MMRC-J-21

ソフトウェアの開発サイクルとその規定要因
―脱パッケージソフトとパッケージソフトの比較―

東洋大学経営学部　藤田 英樹

一橋大学イノベーション研究センター　生稲 史彦

2004 年 12 月

東京大学21世紀COE〔センター・オブ・エクセレンス〕
ものづくり経営研究センター

ソフトウェアの開発サイクルとその規定要因
―脱パッケージソフトとパッケージソフトの比較―

東洋大学経営学部　藤田 英樹

一橋大学イノベーション研究センター　生稲 史彦

2004 年 12 月

要約

フリーウェア、シェアウェアの開発事例を分析する。これらオンラインソフトの開発では、開発者とユーザがより緊密に直接的な結びつきを持ち、あたかもユーザが開発活動の一端を担っているような現象が観察された。この現象をユーザの組織化と呼ぶことにするが、パッケージング販売を最初から放棄し、流通経路をおもにインターネットに求める「脱パッケージ化」というオンラインソフトにおける意思決定が、単なる流通経路の選択にとどまらず、ユーザの組織化と開発への貢献を引き出し、また、ソフトウェアの完成度がパッケージソフトほどは求められないため、開発・流通のコストが比較的小さくなることが、開発スタイルに劇的な変化をもたらすことを明らかにする。このため、オンラインソフトではバージョンアップのペースとして観察される「開発サイクル」の回転を速めることができ、ソフトウェアの品質やパフォーマンスが一般的なパッケージソフトと比べて格段に速く向上していくのである。

キーワード

開発サイクル　脱パッケージソフト　パッケージソフト　ユーザの組織化　試行コスト

１．はじめに

Linux の大成功を受けて、そのライセンス形態、開発の進め方であるオープンソース・ソフトウェア(以下、OSS)に、社会的のみならず学問的にも注目が集まっている。*Research Policy* の von Krogh & von Hippel (2003)を中心とする特集号はその好例であろう。

一般に OSS の成功要因は「無償」「オープンソース」であるとされている。こうした特徴がインターネット上に開発者やユーザのコミュニティ形成を促し、ソフトウェア開発の成功と加速度的な普及を導くと考えられている。しかし、このような成功パターンを有するソフトウェアは、何も OSS だけに限らない。フリーウェア、シェアウェアと呼ばれる日本のソフトウェアでも、そのユーザのコミュニティがインターネット上に形成され、開発者の活動を支援するという状況が多数観察される。フリーウェア、シェアウェアは必ずしも無償のオープンソースではないが、それにもかかわらず多数のユーザに支持され、中にはビジネスとして成功しているソフトウェアすらも存在している。したがって、ソフトウェアの開発・配布活動にインターネットをうまく取り入れ、開発者やユーザのコミュニティ形成に成功することこそが、ソフトウェア開発の成功と結果としての普及の鍵になると考えられる。

少なくとも現代においては、OSS もフリーウェア、シェアウェアもインターネットを利用して開発が進められ、インターネット上で流通している。インターネットを活用した開発・配布形態を採用することができるのは、意識的であれ無意識的であれ、パッケージング販売という流通形態をあらかじめ放棄しているからであろう。そこで我々は、これらのソフトウェアを「脱パッケージソフト」(Non-Packaged Software)と呼ぶことにする。脱パッケージソフトでは、インターネットを介して開発者・ユーザが組織化され、その一つの組織内で開発が進められていると見なすことができる。その対極にあるのが、企業が商用目的で開発しているパッケージソフトであると考えられるので、パッケージソフトの事例とフリーウェア、シェアウェアの事例とを比較することで、両者の開発スタイルの特徴とその規定因について検討していくことにしよう。

事例研究の結果、インターネットを媒介した「ユーザの組織化」を前提とするソフトウェア開発は、実はパッケージング販売を放棄したことで初めて可能になるということがわかった。実は「脱パッケージ化」という意思決定は、単なる流通経路の選択にとどまらず、ソフトウェアに対する開発者とユーザの認識とも表裏一体の関係にある。パッケージソフトはあくまでも、ユーザが対価を支払うだけの価値のある完成された「製品」であるから、企業は相当の時間とコストをかけて開発やバージョンアップを行うことになる。他方、脱パッケー

ジソフトでは、ユーザ自身が入手、インストール、使用に責任をもって当たるものであるとされ、しかもインターネットを通じて適宜ソフトウェアを更新することが開発者、ユーザ共に可能であるため、ソフトウェアの完全性は必ずしも求められない。このため、一回一回の開発・バージョンアップに時間やコストをかける必然性はなく、一般的なパッケージソフトに比べて格段に速い「開発サイクル」で、ソフトウェアの品質やパフォーマンスが向上していくのである。

インターネットとブロードバンドが普及した現在、インターネットを通じてユーザを組織化し、脱パッケージソフトのような「関門の緩やかな相対的に速い開発サイクル」によってソフトウェアを開発できる可能性がある。したがって、ソフトウェア企業はパッケージ／脱パッケージの開発スタイルを組み合わせることで、より効果的にソフトウェア開発・ビジネスを展開していくことができるのではないだろうか。例えば、Red Hat 社をはじめるとする Linux ディストリビューションが実践しているような、ソフトウェア開発の中核的部分を脱パッケージソフトの開発スタイルに委ね、それにパッケージソフトの開発スタイルに則ってユーザビリティ(使い勝手)などの機能追加、不具合修正などを行ってユーザに提供するビジネスモデルなども可能であろう。また、ソフトウェア開発以外の領域でも、企業や製品開発活動を取り巻く技術的条件、ユーザの状況などに応じて、ユーザの組織化や、関門の緩やかな相対的に速い開発サイクルを取り入れることで、製品開発パフォーマンスの向上、顧客満足度の向上などを実現させる可能性も考えられる。

２．既存研究のサーベイ

2.1 ソフトウェア開発に関する既存研究

ソフトウェア開発に関する研究は工学系のソフトウェア・エンジニアリングとして始められた[1]。経営学におけるソフトウェア開発に関する研究は、1980 年代末から本格化した。Cusumano (1991)がその嚆矢であり、ハードウェアを対象にした製品開発論をベースにしてソフトウェア開発について考察した。この研究では、顧客(企業)の注文にしたがって大規模なシステムを構築するカスタマイズド・ソフトウェアの開発を対象としていた。だが、その後ダウン・サイジングが進展し、法人・個人を問わずにパーソナル・コンピュータ(パソコン)が急速に普及し、それに伴ってパソコン向けのパッケージ・ソフトウェアが主流となったこ

[1] コンピュータの誕生と発展過程、ソフトウェア開発研究の成立経緯の詳細については、立本 (2002)を参照のこと。また初期のソフトウェア・エンジニアリングの代表的な著作としては、Brooks (1975; 1995)を参照のこと。

とを受け、同分野のリーダー企業であるマイクロソフト社を研究対象とした Cusumano & Selby (1995)が発表される。さらには、1990 年代後半のインターネット利用の爆発的ともいえる普及とそれに影響されたコンピュータ産業、ソフトウェア産業の変化を踏まえて、ブラウザを中心としたインターネット用ソフトウェアを対象とした研究成果 Cusumano & Yoffie (1998)が発表される。

このうち、Cusumano & Selby (1995)では、Microsoft 社に関する事例記述を通じて、同社が何故ソフトウェア産業におけるリーダー企業になり得たのかを考察している。その中で同社のソフトウェア開発とその管理についても詳述しており、同期安定化(Sync and Stabilize)プロセスという特徴的な[2]開発プロセスが同社の競争力に寄与したと主張している。Cusumano らによれば、その開発プロセスは以下の 3 点によって特徴づけられる。すなわち、①目標設定や概要設計を大まかなものに留めたまま詳細設計とコーディングを行うこと、②開発対象を可能な限りモジュール化して詳細設計とコーディングをモジュール毎に同時並行的に行うこと、③頻繁に結合テストを行ってモジュール間の不具合を早めに発見・修正することである。

続く Cusumano & Yoffie (1998)では、Microsoft 社と Netscape 社の事例を調査し、特にそのブラウザ開発について論じている。それによれば、同期安定化プロセスあるいはその発展型は、インターネット用ソフトウェアの開発にも有用であり、特にそれがインターネットを利用した広範なβテストと結びついて、迅速かつ良質なソフトウェア開発に繋がっていると主張している。

なお、エンジニアリング的バックグラウンドをもつ Iansiti の一連の研究も、「フレキシブル製品開発システム」(Iansiti & MacCormack, 1997)、「技術統合」(Iansiti, 1998)などの概念を提示してインターネット用ソフトウェアの開発について分析を行い、結果として Cusumano らと同様の知見を示し、こうした主張を支持している。

これらの研究の対象は、企業が商用目的で開発したソフトウェアやシステムであった。ところが、1990 年代中盤からインターネットが爆発的に普及し、OSS などインターネットを全面的に利用したソフトウェア開発が成功すると、1990 年代末からはそうしたソフトウェア開発の事例が研究対象になっていく。

[2] ただし、同様のプロセスがアメリカのソフトウェア企業でも同時期に採用されていたことは Cusumano & Selby (1995)でも述べられている。だが、それが最も徹底的に行われ、マイクロソフト社の競争力に寄与しているというのが Cusumano らの主張である。

