魔法のおなべ

The Magic Cauldron

Eric S. Raymond 著　　翻訳: 山形浩生 *

1999/06/25 22:48:58
1999/07/02 訳了

概 要

この論文は、オープンソース現象に生じ発展しつつある、経済的な地層を分析したものだ。まずはプログラム開発費用の手当について、よく主張される神話をいくつか論破しよう。さらにオープンソースでの協力体制の安定性についてゲーム理論分析を提出する。さらにオープンソース開発の資金手当を維持するビジネスモデルを 9 種類提出する。うち 2 つは非営利、7 つは営利目的のモデルだ。続いて、クローズド であることが経済的にみて合理的な場合について、定性的な理論を展開する。そして、営利目的のオープンソース開発に費用手当をするために、市場が開発しつつある斬新な追加メカニズムを検討しよう。ここにはパトロン制の再発明とタスク市場が含まれる。最後に、将来についてとりあえずの予測をいくつか行う。

目 次

1 魔法と区別がつかない

ウェールズの神話では、女神セリドウェンはすばらしいおなべを持っていて、それは滋養あふれる食べ物を魔法で産み出してくれる——ただし女神だけが知っている呪文を唱えれば。現代科学では、バックミンスター・フラーが「ephemeralization」という考え方を与えてくれた。あるテクノロジーの、初期のデザインで採用された物理的なリソースが、どんどん情報によって置き換えられることで、ますます強力になると同時に安くなる現象のことだ。アーサー・C・クラークは「ある程度以上に進歩したテクノロジーはすべて魔法と区別がつかない」という知見によって、この両者をむすびつけた。

多くの人にとって、オープンソース・コミュニティの成功はありえそうにない魔法の一種に見える。高品質のソフトが「無料（フリー）」で現れてくるなんてのは、続けばすばらしいことではあるけれど、競争と稀少資源が支配する現実世界では、長続きするとはとても思えない。なんか落とし穴があるはずだ。セリドウェンのおなべは、ただのインチキ手品なんだろうか。そして、そうでないならこの文脈での ephemeralization はどういうふうに機能しているんだろう——女神の唱える呪文というのはなんなんだろうか。

2　贈与するおたくたちを超えて

オープンソース文化の経験は、それ以外のところでのソフト開発について学んだ人たちを困惑させたのはまちがいない。『伽藍とバザール』[1]は、分散化された共同ソフト開発がどのようにしてみごとにブルックスの法則を打破し、個別プロジェクトにおいて、前例のないほど高い信頼性と品質を実現するのかについて記述した。『ノウアスフィアの開墾』[2]は、「バザール」形式の開発が置かれている社会力学を検討して、それを理解するいちばんいい方法は、従来の交換経済の考え方ではなく、人類学者たちが「贈与文化」と呼ぶもの、つまりその構成員たちがものをあげてしまうことで地位を競い合う文化として考えることだ、と論じた。今回の論文では、まずソフト生産の経済についてのよくある誤解を論破するところから始めよう。そして『伽藍とバザール』『ノウアスフィアの開墾』の分析を経済学、ゲーム理論やビジネスモデルの領域にまで展開し、オープンソース開発者たちの贈与文化が、交換経済の中でどのように維持可能かを理解するための、新しい概念上のツールをつくりだしていこう。

気を散らさない形でこの線にそった分析を続けるためには、「贈与文化」レベルでの説明を放棄する（というか少なくとも一時的に無視する）ことが必要になる。『ノウアスフィアの開墾』は、贈与文化的な行動が起こるのは生存に必須の財がたっぷりあるせいで交換ゲームがおもしろくなくなったときだ、と主張した。でも、これは行動の**心理学的な**説明としてはとても強力だけれど、オープンソース開発者たちが実際に活躍している、入り交じった**経済的な**文脈の説明としては不十分だ。ほとんどの開発者にとって、交換ゲームは魅力はないけれど、でも制約条件としての力はまだ健在なんだ。開発者たちの行動が、充分な物質的・稀少性にもとづいた経済的になりたつものでないと、贈与文化をささえるだけの余剰の水準にはいられなくなってしまう。

だからこんどは、オープンソース開発をささえる協力と交換のモードについて（完全に稀少性経済の枠組みの中で）考えてみよう。そうすることで、「これでどうやって儲けるの？」というプラグマティックな質問に答える。でもその前にまず、この質問の背後にある居心地の悪さのほとんどは、ソフト生産の経済学についての世間的な思いこみから生まれたもので、そしてその思いこみが

[1] The Cathedral and the Bazaar
原文 http://www.tuxedo.org/ esr/writings/cathedral-bazaar/
邦訳 http://www.post1.com/ hiyori13/freeware/cathedral.html
[2] Homesteading the Noosphere
原文 http://www.tuxedo.org/ esr/writings/homesteading/
邦訳 http://www.post1.com/ hiyori13/freeware/noosphere.html

事実に反していることを見ていこう。

（検討をはじめるまえに、最後にひとこと。この論文ではオープンソース開発を論じ、それを支持しているけれど、だからといって、これをクローズソース開発が根本的にまちがっているという議論だとは受け取らないでほしいし、ソフトウェアにおける知的所有権に反対する論だとか、「共有しましょう」という愛他主義的な訴えだとか思ったりもしないでほしい。こうした議論は、オープンソース開発コミュニティの声高な少数派がいまでも大好きなものではあるけれど、『伽藍とバザール』以来の実績は、それが不要だということを明らかにしている。オープンソース開発を支持するための必要十分な議論は、そのエンジニアリング上の成果と経済的な結果だけをもとに展開できる――高品質、高信頼性、低コスト、そして選択肢の増加だ）

3　製造業的な誤解

まずはじめに認識すべきなのは、コンピュータのプログラムは、ほかのすべてのツールや資本財と同じように、二種類のまったくちがった経済価値を持つ、ということだ。いずれも**利用価値**と**販売価値**を持っているわけだ。プログラムの**利用価値**というのは、それをツールとして使うときの経済価値だ。プログラムの**販売価値**というのは、それを販売できる商品として見たときの価値だ（専門の経済学用語でいえば、販売価値は消費財（final good）としての価値で、利用価値は中間財としての価値、ということになる）。

経済学でソフト生産について論じようとするとき、ほとんどの人は「工場モデル」を仮定することが多い。このモデルは、以下の根本的な前提に基づいている。

1. ほとんどの開発者の開発時間に対する賃金はその販売価値に基づいて決まる。
2. ソフトの販売価値は、その開発コスト（つまり、その機能を再現するのに必要な資源のコスト）とその利用価値に比例する。

つまり、人はソフトウェアの価値の特徴が、ふつうの工業製品と同じだとつい想定しがちだということだ。でも、いまの想定はどっちも明らかにまちがっている。

まず、販売用に書かれるコードは、プログラミング氷山の一角でしかない。マイクロコンピュータ以前の時代には、世界中のコードの 90%は銀行や保険会社のインハウスの自社開発ソフトなのが通例だった。たぶんこれはいまではちがっているだろう――ほかの産業分野がもっともっとソフト集約的になってきたし、それにともなって金融業界の占める比率も下がってきたはずだ――でもこのすぐ後で、コードの 95%くらいはいまでも インハウスで書かれているという経験的な証拠を示そう。

この部分のコードには、MIS（経営情報システム）のほとんどの部分、つまりあらゆる大中企業が必要とする、財務・データベースソフトのカスタマイズが含まれる。デバイスドライバのような、技術的に専門性の高いコードも含まれる（デバイスドライバを売って儲けている人はほとんどだれもいない。この点はまた後でふれる）。工作機械からジェット旅客機、自動車や電子レンジや

トースターなど、各種の機械はますますマイクロチップで制御されるようになっているけれど、そのためのいろんな機器組込用コードもここに入る。

ほとんどのインハウス開発のコードは、その環境と密に統合されているので、それを再利用したりコピーしたりするのはとてもむずかしい（ここでいう「環境」というのがビジネスオフィスでの事務手続きであっても、コンバイン刈り取り機の燃料噴射システムであっても話は同じだ）。だから、環境が変わるにつれて、ソフトをそれに適合させるために、たえずいろんな作業が継続的に必要になってくる。

これは「メンテナンス」と呼ばれていて、どんなソフトエンジニアでもシステムアナリストでも、これがプログラマの賃金の大部分（75%以上）を占める点には同意するはずだ。そしてこれにともなって、ほとんどのプログラマの労働時間が割かれるのは（そしてプログラマの給料の大部分を占めるのは）、まったく販売価値のないインハウスのコーディングやメンテナンスだということになる。これは、新聞の求人欄のプログラマ募集広告を見てみればすぐに確認できることだ。

そこらの新聞の求人欄をながめてみるのは、なかなかに啓蒙的な実験なので、読者のみなさんも是非やってみてほしい[3]。「プログラミング、データ処理、ソフトエンジニア」の欄で、ソフト開発に関連した求人を見てみよう。そこで開発予定のソフトが、社内利用か販売用かで求職を分類してみること。

すぐにわかるのが、「販売用」といういちばん広いくくりを使っても、20の求職のうちで少なくとも19は利用価値のみのために支払われる仕事だということだ（つまり、中間財としての価値に支払いが行われているということになる）。ソフト産業のうちで販売価値に基づいて動いているのがたった5%だと考えられるのは、これが根拠だ。しかしながら、この論文での分析はこの比率の数字にはあまり影響されない。これが15%か、ひょっとして20%だったとしても、経済的な意味合いは基本的には変わらない。

（技術関係の会議で話をするときには、ぼくはまず質問を二つすることが多い：聴衆のうちで何人がソフト書きで収入を得ているか、そしてその中で何人の給料が、そのソフトの販売価値に依存しているか。最初の質問では無数の手が上がるけれど、二番目の質問では、手はほとんどかまったく上がらない。そしてこの比率に、聴衆のかなりの部分が驚くのがふつうだ。）

第二に、ソフトの販売価値がその開発コストや代替コストと関係しているという理屈は、消費者の実際の行動をみればもっと簡単に撃破できる。こういう形の関係が（減価償却前に）成り立つ財というのはいろいろある——食物、車、工作機械などだ。物理的な実体のない財でさえ、販売価値が開発コストや代替コストと強く結びついているものはある——たとえば音楽や地図やデータベースの複製権など。こうした財は、最初のベンダーがいなくなっても販売価値が維持されるし、その価値が高まることだってある。

ところがソフトウェア製品のベンダーが倒産・廃業すると（あるいはその製品がうち切られただけでも）、消費者がその製品に対して支払う値段の上限は、その理論的な利用価値や、機能的に同

[3]訳注：アメリカでは、新聞の求人欄が新聞本体並に分厚くて本格的になっているからこれができる。日本だと、技術系の求人情報誌でやってみるといい。

じものを開発するコストとはまったく関係なしに、すぐにゼロに近づいてしまう（この主張を確かめたければ、手近なソフト屋さんの売れ残り商品の棚を見てごらん）。

ベンダーが倒産したときの小売業者の行動はとっても示唆的だ。その行動をみれば、かれらがベンダーの知らないことを知っていることがわかる。小売業者の知っていることは：消費者が支払う価格の上限は、**そのベンダーのサービスの期待将来価値である**ということなんだ（ここでいう「サービス」は、拡張やアップグレード、後続商品を含む広い概念になる）。

