論　説

オープンソースソフトウェアの開発プロセスに関する考察

竹　田　昌　弘

目　次

はじめに

Linux オペレーティングシステムの普及によって，オープンソースソフトウェアが注目を集めるようになってきた。オープンソースソフトウェアとは，ソフトウェアのソースコードをオープンにした上で流通が行われるものであり，そのソースコードを利用することで誰もが自発的に自由にその開発プロセスに参加できるという特徴を持っている。オープンソースソフトウェアを巡っては，これまでの組織プロセスとは異なる新たなプロセスが発現しており，さまざまな視点から分析を行うことが可能であるが，ここでは特にその開発プロセスに焦点を合わせて分析を行う。

コンピュータの誕生以来，ハードウェア技術の進歩にしたがって，高度な機能を持つソフトウェアが求められるようになり，ソフトウェアの構造は複雑化していった。これにともない，その開発プロセスも様々な進歩を遂げている。本稿では，これまでのソフトウェア開発プロセスの変遷を簡単に整理した上で，オープンソースソフトウェアの開発プロセスの特徴的な点を

指摘する。さらに，ソフトウェア流通の形態が異なることによってソフトウェア開発プロセスが受ける影響を考察し，オープンソースソフトウェアの開発プロセスをほかのソフトウェア開発などに導入する可能性などを検討する。

1．ソフトウェア開発プロセスの変遷

ソフトウェアとは，コンピュータのハードウェアを利用して，何らかの目的を達成するために利用する処理手順の記述などである。処理手順の記述は，いわゆるプログラムであるが，コンピュータで実行するプログラムは，そのコンピュータ・ハードウェアに固有の機械語体系に即して 2 進数化されなければ実行することができない。初期には開発者が 2 進数を直接記述することでプログラムを開発していたが，生産性の低さから，開発者が理解しやすく，記述しやすい人工的なプログラミング言語が開発され，開発者はこれを使って処理手順を記述するようになった。プログラミング言語で記述したプログラムをソースコードという。

現在ではソフトウェアの開発は，ソースコードを作成することが目的となり，ソースコードはコンパイラなどによって 2 進数に翻訳される。

（1）開発プロセスとは

ソフトウェアの開発において，そのプロセスを定義しておくことは重要である。

ソフトウェアの規模が巨大化し，複雑化をしていくにしたがって，その開発には多くの開発者が組織的に取り組むようになった。一人の開発者で完結する作業としてソフトウェアを開発するのであれば，その開発の方法は開発者の自由である。しかし，複数の開発者が協働する場合には，開発者相互のコミュニケーション，進捗管理などのために，開発プロセスを定式化し，共有しなければならない。

開発プロセスは，作業の進め方を意味するが，定義されている開発プロセスの多くは，プロセスそのものではなく，そこで生成される成果物[1)] を指定することが多い。さまざまな成果物の連鎖を定義することで，ある成果物から次の成果物にいたるプロセスを指定している。

成果物を中心にプロセスを整理するのは，次のような理由による。

① 開発者相互のコミュニケーション

個々の開発者が担当する部分はそれぞれ独立なものではなく，互いに依存するものを並行して開発を進める。そこで，その依存関係を成果物としての文書で共有することで，開発者間のコミュニケーションを確実に，かつ，最小限のコストで行うことができる。

② 進捗確認

1) たとえば，どのようなソフトウェアを開発するのかを整理した仕様書や，ソフトウェアの詳細な構造を定義した設計文書などが成果物である。また，最終的に作り出されるソースコードも成果物の一つである。

成果物というかたちあるものを基準にすることによって，作業の進捗状況が確認できる。ソフトウェア全体の開発を進める上で，各開発者の分担する部分の進捗を把握し，必要に応じて対策を講じることによって，全体のスケジュールを維持できる。

量的な進捗の確認だけでなく，成果物は開発作業の品質を測るためにも有効である。

③ 運用と保守

ソフトウェアが完成して，利用を開始してからでも，意図しなかったプログラムミス（バグ）の修正や，環境の変化によって生じる処理手順の変更などの必要がある。ソフトウェアの部分的な修正にあたっては，成果物の連鎖の適切な段階に戻って，必要な変更を加えていくことになるので，成果物が体系的に整理されていることが重要になる。

（2）ウォーターフォールモデル

大規模ソフトウェアの開発で長年，支配的に利用されていたのがウォーターフォールモデルである。ソースコード記述の生産性があまり高くない状況では，ソフトウェア開発全体のコストを最小化するため，ソフトウェア開発のすべての段階をやり直しなしに一回だけで終わらせる必要があった。