2.2 インターネットを活用したソフトウェア開発

OSS に関する研究は Raymond (1997)が一つの契機になったと思われる。Raymond (1997)では、Linux に関する考察と筆者自身による「fetchmail」というソフトウェア開発の経験に基づいて、ソースコードを公開したソフトウェア開発を進める際に効果的、効率的な開発のあり方について考察が行われている。Raymond によれば、Linux とそれを参考に Raymond 自身が行った fetchmail の開発では、従来のソースコードを公開したソフトウェア開発、すなわち、フリーソフトウェア開発よりも、効果的、効率的な開発が行えたという。そして、Linux などの開発において効率的かつ効果的に開発が行えたのは、早期の頻繁なリリース、大幅な情報のオープン化と開発者相互間の積極的な分業によるものであるとしている。こうした開発のあり方を、フリーソフトウェア開発の「伽藍方式」と対比して、「バザール方式」と呼ぶことを提唱している。

Raymond の研究は、Linux の開発に関する最初の考察に位置づけられ、その貢献は非常に大きい。しかしながら、同時に最初の研究ゆえの問題があることも事実である。具体的には、記述と分析が体系的とは言えず、特に事実の記述とそれに基づく考察が交錯している点、考察の結果が断片的な断章形式を取っている点、そして、Linux 開発、バザール方式の利点をバグ(ソフトウェアの欠陥)の発見と修正に主に見ている点などが挙げられる。

他方、Linux が生まれるに至った状況について UNIX との関係に着目して歴史的に記述、分析することで、Linux の成功要因、及び OSS の可能性と限界を考察した研究として、高橋・高松 (2002)が挙げられる。高橋・高松 (2002)は、Linux がその開発、普及において成功を収めた要因が、本当に無償の OSS であったからなのか、という疑問から出発している。この疑問に応えるため、Linux 以前のフリーソフトウェア開発活動と、Linux が参考にしたとされる UNIX の歴史を詳述している。考察の結果、Linux が成功を収めたのは、無償の OSS であったからではなく、UNIX のライセンス問題によって UNIX カーネル[3]に代わるカーネルが渇望されていた状況があったこと、それ以前のフリーソフトウェア開発活動によってカーネル以外の OS の構成モジュールが既に作られていたことにあると結論している。換言すれば、Linux の成功は、無償の OSS であったからではなく、その登場したタイミングと UNIX やフリーソフトウェア開発運動などの周辺事情が「奇跡的に」組み合わさったからであると主張している。こうした考察結果に基づいて、Raymond の提唱した伽藍方式とバザール方式の対比、後者の優位性にも疑問を呈し、また、バザール方式とされた開発方式はその開発スタイ

[3] カーネルとは OS の中核部分に当たるものであり、通常はそれに周辺機器を動作させるためのドライバやユーザから見えないところで動作するデーモン、システムをサポートするツールやユーザ・インターフェースなどが付属されて OS として実際に機能するようになる(高橋・高松 (2002)より抜粋)。

ル、開発者のモチベーションなどにおいて、マイクロソフト社のソフトウェア開発のあり方に近いとも主張している[4]。

OSS に関する最新の、多様な問題意識に基づく多様なアプローチの研究は、*Research Policy* の特集号にまとめられている。まず、von Krogh & von Hippel (2003)では、OSS の現状と歴史を概説すると共に、OSS の研究に際してどのような問題意識、研究アプローチがあり得るかを述べている。彼らによれば、OSS を対象とした研究の問題意識として、①開発者がどのような理由、動機づけによって OSS 開発に携わるのか(motivation)、②OSS ではどのように開発が進められているのか(innovation process)、③OSS の成果物であるソフトウェアやシステムが、既存の、主に営利企業によって開発されたソフトウェアやシステムとどのように競争し、普及していくのか(competitive dynamics)が主な問題意識としてあり得るとしている。

この中で本研究と最も関連が深いのが、OSS においてどのように開発が進められているのかについて研究した von Krogh, Spaeth & Lakhani (2003)である。von Krogh et al. (2003)は、OSS 開発に開発者が参加するには、一定のコストが掛かるであろうと想定し、開発者コミュニティの形成、開発者の参加、及びそれらを左右する要因についての仮説導出を試みている。彼らがその分析対象とし、仮説導出の源泉としたのは Freenet の事例[5]である。彼らは開発者のインタビュー、1 年間分の電子メール・ログの分析、1 年間分のソースコードの分析、2 次資料の参照などを行って、Freenet の開発プロジェクトの非常に詳細な事例分析を行っている。具体的には、2000 年の 1 年間の Freenet の開発において、どのように個々の開発者が開発に参加し、その結果どのような開発者コミュニティの形成が行われたのかを検討している。その上で彼らは、開発者の参加、開発コミュニティの形成に関する 5 つの仮説の導出をしている。それは①一定のレベルと内容の活動(joining script)を行った方が開発コミュニティに参加しやすい、②ソフトウェア・アーキテクチャ自体が変化しうる開発プロジェクトでは、その構成要素であるモジュール毎に開発コミュニティへの参加の容易さ／難しさ(contribution barriers)が異なる、③新規参加者独自のソースコード(feature gift)をもって開発コミュニティに参加するか否かが専門化の程度を左右する、④新規参加者独自のソースコード(feature gift)はその新規参加者のそれ以前の知識(domain knowledge)やユーザとしての経験(user experience)から生じる、⑤新規参加者独自のソースコード(feature gift)は開発コミュニテ

[4] この研究でも開発スタイルという言葉を使い、開発のプロセスについて若干の記述、分析が行われている。特に、早期で頻繁なリリース、開発者のモチベーションなどに関する記述とその重要性の認識は本研究の知見とも合致するものである。しかしながら、Raymond (1997)同様、これらの記述が具体的ではなく、それが研究の中心的問題意識になっていない点は、本研究と異なると言える。

[5] 開発の動機等においてかなりの違いはあるものの、匿名性の高い、サーバに依存しないファイル共有ソフト(Peer to Peer ソフト)として、日本には Winny がある。

ィへの参加の容易さ／難しさ(contribution barriers)を変化させうる、というものである。その上で、これらの仮説を他の事例などでも検証し、より包括的でバランスの取れた OSS に関する理論の構築を行うよう、主張している。

von Krogh, et al. (2003)は、単一事例に基づく研究であることを考慮しても、開発コミュニティとその形成に関する多くの示唆を含んだ研究であるといえる[6]。しかしながら、その問題意識が、OSS 開発における開発者の参加、開発者コミュニティの形成に絞り込まれたが故に、限界もまた有している。例えば、彼らの研究は「あらゆる人が自由に OSS に関与することができるという事実が、商用ソフトウェアの開発モデルと大いに異なる OSS 開発の慣行を創り出した(the fact that open source software is freely accessible to all has created some typical open source software development practices that differ greatly from commercial software development models)」という von Krogh & von Hippel (2003, p.1151)の指摘に十分に答えているとは言い難い。なぜなら、彼ら自身も認めているように、彼らは OSS の事例である Freenet の事例の記述、分析に注力しており、商用ソフトウェアとの比較は行っていないからである。加えて、彼らの研究は、開発者コミュニティ（開発組織）の形成過程とそれに影響を及ぼす要因についての仮説導出を主たる目的としている。言い換えれば、彼らの研究では、ソフトウェアが開発され、機能が向上していくプロセスについては分析を行っていない。言うまでもなく、イノベーション、開発活動を分析するに当たっては、それを実行する組織のみならず、プロセスについて分析することが必要であり、OSS を含めたソフトウェアにおいても、その開発プロセス、あるいは開発モデル、開発スタイルに関しての分析が必要とされるであろう。この意味で本研究は、OSS のみを対象にした研究ではなく、開発プロセス、開発スタイルについて同等のカテゴリのパッケージソフトと比較を行っている点において、von Krogh & von Hippel (2003)が見出した事実を裏付け、von Krogh et al. (2003)の研究を補完することができるのではないだろうか。

この他 OSS に関しては、開発者の立場やオープンソース・ソフトウェア運動への参加者からの見解、知見の報告として Dibona et al. (1999)などがある。これらの見解、知見の報告は、各々に示唆に富むものではあるものの、開発の進め方や開発組織に関する体系的な記述、分析を欠いている点で問題がある。

[6] 例えば、単にメーリングリストに参加している人と開発コミュニティに参加していると見なせる人は厳然と区別可能であること、後者は 30 人程度と比較的少数だが入れ替わりが激しいこと、多くの開発者の専門化(specialization)が進むのと同時にコア開発者の非専門化(generalization)が進行すること、開発コミュニティへの参加の容易さ／難しさ(contribution barriers)を左右する要因としてソフトウェアを構成するモジュールの性質、使用開発言語、ソフトウェアのアーキテクチャ、モジュールの独立性の高さがあること、などである。

またより視野を広げると、開発活動そのものについてではないもののユーザと開発主体・開発者との関係、ユーザの開発活動への部分的関与を取り上げた研究も近年見られるようになっている。例えば、宮垣・佐々木 (1998)では、研究対象としてシェアウェアを取り上げ、その開発者とユーザとの関係性に焦点を当てて分析を行っている。この研究では、21 人のシェアウェア作者へのインタビューを通じて、その開発の動機、ユーザとの関わり方について記述し、一般的なソフトウェア製品とは異なるシェアウェアの位置づけ、存在の合理性を明らかにしている。

これに続く研究としては、佐々木・北山 (2000)が挙げられる。この研究は Linux の事例を題材にして、Linux の開発コミュニティがどのような存在であるのか、企業がそうした開発コミュニティとどのような関係を取り結んでいるのかということを記述・分析している。

また野島 (2002)は、オンライン・ゲームを対象に、ユーザが形成するコミュニティを、企業の収益に結びつけるためには企業側にどのような取り組みが必要かを論じている。