いいかえると、ソフトウェアは実はほとんどサービス産業なのに、製造業だという強力だけれど根拠レスな幻想に基づいて運営されているわけだ。

なぜぼくたちは、こういう現実と異なる思いこみをしがちなのか考えてみるといい。それは単純に、販売のためにソフトを作っているソフト産業のわずかな部分が、製品を広告宣伝する唯一の部分だからなのかもしれない。それに、いちばん目について、派手に宣伝されているソフトウェア製品は、ゲームみたいなあぶく製品で、サービスを継続する必要性がほとんどないものだからかもしれない（これはあくまで例外的な製品だ）。

それと気にとめておくべきこととして、この製造業的な誤解のおかげで、開発コストの実際の構成比と病的なまでに乖離した価格構造が採用されているという点がある。もし（一般に認められているように）、ふつうのソフトプロジェクトのライフサイクルコストのうちで 75%がメンテナンスやデバッグや拡張に費やされるのなら、いまみたいに購入価格は高く固定しておいて、サポート料は低いか無料にするというよくあるやり方は、どう考えても万人にとってよからぬ結果になるはずだ。

消費者も損をすることになる。ソフトはサービス産業なのに、工場モデルのインセンティブはあらゆる面で、ベンダーが**まともな**サービス提供を行う妨げにしかならないからだ。もしベンダーのもうけがビットの販売からくるなら、能力のほとんどはビットをつくってそれを出荷することにあてられる。ヘルプデスクは、収益部門ではないから、いちばん能力のない従業員のはきだめになり、顧客の大多数を完全に追い払わない程度の、ギリギリ最低限の資源しかまわされないことになる。

そのまさに裏返しとして、この工場モデルを採用しているベンダーのほとんどは、長期的には失敗することになる。固定価格からの収入だけで、果てしなく続くサポートコストを維持するのは、市場が急速に拡大していて、昨日の売り上げから生じるサポートコストやライフサイクルコストを、明日の収入でカバーできる場合に限られる。いったん市場が成熟して売り上げ成長が鈍ってきたら、ほとんどのベンダーはその製品を見捨てることでコストを削るしかなくなってしまう。

これがはっきりわかる形で行われるか（つまり製品をうち切るか）、あるいはそれとなくやるか（サポートがなかなか得られないようにするか）、どっちにしても、顧客は競合相手のほうに行ってしまうことになる（なぜかというと、それはそのサービスにかかっている期待将来価値を破壊してしまうからだ）。短期的には、バグフィックス版を新製品と称して発表して、新しい値札をつけることでこの罠を逃れられるだろう。でも消費者はすぐにこれにうんざりしてくる。だから長期的には、唯一の方法は競合相手を持たないことだ——つまり、市場を実質的に独占してしまえばいい。最終的には、残るのは一人だけしかあり得ないんだ。

そして確かに、この手のサポート不足による失敗のおかげで、ある市場領域の強力な二番手競合製品でさえつぶれるのは、何度もみてきたはずだ（このパターンは、PC 用独占 OS やワープロ、会計ソフトやビジネスソフト一般の歴史を調べてみた人なら、だれでもはっきりわかるだろう）。工場モデルがもたらすまちがったインセンティブのおかげで、一人勝ち式の市場力学がつくられてしまい、結局は勝者の顧客も損をすることになる[4]。

工場モデルでなければ、どういうモデルがいいんだろう。ソフトのライフサイクルでも実際のコスト構造を効率的に（ここでの「効率的」というのは、ふつうの口語的な意味と経済学の専門用語での意味の両方をさす）扱うためには、サービス契約や購読など、ベンダーと顧客の間の価値交換が**継続的に**行われるような価格構造が必要だ。だから自由市場の効率性追求条件のもとでは、成熟したソフト産業が最終的に採用するのは、こういう価格構造だろうと予想できる。

いままでの議論で、オープンソースソフトがこれまでの体制秩序に対して、技術的だけでなく経済的な意味でも挑戦状をつきつけるものとなる理由について、だんだん見えてきたと思う。ソフトを「自由（フリー）／無料（フリー）」にすることで、どうやらソフトはぼくたちを、サービス料金が支配する世界に無理矢理押しやることになるようだ——そしてクローズド・ソースのビットの販売価値が、実はずっと、かなり貧弱な目くらましだったことを暴露することになるんだ。

「自由（フリー）／無料（フリー）」という用語は、別の意味で誤解を招きやすい。ある財のコストが下がると、それを支えるインフラに対する投資は、減るのではなくて増えることが多い。車の値段が下がると、自動車メカニックへの需要が上がる——それだから、いま販売価値によって対価を得ている 5%のプログラマも、オープンソースの世界で困ったりはしないことになる。この転換の過程で損をするのはプログラマじゃなくて、不適切なクローズド・ソース戦略に賭けてきた投資家たちなんだ。

4 「情報はフリーになりたがっている」というのはウソだ。

もう一つ神話があって、工場モデルの幻想と同じく誤解でありながら、中身はそれとは正反対で、オープンソースソフトの経済学についてのみんなの思考を混乱させている。それは「情報はフリーになりたがっている」というやつだ。これを整理していくと、ふつうはデジタル情報の複製にかかる限界費用はゼロだから、その市場価格もゼロになるべきだ、という主張に落ち着く。

この神話のいちばん広いものは、競合する財へのなんらかの利害にかかわる情報の価値を考えれば、すぐに論破できる——たとえば宝の地図や、スイスの銀行の口座番号、あるいはコンピュータのアカウントのパスワードなど。そこへの利害を持つ情報はコストゼロで複製できるけれど、そこで問題になっている利害はコストゼロで複製できない。すまり、その利害をもたらす物件の限界複製コストはゼロではなく、したがって、そのコストはそれへの権利を左右する情報にも踏襲されることができる。

この神話を持ち出してきた主な理由は、それがオープンソースの経済効用に関する議論と無関係

[4]訳注：ここの ESR の議論は理解に苦しむ部分がある。ここでの議論は実質的に、ソフトの普及度や人気を決めるのは各種サポートの有無だと言っているわけだけれど、どうしてそれが一社独占につながる必要があるんだ？

だということを示すためだ。後で見るように、オープンソースの経済効用に関する議論は、ソフトが実は工業製品のような（ゼロではない）価値構造を持っていると仮定したとしても、同じように成り立つ。だからぼくたちは、情報がフリーである「べき」かどうかという質問に取り組む必要もないわけだ。[5]

5　逆転した共有地

みんなが思っているモデルを疑問視したんだから、こんどはぼくたちが別のモデルをつくれるか見てみよう――オープンソースの協力体制を維持可能にする、ゴリゴリの経済学的な説明だ。

これはいろいろちがったレベルで検討しなければならない問題だ。一つのレベルとして、まずオープンソースプロジェクトに貢献する個人の行動を説明しなきゃならない。別のレベルではLinuxやApacheみたいなオープンソースプロジェクトでの協力を維持する経済的な力を理解しなきゃならない。

またもや、まずはこの問題理解の妨げとなる、えらく広まっている通俗モデルを打破するところから始める必要がある。協力的行動を説明しようというあらゆる試みには、ギャレット・ハーディンの「共有地の悲劇」が影を落とすことになるんだ。

ハーディンの有名な議論では、まずお百姓さんたちの村に共有の草地があるものと想像してみてほしい。お百姓さんたちは、そこで牛に草を食べさせる。でも、牛が草を食べるとその共有地の状態は悪くなる。草は引きちぎられて、泥の穴が残り、そこに草がまた生えるまでには時間がかかる。もし、草の食べ過ぎを防ぐような、放牧権を割り当てる方針が合意され（そして強制され！）なければ、お百姓さんたちは一人残らず、我先に急いでなるべくたくさんの牛をそこに放って、共有地がただの泥沼と化す前に最大の価値をそこから引きだそうとするはずだ。

協力行動について、多くの人の直感的なモデルはいまの例とだいたい同じだ。これは実は、オープンソースの経済問題についてあまりいい診断にはなっていない。オープンソースの問題はただ乗り問題（提供不足）であって、混雑した公共財（使いすぎ）ではないからだ。それでも、最初に出てくる反論のほとんどは、これが裏付けになっている。

共有地の悲劇モデルだと、予想される結果は3通りしかない。一つは泥沼。一つは、だれか脅しの使える主体が、村のために配分方針を強制する方法（共産主義的な解決法だ）。三番目は、村人たちが自分の守れる範囲を柵で囲って、維持可能な状態を保つことだ（知的所有権による解決）。

このモデルを反射的にオープンソースに適用するために、みんなオープンソースの寿命はすごく短いはずだと予想する。インターネット上では、プログラマの割く時間の配分を強制するはっきりした方法はないので、このモデルを使うと、共有地はどうしても細分化して崩壊するしかない。ソ

[5]訳注：ふつう、「情報はフリーになりたがっている」という場合のフリーは自由に公開されるべきだという意味のフリーで、無料という意味ではない。ここんとこのESRの議論は、ちょっと的外れだと思う。でもESRがそれを知らないはずはないんだが……と聞いたら、これに対してESRは「いやそんなことはない、ふつうの用例では無料という意味だ、おまえはフリーソフト勢のイデオロギーに洗脳されている」と返事してくれたんだけれど、でもでも、ぼくが聞くこの用語の用例というのは、むしろジャーナリスト勢からくるもののほうが多いんだけどな。まあいいや。山形の理解が特殊だったのかもしれない。

フトのいろんな部分部分がクローズドになって、共同のプールに還元される作業の量は急激に減ることになる。

ところが経験的にいって、実際のトレンドはその正反対なのは明らかだ。オープンソース開発の広がりと分量（これはたとえば、Metalab への登録や、freshmeat.net での一日あたりアナウンス量なんかで計れる）は着実に増えている。明らかに、「共有地の悲劇」が実際のできごとをとらえきっていない、重要な部分があるんだ。

答えの一部はもちろん、ソフトは使っても価値が減らないという事実からくる。それどころか、オープンソース・ソフトは広く使われることで、その価値を**高める**。利用者たちが自分自身のバグ修正や追加機能（コードパッチ）を追加してくれるからだ。この逆転共有地では、みんなが放牧すればするほど草が増えるわけだ。

答えのもう一つの部分としては、共通のソースコードのベースに対する小さなパッチの推定市場価値を回収するのはむずかしいという点があげられる。たとえばぼくが、悩ましいバグの修正パッチを書いたとしよう。そして多くの人が、そのパッチには金銭的な価値があることを認識したとする。その人たちから、ぼくはどうやってお金を集めればいいだろうか。伝統的な支払い方式では、オーバーヘッドが大きすぎて、こういう場合に適切となるような少額の支払いをするときには大きな問題になる。

でももっと本質的な点として、その価値は回収がむずかしいだけでなく、一般的にいってその**値段を決めることさえ**なかなかできない、ということが指摘できる。思考実験として、理論的にみて理想的な少額支払いシステムがインターネット上にできたとしよう——高セキュリティで、だれでも使えて、オーバーヘッドなし。さて、「Linux カーネルのいろんな修正」というパッチをあなたが書いたとする。これに対する要求価格はいくらにすればいいだろう。買い手候補は、そのパッチを見ないで、いくら支払うべきかどうすればわかるだろう。