ウォーターフォールモデルでは，ソフトウェア開発プロセスを計画，分析，設計，構築，テスト，導入，運用などの段階に分割し，それぞれ次の段階に進む前に十分な検討を行い，一度次の段階に進むと後戻りしないことが特徴である[2)]。部分的に次の段階を進んだり，後戻りをしたりすると，ソフトウェアの各部分の依存性のため，開発のほかの部分でやり直しが発生し，余分なコストが発生するからである。

段階を分け，同期して段階を進むため，大規模なソフトウェアの開発では，巨大な開発プロジェクト組織を設定しなければならなくなる。これは，巨大組織の管理と，一時的に多数の開発者を集めるという課題をともなう。また，各段階での進捗と完了を確認するために，比較的多くの成果物を要する。

（3）ソフトウェア・プロトタイピング

ウォーターフォールモデルでは，ソフトウェアの開発を一連の後戻りのないプロセスで実施しているが，一般の工業製品の開発において，このようなプロセスがとられることはほとんどない。通常は，試作品の作成を繰り返しながら，製品の完成度を高め，完成形ができあがったところで生産活動に移る。ソフトウェアの開発では，大量生産が必要である場合の生産活動は，単純な複製作業であり，CDROM などの媒体に複製する作業は，製品としてのソフトウェアの内容にかかわらず，同一である。すなわち，生産活動にはほとんどコストがかからず，開発が終了すれば，すぐに生産に移ることができる。一方で，開発においては試作品を作成しようと

2) 後戻りしないところを，滝（waterfall）を落ちる水にたとえ，名付けられた。

しても, 実際にコンピュータの上で稼働させて内容を確認しようとすれば, 試作品といえども, 最終製品と同じだけのコストをかける必要がある。

ウォーターフォールモデルでは，やり直しを避けることでコストを最小化できるかもしれないが, 製品の完成度という意味では, 成果物の検査や開発者の能力に依存するところが大きい。これを補うために, プログラミング言語の記述性を高めたり, オブジェクト指向開発のように, ソフトウェアのモジュール化を高めたりすることで，開発の生産性を高め[3]，試作の繰り返しによって完成度を高める開発プロセスが適用されるようになった。

このような開発プロセスをソフトウェア・プロトタイピングという。プロトタイピングによるソフトウェア開発には，大きく二つの方法がある。一つは，文字通り試作品の改善を通じて完成品に近づけていく方法で，もう一つは試作専用のソフトウェアによって，試作のための試作品の開発を短時間で繰り返して仕様の確定をする方法である。前者では，試作の繰り返しを終了した時点で完成品が得られるが，後者では，試作を終えた後で得られた仕様をもとに改めて開発を行うことになる。後者の方が，実行効率の面では優れたソフトウェアが得られる。

(4) RAD

ソフトウェア・プロトタイピングは，完成度の高いソフトウェアを開発するプロセスであるが，大規模なソフトウェアの開発では限界がある。巨大な試作品を作成することは困難であるから，部分的に適用するか，全体をサブシステムに分解した上で適用するしかない。

RDBMS（関係データベース管理システム）の登場[4] によって, ソフトウェアの開発プロセスは, さらに進化する。RAD（Rapid Application Development）である。

RAD では，分析に基づいて，最初にデータベースの構造を確定する。処理の手順，すなわちプログラムは環境によって変わることがあるとしても，ビジネスの本質が変わらなければ，そこで扱われる情報（データ）の構造は大きく変わらないので，ソフトウェア開発の基本をデータベースにおいたのである。RDBMS では，テーブルに保存したデータを自由に再編成して利用できるので，適切に分析を行って構築されたデータベースは大きな変更なしに利用し続けることが可能である。

データベースの構造を確定した後，プログラムを開発する。処理手順はデータの操作が中心

---

3) 開発生産性を高めるためには，開発に利用するソフトウェアの進歩が必要であったが，そのためには，同時にハードウェア技術の進歩によって，コンピュータの高性能化と低価格化が必要であった。開発者の労力と開発に利用するコンピュータの計算時間はトレードオフする。機械語でのプログラミングは，労力はかかるが計算時間はかからない。

ソフトウェア開発以外の領域でも, ウィンドウをベースにしたオペレーティングシステムがコンピュータの普及を促進しているが, そのためには, そのようなオペレーティングシステムが現実的な速度で稼働し，適当な価格で入手できるコンピュータの登場が必要であった。