これらの研究は、いずれも開発主体（企業）とユーザとの関係、相互作用を扱っているため、参考とすべき点を含んでいるが、ユーザあるいはユーザ・コミュニティとの関係の構築、維持にのみ関心が寄せられている。そのため、ソフトウェアの開発組織や開発過程がどのようなものであり、その中でユーザはどのような役割を果たしているのかということまでは明らかにされていない。

2.3 既存研究の問題点と本研究の目的

以上のように、1980 年代末に本格化した、ソフトウェア開発に関する経営学的観点からの研究は、実務界の変化に対応してその対象を広げてきた。特に近年では OSS とその背景にある（オンライン）コミュニティにその研究上の関心が集まっている。

しかし高橋・高松 (2002)が指摘したように、Linux の成功は「無償」で「オープンソース」であったためであるとは言い難く、OSS はソフトウェア開発が成功するための必要条件ではあっても、決定的な成功要因とは言えない。また、オープンソース運動に関係した人々の意見や証言をまとめた Dibona et al. (1999)を参照すれば、OSS がそれ以前のフリーソフトウェア運動の延長線上にあり、開発コミュニティと企業とのアライアンス（佐々木・北山 (2000)）などへの配慮から、半ば「人工的に」創り出されたものであるという歴史的事実もある。

OSS に関するより正確な認識と評価に基づくこれらの研究から、OSS が具体化した「無償」「オープンソース」であることは、ソフトウェアの普及の成功要因として一定の評価ができる（佐々木・北山, 2000; West, 2003）ものの、ソフトウェアの開発の成功要因であるとは言い難い。したがって Linux および OSS の成功のみをことさら重要視して、ソフトウェア開

発の正否について論じることには問題があると思われる。ソフトウェア開発の成功要因は、「無償性」や「オープンソース」にあるとは必ずしも言えず、他の要因も視野に入れてソフトウェア開発の成功要因を考察していくべきであると考えられる。

そこで本研究では、必ずしも OSS ではないが、インターネットを全面的に活用しているソフトウェアとして「脱パッケージソフト」を定義し、研究対象とすることにした。パッケージソフトと、脱パッケージソフトとを比較することを通じて、ソフトウェア開発とそのマネジメントに関する新たな知見を得ることが本研究の目的である。

３．フレームワークと調査の概要

3.1 本研究のフレームワーク

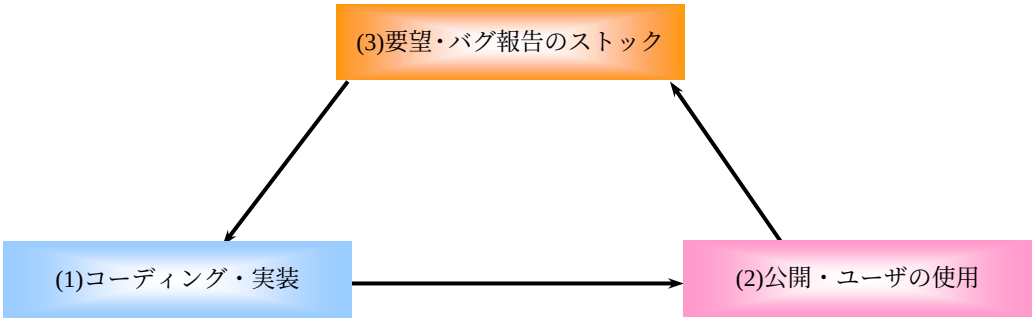
本稿ではソフトウェア進化のレベルの違いを説明するために開発サイクルという概念を提示して用いる。実は、既存研究の多くでは、暗黙の内に、ソフトウェアの開発が、コンセプト創造(目標設定)に始まり開発終了・公開で完結する一回性の高い「プロセス」だと認識していた。

これは、それ以前のソフトウェア・エンジニアリングあるいは製品開発研究の影響であるが、これでは開発プロセス間の連続的運動を認識することができない。ソフトウェアだけでなく製品開発全般においても、前のバージョン／ある製品の開発プロセスで蓄積された知識や、ユーザの使用体験のフィードバックが、次のバージョン／別の製品の開発プロセスに重要な役割を果たすことは、既に指摘されている（例えば、延岡 (1996)など[7]）。そこで本稿では開発サイクルという概念を提示することで、開発プロセス間の連続的運動を分析することにする[8]（図 1）。

[7] ただし、延岡 (1996)、青島 (1997)、青島・延岡 (1997)などが焦点を当てているのは、企業（開発主体）内部のプロセス間の連携・連続性、知識移転である。しかしソフトウェアの場合、バージョンアップにはユーザからのフィードバックが重要であるので、企業外部に存在するユーザの活動を媒介したプロセス間の連続性にも注目する必要があると考えられる。

[8] Iansiti (1998)はユーザからのフィードバックに言及しているものの、それが開発に及ぼす影響を実証してはいない。

図 1　開発サイクル



同時にこの開発サイクルは、企業の境界内に閉じこめて理解しないようにしたい。とくにソフトウェア開発に関しては、優れた能力を持つユーザが数多く存在する。フリーウェア、シェアウェアでは、ユーザ・イノベーション（von Hippel, 1998）のようにユーザが原作者の開発活動に積極的に関与することで、その機能・性能が予想外の方向に進化することが多々あるという（宮垣・佐々木, 1998）。したがって、ソフトウェアの開発サイクルは、企業など元来の開発主体の境界を超えて自由に広がりうるネットワークとしての組織（高橋, 1995; 2003）を前提に機能すると考えたい。

3.2 調査の概要

一口に脱パッケージソフトと言っても、非常に多くのソフトウェアが存在し、種類・機能も多岐に渡っている。そこで、脱パッケージソフトに関する研究の第 1 歩として、その中でも代表的なメールソフトの事例を取り上げることにする。メールソフトを取り上げるのは、コンピュータをインターネットに接続して利用することが当然のこととなった現代において、Web ブラウザに次いで利用頻度が高いソフトウェアになっているからである。また、メールソフトはパッケージソフト、脱パッケージソフトいずれの形態の事例も収集可能だからである。

そこで今回は、パッケージソフトの代表として、日本の代表的なソフトウェア企業Ａ社が開発、発売しているメーラーＸを調査対象とした。脱パッケージソフトに関しては、著名なオンラインソフトの配布サイトに登録されているメールソフトの中から、シェアウェアの「鶴亀メール」、フリーウェアの「電信八号」を選択した。このような配布サイトに登録されていることは、そのソフトウェアの利用者が多いことを示しており、成功事例とみなせるからである。したがって本研究は、パッケージソフト、脱パッケージソフト共に成功事例を対象としていることになる。

調査方法としてはインタビュー調査を主体に実施し、電子メールによる追加質問で適宜補

った。インタビューは 2002 年 10 月～2003 年 9 月に行われた。インタビューでは、事前に質問項目をリストアップしてインタビュイーに送付し、必要に応じて筆者が質問を補いながら質問票に沿って話してもらった。質問項目には、ソフトウェアの概要、ソフトウェア開発の契機、ソフトウェアの開発とバージョンアップのプロセス、開発組織、サポート体制などに関するものがある。

４．パッケージソフトの事例

4.1 Ａ社の概要とメーラーＸの開発契機

Ａ社は、多くのパッケージ・ソフトウェアやシステムを様々なプラットフォームで開発、発売している日本有数の独立ソフトウェア企業である。同社は 1990 年代初頭からネットワーク技術に関する研究、製品の開発を始める。最初に開発されたのは LAN（Local Area Network）用グループウェアであり、その中には LAN 内限定のメールシステムも含まれていた[9]。

1995 年頃になると、Ａ社内部でインターネット用ソフトウェアを開発する動きが本格化し、ユーザ側からも「インターネットを使いたい」という要望が出るようになった。Ａ社はそれまでの技術開発、製品開発の成果の蓄積を活かし、ユーザ側の要望に応える形で、1998 年に社内開発の段階に留まっていたソフトウェアをユーザに提供し始めた。提供したソフトウェアはブラウザとメーラーＸであり、どちらも主力製品のワープロソフトにバンドルする形で提供された。さらにメーラーＸに対する反応が良好であったため、2000 年にその高機能版を単独のソフトウェアとして発売することになった。

メーラーＸは、Ａ社独自のソフトウェアであり、基本性能も高い。例えば、HTML (Hyper Text Markup Language)形式のメールを表示するときに Windows 標準の「Internet Explorer (IE)」のコンポーネントではなくＡ社開発のモジュールを使用するためコンピュータ・ウィルスに強いという特徴がある。この高機能性に加え、インターネットで 2000 年頃からウィルス・メールが蔓延するようになったことがメーラーＸの販売を後押しした。メールを介したウィルス感染の危険性が認知されるまではメールソフトは買うものという意識がなかったのだが、ウィルスの危険性が認知されるにつれ、それに備えてメールソフトを購入しようというユーザが増え、メーラーＸへの需要も増えたのである。こうしたユーザの要望、需要増に応

[9] 当時は、さまざまなソフトウェア会社がそれぞれに独自の仕様にもとづいてメールソフトを開発している時期であり、メールソフトが相互にメールのやり取りができないという問題を抱えていた。この問題は、1994 年にメールソフトを手がけていた企業 8 社でメールを相互にやり取りするために日本初の共同研究を行い、それ以後のメールソフトの技術的な基礎固めを行ったことで解決された。