ここに出てきたのは、F・A・ハイエクの「計算問題（calculation problem）」の戯画版みたいなものだ——この取引をスムーズに行うには、このパッチの機能的な価値を評価して、それに応じた値段をつけてくれると信頼できるような、超越的存在が必要になってしまう。

残念ながら、超越的存在はいまも昔もえらく品薄で、こんな場面にはお出ましいただけない。パッチ作者たるハック素留造くんの選択肢は 2 つしかない。そのパッチを抱え込んでおくか、あるいはフリーでプールに投げ出すか。前者だと、なにも得られない。後者でも、なにも得られないかもしれないけれど、でもほかの人たちに相互に与えあうよう奨励して、いずれハック素留造くんの問題を解決してくれるような贈与につながるかもしれない。二番目の選択は、一見愛他的だけれど、実はゲーム理論的な意味で、最適に利己的な行動になっている。

この手の協力を分析するときに、ただ乗り問題はほとんど起きないということを理解することが大切だ。オープンソースプロジェクトの複雑さとコミュニケーションのオーバーヘッドは、ほぼ完全に、参加している開発者の数の関数になる。ソースなんか見ないエンドユーザをいくら抱えていても、コストは実質的にゼロだ。プロジェクトのメーリングリストに流れてくるまぬけな質問の比率は高くなるかもしれない。でもこれは、FAQ（よくある質問一覧）をつくっておいて、それを明

らかに読んでいない質問者はあっさり無視することで、そこそこ簡単に対処できる（そしてこれはどちらもごくふつうに行われている）。

オープンソースソフトの真のただ乗り問題は、むしろパッチを提出するときの摩擦コストに比例したものとなる。この文化での評判ゲーム（『ノウアスフィアの開墾』を見てね）で失う物があまりない、潜在的な貢献者は、金銭的な見返りがなければこう思うかもしれない：「このパッチを送っても引き合わないよ、パッチをきれいにして、変更履歴のための要約も書いて、さらに FSF への移譲書類にサインしなきゃならないし……」　このために、あるプロジェクトの貢献者の数（そしてその次にくるものとして、プロジェクトの成功）は、各ユーザが貢献するにあたってくぐりぬける必要のある関門の数と、強く反比例している。こういう摩擦コストは、単に機械的なものではなく、政治的なものでもある。この 2 つを合わせると、縛りのゆるい、不定形の Linux 文化が、緊密に組織化されて中央集権化している BSD 類に比べて、何倍もの創造的なパワーを発揮しているのかを説明できそうだ。そして、Linux の台頭とともに、フリーソフト財団の相対的な重要性が退化したのもたぶんこのためだ。

これはこれで結構な話だ。でも、これはそのハック素留造くんがパッチを書いてから、さてそのパッチをどうしようかという事後的な説明だ。ぼくたちに必要な残り半分は、そもそもそのハック素留造くんがどうしそのパッチを書けたかという経済学的な説明だ。クローズドソースのソフトの作業をすれば、販売価値を得られたかもしれないのに、ハック素留造はそうしなかった。オープンソース開発が華開くようなニッチをつくりだすビジネスモデルというのは、どんなものだろう？

6　ソース非公開にする理由

オープンソースのビジネスモデルを分類する前に、まず排除することで一般的にどんなメリットがあるのかを考えるべきだろう。ソースを非公開にするとき、人はいったいなにを守ろうとしているんだろうか。

たとえばだれかをやとって、自分の会社用の特別な会計パッケージを書かせたとする。自分の目的のためには、ソースを非公開にしたって、追加のメリットはなにもない。非公開にしたいまともな理由としては、パッケージをほかの人に売りたいか、競合他社にそのパッケージを使わせたくないからだ。

すぐに出てくる答えは、ソフトの販売価値を守るため、というものだけれど、内部使用のために書かれる 95%のソフトについては、これはあてはまらない。するとほかに、クローズドにしてなにかいいことがあるんだろうか。

二番目の理由（競争優位を守る）は、ちょっと検討に値する。もしその会計パッケージをオープンソース化したとしよう。それが人気が出てきて、コミュニティからの改良というメリットが出てきた。さて、あなたの競争相手もそれを使い始めた。競合相手は開発コストを支払わずにメリットだけを享受して、あなたの商売を脅かしはじめる。さてこれは、オープンソース化に反対する議論になるだろうか。

なるかも――そしてならないかも。本当にたずねるべきなのは、開発の負担をオープンソース化で下げることによるメリットと、ただ乗りの競合からくる損害と、どっちが大きいかということだ。多くの人は、このトレードオフについてあまりきちんとした議論をしていない。それは、自分の開発コストをサンクコスト（埋没費用）として扱わないせいだ。ここでの想定では、その会計ソフトはどのみち必要だったわけだ。だからその開発コストを、オープンソース化するコストに含めるのはまちがっている。

ソースを非公開にする理由の中には、まるでまともとは思えないものもある。たとえば、ソースを非公開にすると、ビジネスシステムがクラッカーや侵入者に対して高セキュリティになるという幻想を抱いている人もいる。もしそうなら、すぐに暗号専門家と話をしてそういう思いこみを治療してもらったほうがいいよ。ほんとうにセキュリティに神経をとがらせているプロは、ソース非公開ソフトのセキュリティを信頼したりなんかしない。かれらはそれを、つらい経験を通じて学びとってきたんだ。セキュリティというのは信頼性の一部だ。アルゴリズムにしてもその実装にしても、十分なピアレビュー（同業者による検査）を経たものだけが、安全なものとして信頼できるんだ。

7　利用価値による開発費用手当

利用価値と販売価値を区別することでぼくたちが気がつくだいじな事実は、クローズドからオープンソースに移行したときに脅かされるのは販売価値だけだ、ということだ。利用価値はまったく影響を受けない。

もしソフト開発を動かしているのが、販売価値よりも利用価値のほうであるなら、そして（『伽藍とバザール』で論じたように）オープンソース開発がクローズドな開発よりも本当に有効で効率が高いなら、期待利用価値だけでオープンソース開発が維持されるような状況が見つかると考えていいだろう。

そして実際問題としても、オープンソースプロジェクト用のフルタイムの開発者の給料が、利用価値だけでまかなわれている重要なモデルが最低でも 2 種類ある。

7.1　Apache の例：コストシェアリング

たとえば、高ボリューム・高信頼性 Web サーバがビジネス上で必須となる会社で働いているとしよう。それは電子商取引のためかもしれないし、広告を売るような、とっても目に付くメディアアウトレットかもしれないし、ポータルサイトかもしれない。一日 24 時間、週 7 日動いてもらわないと困るし、スピードもほしいし、カスタマイズも必要だ。どうやってこれを実現しようか。使える基本的な戦略は 3 つある。

独占 Web サーバを買う。この場合には、ベンダーの目標が自分のとマッチしていて、そのベンダーがそれをきちんと実装するだけの技術能力を持っている、という賭をしていることになる。こ

れがどちらも正しかったとしても、製品はたぶん、カスタム化という面ではかなり劣るものになるだろう。改変は、ベンダが提供しようと思ったフックを通じてしかできない。だからこの独占 Web サーバという道は、あまり人気はない。

自前でつくる。自分で Web サーバをつくるのは、そんなすぐに無視すべき選択肢じゃない。Web サーバというのはさほど複雑じゃない。ブラウザに比べれば絶対に単純だ。そして特化すれば、無駄がなくて非常に高性能になれる。この道をとれば、自分の望むだけの機能とカスタム可能性をずばり得られる。ただし、そのかわり開発時間で支払わなくてはならない。さらに、あなたの会社は、あなたが退職したときに困ってしまうかもしれない。

Apache グループに参加する。Apache サーバは、インターネットで結ばれた Web マスターたちの集団がつくった。かれらは、同じものを並行してたくさん開発するよりも、一つのコードベースの改善のために力を集中するほうが賢いと気がついたわけだ。こうすることで、かれらは独自開発のメリットのほとんどと、超並列ピアレビューの強力なデバッグ効果を両方とも手に入れることができた。

Apache の選択肢のメリットはとても強力なものだ。どれほど強力かは、月刊の Netcraft 調査[6]から判断できる。これによると、Apache はほかのすべての独占 Web サーバに対し、その登場時点から着実に市場シェアをのばし続けている。1999 年 6 月時点で、Apache とその親類は市場シェア 61%——そして法的にはだれにも所有されていないし、宣伝もないし、サービス契約をする組織もまったくない。

Apache の話を一般化すると、ソフトのユーザがオープンソース開発に出資するモデルになる。ユーザとしては、そうすればそれ以外のどの方法よりも優れた製品を、もっと低いコストで手に入れることができるんだ。

7.2 Cisco の例：リスク分散

数年前に、Cisco（ネットワーク機器メーカ）のプログラマ二人が、Cisco の社内ネットワーク用に、分散型印刷スプールシステムを書くという仕事を与えられた。これはなかなかむずかしい。ユーザ A が、任意のプリンタ B（これは隣の部屋にあるかもしれないし、何千キロも離れているかもしれない）で印刷できるようサポートするという能力にくわえて、紙切れやトナー切れのときには、そのジョブをルートしなおして、目標近くの別のプリンタに出すようにしなきゃならない。さらにこのシステムは、紙切れなどの問題はプリンタ管理者に報告する必要があった。

二人は、標準の Unix 印刷スプールソフトに、賢い変更を加えて、さらにラッパーのスクリプトを書いてこれを実現した。ところがそこで、自分たちも Cisco も困ったことになったのに、はたと気がついた。

問題は、二人ともいつまでも Cisco にいるとは考えにくいということだった。いずれ、二人とも Cisco を離れ、ソフトはメンテナンスされなくなって、腐り始める（つまり、現実世界の状況とあ

[6] http://www.netcraft.com/survey/

わなくなってくる）。自分の作品をこういう目にあわせたがる開発者はいないし、豪気な二人は、Cisco が開発費を支払ったのは、自分たちの就職期間以上に使いものになる解決策がほしいという、もっとも至極な期待に基づいてのことだと感じていた。

そういうわけで、二人は上司のところにいって、印刷スプーラソフトをオープンソースとしてリリースを認めるように、説得をした。かれらの議論は、Cisco は別に販売価値を失うわけじゃないし、得るものはほかにいろいろ多いよ、というものだった。いろいろな企業にまたがるユーザや共同開発者によるコミュニティの成長を助けることで Cisco は実際には、ソフトのもとの開発者がいなくなることに対し、実質的にヘッジをかけたことになる。

8 販売価値の困るところ

オープンソースだと、ソフトから直接の販売価値を捕捉するのはちょっとむずかしい。そのむずかしさは、技術的なものではない。ソースコードは、バイナリよりコピーしにくいわけじゃないし、販売価値の捕捉を可能にする著作権法やライセンス法の施行についても、オープンソースがクローズドなソフトより必ずしもむずかしいわけじゃない。

なにがむずかしいかといえば、オープンソース開発をささえている社会的な契約のほうだ。主要なオープンソース・ライセンスは、直接の販売収入獲得に役立つような、利用や再配布や改変に関する制限のほとんどを禁止しているけれど、その理由は 3 つあって、しかもそれが相互に強化しあっている。この理由を理解するには、こうしたライセンスが生まれてきた社会的な文脈、つまりインターネットハッカー文化[7]を見てやらなきゃならない。