4) もちろん，ここでも RDMBS を実用的に稼働させるハードウェアの進化が必要であった。

であって，ほかの処理手順とはデータを経由して依存関係があるとしても，適当なサブシステムに分割すれば，それぞれを独立に開発できる[5]。すなわち，処理に優先順位をつけ，順次開発を進めることができるので，開発プロジェクトの規模をそれほど大きくする必要がない。また，大きな規模の組織によって短期間で開発を終了しようとする場合でも，各部分の依存性が低いので，開発組織を複数に分割して独立に作業を進めることができる。

ソフトウェアを構成するプログラム同士の依存性が低いというのは，開発だけでなく，保守でも有利である。データベースの構造を変えることがなければ，部分的なプログラムの修正がほかの部分に影響することはほとんどない。

RAD では，ソフトウェア開発がダイナミックに進められるため，プログラムの設計書に相当する成果物は，ソースコードと一括して作成，保存，管理することが多い。このために，ソースコードを管理するためのソフトウェアを利用したり，ソースコードの中にコメント（注釈）として，ソフトウェアの文書化したりすることが一般的である。

2. オープンソースソフトウェアの開発プロセス

(1) オープンソースソフトウェアの定義

オープンソースソフトウェアの定義については，オープンソースソフトウェアの発展を目的とした団体 OSI（Open Source Initiative）が管理している。その日本語訳の項目を以下に抜粋，引用する[6]。

5) RDBMS 以前のシステムでは，一時的なデータファイルを仲立ちとして処理を連鎖するバッチ処理をとることが基本であったので，データはプログラムの中に散在し，各プログラムはネットワーク状に依存関係を持ち，部分的に開発を進めることはできなかった。

6) オープンソースの定義の原文は，http://www.opensource.org/docs/definition.php に，八田真行による日本語訳は，http://www.opensource.gr.jp/osd/osd-japanese.html にある。
本稿では，2004 年 9 月 10 日にアクセスした八田訳から前文と項目を抜粋した。

はじめに
「オープンソース」とは，単にソースコードが閲覧あるいは入手できるということだけを意味するのではありません。「オープンソース」であるプログラムの頒布条件は，以下の基準を満たしていなければなりません。
1. 自由な再頒布
2. ソースコード
3. 派生ソフトウェア
4. 作者のソースコードの完全性(integrity)
5. 個人やグループに対する差別の禁止
6. 利用する分野(fields of endeavor)に対する差別の禁止
7. ライセンスの分配(distribution)
8. 特定製品でのみ有効なライセンスの禁止
9. 他のソフトウェアを制限するライセンスの禁止
10. ライセンスは技術中立的でなければならない

ここでは，定義の厳密な解釈をするのではなく，本稿での議論に重要な点を二つ紹介するにとどめる。

まず，自由な再配布（再頒布）であるが，流通の自由を保証されている必要がある。また，再配布にあたって配布者が対価を得ていたとしても，開発者は開発の対価を配布者に求めてはならない。これによって，ソフトウェアを私有化して利益を得ようとすることを防いでいる。

そして，実行可能なプログラムと共に，ソースコードを配布するというのは，オープンソースソフトウェアの名前の起源でもある。ただし，ソースコードは必ずしも，実行可能プログラムと一体化して配布しなければならないわけではないし，ソースコードの配布に妥当な費用を徴収することも妨げてはいない。もう一つ重要なことは，ソースコードがわかりやすいものであることを求めている。つまり，利用者が積極的にソフトウェアのソースコードを閲覧し，改変することを促進している。

このような定義を満たすオープンソースソフトウェアを実現するための手段として利用されているのが，GPL というライセンス文書である。

(2) GPL

GPL（General Public License）は，Richard Stallman が設立したフリーソフトウェア財団（FSF:Free Software Foundation)が中心となっている GNU プロジェクトで開発するソフトウェアのために作成したライセンス文書である。Linux オペレーティングシステムもこの GNU GPL によって保護されている。今日までに，改訂が繰り返されると共に，GNU GPL をもとに

してさまざまな目的で類似のライセンス文書が作られてきている。本稿では GPL の詳細に踏み込むことはしないが，その役割について簡単に指摘する。

GPL は，オープンソース定義にあるソフトウェアの自由を保障するライセンス文書である。ライセンスで保護することで自由を保障するというのは，一見矛盾しそうであるが，ライセンスによる保護なく開発者が権利を放棄してソフトウェアを配布すると，これを改変した者が新たなバージョンのソフトウェアを私有化して営利目的で販売することで自由な開発の連鎖を断ち切ってしまうことがある。そこで，開発者の権利としてオリジナルの再配布を保障するだけでなく，改変した場合にはその配布にも同じライセンス文書を適用することを指定し，その条件の下でソースコードの配布と改変を認めている。

