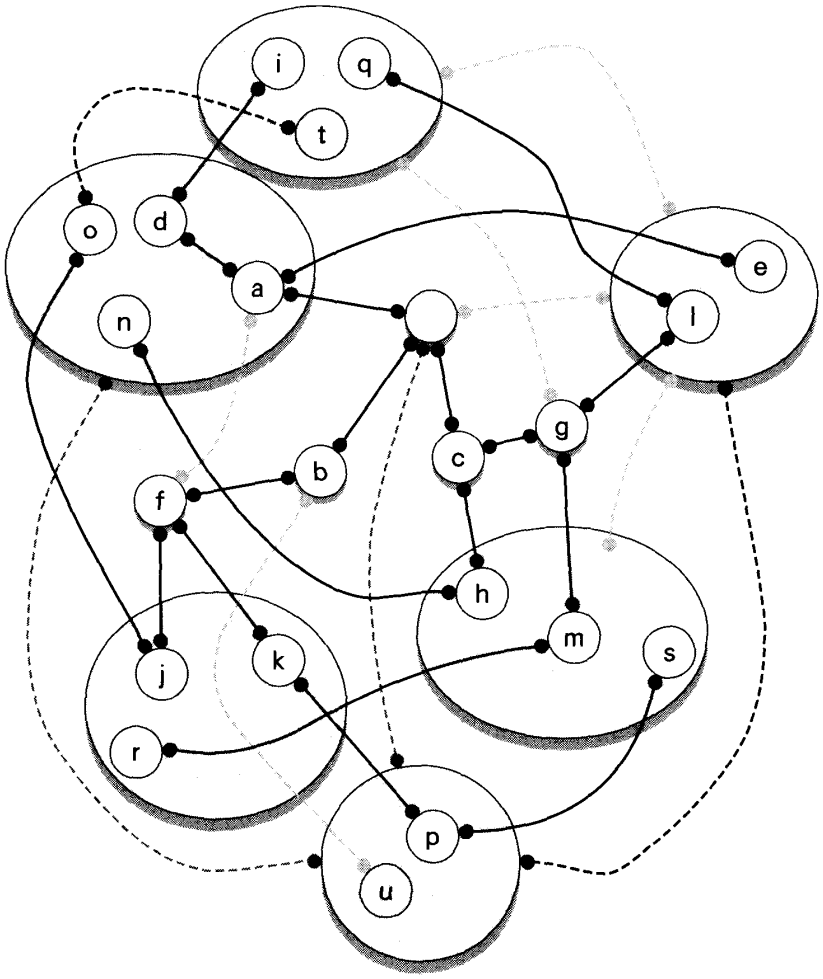


**図4　ネットワーク組織**

成能力にその最大の特質がある。この特質は，組織形態という形式的な構造だけではなく，組織リーダの組織運営に対する資質やメンバに対する求心力，そして各メンバ個人の高い志という人間的な要因も大きく働いている。

階層型組織では，各メンバは自分の役割と上限関係にあるメンバとのやりとりの方法さえ知っていれば，組織機能がその時点の目的に合致し安定しているか，あるいは仮に不安定であっても非常に強権的トップダウンリーダシップ機能が働いていれば，各メンバは組織構造全体の知識や各部分構造の機能や役割を理解する必要はない。組織形成にかかるコストは，立案に参加する組織のコアメンバ以外のメンバにはほとんどゼロである。

ネットワーク型組織では，その目的に応じてその都度さまざまな人間が集まってメンバ構成が定まり，互いに協力しながら目的を達成しようとする動的対応能力が高い。このため，成功するネットワーク組織のメンバには，たとえばメンバの自己目的が組織目的と部分的に重なり合っていたり，組織への参加の強い動機とその持続など，組織の目的に応じた一定の資質が求められる。各メンバはネットワーク組織の目的をよく理解し，自分だけでなく可能な限りの他のメンバの役割を理解しておく必要がある。したがって，各メンバは組織の目的に合致するように動く必要があり，組織形成のコストはコアメンバ以外にも求められる。

