MMRC-J-4

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　3 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

東京大学 COE ものづくり経営研究センター　MMRC Discussion Paper No.4

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　3 月

目次

第１章　はじめに

ソフトウェアという概念は、コンピュータや自動車などのハードウェアと対比されるべき概念である。イノベーション研究、製品開発研究では、ハードウェア製品を研究対象とする場合、その開発対象が多様性を持ち、開発対象のタイプに応じて開発活動のあり方も異なるとする見方が一般的である。例えば、製品のアーキテクチャが異なれば、効果的な開発パターンも異なってくるとする考え方がある。ところが、そのような開発対象および開発活動の多様性がソフトウェア製品にも観察されるとする研究はほとんどなく、もっぱら「ソフトウェア開発」と一括して研究が進められてきた。すなわち、ソフトウェア開発を対象とした従来の研究は、様々なソフトウェアを開発する活動を一括して「ソフトウェア開発」と認識し、それに対応してソフトウェア開発の One-Best-Way を模索してきたように見受けられる。

確かに、ソフトウェアには技術的なあるいはアーキテクチャ上の多様性がハードウェアほどはないかもしれない。しかしながらソフトウェアは、その種類や機能、性能、ターゲット・ユーザ、流通経路などにおいて、ハードウェア以上の多様性を持っている。したがって、ソフトウェアの開発活動を研究するにあたっても、開発対象には多様性があり、そのタイプに応じて開発活動も異なることを予測しつつ研究を進めるべきではないだろうか。

このような認識にもとづいて、この研究では、ソフトウェアという範疇の中にも開発対象と開発活動の多様性があることを前提に、既存のソフトウェア開発活動研究で対象とされてこなかった「オンライン・ソフトウェア」を調査することを通じて、ソフトウェア開発活動の多様性を確かめることにしよう。

１．オンライン・ソフトウェア

コンピュータ・ハードウェアと、ブロードバンドと呼ばれる広帯域通信環境の進歩により、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一般的になりつつある現在、オンライン(インターネット上)[1]で開発活動や配布が行われるソフトウェア[2]の普及・利用が急速に拡大している。こうした開発・配布形態がとられるソフトウェアを、この研究では「オンライン・ソフトウェア」と呼ぶことにするが、その中でも最もよく知られているのは Linux という OS (Operating System)である。Linux はその世界的な規模の成功から注目を集めたが、実は Linux 以外にもオンライン・ソフトウェアは数多く存在し、コンピュータ・ユーザには広く利用されてきた。日本国内の開発者の手になるオンライン・ソフトウェアは一般に、有償のものが「シェアウェア」、無償のものが「フリーウェア」と呼ばれている[3]。

ところが既存研究では、例外とも言える Linux をのぞけば、企業が営利目的で開発しているカスタマイズド・ソフトウェア、あるいはパッケージ・ソフトウェアのみを対象としてきた。すなわち、多くのユーザを獲得することに成功したソフトウェアのみが研究対象とされてきたのである。他方、大多数のオンライン・ソフトウェアは、これら商用ソフトウェアと比べて利用者数も少ないし開発

---

[1] より正確には、インターネットに代表される高速、安価な情報網に接続したコンピュータおよびその利用が「オンライン」の意味するところである。

[2] 一口にソフトウェア、あるいはシステムと呼ばれるものの中には、それが稼動するハードウェア、対象とするユーザが異なるものが含まれる。そこで本稿では一貫して、その研究と議論の対象を、パソコン上でエンド・ユーザが利用可能なソフトウェアのみに絞り込むことにする。

[3] この研究で定義するオンライン・ソフトウェアと「シェアウェア」「フリーウェア」は、厳密には異なるものである。前者がソフトウェアの流通・配布形態に基づく定義であるのに対し、シェアウェア、フリーウェアはソフトウェアのライセンス、権利設定に基づく定義を持つ用語だからである。インターネットが爆発的に普及した現代では、オンラインで流通しないシェアウェア、フリーウェアはほとんど存在しない。したがって、オンライン・ソフトウェアは、シェアウェア、フリーウェアを包含する概念であると言えるだろう。そこで以下では、便宜的にオンライン・ソフトウェアをシェアウェア、フリーウェアの上位概念として位置づけて議論を展開していく。なお、オンライン・ソフトウェア、シェアウェア、フリーウェアの定義・詳細については、補章「ソフトウェアの分類に関する試論」を参照されたい。

組織の規模も小さい。このため、オンライン・ソフトウェアが注目を集めることはほとんどなく、ソフトウェア開発の研究対象としても取り上げられることがなかった[4]。しかし、オンライン・ソフトウェアの中には、機能や安全性の面で明らかに商用ソフトウェアを凌駕しているものが多数見受けられる。しかも、それらのうちのいくつかはシェアウェアとして公開され、ビジネスとして成立するに十分な数のユーザを獲得しているのも事実である。

２．開発サイクル

そこでこの研究では、オンライン・ソフトウェアの一種であるシェアウェア、フリーウェアの中でも比較的成功していると考えられる事例を取り上げ、その開発活動のあり方を調査することにした。具体的には、利用したことのある人が多いと考えられるメールソフト(電子メール・クライアント)を対象に、シェアウェア、フリーウェア各 1 本について、開発者に対するインタビュー調査を実施した。

事例調査を分析するに当たって本研究が依拠するのが、「開発サイクル」という分析視角である。ソフトウェア開発に関する既存研究がおもに考察の対象としてきたのは「開発プロセス」であった。そこでは、コンセプト創造(市場・ユーザおよび開発チームが提示する機能や仕様のスクリーニング)、基本設計・詳細設計(コーディング・実装)、テスト・デバッグ、公開といった一連の活動が、一回性のものとして見なされていた。

他方、本研究では、「開発プロセス」を連続的に循環させる過程全体を、「開発サイクル」と定義し、事例分析に適用する。より厳密に定義すれば、「開発サイクル」とは、1 つの開発プロセスの遂行、ユーザによる使用、ユーザからのフィードバックのストック、新しい(バージョンの)ソフトウェアのコーディング・実装、によって形作られる循環的かつ螺旋的運動である。ここで、循環的かつ螺旋的運動であるとは、開発プロセスの遂行、ユーザによる使用、ユーザからのフィードバックが繰り返し行われること(循環的)、同時に、開発サイクルの出発点である最初の開発プロセスは、それがユーザの使用、ユーザからのフィードバックを経た後の 2 回目の開発プロセスとは異なり、多くの場合、2 回目の開発プロセスの方がその成果であるソフトウェアの質が向上していること(螺旋的)、を指している。

このように定義される開発サイクルは、開発プロセスを包含し、また、それと相補的な関係にある分析視角であるといえる。

このような調査事例と分析視角を用いた本研究から、商用ソフトウェアとシェアウェア、フリーウェアの間には「開発サイクル」のレベルにおいて、表 1 のような相違点のあることが明らかになった。

表１　オンライン・ソフトウェアと商用ソフトウェアの相違点

		シェアウェア、フリーウェア	商用ソフトウェア
①開発サイクル		速い・関門なし	遅い・関門あり
②認識	(1)開発対象	User Supported Software	「製品」「商品」
	(2) ユーザ	実在のユーザ・一人称	平均的ユーザ・三人称
③試行コスト		低い	高い

すなわちシェアウェア、フリーウェアでは、ユーザの要望を取り込み、それをプログラムとして実現(コーディング)し、一般に公開するという 3 つの活動のあいだに、スムーズな移行を妨げるような活動(関門)が存在しないのである。

さらに、この事実発見に基づいて、そうした開発サイクルの相違がなぜ生じるのか、つまり開発サイクルの相違を生じさせる要因について考察を行った。その結果、スピードの速い開発サイクル

[4] シェアウェアに関する唯一の先行研究として、宮垣・佐々木(1998)がある。

を可能にしているのは、開発対象であるソフトウェアとユーザについての認識(「ソフトウェア像」「ユーザ像」)が商用ソフトウェアとは異なること、コーディングと公開・販売に要するコスト(「試行コスト」)が商用ソフトウェアよりも低いことに起因することが明らかになった。

こうした事実発見と考察から、ソフトウェアの開発サイクルには多様性があると考えることができる。シェアウェア、フリーウェアであっても、開発プロセスは商用ソフトウェアのそれと同じような活動から構成されている。ところが、その運用の仕方は両者でまったく異なっている、つまり、これまで商用ソフトウェアで実践されてきたものとは異なるロジックで動く開発サイクルが、シェアウェア、フリーウェアの開発では成立していると考えられるのである。

このような本研究の意義は、商用ソフトウェアを対象としてきた従来のソフトウェア開発研究が持っていた暗黙の前提を見直し、その知見を再検討することができたことにあるといえるだろう。より具体的には、①開発開始と開発終了が明示的で完結的な「開発プロセス」を補う分析視角として、「開発サイクル」という視角が有用かつ必要であること、②開発サイクルのレベルでの記述・分析を実際に行うことによりソフトウェア開発の多様性を示せたことである。これらの知見により、今後のソフトウェア開発に関する研究を進める上で有効なソフトウェア開発の分類枠組み、および企業のソフトウェア開発マネジメントに関する新しい知見を提示し得たと考えられる。

第 2 章　ソフトウェア開発に関する既存研究

1．ソフトウェア・エンジニアリングと経営学のソフトウェア開発研究

コンピュータの誕生からやや遅れてソフトウェアが生まれ、それからさらに遅れてソフトウェア開発に関する研究が始まった[5]。有名な初期のコンピュータである ENIAC は現在のソフトウェアに該当する役割をケーブルの配線で実現していたが、1950 年にその改良版として生み出された EDVAC では、ケーブル配線ではなくコンピュータの記憶領域に蓄えられたプログラムがコンピュータの挙動を決定することになった。これがフォン・ノイマン型、あるいはプログラム内蔵型と呼ばれるコンピュータであり、現在の全てのコンピュータはその子孫にあたると言える。同時に、この EDVAC というハードウェアの登場が、同時にソフトウェアの誕生であった。

このようにして生まれたソフトウェアは、コンピュータ、ハードウェア、あるいは関連する半導体技術の進歩と共に大規模化、複雑化していく。そのため、1960 年代後半になると、その開発活動の効果的・効率的なあり方が模索されるようになる[6]。これは主に工学系のエンジニアリングの立場からの開発活動に関する考察と実践であり、ソフトウェア・エンジニアリングと呼ばれるこの研究分野は現在でも活発に議論と知見の積み重ねが行われている分野である。

その後、ソフトウェアは主に 2 つの面で変化を遂げる。1 つは、ソフトウェア自体が大規模化、複雑化し、個人による開発が不可能になり、代わって組織的な開発活動が行われるようになった。これに伴い、ソフトウェア開発という活動が、工学的立場からの研究対象にとどまらず、組織現象を扱う学問、すなわち、経営学の研究対象となりうることとなった。もう 1 つのより重要な変化は、ソフトウェア開発の成果物であるソフトウェアが、製品として認知されて産業として確立し、しかも急速にその産業規模が拡大して、社会的影響力が強まったことである。

後者の象徴的な出来事としては、1969 年にコンピュータ産業のリーダー企業であった IBM 社が行ったアンバンドリング政策(価格分離政策)と、当時新興企業であったマイクロソフト社の Bill Gates が 1976 年に公開した「ホビイストたちへの公開状(An Open Letter to Hobbyists)」が挙げられるであろう。これら一部のソフトウェア開発主体の活動、およびそれに続くソフトウェアに関する知的財産権の明確化の結果、ソフトウェアの価値が確立し、それが社会的に認知されるようになり、さらにはソフトウェアを開発し、それをユーザに販売することによって収益をあげることが定着する。言い換えれば、「(商用)ソフトウェア」という製品分野が確立することとなった。このことも、営利企業のマネジメントを 1 つの研究対象とする経営学に、ソフトウェア開発への関心を持たせることとなった。

こうしたソフトウェアを取り巻く状況の変化を受け、経営学の分野でもソフトウェア開発を対象とした研究が 1990 年代から始められることとなる。ソフトウェア・エンジニアリングの成果を部分的に受け継ぎつつ、経営学独自の視点を加味した研究は、企業で行われているソフトウェア開発とはどのような活動なのか、それを効果的、効率的なものとし、企業のパフォーマンスを向上させるためにはどのような施策が必要なのかが基本的に問題とされることになる。

その嚆矢となった代表的研究は、Cusumano (1991)である。同書では、ソフトウェア開発で立ち後れていると考えられていた日本企業、すなわち日立、東芝、NEC、富士通といった企業が 1960 年代から 1980 年代にかけて行ったソフトウェア開発とそのマネジメントの革新を中心にソフトウェア開発の事例を調査し、企業における望ましいソフトウェア開発のあり方について考察している[7]。

ソフトウェア開発を工業製品の生産とのアナロジーにおいて論じ、工業製品の生産で有効な工程

---

[5] 以下、本節の前半部分は立本(2002)のサーベイによる部分が大きい。コンピュータの誕生とその発達、ソフトウェア開発およびそれに関する研究などの成立の詳細については、同論文第 2 節、第 3 節を参照されたい。

[6] 初期の代表的な著作としては、Brooks (1975)が挙げられるであろう。IBM の System/360 のソフトウェア開発経験に基づく同書は、体系的に知見をまとめた研究書ではないが、その省察、問題意識は現在でもなお非常に示唆に富む。

[7] この研究で取り上げられた日本企業のソフトウェア開発マネジメントにおいては、1970 年代、1980 年代にソフトウェア・エンジニアリングの分野で積み重ねられた知見、開発ツールの利用があったとされる。それらの知見や開発ツールの詳細に関しては、立本(2002)を参照されたい。

管理や生産管理などの組織的取り組みをソフトウェア開発に取り入れることによってソフトウェア開発は効率化できる、生産性を上げることができるとする論旨に対する反論や、その後の日本企業のソフトウェア開発における総合的パフォーマンス、ソフトウェア部門の業績を考慮すれば、同書に対して全面的に肯定的評価をすることはできないだろう。しかしながら、同書が経営学の立場に立ったソフトウェア開発研究の第 1 歩であり、その後の研究に与えた影響は大きいといえる。

Cusumano はその後もソフトウェア開発に関する研究を続けていくが、その研究対象はコンピュータ産業・ソフトウェア産業自体の変化を踏まえて、常にその中心となっている分野のソフトウェア開発を対象としている。

具体的には、最初の研究である Cusumano (1991)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とを受け、同分野のリーダー企業である Microsoft 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、マイクロソフト社に関する網羅的・体系的な調査に基づいて、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[8]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、マイクロソフト社とネットスケープ社の事例を調査し、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、インターネット用ソフトウェアに関する同様の知見は、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(Iansiti & MacCormack(1996); Iansiti(1998))でも確認されている。こうした諸研究の結果、同期安定化プロセスは、それ以前にソフトウェア・エンジニアリングの分野で支配的であったプロセスモデルであるウォーターフォール・モデル(Waterfall Model)と並ぶ、今日の代表的なソフトウェア開発のプロセスモデルとしての地位を築きつつあるといえるだろう。

このように、ソフトウェア開発に関する研究は、コンピュータ産業、ソフトウェア産業の先進国であるアメリカの研究者によって主に進められてきた。だが、近年では日本企業の事例に基づいた日本の研究者による研究も発表され始めている。

例えば、妹尾(2001)はカスタマイズド・ソフトウェアとパッケージ・ソフトウェアを開発している日本企業の事例に基づいて、主に開発を指揮するリーダーシップの観点からウォーターフォール・モデルの再検討を迫っている。同研究によれば、ソフトウェア開発プロジェクトを取り巻く諸事情の変化、具体的には「開発期間の短期化」と「外部との関係の緊密化」が、プロジェクトとはどうあるべきか(プロジェクト観)、開発する製品はどのようなものか(製品観)、開発に従事する人材はどのような存在か(開発者観)の変化を引き起こしているという。そして、ソフトウェア開発の場合、仕様変更を是とするプロジェクト観、「粘土細工」のように変更が容易であるとする製品観、開発者は代替可能な労働力ではなく創造的作業者であるとする開発者観が整合的であると主張する。

---

[8] 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。

事例に基づくこのような現状認識を踏まえ、妹尾はソフトウェア開発を指揮する人材の役割が、「先に行った自らの行為や、それに対する他社の反応、行為する場の特徴などの、他のリソースも用いて作り出された即興的過程」と見なすことが妥当であるとし、そのような人材の役割を「状況論的リーダーシップ(situated leadership)」と呼ぶことを提唱している。さらに、こうしたリーダーシップに関する見解に基づいて、プロセスモデルの再検討、あるいは、プロセスモデルを開発リソースの 1 つ[9]と見なし、リーダー、リーダーシップにとって従属的なものと位置づけることの必要性を主張している。

また、立本(2002)、立本(2003)では、日本企業が比較的高い競争力を持つとされている家庭用ゲーム機用ソフトウェア(ゲームソフト)の事例に基づいて、ソフトウェア開発プロセスについて考察をしている。この 2 つの論文によれば、ゲームソフト開発においては、ソフトウェア・エンジニアリングの初期から支配的であったウォーターフォール・モデルや、Cusumano らが提唱する同期安定化プロセスとは異なるプロセスモデル、ソフトウェア開発のマネジメントが確認されたという。

まず立本(2002)においては、同期安定化プロセスで確定的に行われている、ソフトウェア開発の初期段階である概要設計の段階において、詳細設計や実験的コーディングをも同時に行い、どのような製品をどのように実現すべきかを検討していると報告している。立本はこれを「探索モデル」と呼ぶことを提唱し、ウォーターフォール・モデルや同期安定化プロセスとは異なるプロセスモデルとして認識する必要があることを主張している。

さらに立本(2003)では、同じ用途であるゲームソフトの中でも、全く新規なソフトウェアの場合と、コンセプト、アイディアやアルゴリズム、データなどを流用できる続編ソフトウェアの場合には、適合的な開発プロセスが異なることも指摘している[10]。この研究では、流用可能な前作の有無、技術的条件の違いによって、個々のゲームソフト開発プロジェクトが直面する「変化量」を推定し、変化量が大きい場合には探索モデルに近いプロセスモデルが、変化量が小さい場合には同期安定化プロセスに近いプロセスモデルが適合的だと主張している。

２．既存のソフトウェア開発研究の問題点

(1) ソフトウェア開発の変遷およびユーザの開発への関与に関する問題点

前節では、コンピュータ産業、ソフトウェア産業の先進国であるアメリカの研究と、日本の研究者によるソフトウェア開発研究を概観してきた。だが、これまで紹介してきた既存のソフトウェア開発研究には暗黙の前提としている 2 つの共通点があると考えられる。その共通点は、以下のようにまとめられる。

第 1 に、ほとんどの既存研究では、商用ソフトウェアのみが研究対象とされてきた。換言すれば、ユーザがソフトウェア開発に携わる機会が少ない、ユーザの開発への関与度が低い事例が多くの場合対象とされてきた。これは、コンピュータ産業、ソフトウェア産業の発展過程と関連を持つ傾向である。

既に述べたようにソフトウェア産業は、それが 1960 年代以降産業として成立し、発展する過程において、まずユーザ(である顧客企業)の要望にしたがってメインフレームなどで動作するシステムを構築するカスタマイズド・ソフトウェアの分野が中心となって立ち上がった。続いて 1980 年代以降、ミニコンピュータ(ミニコン)、パソコンへとダウン・サイジングが進行して、特に 1990 年代以降はそれらの上で動作するパッケージ・ソフトウェアが急成長し、それが同時にコンピュータ・ユーザの急速な増大を招来し、コンピュータに関して詳しい知識を持たないユーザをも取り込むことに繋がった。そのため、パッケージ・ソフトウェアの多くはユーザがほとんどソフトウェアに関する知識、特にその開発に関する知識などを持たないことを前提に開発され、販売されることが多くなった。

このような歴史的変遷をユーザ側から見れば、当初は、コンピュータとその上で動作するソフト

[9] 開発プロセスが選択可能なリソースの 1 つに過ぎないとの見解は、ソフトウェア開発方法論の専門家である Peter Coad の主張とも一致する。Peter Coad の主張については、@IT 記事(2003)を参照されたい。

[10] 同じゲームソフトであっても異なる開発活動が必要とされることは、馬場(1998)でも指摘されている。

ウェアに関し、一定の知識を持ち、開発されるソフトウェアの機能や技術的情報を理解しうるユーザがコンピュータとソフトウェアの利用者であり、彼らはソフトウェアの開発にも一定の関与をしていた。だが、1990 年代以降、ユーザはソフトウェアがそもそも何であるのか、どのようにそれが開発されるのか、どのような技術が盛り込まれているのかを知らずにソフトウェアを購入し、ユーザに残された数少ない作業、すなわち自らのパソコンなどへのインストールと利用、若干のカスタマイズに終始することになった。

そして、Cusumano を代表とする既存研究の多くは、こうしたコンピュータ産業、ソフトウェア産業の変化を十分に踏まえ、常にその中心的な分野に研究対象を絞ってきた。その結果、1960 年代から 1980 年代を対象とした Cusumano (1991)はカスタマイズド・ソフトウェア、1990 年代以降のソフトウェア開発を対象とした Cusumano らの一連の研究や Iansiti らの一連の研究は、パッケージ・ソフトウェアあるいはそれと同等のソフトウェアを研究対象とすることとなった。こうした研究対象の選定の結果、図らずも商用ソフトウェアのみを対象とするものに研究が終始し、ユーザの開発への関与の程度が低い事例に研究対象が絞り込まれることになったのである。

しかしながら、ソフトウェア「産業」ではなくソフトウェア「開発」の全般の状況を踏まえれば、こうした研究対象の偏りは、必ずしも全面的に肯定できるものではない。この章の冒頭で述べたように、コンピュータの誕生に若干遅れてソフトウェアという存在が生まれ、それはその後ソフトウェア自体の製品化、ソフトウェア産業の成立へと繋がってゆく。

だが、当時のリーダー企業であった IBM がソフトウェアのアンバンドリング政策を採り、マイクロソフト社の Bill Gates が「ホビイストへの公開状」を公開し、ソフトウェアに排他的な知的財産権が設定されたと言うことは、逆に言えば、それと反対の状況がそれ以前にあったことを示唆している。それは、ソフトウェアがハードウェアとバンドルされたり、ソフトウェアに排他的な知的財産権を認めず、それが製品と認識されたりすることを拒否する状況が存在していたことである。

この中で、ソフトウェアとハードウェアをバンドル化し両者を一体と認める状況こそ、主に技術的な要因により、現在ではなくなった。だが、ソフトウェアに排他的な知的財産権を認めず、したがって、それが製品となることを拒否する状況は現在もなお生き続けている。この立場を採る人々によれば、ソフトウェアはコンピュータおよびソフトウェア利用者の共有の財産であり、特定の個人や企業が財産権を設定できるものではないとされる。

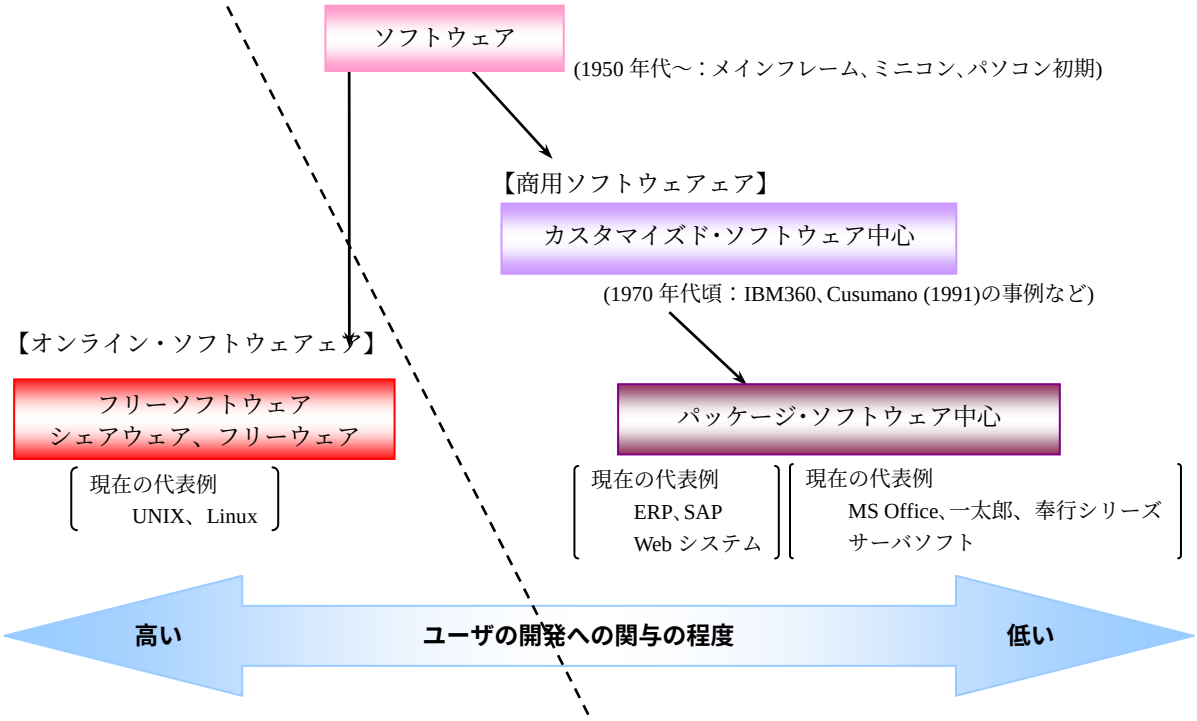
そもそもコンピュータとソフトウェアの成立初期には、ソフトウェアに関するこうした立場の方が優勢であり、コンピュータとソフトウェアの利用者自らがソフトウェアを開発し、利用し、そして自由に配布していた。その後ソフトウェアの製品化、商用ソフトウェアの主流化、産業としての確立の陰に隠れる形で、このようなソフトウェア開発と利用のあり方は相対化され、市場規模などの金銭に換算して評価した場合に把握しにくいことや、企業として実在が無いために、特に経営学の対象からは外されていった。