このようにして, GPL はオープンソースソフトウェア開発の基本的なプロセスを保障する仕組みになっている。GPL がなければ，現在のように高品質のオープンソースソフトウェアが多数流通し，普及することはなかったといっても言い過ぎではない。

（3）プロジェクト組織の構成

一般のソフトウェア開発と比較して，オープンソースソフトウェアの開発組織に独特な点には，開発プロジェクトへの参加動機，コミュニケーションの手段，管理統制の方法，利用者との関わりなどがある。

① 参加動機

一般のソフトウェア開発では，開発者が企業勤務者であれ，フリーランスであれ，職務として開発組織に参加し，報酬を得ている。一方，オープンソースソフトウェアの開発の多くは，無報酬で自発的に参加する開発者を中心として行われている。大多数の開発者は，それぞれの勤務先でソフトウェア開発に携わり，勤務外の時間を使って，プロジェクトに参加している。自発的プログラマの典型は，Linux プロジェクトを開始した Linus Torvalds である。彼の共著書名 *Just for Fun*[7] にも見られるように，知的好奇心を満足させることを動機付けにプロジェクトを開始している。

しかし，オープンソース定義でも，GPL でも，開発から報酬を得ることは禁じてはいない。実際のところ最近では IBM などの従業員が Linux 開発プロジェクトに業務として参加している。

オープンソースソフトウェア開発プロジェクトへの参加者の動機付けは，多様である。各種調査[8] によると，知的刺激や能力向上などが上位を占めるが，中にはオープンソースソフト

7) Linus Torvalds and David Diamond, *Just for Fun: The Story of an Accidental Revolutionary,* Harpercollins, 2001.（風見潤訳：リーナス トーバルズ，デビッド ダイヤモンド，『それがぼくには楽しかったから』，小学館プロダクション，2001.）

8) オープンソースプロジェクトへの参加動機を調査したものとしては，OSDN（Open Source

（次頁に続く）

ウェアコミュニティでの評判や，専門家としての地位向上[9] なども動機として挙げられているように，非常に多様な動機付けによる開発者が集まっているのが，オープンソースソフトウェアの開発組織である。

② コミュニケーション手段

一般のソフトウェア開発においては，通常は十分なコミュニケーションをとるため，プロジェクトを遂行するための場所を用意し，そこに集まって作業をする。一方で，オープンソースソフトウェアの開発は，自発的かつパートタイムでの参加が中心であるため，物理的に同じ場所に集まって作業をすることはできない。作業の場はほとんどが開発者の自宅であり，情報交換の場は，インターネットである。インターネット上に整備されたプロジェクトの Web サイトや，CVS と呼ばれるソースコード管理システムが，バーチャルなプロジェクトルームとして機能し，メーリングリストがコミュニケーションの基本となっている。

対面でのコミュニケーションがないことは，一見不利なように見えるが，自発性に基づいて開発者の出入りがあるオープンソースソフトウェアの開発においては，すべてのコミュニケーションがメーリングリストなどの明示的な手段で行われ，履歴が残っていることは大きな利点となっている。

③ 管理統制

管理統制について，一般のソフトウェア開発では，通常の企業組織同様に組織構造による公式の権威による統制が中心となり，専門性によるリーダーシップが一部を補完している。オープンソースソフトウェアの場合には，自発性による参加のため，基本的には統制の強制力はない。専門性，能力によるリーダーシップが中心となるが，それでも開発作業の分担を強制したりすることはできず，開発者各自が自らの貢献方法を選択し，プロジェクトに参加している。しかし，プロジェクトに参加し，貢献していると思っていても，実際にソースコードに貢献しているとは限らない。ソフトウェアの改善のためのソースコードを作成して提出しても，これが必ず採用されるわけではない。同じ部分に複数の開発者が並行して独立にソースコードを開発し，提出されたソースコードの中から，最良のもの，プロジェクトの方向性に合致したものが選択され取り込まれていく。

プロジェクトリーダーの影響力は，この選別を通じてプロジェクト全体に働きかける。選別

Development Network）と Boston Consulting Group が 2002 年に報告した Hacker Survey（http://www.ostg.com/bcg/）や，International Institute of Infonomics が 2002 年に実施した Free/Libre and Open Source Software: Survey and Study（http://www.infonomics.nl/FLOSS/report/）などがある。

9) このような自らの技術者としての市場価値を高めようとするためには，Linux や Apache など有名な開発プロジェクトに参加して，貢献しなければならない。すべてのオープンソースソフトウェア開発プロジェクトに関しての調査では，このような動機付けの比率は必ずしも高くないが，Linux プロジェクトに限定した調査では，このような動機付けによる参加の比率が高いことが予想される。