### 9.1.1 組織内のメッセージ伝達

組織内を伝搬するメッセージの流れと効率性を考えてみる。階層型組織では，階層内の各メンバにはその上位メンバと下位メンバが存在するというピラミッド型の上下関係が規定されている。この関係を通じて，指令が下位構造を管理し，下位メンバからは上位メンバへとメッセージが集約されていく。このため，9.1.2節で述べるように，メッセージ伝達の非対称性が発生する。

階層型組織のメッセージ伝達では，上位からのメッセージは中間位にあるメッセージを通じて，そのままメッセージが下位構造に向けて効率よく流れていくため，強い組織リーダと優れた組織戦略によってメンバが配置されている場合には，このトップダウン方式は大きな能力を発揮する。一方，下位メンバからの意見の集約プロセスでは，複数の下位メンバからのメッセージは一旦中位メンバに集まり，中位メンバにより選択・翻訳されて上位メンバにパスされる。階層構造の深さを $n$ としたとき，階層型組織のメッセージ伝達は，トップから最下層メンバには $n$ 回のメッセージパスが必要で，組織内の任意のメンバ間同士のメッセージ伝達には最悪 $2n$ 回のパスが必要となる。階層型組織では，下位メンバであるほどプロジェクトの現場に近く，下位メンバ間のコミュニケーションは，プロジェクトに含まれる問題の発見と解決には欠くことができない。階層組織モデルにおいては，メンバ間でのコミュニケーションに要するコストは高い。

一方，ネットワーク型組織では，各メンバは必ずしも上位・下位関係が定まっていない。階層型組織と違い，組織内の任意のメンバ間を結ぶパスは複数あり，他のメンバとの関係性において複数のチャンネルをもつことが組織上保証されている。ネットワーク型組織のメンバ間の平均メッセージパス数（平均距離）は小さく‘Small Word 効果’があることが知られている。たと

えば，世界で10億以上あるWebページ間の平均距離$d$は，Webページ総数を$N$としたとき実験的に$d=0.35+\log_{10}N$が成立し，1998年末の時点で$d=$約19，つまり任意のWebページ間は19クリック離れているという報告がある[8][6]。特に，メンバの周辺近傍を飛び越えて‘遠くの’メンバへのランダムなリンクがある場合，たとえそのような遠方リンクの頻度が小さくともメンバ間の平均距距離は劇的に減少することが知られている[5][7]。したがって，メッセージ伝達に関しては，階層型組織に比べてネットワーク組織の方が効率がよい。

### 9.1.2　組織の意思決定プロセス

組織構造は意思決定プロセスのあり方にも大きな影響を与える。階層型組織ではメンバは職階などの上下関係のなかに組み込まれ，あるメンバからからの上方または下方のメンバへの伝達に際して，直接の上位および下位メンバはインターフェースの役割を果たしている。このため，上位（または下位）に広がっているメンバ群には直接にコミュニケートすることはできない。上方からのメッセージは下方に向けて，そのまま直接伝えること可能であるが，複数の下位メンバからの意見を受けた上位ノードにあるメンバは，それらの意見を選択・翻訳して上位メンバにメッセージを渡すことになり，階層型組織ではメッセージ伝達の非対称性は生じる。下方からのメッセージをそのまま上方にパスする組織モデルもありえるが，その場合，中間位としての機能を失うか，または責任回避の仕組みになりやすい。したがって，階層型組織の意思決定プロセスではトップダウン方式がもっとも効率がよく，正確に伝達される。

階層型組織の原理的に有するメッセージ伝達の非対称性のために，下位組織からボトムアップ方式で意見を集約しながら組織全体の意思決定を行うことは，下位のメッセージを上位に渡すためのメタ表現の方法が確立していないために容易でない。上位レベルへと集約する各段階で，各メンバは自分の役割を上位および下位メンバとの相対的な役割として知るだけで，先に述べたように階層型組織では，組織全体および部分組織の目的を正確に把握して行動することは，軍隊組織がそうであるように，本来求められていない。また，個別メンバの局所的な意見集約が，必ずしも組織全体の意思を代表していないというパラドックスも生じる場合もあり得る。したがって，階層型組織で有効な意思決定を行うためには，オープンソースプロジェクトが採用しているように，階層の深さを浅くする必要がある。