しかしながら、このような立場に立つソフトウェアの開発、利用、配布は UNIX および UNIX 用ソフトウェア、フリーソフトウェアの開発[11]などで実際に行われ、それは現在、オープンソース・ソフトウェア(Open Source Software; OSS)、Linux の開発などにおいて生き続けている。これは、前述の用語法を使えばユーザの開発への関与の程度が高いソフトウェアが存在するということである。つまり、ソフトウェアに排他的な知的財産権を認めず、それが製品とされることを拒否する立場に立つ人々と彼らによって開発されるソフトウェアは、ソフトウェアの草創期から存在し続けて現在に至っているものの、ソフトウェア産業の確立と巨大化、商用ソフトウェアの主流化に覆い隠されてしまい、既存の経営学のソフトウェア開発研究では対象となってこなかっただけなのである。

以上のソフトウェアの変遷と、そこにおけるユーザの開発への関与の程度の変化をまとめると図 1 のようになろう。

[11] UNIX および Linux の開発に関しては、高橋・高松(2002)が詳述している。

図 1　ソフトウェアの変遷とユーザの開発関与の程度の変化



このようなソフトウェアとその開発へのユーザ関与度の整理に基づいて再論すれば、既存研究はコンピュータ産業、ソフトウェア産業の変遷に忠実に従った結果、上図斜線の右側のみを研究対象としてきたといえる。したがって、上図斜線左側に含まれるソフトウェア、すなわち、商用ソフトウェアでないソフトウェア、ユーザの開発への関与の程度が高い開発事例を研究対象に加えることが必要だと考えられる。

ただし、ごく最近の Linux の成功を受け、Linux の事例を中心としたオープンソース・ソフトウェア(Open Source Software; OSS)[12]の研究も行われるようになってきてはいる。Linux によって刺激された最近の研究の端緒となったのは、Raymond (1997) であろう。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[13]を進めるに当たって効果的、効率的なソフトウェア開発について考察が行われている。

Raymond に依れば、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てのソフトウェア開発、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。この Linux などの開発における効率と効果の高さは、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。そして、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、同時に最初の研究故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

[12] OSS に関しては、第 6 章においてより詳しく取り上げる。
[13] ソースコードを公開したソフトウェアは、それ以前はフリーソフトウェアと呼ばれていたが、現在ではオープンソース・ソフトウェアという呼称の方が広く使われている。そこで本研究ではこれ以降、ソースコードを公開したソフトウェアを、基本的にオープンソース・ソフトウェア(OSS)と統一して呼ぶことにする。なお、オープンソース・ソフトウェアという呼称の誕生、およびその定義については高橋・高松(2002) で紹介されている。

章形式[14]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)に主に見ている点などが挙げられる。

そこで、より緻密な研究として、高橋・高松(2002)が発表された。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、本当に無償のオープンソース・ソフトウェアであったからか、という疑問から出発している。こうした疑問に応えるため、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、無償のオープンソース・ソフトウェア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[15]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、無償のオープンソース・ソフトウェア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、さらに Raymond の提唱した伽藍方式とバザール方式の対比、後者の優越性にも疑問を呈し、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[16]。

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、知見の報告が存在する。Dibona, Stone & Ockman (1999)、g 新部(2003)などが挙げられる。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一貫して、体系的な記述、分析を欠いている点で問題がある。言い換えれば、ここで取り上げた Linux や OSS に関する諸研究は、既存研究との接合性、OSS の中で閉じた問題意識・記述に終始していること、分析の包括性などにおいて十分であるとは言い難い。

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、宮上・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)では、オンライン・ゲームを対象に、ユーザがゲーム世界の中で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。

---

[14] 考察の結果が断章形式を取っているのは、ソフトウェア・エンジニアリングの初期の代表的な著作である Brooks (1975) と共通している。ソフトウェア開発に関する初期の代表的研究、考察が共に断章形式をとっているのは、興味深い一致点はないだろうか。

[15] 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して 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)

[16] この研究でも開発スタイルという言葉を使い、開発のプロセスに関し若干の記述、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、これらの記述が詳細ではなく、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。

(2) 開発プロセスの視点の堅持

既存研究の第 2 の共通点は、ソフトウェア開発を、完結的で一回性の高い「プロセス」あるいはその理念型である「プロセスモデル」のレベルで捉えている点である。より具体的に述べれば、既存研究では､新しいソフトウェア､あるいはソフトウェアの新しいバージョンの 1 つ 1 つについて、その開発活動をコンセプト創造や目標設定などを起点とし、結合テストおよび公開・ユーザへの引き渡しで終了する完結的なものと見なしている。

しかしながら、第 3 章において詳しく述べるように、現在の製品開発研究の潮流を踏まえ、あるいはソフトウェアの技術的特性を考慮すると、上記のように完結的で一回性の高いプロセスやプロセスモデルでのみソフトウェア開発を論じることには限界と問題があるように思われる。したがって、そうした研究と相補的な研究の視点として、プロセスが螺旋的に連鎖[17]、すなわち 1 つのプロセスの遂行、ユーザの使用、ユーザからのフィードバック、新しいプロセスの開始へと活動が連鎖する過程を捉える視点、換言すれば、ソフトウェアが開発されその機能が向上していく過程を包括的に捉える「開発サイクル」の視点が必要であると思われる。

３．既存研究概観のまとめ

以上、ソフトウェア開発に関する既存研究を概観してきた。コンピュータの誕生にやや遅れて始まったソフトウェアの歴史とその産業化の過程を踏まえ､1960 年代から研究が始まったソフトウェア・エンジニアリング、さらにその成果とソフトウェア産業の確立・成長を背景にした経営学のソフトウェア開発研究について述べてきた。ここで、これらの既存研究が有する問題点を再び整理すると以下のようになろう。

まず､既存研究の多くは商用ソフトウェア､ユーザの開発への関与の程度が低い事例を対象とし､それに基づく議論が多い。次に、既存研究がソフトウェア開発を分析する際の視角が、開発プロセス、あるいはその理念型であるプロセスモデルに終始している。だが、第 3 章で詳述するように、近年の製品開発研究の知見、ソフトウェアの技術的特性を踏まえれば、プロセスがユーザによる使用、ユーザからのフィードバックを受け入れて新たなプロセスに繋がるという、開発サイクルの視点が必要とされると考えられる。

そこで本研究では、シェアウェア、フリーウェアというユーザの開発への関与の程度が高い事例を記述し、それを開発サイクルという分析視角で分析し、既存のソフトウェア開発研究の知見と比較することにする。

[17] プロセスの螺旋的連鎖は、プロセス内部での活動の螺旋的連鎖とは異なる。プロセス内部での活動の螺旋的連鎖は、Boehm (1988) が「スパイラル・モデル」として提唱しているが、それはあくまで、1 つのソフトウェア開発プロセスの内部において、目標設定、概要設計、詳細設計、コーディング、結合テストなどの活動を繰り返し行うことである。

第 3 章　オンライン・ソフトウェア調査

1．調査対象

一口にシェアウェア、フリーウェアと言っても、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、この研究ではシェアウェア、フリーウェアに関する研究の第 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、調査を進めていくことにする。

具体的な事例としてメールソフトを取り上げるのには、次のような理由がある。まず第 1 に、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、メールソフトは Web ブラウザに次いで利用頻度が高いソフトになっている。しかも、そのユーザは、コンピュータに触れ始めたばかりの初心者から、コンピュータを日常的に使う上級者まで、幅広い。こうしたコンピュータ・ユーザにとっての利用頻度の高さ、ユーザの多様性、多さもまたメールソフトを研究の対象とする理由である。

第 2 に、メールソフトは日常的に利用するソフトウェアであるので、ユーザが自分の使いやすいようにカスタマイズしたり、独自の機能を追加したりすることが多く、実際にそうした付加機能がプラグイン(plug-in)として提供されているからである。このため、開発可能な対象(機能)が広範におよび、また、それに応じた開発者(群)と開発者(群)の間の連携が図られる可能性のあるメールソフトを研究対象とすることは、今後のより広範なオンライン・ソフトウェアを対象とした調査・研究に資する点が多いと考えられる。

第 3 に、ほぼ同等の機能を持つ商用ソフトウェアが存在する事が挙げられる。具体的には、Microsoft 社の Outlook Express、ジャストシステム社の Shuriken Pro などが商用のメールソフトとして存在している。本研究では、これら商用のメールソフトの事例を調査するには至らなかったが、今後のより詳細な事例比較をするにあたり、そうした比較対象のソフトウェアが存在することは、重要であると考えられる。

第 4 に、メールソフト本体ならびにプラグインが、かなり頻繁に新規開発あるいはバージョン・アップされており、総体としてのメールソフトの機能が非常に速いペースで向上しているからである。このような頻繁なバージョン・アップとそれにともなう機能の向上は、前述の Cusumano & Selby (1995)が発見した Microsoft 社の同期安定化プロセスや、Cusumano & Yoffie (1998)、Iansiti & MacCormack (1996)、Iansiti (1998)などで明らかにされた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、「先進的」とされたソフトウェア開発のあり方とそのパフォーマンスに通じるものがある。したがって、この点からもこれらメールソフトを研究対象とすることは意義が高いと考えられる。

2．ケースの選択と調査方法

このような理由から、本研究ではメールソフトを調査対象とすることにした。だが、メールソフトにも多数のサンプルがある。そこで本研究では、代表的なシェアウェア、フリーウェアの配布サイトである窓の杜(http://www.forest.impress.co.jp/)において紹介されているメールソフト、あるいは Vector (http://www.vector.co.jp/)において専用のカテゴリーが設けられているメールソフトにまず対象を絞った。代表的なシェアウェア、フリーウェア配布サイトで紹介、あるいは専用カテゴリーが設けられているということは、そのメールソフトが広く知られており、使用されていることの証左である。換言すれば、そうしたメールソフトは、「メールソフトの成功事例」と考えられるからである。

ただし、このように対象を絞り込んでもなお、シェアウェア、フリーウェアとして公開されているメールソフトは 10 本近くになる。そこで、現在でも開発が継続されているものにさらに対象を絞り込み、シェアウェアとフリーウェアを各 1 本ずつ研究対象とすることにした[18]。具体的には、

[18] 今回の調査対象とならなかった代表的なメールソフトは、「Becky! Internet Mail」「AL-Mail」「Ed-Max」である。これらのソフトウェアに関しての調査、分析は今後の課題であると考えている。

シェアウェアの代表的事例として「鶴亀メール」、フリーウェアの代表的事例として「電信八号」を研究対象とした。

これらの研究対象に対する調査は、インタビュー調査を主体とし、それを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 月から 11 月にかけて行われ、各々のインタビュー時間は 3 時間から 6 時間程度である。調査に当たっては、事前に質問票を送付し、それに沿ってインタビューを行った。

調査項目は、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、開発組織についてなどである。

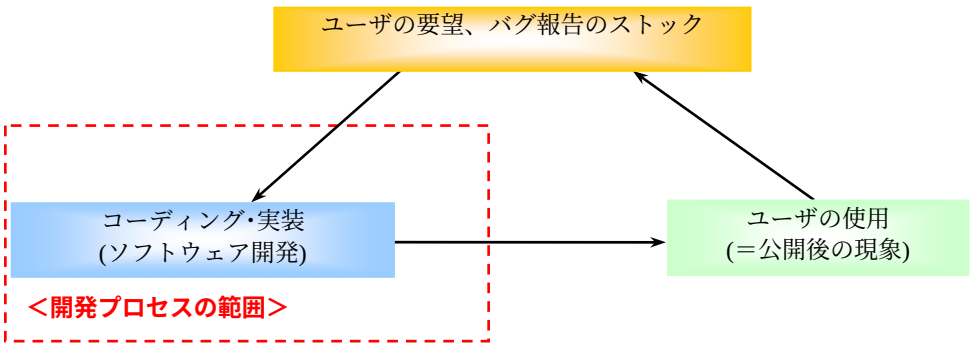
(1) 分析フレームワーク「開発サイクル」

このようにして収拾した事例を分析するに当たり、本研究が分析フレームワークとして採用するのは、「開発サイクル」という分析視角である。以下、この分析視角の詳細とそれを採用した理由について述べる。

開発サイクルとは、1 つの開発プロセスの遂行、ユーザによる使用、ユーザからのフィードバックのストック、新しい(バージョンの)ソフトウェアのコーディング・実装、によって形作られる循環的かつ螺旋的運動である。

循環的かつ螺旋的運動であるとは、開発プロセスの遂行、ユーザによる使用、ユーザからのフィードバックが繰り返し行われること(循環的)、同時に、開発サイクルの出発点である最初の開発プロセスは、それがユーザの使用、ユーザからのフィードバックを経た後の 2 回目の開発プロセスとは異なり、多くの場合、2 回目の開発プロセスの方がその成果であるソフトウェアの質が向上していること(螺旋的)、を指している。

図 2　開発サイクルの基本的概念図



この開発プロセスの循環性を中心に図示すると図 2 のようになる。上記のような定義である開発サイクルは、図にも表されているように開発プロセスを包含し、また、それと相補的な関係にある。

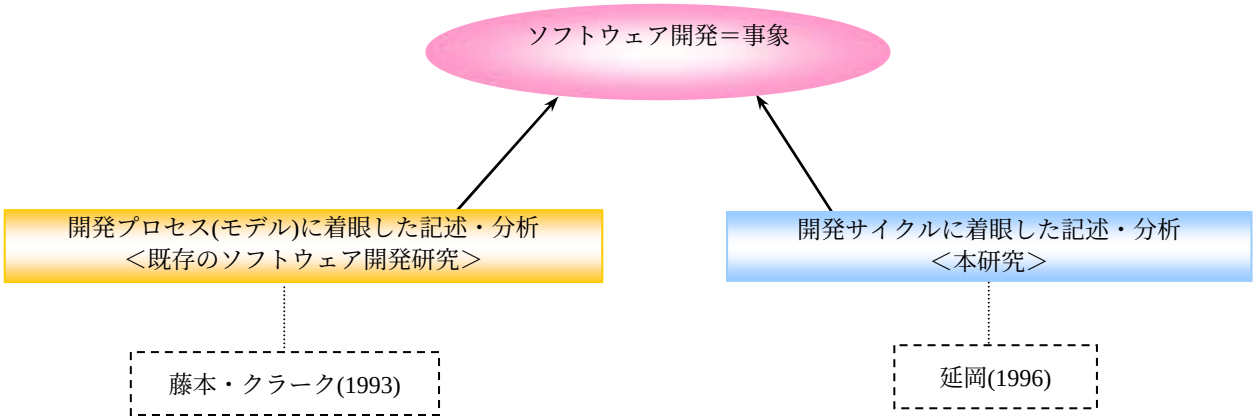
開発サイクルの視角と開発プロセスの視角が相補的であると言うことは、両者が同じ対象―この場合はソフトウェア開発活動―を記述、分析する上で、各々が異なる記述、分析をしながらも、それらが矛盾せず、むしろ他方の記述や分析の不足点を補いうるということを意味している。

本研究が目指しているように、同一対象を異なる視角から分析し、各々が優れた知見を持ち得た研究の例は、自動車産業の製品開発研究において見られる。それは、藤本・クラーク(1993) と延岡(1996) である。これらの研究では、まず藤本・クラーク(1993) が自動車の開発プロセスを詳細に分析し、優れた知見を得た。それを踏まえて、延岡(1996) では複数の開発プロセス(開発プロジェクト)間の関係に焦点を当て、その結果、藤本・クラーク(1993) とは異なるが、やはり優れた知見を得ることに成功している。

本研究およびそれが既存のソフトウェア開発研究との間で構築しようと目指す関係も、この 2 つの自動車産業の製品開発研究と同様の関係である。前章で述べたように、既存のソフトウェア開発研究においては、藤本・クラーク(1993) に相当する開発プロセスの記述、分析を目指した研究が一定数あるため、それを補い、それとは異なる知見を得るために、本研究は異なる分析視角である開発サイクルを提唱、採用しようと考えたのである。よって、本研究を自動車産業での研究とのアナロジーで位置づければ、延岡(1996) のようなポジションを狙っていると言えよう。

このような本研究の既存研究に対するポジションを、自動車産業の製品開発研究の結果を例に図示すると、下図のようになる。

図 3　分析視角としてのプロセスモデルと開発サイクルの関係



つまり、ソフトウェア開発という同一の事象に対し、これまでのプロセス(モデル)に基づく研究、議論とは異なる分析視角として、開発サイクルの分析視角を明確化し、それを実際の事例分析に適用することで、分析視角の有用性と新たな知見の導出を目指すのが、本研究の立場である。

(2) 開発サイクルという分析視角を採用する理由

では、なぜ本研究では開発サイクルという新たな分析視角を採用するのか。その理由は、大きく 2 つ挙げられる。その第 1 は、近年における製品開発論全体の潮流の影響である。前項で挙げた延岡(1996) に刺激を受ける形で、製品開発論では個々の開発プロセスや開発プロジェクトにのみ焦点を当てた研究に対する再検討と、それを踏まえた実証研究が行われるようになってきている。

具体的には、青島(1997) が、藤本・クラーク(1993) に代表される個別の開発プロセス、開発プロジェクトに焦点を当てた研究を、「物理的な構成要素となる単層システム」として対象(製品)を位置づけ、かつその開発活動を「自己完結的な問題解決プロセス」として研究してきた、と相対化している。その上で青島は、「複数のシステムからなる多重システム」として対象を位置づけ、かつその開発活動を「(製品)システムを理解するための継続的探索プロセス」として研究する必要性が生じてきていると主張するのである。そして、青島が主張する後者の立場に立った実証研究として、前述の延岡(1996) や青島・延岡(1997) などが発表され、従来の個別の開発プロセス、開発プロジェクトに焦点を当てた研究とは異なる有用な知見を得ている。

そこで本研究は、青島(1997) の主張を取り入れ、またソフトウェア開発に関する研究が個別の開発プロセス(プロセスモデル)、開発プロジェクトを対象としてきたという事実を踏まえて、開発活動をより継続的、循環的な開発サイクルという視角で記述、分析することを試みることにした。具体的には、事例に基づいてシェアウェア、フリーウェアの開発サイクルを記述し、それを既存研究から推察される商用ソフトウェアの開発サイクルと比較して、両者の相違点を対照的に描き出すと共に、その背後にある要因について考察を進める。

本研究が、開発サイクルという分析視角を採用する第 2 の理由は、ソフトウェアの技術的特性に求められる。ソフトウェアがハードウェア製品に比べ変化が容易である(「可変性」)、開発済みのソフトウェアの一部もしくは大部分を、新規ソフトウェアおよびその開発に利用することができる(「再利用性」)ことは Brooks (1975) など多くの既存研究において指摘されているソフトウェアの技術的特性である。このような技術的特性は、ソフトウェアが世に出る際に、全く新規なソフトウェア(製品)という形態を取ることより、既存のソフトウェア(製品)のバージョン・アップという形態を取っている現実とも整合的である。

こうした点を踏まえれば、ソフトウェア開発という活動を、一回性の高い、完結的なプロセスとして記述、分析するよりも、それが公開されて、ユーザに使用され、その使用体験が開発主体にフィードバックされて、新たなソフトウェア(製品)あるいは既存のソフトウェア(製品)のバージョン・アップに繋がるという開発サイクルという分析視角の方が有用性を持ちうる可能性が高いと考えられる。

３．事例研究 1―シェアウェア「鶴亀メール」の事例―

本節と次節では、第 2 節で紹介したように、シェアウェアの事例として「鶴亀メール」の開発事例、フリーウェアの事例として「電信八号」の開発事例を紹介する。

(1) 「鶴亀メール」の概要

サイトー企画と「鶴亀メール」

「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、今や Windows 用エディタの定番とも言うべき「秀丸エディタ」のコンポーネントが利用されている。ソフトウェア自体の機能の充実度もさることながら、ユーザへのサポートが非常に丁寧で親切なところが特筆すべき点である。

鶴亀メールおよびそれと密接に関連する秀丸エディタの開発・配布は、有限会社サイトー企画によって有償で行われている。ただし、サイトー企画は会社組織になってはいるものの、ソフトウェアの開発をチームで行うことはない。会社形態をとっているのは、ソフトウェアの配布やライセンシング契約の便宜のためという側面が強い。斉藤氏によれば「有限会社になっていることに深い意味はない。強いて言えば、個人経営に比べ、税金などの申告や各種保険・年金の手続きが容易である、あるいは、他の会社や学校などとライセンス契約を結ぶ際に、その契約相手として企業形態をとっていることが必要な場合もあるためである」とのことである。

サイトー企画では、前述の秀丸エディタを代表としたエディタ類、このケースが対象とする鶴亀メールを初めとするインターネット用ソフトウェア、「秀 Term」などの通信ソフトを中心にセキュリティ関連のソフトウェア、ユーティリティ・ソフト、開発ツールなど、計 31 タイトルのソフトウェアを開発し、提供している。また、自社のサーバを活用したネットワーク・インフラ「コミュニテックス(旧「秀ネット」)」とソリューションの提供も行っている。

これらのソフトウェア群の中でサイトー企画の中心となっているのは、秀丸エディタ、鶴亀メール、秀 Term である。各々のソフトウェアの登録ユーザ数、すなわちシェアウェア代金納入者は、秀丸エディタで 15 万人程度、鶴亀メールで 1000 人程度、秀 Term で 9 万人程度である。ただし、秀丸エディタのコア・コンポーネントが鶴亀メールには内蔵されているので、秀丸エディタの登録ユーザは鶴亀メールを無料で使用することができるようになっている。このため鶴亀メールの利用者数は、実際の登録者数よりもずっと多いと考えられる。

登録ユーザ数からも分かるように、サイトー企画において、最も中心的な位置づけを占めているのは「秀シリーズ」のソフトウェアである。同社の収入の大半は秀シリーズによっており、鶴亀メールも秀丸エディタの付随的ソフトとして位置づけられている。

同社の今後の見通しに関しては、斉藤氏は「自分自身はやる気はない。新規事業や新規ソフトウェアの開発を立ち上げる計画はなく、現行のソフトウェアのバージョン・アップを続けることのみが関心事項である。会社自体は、シェアウェアの収入であと数年運営できる見込みである」と述べている。

鶴亀メールの詳細

鶴亀メールは、内蔵されている秀丸エディタの部分を除いた本体プログラムの行数が 117,582 行である[19]。したがって、秀丸エディタの行数 91,795 行を加えると、鶴亀メールの本体プログラムの総行数は、209,377 行となる(いずれも 2002 年 12 月 13 日時点での数値)[20]。

この鶴亀メールにユーザが独自の機能を追加するためには、マクロを作成する必要がある。というのも、サイトー企画・斉藤氏は基本的に開発者向けキット(SDK; Standard Development Kit)やソフトウェアのインターフェースを公開していないためである。

---

[19] これは、*.h と*.cpp ファイル全体の行数のみをカウントした結果である。インストーラや付属ソフトウェアのファイルは含まれていない。

[20] ちなみに、サイトー企画のもう 1 つの代表的ソフトウェア「秀 Term Evolution」の総行数は 71,247 行である。

しかし、こうした状況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、表 2 に示すよう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 2　鶴亀メール用マクロの代表例