える形で、A社は 2002 年、2003 年と約 1 年おきにメーラーXをバージョンアップして現在に至っている。

なおメーラーXの販売形態は、最初のバージョンではユーザがインターネットを通じてソフトウェア本体をダウンロードして購入するダウンロード販売とパッケージ・ソフトウェアとしての販売を行い、2002 年のバージョンではダウンロード販売のみにし、2003 年のバージョンでは、ユーザと小売店の要望に応えるためにダウンロード販売とパッケージ・ソフトウェアとしての販売を並行して行っている[10]。

4.2 A社の開発組織とメーラーXの開発プロセス、バージョンアップ・プロセス

A社の開発活動は、ソフトウェアを全てコンポーネントに分割して開発する、オブジェクト指向、コンポーネント中心の活動になっている。これは、同社が基本的に内製で開発をまかなう企業であり、しかも製品数が多いことによる。A社が採用するオブジェクト指向型の組織において、1 コンポーネントは数百行～数万行の規模であり、このコンポーネントの開発に携わるのは 1 人～数人である。A社全体としては、このようなコンポーネントを数万～数十万の単位で保有、開発をしており、これらのコンポーネントを組み合わせる形で各々のシステム、ソフトウェア製品が開発されている[11, 12]。

このようなオブジェクト指向の全社的開発組織の中で、個々のシステム、ソフトウェア開発の中心的役割を果たすのは、プロダクトオーナーとビジネスオーナーである。プロダクトオーナーは、約 30～40 の製品数とほぼ同数がおり、彼らが技術面の管理を行って、コンポーネントを製品として取りまとめている。プロダクトオーナーに求められるのは、基本的にソフトウェアの品質管理と納期管理である。一方、ビジネスオーナーは、営業部門とともに市場動向、「製品の売れ筋」に目を配っている。

そして、プロダクトオーナー、ビジネスオーナー、営業の 3 者がメール、テレビ会議などで情報交換をし、製品開発管理を行っている。具体的には、マーケティング担当者が常時行っているユーザや出版物などからの情報収集の結果を踏まえ、プロジェクト開始前に営業企画の代表者であるビジネスオーナーと、開発者を管理しているプロダクトオーナーが意見交

[10] メーラーXの売上本数は数万本クラスであり、A社の主力製品であるワープロソフトと比べると利益率はそれほど高くない。したがって、ダウンロード販売を行ってコストを削減したいのだが、ユーザなどの要望とA社全体としての販売促進活動も兼ねていると考えて、パッケージ・ソフトウェアとしての販売も行っているという。

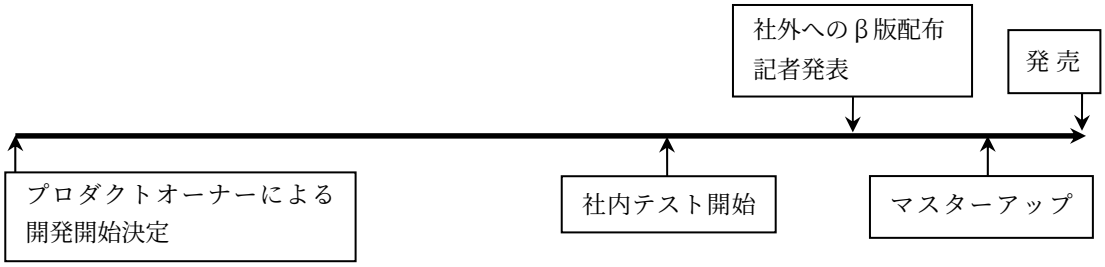
[11] このように、製品構造上冗長ともいえる形態であるため、A社製品のプログラム･サイズは大きい。

[12] こうした開発活動、開発組織構成を採っているため、必然的に組織構造は複雑になる。また、企業全体としてこのようなコンポーネント・ベースの開発組織を採用し、開発活動を実施しているため、製品毎に人月ベースなどで開発コストを計算することは難しい状況になっている。

換を行い、その結果を「製品化申請」として社内で公式に提示する。この「製品化申請」が上司、経営陣による「審査」を経て、申請が受理された後、製品開発が正式に開始される[13]。同時に、これら 3 者の情報交換に基づいて個々の製品の開発活動が具体化すると、プロダクトオーナーを中心とした製品開発のチームが立ち上げられる。このチームには、専任者と兼任者双方が含まれており、人数は最大でも 30～40 人である。

製品開発が正式に開始された後は、製品にもよるが、約 1 年ほどで製品が発売される。そのスケジュールを大まかに示すと下図のようになる。

図 2　製品開発のスケジュール



なお、A社は過去に、深刻なバグ問題が生じて数億円の損害を被った経験がある。こうした経験も踏まえ、現在はかなり手厚く、会社全体でテストを行っている[14]。また、開発者、会社の意識として、「バグを出したら信用はゼロになる」「アップデート・モジュールもほんとうは出したくない」「不具合修正のためのダウンロード・モジュールを出すことは「恥ずかしい」」という意識を持っているため、相当のコストを掛けてデバッグを行って、製品に含まれるバグなどがゼロになるように努めている[15]。

メーラーXの開発組織、開発プロセスも基本的には上述の通りであるが、いくつか特徴的な点もある。A社にとってメーラーXは、「基本的に「テキストデータのバケツリレー」を行うソフトウェアであり、ファイル（テキスト）の管理とテキストの入出力が中心のソフトウェアである」と捉えているが、その一方で、ユーザからの要望の「振れ幅」が依然として大きく、開発者の側が考える改良点と新規機能の追加が依然として幅広いものであるため、

[13] ただし、製品開発の正式開始前に、部分的に開発活動が始められることもあるという。
[14] 具体的には、開発部門だけでなくスタッフ部門もテストを担当している。この社内テストに当たっては、開発側からチェック項目に関する要望を提示し、それを中心に実際に使ってみた結果を報告してもらう形式を取っている。
[15] ただし、一般にソフトウェア開発において問題となる、テスト途中で発見されるバグおよびバグ・フィックスによるマスターアップや発売の遅れは基本的にないという。それは、前述のようにコンポーネント化が進んでおり、かつコンポーネントの最適化がなされているので、バグ・フィックスは容易であるからだという。

バージョンアップを定期的に行えるようになっていない。開発部門内の動向、市場動向、技術動向、ユーザからの要望などを勘案して、プロダクトオーナーが開発開始時期を検討し、それらに応じてバージョンアップ、アップデート・モジュールの提供が決定される。

そのバージョンアップに際しても、A社は基本的に慎重な姿勢をとっている。その背景には、流通を抱えており、かつ在庫はメーカー負担であるので過剰なバージョンアップはコスト負担が大きいことに加え、ユーザから見ても頻繁なバージョンアップには否定的だろうという見方がある。加えて、A社にとって製品は「1つのスタイル、使用・用途・役割・機能についての答え・結論・提案・最適化の結果」であるという認識もある。したがって、ユーザの声、要望を聞く耳は持っているものの、ソフトウェアを安易に修正・改変することはしないのだという。バージョンアップの際には、「この機能の追加でバージョンアップに値するのか」という開発者の悩みと、「内部的にはかなりの変化を遂げていても、それがユーザにとって目に見える変化であるのか」という課題が生じ、バージョンアップに際して、開発者には相当の重圧がかかっているという。

4.3 A社のサポート体制

A社は会社設立当初からサポート・センターを設置していた。一時期サポート・センターは、開発部門の下部組織だったが、現在はCS室として独立している。CS室は基本的に電話を通じたユーザ・サポートを行っており、その下部組織には、購入前のユーザを対象にした「インフォメーション・センター」と購入後のユーザを対象にした「サポート・センター」がある。インフォメーション・センターの役割は、的確な情報提供を行うことが目的である。他方、サポート・センターの役割は、ユーザの抱える問題を解決することと、次の製品へのヒントを得ること、情報収集などである。サポート・センターの場合、その目的を果たすためには自社製品をよくわかっている人、開発のことがわかる人が担当しなければならないので、基本的に正社員が対応に当たっている[16]。

同時に、インターネット（Web）を通じたサポートもA社は提供している。インターネットを通じたサポートは、情報の提供とプログラム・モジュールの提供に分けられる。情報の提供は、「よく寄せられる質問と回答（FAQ）」が中心であり、モジュールの提供は、製品の不具合の修正プログラムと、ユーザが製品をより便利に使えるようにするツール類の提供が含まれる。

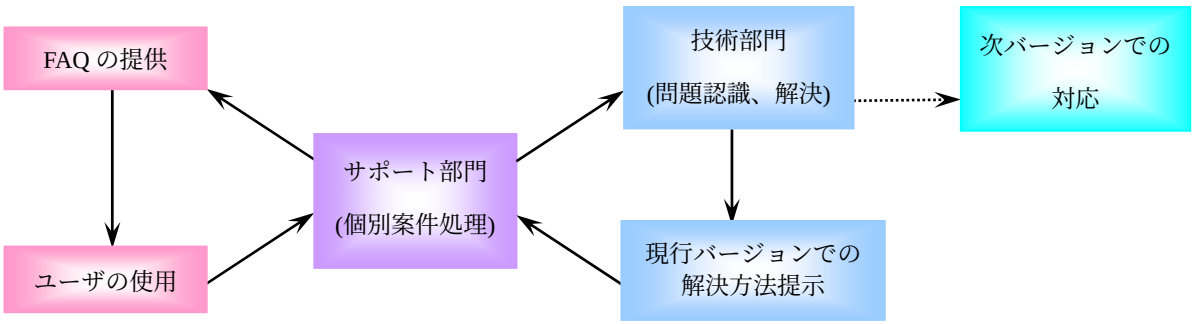
[16] ユーザに対応し、A社内に情報を伝える窓口となっているサポート部門は、ニュアンスまで含めてユーザが「何を言っているのか」を引き出し、A社の言葉に置き換え、不要な言葉を削ぎ落として、開発部門に伝える役割を担っている。