ハッカー文化について、その外部でいまだに（1999 年の時点で）広く信じられている神話とはうらはらに、こうした理由はどれ一つとして、市場への敵意なんかとは関係ない。ハッカーの中でも、収益という動機づけに対して敵意を持つ少数派はまだいるけれど、コミュニティは一般に、Red Hat や S.u.S.E. や Caldera など営利目的の Linux パッケージ屋さんに積極的に協力しているし、自分たちの目的にあえば、企業世界とも喜んで協力することがわかる。ハッカーたちが売り上げを直接捕捉するようなライセンスに顔をしかめるのは、もっと深くておもしろい理由からだ。

一つの理由は対称性に関係したものだ。ほとんどのオープンソース開発者は、他人が自分の贈り物で儲けても、それだけでは反対しないけれど、でも一方で、だれ一人としてそこからもうけを得る特権的な立場にはならないことを要求する（ただしそのコードをもともと書いた人は例外かもしれない）。ハック素留造くんは、ホゲホゲ社が自分のソフトやパッチを売っても儲けても気にしないけれど、でもそれは、ハック素留造くん自身も潜在的にそれができる場合に限られる。

別の理由は、意図せざる結果と関係がある。ハッカーたちは、「商業」利用や販売についての制限や手数料を含むライセンス（直接の販売価値を回収しようといういちばんよくある試みで、一見するとそんなに無体なものには見えない）が、かなり深刻にこわい影響を持つのを見てきているん

[7] How to Become a Hacker http://www.tuxedo.org/ esr/faqs/hacker-howto.html
ハッカーになろう http://www.post1.com/ hiyori13/freeware/hacker.html

だ。具体的には、安いCD-ROMアンソロジーに入れて再配布したりしようとしたとき、法的な問題が出るかもしれない。もっと一般的には、利用・販売・改変・配布についての制限（およびその他ライセンスをややこしくするもの）は、その遵守状況を調べるためのオーバーヘッドがかかるし、（みんなの使うパッケージの数が増えるにつれて）組み合わせによって生じる、見た目の不確実性と法的リスクの可能性も爆発的に増える。こういう結果は有害だとみなされ、したがって、ライセンスを単純で制限なしにしろという強い社会的な圧力が生じる。

最後の、そしていちばんかんじんな理由は、ピアレビューだ。『ノウアスフィアの開墾』で論じた、贈与文化の力学。知的所有権を守ったり、直接の販売価値を捕捉するようにつくられたライセンス制約は、プロジェクトを分岐（フォーク）させるのを法的に不可能にしてしまうという副作用を持つ（これはたとえば、JiniやJavaの「コミュニティソース」ライセンスにもあてはまる）。プロジェクトの分岐（フォーキング）はあまりいい顔をされないし、最後の手段と思われているけれど（その理由は『ノウアスフィアの開墾』でたっぷり説明した）、管理者が無能だったり、寝返ったり（たとえばクローズドなライセンスに移行したり）する場合にそなえて、その最後の手段が残されていることは、とてもだいじだと思われているんだ。

ハッカーコミュニティは、対称性という点では多少の妥協はする。NetscapeのNPLみたいに、コードの大本の作者に収益上の特権を与えるようなライセンスでもがまんはする（NPLに限って言えば、オープンソースのMozillaコードを、クローズドソースなものも含めて派生的な製品に使う独占権を与えている）。意図せざる帰結のほうとなると、それほど妥協はしないし、分岐（フォーク）するオプションの保全についてはまったく譲らない（だからこそ、SunのJavaとJiniの「コミュニティライセンス」方式は、ほとんどハッカーコミュニティに受け入れられていない）。

こうした理由をみれば、Open Source Definitionの条項も説明できる。これは、よく使われるライセンス（GPL、BSDライセンス、XのMITライセンス、PerlのArtisticライセンス）のだいじな特徴について、ハッカーコミュニティが抱いている合意を明示するために書かれている。こうした条項は（そういうつもりではないのだけれど）直接の販売価値の実現をとてもむずかしくしてしまう。

9　間接販売価値モデル

さはさりながら、間接的な販売価値みたいなものを捕捉できるような、ソフト関係サービスの市場をつくる手はある。この種のモデルとしては、すでに存在するものが5つ、アイデアレベルのものが2つある（将来はもっと出てくるかもしれない）。

9.1　ロスリーダー・市場ポジション確保

このモデルでは、オープンソース・ソフトで独占ソフトのポジションをつくりだし、その独占ソフトから直接の収入を得つづける。いちばんよくあるたぐいとしては、オープンソースのクライア

ントソフトをばらまいて、サーバソフトが売れるようにするとか、ポータルサイトと関連した購読・宣伝収入を得るとかいう例がある。

Netscape Communications, Inc. は、1998 年はじめに Mozilla ブラウザをオープンソース化したときにこの戦略を追求していた。かれらのブラウザ側での商売は、売り上げの 13%で、マイクロソフトが初めてインターネット・エクスプローラを出荷したときには、すでにこの比率は下がりつづけていた。IE のすさまじいマーケティング（そして後に反トラスト法訴訟の争点となる、いかがわしいバンドル慣行）によって、Netscape のブラウザの市場シェアはすぐにボロボロになり、マイクロソフトがブラウザ市場を独占するのではないか、そしてそのデファクトの力をもって、HTML を左右して、Netscape をサーバ市場からも駆逐するのではないか、という不安が出てきた。

まだ非常に人気のある Netscape ブラウザをオープンソース化することで、Netscape は実質的に、マイクロソフトによるブラウザ独占の可能性を否定したことになる。かれらは、オープンソースの協力体制がブラウザの開発とデバッグを加速するものと期待し、マイクロソフトの IE が後塵を拝し続けるようにして、かれらが独占的に HTML を決めてしまうのを防いでほしいと願っていたわけだ。

この戦略はうまくいった。1998 年 11 月、Netscape 社は IE からビジネス市場シェアを奪回しはじめたんだ。Netscape が AOL に 1999 年はじめに買われたとき、Mozilla をそのままつづけさせる競争優位はかなりはっきりしていたので、AOL の最初の公約は、まだアルファ段階でしかない Mozilla プロジェクトのサポートをつづける、というものだった。

9.2 刺身のツマ

このモデルは、ハードウェアメーカ用だ（ここでのハードウェアというのは、Ethernet や周辺機器ボードから、コンピュータシステム丸ごとまでなんでも含まれる）。市場の圧力のせいでハード企業もソフトを書いてメンテナンスしなくてはならない（下はデバイスドライバから、設定ツールを経て、上は全 OS にいたるまで）。でも、ソフト自身は収益を産まない。これはオーバーヘッドだ——しかも、しばしばそれがかなりの負担になる。

この状況では、オープンソース化はだれでも思いつく。失う収入はゼロだから、なにも損はしない。ベンダーとして手に入れるものは、劇的に大きくなった開発者プールと、顧客ニーズへの高速で柔軟な対応、そしてピアレビューを通じた信頼性向上だ。たぶん顧客のロイヤルティも高まるだろう。顧客の技術スタッフたちが、そのコードにますます時間をさいて、必要なカスタマイズを行えば行うほどそうなるはずだ。

ハードウェアのドライバをオープンソース化することについては、ベンダーからよく出てくる反論がいくつかある。それをここでの一般的な議論とごちゃまぜにするよりこの問題だけをまとめた「補遺」を書いておいた。

オープンソースの「将来に不安がない」効果が特に強いのは、この「刺身のツマ」モデルの場合だ。ハードウェア製品は有限の製造期間とサポート期間しかない。それ以降になると、顧客は自

分で何とかするしかない。でもドライバのソースが手に入って、それを必要に応じてパッチできたら、同じ会社の満足したリピート顧客になる可能性が高まる。

この「刺身のツマ」モデルを採用した非常に劇的な例は、1999 年 3 月半ばに Apple Computer が、MacOS X の核である「Darwin」をオープンソース化すると決断したことだ。

9.3 レシピをまいて、レストランを開け

このモデルでは、あるソフトウェアを使って、クローズドなソフトのポジションを確立するのではなく（これだと、「ロスリーダー・市場ポジション確保」ケースになる）、サービスのポジションを確立する。

（むかしはこれを「カミソリを配って、カミソリの刃を売れ」と読んでいたけれど、でも製品とサービスの結びつきは、カミソリとカミソリの刃のアナロジーほどは緊密なものじゃない。）

Red Hat をはじめとする Linux ディストリビュータがやっているのは、こういうことだ。かれらが実際に売っているのは、ビット自体という意味でのソフトではない。ちゃんとうごく OS を組み立てて試験することによる付加価値なんだ。さらに、その OS は売り物になって、同じブランドのほかの OS と互換性があるという保証が（明示的にではなくても）ある。かれらの価値提案のほかの面としては、無料インストールサポートや、サポート契約継続オプションの提示なんかがある。

オープンソースの市場構築効果は、そもそも最初からサービス的な立場にいるような企業にとっては特に強力なものとなる。最近の、とても示唆的な事例が Digital Creations だ。ここは web サイトのデザイン会社で、1998 年に創業し、複雑なデータベースや取り引きサイトを得意としている。かれらの主要ツール、この企業の知的所有権の至宝は、オブジェクト出版ソフトで、いくつか名前を変えて装いも変わってきたけれど、いまは Zope と呼ばれているソフトだ。

Digital Creations の連中がベンチャーキャピタルを探しにいったとき、かれらがつかまえてきた VC は、この予想市場ニッチと、その人々、そしてそのツールを慎重に評価した。そしてかれは、Digital Creations に対し、Zope をオープンソースにしなさいと提言したんだ。

伝統的なソフト産業の基準からすると、これは完全に発狂した動きにしか見えない。従来のビジネススクール型の知恵だと、Zope みたいな中核知的所有権は、その会社の至宝であって、なにがあろうとくれてやるようなものじゃない。でもこの VC は、2 つのからみあった洞察をもっていた。一つは、Zope の真の核となる資産は、実はその人々の脳と技能だ、ということ。そして、Zope は秘密兵器としてよりも、市場作成ツールとしてのほうが、高い価値を生み出しそうだということ。

これを理解するには、2 つのシナリオを比べてみよう。伝統的なシナリオでは、Zope は Digital Creation の秘密兵器のままだ。ここでは仮に、それがとても強力なものだとしよう。結果として、この企業は優れた品質のものを短期間で提供する能力を持っている——が、だれもそれを知らないわけだ。顧客を満足させるのは簡単だけれど、そもそもの顧客層をつくりだすのが一苦労だ。

ところが VC は、Zope をオープンソースにすることで、Digital Creation の真の資産——その人々——についてのすばらしい宣伝になると見て取ったわけだ。Zope を評価した顧客は、インハ

ウスで Zope についてのノウハウを開発するよりも、同社の専門家を雇ったほうがいいと思うだろうというのがかれの予想だった。

Zope の重役の一人は、それ以来公然と、オープンソース戦略が「ほかでは考えられなかったような門戸も開いてくれた」と語っている。潜在的な顧客は、確かにこの状況の論理に応じた行動をとった——そして Digital Creations も、それに応じて繁栄している。

もう一つ、ホヤホヤの例が e-smith, inc だ[8]。この会社は、Linux をカスタマイズした、オープンソースのターンキー・インターネットサーバソフトのサポート契約を売っている。その重役の一人は、e-smith のソフトの無料ダウンロードが増えていることに触れて、こう語っている：「ほとんどの会社は、これをソフトの海賊コピーだと考えるでしょう、ウチは無料マーケティングだと考えているんです[9]」