用途	名称	作者	備考
印刷補助	PrintMessenger Ver.02.10.16 ―― 秀鶴印刷補助	slide_moon	他 1 点
カスタマイズ	鶴亀アタッチ	KAZZ	他 14 点
検索	固定条件検索マクロ　Ver 1.04	hi_sugar	
セキュリティ	鶴亀メール用簡易暗号化マクロ	山紫水明	
添付ファイル操作	添付ファイル圧縮マクロ（受信系/送信系兼用）	秀まるお	他 4 点
テンプレート	複数のひな形(テンプレート)、定型文、署名を使い分けるマクロ Ver1.08	slide_moon	他 2 点
入力補助	鶴亀メール作成支援マクロ Ver.1.32	ひろ	他 10 点
フィルタ	鶴亀メール携帯電話向け分割送信マクロ Ver 1.06	hi_sugar	他 9 点
編集	テンプレート展開付きコピー機能(裏技シリーズ) v1.01	ダヴィンチ	他 12 点
メール操作	スレッド抽出マクロ	かわした	他 7 点
その他	メール送信支援マクロ集	KAZZ	他 15 点

※　2003 年 6 月 28 日現在。公式サイトに掲載されているマクロを集計したもの。用途の分類は筆者による。

また、マクロの中には秀丸エディタや鶴亀メール本体の機能を呼び出すものもあるので、マクロを介してユーザ独自のライブラリ、すなわち C 言語で記述されたプログラムやプラグインに相当するものを機能させることは可能である。

また、後述するようにユーザの要望に応えたバージョン・アップも頻繁に行われている。その結果として、鶴亀メールは非常に多機能で(「オプションが多い／多すぎる」)、カスタマイズ度が高いメールソフトとなっている。

ソフトウェア開発環境

サイトー企画のソフトウェア開発環境は、次のとおりである。まず、開発言語にはビジュアル C を使用している。そして、機材としては 10 台ほどのコンピュータを使用しており、その内訳は、開発者の作業用コンピュータが各 1 台で計 3 台、会計処理用のコンピュータが 1 台、その他に動作テストやバックアップのためのコンピュータが 6 台ほどであるという。テスト用のコンピュータは、SOHO 用サーバ、バックアップ用途をかねているものもあり、WindowsNT 3.51、WindowsNT 4.0、Windows95 の環境が用意されているという。また、テスト用コンピュータの中には、英語版の秀丸エディタのために英語版 OS をインストールされたものや、アルファ系のプロセッサを搭載した非常に旧式のハードウェアも含まれている。

(2) 開発活動の開始・継続

ソフトウェア開発開始の契機とサイトー企画の設立

ソフトウェアの開発に携わるようになった契機について、斉藤氏は次のように述べている。ソフトウェア開発というものは、パソコン関連書籍の付録プログラムなどを自分で使用してみて、楽しいと感じることができたときに、そうしたプログラミング・リストの入力などから入っていくものではないか。そしてその延長線上で、自分が欲しかったり実現したかったりする機能を盛り込んだソフトウェアを、一から作るようになるものだという。しかも、こうしたソフトウェア開発の「とっかかり」は、趣味でギターを弾くこととなどと同様、自己満足のためであるとも言っている。

斉藤氏の場合、こうした「とっかかり」を 1980 年代初めに経験している。当時のホビー・マシンであった NEC の PC-6001 を使い、1982 年創刊のベーシック・マガジンなどの雑誌を通じてプログ

ラミングの仕方を覚えていき、ソフトウェア開発を行うようになった。

こうしてソフトウェア開発に対する関心をもった斉藤氏は、中学卒業後は高等専門学校に進学し、1988 年に地元の富士通に入社した。富士通ではソフトウェア開発の仕事をできると期待していた斉藤氏だが、実際にはプログラミングをする機会には恵まれなかった。そこで、斉藤氏は業務外で独自にソフトウェアを開発するようになった。

当時、斉藤氏が開発していたのは MS-DOS 用あるいは OS/2 用のソフトウェアであり、種類としては通信ソフトとエディタであった。これらを社内で同僚たちに配布したところ、このころは OS/2 用ソフトウェアが少なかったこともあり、業務には使えないにもかかわらず、珍しがられ、喜ばれたという。

このように斉藤氏は、仕事の合間を縫って独自ソフトウェアの開発に取り組んでいたのだが、1990 年に転機が訪れることになる。Windows3.0 の発売と DOS/V の登場であった。これらの OS (Operating System)が日本で使用できるようになったことにより、コンピュータ、とくにパソコンのグラフィック性能は格段に向上した。斉藤氏はこの変化を「ビジネス・チャンス」と認識し、すでに他の OS 用に開発してあったエディタや通信ソフトを Windows3.0 環境に移植することを始めた。

さらに同じころ、日本のパソコン通信の草分け的存在である Nifty-Serve がシェアウェア代金の徴収代行サービスを提供し、シェアウェアがある程度まではビジネスとして成り立つ環境が調いつつあった。この事実を踏まえて、斉藤氏は自らの会社であるサイトー企画を設立し、同社を通じて開発済みのソフトウェアをシェアウェアとして公開していった。具体的には、1992 年に通信ソフトの秀 Term を、1993 年には秀丸エディタを公開した。これらのバージョン・アップ、あるいは鶴亀メールなどの付随的なソフトウェアの開発・公開を続けて現在に至っている。

なお、冒頭で述べたように、秀丸エディタは今でこそ Windows 用エディタの代表的なソフトウェアとして定着しているが、秀シリーズのソフトウェアの普及やサイトー企画の立ち上げにおいて、Nifty-Serve の役割は決して小さくなかった。小規模な企業にとっては負担のかかるシェアウェア代金徴収の手間と費用が Nifty-Serve によって軽減されたことはもとより、パソコン通信が提供したサポート・フォーラムの活用、あるいは Nifty-Serve 自体がユーザ獲得のために無料配布していた CD-ROM に秀丸エディタが収録されていたことなどは、秀シリーズの普及、サイトー企画の立ち上げにとって大きなプラス材料だったと言える。

鶴亀メールの開発経緯

このように、秀丸エディタを中心に立ち上げられたサイトー企画であったが、斉藤氏と同社がメールソフトの開発を手がけるようになったのは他社の関与がきっかけであった。

プロバイダ事業を行っていた Xaxon (ザクソン)社が、自社でメールソフト「NetMail」を開発・販売したいので、そのモジュールとして秀丸エディタを提供して欲しいと申し入れてきた。この当時は、マイクロソフト社の Internet Explorer のバージョンが 2.0、Outlook は存在しない状況であったため、プロバイダの独自開発やシェアウェア、フリーウェアによってメールソフトが提供される必要があったのである。サイトー企画ではこの申し入れを受け入れ、メール本文を編集する機能を担うモジュールとして秀丸エディタを提供することに同意した。

しなしながら、その後 Internet Explorer がバージョン・アップされ、Outlook も Windows に同梱されるようになったため、状況が大きく変わる。NetMail の開発は継続され、発売された[21]ものの、その先行きは当初考えられていたほど思わしくなかった。

こうした状況の変化に対し、Xaxon 社と NetMail の動向を見ていたユーザ・グループの 1 つが Xaxon 社の態度変化に気づき、「どうも NetMail がおかしい」と感じ始めた。そこで彼らはモジュールとして秀丸エディタを提供していたサイトー企画に、「サイトー企画さんで(メールソフト)を何とかできないか」「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」との要望を寄せた。

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

[21] NetMail の詳細に関しては、「NetMail Unofficial Site」に情報がある。
http://www.ops.dti.ne.jp/~shinya-m/nm/index.html を参照されたい。

ルソフトの開発を開始し、同年 8 月に Windows 用メールソフトである鶴亀メールを公開したのである。

(3) 開発活動の詳細

ソフトウェアのバージョン・アップについて

鶴亀メールのバージョン・アップは、基本的にユーザからの機能追加の要望から始まるという。機能追加のうち、マクロによって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、マクロで対応不可能なものに関してはバージョン・アップで応じるという。

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。

こうしたユーザから寄せられる機能追加などの要望には、他の有名なソフトウェアを参考にしたものも多いという。そのため、ユーザに促される形で斉藤氏は有名なソフトウェア、特にエディタやメールソフトにはひととおり目を通すことになり、それが鶴亀メールおよびそれが内蔵する秀丸エディタの開発に活かされることに繋がっている。ただし、マクロ開発やバージョン・アップを行っても、対応できない機能追加も当然ありうる。その場合には「鶴亀メールでは無理です」とユーザに説明して、その機能追加を断ることもあるという。

バージョン・アップに際しては、鶴亀メールの場合、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・フィックス、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、結果的に表 3 のように、非常に頻繁なバージョン・アップが行われることとなっている。

表 3　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。

ユーザとの接点とその役割

前述のように、鶴亀メールのバージョン・アップに際しては、ユーザから寄せられる要望がその端緒となっている。そうした機能追加・バグ報告を含めたユーザからの意見・要望が寄せられるのが、サイトー企画が運営するコミュニテックスに設置されたサポート・フォーラムである。

このフォーラムで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 (http://hide.maruo.co.jp/)」から簡単に行くことができるようになっている。

斉藤氏は、基本的に「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」と考えている。言い換えれば、「質問者をいじめるようなサポートではダメだ」ということである。この背後には、「通常、企業のサポート・センターに質問して、企業から文句を言われたり、質問に答えてもらえなかったりする事はない。したがって、シェアウェアのサポートも同様に行うべきだ」という考えがある。

この他に、ユーザがサイトー企画にコンタクトする手段としては電子メールと電話があるが、これらの手段によって寄せられるソフトウェアに関するユーザからの要望・意見はそれほどないという。ただし、中にはこれらのコンタクト手段でソフトウェアに関する要望・意見や質問を寄せてく

るユーザも存在するので、その場合には電子メールで返信したり、それに該当するサポート会議室を紹介したりするなどして対応している。
電子メールや電話による問い合わせは、主にシェアウェアのライセンス登録に関するものが多く、その場合でもサイトー企画ではできるだけ丁寧に応えるようにしているという。例えば、「レジスト・キーの入力の仕方が分からない」といった非常に初歩的な質問が電話で寄せられた場合にも、丁寧に答えるようにしているという。
ただし、ユーザやマクロ開発者との接点は、ホームページ、フォーラム、電子メール、電話などに限定されており、オフラインでの交流、接触はないとのことである。

(4) 開発組織

現在のサイトー企画は 3 名の開発者を抱えているが、設立当初は斉藤氏が 1 人で開発を行っていた。1992 年のサイトー企画設立当初からの数年間は、秀丸エディタと秀 Term のみが開発対象であり、斉藤氏 1 人でも開発やサポートをカバーすることができた。
しかし、1995 年に OS が 16bit ベースの Windows3.1 から 32bit ベースの Windows95 に移行し、それに対応して同社のソフトウェアを移植する作業が必要となった。そのため、ソフトウェア開発の仕事が非常に忙しくなり、新しい開発者として山本氏が入社して秀丸エディタを担当することになった。
その後 1998 年ごろ、サイトー企画の業務拡張のために求人広告を出したところ、KON 氏が応募してきた。KON 氏はビジュアル・ベーシックでのプログラミングができるということが分かったので、当初予定していたサイトー企画宛の電子メールへの返信作業だけではなく、秀 Term のアドイン・ソフトや、サーバの立ち上げ、サイトー企画のホームページの管理、サイトー企画が運営するネットワーク・サービスのコミュニテックスなどを担当することになった。
こうした経緯から、現在は斉藤氏が鶴亀メールを中心にしつつコミュニテックス運営の一部、他の小規模なソフトウェアの開発を、山本氏が秀丸エディタの開発を、KON 氏がコミュニテックスの運営を担当する体制となっている[22]。

(5) その他

ソフトウェア開発における開始のインセンティブ

斉藤氏によれば、一般に、独自ソフトウェアの開発に取り組もうとするのは、ソフトウェアを開発できる能力と環境がありながら、その能力を発揮できない、開発をさせてもらえないからであるという。すなわち、能力・環境と機会の不一致がフリーウェアやシェアウェア、場合によってはコンピュータ・ウィルス作成に繋がるのだという。
これを斉藤氏の場合に当てはめてみると、氏が入社した当時の富士通はバブルの真っ只中にあり、社内ではソフトウェアの仕様作成やテストのみが行われ、実際のプログラミングは外部に委託されていた[23]。斉藤氏にしてみれば、「自分はプログラミングがしたくてこの会社に入ったのに…」という思いが生じ、自分でプログラミングをしたいという欲求が強まったのだという。
このような欲求に突き動かされて、斉藤氏は会社の仕事は最低限に留め、それ以外の時間を独自ソフトウェアの開発に当てることにした。この背景には、当時の職場が残業をほとんどしなくても許され、割り当てられた業務以外のことをしていても黙認される環境だったことがある。

ソフトウェア開発継続のモチベーション

斉藤氏によれば、ソフトウェア開発に「やる気があった」のは、学生時代と秀丸エディタ開発の初期のみであったという。対照的に現在は、ソフトウェア開発継続のモチベーションについて、「基本的にやる気はない」「バージョン・アップのみなのでやる気があるとは言えないのではないか」と

[22] なお、3 人はいずれも 20 代半ばから 30 代半ばの年齢である。
[23] 当時の富士通について、斉藤氏はこうも述べている。「当時の富士通にはソフトウェア開発をマネジメントする人材がいなかった。とりあえず人材を集めて、ソフトウェア開発ができる人とできない人を適当にばらまいていた。そのため、部門全体としてのレベルが上がらず、ソフトウェア部門が弱くなってしまった」

述べている。同時に、こうした「やる気のなさ」を反映して、シェアウェア開発者が通常行っているソフトウェアのマーケティング・リサーチなどはまったく行っていないという。

それにもかかわらず開発を終了しないのは、収入があるからだという。つまり、「収入があるから開発を続けている」ということだ。こうしたことから、斉藤氏にとっては金銭報酬が動機づけ要因にはなっていないということがわかる。すなわち金銭報酬は、ソフトウェアの開発を辞めないという参加(退出)の意思決定に関わるインセンティブ要因にすぎないのである。

それでは、開発活動の継続を支える動機づけ要因として、金銭報酬以外の報酬は存在しないのであろうか。

斉藤氏は、昔は雑誌に自分の作ったソフトウェアの特集記事などが掲載されたりすると「有名になってしまうのではないか」「変に名前が売れても困るなぁ」と考え断っていたが、現在は雑誌に掲載されてもそれほど有名にならないことが分かったので、さらにそうした要請を断るのにも「エネルギーがいる」ため断らなくなっていると述べている。つまり斉藤氏は、地位や名誉といった言語報酬を獲得したいなどとは考えていないのである。こうしたことから、外発的な動機づけ要因とされる報酬一般は、斉藤氏の動機づけに対してはまったく影響を及ぼしていないと考えられる。

一方、内発的な動機づけについては、それが斉藤氏の開発活動の根底に存在していることを示唆する発言があった。例えば、「自分が意図した機能が実現されたときは『うまくできた』という達成感がある」という発言は、有能さの感覚が動機づけ要因として働いていることを示していると理解することができる。

また、「ファン・サイトを作ってくれているユーザやマクロ制作者などの期待に応えなければ」という思いがあり、それが開発継続のモチベーションにつながっていることは斉藤氏自身も認めていた。このように、自ら進んで責任を負うことの背後には、自分の仕事には社会的な価値があり、社会的に評価されていると感じていることがあると考えられる。つまり、仕事への「誇り」が有効な動機づけ要因となっている可能性があるのである(藤田(2000); 藤田(2002))。

仕事・会社に対する「誇り」と動機づけ

藤田(2000)、藤田(2002)によって、仕事や会社に対する「誇り」は動機付け要因になることが明らかにされている。そこで、斉藤氏においてどのような事柄が「誇り」を醸成し、ソフトウェア開発の動機付け要因になっているのかについて、より詳細な回答を得ることにした。

まず、自分の仕事や会社に対してどのように感じているかをより具体的に質問してみた。すると斉藤氏は、「自分の仕事に誇りを持っているというような、とくにそういった意識はない」と述べている。ところが、自分の仕事にそれなりの社会的価値があること、それが社会的にそれなりに評価されていること、「使っている人がいるわけだから」開発したソフトウェアが社会的に役立っていることは認めている。こうした質問に対して、斉藤氏は必ず「それなりに」「それなりの」という言葉を用いていたが、この点は、自分の仕事がどのような人にも普遍的に価値があり評価されているわけではなく、少なくとも自分の作ったソフトウェアの利用者間ではそうであるという意味だと思われる。こうしたことから、斉藤氏は自分の仕事に「誇り」を持っており、それが仕事に対する動機づけの基盤となっていると考えられる。

では、会社への「誇り」についてはどうであろうか。斉藤氏は「サイトー企画の社長であることに誇りを持っているというような意識はとくにない」と述べている。ただし「仕事への誇り」と同様、サイトー企画という会社に関しても、それが社会的に価値を持っていること、社会的な評価を受けていることは認めている。

だが、サイトー企画が同業他社の間でも評判のよい会社であるか否かについては、「同業他社とのつきあいがないのでまったく分からない」としている。また、仕事および会社に関する好悪の感情については、仕事に関しては「好きな仕事もあるが、嫌いにも関わらず我慢している仕事も多い。どちらかというと後者の方が多い」と述べている。また会社に関しては、「特に嫌いではないが、特に好きでもない。普通」としている。

４．事例研究 2―フリーウェア「電信八号」の事例―

(1) 「電信八号」の概要

電八倶楽部・電八開発倶楽部と「電信八号」

「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; http://www.iajapan.org/) が主催した、第 5 回フリーソフトウェア大賞(FSP'96)において入賞を果たしている。

その特徴は、電子メール・クライアントとして必要十分な機能を備えつつも、プログラム・サイズが非常に小さく、起動・動作が軽快な点にある。加えて、メールを編集するエディタの機能、メールを閲覧するビューワの機能を内蔵せず、ユーザが好きなエディタ・ビューワを利用できる点も、人気を博している理由の 1 つである。

現在の開発主体である電八倶楽部、電八開発倶楽部とも、「誰でも入れて、いつでも抜けられる」メーリング・リストで、それぞれの参加者数は電八倶楽部で 1000 名程度、電八開発倶楽部で 300 名程度である。両者はそれぞれ独立したメーリング・リストで、その違いは、電八開発倶楽部の方が開発活動により関わりが深いことである。安定していないバージョンである α 版の配布、開発に関する技術的な情報の交換などは主に電八開発倶楽部で行われている。

電信八号(V32.1.3.1)の延べダウンロード数は、初期バージョン(「無印」)が 58,642 件[24]、別バージョンのフル・パッケージ(V32.1.3.1 ; Sp2)が 123,727 件[25]である。

電信八号の配布形態

電信八号は、開発当初からフリーウェア[26]であり、現在ではフリーウェアであることに加えて、条件付きながらソースコードが公開されている。

この配布形態に関し、現在の電信八号開発における中心メンバーの 1 人である中村賢一郎氏は次のように述べている[27]。

> アメリカでは、原作者による著作権放棄が可能なため、ソフトウェアを Public Domain において自由にダウンロードするようにすることができる。ただし、こうした Public Domain Software をもとに製品化し利益を上げたとしても、それを Public Domain に還元する義務はなく、そのソースコードを秘匿しても法律上問題がない。これではソースコードを公開し、自由にそれを改変し、再配布してもらいたいという意図を作者が持った場合に不都合であるため、著作権(Copyright)を放棄せず、むしろそれを利用するかたち(Copyleft)で、改変済みのソースコード公開を義務づけるライセンス形態として GPL (GNU General Public License) が存在する。だが、日本の著作権や頒布権の下では、原作者による著作権放棄が不可能なため Public Domain Software 化もできないし、GPL を厳密に履行させるだけの法的根拠も存在しない。そのため、日本の著作権法、および万国著作権法に基づき、さらには日本の歴史的、文化的要素を反映して、フリーウェア[28]やシェアウェア[29]として公開・配布されるソフトウェアが出てきたと考えられる[30]。

---

[24] 2000 年 12 月 24 日以降の数値、2003 年 3 月 8 日現在。

[25] 2001 年 12 月 12 日以降の数値、2003 年 3 月 8 日現在。

[26] 石岡氏は「当初、若干の『金儲け』の意図もあったが、思ったよりシェアが取れず、後発の『Becky!』が急速にシェアを伸ばしたため『金儲け』を断念し、フリーウェアであり続けることにした」とも述べていた。

[27] 中村氏の以下の発言は、「『厳しすぎる』との批判がある電信八号のライセンス形態は、日本の著作権法、文化的・歴史的背景を考慮すれば、妥当なものである」という趣旨にもとづいている。

[28] フリーウェアに関する原作者の石岡氏の意見はシンプルで、フリーウェアとは無料のソフトウェアであると考えている。石岡氏の場合は、自作フリーウェアを再配布(FTP サーバへのアップロードや CD-ROM への収録)する際には通知してもらうことにしているが、「ただ単にどういうところで使われているのか知りたいから」であるという。

[29] また、もう 1 人の主要な開発メンバーである福井貴弘氏はシェアウェアについて次のように述べている。「シェアウェアとは、その名が示すとおり、元来は「モノ(ソフトウェア)を共有しよう」という意識で始まっており、それを継続使用するための支払いも元来は「寄付(donation)」であった。現在でも継続使用にあたって任意のチャリティへの寄付を求める careware、葉書の郵送を求める cardware、何でもいい何かと交換を求

ただし、ソフトウェアとともにソースコードも公開すると、ソフトウェアの系統が複数に分岐し、同一ソフトウェアをルーツ・名称に持っていながら、バージョン間で互換性がとれなくなるおそれがあり、実際そのようなソフトウェアも過去に存在した。

こうしたソフトウェアの系統の分裂は、リーダーがいなかったり、その影響力が弱かったりする場合に起こりうると考えられる。しかし電信八号の場合は、原作者の石岡氏が開発主体・開発対象を一本化することを条件に開発を移管し、さらに条件付きでソースコードを公開したので、その轍を踏むことはなかった。

また、もう 1 人の主要な開発メンバーである福井貴弘氏は次のように述べている。コンピュータとソフトウェアの社会における 1 つの強固な文化である UNIX 文化は、「ないものは作る」「作ったものは全体還元する」という考え方を持っている。福井氏はこの文化の 1 つの具現化として、電信八号の開発に当たっているようである。

電信八号の詳細

電信八号のソフトウェアとしての規模は、原作者の石岡氏が開発していた時代から 5 万行程度であり、現在でも 10 万行を上回ってはいないという。

また、電信八号は冒頭で述べたようにエディタを自由に選ぶことができる(エディタ・フリーな)メールソフトである。これは原作者の石岡氏が当初から意図していたことであり(「メールの編集機能は当初から外部のソフトウェアに任せることにしていた」)、それに応えて熱心にエディタを開発した人達の要望に呼応するかたちで 1995、1996 年頃に DDE[31]を作成し、公開済みである。また、その後コマンド・オプションも公開したため、現在ではエディタを中心に少なからぬ数のソフトウェアが電信八号と連携して動作するようになっている。電信八号と連携する外部ソフトウェアの概要は表 4 のようになっているが、この他にも、標準メニューへのメニュー項目の追加、ログ拡張、時刻合わせ、PGP(暗号化)プラグイン、自動アップデータ(例えば、中村氏作成の「最新八号」)などが提供されている[32]。

表 4　電信八号連携ソフトウェアの代表例

用途	名称	作者	備考
エディタ	ペン猫	山本和隆	他 4 点
ビューワ	電ラブ－電信八号らぶらぶビューワ	山田智史	他 5 点
カスタマイズ	電信八号拡張メニュー	山田智史	他 2 点
コーディング	UUDeview for Windows	Frank Pilhofer & Michael Newcomb	他 3 点
通信	SHOWPC for Windows	KAWA &【なかま】	他 3 点
テンプレート	電信八号テンプレートエディタ	じゅんくん	
フィルタ	Def	ラッタッタ信玄	
編集	Creole	井上亜晴	他 1 点
メール操作	返信八号	山田智史	他 8 点
その他	DenZWrap	大和田哲	他 5 点

※　2003 年 6 月 28 日現在。公式サイトに掲載されている連携ソフトウェアを集計したもの。用途の分類は筆者による。

このような経緯を経て、DDE とコマンド・オプションが公開されているが、電信八号の拡張性・利便性をさらに高めるため、現在よりいっそうの情報公開や改良が検討されているという。具体的