このうち、不具合修正のためのモジュール提供は、テスト担当部門や OS 研究を担当している部署からの報告を受けて開発部門が中心となって対処し、不具合修正モジュールができ次第、提供している。他方、ツール類は、営業および販売促進部門の提案によるものと開発部門の提案によるものがあり、どちらも開発部門が開発し、適宜提供している[17]。

こうしたA社のサポート体制を支える社内の情報交換は、基本的に確実性の高いメールベースで行われている。CS 室の下部組織であるサポート・センター、インフォメーション・センターといったサポート部門の受け取った情報は、報告書としてまとめられ、全社員が閲覧可能なようにデータベース化される。同時にサポート部門は、受け取った情報を基本的にそのまますべて開発部門に流しており、それに重み付けをしたり、重要度を設定したり、取捨選択したりすることは開発部門によって行われる[18]。

このようにサポート部門を中心にして様々なルートから開発部門に流入する情報を総合してビジネスオーナー、プロダクトオーナーが判断を下し、現行バージョンで解決をしたり、次バージョンで対応したりするなどしている。以上のような、サポート部門を中心とした情報の流れを図示すると図 3 のようになる。

図 3　サポート部門を中心とした情報の流れ(「8 の字型」の情報の流れ)



※簡略化のため、A社とユーザ間の情報の流れ、A社内部の情報の流れのみを記述した。

サポート部門、電話での対応と並んでA社のユーザ・サポートの一環をなしているのが FAQ であるが、FAQ の整備には、近年かなり力が入れられている。FAQ およびニュースリリースは 1990 年代後半には提供されるようになっていたが、1998 年、1999 年頃から FAQ の整備に本格的に取り組み始めた。FAQ に関しては、文書の作成の仕方、複数の形式で提示

[17] 不具合修正、ツール類共に十分なテストを行っているため、不具合修正やツール類の提供の結果、ダウングレード、新たな不具合の発生が生じることは稀であるという。

[18] サポート部門以外に外部から情報を取り入れるルートとして、営業部門が販売店などから入手した要望や売れ筋情報、他社製品の情報、マスコミなどから得られる一般情報などがあり、これらも開発部門に流れ込む。

するなどといった提示の仕方を含め、改善に努めてきたという。

特に、現在の FAQ の根幹をなす自社開発の自然文検索システムは、1997 年頃から本格的に利用が始められた。自然文検索システムの開発、活用に力が入れられているのは、ユーザから寄せられる情報は「○○ができない」が中心であり、A社が提供する情報は「□□ができる」「□□するためには」が中心であるため、両者のミスマッチを防ぐために、このシステムが有効であるからだという。

以上のようなサポート部門のノウハウの蓄積、FAQ の整備、そしてユーザ側のソフトを使う能力向上の結果、現在ではサポート部門の負担、人員数は減らすことができるようになり、コールセンターの対応時間も短くなったという。これは「ユーザの行動をある程度コントロールし、スクリーニングできるようになった」ということである[19]。

５．脱パッケージソフトの事例[20]

5.1 ソフトウェアの概要と開発契機

シェアウェアの鶴亀メールは、斉藤秀夫氏が開発した Windows 対応の高機能メールソフトである。メール本文を編集するエディタ部分には、同じく斉藤氏が開発し、日本の Windows 用エディタの定番とも言える「秀丸エディタ」のコンポーネントが利用されている。

フリーウェアの電信八号は、石岡隆光氏が開発し、現在は任意団体「電八倶楽部」および「電八開発倶楽部」が開発を引き継いでいるメールソフトである。その機能性、完成度は非常に高く評価され、財団法人インターネット協会(Internet Association Japan)が主催した、第 5 回フリーソフトウェア大賞(FSP’96)で入賞を果たしている。

いずれのソフトウェアとも、ユーザからの要望が開発着手のきっかけになっている。もともと斉藤氏は Xaxon 社のメールソフト「NetMail」に秀丸エディタをモジュールとして提供していたが、NetMail の販売停止にともなってそのユーザから、「秀丸エディタ・ベースのメールソフトで、動作するものであればいいから提供して欲しい」との要望が寄せられた。斉藤氏はこうした要望を受けて、2000 年 4 月からフルスクラッチでコードを書き起こしてメールソフト開発を開始し、同年 8 月に鶴亀メールを公開している。

電信八号の場合は、石岡氏のユーザとしての要望が開発の起点になっている。石岡氏がインターネットを利用し始めたころに出逢ったメールソフトは、どれも非常に使いにくく感じ

[19] ただし、A社においてサポート部門の運営、FAQ の作成・提示が比較的容易なのは、国内限定で、かつユーザのレベルが比較的一定であるという要因も作用しているという。
[20] 詳細なケースは、オンライン・ソフトウェア研究会のホームページ (http://www.gbrc.jp/onlinesoftware/)、藤田・生稲(2004)を参照のこと。

られ、1995 年春頃には「自分が使いやすい Windows 用メールソフトを作ろう」と考えるようになり、開発に着手したのである。開発開始から約半年後の 1995 年秋にはソフトウェアが安定してきたので、Asahi ネットのソフトウェア掲示板に電信八号を投稿し公開に至っている。

電八倶楽部への開発移管の理由は、石岡氏の開発への熱意の低下である。1998 年頃には「自分で使うには十分」と思うようになり、それまではユーザからの要望に応えて 1 ヶ月に複数回バージョンアップを行っていたが、事実上バージョンアップを止めてしまっていた。ところが電八倶楽部というユーザの有志団体が熱心に活動していることを知り、その熱意に対して石岡氏はソースコード公開というかたちで応えることになる。

5.2 バージョンアップ・プロセス

鶴亀メールも電信八号も、バージョンアップは基本的にユーザからの機能追加の要望、バグ報告が起点となっている。鶴亀メールの場合は、斉藤氏の経営する有限会社サイトー企画のホームページ内に設置されたサポートフォーラムに、ユーザからの要望やバグ報告が寄せられる。機能追加のうち、マクロ[21]で対応可能なものはマクロ開発や関数の追加で要望に応え、対応不可能なものに関してバージョンアップで応じる。

斉藤氏によれば、マクロによる対応を優先するのは、バージョンアップで対応すると鶴亀メール本体において新たなバグが発生する可能性があるためだという。実際、これまでのバージョンアップの際にもバグが発生し、「ほら、要望に応えたらバグが出ちゃったじゃないか…どうしてくれるんだ」という思いを抱いたことがあるという。

また、バージョンアップに際してのテストやデバッグをサイトー企画で行うことはない。すなわち、開発したものはすぐに公開し、サポートフォーラムを通じてバグ報告や機能追加の要望を受け付けて、次回バージョンアップ時にバグ修正、機能追加として反映させる。バージョンアップに対してはこのような姿勢をとっているため、表 1 のように、結果的に非常に頻繁なバージョンアップが行われている。

[21] マクロとは、ユーザが行う、決まり切った操作などを自動化するために作成する簡単なプログラムである。鶴亀メールでは、ユーザによる独自機能の追加はマクロによって実現されている。斉藤氏は基本的に開発者向けキットやソフトウェアのインターフェイスを公開していないためであるが、これは鶴亀メールへの独自機能の追加が制限されていることを意味しない。むしろ、非常に多数のマクロが作成・提供されており、しかも内蔵エディタである秀丸エディタについてはさらに多くのマクロが提供されている。