これに対して，ネットワーク型組織では，下位上位の概念が希薄で，しかもメンバ間のリンクは複数のチャンネルがある。このため，各メンバは，組織全体のメッセージの流れにさらされながら，組織内の自分の役割と機能を常時確認することができる。成功するネットワーク型組織では，メンバは組織全体の状況を把握することが求められる。その結果，ネットワーク型組織では，意思決定はコンセンサスによって行われる。組織が‘やさしい独裁者’によって率いられているならば，最終的な組織の意思決定をその‘独裁者’にゆだねるというコンセンサスがすでにメンバ間にできあがっている。やさしい独裁者をもたないネットワーク型組織では，コアメンバまたはメンバ全員によるコンセンサスによって意思決定する。ただし，コンセンサス自体に曖昧さが

混入する可能性がある。実効性ある意思決定をするために，各メンバ，特にコアメンバが組織の目標を理解した上で組織運営にコミットし自分の責任を果たすというモラル文化が組織に根付いていることが前提となる。

### 9.2 組織の動的柔軟性

オープンソースプロジェクトは，ソフトウエアの生産という明確な目的のもとに，その実現と開発に必要な技術や熱意を持った協力者や企業が集い，一時的に連携してネットワーク組織を形成する。オープンソースプロジェクトは，目的とするソフトウエアの生産という問題解決指向のネットワーク型組織で，問題の規模や種類，そして組織を率いるリーダの資質に応じてネットワーク型組織形態があり，その逆ではない。図4で示したように，問題が与えられているネットワーク型組織では，必要に応じて適宜メンバをグループ化するコストは低く，さまざまな問題に柔軟に対応することができる。

一方，階層型組織においては，個別プロジェクトに応じて職階や機能などの階層組織構造を再構成してプロジェクトに臨むことはまれで，組織構造は変えずにプロジェクトを実施することが検討される。言い換えれば，階層型組織は，具体的な問題解決指向の組織ではなく，組織が将来対応するであろう問題群を解決するために事前に用意された，抽象的に機能分化しておいた組織と考えることができる。

オープンソースプロジェクトでは，プロジェクトを進める中での状況変化，たとえばプロジェクト内部の方針の変更やプロジェクト外の市場や協力企業の状況変化に応じて，自立的に問題解決がなされる。重要なことは，この問題解決が，組織全体を組み替えることによってではなく，組織の局所的部分だけを変更することによって行われることにある。成功するネットワーク型組織では，状況の変化に対してネットワークの局所的変更によって組織全体の問題を解決できるように，あらかじめメンバに高度な機能単位であるように権限と責任というインセンティブを与えている。言い換えれば，このように設計されているネットワーク型組織のメンバであるためには，その活動において他のメンバと信頼と責任で結びついていることが要求される。

オープンソースプロジェクトでは，メンバはプロジェクトへの貢献としての自己実現感，他のメンバによる業績評価，およびに信頼という名誉によって報われる。信頼と業績で維持されるネットワーク型組織では，さまざまな役割をもつ各様な個性をもったメンバが集まって，共通の目標に向けて相互協力し，状況が変化した場合には即応するメンバからなる部分的なネットワーキングの変更だけで事態の変化に対応することができる。

一方，階層型組織では，メンバの役割は事前に職階や抽象的機能に応じて固定的に割り当てられ，メンバは厳格に管理されている。このため，プロジェクトに対して大きな状況変化が生じた場合，現在の組織構造をこの変化に最適に対応できる構造へと変更するには大きなコストと時間を要する。組織変更のために，妥当な職階と機能を有する階層モデルを再設計して，新しい階層組織のそれぞれの場所にメンバをマッピングしなければならない。

## 10 まとめ

本論文では，オープンソースソフトウエアの概念を紹介し，あらたなソフトウエアビジネスの現状を踏まえた上で，企業組織のあり方としてオープンソースプロジェクトで行われているネットワーク型組織を考察した。