める swapware など多様な寄付の形態が存在している。」

[30] 中村氏は、「日本のフリーウェアはアメリカにおける User-Supported Software、すなわち、ユーザの自発的サポート、自発的寄付(donation)という思想に影響を受けて成立したが、自発的寄付では開発およびサポートを支えるに足るだけの資金などが集まらないため、シェアウェアというカテゴリーのソフトウェアが成立したのではないか」と述べている。

[31] Dynamic Data Exchange の略語で、Windows 用ソフトウェアのあいだでデータやコマンドをやりとりする手順のこと。

[32] 公開されたインタフェースによらずに電信八号の機能を利用するようなソフトウェアも存在している。山田智史氏作成の「電極 Z 号化計画」がその代表である。

には、「フォルダ構造の公開」(青木茂氏)や「インストーラなどを追加してもっと環境設定を容易にし、初心者向けの敷居を下げること」(岩井雅治氏)などが議題に上がっているという。ただし、これらはいずれも検討中の課題であり、今後実現されるかどうかは確定していない。

開発環境

電信八号の開発言語には、Visual C++が一貫して使われている。

具体的な開発環境は、後述するように多数の開発者・テスタが参加する開発形態をとっているため、各人各様である。しかし、C++のコンパイラがあればそれ以外に特別な開発環境は必要なく、ごく普通のコンピュータを利用して開発に参加できる。

(2) 開発活動の開始・継続

原作者石岡氏による開発

原作者である石岡氏[33]が電信八号を開発しようと思った動機は、「『わかりやすいメールソフト』を自分が使いたいこと」と若干の「金儲け」であったという。

1994 年、IIJ が個人向けインターネット接続サービスを提供し始め、石岡氏もそれに加入した。だが、電子メールを使うに当たって困惑したことがあった。というのも、当時 Windows 用に提供されていたメールソフトは、「英語版 Eudora」「AL-Mail」「WinBiff」しかなく、どれも非常に使いにくく感じたからである[34]。それでも、しばらくは Eudora のコードを変換して使っていたが、1995 年春頃になると、「自分で作っちゃえ」「自分が使いやすい Windows 用メールソフトを作ろう」と考えるようになった。

開発に当たっては、UNIX 環境で広く使用されていたメールソフトである MH(Mail Handler)を意識し、平日の夜と土日を開発活動に当てて、開発開始から 3 ヶ月程度経った 1995 年夏にはソフトウェアが使える状態になったという。さらに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と感じ、Asahi ネットのソフトウェア掲示板に電信八号を投稿した。また、それ以前から、所属していた会社の同じ部署の同僚には自らの手で配布していたという。

こうした社内での配布活動から得られた反応や、Asahi ネットで「センスがよい」と賞賛を受けたことに調子づいて、どんどん改良を加えていったという。すなわち、石岡氏自身の願望や会社の同僚の反応、メールによるソフトウェアへの反応[35]を受け入れ、それらに応えるために開発を継続していった。

石岡氏によれば、当時は「素直に言われることを聞いていた」状況にあり、その背景には、ユーザから寄せられる要望・期待に応えたいという気持ちがあったという。石岡氏の言葉を借りれば、「できますか？」と問われると、「できるわい」と思って応えてしまう状況だったという。この時期の石岡氏は、「ユーザの期待に応えたい」という思いを抱くとともに、公開して満足感を得ること、配布サイトでの評価、雑誌などの評価記事、ダウンロード数の多さなどを通じて、うれしさや誇らしい気持ちを感じており、それが励みになっていた。とくに代表的な配布サイト「窓の杜」でダウンロード数がトップになったときは非常に嬉しかったという。

このように高いモチベーションに支えられ、当時は 1 ヶ月に複数回のバージョン・アップを行っていた。つまり、自分自身の願望、ユーザの反応を元にコードを記述し、コンパイルができたら即時公開していた。ただし、デバッグやテストも行わずに公開することに問題を感じたため、1996 年夏頃からは、リリース版と開発室版(β版)を分けて公開するようなった。その後、Windows95 が発売されたことや、1997 年頃から本格的にインターネットが立ち上がったことにより、電信八号のユーザも増えていった[36]。

---

[33] 石岡氏の専門は自然言語処理、テキスト・マイニングである。以前は機械翻訳などの開発に携わっていたが、現在は日本語解析プログラムおよびその周辺技術の開発に従事している。

[34] 石岡氏によれば「ヘルプを読む気もしない」ほど使いにくかったという。

[35] ユーザの中には、組み込み可能な APOP や MD5 のソースコードをメールに添付してくる人もあったという。

[36] この時期に行われた、電信八号を 16bit ベースのアプリケーション(現「電信八号 16bit 版」)から 32bit ベースのアプリケーションに転換する作業にはかなり「手こずった」記憶があるという。また、コンパイラを変え

しかしながら、1998 年頃には徐々にバージョン・アップの頻度が下がっていく。

ユーザから寄せられる要望の数に変化はなく、石岡氏もそれに応えようとしたが、ある時期からは「自分で使うには十分」と感じるようになった。同時にメールソフトとしては後発の「Becky!」が急速にユーザ数を増やして、当初抱いていた「電信八号で金儲け」というささやかな希望もかなえられる見込みが少なくなった。これらのことから、石岡氏が開発に寄せる情熱は低下してしまった。そのため、2 ヶ月に一度程度、自分自身が納得できるような要望や深刻なバグには対処していたものの、それ以前のように積極的にソフトウェアのバージョンを上げることはなくなってしまった。

その結果、以前と変わらないペースで寄せられるユーザからの要望に十分に応えられなくなった。この状況は、電信八号を改良するための要望を受け付けながらそれに応えられない「ソースコード死蔵」の状況であり、石岡氏にはそのことが苦痛に感じられるようになった。当時は、電八倶楽部のメーリング・リストには参加し続けていたものの、そのメールやユーザから寄せられる要望のメールを読むことすらしない「事実上の停止状態」だったという。

しかし、1999 年 7 月のある日、石岡氏はふと思い立ち、電八倶楽部のホームページを訪問して、その熱心さ、熱心なユーザと熱心な運営者の存在を感じ取った。その熱心さに対し、石岡氏は「要望が多数あり、熱心なユーザも多くいるのに、ソースコードを死蔵しているのは悪いことだ」と考え、ソースコード公開[37]を決断したという。

その後、電信八号に関するメーリング・リスト「電八倶楽部」で積極的な活動を行っていた中村氏に連絡を取り、1999 年 7 月 13 日に条件付き[38]でソースコードの公開に踏み切ることになった。

電八倶楽部への開発移管

現在の電八開発倶楽部・電八倶楽部の主要メンバーである中村氏、伊澤正之氏によれば、山田智史氏作成の「電極 Z 号化計画」のように、ソースコードが公開される以前から電信八号の改良ソフトは存在していたという。また、ユーザ側に広く「電信八号をもっとよくしたい」という欲求があり、「電信八号を無理矢理よくするメーリング・リスト」なども立ち上がっていた。

しかし、1999 年 7 月に石岡氏より実際にソースコードが条件付きで公開されると、それまでユーザの情報交換の場として存在していた電八倶楽部は、それに対応して開発を引き継ぐために、相応の準備をすることが必要となった[39]。

石岡氏がソースコード公開に当たって提示した条件を満たすため、配布用サーバをどのように確保するか、といったことについて電八倶楽部内で議論が行われた。この議論は電八倶楽部全体というより、後に発足する電八開発倶楽部のメンバーが既に中心になっていた。

その結果、サーバの確保に目処をつけた後、もう一つの石岡氏の条件である「電信八号を複数の系統に分裂させないこと」を守るため、「公式ビルダ[40]」を任命して、公式ビルダがバージョン管理を行うことが決定された。そして、斉藤氏によって公式ビルダの役割が明文化された。その役割とは、ソースコード管理、ビルド管理、サイト管理であった。さらに斉藤氏は自らが公式ビルダに就任しようとするが、諸般の事情により断念せざるを得なかった。そこで、1998 年 8 月 21 日、機材

ることにより、それまで顕在化していなかったバグが顕在化する現象にも見舞われたという。

[37] ソースコードの公開に関しては、それ以前に川瀬裕氏などから要望があったが、「恥ずかしいコード(石岡氏)」であると思っていたため、公開に踏み切れなかったという。ただし、ユーザ側から見れば「感動的なコード(青木茂氏)」「遊び心のあるすばらしいコード(中村氏)」に感じられたという。

[38] ソースコード公開の条件とは、(1)配布用サーバの確保、(2)電信八号を複数の系統に分裂させないこと、であった。現在この条件を満たして原作者の石岡氏の他にソースコードを保有しているのは、電八開発倶楽部と「のびのび情報教育研究会」、その他 1 団体である。

[39] この時期の電八倶楽部に関し、そのメンバーの 1 人である本田善久氏は次のように述べている。「電八倶楽部は元来ユーザの集まりであり、それがソースコードを預けられて多少混乱している観があった」。また、電八倶楽部の他のメンバーも、この当時、既存の電八倶楽部と開発継続のために新設した電八開発倶楽部の軋轢があったこと、ソースコードが条件付きで公開されたことに不満を持つ人々が存在し、彼らが電八倶楽部から離れていったことを認めていた。

[40]「公式ビルダ」とは、β版の開発に携わる人のことである。

購入を機に中村氏が公式ビルダに立候補し、倶楽部の信任を経て公式ビルダに就任した。

各開発者の開発への参加経緯

公式ビルダの 1 人である中村氏は、以前から自作ソフト、移植ソフトを多数開発している人物である。

もう 1 人の公式ビルダである福井氏は、1995 年頃からパソコンを使うようになったが、Windows95 も 16bit 版の環境も「嫌いで仕方なかった」ため、その当時から WindowsNT 3.1 を使用していた。電信八号に関しても当初は 16bit 版を使用していたが、後に 32bit 版に移行した。

福井氏は 1996 年頃からインターネットを利用するようになった。この当時は定番のメールソフトと呼べるものがなかったため、Internet Magazine の付録 CD に入っていた電信八号をメインのメールソフトにしようとも考えたが、最初のうちは Web ブラウザの Netscape Navigator のメール機能を利用していた。

しかし、後にメール・データを変換、利用することを考えた場合、メール・データがテキスト・ファイルになっている電信八号の方が好都合であると考えた。さらに、福井氏は何をするにもエディタを使い、しかも使い慣れないエディタは絶対に使いたくないというタイプであるため、エディタが自由に選べる電信八号が非常に良いと判断して本格的に使用するようになり、1996 年 12 月には電八倶楽部にも参加するようになった。ただし、電八倶楽部への参加は購読のみ(ROM; Read Only Member)であり、あまり積極的な参加とは言い難かった。さらに、1998 年頃には、電信八号の使用をやめようかと考えるようになっていた。それは、バージョンが上がらず不便さが解消されなかったからだという。

このように考えていた福井氏が、再び電信八号に深くコミットするようになったのは、1999 年のソースコード公開が契機である。福井氏は、以前から他のソフトウェアの開発などでプログラミングの経験を持っており、かつ「オープンソース」という考え方に賛同していたため、1999 年 7 月にソースコードが公開され、誰でも開発に参加できるようになったことを知ると、その開発を担う電八開発倶楽部に設立当初から参加する[41]。

しかし、電八開発倶楽部の設立初期は、開発が思うように進まなかった。すなわち、アップデートのためのパッチを誰もまとめておらず、ほぼ 1 年間にわたってパッチが溜まって、それがバイナリ版に反映されない状況が続き、電八開発倶楽部内でフラストレーションが高まっていた[42]。

これを受けて福井氏は、「ユーザにフィードバックできるなら」と考え、自ら公式ビルダに立候補する。福井氏は、「パッチだけがあるという状況は、個人的にビルドができる人には、改良されて不具合がなくなった電信八号が使えるが、それができない普通の人には使えないという、不公平で不条理な状況だ」と考え、同時に「(よりよい電信八号を)誰でも使えるようにしたい、みんなに使って欲しい」「みんなの役に立ちたい」と思い、公式ビルダに立候補したのだという。加えて、メーリング・リストを通じて大変な状況にあることが推察された中村氏をサポートし、自分がビルドを引き受けることで中村氏にパッチ作成をして欲しい、という考えもあったという。

福井氏の立候補は所定の手続きを経て信任され、同氏は 2000 年 6 月 15 日に公式ビルダに就任した。就任当初の福井氏の方針は、「溜まったパッチを消化しよう」というものであった。そこで、バグ修正のみだった電八開発倶楽部の最初のビルド(2000 年 3 月 15 日：V321.2b6-stable)にはタッチせず、その次の公式ビルド(V321.2a71)に注力して、溜まっていたパッチの取り込みによる機能向上、不具合修正を果たした。この当時の 1、2 ヶ月は「電信八号に触るのが仕事」のような状況であり、α 版[43]の公開も非常に早い「週刊電八状態」であったという[44]。

また現在、電八倶楽部において連絡・渉外係を担当している伊澤正之氏は、会社の業務において

---

[41] 当時の心境について福井氏は「ソースコードにさわれるなら参加せざるを得ない」と思ったと述べている。

[42] こうした状況を反映して、公式ビルダの中村氏が「辞職する」と言い出すまでになっていた。

[43] この時期から、特に不安定なバージョンをα版として電八開発倶楽部内限定で公開することが通例になったという。

[44] 公式ビルダになったことにより、時間的余裕がなくなり、自分でパッチを作成することは少なくなってしまったという。

工場の生産管理システムや受発注などの基幹系業務に携わっていたため、コンピュータの利用はしていたという。ただし、こうした業務ではメインフレームや UNIX が使用されていたそうである。

このようなコンピュータ歴を持つ伊澤氏がパソコンに触れるようになったのは、やはり会社の業務がきっかけである。1995 年夏に会社がホームページを開設し、顧客からの問い合わせのメールを受け付けるようになったため、システム部門の担当者としてそのパソコンを管理する立場に伊澤氏が就いた。同時に、顧客からの問い合わせのメールにも応える必要が生じたため、1996 年春頃、遅くとも 1996 年夏頃には電信八号を使うようになったという。

顧客からの問い合わせメールへの対応用に伊澤氏が電信八号を選択したのは、それまで使用していた商用のインターネット用ソフトウェアでは、複数のメール・アカウントを使用することができず、業務上受付アドレスを複数化したいという当時の要望に応えられなかったためであった。つまり、共用ダイアルアップ接続のパソコン上で、複数のメール・アカウントを切り替えられるソフトウェアとして電信八号を使用することにした。伊澤氏はメールソフトとして電信八号に決定するまでに、このような機能を持つメールソフトをネット上で検索したが、当時該当するフリーソフトは電信八号のみだったという。加えて、電信八号が MIME や Base64 などの当時最新の RFC[45]準拠であったことも採用の理由であった。

こうして電信八号のユーザとなった伊澤氏が電八倶楽部に参加したのは、1997 年 9 月頃であった。顧客から寄せられるメールの中に、変則的で読めない形式のもの[46]が少なからず混じるようになったため、これに対処する方法、情報を得たいと思い、電八倶楽部に参加したという。

ついで、中村氏や福井氏のように公式ビルダではないが、電八倶楽部の主要メンバーやプラグイン・ソフトの開発者、あるいはユーザ兼テスタとして電信八号の開発に参加している人々について紹介する。

外部ソフトのビューワである「電ラブ」や、Perl を利用して返信フォーマットを作る「返信八号」「電信八号++」などの電信八号の「ヘルパー・アプリ(お助けソフト)」の作者である山田智史氏は、自身がフリーウェア作者である。現在は開発に参加しておらず、電信八号のユーザではないが、石岡氏が開発をしていた時期には、上記のようなヘルパー・アプリを開発、提供し、1996 年頃から 2000 年頃までは電信八号のユーザであった。

山田氏は 1996 年当時 NEC のソフトウェア開発子会社に勤務していた。当時はやはり、メールソフトとして定番と呼べるものがなく、その中でも AL-Mail や「WeMail」を使ってみたが、特別使いやすさを感じなかった。そこで直接の面識は無かったものの、同じ NEC に勤務していた石岡氏が電信八号を開発・公開していることを知り、またその説明に「MH を意識した」とあったので、電信八号を使ってみた。さらに、プログラミング技術を活かして電信八号のヘルパー・アプリを開発・提供するようになったという。

岩井雅治氏もまた、現在はコンピュータ環境が Linux ベースに移行したため電信八号を使用していないが、1999 年から 2000 年初め頃までユーザであり、現在も電八倶楽部のメーリング・リストに参加している人物である。

岩井氏は Outlook Express からの乗換で電信八号を使用し始めた。電信八号を選んだ理由は、他にメールソフトが見当たらなかったこと、無料で使用できること、エディタが選べたことであったという。また、学生時代から Windows と UNIX の環境に触れていたため、電信八号の環境設定やユーザ・インターフェースには抵抗感がなかったという。

電信八号について岩井氏は、「無料であり、かつシンプルな機能美を目指しているように思われる。『電信八号の方向性や環境設定、ユーザ・インターフェースに抵抗のある人は、他のメールソフトを使えばよい』、という考え方があるのではないか」と述べていた。

なお、同氏の電八倶楽部への参加は、「ユーザであるので、そのメーリング・リスト、つまり電八倶楽部が存在するのなら参加しよう」という心情からであり、現在は電信八号のユーザでないこともあって、メールの購読が中心であるという。

---

[45] Request for Comment[s]。インターネットに関する技術仕様。

[46] これらの変則的なメールは、マイクロソフト社の Outlook Express や Instant Messenger から発せられたものが多かったという。

寺島裕貴氏は、現在電八倶楽部に参加しているメンバーである。寺島氏は、1997 年頃から父のインターネット環境、パソコンを利用し始め、Internet Magazine の付録で電信八号を知るが、当時はその価値が分からなかったので Netscape を利用していたという。

その後、パソコンにのめり込み、自分でプログラミングをするようになって自分の環境構築にこだわりを持つようになり、その一環としてメール・データの保存形式が重要であることに気づいた。このメール・データの保存形式という観点から見ると、AL-Mail や Netscape では不満があり、他方、電信八号の価値を再発見したので、「1997 年当時とは状況が変わっているな」と思いつつも、それを利用するようになったという。寺島氏は、「自分がそうであったように、電信八号の価値、良さが分かるためにはある程度のパソコン、インターネットに関する知識が必要だろう」と述べていた。

なお、寺島氏の電八倶楽部への参加は、学校関係のメーリング・リストに参加し、その繋がりでメーリング・リスト「is-uno」に参加、さらにその繋がりで電八倶楽部に参加するようになったという。

本田善久氏は、本来の仕事においてパンチカード時代からメインフレームのコンピュータを使用している人物である。そのため、電子メールもメインフレーム上で IBM の BITNET を使用している。

本田氏もダウン・サイジングの流れに従い、パソコンも併用するようになったが、当初はパソコンをクライアントにしてメインフレームを利用していた。しかし、やはりその使用方法では面倒なため、パソコンとインターネットを利用したメールソフトを探し、無料でかつ、メインフレームの電子メール同様にフォルダ構造、ヘッダ情報を隠さない電信八号を使用するようになった。

本田氏が電八倶楽部に参加するようになったのは、1997 年末～1998 年初頭ごろに「RFC 準拠であるのであるならば、大文字と小文字を区別せずに処理すべきだ」とメールで発言したことがきっかけであった。本田氏の電八倶楽部への参加状況は、基本的に 1 ユーザであり、メールの購読と投稿をするにとどまっており、開発には全く参加していないという。

和田和子氏は自身も認める初心者ユーザである。伊澤氏に誘われてメーリング・リスト is-uno に参加し、そこで電信八号の存在を知った。現在は電信八号を受信メールのスクリーニングのために使用している。電信八号は、メール受信時に POP サーバ上のメールのヘッダ部分だけを取得して表示する機能が備わっている。そのため、受信しないメールをサーバ上から削除したり、必要なメールだけを選んで受信するという和田氏のような利用方法が可能なのである。

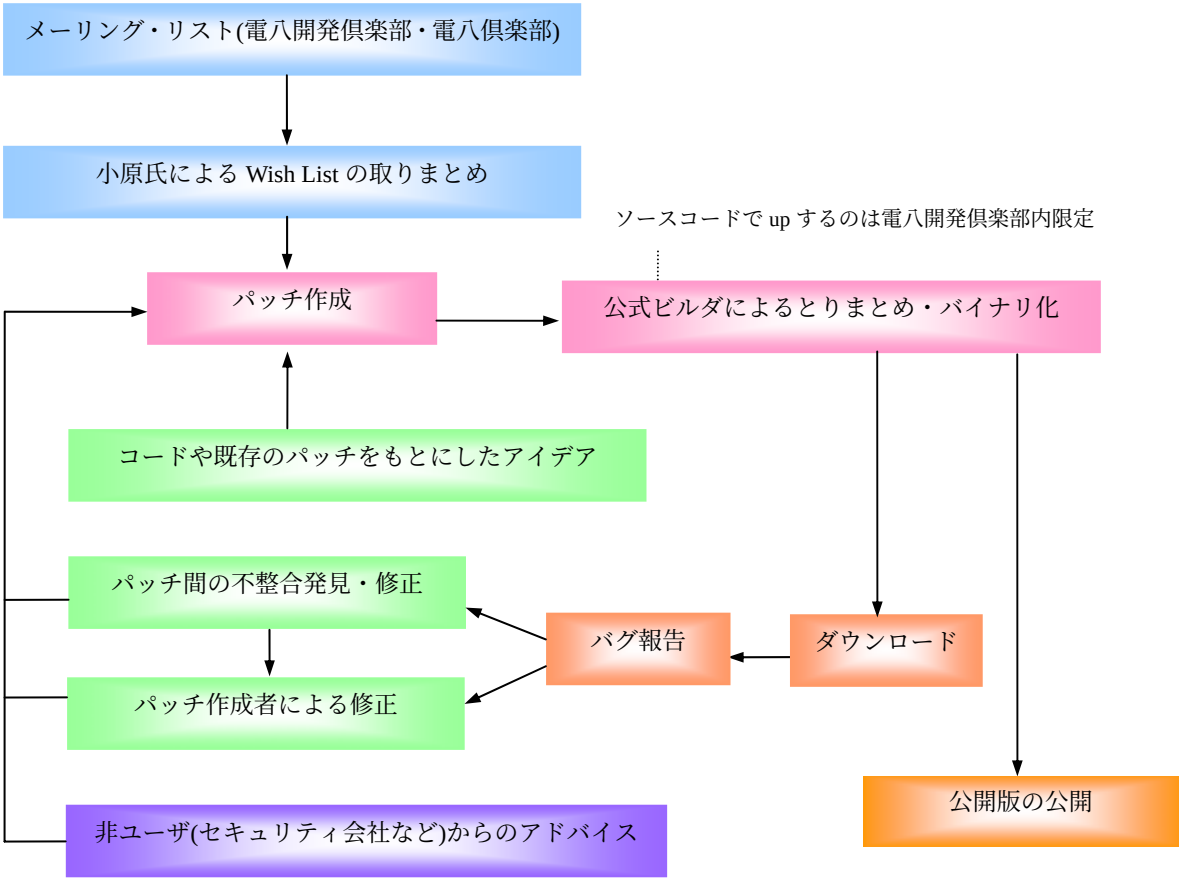
電信八号のメーリング・リスト電八倶楽部への参加は、「尊敬できる人が多いな」と思ったからだという。和田氏にとって同メーリング・リストは本来の仕事にも役立つため、頻繁に目を通しているという。また、自分自身が初心者であることを踏まえて、初心者が電信八号を利用するとどのようになるのかを、メーリング・リストに報告したこともあるという。

(3) 開発活動の詳細

バージョン・アップ・プロセスの概要

現在の電信八号のバージョン・アップ・プロセスは、①パッチ作成、②公式ビルダによるとりまとめ・バイナリ化(ビルド)、③配布およびバグ報告、①'新たなパッチ作成、というサイクルが基本になっている。バージョン・アップに関わるさまざまな活動の流れを図示したものが図 4 である。

図 4　電信八号のバージョン・アップ・プロセス[47]



パッチ作成の際に主に参照され、ビルドの際にもパッチを取り込む優先度を判断するために参照される役割を果たすのが、公式ビルダの 1 人である小原良仁氏がとりまとめる Wish List である。

Wish List は、ユーザから電八倶楽部・電八開発倶楽部に投稿された修正や機能追加の要望をとりまとめて、日本語で記述したものである。このリストは、あくまで修正や機能追加の要望をプールしたものであり、それが開発グループの誰かにパッチ作成の義務を負わせるものではないとされている。言い換えれば、公式ビルダを含めた開発グループの誰かが、寄せられた修正や機能追加の要望に共感すれば、それ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ことになるが、そうでない要望は未処理のまま Wish List に残り続けることになる。