表 1　鶴亀メールのバージョンアップ履歴と頻度

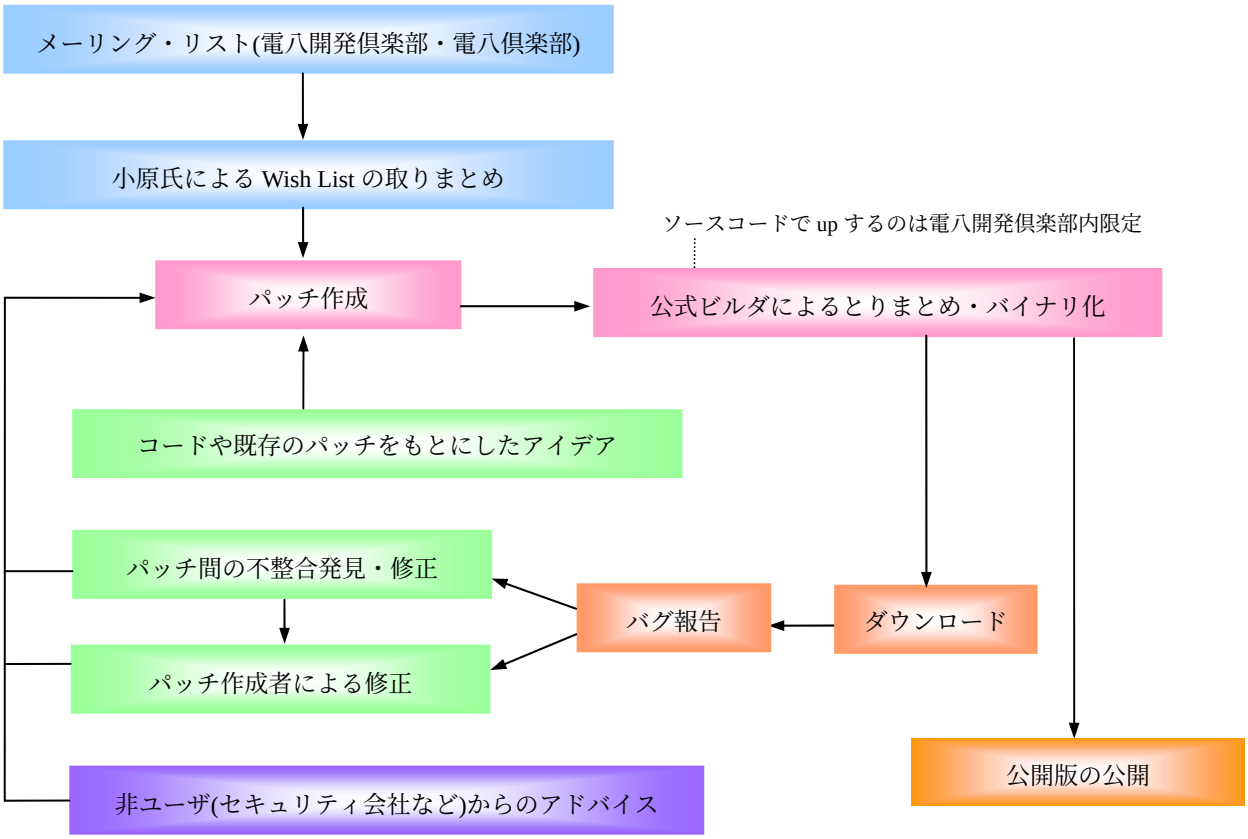
	公開日	一日あたり問題解決数	1 バージョンあたり開発日数	1 バージョンあたり問題解決数	バージョン数	日数	問題数
ベータ版	2000.8.21～	5.310	2.598	13.793	82	213	1131
Ver.1	2001.3.27～	3.017	4.724	14.255	98	463	1397
Ver.2	2002.6.28～	2.290	4.056	9.289	90	365	836
全体		3.232	3.856	12.459	270	1041	3364

※2003 年 6 月 28 日現在。公式サイト「改訂履歴」をもとに筆者が集計。

電信八号では、電八倶楽部・電八開発倶楽部のメーリングリストにユーザからの要望やバグ報告が投稿される。これらの情報は、公式ビルダと呼ばれる主要メンバーの 1 人によって WishList に集約される。この WishList をもとに、機能追加やバグ修正をプログラムとして実現するためのパッチが作成される。パッチ作成は、公式ビルダを含めた開発グループの誰かが、寄せられた要望や修正点に共感すれば行われるが、そうでなければ未処理のまま WishList に残り続ける。また一部のユーザが自ら問題修正のためにパッチを作成したり、原作者のコードの技術的な改良のためにパッチが作成されたりすることもある。

これらのパッチを統合し、バイナリ形式に変換してサーバにアップするのが公式ビルダである中村賢一郎氏と福井貴弘氏である。新しいパッチを統合し、バージョンアップされた電信八号は、動作が不安定になる可能性がある場合にはα版として、その危険性がなく安全だと思われる場合にはβ版としてサーバにアップされ、α版は電八開発倶楽部登録者が、β版は電八倶楽部登録者がダウンロード可能になる。多くの人が関わる電信八号のバージョンアップのプロセスは鶴亀メールよりいくぶん複雑で、図示すると図 4 のようになる。

図 4　電信八号のバージョン・アップ・プロセス[22]



電八開発倶楽部および電八倶楽部の登録者はα版やβ版をダウンロードし、使用してみて、バグ報告やバグを修正するためのパッチを提出する。それらを受けて再度パッチをとりまとめ、不具合の修正が行われると、一般向けの公開版がリリースされる。中村氏らは口をそろえて、電信八号の「開発スピード、バージョンアップのペースは非常に速い」という(表 2)。

[22] 修正･改良されたバージョンの正確な名称と公開先については、http://denshin8.esprix.net/patches.html - develop を参照されたい。

表 2　電信八号のバージョンアップ履歴と頻度

	公開日	一日あたり問題解決数	1 バージョンあたり開発日数	1 バージョンあたり問題解決数	バージョン数	日数	問題数
Ver.1	1995.7.14〜	0.196	20.067	3.933	15	301	59
Ver.32.1	1996.5.10〜	0.152	43.083	6.533	60	2585	392
全体		0.156	38.480	6.013	75	2886	451

※6 月 28 日現在。電信八号のパッケージに同梱されているリリースノートと、公式サイト「電信八号ー移管後の歩み」をもとに筆者が集計。各バージョンアップで解決された問題数は、リリースノートに記載されていないものもあったので、実際にはこれより格段に多いはずである。

バージョンアップの目安であるビルドは、毎週提出されるいくつかのパッチを受け入れる形で行われる。ビルドは中村氏と福井氏がほぼ交代で行っている。ビルドの作成自体は半日〜1 日ですべて終わらせることができる機械的な作業のため、平均してほぼ 1 ヶ月に 1 回のペースでビルドが続けられ、公開版が公開されている。

5.3 サポート体制とユーザの役割

鶴亀メール、電信八号とも、ユーザからの意見・要望・バグ報告はインターネット経由で開発者の元に届けられる。いずれも投稿や発言のためには登録手続きが必要ではあるものの、制限は一切なく誰でも自由に登録が行える。

鶴亀メールではサポートフォーラムと呼ばれる掲示板が利用されており、発言するためには会員登録が必要であるが、閲覧は自由にすることができる。また、サポートフォーラムへは、サイトー企画のホームページである「秀まるおのホームページ」から簡単に行くことができるようになっている。サポートフォーラムへのアクセスが簡便であるだけでなく、斉藤氏は「サポート会議室に寄せられるユーザからの質問にはきちんと答えるべきである」と考えており、ユーザへの対応は懇切丁寧である。ユーザの中には電子メールを利用して意見・要望を伝えてくる人もいるが、その場合でもきちんと返信し、適切なサポート会議室を紹介したりして対応している。なお、ユーザとのオフラインでの交流はほとんどない。

電信八号ではメーリングリストが利用されているため、投稿には登録が必要であるが、「誰でも入れて、いつでも抜けられる」メーリングリストなのでその敷居は低い。現在のバージョンアップに対するユーザの貢献に関しては、中村氏らが異口同音に、「電信八号の開発において、メーリングリスト参加者の中に優秀なデバッガ（バグを出し、報告してくれる人材）

がいることが大きい」と述べていた。ユーザの中には、修正項目のすべてをテストしたり、バグの再現条件を克明に記したレポートを提出したりする人もいるという。ただし、電八倶楽部・電八開発倶楽部の開発者、ユーザ間の交流はメーリングリストがほぼすべてと言ってよく、オフラインでの交流は皆無に等しいという。

６．パッケージソフトと脱パッケージソフトの事例比較

6.1 事例の整理と比較

まず、前節の事例を整理し、パッケージソフトと脱パッケージソフトの開発活動において異なる点をまとめておこう。パッケージソフトと脱パッケージソフトの事例において、異なる点は主に 3 つにまとめられる。

その第一は、ソフトウェアをユーザに提供する前のテスト、デバッグの厳密さに関する違いである。Ａ社メーラーＸの場合には、非常に時間と人手を掛けて厳密なテストとデバッグが行われていたのに対し、鶴亀メールや電信八号の場合には、それがあまり厳密には行われていなかった[23]。

第二の相違点は、ユーザからのフィードバック―質問、要望、バグ報告など―のなされ方である。メーラーＸではＡ社のサポート部門が電話で受け付け、それが開発部門に渡されるという手順を踏む「間接的」なものであったのに対し、鶴亀メールと電信八号では、メールや Web ページ上の掲示板、フォーラムなどを通じて開発主体に「直接」情報がフィードバックされていた。

第 3 の相違点は、ユーザからのフィードバックが開発主体によって取捨選択（スクリーニング）がなされているか否かである。メーラーＸの場合には、開発部門が情報の取捨選択を行っていたが、鶴亀メールの場合にはサイトー企画で情報の取捨選択を行うことは少なく、電信八号の場合には WishList という形で全てのフィードバックがリストアップされ、そこから情報は自然淘汰される仕組みになっていた。つまり、メーラーＸの場合には情報の取捨選択が意識的に行われていたが、鶴亀メール及び電信八号の場合にはその意図が弱いといえる。

これらの違いは表 3 のようにまとめることができる。

[23] より正確に言えば、電信八号の場合には、電八開発倶楽部で配布されるα版で最もテストとデバッグの厳密性が低く、ついでβ版でのテストとデバッグも行われるため、一般公開されるバージョンではかなりテストとデバッグがされた状態のソフトウェアが提供されていると考えられる。

表 3　事例の比較結果

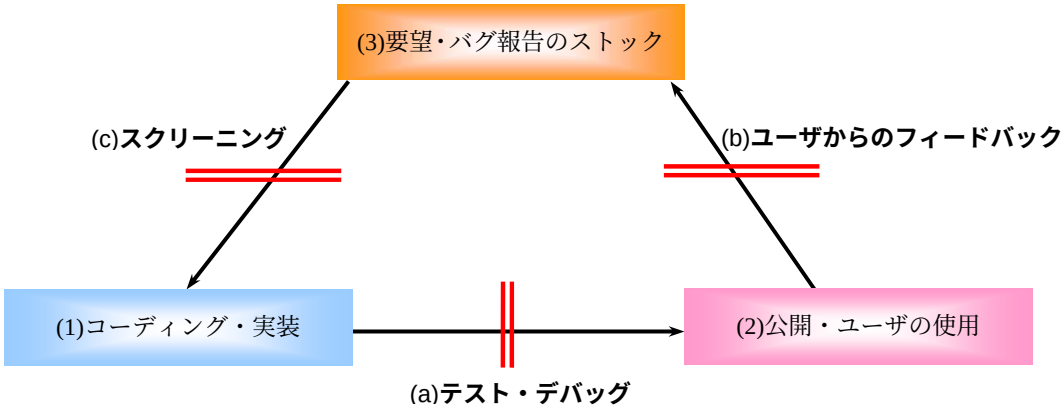
	パッケージソフト	脱パッケージソフト
(a)ユーザに提供する前のテスト、デバッグの厳密性	厳密	緩やか
(b)ユーザからのフィードバック	間接的	直接的
(c)フィードバックされた情報の取捨選択	強い	弱い