9.4 アクセサリー

このモデルでは、オープンソース・ソフトのアクセサリーを売ることになる。いちばん低いところでは、コーヒーカップや T シャツ。ハイエンドでは、質の高い編集と製本でのドキュメンテーション。

アクセサリー会社としては、オープンソース関連ソフトのすぐれた参考書をたくさん出している出版社 O'Reilly Associates がいい例だ。オライリーは、有名なオープンソース・ハッカー（たとえば Perl のラリー・ウォールやブライアン・ベーレンドルフ）を実際に雇っている。これは、選んだ市場での評判を高めるためだ。

9.5 未来をフリーに、現在を売れ

このモデルでは、クローズドなライセンスでバイナリとソースをリリースするけれど、そのライセンスに期限をつける。たとえば、フリーな再配布は認め、商業利用は有料にして、リリースの 1 年後か、あるいはベンダーが倒産・廃業した場合には GPL になるようなライセンスを書けばいい。

このモデルでは、顧客はその製品がニーズにあわせてカスタマイズできるのが確実にわかる。ソースが手に入るからだ。その製品は、将来の不安がない——ライセンスは、最初の会社がつぶれても、オープンソースコミュニティがそれを引き継げると保証しているから。

販売価格と販売量は、こういう顧客の期待に基づいているので、最初の企業としては、完全なクローズドソースのライセンスでリリースするときよりも高い収入を得られるはずだ。さらに古いコードが GPL 化されるから、念入りなピアレビューも受けるし、もとの作者としては 75%のメンテナンスコストも、ある程度節約できることになる。

このモデルは、Aladdin Enterprises がうまいこと採用している。かれらは有名な Ghostscript を開発している（Postscript のインタープリタで、それを各種プリンタの独自言語に変換する）。

[8] http://www.e-smith.net/
[9] http://www.globetechnology.com/gam/News/19990625/BAND.html

このモデルの最大の欠点は、製品サイクル初期にはライセンスがクローズドなので、ピアレビューが受けられないという点だ。ピアレビューが必要になるのは、まさにその開発初期なのに。

9.6 ソフトをフリーに、ブランドを売れ

これは実例のないビジネスモデルだ。ソフト技術をオープンソース化して、テストスイートや、互換性標準を持っておく。そしてユーザに対し、その技術の実装が、同ブランドをつけた他製品すべてと互換性があるという証明ブランドを発行する。

（Sun Microsystems は、Java や Jini をこういうふうに扱えばいいのに）

9.7 ソフトをフリーに、コンテンツを売れ

これもまた、まだ実例のないビジネスモデルだ。株価のリアルタイム表示サービスのようなものを考えて欲しい。価値はクライアントにもサーバにもない。客観的に信頼できる情報を提供することにある。だからソフトはすべてオープンソース化して、コンテンツの購読を売ろう。ハッカーたちがクライアントを新しいプラットホームに移植して、それをいろいろ保守拡張してくれるにつれて、市場は自動的に拡大する。

（だからこそ AOL はアクセス用のクライアントソフトをオープンソース化すべきなんだ）

10 オープンにするとき、クローズドにするとき

オープンソースのソフト開発をサポートするビジネスモデルを見てきたので、オープンにするのと、クローズドにするのとが、それぞれどういうときに経済的になりたつのかという、もっと一般的な質問に取り組めるようになった。まず、各戦略のメリットがなにかをはっきり理解しておくべきだろう。

10.1 メリットはなんだ？

ソース非公開のアプローチは、自分の秘密の部分からのレントを集められるようにしてくれる。一方で、それは完全に独立したピアレビューの可能性を閉ざしてしまう。オープンソースアプローチは、独立ピアレビューの条件を確立するけれど、秘密の部分からレントを集めることはできない。

秘密の部分を持つ見返りはよく理解されている。伝統的に、ソフトウェアのビジネスモデルは、これを中心として構築されてきている。ごく最近まで、独立ピアレビューからくるメリットは、あまり理解されていなかった。でも、Linux OS は、インターネットの中核ソフトやその他エンジニアリング領域の歴史から、ぼくたちが何年も前に学んでおくべきだった教訓を、改めて思い出させてくれる——つまり、オープンソースのピアレビューは、高信頼性と品質を実現するスケーラブルな方式としては、唯一のものだということだ。

だから競争市場では、高い信頼性と品質を求める顧客は、ソフト製作者がオープンソース化して、サービスや付加価値や隣接市場で収入を確保する方法を見つけると、ごほうびをあげるようになる。この現象こそが、Linux の驚異的な成功の裏にあるものだ。Linux は 1996 年にはないも同然だったのが、1998 年末にはビジネスサーバ市場の 17%以上を制し、2 年以内にその市場で圧倒的シェアを占めそうな動きを見せている（1999 年頭の IDC の予測では、Linux が 2003 年までは、ほかの OS すべてをあわせたよりも急成長をとげる）。

オープンソースの同じくらいだいじな見返りは、オープン規格を広めてそのまわりに市場をつくりあげる手段としての効用だ。インターネットの劇的な成長は、だれも TCP/IP を所有していないという事実に負うところが大きい。だれもインターネットの核になるプロトコルに対して、独占的な鍵を持っていないんだ。

TCP/IP と Linux の成功の背後にあるネットワーク効果は、かなりはっきりしているし、最終的には信頼と対称性の問題に還元される——共有インフラに参加しようとする主体としては、その仕組みがいちばん底までわかれば、合理的にそれを信頼できるし、さらに特定の主体がレントを引き出したり、支配権を行使したりする特権的な地位にあるものよりは、参加者すべてが対称の権利を持っているほうがうれしいはずだ。

でも、ソフト消費者にとって、対称性の問題がだいじになるためには、別にネットワーク効果がなくったってかまわない。もしそれなりの品質を持ったオープンソースの代替手段があるなら、クローズドソースなんかに依存して、サプライヤに牛耳られた独占状態に囲い込まれることを望む消費者なんかいない。この議論は、ソフトウェアが消費者のビジネスにとって重要性を増すにつれて、強力なものとなってくる——それが重要になればなるほど、それが部外者に左右されるのは大きな問題になるわけだ。

最後に、オープンソース・ソフトで信頼の問題と関係しただいじな消費者メリットとしては、将来の不安がない、という点がある。ソースが公開されていれば、消費者は、ベンダーが倒産したときにも多少は手のうちようがある。これは特に、「刺身のツマ」モデルの場合にだいじだ。ハードウェアのライフサイクルは短いけれど、その影響はもっと一般的で、つまりはオープンソース・ソフトの価値が高くなるということだからだ。

10.2 その絡み合いはどうなる？

秘密部分からのレントがオープンソースの利益より高ければ、クローズドソースのほうが経済的に引き合う。オープンソースの利益が秘密部分からのレントより高ければ、オープンソースのほうがいい。

これだけ見ると、大した洞察じゃない。これが大したものになるのは、オープンソースからの見返りが、秘密の部分からのレントよりも計測や予測しにくいということを認識したときだ——そして、その見返りは、大目に見積もられるよりは、ものすごく低めに見積もられることのほうが多い。実際問題として、主流ビジネス界は 1998 年初頭の Mozilla ソースリリース以来、その置かれ

た立場を考え直しはじめているけれど、それまではオープンソースの見返りは、ごく一般的にはゼロであると（まちがって）想定されていたんだ。

では、オープンソースの見返りはどうやって評価しようか。一般的なかたちで答えるのはむずかしいけれど、でもほかのいろんな予測問題に取り組むのと同じやりかたで手をつけてみよう。まず、オープンソースのアプローチが実際に成功したり失敗したりした観察事例からはじめる。それを一般化してモデルに仕立て、収益最大化を目指す投資家や企業にとって、オープンソースが差し引きで儲けをもたらすような条件について、少なくとも定性的な感触くらいは得られるようにしよう。それからデータにもどって、磨きをかけていこうか。

『伽藍とバザール』で提出された分析から見て、オープンソースのメリットが高いのは次のようなときだと予想できる。(a) 信頼性、安定性、スケーラビリティがとても重要な場合、そして (b) デザインや実装の正しさが、独立ピアレビュー以外の方法ではきちんと検証できない場合（この 2 番目の基準は、現実的にはちょっとでも複雑なソフトならほとんどの場合にあてはまる）。

消費者の合理的な願望としては、独占サプライヤに囲い込まれるのを避けたいと思う。だからソフトがその消費者にとっての役割を高めるにつれて、オープンソースに対する興味も高まることになる（そしてその結果、サブライヤ側がオープン化する競争市場価値も高まることになる）。したがって、ソフトがそのビジネスにとってクリティカルな資本財になるにつれて、別の基準 (c) がオープンソースへの圧力を高めることにある（たとえば、多くの企業の MIS 部門など）。

アプリケーションの領域となると、さっき見たように、オープンソース・インフラは信頼と対称性をつくりだし、それが長期的には、もっと顧客を引きつけて、クローズドソースのインフラを競争でうち破ることになる。そして、こうした急成長市場に小さな部分を持っておくほうが、クローズドで停滞した市場の大きな部分を持っているよりも得なことが多い。同じように、インフラソフトの場合には、オープンソースのように、いたるところに存在することを目指すほうが、長期的には、クローズドソースのように知的所有権からのレントで得られるメリットより高い見返りを得られる。

実は、潜在的な顧客が、ベンダー戦略の将来の帰結について論理的に考える能力を持ち、サプライヤによる独占を受け入れたがらないということは、もっと強い制約条件を意味している。すでにすさまじい市場支配力を持っていないなら、オープンソースの遍在性を選んでもいいし、クローズドソースからの直接収入方式を選んでもいい——でも、両方はできない。（この原理と似たものは、ほかの分野でも見られる。たとえば、電子部品の市場では、顧客はしばしば単一ソースからしか得られないデザインは拒否する。）この議論は、もっと否定的でない形でも表現できる。ネットワーク効果（正のネットワーク外部性）が強いときには、オープンソースが正しいことになることが多い。

この論理をまとめると、オープンソースがクローズドソースよりも大きなリターンを生み出すのは、それが (d) 共通のコンピュータ・通信インフラを確立するか可能にする場合である、ということになる。

最後に、ユニークなサービスや、非常に差別化されたサービスの提供者たちは、自分たちのやり

方が競合相手にまねされるのをおそれるインセンティブが高い。かなめのアルゴリズムや知識ベースがよく知られているようなサービスベンダーの場合は、それが低いだろう。つまりは、(e) カギとなる手法（あるいはそれと機能的に等価なもの）が一般的なエンジニアリング知識の一部であるときのほうが、オープンソースが有力になりやすい。

インターネットの中核ソフト Apache と、Linux の ANSI 標準 Unix API 実装は、この 5 つの基準すべての見事な見本になっている。こういう市場がオープンソースへと進化する道筋は、DECNet、XNS、IPX みたいなクローズドなプロトコルで帝国を築こうとする、15 年にわたる試みの失敗に続いてデータネットワーキングが TCP/IP に再統合されたのを見てもよくわかる。