このようにパッチ作成の 1 つの端緒は、Wish List に上げられた要望を消化することである。加えて、ユーザ自身が使っている中で生じた問題を解決するためにパッチを作成したり、石岡氏のコードを参照してそれを技術的に改良するためにパッチが作成されることもある。さらには、RFC が提示する規格に沿うためにパッチが作成されたり、セキュリティ・ホールを探す企業からの報告に基づいてパッチが作成されることもある。

こうした様々な経緯、背景をもって作成されたパッチを統合し、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氏と福井氏である。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、すなわちデータ消失の危険が伴う場合には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。

電八開発倶楽部および電八開発倶楽部の登録者はα版やβ版をダウンロードし、使用してみて、

[47] 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。

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て、一般向けの(誰でもダウンロードできる)公開版がリリースされる。

現在のバージョン・アップ状況

中村氏、伊澤氏によれば、「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。実際、更新履歴に基づいて、バージョン・アップのペースを把握すると、表5のようになり、そのペースの速いことが見て取れる。

表5　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

※　2003年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。

すでに述べたように、電八倶楽部としての最初の公式ビルドこそ、開発移管時の混乱と未処理のパッチの膨大さのためビルドの作成・公開が遅れたが、福井氏の協力もあって大量の未処理パッチを吸収して最初のビルドを公開することに成功し、その後は順調にバージョン・アップを重ねている。

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。福井氏が自らの作業と認識した電八 FAQ 集の移管作業終了後、ビルドは中村氏と福井氏がほぼ交代で行ってきた[48]。公式ビルダが 2 人いることにより、最新のビルドの作成・公開は 1 週間に1度のペースで行うとしても､それぞれの公式ビルダの作業は隔週で行えばよいことになり、作業が軽減されるという[49]。したがって、その公開ペースは早い場合には 3 日、遅くとも 6 ヶ月毎にビルドが行われてきた。

最近では、提出されるパッチが減ったこともあってビルドが滞りがちだが、それでも平均してほぼ 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[50]。ただし、中村氏が多忙なため福井氏がビルドを行うことが多くなっているという。

公式ビルドの具体的な作業自体は、機械的な作業であるという。そのため、提出済みのパッチの数を勘案した上で作業に取りかかり、ほぼ半日〜1 日で履歴テキストの更新までを済ませることが可能[51]だという。

なお、公式ビルドの作成に当たっての留意点、ポイントは 2 人の公式ビルダの間で微妙に異なっている。

中村氏にとって電信八号は、RFC 完全準拠をベースに追加的要素のあるメールソフトである。そのため、電信八号本体とプラグインとの切り分けに留意し、電信八号本体が上記のようなメールソ

---

[48] 公式ビルド作成を交代で行うことに関して福井氏は、「こうした役割分担は自然とできあがったもの」で、「自然の成り行きに任せた結果、どちらが電信八号のバイナリ化を行うかを明確化しなかった」と述べていた。さらに、「電八倶楽部・電八開発倶楽部はボランタリーな組織なので、仕事を割り当てるようなことはしたくない」「できる人ができることをするのが理想」とも述べていた。

[49] 「公式ビルダ 2 人制」について中村氏と伊澤氏は、「分裂の危険性もあるが、助けられること、相互に補完しあえるメリットが多い」と述べている。

[50] 電信八号の開発はあくまで「趣味」のため、公式ビルドの公開が遅れがちになる傾向もあるという。そのため、公式ビルドの公開時期を予め決めておくこともある(例えば「クリスマス・ビルド」など)。

[51] このことを利用して、「数字遊び」が好きな福井氏はキリのいい日、節目の日にビルドを作成し、タイムスタンプを印象的なものにするようにしている。福井氏は「ビルドした日付がファイルのタイムスタンプに残るとうれしい」と述べている。

フトであり続けるように心がけているという。また、バージョン・アップの積み重ねによって内部的には大分変化しているものの、外観・外見は当初の電信八号を維持している。つまり、RFC 完全準拠、外部仕様の固定化[52]、外観・外見の維持をしつつ、ソフトウェア内部の挙動、機能を可能な限り向上させることが、中村氏が公式ビルド作成に当たって留意している点である[53]。

一方、福井氏の公式ビルド作成に当たっての留意点は、「他人に使ってもらうもの」なので、致命的なバグが存在しないようにすることだという。また、パッチのとりまとめを極力「機械的に」行い、福井氏のプログラマとしての独自性が電信八号に表れないよう心がけているという[54]。

電信八号のバージョン・アップにおけるユーザの役割

初期の電信八号のバージョン・アップにおけるユーザの役割について、原作者の石岡氏は、作者が想定しなかったような使い方を見いだし、使用していたことが印象的だったと述べている。具体的には、電信八号のデータ形式が単純な S-JIS テキストであることを活かして電信八号を他のソフトウェアで操作したり、プログラム・サイズの小ささを活かしてフロッピー・ディスクに携帯したりするユーザがいたという。また、作者が考えるアピールポイントとユーザが評価するポイントが異なっていることも印象的だったという。

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、伊澤氏、青木氏が異口同音に、「電信八号の開発において、ユーザすなわちメーリング・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。中村氏などによれば「一度集まってみたい」という欲求はあるが、その機会がないとのことだ。

この点に関して岩井氏は、「オンラインで十分な交流があるので、オフラインでの交流の必要性は高くない」と述べていた。岩井氏は、オフラインの交流が皆無に等しいのは、電信八号の開発が本業ではないことと、ミーティングよりもメール、メーリング・リストの方が効率的[55]であることによると述べていた[56]。

(4) 開発組織

電八倶楽部・電八開発倶楽部の成立経緯

電信八号の開発初期の 1995 年末から 1996 年初めに、石岡氏は自ら掲示板を設置し、ユーザから

---

[52] より平易に言えば、「同じソフトウェアが電信八号と連携して機能すること」である。

[53] こうした中村氏の方針に満足している人もいるが、不満を持っている人もいるという。その不満は、「電信八号は何も変わっていないじゃないか」というものである。だが、中村氏は、「『何か変わった』と明らかに分かるようなバージョン・アップ、すなわちメジャー・バージョン・アップは『一体、電信八号とはどのようなソフトウェアなのか』という問題を生じさせるし、フル・スクラッチでコードを書き直したものは電信八号と呼ぶべきではないだろう」と述べている。

[54] この他に、中村氏と伊澤氏によれば、「中村氏は完璧主義、福井氏はアバウト＝多少不完全でも公開し問題が生じたら修正すればいいだろう」という違いがあるという。

[55] ここで「効率的」というのは、開発者全員が参加して開催するミーティングを持ち、開発を効率化することの利点を否定しているのではなく、物理的・時間的な制約を加味すればオフラインのミーティングにはその開催自体にコストがかかりすぎるため、メールおよびメーリング・リストでの情報交換がトータルとして効率がよくなる、という意味である。

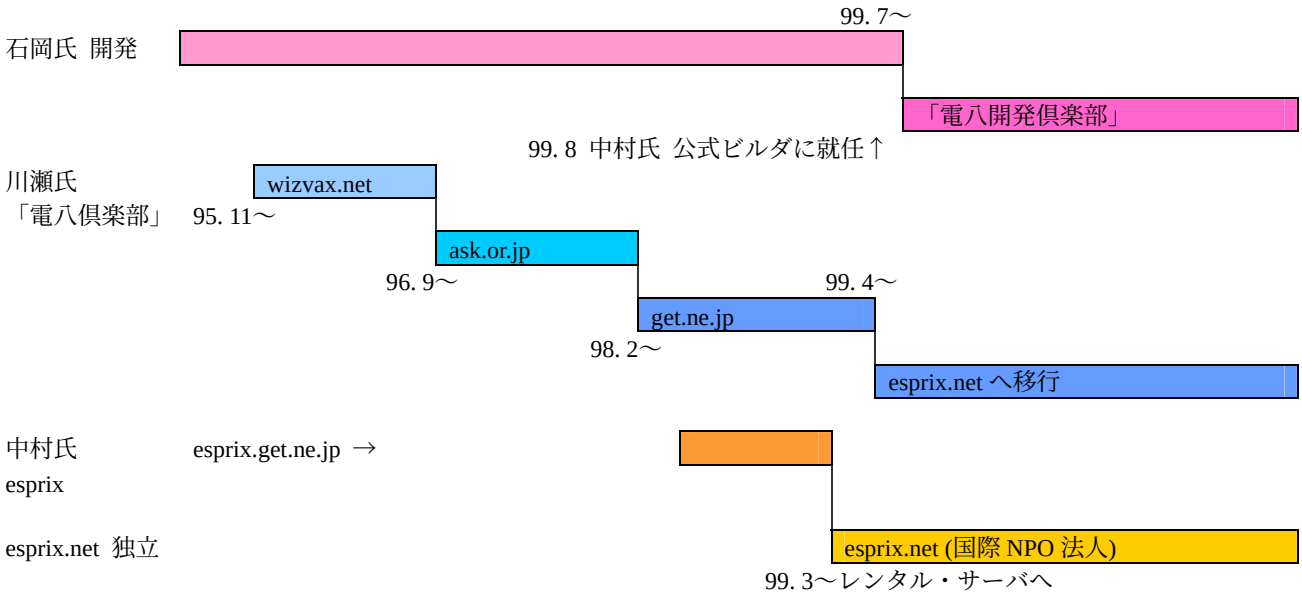
[56] ただし、岩井氏によると、他のメーリング・リストでオフラインの交流(オフ会)や、ソフトウェアの作者を囲むオフラインの会合は行われているという。ただし、ソフトウェア開発者同士の横の連携はそれほどなく、むしろ開発言語毎のメーリング・リストを起点にした勉強会が開かれているという。OS 別で言えば、Linux や BSD 関連で交流は盛んだが、Windows では「日本ウィンドウズ・エヌティ・ユーザ会(Japan WindowsNT Users Group)」が目立った活動をしている程度である、という差があるという。さらに Linux や BSD では企業が啓蒙目的で行うセミナーも開かれているという。

の同じような質問に答えなくていいようにしていた。だが、それとは別にユーザ側で独自にメーリング・リストが立ち上げられ、このメーリング・リストが電信八号に関する要望の提示・受付の窓口に変わっていった。このユーザ独自のメーリング・リストの延長線上に現在の電八倶楽部、電八開発倶楽部がある。

より詳細な電八倶楽部の設立経緯を述べると、元来はユーザの情報交換サイトとして 1995 年に川瀬裕氏がホームページを立ち上げた。1996 年の時点では約 40 人が参加し、設立初期から原作者の石岡氏に要望を取りまとめて提出することを行っていた。

その後、石岡氏とユーザとの接点が基本的に電八倶楽部に一本化されるようになり、石岡氏も電八倶楽部において開発室版(β版)の配布などを行うようになった。これ以降、川瀬氏と中村氏が立ち上げた現在のメーリング・リストの基盤である esprix.net に至るまで、ホスト・サーバなどの移転が幾たびか行われたが、電八倶楽部は常にユーザの情報交換の場としての機能を維持し続けて現在に至っている。さらに、1999 年 7 月にソースコードが公開されたことに伴い、それまでの電八倶楽部とは別個のメーリング・リストとして電八開発倶楽部が立ち上げられ、現在の開発、配布、情報交換の体制ができあがった。

図 5　電八倶楽部・電八開発倶楽部の年表[57]



現在の開発組織

現在、電八倶楽部の総参加者は 1,000 人、メーリング・リストに積極的に発言する活動的な人は常時 40～50 人いるだろうとのことである。他方、開発により深くコミットする電八開発倶楽部は、総参加者 300 人、活動的な人は常時 4～5 人である。ただし、いずれのメーリング・リストにおいても活動的な人は一部を除き入れ替わっている可能性があるという。

電信八号の開発活動、すなわちソフトウェアのコーディング、検証・テストは、これら電八倶楽部、電八開発倶楽部全体で行われているが、強いて、より開発に深く関わっている人物を特定すると、それは 4 人の公式ビルダと 3 人の連絡・渉外担当者である。

公式ビルダは、パッチのとりまとめを行うプライマリ・ビルダの中村氏と福井氏、Wish List 担当の小原氏、最終公式ビルダとして裁定者の役割を果たす川瀬氏で構成されている。また、連絡・渉外係として伊澤氏が、中村氏の負荷を軽減するために 2000 年 5 月に立候補して信任され、現在ではさらに小川修氏と近田守也氏が加わって 3 人で連絡･渉外業務を行っている(2002 年 4 月より)[58][59]。

---

[57] 1999 年にホスト・サーバの移転と組織の整備が行われたのは、それまで使用していた get.ne.jp が売却される懸念が生じたためである。それにともない、get.ne.jp にあったメーリング・リストも esprix.net に移管された。

[58] なお、4 名の公式ビルダと連絡・渉外係の業務連絡のためのメーリング・リストとして den8-contacts が稼

開発組織の紹介の最後として、電信八号の開発組織においてリーダーに該当する人物はいるのかどうか述べておく。この点について中村氏は、電信八号の開発組織にリーダーはおらず、「宙ぶらりん」「不安定な状況」になっていると述べていた。さらにそうしたリーダー不在の状況は、一長一短であり、メジャー・バージョン・アップ、例えば「電信八号 2」の開発などに踏み切れない点においてリーダーの不在は欠点だが、Linux のように最初の開発者がリーダーとして開発を押し進める状況と比較した場合に、必ずしも悪い点ばかりではない、よい面もあると述べていた。ただし、リーダー、サブリーダーといった役割の人物はいないものの、公式ビルダに「技術的拒否権」すなわち、不具合を生じさせる可能性があるパッチをビルドに統合しないことを決定する権利があることを指摘していた[60]。

他方、もう 1 人のプライマリ・ビルダである福井氏は、電信八号のコミュニティは自然発生的、リーダー不在の特殊なコミュニティであると述べていた。しかし、筆者達が原作者である石岡氏のコードの存在そのものや、ソースコードの公開条件などがリーダーに代わる役割を果たしているのではないかと尋ねたところ、「石岡氏の存在も大きいが、それを熱心に説き、公式ビルドに反映させてきた中村氏の役割が非常に大きいだろう」と述べていた[61]。

(5) その他

電信八号の開発・使用が継続されている理由

電信八号の開発・使用が継続されている理由、つまり、電信八号のソフトウェアとしての魅力をインタビュー参加者の意見にしたがってまとめると以下のようになる。

まず、中村氏、伊澤氏、青木氏は、①データがわかりやすく配置されていてその挙動が見えること、②データのほとんどがテキスト・ファイルで構成されており、メール本体もテキスト形式なので管理とリカバリーが容易であること、③エディタ・フリー[62]であること、④UNIX 上で使い慣れた MH、emacs 上で動作する MH-e に近い操作感が Windows 上で実現されていること、などを挙げていた。

さらに中村氏は、電信八号が「原作者である石岡氏の人柄がにじみ出ているメールソフト」「職人の道具のようなソフトウェア」であると述べていた。また、プロセッサが低性能でも軽快に動くこと、「プログラムはソースコードで公開することが普通」という UNIX 文化と親和性が高いことも魅力であるとしていた。

電信八号と UNIX および UNIX 文化との親和性に関しては、山田氏も「UNIX フレーバー」という言葉で同様の評価をしていた[63]。具体的には、エディタ・フリーであること、1 メールが 1 つのテキスト･ファイルであること､動作の仕方などが電信八号と UNIX の共通点であると述べていた。

山田氏が挙げたその他の電信八号の魅力は、プログラマからの観点として、電信八号の設定ファイルなどがテキスト形式で記述されているため、「いじり易い」「いじくり倒しやすい」点にあると述べていた。

他方、福井氏は電信八号のユーザ・インターフェースは 16bit 時代の Windows3.1 に近く、むしろその背後の動作が「UNIX ライク」であると感じていた。

動している。

[59] このケース作成後、さかの氏、青木氏、寺島氏が連絡係になっている。3 氏の役割分担は、さかの氏が渉外係相当、青木氏は見本誌保管係相当となっている。信任の日付等は、インターネットメーラー電信八号オフィシャルサイト (http://denshin8.esprix.net/) を参照のこと。

[60] このような開発組織、開発リーダー不在の状況を、中村氏は「電信八号の開発の進め方は BSD をモデルにしている」という言葉でまとめていた。

[61] この福井氏の発言を解釈すれば、電信八号の開発組織の組織文化を醸成したのが石岡氏であり、それを守り育てている、すなわち管理している中村氏が現在のリーダーである、と言えるだろう。

[62] 伊澤氏によれば、電信八号のこの特性を利用して、連携するビューワやエディタと組み合わせ、電信八号をパソコン上で常駐させておいて、ビューワやエディタから電信八号を操作する、という使い方もありうるという。

[63] 電信八号と同時期に NEC グループの社員によって開発されたメールソフトとして WeMail があったが、UNIX フレーバーだったため、山田氏は電信八号を使うようになったという。

福井氏自身は UNIX の使用経験はないが、DOS の使用経験はあり、そのため「動作が透けて見える」電信八号のようなソフトウェアが魅力的なのだという。加えて、前述のようにエディタが好き、すなわち可能な限りエディタを使いたいので、エディタ・フリーの電信八号がメールソフトとして好都合なのだという。

これらの発言をまとめて福井氏は、電信八号の魅力として、①「皮の薄さ」「動作が透けて見えること」「ゴテゴテしたユーザ・インターフェースをかぶせていないこと」、②動作のシンプルさ、③エディタ・フリーであること、④バックアップの容易さ、の 4 点を挙げていた[64]。

他の意見としては、岩井氏が無料であることを挙げていた。加えて岩井氏は、エディタ・フリーであること、1 通のメール・データが 1 つのテキスト・ファイルになっていることも挙げていた。

同様に寺島氏も無料であることとエディタ・フリーであること、1 通のメール・データが 1 つのテキスト・ファイルになっていることを魅力として挙げていた。また寺島氏も福井氏と同様に「皮の薄い」ソフトウェアであることが電信八号の魅力の 1 つであると述べていた。

これら各氏とやや異なった観点から電信八号の魅力を述べていたのは、本田氏と和田氏である。

まず本田氏は、電信八号の魅力を「わかりやすさ」という言葉で表現していた。その意味するところは、例えば、ログ・ウィンドウがあるのでサーバとのやりとりがどうなっているのかわかり、エラーの原因が特定しやすい、ということである。

ただし本田氏は、他の人に電信八号を薦めることには慎重であるという。というのも、多くの人は、メールソフトとは Microsoft 社の Outlook のようなものであるという先入観を持っており、また、電信八号はある程度のコンピュータ・スキルがあれば使いやすいが、そうでなければ使えないメールソフトだからだという。

次に和田氏は、電信八号の魅力をその開発姿勢に対して感じていた。具体的には、視覚障碍者へのフォローを非常に意識していることや、いろんな人達に優しいソフトウェアである、そうあろうとしていることが魅力的だと述べていた。

メーリング・リストへの参加継続の動機づけ

公式ビルダの中村氏は、自分が開発をし続ける理由を、「電信八号の公式ビルドを維持することは自己実現」であり、「どうしても止めることができないこと」であると述べていた。また、「フリーソフトは自己実現の発露の 1 つである」とも述べていた。

より具体的には、電信八号、電八倶楽部に愛着を持つ人達のために開発をしており、そうした人々、例えば国内のユーザや外国のユーザなどからメールを受け取って「世界中から応援を受けている感じ」を持ち、自分自身でもユーザ達に貢献することができている、役に立っているという実感を得ているという。さらに、「作ること、使うこと、メーリング・リストに参加すること自体に誇りを感じて」おり、「自分の生き甲斐として」1 日 3 時間ほどの開発作業を続けているという。

もう 1 人の公式ビルダである福井氏は、当初は大変そうな状況にあった中村氏のサポート、バイナリ版に反映されないパッチの氾濫という状況の収拾のために開発に参加し始めたという。「自分が公式ビルダになれば、みんなに最新のバイナリ版を届けてあげられる」というのが当時の率直な心境であった。

ただし、福井氏は自分自身を「求められれば何でもやるタイプ」と思っており、正直な気持ちとしては、「公式ビルダになるよりパッチ作成を続けている方がよかった」「プライベート・ビルドを自分で作っていた方が楽だった」とも述べている。

しかし、公式ビルダ就任後、福井氏の心境に変化が生じたようであり、「公式ビルダになってみてプライベート・ビルドよりも大変だが、自分が作ったものが『公式』として認められることに喜びを感じるようになった」とし、さらに中村氏同様、公式ビルダの役割を果たすことで溜まっているパッチをユーザに届けられるという意味でみんなの役に立っていることを実感しているという。また、現在も公式ビルダとして開発に参加し続けている理由については、「自分で使っているモノは自分でメンテナンスしたい」「自己満足」であると述べていた[65]。さらに、「たとえ公式ビルダが

---

[64] なお、中村氏が挙げた動作の軽快さが電信八号の魅力の 1 つであることには、福井氏も同意していた。

[65] これらの理由に加えて「自分で公式ビルドを作っていると、自分が改善・改良したい点を組み込みやすい

死んだとしても、(電信八号に関する)コミュニティが生き続けることが理想」だとも述べていた。

連絡・渉外係を担当の伊澤氏は、電八倶楽部に参加したことにより、自身に欠けていたインターネットでのメールに関する知識を豊富に得ることができたと感じ、同時に電八倶楽部のようなコミュニティのあり方、それを運営している川瀬氏の活動にも惹かれるものがあったという。こうした電八倶楽部への興味に加え、中村氏が多忙に見えたこと、自分が「お世話になった」コミュニティに何らかの形で役に立ちたいという気持ち、それ以前に発生したメーリング・リスト上での問題には接続環境の貧弱さなどのため問題処理に参加できずその処理過程を見ているにしかなかったが、連絡・渉外係に立候補する段階では「今度は参加できそう」と思われたこと、などがあったため、連絡・渉外係に立候補したのだという。また伊澤氏は、自らの参加理由を、電信八号が「1997 年から使い続けていて『無いと困る道具』だから」「電八倶楽部・電八開発倶楽部というコミュニティの動きを見ていることも面白い」とも述べていた。

以下、各ユーザの参加理由を順に見ていくことにしよう。

青木氏[66]の参加理由は、「現実逃避」であるという。また、困った人を助けてあげることによって満足感が得られることもその参加理由に挙げていた。

岩井氏の場合、現在は電信八号というソフトウェアを使ってはいないが、それでも電八倶楽部に参加し続けているのは、面白いから、勉強になることがあるからだという。

寺島氏は、電八倶楽部は「居心地がよい」メーリング・リストであり、「優しさを感じる。何でそこまで優しくなれるかなと思うときすらある」ものだという。電八倶楽部の雰囲気[67]、とくにその優しさは、他の技術系メーリング・リストがドライな感じであることと、大分異なっているという。

加えて、メールソフトを含めたコンピュータ全般に関して「道具そのものにこだわりたい」、すなわち自分でカスタマイズし、メンテナンスしたいという欲求があり、使用するソフトウェアも「その奥が見えないと気持ち悪い気がする」[68]のだという。そこで、電八倶楽部というメーリング・リストを通じてコンタクトをとることができ、信用できる人達だと感じられる人々が開発している電信八号を使用しているのだという。

さらに寺島氏の場合、将来的に開発に参加したいという希望を持っているので、そのための情報、とくに公開版以外のβ版が手に入ることが魅力的であると述べていた。

和田氏もまた、電八倶楽部に優しさを感じている。具体的には、同じような質問に丁寧に返信を返していたり、返信が非常に早かったりすると「なぜそこまで手厚いのだろう」と思うのだという[69]。

また和田氏は、電八倶楽部に参加してメールを読むことにより勉強させられることがあり、同時になぜ他人のためにそこまでサポートできるのかという電八倶楽部参加者のメンタリティに知的好奇心を持っているため、電八倶楽部に参加し続けているのだという。

対照的に本田氏は、電八倶楽部に優しさは感じないという。本田氏が電八倶楽部・電八開発倶楽部に参加しているのは、ヘルパー・アプリに関する情報やβ版が入手できること、自分が使うソフトウェアの開発の方向性に意見を述べる[70]ことができ、不具合が修正されやすいことがあるからだという。また、本来の仕事として行っている困った人を助けることを、気楽にできる場として電八倶楽部をとらえている面もあるという。