6.2 開発サイクルの相違

続いて、本研究の分析視角である開発サイクルを適用して、パッケージソフトと脱パッケージソフトの開発事例を比較してみよう。

既に述べたように開発サイクルは、(1)コーディング・実装、(2)ユーザの使用、(3)ユーザからの要望、バグ報告のストック、という 3 つの基本的活動から成り立っている。これに表 3 で示した 3 つの活動 ―(a)ユーザに提供する前のテスト、デバッグ、(b)ユーザからのフィードバック、(c)スクリーニング―を重ね合わせると、(1)～(3)の活動間の移行を妨げる「関門」として位置づけられる。こうした関係を図示すると、パッケージソフト、脱パッケージソフトそれぞれ図 5、図 6 のようになる。

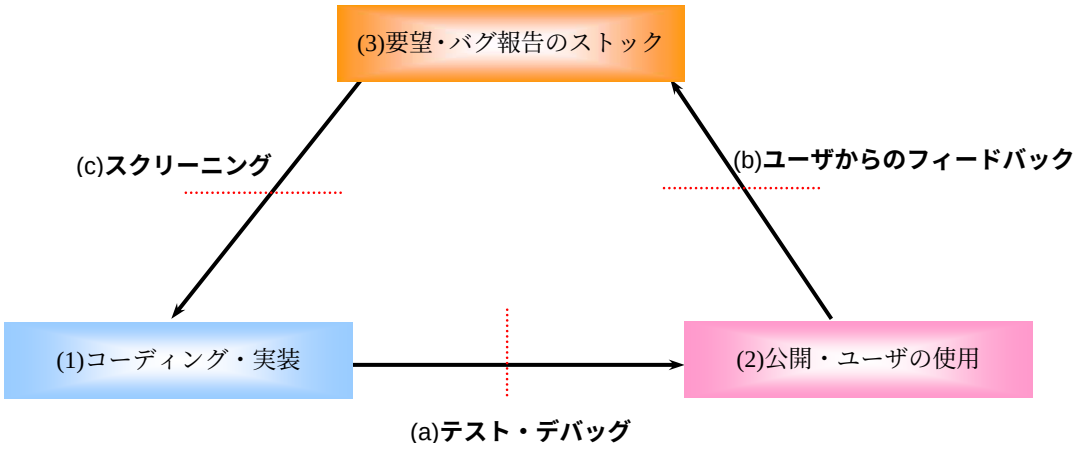
図 5　パッケージソフトの開発サイクル



事例及び図 5 で示したように、パッケージソフトでは、開発サイクルを構成する基本的活動の間に(a)～(c)の活動が関門として厳然と存在し、一つの基本的活動から次の基本的活動に移行する際に時間が掛かることになっていると考えられる。したがって、このような開発サイクルは「関門の厳格な相対的に遅い開発サイクル (Relatively-Dull Rigid-Gated Development Cycle) 」と呼ぶことができるであろう。

他方、脱パッケージソフトの場合は、図 6 のようになる。

図 6　脱パッケージソフトの開発サイクル



脱パッケージソフトの場合には、基本的活動の間に挟み込まれる(a)～(c)の活動が関門としてそれほど機能せず、ある活動から次の活動への移行に時間が掛からないと考えられる。したがって、脱パッケージソフトの開発サイクルは「関門が緩やかな相対的に速い開発サイクル (Relatively-Rapid Loose-Gated Development Cycle) 」と呼ぶことができるであろう。

そして、このような開発サイクルの違いが、実際にはソフトウェアのバージョンアップ・ペースとして観察されると考えられる。

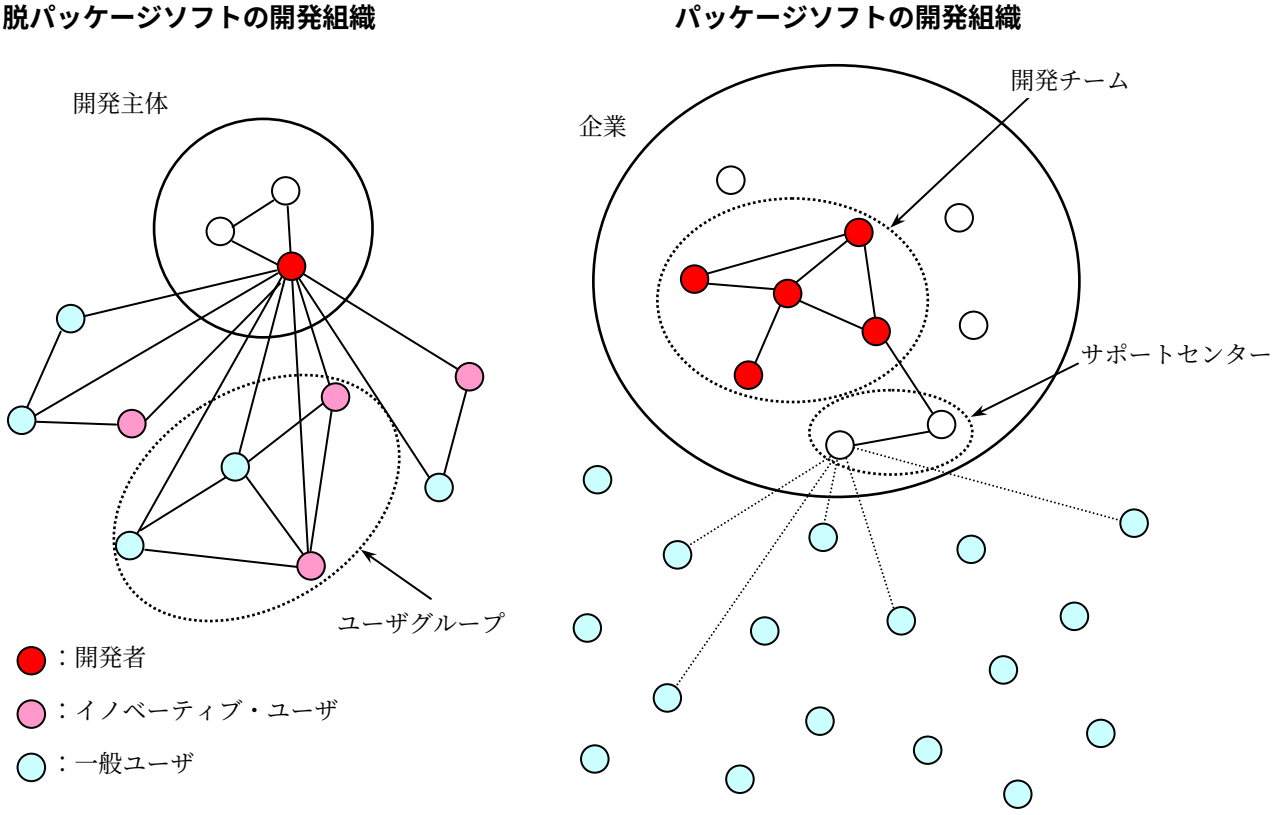
6.3 開発サイクルの規定因

では、このような開発サイクルの相違はどのような要因から生じるのであろうか。大きく 3 つの要因が挙げられよう。第一の要因は、開発主体とユーザが相互諒解し、共有するソフトウェアに関する認識すなわち「ソフトウェア像」である。メーラーXのようなパッケージソフトの場合には、多くの場合ソフトウェアはユーザが対価を支払うべき「製品」であると認識されている。それ故、開発主体は十分なテストとデバッグを行って、ユーザに提供されるソフトウェアが可能な限り完全なものであるように努めるであろう。他方、鶴亀メール及び電信八号は、ユーザ自身が入手、インストール、使用に責任をもって当たる User Supported Software であると認識されているため、テストとデバッグはそれほど厳密である必要は必ずしもない。つまり、ソフトウェア像がテストとデバッグの厳密性を左右すると考えられる。

第二の要因は「ユーザの組織化の程度」である。ここでユーザの組織化の程度とは、①ユーザと開発者／開発主体が 1 対 1 の関係を結べるか、②ユーザと開発者／開発主体の関係が

直接的か、③ユーザの中に開発に積極的に関与する「イノベーティブ・ユーザ」が存在するか、を指す。この概念を図示すると、図 7 のようになる。

図 7　ユーザの組織化



事例の中でも特にＡ社及び脱パッケージソフトのサポート体制の箇所で述べたように、パッケージソフトではユーザの組織化の程度が低く、脱パッケージソフトではユーザの組織化の程度が高いと考えられる。この相違は、ユーザからのフィードバックに時間が掛かるか、掛からないかを左右すると考えられる[24]。

第三の要因は、ソフトウェアのコーディング・実装を行い、公開するために要するコストすなわち「試行コスト」である。パッケージソフトの場合には、開発者の人件費、開発機材などの固定費に加え、パッケージングや流通に掛かるコストもあるため、試行コストは高くなる。そのため、無駄なコーディング・実装を避けようとする誘因が働き、その前段階にあ