を何度も経験するうちに，プロジェクトが求めている方向性がどのようなものであるのかをプロジェクトの中で暗黙の合意とすることができる。一方で，選別が不適切だと考える開発者はプロジェクトから離れてしまうかもしれない。このような形での統制が可能であるのは，オープンソースソフトウェアの開発が，利益を目的としたものではなく，十分な冗長性が許容されるからである。

④ 利用者との関わり

利用者との関わり方は，さまざまな形態がある。業務システムのように，開発の最初で業務内容を確認するために深く関わり，以降，開発を終了して運用を開始する直前までほとんど関わりがないケースもあれば，ソフトウェア・プロトタイピングのように利用者の介在によって製品の完成度を高めるケースもある。市販パッケージソフトにおいては，市場調査のような形で利用者の声を聞くにしても，密接な関わりがなく開発が進められ，製品として初めて利用者（顧客）の前に出現する。

オープンソースソフトウェアでは，配布されるソフトウェアにそのまま開発に利用できるソースコードがついてくるのであるから，利用者が開発者になることができる。換言すれば，開発者はすべて基本的にはそのソフトウェアの利用者である。すなわち，オープンソースソフトウェアの開発組織とは，利用者の集合であって，プロジェクトへの関わりの深さによって，ソースコードを開発して提供することで貢献する者，修正バージョンを積極的に利用して問題を発見して報告する者，日常の利用の中で発見した問題を報告する者，ただ利用するだけの者などに分類できる。しかし，この分類にはきちんとした明確な境界があるわけではなく，曖昧な境界のグループが階層的に存在している。また，各グループから別のグループへの移行がダイナミックに行われている。

（4）並行開発モデル

前項ではオープンソースソフトウェアの開発プロセスにふれながら，一般のソフトウェア開発組織と，オープンソースソフトウェアの開発組織を比較してきた。ここでは，多くのオープンソースソフトウェアの開発で採用されている並行開発モデルを取り上げる。

まず，一般のソフトウェア開発とオープンソースソフトウェアの開発プロセスとでは，扱っている部分に違いがあるので，これを明確にしておこう。ソフトウェア開発プロセスの説明からも分かるように，一般のソフトウェア開発では，ソフトウェアとしてのかたちのないところから，稼働するプログラムとしてのソフトウェアを完成させるところまでを開発プロセスとしてみている。一方で，オープンソースソフトウェアの開発では，ソースコードを公開しているということからも分かるように，完成度は十分ではないかもしれないが，実際に稼働するプログラムが完成したところから，独特の開発プロセスが始まっている。従来のソフトウェア開発プロセスでいえば，保守の段階に相当する部分だといってもよい。オープンソースソフトウェ

アの開発では，一部の中心的メンバーの努力による場合[10] や，既存のソフトウェアがプロジェクトリーダーの都合[11] などで，最初のソースコードとして公開され，以来，開発が継続している。この初期のソースコードの品質とその後の開発プロセスの管理は，オープンソースソフトウェアの開発を成功に導く上で等しく重要である。

ソースコードが公開されたあとのオープンソースソフトウェアの開発プロセスは，一般のソフトウェア開発におけるソフトウェア･プロトタイピングに似たプロセスをとる。オープンソースソフトウェアの場合，利用者の要望に対応することもあるが，ほとんどは開発者たちが機能追加や信頼性向上のための提案を行い，それに従ってソースコードの開発が進められている。開発の方向性を決定する主体が利用者ではなく，開発者であるが，開発を進める方法はソフトウェア･プロトタイピングそのものといってよいだろう。市販ソフトウェア製品よりも高機能，高信頼性の水準に至っているオープンソースソフトウェアがあるのは，試作を繰り返して，完成度を高めているからである。

また，モジュール化によって分割開発を行い，モジュール間の依存性を削減し開発作業の独立性を確保するところは，RAD に似ているともいえる。オープンソースソフトウェアは必ずしも RDBMS を使っているわけではないし，RDBMS の基盤になるオペレーティングシステムや，RDBMS そのものもオープンソースソフトウェアとして開発されているのであるが，ソフトウェア全体をうまくモジュールに分割して，作業の依存性を低く抑えることが，プロジェクト成功の鍵にもなっている[12]。初期のソースコードがモジュール化を意識して作成されていなければ，インターネットを利用した作業では開発が継続しにくい。開発が進むことによって，適正なモジュール分割の境界がわかってくることもある。そのようなときには，ある時点でのソースコードをもとにして，新たな分割によるソースコードを構成して，新バージョンのソフトウェアとして公開し，開発の方向転換をすることがある。