という実利的なメリットもある」と述べていた。

[66] 青木氏は 1995、1996 年から電信八号を使用し始めた。当時の雰囲気は、「たとえ問題が生じるとしても最新のソフトウェアを使いたいという欲求が強かった」と述べている。

[67] 電八倶楽部・電八開発倶楽部のメーリング・リストの雰囲気に関しては、福井氏が「電八倶楽部は元になったメーリング・リスト時代から、寺島氏が現在感じているのと同様だった」と述べ、山田氏も「初期の頃から温和な人が多かった」と述べていた。

[68] こうした心境に関しては福井氏も同意していた。福井氏の言葉を借りれば「極論すれば、他人の作ったものは使いたくない」のだという。

[69] こうした「優しさ」は和田氏の仕事にも参考になる点が多いという。

[70] ただし、本田氏自身の自らの位置づけはあくまでもユーザであり、開発に参加している意識はないという。

最後に、ユーザではないがヘルパー・アプリの開発者である山田氏のソフトウェア開発動機について触れておこう。山田氏によれば、「ないものは作る」の精神[71]がソフト開発の基本であるという。言い換えれば、「自分が使いやすいように使いたい」と思って開発をし、その追加的要素として「(他の人に)使って、喜んでもらえると嬉しい」ものだという。加えて、ソフトウェアを開発し、公開したこと自体を評価、承認してもらえることも喜びになるのだという。

これらは電信八号用のヘルパー・アプリ、例えば「サクサク動くビューワとしての電ラブ」「お遊びとしての電極 Z 号化計画」にも当てはまるし、他のソフトウェア、例えば「Windows Popup Biff」などにも当てはまる。とくに、Windows3.1 の時代にはソフトウェアが本当に少ない状況だったので、色々なものを開発したという。

しかし、現在は「自分が作らなくてもあるから…」「もっと若い人に作って欲しい」と思ってしまうため、あるいは「頭に浮かんだコードを実際にエディタに打ち込むことすら面倒になっているので」、開発をあまりしていないという。さらに、開発すること、そしてそれをサポート[72]することは大変なことで余裕がないとできないため、開発をあまりしていないのだという[73]。

５．開発事例のまとめ

本節で述べてきたシェアウェア、フリーウェアの開発事例の主要な点をまとめると、次表のようになる。

表６　シェアウェア、フリーウェアの開発事例のまとめ

	鶴亀メール	電信八号
開発開始の契機	Xaxon 社の NetMail にサイトー企画が秀丸エディタのコンポーネントを供出していた ↓ NetMail の販売終了 ユーザが秀丸エディタ内蔵のメールソフトを要望 ↓ 2000 年 4 月コード記述開始→8 月公開	「わかりやすいメールソフトを自分が使いたい」 「若干の金儲け」
開発プロセス (バージョン・アップ)	ユーザからの要望が始点(サポート会議室・メール) ↓ ソフトウェア本体に実装(バージョン・アップ) マクロで対応(新しい関数の追加) ↓ バージョン・アップ完了後、即時公開 動作テストやデバッグはサイトー企画内では行わない ↓ サポート会議室などでバグ報告、機能追加の要望を受け付ける ↓ 次のバージョン・アップへ	ユーザの要望、期待が始点(メール・会社の同僚) ↓ バージョン・アップ ↓ コンパイル後、即時公開(1996 年頃からβ版として公開することも) 動作テストやデバッグは行わず ↓ ユーザからのバグ報告、修正要望など受け付け ↓ 次のバージョン・アップへ
開発組織	基本的に 1 つのソフトウェア(サービス)を 1 人の開発者が担当 鶴亀メール…斉藤氏 秀丸エディタ…山本氏	初期は石岡氏が 1 人で開発 電八倶楽部移管後は ML による開発 公式ビルダ 4 人体制(ビルドするのは 2 人) パッチの作成は電八開発倶楽部(参加者約 300

[71] これは、事例の冒頭で紹介した福井氏の「UNIX 文化」と非常に似通っている。

[72] 山田氏は現在電八倶楽部に参加していない。参加を辞めるときには「自分のソフトを使ってくれている人に申し訳ない」という思いが生じ、かなりの思い切りが必要だったと述べている

[73] 山田氏は現在、Web 系のシステム構築を手がける会社を経営している。SOHO になったことにより、会社に勤務していた頃に比べ時間的余裕がなくなり、フリーウェアやシェアウェアを開発したくてもできない状況だという。

	コミュニテックス…KON 氏、部分的に斉藤氏 その他の小規模ソフトウェア…斉藤氏	人) デバッグ・動作テストは実質的に ML メンバによる
開発者の履歴	ソフトウェア開発のきっかけ 楽しそうだからやってみよう 雑誌付録のプログラミング・リストの入力など自分で作ってみたいと思うようになる 「自己満足」 「ギターを弾いたりするのと同じ」 プログラミングの仕事をしたくて富士通に入社('88) 実際にはプログラミングの仕事はなかった 「隠れて」ソフトウェア開発 同僚たちに好評 コンピュータのグラフィクス性能の向上 DOS/V、Windows3.0 の登場 「ビジネス・チャンス」と見て開発済ソフトウェアを Windows 環境に移植 Nifty-Serve でシェアウェア・ビジネスが成立 Windows ソフトウェアをシェアウェア公開	
開発のインセンティブ	シェアウェア、フリーウェア開発は自己実現の場 報酬は満足感 開発継続の理由は収入があること	自分が使いたいソフトウェアだから 「若干の金儲け」
開発のモチベーション	自分の作ったソフトウェアの利用者の期待に応えたい (範囲は狭いが＝自作ソフトウェアの利用者間に限られるが)自分の仕事には社会的な価値があるし、評価もされていると感じていること ソフトウェア開発の仕事は好き(ただし、嫌いだけど我慢してやっている仕事が多い) (社会的な)責任感 サポート会議室に寄せられる質問・要望にはきちんと対応すべきだと考えている 現在の自分は「基本的にやる気がない」と思っている 「シェアウェア開発者には基本的にやる気がないのではないか」 学生時代、秀丸エディタ開発初期は「熱かった」(やる気があった)	ユーザの期待に応えたい 公開することによる満足感 うれしさ、「誇り」、生きがい 社会的な貢献(ユーザの役に立っている) 社会的な承認

第 4 章　調査事例の検討 1―開発サイクル―

1．開発サイクルの比較

前節で紹介した事例において、シェアウェア、フリーウェアが商用ソフトウェアと最も明確に異なっていたのは、そのバージョン・アップ・ペースとその内容である。

商用ソフトウェアのバージョン・アップは、機能追加や仕様変更などが行われるメジャー・バージョン・アップが最低 1 年程度、通常 3～4 年程度に掛かる。ただし、その間にサービスパックやセキュリティ・パッチと呼ばれるソフトウェア・モジュールの提供による不具合修正、すなわちマイナー・バージョン・アップが行われているが、このマイナー・バージョン・アップを勘案した場合に漸くそのバージョン・アップ・ペースが数週間程度に縮まる程度である。

他方、シェアウェア、フリーウェアの事例では、長くても 1 ヶ月、短い場合には数日か、数時間でバージョン・アップが行われている。さらに、その内容においても、メジャー・バージョン・アップ／マイナー・バージョン・アップの区別がほとんどなく、バージョン・アップにおいて、不具合修正はもちろんのこと、機能追加や仕様変更まで実現されている。

では、このようなバージョン・アップのペースとその内容の違いは何故生じるのであろうか。本稿ではこれ以降、それらの背後には「開発サイクル」の違いがあるためであるという前提に立って議論を進めていくことにする。

そのためにまず本節では、事例と既存研究の知見に基づいて、シェアウェア、フリーウェアと商用ソフトウェアの開発サイクルの比較を行う。続いて、その比較を通じて浮かび上がってきた 2 つの開発サイクルの違いが何故生じるのかについて考察する。

具体的には、まず第 3 章の事例に基づいてシェアウェア、フリーウェアの開発サイクルがどのようなものであるか、その特徴を中心に分析する。続いて、既存研究の知見を踏まえ、シェアウェア、フリーウェアの開発サイクルと比較対照するかたちで、商用ソフトウェアの開発サイクルを描く。

2．シェアウェア、フリーウェアの開発サイクル

(1) ケースから読み取れる開発サイクルの 3 つの特徴

事例から読み取れるシェアウェア、フリーウェアの開発サイクルの特徴として、大きく 3 つの点が挙げられる。

その第 1 は、「組織化されたユーザからのフィードバック」がある点である。ここで「組織化されたユーザ」、あるいは「ユーザの組織化」[74]とは、以下の 3 つの要素を含んでいる。

まず、(a)ユーザが開発者と 1 対 1 の関係を結べることである。次に、(b)こうした 1 対 1、直接的なユーザと開発者の関係が緊密で強い、すなわち、ユーザと開発者の間で頻繁な情報のやり取りが行われていることである。さらに、このようなユーザと開発者との直接的で強い繋がりの有無のために、(c)開発に積極的に関与するような、いわば「イノベーティブ・ユーザ」が存在することである[75]。

例えば、鶴亀メールでは、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で始まるとされている。同ソフトウェアの場合、ユーザからの要望やバグの報告は、サイトー企画が運営するサポート・フォーラムが主たる受付窓口になっており、そこへのアクセス、発言のための会員登

[74] 厳密に言えば、「ユーザの組織化」には、2 つの種類があると考えられる。本研究で取り上げるのは、開発主体(企業)が主導する「(開発主体主導による)ユーザの組織化」である。だが、この他に、ユーザの中の 1 人もしくはグループが中心となって組織化をする事例もあり得る。こうした組織化は、「(ユーザ主導の)ユーザの組織化」、あるいは「ユーザの自己組織化」と呼ぶべきであろう。ユーザの自己組織化の具体例としては、音楽や文学の世界における「ファンジン(fanzine)」の存在と活動が挙げられる。

[75] 例えば、電信八号の事例では、メーリング・リスト参加者の中に優秀なデバッガが存在するとの記述があったが、このようなユーザが③の例としてあげられるだろうし、事例の他の箇所でも、こうしたフィードバックが積極的に行われていることが読み取れる。

録、閲覧は自由にできる[76]ようになっている。そのため、ユーザは文字通り気楽にサポート・フォーラムにアクセスし、要望やバグの報告などができるようになっていると言えるだろう。

電信八号の場合にはこの特徴がさらに顕著である。電信八号では、自由に参加、退出ができるメーリング・リスト、電八倶楽部が用意されており、それを通じてソフトウェアに関する要望、バグ報告はもとより、ソフトウェアに関するその他の情報を発信し、閲覧することができるようになっている。

ユーザが意見や要望、バグ報告などを寄せる積極性もケースに表れている[77]。例えば、鶴亀メールでは、「ユーザが他のソフトウェアを参考にして機能追加の要望を提示してきた」結果、鶴亀メールが非常に多機能なメールソフトになった」という記述があった。また、電信八号の場合、「メーリング・リスト参加者の中に優秀なバグを発見し、報告してくれる人材がいることが電信八号の開発に大きな貢献をしている」という発言があり、インタビュー参加者が実際に、具体的な要望やソフトウェアに関する情報を発信したという記述[78]もあった。

シェアウェア、フリーウェアの開発サイクルに見られる第 2 の特徴は、テスト、デバッグ作業が開発主体、すなわち、サイトー企画や電八倶楽部でほとんど行われていない点である。

鶴亀メールの場合、斉藤氏が「テストやデバッグをサイトー企画が行うことはない」「開発したものはすぐに公開する」と述べている。電信八号に関しては、福井氏が公式ビルドの作成に当たって「他人に使ってもらうものなので、致命的なバグがでないように留意している」と述べていたのに止まり、テストやデバッグを電八倶楽部内、電八開発倶楽部内で行われているとは述べられていなかった[79]。

最後の第 3 の特徴は、ユーザからのフィードバック、要望やバグ報告が開発に反映されるにあたり、ほとんど選別されていない、つまりスクリーニングを経ない、という点である。

鶴亀メールの場合には、斉藤氏が極力ユーザからの質問、要望に応えていると述べており、機能追加の要望に応えるに当たって鶴亀メール本体かマクロかという選別は行うものの、基本的に全ての追加の要望を受け入れている。そのため、「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」という思いを抱くことすらあるのである。

また、電信八号の場合、図 4 に示されたように、ユーザなどから寄せられる要望や報告などは、WishList に全て反映される。WishList がパッチ作成に結びつくか、WishList を経ないパッチが作成されるかは、開発能力を持つパッチ作成者の興味と意欲によるが、少なくとも、ユーザなどから寄せられる要望や報告などが、何らかの意図や基準を有した一定の主体によってコントロールされ、選別される、すなわちスクリーニングされることが無いことは明らかである。さらに、そうしたスクリーニングがしない方針が、パッチを取りまとめる公式ビルド作成の段階でも貫かれている。公式ビルダの 1 人である福井氏が「(極力)機械的に」パッチのとりまとめを行う、と述べていることがその現れである。

(2) 開発サイクルに見られた 3 つの特徴の相互依存関係

加えて、さらに重要なことは、前述した開発サイクルの 3 つの特徴が、相互に依存的な関係にあることである。

---

[76] 2003 年 1 月 24 日付で、鶴亀メールや秀丸エディタなどに関するサイトー企画とユーザの過去のやりとり(過去ログ)を閲覧できるようになり、ソフトウェアに関する情報の閲覧がさらに容易になった。

[77] この点は各々のホームページ(鶴亀メール：http://hide.maruo.co.jp/ 、電信八号：http://denshin8.esprix.net/)などを見ても確認することができる。

[78] 例えば、本田氏の「RFC 準拠であるならば、大文字と小文字を区別せずに処理すべきだ」という改善の要望や、和田氏が初心者が電信八号を利用するとどのようになるかについて報告したことなどが挙げられる。

[79] ただし、このことは、テストやデバッグといった作業を全く行わないことまでは意味しないだろう。例えば、鶴亀メールの開発主体であるサイトー企画には、開発用の機材の中にテスト用のマシンも含まれていた。したがって、シェアウェア、フリーウェアのテスト・デバッグも、動作確認などの基本的な部分は行われているが、それが、商用ソフトウェアのように、専門の部門や人員、ツールなどを使って行われていない、という意味において「テスト・デバッグを開発主体が行わない」という発言を解釈すべきだろう。

まず、ユーザが組織化され、そのように組織化されたユーザからのフィードバックが得られるのは、自分の発信した要望やバグ報告がスクリーニングされず、何らかの形でソフトウェアの機能向上、不具合の修正に反映されるだろうという楽観的見通しがその背景にある。換言すれば、たとえ些細なことでも、用意された簡便な発信手段―鶴亀メールの場合のサポート・フォーラム、電信八号の場合の電八倶楽部―を用いて発信すれば、開発主体―サイトー企画および電八開発倶楽部―の目に触れることになり、ソフトウェアが「より良くなる」という見通しがあるからこそ、組織化されたユーザが、積極的にフィードバックをするのであろう。

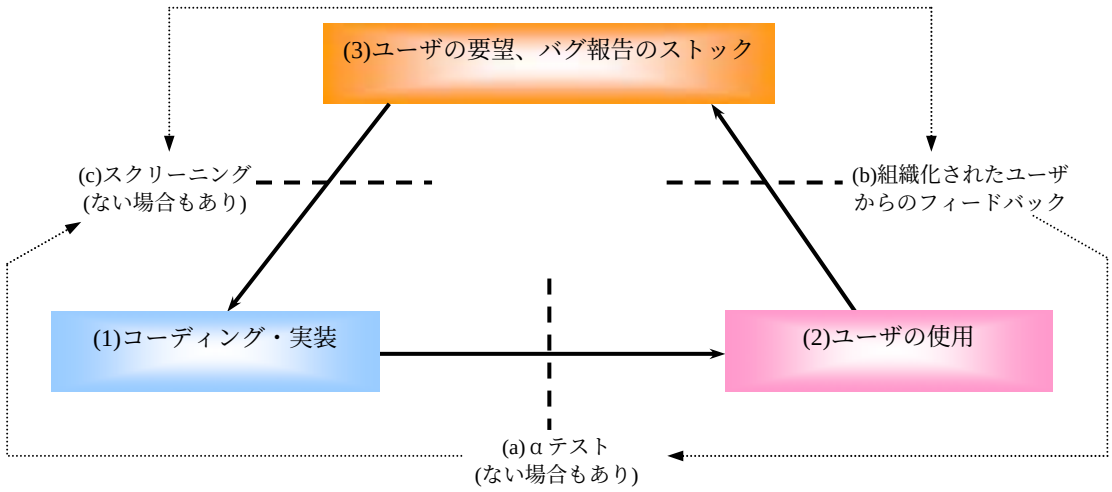
次に、組織化されたユーザからの積極的なフィードバックが期待できるからこそ、開発主体がテストやデバッグをせずにソフトウェアを公開することができる。言い換えれば、バグ、不具合などは組織化されたユーザの積極的なフィードバックによって発見されるであろうから、次回のバージョン・アップで修正すればよいという見通しにたった公開、つまり「修正を前提とした公開」が行うことができるため、開発主体が自らテストやデバッグをしないのであろう。なお、「修正を前提とした公開」と対照的なのが、企業が商用ソフトウェアを公開、発売するに当たって取っている姿勢である。例えば Impress 記事(2003) によれば、マイクロソフト社はソフトウェアのバグ、セキュリティ・ホールを修正するためのパッチの公開に関して、「…不十分な検証のままパッチを提供し、『パッチにパッチを当てる』ような状態にはしたくない。そのような不完全なパッチは、ユーザにとって手間がかかるだけで無意味だともいえる」と述べている。

最後に、開発主体は自らテストやデバッグをしないでユーザによる評価、不具合の発見などを期待しているため、スクリーニングを行わないのであろう。つまり、最終的に公開されたバージョンのソフトウェアを使用してみて、ユーザが選択、判断、評価をし、新機能の善し悪しやバグの有無を報告してくれる見通しが立っているため、開発主体自身はスクリーニングをしないで済むのであろう。さらに、開発姿勢やソフトウェアの基本的方向性において共感している組織化されたユーザがフィードバックしてくれ、さらにそうしたユーザの中にはコンピュータやプログラミングに精通した人も多く、提出される報告や要望にすでにある程度のスクリーニングがかかっている可能性もある。

(3) シェアウェア、フリーウェアにおける開発サイクルの全体像

以上述べてきたような、シェアウェア、フリーウェアの開発サイクルの特徴と、その諸特徴間の相互依存関係を、図 2 として示した開発サイクルの基本的概念図に当てはめて図示すると、下図のようになろう。

図 6　シェアウェア、フリーウェアの開発サイクル



※　――→ は活動の連鎖、┈┈→ は依存関係を表す。
※　簡略化のため、開発者(企業、集団、個人)が独自に盛り込む新規なアイディア、機能・仕様については記述しない。

すなわち、シェアウェア、フリーウェアの開発サイクルの全体としての特徴は、開発サイクルを構成する 3 つの活動―コーディング・実装、ユーザの使用、ユーザの要望・バグ報告―を結ぶ段階において、1 つの活動から他の活動への移行を妨げるような活動、いうなれば「関門」が存在しないか、存在しても非常にその存在が希薄な(“Loose-Gated”)点にある。

さらに、このような開発サイクルが機能している結果、シェアウェア、フリーウェアでは、表 3 および表 5 で示したように、非常に頻繁にバージョン・アップが行われ、その機能が迅速に向上していると考えられる。

つまり、シェアウェア、フリーウェアの開発サイクルは、「関門のない、相対的にスピードの速い開発サイクル」(Relatively-Rapid Loose-Gated Development Cycle)であるといえるだろう。

３．商用ソフトウェアの開発サイクル

では、こうしたシェアウェア、フリーウェアの開発サイクルを念頭に置いて、商用ソフトウェアの既存研究、特に現在主流となっているパッケージ・ソフトウェアについての既存研究を再検討し、その開発サイクルを記述してみると、どのようになるであろうか。

まず、商用ソフトウェアの開発サイクルの特徴を、やはり 3 つ挙げると、以下のようになろう。第 1 に、コーディング・実装といったソフトウェアの開発プロセスが完了し、それが公開βテスト版、あるいは製品としてユーザに提供される前に、α版テストや社内βテストなどと呼ばれる開発主体(これは基本的に企業である)による綿密なテスト、デバッグが行われることが挙げられる。

第 2 に、そうした公開βテスト版、あるいは製品をユーザが使用した後、そうした使用体験から得た要望やバグなどの報告などを行うユーザが組織化されていないことが挙げられる[80]。すなわち①’ユーザからのフィードバックは、基本的にサポートセンターと呼ばれる部門が受け付けている。しかも、サポートセンターはサポート専門の部隊なので開発には直接的にはタッチしていない。つまり、ユーザと開発者(開発部門)の間にサポート部門が介在し組織的バッファとなっている。また、②’ユーザと開発者の関係が商用ソフトウェアでは稀薄で弱い。さらに、③’開発に積極的に関与するイノベーティブ・ユーザの存在は期待しにくい。

第 3 に、そのようにして寄せられたユーザからの要望やバグなどの報告が、開発主体によって選別、取捨選択、すなわちスクリーニングされて、次バージョンのコーディング・実装に反映されることが挙げられる。

そして、こうした商用ソフトウェアの開発サイクルに見られる 3 つの特徴は、シェアウェア、フリーウェアの場合と同様、やはり相互依存的な関係にあるといえる。

まず、組織化されていないユーザ、すなわち、どのようなユーザが公開βテスト版や製品を使用するのか分からない状況に開発主体が置かれているため、どんなユーザが使用しても問題が生じないような水準に達した公開βテスト版や製品を出さなければならないという、ややもすると過剰気味な設計品質目指すような意識を開発主体が持つ。換言すれば、機能不足やバグを可能な限り含まない公開βテスト版、あるいは製品版を出そうと開発主体が意識する。そのために、開発主体の内部でα版テストや社内βテストが厳重に繰り返し行われることになる。

次に、開発主体内部でのα版テストや社内βテストを厳重に繰り返し行わなければならないこと

[80] 商用ソフトウェア(および一般の製品)のユーザの使用体験のフィードバックは、それを発売した企業に直接報告されるのではなく、ユーザ間の情報交換を目的としたホームページや掲示板などに報告されることが多い。そうしたホームページや掲示板は、開発主体以外の企業が設置したものや、ユーザの中の有志が設置したユーザサイト、ファンサイトなどである。ホームページや掲示板の設置主体の如何に関わらず、これらが何れも開発主体にとって直接的に情報を得るための窓口になっていない点では共通している。前述の用語法を使って換言すれば、商用ソフトウェアの場合ユーザの組織化は行われないか、行われたとしても「ユーザの自己組織化」に留まり、シェアウェア、フリーウェアのように「開発主体主導によるユーザの組織化」には至っていないと言えるだろう。

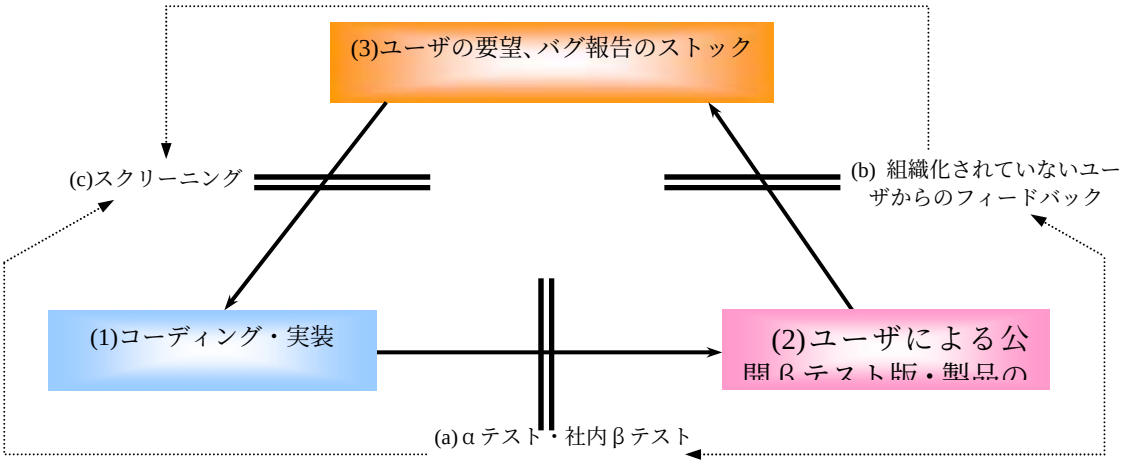
このようなユーザの組織化の有無、あるいはそのあり方の違い(開発主体主導であるか否か)のため、商用ソフトウェアにおいて開発主体が直接受け取ることができるユーザからのフィードバックは、量的にも内容的にも限られたものになる傾向が生じるであろう。