一方では、オープンソースがほとんど意味をもたないのは、価値を生み出すソフト技術を独占的に保有していて（これで条件 (e) は立派に満たされる[10]）それが (a) 失敗や欠陥があまり問題にならなくて、(b) 独立ピアレビュー以外の方法で簡単に検証できて、(c) ビジネスの生死を左右するものではなく、(d) ネットワーク効果や遍在性によって、その価値が極端に増えたりしないような企業の場合だ。

この極端なケースの一例として、1999 年頭にぼくは、丸太から垂木をなるべくたくさんとりたい製材所向けの、のこぎりの入れ方パターンを計算するソフトを書いている会社から相談を受けた。「うちはオープンソースにすべきでしょうか？」ぼくは「ノー」と結論した。このソフトが満たす基準といえば、かろうじて (c) だけだ。でもいざとなったら、経験豊かなオペレータが自分でのこぎりの入れ方を手動で計算できるだろう。

だいじな点は、その製品なり技術なりがこういう物差し上でおかれた位置というのは、時間と共に変わることがある、ということ。これについては、次のケーススタディで見ていこう。

まとめると、オープンソースをすすめる差別化要因としては次のようなものがある。

(a) 信頼性、安定性、スケーラビリティがとても重要な場合。

(b) デザインや実装の正しさが、独立ピアレビュー以外の方法ではきちんと検証できない場合。

(c) そのソフトがその利用者のビジネス展開を決定的に左右するような場合。

(d) そのソフトが、共通のコンピュータ・通信インフラを確立するか可能にする場合。

(e) その核となるメソッド（あるいは機能的にそれと等価なもの）が、よく知られた工学的な知識の一部であるとき。

10.3 Doom: ケーススタディ

id Software のベストセラーゲーム Doom の歴史は、市場の圧力と製品の進歩によって、クローズド VS オープンソースのそれぞれのメリットが大きく変わってしまうことを示す例だ。

Doom が 1993 年末に初めてリリースされたときには、その一人称的リアルタイムのアニーメーションはまったくユニークなものだった（条件 (e) の対極、といおうか）。ソフト技術の見た目のインパクトも驚異的だったけれど、それを当時の非力な CPU でどう実現していたのか、何ヶ月に

[10]訳注：なぜ？独自の技術ならよく知られてないのではないの？

わたってだれも解明できなかった。この秘密の部分は、かなり大きなレントをとれるだけの価値を持っていた。さらに、オープンソース化からくる潜在的なメリットも小さかった。単独ゲームだったから、ソフトは (a) 欠陥があってもコストは我慢できるくらい小さかったし、(b) 検証するのもそんなにむずかしくないし、(c) どの消費者にとっても、ビジネスを左右するようなものではないし、(d) ネットワーク効果のメリットもない。だから Doom がクローズドソースなのは、経済的にも合理的だった。

でも、Doom をとりまく市場はじっとしてはいなかった。競合候補たちは、そのアニメ技術と張り合うものを開発してきたし、Duke Nukem みたいなほかの「一人称撃ちまくり」ゲームも出てきた。こういうゲームが Doom の市場シェアを喰いだしてくると、秘密の部分からとれるレントの価値も下がってきた。

一方で、シェアを拡大しようという試みは、新しい技術的な課題をもたらした——信頼性を高め、ゲームの機能を増し、ユーザベースを広げ、マルチプラットホーム対応にする。さらにマルチプレーヤによる「死闘」モードと、Doom ゲームサービスの登場で、この市場にもかなりのネットワーク効果が出てきた。このすべてはプログラマの労働時間をとるものだったけれど、id Software としてはそれを次のゲームに割きたかったわけだ。

このトレンドすべてが、オープンソースにするメリットを増やす方向に動いていた。ある時点で、このメリット曲線が逆転して、id が Doom のソースを公開し、ゲームシナリオのアンソロジーみたいな二次市場でお金を儲けるようにシフトするほうが、経済的に合理性が高くなった。そしてこの時点からしばらくして、それが実現した。Doom のソースコードすべてが 1997 年末に公開された。

10.4 いつ手放すべきかを知る

Doom がケーススタディとしておもしろいのは、OS でもなければ通信・ネットワークソフトでもないからだ。したがってこれは、ふつうのよくあるオープンソースの成功とはかけ離れている。実は Doom のライフサイクルは、逆転ポイントもちゃんとあって、今日のコード生態におけるアプリケーションソフトの典型になりつつあるといえる——つまり、通信と分散計算が、深刻な堅牢性・信頼性・スケーラビリティの問題をうみだし、それはピアレビューによってしか対処できないわけだ。そして技術環境の境界もしょっちゅうまたがり、競合する役者の間の境界にもまたがるものとなる（そしてそこから出る信頼や対称性の問題もすべて生じる）。

Doom は単独ゲームから対戦型の死闘ゲームへと発展した。ますますネットワーク効果こそがコンピュータ**そのもの**になりつつあるんだ。同じトレンドは、ERP などのいちばん重たいビジネスアプリケーションにまで見られるようになってきている。企業がサプライヤや顧客とますます密接にネットワーク化されるようになっているからだ——そしてもちろん、これは World Wide Web のアーキテクチャ全体に暗示されていることでもある。ここから導かれる結論として、ほとんどあらゆるところで、オープンソースの見返りは着実に増えつつある、ということになる。

もしいまのトレンドが続くなら、21 世紀でのソフト技術と製品管理の重要な課題は、いつソフトを手放すか、ということになる――つまり、クローズドなコードをオープンソース・インフラのほうに渡して、ピアレビュー効果を活用し、サービスなどの二次市場での高い収益性を獲得するのをいつにするのか、という問題だ。

収益的なインセンティブから見て、この逆転ポイントははずしたくないだろう。早すぎても遅すぎてもいけない。それ以上に、手を打つのが遅すぎると、機会リスクが深刻になってくる――同じ市場ニッチでオープンソース化する競争相手に足をすくわれる可能性があるからだ。

これが深刻な問題なのは、その製品カテゴリーについても、ユーザのプールやオープンソースの協力へと引き込める才能のプールも、有限だからだ。そしていったんそこに引き込まれたら、それはかなり続く。もし二つの企業があって、だいたい同じくらいの機能を持つ競合コードをどっちもオープンソース化した場合、たぶん最初にオープンにしたほうが、最大のユーザと、最大最高の開発者を引きつけるだろう。2 番手は残り物に甘んじるしかない。そうやって引き込まれたら、ユーザもそれに慣れるし、開発者たちもコード自体に時間を投資するので、それは尾を引くことになる。

11　オープンソースのビジネス生態学

オープンソースコミュニティは、オープンソースの生産性効果を増幅するような形で、自らを組織してきた。特に Linux の世界では、複数の Linux ディストリビュータが競合していて、それが開発者とは別の階層を作りだしているということが、経済的にだいじな事実となっている。

開発者たちはコードを書いて、そのコードをインターネット上で公開する。各ディストリビュータは、出回っているコードの何らかのサブセットを選んで、それを統合してパッケージ化し、ブランド名をつけて、顧客に売りつける。利用者は、各種ディストリビューションからどれかを選び、さらにコードを開発者のサイトから直接ダウンロードして、そのディストリビューションを補うこともある。

こうして階層がわかれていることで、とても流動的な改善の内部市場がつくりだされるという効果がある。開発者たちは、ディストリビュータやユーザの関心を求めて、ソフトの品質によりお互いに競合する。ディストリビュータは、自分たちの選択方針と、ソフトに付加できる価値によって、ユーザの財布をめぐって競争する。

この内部市場構造の一次的な効果としては、ネットの中のノードがすべて交換可能だ、ということだ。開発者が脱落してもいい。かれらのコードベースが、直接だれかに拾われなくても、関心をめぐる競合のために、すぐに機能的な等価物ができる。ディストリビュータが脱落しても、共通のオープンソース・コードベースは影響を受けず、びくともしない。全体としての生態は、クローズドソース OS のどんな一枚岩ベンダーが実現するよりも、市場の要求にもっとはやく反応するし、ショックへの耐久力もずっと高く、自己再生能力も高い。

もう一つだいじな影響としては、専門特化による効率向上とオーバーヘッドの低下だ。開発者た

ちは、伝統的なクローズドのプロジェクトをしょっちゅうダメにして、泥沼化させてしまう圧力を味わう必要がない――営業部門からの無意味で気が散るチェックリストもないし、不適切で古くさい言語や開発環境を使えなんていう経営陣からの命令もないし、すでにあるものを、製品差別化だの知的所有権保護だのと称し、新しく互換性のない形で発明しなおす必要もないし、そして（いちばんかんじんなことだけれど）締め切りもない。きちんとできてもいない 1.0 版を出荷させられるなんてこともない――これは（デマルコとリスターが『ピープルウェア』で「できあがったら起こしてくれ」式マネジメントの議論で述べたように）高品質な結果をもたらすだけでなく、本当に動く結果をいちばん早く出荷する方式でもあるんだ。

一方のディストリビュータは、ディストリビュータがいちばん上手にできる部分に特化できる。単に競争力を保つためだけに、巨大で果てしないソフト開発に予算をつけるというニーズから解放されて、システムインテグレーションやパッケージ、品質保証やサービスなんかに専念できるようになるわけだ。

ディストリビュータも開発者も、オープンソース手法の不可分な一部であるユーザからの、たえまない監視とフィードバックによって、正直であるようにし向けられるわけだ。

12　成功に対処する

「共有地の悲劇」は、今日のようなかたちでのオープンソース開発には適用できないかもしれないけれど、だからといって、今日のようなオープンソースの勢いがこのまま続くかどうか、心配すべき理由がないということにはならない。もし見返りが大きくなれば、キーとなるプレーヤーたちは協力から寝返るのではないか？

この質問は、いくつかのレベルで答えることができる。ぼくたちの対抗物語である「共有地の喜劇」は、オープンソースの個々の貢献は金銭化するのがむずかしいという議論をもとにしていた。でもこの議論は、すでにオープンソースと結びついた収入を持っている企業（たとえば、Linux のディストリビュータ）の場合はかなり力を失う。かれらの貢献は、すでに日々金銭化されている。かれらのいまの協力的な役割は、安定しているんだろうか。

この質問を検討することで、現在の実世界でのオープンソースソフトの経済について、いくつかおもしろい洞察が導かれる。そして、真のサービス産業的パラダイムが、未来のソフト産業にとってどんな意味を持つかについても。

現実的なレベルでは、いま現在のオープンソースコミュニティに適用されたとき、この質問はふつうは二種類の形で提起される。その 1：Linux は分裂しないの？その 2：Linux には支配的で独占に近いプレーヤーが登場しないの？

Linux が分裂すると主張するときにみんなが頼る歴史的なアナロジーは、1980 年代の独占 Unix ベンダーたちの行動だ。オープン規格について果てしなく語っていたのに、連合やコンソーシアムや合意が山ほどあったのに、独占 Unix は崩壊した。ベンダーたちが、自社製品を差別化しようとして OS 機能を追加したり改変したりする欲望は、互換性を保つ（そして結果として独立ソフト開

発者の参入障壁を下げ、消費者の総保有コストを下げる）ことで Unix 市場の総サイズを拡大するという関心よりも強かったわけだ。

これは Linux ではほとんどありえないだろう。すべてのディストリビュータはオープンなソースコードという共通基盤をもとに運営するよう制約されているからだ。どこか一社が差別化を維持するのは、現実的には不可能だ。Linux コード開発が基づいているライセンス条項は、コードをみんなと共有することを実質的に強制しているからだ。あるディストリビュータが新しい機能を開発した瞬間に、どの競合相手もそれをコピーしていいということになる。