ソフトウェアのバージョン更新は，このような大きな構成変更や機能追加によって行われるもののほか，ソフトウェア・プロトタイピングのように少しずつ修正を加えることで行われるものがある。

オープンソースソフトウェアの開発は，細かな修正によって頻繁にバージョンが更新されていく。多くの労力をかけなければ成果を完成して達成感を味わうことのできないソフトウェア

10) Linux プロジェクトでは，Linus Torvalds が，Minix というパソコン用の Unix オペレーティングシステムのコンパクトなバージョンを参考に，独自のコードを作成し，インターネットで仲間を募って作業を行い，初期バージョンのソースコードを作成して公開にいたっている。

11) Web サーバの業界標準ともいえる Apache サーバは，イリノイ大学 NCSA（the National Center for Supercomputing Applications）の開発した HTTP サーバが組織的に開発活動を維持できなくなったため，ソースコードの公開を行い，オープンソースソフトウェアとして開発が継続されている。

12) したがって，初期ソースコードを公開してプロジェクトを開始するプロジェクトリーダーには，ソフトウェア開発に関する先見性が要求される。

開発のなかで，頻繁にバージョン更新を行ってソースコードへの貢献の達成感を味わうことができるオープンソースソフトウェアの開発は，開発者を引きつける大きな魅力になっている。

しかし，一方で頻繁なバージョン更新は，利用者にとっては問題を引き起こす。利用者にとっては，機能や信頼性がわずかばかり向上するよりも，安定して利用できる製品の方が有益である。コンピュータ上のソフトウェアを新しいものに更新する労力を最小化し，本来の用途でソフトウェアの利用を最大化したい。

頻繁なバージョン更新と安定した利用という一見矛盾する要求に応えるため，Linux プロジェクトのように一定の利用者を持つオープンソースソフトウェア開発プロジェクトでは，並列開発が行われている。つまり，頻繁なバージョン更新を通じて，機能追加や信頼性向上を実験的に進めていく開発系列と，安定した機能を提供しバージョン更新の間隔を十分にとった開発系列を並行し提示する。継続開発の系列では，新しいソースコードを追加して，全体に問題がないかどうかを確認し，問題があれば前に戻したり，新たなソースコードで対応したり，まさに実験的な開発作業が行われている。そういう意味づけをしている開発作業であるから，大胆な改変を行うことも可能になる。一方で，安定したバージョンの開発系列では，新規機能の追加などは原則的に行わず，公開以降に発見された問題点に対する対応などのために，最小限の更新をするのみである。

二つの開発系列は，実験的バージョンの開発系列のソースコードが安定したところで合流する。新機能などの大きな改変が落ち着いた実験バージョンは，新たな安定バージョンとして公開され，実験バージョンは新たな目標を設定して，次の実験的な開発系列を開始する[13)]。

このような並列開発プロセスはオープンソースソフトウェアに固有のものである。開発者の革新を求める参加動機を維持しながら，利用者の求める安定性と効率を提供している。これが可能になるのは，前にも述べたが，オープンソースソフトウェアの開発が自発的参加による，原則的には無報酬の作業だからであって，このような冗長性は効率が優先されるビジネスとしてのソフトウェア開発では適用が困難である。

（5）方向性の維持

並列開発は開発者の参加動機を維持するために有効であるが，一方で，多様な開発者の集合であるプロジェクトの方向性を維持することも，オープンソースソフトウェアの開発にとって必須の事項である[14)]。その方法としては，さまざまなリーダーシップや，カリスマ的な指導者

13) Linux プロジェクトでは，二つの開発系列を区別するため，バージョンを識別する番号（バージョン番号）を奇数と偶数とに分けて管理している。

14) ソフトウェア開発を通じて新たな価値を創造するという前向きの目標だけが求心力を作り出すのではなく，Microsoft などの巨大企業によるソフトウェアの支配に対する反発という後ろ向きの目標が求心力を作り出すこともある。

の存在などプロジェクトによってさまざまである。ここでは，Linux プロジェクトで Linus Torvalds が実行し，成功している方法を，その一例として採り上げる。

Torvalds は，開発作業にあたって，メーリングリストを通じて「この部分が，こうなるといいんだけど...」というような示唆を開発者たちに与えることはあるが，個々の開発者に具体的な作業を指示することはほとんどない。Torvalds はメーリングリストへの投稿を非常に積極的に行っている。また，前述のように開発者たちから提出されるソースコードの中から，採用するソースコードを選別することで暗示的にLinux開発の方向性をプロジェクト全体に示している。メーリングリストでの示唆やソースコードの選別を通じた方向性の暗示は，いずれも，プロジェクト全体に対してオープンに行われ，この公正さが Linux プロジェクト内部での価値と文化の共有を促進し，開発者の求心性の源泉になっている。