が前提となるため、ユーザからの要望やバグ報告の取り扱いに慎重になり、それにスクリーニングが不可避となる[81]。

最後にユーザが組織化されたおらず、様々な、場合によっては全く的はずれな要望やバグ報告も寄せられる可能性があることからも、コーディング・実装の前段階としてスクリーニングが必要となるであろう。

このような商用ソフトウェアの開発サイクル特徴と、その諸特徴間の相互依存関係を、図 2 として示した開発サイクルの基本的概念図に当てはめて図示すると、次のようになろう。

図 7　商用ソフトウェアの開発サイクル



※　⟶ は活動の連鎖、⇢ は依存関係を表す。
※　簡略化のため、開発者(企業、集団、個人)が独自に盛り込む新規なアイディア、機能・仕様については記述しない。

すなわち、商用ソフトウェアの開発サイクルの全体としての特徴は、開発サイクルを構成する 3 つの活動を結ぶ段階において、1 つの活動から他の活動への移行を妨げるような活動、「関門」が厳然と存在する(“Rigid-Gated”)点にある。

さらに、このような開発サイクルが機能している結果、商用ソフトウェアでは、シェアウェア、フリーウェアに比べバージョン・アップに時間が掛かり[82]、その機能がゆっくりとしたペースでしか向上しないと考えられる。

つまり、商用ソフトウェアの開発サイクルは、「関門のある、相対的にスピードの遅い開発サイクル」(Relatively-Dull Rigid-Gated Development Cycle)であるといえるだろう。

4．開発サイクルの違いについてのまとめ

以上、事例に基づいて、シェアウェア、フリーウェアと商用ソフトウェアを開発サイクルという視点から比較することを試みた。

コーディング・実装、公開・ユーザによる使用、ユーザからのフィードバックのストック、新たなコーディング・実装という連続的・螺旋的な循環、すなわち開発サイクルに関し、シェアウェア、

---

[81] こうした商用ソフトウェアの状況をシェアウェア、フリーウェアの開発サイクルと対比的に見れば、シェアウェア、フリーウェアの開発サイクルが将来、すなわち開発サイクルの次の活動に関する楽観的な見通しによって支えられ、循環的な運動をしているのに対し、商用ソフトウェアェアの開発サイクルは、悲観的な見通しによって支えられているといえるだろう。

[82] ただし、シェアウェア、フリーウェアと商用ソフトウェアェアの間でバージョンアップの頻度が異なるのは、単に開発するソフトウェアの規模の違いによるという説明も成り立ちうるだろう。確かにそうした要因は重要な要因であり、後述する開発サイクルの違いを生む要因((2)試行コストの違い)で考慮されている。だが、それ以外の要因がそれ以上の影響を及ぼしてバージョンアップの頻度を含めた開発サイクル全体の違いを生み出しているというのが本研究の見解である。

フリーウェアでは、これらの各活動の間に次の活動への移行を妨げるような活動、「関門」が存在せず、結果として開発サイクルが速いペースで回転する。他方、商用ソフトウェアでは、活動間に「関門が」が存在するため、開発サイクルの回転ペースは比較的遅くなる、というのが事例に基づく開発サイクル比較の結果である。

第５章　調査事例の検討 2―開発サイクルの規定因―

では、こうした開発サイクルの相違はどのような要因によって生じているのだろうか。続いてこのことを検討してみよう

１．「ソフトウェア像―開発主体のソフトウェアに対する認識―」の違い

開発サイクルの相違を生む要因としてまず挙げられるのは、開発主体(企業、集団、個人)が持つ開発対象、すなわちソフトウェアに対する認識である。

商用ソフトウェアの場合、ソフトウェアはユーザがそれに対して代価を支払うべき、そしてそれに値する価値を有する「製品」である。したがって、開発主体はソフトウェアの機能とその完全性を一定以上のレベルにする義務があると認識している。さらに、こうした義務を遂行しなかった場合、訴訟によって損害賠償請求をされたり、企業イメージ、製品ブランドが低下したりする危険性、あるいは企業の社会的責任責任が果たされないなどといった問題が生起する危険性も認識されている。

他方、シェアウェア、フリーウェアの場合、ソフトウェアはユーザ自身がその入手、インストール、使用に当たって責任を持って当たるものである、ユーザ・サポーテッド・ソフトウェア(User-Supported Software)であるとされている[83]。換言すれば、開発主体が機能とその完全性を一定以上のレベルにしなければいけない義務を負ってはいないと認識している。

このような開発主体のソフトウェアに対する認識の違いは、特にユーザの使用に供されるソフトウェア―公開版、公開β版、製品版など―の機能とその完全性、すなわち、ソフトウェアの「完璧性(保証性)」に反映される。そして、このソフトウェアの「完璧性(保証性)」を確保するために、その前段階にあたる活動であるα版テストや社内βテストによるテストの有無とその厳密性を左右と考えられる。

具体的に述べれば、ユーザの使用に供されるソフトウェアの完璧性(保証性)が高いことが必須であると考えられている商用ソフトウェアの場合、α版テストや社内βテストが行われ、しかもそれが厳密になる。他方、そうした完璧性(保証性)が必須とは考えられていない、つまり不完全性(無保証性)が許されるシェアウェア、フリーウェアでは、α版テストや(開発主体内部の)βテストは行われず、行われても簡略なものとなる。

２．「ユーザ像―ユーザに関する認識―」の違い

(1) 想定するユーザ像の違い―「3 人称の開発」と「1 人称の開発」―

商用ソフトウェアとシェアウェア、フリーウェアの開発サイクルを異なるものにさせる第 2 の要因は、各々のソフトウェアの開発主体が有するユーザに対する認識、すなわち開発において想定する「ユーザ像」における違いである。商用ソフトウェア、シェアウェア、フリーウェアとも、結果的に、多くの不特定多数のユーザを対象に開発、配布が行われ、多くのユーザに使用されるにもかかわらず、その開発契機、および開発過程において想定されるユーザ像が異なっている。このことは、製品開発において「外的統合」(クラーク・藤本, 1991)を実現するための前提条件が異なっている、と言い換えることもできる。

商用ソフトウェアの場合、開発に際して平均的なユーザ、一定の人数のユーザ層が想定される。この「平均的」ユーザを「想定」し、ソフトウェアに盛り込まれる新機能や既存機能の改良方針を示すのが、商用ソフトウェアにおけるコンセプト創造であり、そのようにして創り出されたコンセプトに基づくソフトウェア(製品)開発は、ユーザの使用体験を「シミュレート」する活動となる[84]。

このようなコンセプト創造とそれに基づくソフトウェア開発は、開発主体が自分自身、あるいは

---

[83] シェアウェア、フリーウェアの付属文書(Readme.txt)には、免責事項(「このソフトウェアに不備があっても作者は訂正する義務を負いません。」「本プログラムを使用した結果いかなる損害が生じても、作者は責任を負いかねます」などという趣旨の項目)が必ず明記されている。

[84] クラーク・藤本(1993)。

現実に直面する主体に依存せず、想定上の主体・ユーザを念頭に進める活動である点において、「3人称の開発」と呼ぶことができるだろう。そして、こうした「3 人称の開発」、およびその成果物であるソフトウェアの普及の正否は、平均的なユーザ像が現実のユーザ層とどれほど一致するか、つまり想定の精度によって決まることになる。したがって、想定する平均的なユーザ像がなるべく多くのユーザをカバーするように、製品コンセプトとそれに基づくシミュレート＝ソフトウェア(製品)開発は多機能指向、ローリスク・ローリターン指向になりやすい。

他方、シェアウェア、フリーウェアの開発では、「実在のユーザ」(しかも多くの場合は自分自身の欲求[85])を「実感」し、それを満足させるためにソフトウェアが構想され、開発が開始される[86]。そのため、少なくとも 1 人のユーザは確実に存在し、その極めて具体的な要望・欲求に応えるためにソフトウェアが開発される。このようなコンセプト創造、ソフトウェア開発のあり方は、開発主体が自分自身あるいは現実に直面する主体に依拠することから、「1 人称の開発」と呼ぶことができるだろう。

こうした「1 人称の開発」では、同様の要望・欲求を抱えるユーザ数の多寡、要望・欲求の共感の程度が開発とその成果物であるソフトウェアの普及の正否を左右する。

同時に、「1 人称の開発」の開発では、自分自身あるいは現存するユーザに応えることを主眼において開発が開始されるため、当初のソフトウェアはシンプルなものになりやすい。換言すれば、そのソフトウェアは単機能指向である。だが、出発点がシンプルで単機能指向であるが故に、新機能の実装や既存機能の改良には積極的であり、ハイリスク・ハイリターン指向になりうる[87]。

なお、ここで本研究が「1 人称の開発」と概念を新たに提示し、既に定着している「ユーザ・イノベーション」(von Hippel, 1988)という概念を使用しない理由は、両者の間に微妙な違いがあるためである。

両者の違いを挙げると、ユーザ・イノベーションでは「イノベーターは誰か？」「ユーザを含めたイノベーターが共同して(組織化されて)イノベーションを実現するのはどのような状況においてなのか？」といった問題意識から出発し、ユーザとイノベーションの完遂者が独立した状況を暗黙のうちに前提にして、両者が協働することができるか否かが重要な論点となっている。

他方、「1 人称の開発」の開発ではユーザとイノベーターが一致している状況を前提にし、そうした状況で行われるイノベーション(開発)プロセスの特徴に着目している。より具体的には、シェアウェア、フリーウェアに見られた「1 人称の開発」の場合、①当初の開発がユーザ自身によって行われている、②オンライン上で全ての情報の受発信が行われるため、開発「組織」が自由に広がる可能性が高く、実際に広がりを持っている、③インターネットという経路を用いて情報(ソースコード、あるいはそれに近接したインターフェースなど)の効果的な共有が行われているため「情報

---

[85] 「我々は手始めに、毎日見ているものを願望する」(Harris, 1988)とすれば、自己の欲求は、ユーザでありプログラマである人物が他のソフトウェアを使用し、見ることから自己の欲求を育てると考えられる。シェアウェア、フリーウェアを含めたフリーソフトウェアの多くがその模倣対象を持つという、これと同様の主張は、高橋・高松(2002, p.16)でもなされている。

本研究が記述した事例でも、鶴亀メールは(事後的にではあるが)斉藤氏がユーザに促される形で他のソフトウェアを参考にしてバージョンアップをしており、また電信八号の場合には原作者の石岡氏が UNIX 環境で広く使われていた MH (Mail Handler) を意識して開発したと述べられていた。

[86] 本研究で記述した事例の中では、鶴亀メールが NetMail の潜在的なユーザが、サイトー企画(斉藤氏)に「メールソフトを何とかできないか」、「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」という非常に具体的な要望が開発の端緒となっていた。また、電信八号の場合は、原作者の石岡氏自身が当時提供されていたメールソフトが「非常に使いにくい」ことに不満を感じ、「自分が使いやすい Windows 用メールソフトを作ろう」と考えたことが開発の契機であった。

[87] また、単機能指向が基本であるシェアウェア、フリーウェアが、いつまでも単機能指向で止まるとは必ずしも言えない。単機能の積み重ねの結果、多機能化することもあり得るからである。しかしながら、そのようにして多機能化したシェアウェア、フリーウェアは、当初から多機能指向の商用ソフトウェアエアとは、その発展プロセスや構造において違いが見られるであろう。

なお、発展プロセスにおけるこうした違いは、Raymond (1997) が「伽藍」と「バザール」として象徴的に述べた違いと通底するものであると考えられる。

の移転コスト」が非常に低くなっている、といった点が特徴的である。

さらに言えば、von Hippel は「ユーザ・イノベーション」という概念を提示したが、(書名にも現れているように)彼の研究上の関心は、イノベーションの源泉がどこにあるかに焦点が置かれており、その研究成果としてユーザを見出して、ユーザ・イノベーションという概念を提示した。言い換えれば、von Hippel が主張したのは、イノベーションの源泉・起点となった主体がユーザであるということであり、その実現(インプリメンテーション)までがユーザによって行われたとは主張していない。他方、我々が本研究で見出した「1 人称の開発」は、イノベーションの源泉のみならず、それを実現する主体までがユーザであるという概念である。この観点から von Hippel のユーザ・イノベーションを捉え直すと、彼が調査事例から見出したのは、ユーザの具体的な要望に対して、それと直接の関わりを持つ異なる主体がイノベーションを実現した、という事例である。これは、ユーザとそれを実現する主体が分離しつつ、かつ両者が直接的関係を有していることから、「2 人称の開発」と呼ぶのが妥当であると考えられる。ソフトウェア・システム開発の分野で「2 人称の開発」に該当するものを挙げるとすれば、Cusumano (1991)が対象としたカスタマイズド・ソフトウェアなどが考えられるであろう。

(2) 「ユーザの組織化」とサポート体制が影響を及ぼすフィードバックの違い

商用ソフトウェアとシェアウェア、フリーウェアの間に見られるユーザに対する認識の違い、ユーザ像の違いは、前項で述べたソフトウェア像の違い、すなわち開発主体の開発対象に対する認識—「製品」か「ユーザ・サポーテッド・ソフトウェア」か—の違いと相俟って、商用ソフトウェアとシェアウェア、フリーウェアでは、開発「組織[88]」の広がりに違いがある、つまり前項の開発サイクルの相違点として指摘した「ユーザの組織化」の程度の違いが生じている、という現象を説明できるようになる。

商用ソフトウェアでは、ソフトウェアは製品である。したがって、ソフトウェアはそれが公開される時点で十分な機能と完成度を達成しているはずであり、たとえそうでないとしても、それは開発主体によって補われるべきであると認識されている。このことは同時に、ユーザが基本的に別主体、第 3 者であり続けることを意味する。したがって、こうした認識に基づくパッケージソフトでは、ユーザの開発関与には消極的であり、そのためユーザの組織化がなされることへの抵抗があるといえるだろう。

他方、シェアウェア、フリーウェアの場合、ユーザ・サポーテッド・ソフトウェアであるため、開発主体は、公開されたソフトウェアが不完全である可能性があること、そして、それが開発主体自身によって、あるいはユーザによって補われ、修正される可能性があることが認識されている。同時にユーザの側も自己責任でソフトウェアを入手、使用することが求められている。そのため、ユーザが不具合や機能不足に直面した場合、開発主体にそれをフィードバックしたり、自ら追加のコーディングを行ったりするなどして、自ら積極的に関与してそれを解決しようとするだろう。言い換えれば、ユーザ・サポーテッド・ソフトウェアというソフトウェア像は、ユーザの役割を単なる「使用者」に留めず、ユーザがそのソフトウェアを使い続けたい、より機能が向上して欲しいと思うのであれば、動作テストやデバッグといった開発活動の一部を引き受け、開発者へのフィードバックを行わざるを得ない存在にする、と言えるだろう。

このようなソフトウェアに対するの認識、および開発主体とユーザの認識が、ユーザが開発への関与に前向きな状況、ユーザの組織化がなされ易い状況を創り出す。換言すれば、ユーザ・サポーテッド・ソフトウェアという前提に立つシェアウェア、フリーウェアでは、1 人称の開発であることにより、開発組織が当初の開発主体・開発者に限定されず、より広範なコンピュータ・ユーザにまで広がりうる、開発組織が拡張されうる素地が十分に備わっている[89]。

---

[88] ここでの「組織」の定義および、以下での企業の定義は、高橋(1995; 2003)、高橋編(2000)によっている。

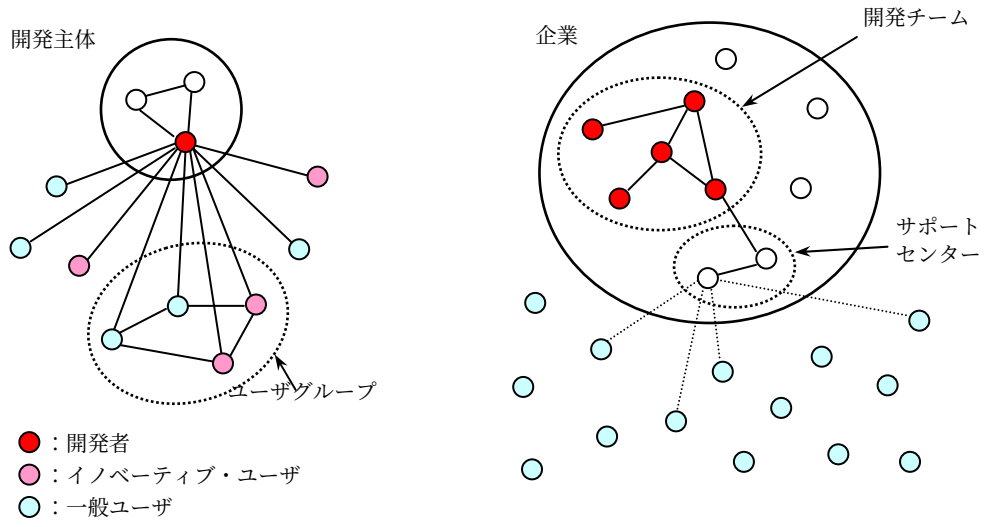
[89] さらに言えば、シェアウェア、フリーウェアでは、情報の発信力・発信意欲をもつユーザの割合が大きく、そうした力と意欲を持たないユーザの割合が小さい(“Loud Majority and Silent Minority”)であるのに対し、商用ソフトウェアでは、情報の発信力・発信意欲をもつユーザの割合が小さく、そうした力と意欲を持たないユーザの割合が大きい(“Silent Majority and Loud Minority”)という違いがあると言えよう。

さらに、こうした商用ソフトウェアとフリーウェア、シェアウェアのユーザの組織化における違いを、実質的な違いに昇華させているのが、両者のサポート体制の違いである。

流通経路をインターネットのみに求めるシェアウェア、フリーウェアでは、サポート活動もインターネット経由で行われることがほとんどである。その理由はいくつも考えられる。例えば、シェアウェア、フリーウェアの開発者がソフトウェア開発を専業としていないことも多いため、サポートセンターを設置すること自体が現実的ではないこともある。専業であっても、ソフトウェアの規模やユーザ数がそれほど大きくはないので、サポートセンターの設置がコスト的に見合わないこともあるだろう。また、開発自体を個人や少人数のチームで行っていることも多いので、開発者がサポートも担当した方が効率的であるということも考えられる。

鶴亀メールでも電信八号でも、開発者への受付窓口はインターネット上に設置された電子会議室やメーリング・リストで、ユーザはそれらのシステムに自由に参加できるようになっている。これらのシステム上でユーザは、バグ報告、修正や機能追加の要望を送信するだけでなく、他のユーザの発言も閲覧することができる。実際にこれらのソフトウェアのホームページを見てみると、膨大な数の意見や要望が提出されていることがわかる。そしてこうした報告や要望に対して、開発者が自ら回答したりコメントを付したりしているものも多い。また、ユーザが開発者と直接やりとりすることができるため、コンピュータとソフトウェアに関する知識レベルが高いユーザなら開発に関与することすらできる。

図 8　ユーザの組織化



他方商用ソフトウェアでは、このようなユーザからのフィードバックは、基本的にサポートセンターと呼ばれる部門が電話で受け付けている。しかも、サポートセンターはサポート専門の部隊なので開発には直接的にはタッチしていない。つまり、ユーザと開発者(開発部門)の間にサポート部門が介在し、組織的バッファとなっている。

つまり、(a)シェアウェア、フリーウェアではユーザが開発者と 1 対 1 の関係を結べるのに対して、商用ソフトウェアではサポートセンターを介して間接的にしか結びついていない。そのため、ユーザと開発者の関係が、(b)シェアウェア、フリーウェアでは緊密で強いのに対して、商用ソフトウェアでは稀薄で弱いと考えられる。さらに、このようなユーザと開発者との直接的で強い繋がりの有無は、ユーザの開発への参加、関与の有無と関係があると考えられる。すなわち、ユーザが開発者と直接やりとりできるシェアウェア、フリーウェアでは、(c)開発に積極的に関与するような、いわば「イノベーティブ・ユーザ」が存在するが、商用ソフトウェアではそうしたユーザの存在は期待しにくい。電信八号の事例では、メーリング・リスト参加者の中に優秀なデバッガが存在するとの

記述があったが、このようなユーザが(c)の例としてあげられるだろう。

このようにソフトウェア像とユーザ像、および後者と関連するサポート体制の違い[90]の結果、「ユーザの組織化」の程度が商用ソフトウェアとシェアウェア、フリーウェアでは異なっていると理解することができる(図 8)。シェアウェア、フリーウェアではユーザの開発への関与度が高く、開発組織が開発者個人やグループの境界を超えて多くのコンピュータ・ユーザまで広がっている、つまりユーザの組織化の程度が高い。他方、商用ソフトウェアではユーザの開発関与度は低く、開発組織は開発主体である企業の境界内で完結しているか、ごく一部のユーザがたまに参加するにとどまる、つまりユーザの組織化の程度が低い。

結果として、こうした想定するユーザ像の違いを根本に持つ、ソフトウェア開発のコンセプト創造と開発プロセスのあり方、ユーザの組織化の違いは、開発サイクルにおいてユーザからの要望、バグなどの報告、およびその前段階に当たるユーザからのフィードバック、その参加者の広がりと寄せられる意見の多様さに影響を与えると考えられる。

具体的に述べると、当初から多機能指向、ローリスク・ローリターン指向で、ユーザの組織化の程度が低い商用ソフトウェアの場合、ユーザは不足している機能や不具合を見いだしにくく、たとえそれを見出してもそれを開発主体に報告しにくい、いわば「つっこみにくい」状況になっている[91]。結果として、ユーザからのフィードバックのストックは蓄積しにくくなり、次バージョンのソフトウェアのコーディング・実装にとって有用なフィードバックを、開発主体である企業がその中に見出すことが難しくなっているのであろう。

対照的に、当初は単機能指向であり、それ故にハイリスク・ハイリターン指向であり、さらにユーザの組織化の程度が高い、シェアウェア、フリーウェアの場合、ユーザが不足している機能や不具合を見いだしやすく、しかも見出した機能の不足や不具合、バグを開発主体に報告しやすい、いわば「つっこみやすい」状況が生じる。そのため、そのソフトウェアの基本的機能や開発の方向性、開発姿勢などに賛同できたユーザが積極的に意見を提示してくれる可能性が高く、ユーザからのフィードバックのストックが形成されやすいのだろう。

---

[90] ここで述べたシェアウェア、フリーウェアの場合のサポート体制は、「オープンなサポート体制」と呼ぶことができるだろう。本稿の立場は、「オープンなサポート体制」こそ、ユーザの組織化とそれに伴う開発組織の広がりにとって必須であるという立場である。

他方、一連の Linux に関する研究では、特に OSS に関する研究では、「オープンソース」という言葉に代表される、ソフトウェアに関する情報の提供がなされること、換言すれば「オープンな情報提供」がソフトウェア開発のあり方に大きな影響を与えるとの論調が強い(Raymond (1997)、Dibona, Stone & Ockman (1999)、佐々木・北山(2000))。

しかしながら、本研究の事例も含め、シェアウェア、フリーウェアの多くが、(少なくともオープンソースではない)オープンな情報提供がなされていない状況下、マイクロソフト社の Windows 用ソフトウェアとして開発されている現状を踏まえれば、オープンな情報提供が必ずしもソフトウェア開発のあり方に大きな影響を与えるとは言い難いのではないだろうか。つまり、Windows ようにクローズドソースであり、その情報提供は API (Application Program Interface)程度であっても、開発能力を有したユーザを惹き付け、彼らにシェアウェア、フリーウェアというソフトウェア開発をさせているという事実は、「オープンソース」という言葉が現在過大評価されているのではないかという疑問を抱かせるに足ると思われる。

同様の主張は、「無償オープンソース」を別の観点から考証し、それがソフトウェア開発(Linux 開発)の必要十分条件では無かったことを明らかにした、つまり Raymond らによるオープンソースの過大評価を是正しようとした高橋・高松(2002)と通底するものがあると考えられる。

我々の立場を改めて明らかにすれば、(オープンソースを含めた)オープンな情報提供だけでは、つまりオープンなサポート体制を欠いていては、ユーザの組織化はなされず、したがって開発サイクルというレベルでのソフトウェア開発のあり方の変化は現出しない。オープンな情報提供は、オープンなサポート体制を伴ってこそ、ユーザの組織化を現出させ、開発サイクルというレベルでのソフトウェア開発のあり方の変化を引き起こすが、オープンなサポート体制を欠いてしまえば、開発主体を中心としないユーザの自己組織化が生じてしまい、元来の開発主体が関わるユーザの組織化による変化(開発サイクル・レベルの変化)を実現する機会を逸することになるであろう。