[24] ユーザが組織化されることにより、本文のようにフィードバックが迅速に行われる効果以外に、鶴亀メールのサポートフォーラムのように、サポートが開発主体の手によらず、ユーザ相互間で行われる効果も期待できる。ただし、開発主体が関与しないユーザの組織化が進みすぎると「ユーザの自己組織化」とも呼べる状態になり、開発主体がフィードバックを受け取れなくなる障害などが生じる場合もある。

たるユーザからの要望、バグ報告などの情報を取捨選択すること、すなわちスクリーニングが厳密になる必要が生じる。他方、脱パッケージソフトの場合には、開発活動の一部をユーザが担い、しかもインターネットなどの手段を通じて配布するため試行コストは低い。したがって、結果的に無駄な、あるいは実験的なコーディング・実装を行う余地が生じ、その源泉となりうるユーザからの要望、バグ報告に対するスクリーニングが緩やかになる。つまり、試行コストの差異がユーザからの要望、バグ報告に対するスクリーニングの厳密性を左右すると考えられる。

７．結論

本稿では、既存研究でほとんど注目されることのなかった脱パッケージソフトを対象に、開発サイクルという分析視角を用い、事例に基づいてパッケージソフトと脱パッケージソフトの開発活動、開発スタイルの比較を行った。「良いソフトウェアを開発する」という目標はパッケージソフトでも脱パッケージソフトでも何ら変わるところがないが、その実現に際して脱パッケージ化という意思決定を行うことにより、異なる開発スタイルが現出することが示された。

具体的に述べれば、脱パッケージ化という意思決定は単なる流通経路の選択にとどまらず、ソフトウェアに関する認識と開発サイクルの選択であり、同時に流通経路の違いは、ユーザの組織化の程度、試行コストの違いを必然的に生じさせ、それがソフトウェアのバージョンアップ・ペースに結びつく。また、ユーザの組織化の程度と試行コストは、ソフトウェアの流通経路をパッケージ／脱パッケージのいずれにするかを選択した時点で決まるので、もともとパッケージング販売されていたソフトウェアが脱パッケージ化してダウンロード（オンライン）販売しても、その開発スタイルは変化しないことも理解される。

インターネットとブロードバンドが普及した現在では、インターネットを通じてユーザを組織化し、脱パッケージソフトのような「関門の緩やかな相対的に速い開発サイクル」によってソフトウェアを開発していくことが可能である。具体的には、ソフトウェア企業はパッケージ／脱パッケージの開発スタイルを組み合わせることで、より効果的にソフトウェア開発・ビジネスを展開していくことができるだろう。例えば、ソフトウェアをコア・コンポーネント（本体プログラム）と、追加機能・不具合修正のためのサブ・コンポーネント（パッチ）とに大別し、各々がパッケージ／脱パッケージで提供される可能性を考えてみると次の表のようになる。

表 4　ソフトウェアの開発・提供方法の組合せ

コア	サブ	例	
パッケージ	パッケージ	①業務用基幹システム、 一太郎	Cusumano, Iansiti らの
	脱パッケージ	②Microsoft Windows, Office	一連の研究
脱パッケージ	パッケージ	③Linux ディストリビューション	佐々木・北山 (2000)
	脱パッケージ	④シェアウェア、フリーウェア	本研究

従来ソフトウェア企業は、表 4 の①と②を中心にソフトウェア・ビジネスを展開しており、近年になってようやく③に取り組み始めた。しかし、④のような開発・提供方法もありうるべき選択肢であることが今回の事例からは示唆される。

しかし、残された課題が多いことも否定できない。今回の調査では、膨大な数のソフトウェアの中から、たった 3 つの事例しか取り上げていない。したがって、この研究で発見された開発サイクルやその規定因について、より多くの事例収集や実態調査を通じて、妥当性・一般性を確認していかなくてはならない。

また、この論文で提示した 2 つの開発スタイルは、現時点で観察された事例から抽出された典型的類型であるということにも注意しておく必要があるだろう。現実にはパッケージソフトと脱パッケージソフトの開発サイクルの対応関係は、これまで議論してきたような明確な対応関係を持っていない可能性もあるし、将来その対応関係が変化するかもしれないからである。したがって今後の研究においては、ソフトウェア像や試行コストといった、開発サイクルの規定因に遡って、個々のソフトウェアおよびその開発サイクルを適切に位置づけることが必要であると考えられる[25]。

[25] 加えて、高橋・高松 (2002)、von Krogh & von Hippel (2003)で指摘されたソフトウェア開発における動機づけの問題も、今後の重要な研究課題であろう。脱パッケージソフトは、多くの場合、開発者がボランティア精神で開発に携わり、ユーザがリスクを冒して使用しているソフトウェアである。こうした開発、使用が継続されるのは、時間を経てソフトウェアが「良くなる」という楽観的な見通しに支えられている。こうした楽観的な見通しを開発者、ユーザに与え、与え続ける動機づけのあり方については今後の研究で明らかにする必要があると考えている。

参考文献

青島矢一(1997)「新製品開発研究の視点」『ビジネスレビュー』45(1), 161-179.

青島矢一・延岡健太郎(1997)「プロジェクト知識のマネジメント」『組織科学』31(1), 20-36.

Brooks, F. (1975; 1995) *The Mythical Man-Month: essays on software engineering, 20th Anniversary edition.* Addison-Wesley, Boston; MA.（滝沢徹・牧野祐子・富澤昇訳『人月の神話［増訂版］』アジソン・ウェスレイ・パブリッシャーズ・ジャパン(星雲社), 1996）

Cusumano, M. A. (1991) *Japan's Software Factories: A Challenge to U.S. Management*. Oxford University Press, Inc.（富沢宏之・藤井留美訳『日本のソフトウェア戦略：アメリカ式経営への挑戦』三田出版会, 1993）

Cusumano, M. A. and R. W. Selby (1995) *Microsoft Secret: how the world's most powerful software company creates technology, shapes markets, and manages people*. The Free Press.（山岡洋一訳『マイクロソフトシークレット―勝ち続ける驚異の経営―』日本経済新聞社, 1996）

Cusumano, M. A. and D. B. Yoffie (1998) *Competing on Internet Time: lessons from Netscape and its battle with Microsoft*. The Free Press.（松浦秀明訳『食うか食われるか―ネットスケープ vs.マイクロソフト―』毎日新聞社, 1999）

Dibona, C., M. Stone, and S. Ockman (1999) *Open Sources: Voices from the Open Source Revolution.* O'Reilly & Associates. （倉骨彰訳『オープンソースソフトウェア』オライリー・ジャパン, 1999）

Iansiti, M. and A. MacCormack (1997) "Developing Products On Internet Time," *Harvard Business Review*, Sep-Oct.（Diamond ハーバード・ビジネス・レビュー編集部訳「インターネット時代の製品開発」『ネットワーク戦略論』ダイヤモンド社, 2001）

Iansiti, M. (1998) *Technology integration: Making critical choices in a dynamic world.* Harvard Business School Press, Boston; MA.（NTT コミュニケーションウェア訳『技術統合―理論・経営・問題解決―』NTT 出版, 2000）

藤田英樹・生稲史彦(2004)「オンライン・ソフトウェアの開発実態に関する調査報告書」東京大学 21 世紀 COE ものづくり経営研究センター ディスカッションペーパー (2004-MMRC-4). http://www.ut-mmrc.jp/DP/PDF/MMRC4_2004.pdf

宮垣元・佐々木裕一(1998)『シェアウェア』(金子郁容監修) NTT 出版.

延岡健太郎(1996)『マルチプロジェクト戦略―ポストリーンの製品開発マネジメント―』有斐閣.

野島美保(2002)「コミュニティと企業戦略の適合モデル―オンライン・ゲームの産業の事例

—」『赤門マネジメント・レビュー』1(7), 1-33. http://www.gbrc.jp/

Raymond, E. (1997) *The Cathedral and the Bazaar*. Retrieved April 2002. from http://www.catb.org/~esr/writings/cathedral-bazaar/（山形浩生訳「伽藍とバザール」http://cruel.org/freeware/cathedral.html）

佐々木裕一・北山聡(2000)『Linux はいかにしてビジネスになったのか—コミュニティ・アライアンス戦略—』(國領二郎監修) NTT 出版.

高橋伸夫(1995; 2003)『経営の再生［新版］—戦略の時代・組織の時代—』有斐閣.

高橋伸夫・高松朋史(2002)「オープンソース戦略の誤解—Linux はなぜ成功したのか—」『赤門マネジメント・レビュー』1(4), 1-26. http://www.gbrc.jp/

立本博文(2002)「ソフトウェア開発プロセスに関するレビュー論文—ソフト開発プロセスモデルと製品属性—」『赤門マネジメント･レビュー』1(4), 309-336. http://www.gbrc.jp/

von Hippel, Eric A. (1988) *The Sources of Innovation*. Oxford University Press, Inc.（榊原清則訳『イノベーションの源泉』ダイヤモンド社, 1991）

von Krogh, G., and E. A. von Hippel (2003) “Special issue on open source software development,” *Research Policy*, 32(7), 1149-1157.

von Krogh, G., S. Spaeth, and R. Lakhani (2003) “Community, joining, and specialization in open source software innovation: a case study,” *Research Policy*, 32(7), 1217-1241.

West J. (2003) “How open is open enough?: Melding proprietary and open source platform strategies,” *Research Policy*, 32(7), 1259-1285.