3. 流通形態による開発プロセスの違い

さまざまなソフトウェア開発プロセスを見てきたが，実際の開発プロセスは，ソフトウェア製品の流通形態にも大きな影響を受ける。オープンソースソフトウェアの開発プロセスの利点をビジネスとしてのソフトウェア開発へ導入する可能性を探るため，流通形態による開発プロセスの違いを整理する。

（1）カスタムソフトウェア

カスタムソフトウェアとは，特定の顧客の要求に対応して，個別に開発されるソフトウェアである。顧客の要望を明確化して仕様定義をすることから開発が始まり，完成したソフトウェア製品の納品で一段落する。その開発のプロセスは，開発するソフトウェアの性格，規模などによって様々である。

納品して，稼働することで，開発の作業は区切りがつくが，それ以降が重要である。なぜなら，後で発見されるソフトウェアの不具合の修正や，環境変化に対応する仕様変更への対応などが必須だからである。カスタムソフトウェアにかかる経費は，開発が完了して稼働するまでと，それ以降とはほぼ同額がかかるといわれている。

稼働後に手直しのバージョン更新はあるものの，大規模な更新はソフトウェア開発への投資を回収するまで待つか，戦略的な意思決定によらなければ行うことはできない。

（2）パッケージソフトウェア

パッケージソフトウェアには，大きく分けて 2 種類の流通形態がある。

一つ目は，業務アプリケーションを対象としたソフトウェアで，基本的な業務分野で標準的な処理をシステムとして提供し，顧客の環境に合わせてカスタム化作業を行うことで，提供するパッケージソフトウェアである。完全に要求にあったソフトウェアになるわけではないが，カスタムソフトウェアの開発経費と保守経費の問題点に対応するものである。開発と保守を多

数の顧客で分け合うことで，コスト低下をしている。カスタム化作業では，顧客の要望が整理されパッケージソフトウェアへの改変が行われ，販売後に判明した不具合についての対応は，顧客ごとに変わる可能性がある。

もう一つは，パソコンソフトウェアのように，まさにパッケージの形をとって店頭で販売されるものである。このパッケージソフトウェアは開発者からすれば手離れがよく，販売後は使用方法などに関するサポートが中心となる。販売後のバージョン更新は原則的に行われず，それだけの品質水準を達成していることが販売を行う前提となる。販売後に致命的な不具合が発生した場合には，修正版の配布が無償で行われることもあるが，非常に例外的である。手離れがよいからこそ，大量生産，大量販売によって，ソフトウェア製品の価格を低く抑えることが可能になる。

パッケージソフトウェアの大規模なバージョンアップは，機能面での大きな拡張が期待される。単なる不具合の対応などによる信頼性の向上だけで，新バージョンを販売しようとすれば，旧バージョンの顧客は信頼性の低い製品を買わされたことになり，旧バージョンへの保証の範囲内で提供するように求めるだろう[15]。したがって，新バージョンを適正な対価で購入してもらうためには，旧バージョンと明らかな差別化がされている必要がある。

（3）オープンソースソフトウェア

オープンソースソフトウェアの開発とバージョン更新については，前述の通りである。流通は，インターネットでの配布が一般的であるが，CDROM などの媒体に記録して販売するビジネスも存在する。

CDROM は，インターネット回線の容量が十分でなかったころに，インターネットからの入手が困難な利用者を対象に始まったものであるが，オープンソースソフトウェアの利用者の底辺が広がり，Microsoft の Windows オペレーティング・システムで稼働するものも増えてきていることから，十分な知識のない利用者でもコンピュータに設置して利用できるような機能を付加することでパッケージ化して継続している。

形態としてはパッケージソフトウェアになっているが，そのオリジナルはインターネットから自由に入手できるものであるから，参入障壁は低く，高価な価格設定をすることはできない。また，バージョン更新も適宜行われるため，利用者にとっては受容可能な価格で流通している。

15) このために，新バージョンの発売時に，旧バージョンの顧客に安価な価格で新バージョンを提供することが多い。

4．オープンソースソフトウェア開発プロセスからのインプリケーション

オープンソースソフトウェアの開発プロセスから，カスタムソフトウェアの開発に適用の可能性がある部分は，保守段階である。カスタムソフトウェアは，特定企業の所有物であり独占的に利用されていることが多いが，その運用，保守のコストは多大である。これを複数の企業で配分できるのであれば，独占的な利用にこだわる必要はない。