みんながこれを理解しているので、だれも独占 Unix を分裂させたような手練手管をしようなんて、考えもしない。かわりに、Linux のディストリビュータは、消費者と市場全体のメリットとなるような形で競争する立場に追い込まれる。つまり、サービスやサポート、そしてインストールや利用を本当にやりやすくするインターフェースは何かという、かれらの設計上の賭けで競い合うことになるわけだ。

共通のソースベースのため、独占の可能性は閉ざされている。Linux 界の人が独占の心配をするときには、「RedHat」という名前がつぶやかれることが多い。ディストリビュータの中で、最大でいちばん成功しているところだ（アメリカでは市場の 90% くらいを占めると推定される）。でも、1999 年 5 月に Red Hat が、みんなの待ち望んでいた 6.0 リリースを発表してからものの数日で——Red Hat 自身の CD-ROM がまるで出荷しないうちに——Red Hat 自身の公開 FTP サイトからつくったリリースの CD-ROM 版が、ある出版社やほかの CD-ROM 業者から発表されて、しかも値段は Red Hat の予定定価より低かった。

Red Hat 自身は、これについて顔色一つ変えなかった。というのも Red Hat 創業者たちは、自分たちが自社製品のビットを所有していないし、また所有できない、ということをとてもはっきり理解しているからだ。Linux コミュニティの社会規範が、そういう所有を禁止しているんだ。ジョン・ギルモアの有名な洞察として、インターネットは検閲をダメージと見なして、その迂回路をつくる、というものがあるけれど、それにちなんで言うなら、Linux を扱うハッカーコミュニティは、コントロールしようという試みをダメージと見なして、それを迂回する。最新製品をリリース前にクローンされたことについて、Red Hat が抗議を行ったりしたら、Red Hat は開発者コミュニティから将来協力をとりつける能力 を大幅に失うことになっただろう。

現在ではさらに重要になったことかもしれないけれど、こうしたコミュニティの規範を、強制力ある法的な形式で表明するソフトライセンスは、Red Hat 製品がベースとしているコードのソースを同社が独占してしまうことを、積極的に禁止している。Red Hat が売れる唯一のものは、ブランド・サービス・サポート関係で、しかもその相手は、自由意志でそれに対して支払いをしたがる人に限られる。こういう状況では、客を食い物にするような独占の可能性が大きく出てくるようなことはないだろう。

13　オープンR&Dとパトロン制の再発明

本物のお金がオープンソース界に注入されることで、変化が生じている面がもう一つある。ハッカーコミュニティのスターたちは、稼ぎは昼間に別の仕事で得てオープンソースは趣味、というのではなく、自分たちのやりたいことをやって、支払いを受けられるようになってきたんだ。Red Hat、O'Reilly Associates、VA Linux Systemsみたいな企業は、オープンソースの才能の基盤を雇い、維持するという目標をかかげた、いわば半独立の研究部門をつくりあげつつある。

これが経済的に成り立つのは、こういう研究所を維持するための一人あたりコストが、その企業の市場をもっと急速に拡大させることからくる期待収益によって、充分にまかなえる場合だけだ。O'Reillyが、PerlとApacheの主要著者が好きなことをやるのに金を出せるのは、かれらの努力のおかげで、PerlやApache関係の本がもっと売れるようになるからだ。VA Linux Systemsが研究部門を維持できるのは、Linuxを改善すると、同社の売っているワークステーションやサーバの利用価値が高まるからだ。そしてRed HatがRed Hat高等開発研究所を維持しているのは、自分たちの提供するLinuxの価値を高めて、もっと顧客を集めるためだ。

ソフト産業の中でも、もっと伝統的な部門からやってきて、特許や商売上の秘密に守られた知的所有権こそが企業の至宝だと考える文化に育った戦略家たちにとっては、この行動は（それが市場をのばす効果を持つにしても）まるで説明がつかないように思えるだろう。なぜ研究にお金をつけて、それを競合社たちがだれでも（その定義からして）自由にコストなしで利用できるようにするんだ？

これを左右する理由は2つあるようだ。一つは、こうした企業が自分の市場ニッチでの有力プレーヤーである限り、かれらはオープンR&Dからくるリターンについても最大シェアを獲得することが期待できる。R&Dを使って将来の利益を買うというのは、目新しくもなんともない。興味深いのは、その期待将来収益が十分に大きいので、こうした企業はただ乗り連中をあっさり容認できる、というこの選択で暗示されている計算のほうだ。

この明らかな期待将来価値分析は、ROIばかりに注目するガチガチ資本主義者の世界では不可欠なものだけれど、これは実は、スターを雇う説明としていちばんおもしろいものではない。当の企業たち自身が出してくる説明が、もっとモワッとしたものだからだ。きいてみるとかれらの答は、これは単に自分たちの出身コミュニティで正しいとされることをやっているだけなんだ、と語るだろう。不肖この著者めは、前出の3企業の重役たちとかなり深い知り合いなので、こうした主張がただのつつましやかなポーズとして無視していいものではないということは証言できる。それどころか、ぼく自身がVA Linux Systemsの役員になれと誘いを受けたんだけれど、その理由ははっきりと「なにが正しいことか」を助言してくれ、というものだったし、「なにが正しいか」をぼくが実際に説明したときも、いやがる様子なんかまったく見せなかった。

経済学者なら、ここでどんな見返りが出てきているのかをたずねてもいい。もし、正しいことをやるという言いぐさが、中身のないポーズではないということを受け入れたら、ぼくたちは次に、その企業にとって「正しいこと」がどんな自己利益をもたらすのかを追求すべきだ。そして答え

は、それだけで見れば、驚くべきものでもないし、質問さえ正しければ確認困難なものでもない。そしてほかの産業でわざとらしいほど愛他的な行動と見られるものと同じく、こうした企業たちは実は、自分たちが善意を買っていると思っているわけ。

善意を勝ち取るべくがんばって、その善意を、将来の市場利益を予測する資産として評価する、というのは、これまたまるで目新しいことなんかじゃない。おもしろいのは、こうした企業たちの行動から見て、かれらがその善意にとても高い評価を与えているということだ。かれらは明らかに、直接的な収入を産み出さないプロジェクトのために高価な才能を喜んで雇おうとする。企業のたちあがりから IPO までの、いちばん資金的に苦しい時期でさえ。そして少なくともいまのところ、市場はこうした行動に報いている。

これら企業の重役たち自身、善意がかれらにとって特にだいじな理由をはっきり認識している。かれらは、製品開発と、非公式なマーケティング部隊として、顧客のなかのボランティアたちにかなり頼っている。かれらの顧客層との関係は親密で、企業内外の個人同士の、個人的な信頼関係に依存することもよくある。

こうして見てくると、ぼくたちが別の理由づけにしたがって学んできた教訓が、ここでまた再確認されていることがわかる。Red Hat/VA/O'Reilly とその顧客・開発者との関係は、製造業企業の場合とはかなりちがう。それはむしろ、知識集約型のサービス産業に典型的な特徴を持っている。技術産業の外を見てみると、こうしたパターンは（たとえば）法律事務所や医者や大学なんかで見られる。

それどころか、オープンソース企業がスター級ハッカーを雇うのは、大学がスター学者を雇うのと同じ理由だと見ることもできる。どちらの場合にも、この慣行は仕組みといい効果といい、産業革命の後までほとんどの芸術を支えていた、貴族によるパトロン制とよく似ている——そしてその類似について、一部の人たちは十分に承知している。

14　目標到達までの道のり

オープンソース開発に予算を手当する（そして利益をあげる！）ための市場メカニズムは、まだ急速に発展しつつあるところだ。この論文でみてきたビジネスモデル以外にも、どんどん新しいモデルが発明されるだろう。ソフトウェア産業をクローズドな知的所有権ではなく、サービスのほうに焦点をあてて発明しなおしたときになにがどう変わるかについて、投資家たちはいまだに思案してる最中だし、そのプロセスはしばらく続くはずだ。

この概念上の革命は、ソフト産業の中で販売価値に依存している 5%に投資してきた人たちにとっては、将来の収益が失われるという意味でコストがかかる; 歴史的に見て、サービス産業は製造業ほどはもうからない（とはいえ、医者や弁護士にきけばわかることだけれど、実際にそれを商売にしている人たちへのもうけはもっと高いことが多い）。でも、失われてしまった分の利益よりも、コスト側でのメリットのほうがあらゆる場合に大きくなるはずだ。ソフト消費者はオープンソース製品からすさまじいコスト節減と能率向上を実現できるからだ。（ここで起きていることは、従来の

音声用電話網がインターネットに取って代わられて、あちこちで生じている影響とよく似ている。）

こうしたコスト節減と能率向上の見込みのおかげで、市場機会が生まれつつあって、それを活かそうとして起業家たちやベンチャーキャピタリストたちが群がってきている。この論文の初稿を書いているときに、シリコンバレーのいちばん高名なベンチャーキャピタル会社が、一日 24 時間週 7 日の Linux 技術サポートを専門とする初のスタートアップ企業に対して筆頭ポジションで出資をしている。1999 年末までに、いくつか Linux やオープンソース関連企業の IPO が公開されるものと一般に期待されている——そして、大成功するというのが一般の見通しだ。

もう一つ、とてもおもしろい展開として、オープンソース開発においてタスクごとの市場をつくろうという組織的な動きが始まっていることが挙げられる。SourceXchange[11] と CoSource [12] は、それぞれちょっとちがったかたちで、裏返しの競売市場をオープンソース開発の予算手当に適用しようとする試みだ。

全体的なトレンドははっきりしている。Linux が 2003 年までは、ほかの OS すべてを**あわせたよりも**急成長をとげるという IDC の予測についてはすでにふれたとおり。Apache は市場シェア 61%を持ち、しかも着実にそれをのばしている。インターネットの利用は爆発的に増えていて、インターネット OS カウンタのような調査では、Linux をはじめとするオープンソースの OS がすでにインターネットのホストの多数派になっていて、クローズドシステムに対して着実にシェアをのばしているのがわかる。オープンソースのインターネットインフラを活用しなくてはならないというニーズは、単に他のソフトのデザインだけでなく、この世のありとあらゆる企業のビジネス慣習のみならず、利用・購買パターンまで条件づけるようになってきている。このトレンドは、どう考えても加速することだけはまちがいない。

15　結論：革命のあとの人生

オープンソースへの転換が完全に終わったあとのソフトウェア界は、どんな様子になっているだろう。

この質問を考えるためには、そのソフトが提供するサービスがどこまでオープンな技術規格に基づいて表現できるかによって、ソフトの種類を仕訳すると役にたつ。これは、そのソフトのベースとなるサービスがどこまで共有物化しているかときれいに相関している。この軸は、みんなが「アプリケーション」（まったく共有物化されておらず、オープンな技術規格は弱いか、まるで存在しない）、「インフラ」（共有化されたサービス、強い標準規格）、そして「ミドルウェア」（一部は共有物になっていて、技術規格はあるけれど不完全）というときに、おおむね想定しているものとそこそこ対応している。1999 年現在でそれぞれの代表例としては、ワープロ（アプリケーション）、TCP/IP スタック（インフラ）、データベースエンジン（ミドルウェア）になるだろう。