1節では，オープンソースソースソフトウエア問題を考える上での本論文の問題意識を述べた。本論文は，オープンソースをソフトウエア技術の問題ではなく，経営の問題としてとらえている。

2節では，オープンソースソフトウエアが提起している問題の広がりを概括し，オープンソースソフトウエアはすでに政治や社会文化の問題でもあることを述べた。日本では，オープンソースソフトウエアへの取り組みは欧米のそれと比べてまだ活発ではなく，これからの発展が期待される。知的資源のオープンソース的認識は，今後の社会資産における中心的課題となっていくだろう。3節では，代表的オープンソースソフトウエアを紹介した。オープンソースソフトウエアは特にインターネットサーバなどの広く利用されており，今後はエンドユーザ向けのアプリケーションの開発も積極的に行われていくことになる。4節では，急激に展開しているオープンソースビジネスを紹介し，その背景を述べた。インターネット利用の普及は，コンピュータによるソフトウエア利用という概念を超えてインフォウエアを生み出し，結果として，ソフトウエアの販売価値の減少を招いている。5節では，オープンソースソフトウエアの定義とユーザの権利について述べた。自由ソフトウエア運動の歴史を受けて，我々の社会はソフトウエアを複製し配布する権利，ソースコードを入手して改良する権利を認めるオープンソースソフトウエアを広く認知するようになったのである。6節では，オープンソースソフトウエアのライセンス問題を述べた。オープンソースソフトウエアの問題は，まずライセンスの問題である。オープンソースソフトウエアには，ひとたび自由であると宣言したソフトウエアからの派生ソフトウエアは将来にわたって自由が保証されるコピーレフトなGPLなど，さまざまなライセンス形態がある。ビジネスや社会資源として活用していくためにはこれらのライセンスの正しい認識が必要である。7節では，オープンソースビジネスの代表的なモデルを述べた。企業自らが開発したプロプライエタリソフトウエアをオープンソースソース化したり，オープンソースコミュニティに寄贈するなどオープンソースへのコミットが活発化している。この傾向について考察した。

以上の論考で，オープンソースソフトウエア全般の課題を明らかにした。これらを踏まえて，以下の節で経営の問題としてのオープンソースソフトウエアを論考した。8節では，オープンソースソフトウエアの開発組織を分析した。オープンソースソフトウエアの開発の特徴を述べ，オープンソースソフトウエアの開発環境を開発組織，開発支援環境および運営組織と意思決定プロセスとから紹介した。9節では，企業経営の課題として企業組織の姿を分析した。従来の階層型組織とオープンソースプロジェクトが採用しているネットワーク型組織を取り上げ，その組織構造をメッセージ伝達と意思決定プロセスの観点から考察した。また，動的に柔軟な組織を問題

解決型の組織として分析した。

成功したオープンソースソフトウエアのプロジェクトメンバは，プロジェクトの目的に共感し，自由意志で参加している。メンバのインセンティブは，プロジェクトに貢献することができたという自己達成感とメンバ間の評判と名誉である。プロジェクトは，メンバの自覚的な責任とメンバ間の相互信頼に基づいて自発的に活動するネットワーク型組織である。

このような組織を企業組織として実現することが可能だろうか，これが本論文の出発点であった。企業に属する社員が，その組織のもとで働き続ける根拠はあるのだろうか。一方，企業経営者にとって，雇用者が転職することなく長年働けるように努力することは企業発展にとって有利であるという根拠はあるのだろうか。オープンソースプロジェクトに参加する技術者は，ソフトウエアの開発という同じ仕事に対して，企業組織と企業外のオープンソースコミュニティという2つまたはそれ以上の社会組織に属している。

なぜ，技術者はオープンソースプロジェクトに参加するのだろうか。技術者が，自己実現を求めて自由意志でプロジェクトに参加し，名誉と信頼に基づいた組織—それは科学研究における科学者の社会に酷似している—での活動を望むのであれば，その技術者の視線は自らが属する企業組織を超えている。そのとき，社員が経済的条件を含むより有利な環境を提示する企業へと去っていくことを止めるものはもはやないだろう。これからの企業組織が抱える本当の課題はここにある。