[91] 商用ソフトウェアェアなどの製品において、「機能が多すぎて使い切れない」というユーザの不満すら存在する。

３．試行コストの違い

開発サイクルの相違を生じさせる第 3 の要因として挙げられるのは、試行コストの違いである。ここで試行コストとは、ソフトウェアを開発するコスト(コードを書いて実装するコスト)と、それを公開・配布するコストを指す。商用ソフトウェアとシェアウェア、フリーウェアでは、開発対象のソフトウェアの規模[92]、開発集団の規模や運営原理の違いから、開発と公開における試行コストが大きく異なる。

すなわち、商用ソフトウェアは、比較的ソフトウェアの規模が大きく、そのためソフトウェア開発集団の規模が大きくなることに加え、賃金を支払われて開発に当たる「プロフェッショナルな」開発者が開発に当たる。つまり、開発主体が多くの人材と開発機材を抱え込むことになり、そのため人件費も固定費用も膨れあがる。さらに、製品をパッケージングして流通させるためのコスト負担もある。結果として、こうした事情により試行コストは大きくなる。

他方、シェアウェア、フリーウェアの場合、比較的ソフトウェアの規模が小さく、その開発集団も小さいことに加え、その開発に当たるのは、当該ソフトウェアの開発によって賃金報酬を得ることがない「アマチュアの」[93]開発者であり、その活動はボランティア精神によって支えられている。つまり、シェアウェア、フリーウェアでは、開発者集団も小さいし特別な開発機材もあまり必要にならない。しかもできあがったソフトウェアはオンライン公開されるので、パッケージングや流通のコストはほとんどないことになる。そのため、貨幣ベースの試行コストは低くなる。

このようなソフトウェア開発における試行コストの違いは、その前段階であるスクリーニングの有無、厳密性を左右すると考えられる。

具体的には、試行コストが高い商用ソフトウェアの場合、結果的に無駄になると予想される試作は開発コストの膨張という明確な開発パフォーマンスの低下をもたらすため、無駄になる可能性のある試作をしないように注意が払われる。そのため、スクリーニングは存在し、しかも厳密になる。

対照的に、試行コストが低いシェアウェア、フリーウェアの場合には、結果的に無駄になるような試作をしても開発パフォーマンスの低下は起こりえないか、無視できる程度にとどまる。そのため、スクリーニングが行われなかったり、行われても簡略なものになったりする状況が生じる[94]。

４．開発サイクルの規定因についてのまとめ

ここで、本章の議論をまとめておくと以下のようになる。

まず、開発主体(企業、集団、個人)が持つソフトウェアに対する認識―ソフトウェア像―は、開発主体が内部で行うαテスト、βテストの厳密性を左右する。ソフトウェアを「製品」であると見

---

[92] 試行コストの違いを生じさせる開発対象のソフトウェアの規模の違いは、その背景に前項で述べたソフトウェアの単機能指向／多機能指向の違いがある。多機能指向のソフトウェアでは、ソフトウェアの規模は大きくなるであろうし、単機能指向のソフトウェアはその規模が小さくなる可能性が高いであろう。

[93] このことは、シェアウェア、フリーウェアの開発者がソフトウェア開発によって賃金を得ていないことを必ずしも意味しない。ただ単にシェアウェア、フリーウェア開発の報酬として賃金を得ていないことを意味している。

本研究の事例に則して述べれば、シェアウェアである鶴亀メールの斉藤氏はその開発報酬として賃金を得ていないとは言い切れない面もあるが、フリーウェアである電信八号の場合は、原作者の石岡氏、現公式ビルダの福井氏、ヘルパーアプリの作者山田氏などはソフトウェア会社に勤務し、そこで開発活動に従事して報酬を得ているものの、電信八号の開発活動からは報酬を得ていない。

[94] スクリーニングを誰が行うか(企業内の人材＝一定の範囲の人間か、広く一般の人によるか)、それをどの程度厳密に行うかの違いは、過去のゲーム産業において、プラットフォームホルダーである任天堂およびソニー・コンピュータエンタテインメントがソフトウェアの発売、出荷量などを制限する際に持っていた方針と重なる点がある。すなわち、任天堂が自社内のリソースによるソフトウェアの評価に基づいてソフトウェアの発売などを制限していたのに対し、ソニー・コンピュータエンタテインメントは、それを自社内のリソースで行わず、基本的に市場での評価にゆだねていた。ゲーム産業のこうしたスクリーニングメカニズムの違いは、新宅・柳川・田中(2003)第 1 章を参照のこと。

なせばテストの厳密性は強くなり、ソフトウェアがユーザ・サポーテッド・ソフトウェアであると認識していれば、テストの厳密性は弱くなる。

次に、開発主体が持つユーザに対する認識―ユーザ像―と、それに関連するユーザの組織化の程度、サポート体制は、ユーザからのフィードバックのストック形成を左右する。

開発に際して平均的なユーザを想定し、より多くのユーザを満足させるべく多機能の実装、機能不全の回避を優先する場合には、それ故にユーザ側に不満が多くないはずだと考えてサポートが行なわれる。この場合、開発組織は元来の開発主体の外部への広がりを見せず、ユーザの組織化は行われにくい。その結果、ユーザからのフィードバックのストックは形成されにくくなる。

対照的に、開発に際して開発主体自身あるいはそれと直接関わりを持つユーザを実感し、その満足、すなわち単機能の実装、最低限の動作を優先する場合には、それ故に開発主体以外のユーザにとって不満が生じるはずだと考えてサポートが行われる。この場合、開発組織は元来の開発主体にとって外部の人や組織に対しても開かれうる可能性を内包しており、ユーザの組織化が進みやすい。その結果、ユーザからのフィードバックのストックが形成されやすくなる。

最後に、開発主体がソフトウェアを開発し、公開、配布するために要するコスト―試行コスト―は、ユーザからのフィードバックのストックをバージョン・アップ時に反映させるか否かを判断するスクリーニングの厳密性を左右する。試行コストが高い場合、ユーザに提供した際に不要と認識される可能性がある機能などを可能な限り開発・テスト・公開しなくて済むように、スクリーニングが厳密に行われる。他方、試行コストが低い場合には、ユーザに提供した際にムダ、不要と見なされる可能性がある機能なども開発・公開しても良いため、スクリーニングの厳密性が低くなる。

このように、本章で見出した 3 つの要因―ソフトウェア像、ユーザ像、試行コスト―は、前章で関門と呼んだ活動に影響を及ぼし、結果として異なる開発サイクルを生じさせると考えられる。

第 6 章　結論と今後の課題

本研究では、従来の研究において看過されてきたオンライン・ソフトウェア、中でもシェアウェア、フリーウェアと呼ばれるソフトウェアの事例を記述し、それを開発サイクルという分析視角から、商用ソフトウェア開発との比較を試みてきた。その結果、シェアウェア、フリーウェアと商用ソフトウェアでは、開発サイクルのレベルで違いがあり、その背景には、ソフトウェア像(開発主体の開発対象に対する認識)、開発コンセプトの源泉となる想定するユーザ像の違い、試行コストの高低、があるとされた。

本研究から得られた主要な知見をまとめると、次表のようになる。

表 7　本研究の知見のまとめ

	シェアウェア フリーウェア	Linux (参考事例)	商用ソフトウェア
開発サイクル	速い	速い	遅い
ユーザの組織化の程度	高い	高い	低い
試行コスト	低い	低い	高い
開発目的	自己目的・自己実現	自己目的・自己実現	収益
内発的動機づけ	あり	あり	(企業内施策による)

ただし、この 2 つの開発サイクルは、どちらかが他方に対して一方的に優れているわけではなく、各々に一長一短を持っている点に注意しておく必要があろう。換言すれば、どちらかの開発サイクルが常に有用なのではなく、開発主体が置かれている状況や開発対象のソフトウェアに応じて使い分けられるべき選択肢として本研究が見いだしたのが、2 つの典型的な開発サイクルであると考えられる。

例えば、金融機関の ATM などをはじめとする信頼性が最優先課題であるソフトウェアおよびシステムでは、「関門のある、相対的にスピードの遅い開発サイクル」が適合的であろう。反対に、本研究の事例で取り上げたインターネット用ソフトウェアのように、市場および技術の変化が速い状況下でのソフトウェアの開発には、「関門のない相対的にスピードの速い開発サイクル」が適合的であろう。したがって、ソフトウェア開発に携わる企業は、開発対象のソフトウェア像と、それを使用すると考えられるユーザ像のタイプに応じて、これら 2 つの開発サイクルを使い分けることで、より効果的な開発活動を行うことができるだろう。

また、より具体的にソフトウェア企業の開発戦略を考える場合、2 つの開発サイクルを組み合せることで、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もできるだろう。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、各々が脱パッケージ／パッケージで提供される可能性を考えてみると表 8 のようになる。オンライン・ソフトウェアは流通経路をインターネット上に求め、パッケージングされた製品としての体裁を捨て去っているとみなせるので、これを「脱パッケージ」ソフトウェアと呼んでいる。

表 8　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 らの一連の研究
	脱パッケージ	②Microsoft Windows, Office	
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 8 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。実はユーザの組織化による情報収集が、ソフトウェアの迅速な開発に非常に有効であることは、実務界でも認められ始めている[95]。

しかしながら、本研究は、オンライン・ソフトウェア、シェアウェア、フリーウェアに関する研究の第 1 歩であり、依然として不足している点、今後の研究課題は多い。

今後の研究においては、まず、より多くのシェアウェア、フリーウェアに関し、事例収集や実態調査を行うことが必要であろう。その際には、本研究で提示した開発サイクルがシェアウェア、フリーウェア一般に該当するか否かという、本研究で得られた知見のロバストネスの検証はもちろんのこと、より詳細な調査、記述、分析も必須であろう。例えば、ほとんど無報酬にも関わらず、日夜開発に取り組むシェアウェア、フリーウェア開発者のモチベーションがどのようなものであるのかを明らかにすることは、製品開発研究のみならず、ミクロ組織論の観点からも非常に興味深い点であろう。

また、本研究が記述、分析し、第 4 章においてその有り様を示した 2 つの開発サイクルは、現時点で観察された事例や文献から抽出された「典型的類型」だという点も述べておく必要があろう。換言すれば、研究対象をさらに広げた場合、商用ソフトウェア、シェアウェア、フリーウェアと開発サイクルの対応関係は本研究で述べたように明確でない可能性があり、またその対応関係が将来変化する可能性がある。このことを含め、今後の研究ではオンライン・ソフトウェアの研究と並行して、商用ソフトウェアなどに関する調査と分析も行い、想定するソフトウェア像、ユーザ像(コンセプト創造段階)や試行コスト、といった、開発サイクルの決定要因に遡って、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することが必要であると考えられる。

[95] ZD Net Japan 記事(2003a)。

補章　ソフトウェアの分類に関する試論
―ソフトウェアの配布方法とソースコード公開・改変、対価徴収などに基づく分類―

１．配布方法による分類―パッケージ・ソフトウェアとオンライン・ソフトウェア―

まず、最も単純なソフトウェア分類を試みると、「パッケージ・ソフトウェア」と「オンライン・ソフトウェア」に分けられると考えられる。

ここで「オンライン・ソフトウェア」とは、次に述べるような定義に依拠する。そもそも、オンライン上、すなわちインターネットを介して行われる可能性がある活動としては、以下の 3 つを想定することができる。

①ソフトウェアの「配布」

②バグなどに関する「情報の交換」

③(ソースコードの公開を前提とした)ソースコードの交換＝「開発」[96]

このうち、①ソフトウェアの「配布」がオンライン上で行われるソフトウェアを、本稿では「オンライン・ソフトウェア」と定義する[97]。

したがって、この定義に基づけば、①ソフトウェアの「配布」がオンライン上で行われていないものが「パッケージ・ソフトウェア」である。これには、マイクロソフト社の Windows や Office、データベースソフト、ゲームソフトなどが含まれる。

２．オンライン・ソフトウェアに関する基本的分類と細分類

(1) オンライン・ソフトウェアについての基本的分類(I)―ソースコード公開と対価支払い―

上述の定義に基づくとすると、非常に多くのソフトウェアがオンライン・ソフトウェアに含まれることになる。

この定義に該当するソフトウェアとして、まず想定されるのは、窓の杜や Vector などで入手できる、一般にフリーウェア、シェアウェアと呼ばれているソフトウェアである。しかし、その他にも、GPL (Gnu General Public License)に基づいて配布され、改変も行うことができるソフトウェア、無償オープンソース・ソフトウェアなどもオンライン・ソフトウェアに含まれることになる。さらに、最近では一部の「商用」ソフトウェア、例えばアンチウィルスソフトなどもオンライン・ソフトウェアに含まれることになる。

そこで、上記の定義を前提にしつつ、より細かな分類を提示することにしよう。

第 1 に提示する細分類において、分類軸となるのは、③(ソースコードの公開を前提とした)ソースコードの交換＝「開発」と、ソフトウェアに対する対価の支払いである。この 2 軸によって細分類を試みると下表のようになる。

---

[96] 厳密にいえば、「開発」活動(＝新規なソースコードの作成、既存のソースコードの改変)は、個々の制作者のコンピュータ上･クライアント上で行われるため、オンライン上で行われるのはソースコードの交換のみになる。

[97] この直後で述べるように、③(ソースコードの公開を前提とした)ソースコードの交換＝「開発」は、オンライン･ソフトウェアの細分類に用いられる。したがって、ここで提示した 3 つのオンライン上で遂行可能な活動の内、②バグなどに関する「情報の交換」は、定義では用いられないことになる。

しかしながら、②バグなどに関する「情報の交換」は、オンライン･ソフトウェア、パッケージ･ソフトウェアを問わず行われている活動であり、(少なくとも)オンライン･ソフトウェアの開発に対して大きな影響を活動として考慮に値するものであるといえる。

補表１　オンライン上のソフトウェア開発と対価支払いの有無に基づく分類

		ソフトウェアに対する対価の支払い		
		多額・有り (開発費+利益をカバーする程度)	少額・有り (開発費をカバーする程度)	無し
ソースコードの公開・交換（開発）	有り	オープンソース・ソフトウェア[98]	オープンソース・ソフトウェア	無償オープンソース・ ソフトウェア
	無し	商用ソフトウェア	シェアウェア (含、カンパウェア等)	フリーウェア (含、メールウェア等)

ここで、その提供が無料もしくはそれに近い低価格でなされており、同時に、「開発」は行われていないものの「配布」がオンライン上で進められているという 2 点において、ソフトウェアに対する対価が無く提供されているフリーウェアと、開発費をカバーする程度の少額で提供されているシェアウェアは、一括して扱うことが可能であると考えられる。

そこで、本研究では、ソフトウェアに対する対価が無く提供されているフリーウェアと、開発費をカバーする程度の少額で提供されているシェアウェアを一括して(広義の)「無料ソフトウェア」と総称することにする。

ソースコード公開に関する更なる細分類

なお、一口に「ソースコードが公開されている」としても、現在それには多様なパターンがあり得る。したがって、ここではソースコードの公開に関し、より細かい分類を試みることにしたい。

その分類軸は大きく 3 つある。1 つは、ソースコードの「閲覧」が可能であるか否かである。第 2 の分類軸は、ソースコードの「(自由な)改変」が可能であるか否かである。そして第 3 の分類軸は、これらの「閲覧」、「改変」が有償で許されているのか、無償で許されているのかである。

以下では、第 1 と第 2 の分類軸を用いて、ソフトウェアのソースコード公開に関する細分類を提示してみることにしよう。その細分類は、次表のようになると考えられる。

[98] GPL (Gnu General Public License)に基づく「オープンソース・ソフトウェア」に関しては、その呼称や内容において、変遷･混乱が生じてきた歴史があり、現在に至っている。現在一般的には「利用、頒布・再頒布・改変が自由」に行われることを許可したソフトウェアがオープンソース・ソフトウェアと呼ばれている。したがって、補表 1 で示したように、オープンソース・ソフトウェアであっても多額あるいは少額の対価を受け取ることは可能である。なお、GPL 並びにオープンソース・ソフトウェアの厳密な定義、過去の経緯については、Dibona, Ockman & Stone (1999)、高橋・高松(2002) などを参照のこと。

補表 1’　ソースコードの閲覧、改変に基づく分類

		ソースコードの改変	
		可能	不可能 (原著作者に改変したコードの権利が帰属する場合も含む)
ソースコードの閲覧	可能	オープンソース	シェアードソース
	不可能	×	クローズドソース

上表の 3 つのソースコード公開パターン、各々について、(第 3 の分類軸である)有償・無償である場合があり得る。例えば、近年注目を集めた、Linux は「無償」オープンソースであるといえる。

(2) オンライン・ソフトウェアに関する細分類(II)―試用と継続使用―

第 2 に提示する分類の分類軸は、「ソフトウェアに対する対価の支払い」に関するより細かな分類軸である。具体的には「期間限定の使用(＝試用)」と、「(期間限定なしの)使用」が、無料／有料で提供されているか否かによって、さらに細かな分類が可能となる。

補表 2　ソフトウェアの継続使用の無償性と試用可能性に基づく分類

		ソフトウェアの試用	
		無料	有料
ソフトウェアの継続使用	無料	フリーウェア	×
	有料	シェアウェア (一部の商用ソフトウェア)	パッケージ・ソフトウェア (多くの商用ソフトウェア)

(3) オンライン・ソフトウェアに関する細分類(III)―対価徴収方法と最終コスト負担者―

第 3 の分類に当たっての分類軸は、試用・使用が有料／無料で行われる際の、実際の取引形態に基づく分類軸である。取引形態に基づく分類では、「対価徴収方法」と「最終コスト負担者」の 2 軸によって、さらなる細分類が可能となる。

ここで最終コスト負担者として想定されるのは、(a)ソフトウェアの「提供者」、(b)「使用者」、(c)その他「第 3 者」であるが、「提供者」が最終コスト負担者であることは、事実上あり得ないので、ここでは、(b)「使用者」と(c)その他「第 3 者」を最終コスト負担者の候補とする。

一方、対価徴収方法に関しては、「ソフトウェアの(直接的、明示的に)対価を徴収」する場合の他、「補完的なソフトウェア、あるいは財・サービスの対価(＝補完財)として徴収」する場合、「消費者の個人情報を対価として徴収(＝顧客、ユーザデータベースの構築などが目的)」が考えられる。

補表３　最終コスト負担者と取引形態による分類

		取引形態		
		対価を徴収	補完財による徴収	個人情報を 対価として徴収
最終コスト 負担者	利用者	商用ソフトウェア (販売)	バンドルウェア (抱き合わせ販売) 例：Internet Explorer	スパイウェア[99]
	第３者	アドウェア (「広告モデル」) 例：Opera		

上表において、斜線部分は使用者＝一般的な消費者から見て、一見無償でソフトウェアの提供を受け、利用しているように見受けられるが、実際には、補完財の購入(バンドルウェア)、個人情報の提供や広告の閲覧(アドウェア)などの形で、間接的、非明示的にコストを支払っている。

これより、直接的・明示的な対価支払いが行われている場合はもちろんのこと、バンドルウェアやアドウェアのように間接的もしくは非明示的に対価を支払っていない場合に限り、使用者が根本的に「無料で」ソフトウェアを利用していることになる。したがって、こうした使用・使用者を許可しているソフトウェアを、(狭義の)「無償ソフトウェア」と呼ぶことにする。

３．まとめ―本論の事例とシェアウェアとフリーウェアの違い―

最後にこの補論のまとめとして、以上の分類を、本論で取り上げた事例に適用してみるとしよう。まず、これまで挙げてきた分類軸のうち、(1)配布方法による分類(パッケージ・ソフトウェアとオンライン・ソフトウェア)の分類軸に関しては、両事例とも同一範疇に属すると言える。すなわち、鶴亀メール、電信八号とも、配布がオンライン上で行われているため、オンライン・ソフトウェアに分類することができる。

さらに、鶴亀メール、電信八号とも、その提供が無料もしくはそれに近い低価格でなされており、同時に、少なくともその「配布」がオンライン上で進められているという２点において共通点を有している。したがって、本研究での定義に従えば、両事例とも(広義の)「無料ソフトウェア」に分類することができるであろう。

ただし、この ２ つの事例の間には、オンライン・ソフトウェアについての基本的分類(I)(ソースコード公開と対価支払い)とオンライン・ソフトウェアに関する細分類(II)(試用と継続使用)において、明確な違いがある。

この点を考慮し、分類枠組みを両者に適用すると、その結果は以下のように要約できる。

[99] この表で提示した「スパイウェア」と「アドウェア」という呼称に関しては、現在その定義を巡って論争が行われている。(ZD Net Japan 記事(2003b)など)

補表４　本研究の事例への分類の適用

		ソースコードの公開	
		公開 (一定の条件付を含む) 「オープンソース」	非公開 「クローズドソース」
継続使用の使用料	有料	×	シェアウェア (秀丸エディタ、鶴亀メール、Becky! Internet Mail)
	無料	フリーウェア (Mozilla、OpenOffice.org、電信八号)	フリーウェア (Lhasa、FFFTP、Disk Mirroring Tool)

つまり、ソース公開と継続使用の有料／無料がシェアウェアとフリーウェアを分ける基本的な分類軸になるといえる[100]。

最後に、フリーウェアに分類される電信八号などの事例について、オンライン・ソフトウェアに関する細分類(III) (対価徴収方法と最終コスト負担者)の分類を適用してみよう。

すると、電信八号では直接的な対価徴収はもちろんのこと、間接的、非明示的な対価徴収も行われていない。したがって、電信八号は「(狭義の)無償ソフトウェア」に分類することができる。

[100] また、さらに細かく見れば、開発継続のモチベーションの違いも両者の間には見られるであろう。

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』Vol.45, No.1, pp.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』Vol.31, No.1, pp.20-36.

Boehm, B W (1988) "A Spiral model of software development," *IEEE Computer*, Vol.21, No.2, pp61-72.

Brooks, F. (1975) *The Mythical Man-Month : essays on software engineering.* Addison-Wesley, Boston; MA.（滝沢徹・牧野祐子・富澤昇訳『人月の神話』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）

馬場靖憲(1998)『デジタル価値創造―未来からのモノづくり原論―』NTT 出版.

Cusumano, M. A. (1991) *Japan's Software Factories: A Challenge to U.S. Management*, Oxford University Press, New York.（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, New York.（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, New York.（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, Cambridge, MA.（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）

藤田英樹(2000)「誇り動機づけ理論」『組織科学』32(4), pp.59-75.

藤田英樹(2002) 「『誇り』『見通し』と動機づけ」東京大学大学院経済学研究科博士学位論文, 2002 年 3 月.

藤本隆宏=キム・B・クラーク(1993)『製品開発力』ダイヤモンド社.

g 新部裕(2003)「フリーソフトウェア活動と国際協力」『赤門マネジメント・レビュー』Vol.2, No.2, pp.107-114, GBRC.

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）

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』Vol1, No7, pp.1-33, GBRC.

Raymond, E. (1997) *The Cathedral and the Bazaar*, Retrieved April 2002, from http://www.tuxedo.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.

妹尾大(2001)「ソフトウェア開発の新潮流―状況論的リーダーシップの胎動―」『組織科学』Vol.35, No.2, pp.65-80.

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.

高橋伸夫編(2000)『超企業・組織論』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』Vol.1, No.4, pp.1-26, GBRC.

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』Vol.1, No.4, pp.1-28, GBRC.

立本博文(2003)「製品タイプと開発プロセスの適合性」『ゲーム産業の経済分析―コンテンツ産業発展の構造と戦略―』東洋経済新報社.

@IT 記事(2003)「ピーター・コードが語る開発プロセスの選び方」(http://www.atmarkit.co.jp/fjava/devs/interview03/interview03.html).

Impress 記事(2003)「マイクロソフト、相次いで発見されたパッチ未公開脆弱性の見解を語る」(http://internet.watch.impress.co.jp/cda/special/2003/12/24/1593.html).

ZD Net Japan 記事(2003a)「ゲーム開発からビジネスソフトへ、浸透を始める“ ブロッグ ”」(http://www.zdnet.co.jp/news/0302/03/ne00_blog.html).

ZD Net Japan 記事(2003b)「「スパイウェア」呼ばわりはダメ――Gator、批判者に法的措置

も」
(http://www.zdnet.co.jp/news/0310/23/ne00_gator.html).