さっきやった、見返り分析から考えると、インフラとアプリケーション、ミドルウェアは、それ

[11] http://www.sourcexchange.com/process.html
[12] http://www.cosource.com/

ぞれ別の形で変化をとげて、オープンソースとクローズドの均衡比率もちがってくるはずだ、ということになる。さらにさっきの分析の結果として、あるソフトウェア領域でオープンソースがどこまで広がるかは、そこでネットワーク効果がどこまで効いてくるか、失敗したときのコスト、そしてそのソフトがビジネスにとって必須の資本財かどうかにかかってくることも思い出そう。

こういう発見成果を、個別の製品にではなくて、ソフト市場のあるセグメント全体に適用すれば、多少の予測をしてみることもできる。こんな具合だ：

インフラ（インターネット、Web、OS、競合製品の間の境界を越えて行き来しなければならない、通信ソフトの低レベルの部分）はほとんどすべてオープンソースになるだろう。ユーザの連合や、営利目的のディストリビューションやサービス企業が、今日の Red Hat みたいな役割を果たしつつ、協力しながらそれを維持することになる。

一方のアプリケーションは、たぶんいちばん非公開のままでとどまりやすい。非公開アルゴリズムや技術の利用価値が十分に高い（そして信頼性の低さからくるコストが低くて、供給元の独占からくるリスクもがまんできる）場合はあるだろう。このとき、消費者たちは非公開ソフトに対してお金を払い続ける。これは、ネットワーク効果の小さい、スタンドアローンの垂直市場にあてはまり続ける可能性が高い。前にあげた製材所がこの一例だ。1999 年の人気分野としては、生物学的認証ソフトもそういうソフトの例としてあげられるだろう。

ミドルウェア（たとえばデータベース、開発ツール、あるいはアプリケーションのプロトコルスタックの、カスタム化したトップエンドなど）は、もっと両者が混ざってくるだろう。ミドルウェアのカテゴリーが非公開になるかオープンになるかは、どうも失敗コストによるみたいだ。それが高いと、市場からもっとオープンにしろという圧力がかかる。

でもこの構図を完成させるには、「アプリケーション」にしても「ミドルウェア」にしても、実は安定したカテゴリーではないことを考えるべきだ。前出の「いつ手放すべきかを知る」で見たように、個別のソフト技術は自然なライフサイクルを持っていて、合理的にクローズドな状態から合理的にオープンな状態へと推移する。同じ理屈が、もっと大きなくくりでもあてはまる。

標準化された技術が生まれてきて、サービスの一部が共有物化されるにつれて、アプリケーションはだんだんミドルウェアになってくる（たとえばデータベースは、SQL の登場でフロントエンドとエンジンが切り離されてから、ミドルウェアになった）。ミドルウェアのサービスが共有物化してくるにつれて、それはこんどはオープンソース・インフラになってくる——いま、OS の領域でこの転換がまさに起こりつつある。

オープンソースからの競争も含む未来には、あらゆるソフト技術の長期的な運命は、死滅するか、あるいはオープンインフラそのものの一部になるかのどっちかだと予想できる。これは、ソース非公開ソフトからレントを徴収し続けたい起業家たちにとっては、どう考えてもうれしい報せではないけれど、でもソフト産業が全体としては起業精神を**持ち続ける**ことを示唆するものではある。新しいニッチがたえずてっぺん（アプリケーション）で生まれ続け、そしてその製品カテゴリーがインフラ化するまでの間、非公開の IP（知的所有物）独占が限られた期間だけ存在できるわけだ。

そして最後に、いうまでもなく、この均衡はそのプロセスを動かしているソフト消費者にとって

は、とてもありがたいものだ。次から次へと高品質のソフトが永久に提供され続ける。製品がうち切られてしまったり、だれかの隅っこにロックインされてしまう心配もない。セリドウェンの魔法のおなべは、ここでついにメタファーとして弱すぎることになる——というのも、食べ物は消費したり腐ったりしてなくなってしまうけれど、ソフトは可能性としては永久に残るものだからだ。取引だろうと贈与だろうと**あらゆる**強制されない活動すべてを含む、いちばん広いリバータリアン的な意味での自由市場は、あらゆる人のために、いつまでも増え続けるソフトウェアの富を産みだし続けることができるんだ。

16 書誌と謝辞

[CatB] The Cathedral and the Bazaar

http://www.tuxedo.org/ esr/writings/cathedral-bazaar/

[HtN] Homesteading the Noosphere

http://www.tuxedo.org/ esr/writings/homesteading/

[DL] De Marco and Lister, Peopleware: Productive Projects and Teams (New York; Dorset House, 1987; ISBN 0-932633-05-6)

David D. Friedman との刺激的な議論数回によって、オープンソースの協力に関する「逆転した共有地」モデルの洗練に役立った。また、Marshall van Alstyne には、ライバル関係にある情報財の概念的な重要性を指摘してもらって感謝している。インディアナグループの Ray Ontko は優れた批評を提供してくれた。1999 年 6 月までのぼくの講演の聴衆たちからも教わるところが大きかった。心当たりのある人は、自分でわかるはず。

この論文は、リリースしてものの数日以内に電子メールのフィードバックで大いに改善されたというのは、これまたオープンソース・モデルの有効性を示す好例でもある。Lloyd Wood は、オープンソースに「将来の不安がない」ことの重要性を指摘してくれた。Doug Dante は、「未来をフリーに」というビジネスモデルについて思い出させてくれた。Adam Moorhouse からの質問で、排除することのメリットに関する議論が生まれた。Lionel Oliviera Gresse gave は、ビジネスモデルの一つについて、もっといい名前を提案してくれた。Stephen Turnbull は、ただ乗りの影響についての粗雑な扱いをボコボコにしてくれた。

17 補遺: なぜドライバをクローズドにするとベンダにとって損なのか

[13]歴史的に、(イーサネットカード、ディスクコントローラ、ビデオボードのような) 周辺機器のメーカはオープンにするのを渋ってきた。これは今では変わってきていて、Adaptec や Cyclades のようなメーカは自社ボードの仕様とドライバのソースコードを当たり前のように公開し始めてい

[13]補遺ⓒ田宮まや

る。でも一般的にはオープンにすることに対してまだ抵抗がある。この補遺では、そのような抵抗を長らえさせている、いくつかの経済的な誤解を晴すことを試みる。

ハードウェアのドライバをオープンソース化することに対して出てくる異議の一つは、それがハードの仕組みについて重要な情報を明らかにしてしまうので、それが競争相手にコピーされて、不公平な競争優位を与えてしまう、というものだ。製品サイクルが3年から5年だった昔なら、これは議論としてなりたっただろう。でも、いまでは競争相手のエンジニアたちが製品をコピーし、コピーした物を理解するのに費やす時間で、製品サイクルの大半はもう終わってしまっている。そしてその間に、その連中は自分の製品を革新したり、差別化したり**できなくなる**わけだ。盗用は**罠**で、むしろあなたとしては、競争相手がそれにはまってほしいと願うべきなんだ。

いずれにしても最近では、そのような詳細はそう長く隠し通せるものではない。ハードウェアのドライバはOSやアプリケーションとはちがって、小さくて、逆アセンブルが簡単で、クローンを作るのは簡単だ。ティーンエイジの初心者プログラマにだってできる——そして頻繁に行われていることだ。

LinuxやFreeBSDのプログラマで、新しいボード用のドライバを書く能力と動機を兼ね備えている人は、文字通り何千といる。これらの熱心なハッカーたちは、比較的単純なインターフェースと良く知られた標準規格とを持つ（ディスクコントローラやネットワークカードのような）多くの種類の装置に関して、君（ハードウェアベンダ）自身のショップと同じくらいの速さでドライバのプロトタイプを作ってしまうことがよくある。しかも、ドキュメント無しで、既存のドライバをディスアセンブルしたりもしないで、だ。

ビデオカードのような扱いにくいデバイスについても、逆アセンブラで武装した頭のいいプログラマを止める方法なんてほとんどない。そんなにお金はかからないし、法的な障壁も穴だらけだからだ。Linuxは国際的なプロジェクトだから、リバースエンジニアリングが合法な司法的管轄地域がいつだってどこかしらに存在している。

これらの主張が真である動かぬ証拠としては、Linuxカーネルでサポートされているデバイスのリストや Metalab[14] のようなサイトのドライバのセクションを良く見てみるといい。そして新しいものが追加される速度に注目だ。

つまり、何が言いたいか？ドライバを秘密にしておくのは、短期的には魅力的に見えるけど、おそらく長期的にはまずい戦略だ。（すでにオープンにしている他のベンダと競合している場合には確実にね。）でもどうしてもクローズドにしておく必要がある場合は、オンボード ROM にコードを焼き付けるんだ。そして、それへのインターフェースを公開する。あなた（ハードウェアベンダ）の市場を構築するために、また、あなたが重要な点において競争相手よりもより良く考え、より優れて革新できる能力に自信を持っているということを潜在的顧客に示すために、可能な限りオープンにしよう。

クローズドにし続けた場合、通常はオープンソースとクローズドソースの悪いところばかりがふりかかることになる。あなたの秘密は暴露され、フリーな開発支援は得られず、より頭の悪い競合

[14] http://metalab.unc.edu/pub/Linux/hardware/!INDEX.html

相手らがクローン製作で時間を浪費することもなくなる。最も重要なことは、早期採用を普及させる手段を君が取り逃してしまうことだ。巨大で影響力のある市場（インターネットのすべてとビジネスデータセンターの 17%以上を効果的に運用するサーバを管理している人々）は、君が以上のようなことを理解しなかったという理由で、君の会社をわかってなくて自己防衛過剰だと正しく見限るだろう。そして、ちゃんとわかっている他のだれかからボードを買うことになるんだ。

18 改訂履歴

This is *Revision* : 1.14.

ここに上がっていないバージョンは、細々した編集上や誤植訂正のアップデートだ。

20 May 1999, version 1.1 – 草稿。

18 Jun 1999, version 1.2 – 内部レビュー版。

24 Jun 1999, version 1.5 – 初の公開。

24 Jun 1999, version 1.6 – ちょっとしたアップデート。「ハッカー」の定義のところ。

24 Jun 1999, version 1.7 – ちょっとしたアップデート。オープンソースの条件 (e) を明確化。

24 Jun 1999, version 1.9 – 「将来の不安をなくす」「未来を自由に」モデルと、排除することのメリットについての部分を追加。

24 Jun 1999, version 1.10 – 「カミソリの刃」モデルをもっといい名前に改名。

25 Jun 1999, version 1.13 – Netscape の収入について、13%という数字を訂正。ただ乗りの影響をもっときちんと処理、クローズドなプロトコルの一覧を訂正。

25 Jun 1999, version 1.14 – e-smith, inc. を追加。

9 Jul 1999, version 1.15 – ハードウェアドライバに関する新しい補遺と、Rich Morin のおかげで競合のある財についてもっといい説明を追加。[15]

[15]訳注：補遺の部分の訳は、ChangeLog(http://www.changelog.net/) の田宮まや訳を提供していただいた。多少の標記統一以外はほぼそのまま。ありがとうございます。