**参考文献**


[1] Eric S. Raymond,「伽藍とバザール（*The Cathedral and the Bazaar*）」(1997年), http://cruel.org/freeware/cathedral.html

[2] Eric S. Raymond,「ノウアスフィアの開墾（*Homesteading the Noosphere*）」(1998年), http://cruel.org/freeware/noosphere.html

[3] Eric S. Raymond,「魔法のおなべ（*The Magic Cauldron*）」(1999年), http://cruel.org/freeware/magicpot.html

[4] Ed Scannell, *Linus Torvalds says open source not a guarantee of success* (1999年), http://www.infoworld.com/cgi-bin/displayStory.pl?99106.pitorvalds.htm

[5] D. J. Watts, *Small Worlds–The Dynamics of Networks between Order and Randomness*, Princeton University Press(1999)

[6] A.–L.Barabási, *LINKED: The New Science of Networks,* Persus Publishing (2002)

[7] D.J. Watts, *Six Degrees–the sicence of a connected age*, Norton and Company(2002)

[8] R. Albert, H. Jeong, and A.–L. Barabási, *Diameter of the World–Wide Web*, Nature 401 (1999). pp.130–131

[9] L. レッシグ,「CODE～インターネット合法・違法・プライバシー」, 翔泳社 (2001年)

[10] L. レッシグ,「コモンズ～ネット上の所有権強化は技術革新を殺す」, 翔泳社 (2002年)

[11] L. レッシグ,「FREE CULTURE」, 翔泳社 (2004年)

[12] 経済産業省,「オープンソースソフトウエアの利用状況調査／導入検討ガイドライン」(平成15年), http://www.meti.go.jp/kohosys/press/0004397/

[13] F. P. Brooks,「人月の神話—狼人間を撃つ銀の弾はない」(1975年), ピアソンエデュケーション (2002年)
[14] C. デボナ, S. オックマン, M. ストーン編著,「オープンソースソフトウエア」, オライリー・ジャパン (1999年)
[15] 川崎和哉編著,「オープンソースワールド」, 翔泳社 (1999年)
[16] 米持幸寿著,「オープンソースがビジネスになる理由」, 日経BP社 (2003年)
[17] Jan Sandred,「オープンソースプロジェクトの管理と運営」, オーム社開発局 (2001年)
[18] 名和小太郎,「ディジタル著作権」, みすず書房 (2004年)
[19] 國領二郎監修,「Linux いかにしてビジネスになったか」, NTT出版 (2000年)
[20] Internet Society (ISOC), http://www.isoc.org/
[21] IAB (Internet Architecture Board), http://www.iab.org/
[22] IETF (Internet Engineering Task Force), http://www.ietf.org/
[23] OSDN, http://www.osdn.com (日本語サイト OSDN Japan, http://osdn.jp/)
[24] SourceForge, http://sourceforge.net/ (日本語サイト SourceForge.Jp, http://sourceforge.jp/)
[25] e-Japan 重点計画—2004, http://www.kantei.go.jp/jp/singi/it2/kettei/ejapan2004/040615honbun.html
[26] AITEC,「わが国が行う情報技術開発のあり方に関する調査研究 (その7)」, http://www.icot.or.jp/FTS/REPORTS/H14-reports/PDF/H14-report-1.pdf
[27] 高度情報通信ネットワーク社会推進戦略本部 (IT戦略本部)」, http://www.kantei.go.jp/jp/singi/it2/
[28] 三菱総研,「オープンソースソフトウェア技術者の人材評価に関する調査 (2004年)」(経済産業委託調査), http://oss.mri.co.jp/reports/florist/index.html
[29] 国連貿易開発会議 (UNCTAD), *E-Commerce and Development Report: Chapter 4(2003)*, www.unctad.org/en/docs/ecdr2003ch4_en.pdf
[30] 情報社会世界サミット2003, http://www.itu.int/wsis/documents/doc single-en-1161.asp
[31] 世界各国政府のオープンソース採用動向, http://oss.mri.co.jp/news/
[32] Pooling Open Source Software(2002 June), http://europa.eu.int/ISPO/ida/export/files/en/1115.pdf
[33] BBC Creative Archive, http://www.bbc.co.uk/pressoffice/pressreleases/stories/2004/05 may/26/creative_archive.shtml
[34] Creative Commons, http://creativecommons.org/
[35] ミッション期間の延長で生きながらえる火星のOSS (2004年9月30日), http://japan.linux.com/opensource/04/10/04/0257212.shtml?topic=1
[36] 特集オープンソースソフトウエア, 情報処理43 No.12 (2002), p.1317, 情報処理学会
[37] A. Mockus, R.T. Fielding, and J. Herbsleb, *A Case Study of Open Source Software Development:The Apache Server*, Proc. ICSE 2000, IEEE CS Press, pp.263-272(May 2000), http://opensource.mit.edu/papers/mockusapache.pdf
[38] N.W. Whitlock,「オープン・ソースは, オープン・ドアか?」, http://www.ibm.com/jp/developerworks/linux/010615/j l-oss.html
[39] John Viega,「ソース開示ソフトウェアで, 安全を確保できるか?」, http://www.ibm.com/jp/developerworks/security/sec oss_security.html

[40] Open Source Initiative, http://www.opensource.org/
[41] The Open Source Definition, http://www.opensource.org/docs/definition.php.
(非公式日本語訳：八田真行氏 http://www.opensource.jp/osd/osd-japanese.html)
[42] OSI 認可のオープンソースライセンス一覧, http://www.opensource.org/licenses/
[43] The Halloween Documents, http://www.opensource.org/halloween/.
(日本語訳 http://cruel.org/freeware/halloween.html)
[44] オブジェクト指向スクリプト言語 Ruby, http://www.ruby-lang.org/ja/
[45] KAME Project, http://www.kame.net/
[46] USAGI Project-Linux IPv6 Development Project, http://www.linux-ipv6.org/
[47] GNU Operating System-Free Software Foundation, http://www.gnu.org/
[48] GNU General Public License, http://www.gnu.org/copyleft/gpl.html
[49] The BSD License, http://www.opensource.org/licenses/bsd-license.php
[50] X11 Terms and Conditions, http://www.x.org/Downloads terms.html
[51] The MIT License, http://www.opensource.org/licenses/mit-license.php
[52] FreeBSD Copyright, http://www.freebsd.org/copyright/
[53] The Apache Software Foundation, http://www.apache.org/
[54] Apache License, http://www.apache.org/licenses/
[55] Apache Voting Process,http://www.apache.org/foundation/voting.html
[56] NetcraftWeb Server Survey, http://news.netcraft.com/archives/2004/08/index.html
[57] FreeBSD, http://www.freebsd.org/, (日本語サイト http://www.freebsd.org/ja/index.html)
[58] Mozilla Public License, http://www.mozilla.org/MPL/
[59] はじめてのバグジラ, http://www.mozilla.gr.jp/docs/beginbugzilla/
[60] CVS home, https://www.cvshome.org/
[61] BitKeeper, http://www.bitkeeper.com/
[62] FireFox, http://www.mozilla-japan.org/products/firefox/
[63] GNU のさまざまなライセンスとそれらについての解説,
http://www.gnu.org/licenses/license-list.ja.html
[64] Artistic License, http://www.artisticlicence.com/
[65] PEP Purpose and Guidelines, testtthttp://www.python.org/peps/pep-0001.html
[66] Perl6 RFC Index, http://dev.perl.org/perl6/rfc/
[67] Apple Public Source License, http://www.opensource.apple.com/apsl/
[68] Darwin, http://developer.apple.com/darwin/
[69] IBM Public License, http://www.research.ibm.com/jikes/license/license3.htm
[70] Linux Professional Institute, http://www.lpi.org/
[71] IBM alpha Works, http://www.alphaworks.ibm.com/
[72] IBM Developer Works, http://www.ibm.com/jp/developerworks/