実際に，外食産業のニュートーキョーでは，食材発注システム「セルベッサ」を外部委託開発した後にオープンソースソフトウェアとして公開している[16]。このような行動に出るためには，ソフトウェアの対象領域が本業の中核ではない[17]，対象領域に興味を持つ企業が複数ある[18]，などの条件が必要であるが，今後，このような形でのソフトウェア開発は増えていくだろう。

パッケージソフトウェアの場合，物理的な製品としてパッケージが存在するために，頻繁なバージョン更新は困難である。しかし，オープンソースソフトウェアの開発プロセスは二つの方向で適用され始めている。

一つ目は，Microsoft がオペレーティングシステム製品で提供している Windows Update 機能である。インターネットが普及して，パソコンをインターネットに接続して利用することが当たり前になったことで，インターネット経由で頻繁な更新を行っている。この更新は，不具合への対応が中心である。その裏で，次世代製品の開発を進めているのであるから，製品のソースコードや開発のプロセスを公開してはいないが，開発の進め方，利用者への製品の提供方法などは，オープンソースソフトウェアと非常に似通ったものといってよい。

もう一つは，シェアウェア[19] の流通で見られる。ライセンスの発行によって，利用者の識別が可能になっているのだから，開発者はソフトウェア製品の改善のために頻繁なバージョン更新ができる。更新バージョンをインターネットに順次公開し，ライセンスを所有する利用者は

---

16)「Linux と Java の魅力を訴える」，日経ソリューションビジネス，2000.2.21.

17) ビジネスの中核は戦略的な重要性から，システムの公開がビジネスモデルの開示につながるおそれがある。自由化以前の銀行業界では，銀行の中核システムの開発経費を確保するため，中核都市銀行の開発したシステムを系列の地銀などにパッケージ化して販売していた。しかし，当時の銀行業界は横並びで，戦略的に重要な意味を持つビジネスモデルがシステム化されていたわけではない。

18) オープンソースソフトウェアの開発プロセスの利点を活用しようとすれば，当然必須である。利用者数がある程度確保できなければ，他者による不具合の指摘，対応，新機能の追加などが期待できない。

19) シェアウェアとは，インターネットなどオンラインで流通し，期間限定，あるいは，機能限定バージョンを無償で自由に利用できるが，料金を支払うことでライセンスを得ると完全バージョンとして利用できるようになる。なかには，利用者の良心に依存し，最初から完全バージョンを提供しているものもある。
　アマチュアプログラマが開発したソフトウェアを配布したのが始まりであるが，今ではビジネスとして，企業がシェアウェアの形でソフトウェア製品の販売経路として利用する例も出てきている。パッケージの製造，流通経路が不要なので，資金が少なくてもソフトウェアビジネスに参入できるからである。

必要に応じて入手し, 所有するライセンスを適用することで, 新バージョンが利用できる。パッケージの形でのソフトウェア製品を持たないことで，このようなプロセスが可能になる。大規模なバージョン更新にあたっては，新たなライセンス体系を設定し，以前のライセンスでは利用できないようにすれば，新規のソフトウェア製品の販売ができる。ラインセンスの仕組みでソフトウェア製品が販売できるが，パッケージとしての製品を持たないことによって，オープンソースソフトウェアと同様のバージョン更新ができる。

ま　と　め

本稿では，まず，オープンソースソフトウェアの開発プロセスを一般のソフトウェア開発プロセスとの比較として整理した。その上で，オープンソースソフトウェアでとられている開発プロセスの利点を一般のソフトウェア開発プロセスへ適用する可能性を検討した。

オープンソースソフトウェアの開発は，主に自発的に参加する開発者によって行われ，開発経費の制約を受けない冗長性のある開発作業が行われている。一方で，自発的な参加者の方向性を整合させるため，これまでのソフトウェア開発プロジェクトとは異なる方法がとられている。

オープンソースソフトウェアの開発プロセスの一部は，さまざまなソフトウェア開発のプロセスと共通点があり，されに，一般のソフトウェア開発のプロセスにすでに取り入れられ始めている。ソフトウェア開発のプロセスは日々変革しており，オープンソースソフトウェアの開発プロセスは，そこに一つのバリーションを加えている。その利点は今後もさまざまなところに適用されていくことが期待される。

オープンソースソフトウェアを巡っては, さまざまな視点から興味深いプロセスが見られる。インターネット上でソフトウェアという知識を作り出すプロセスは，ネットワークを経由したコミュニケーションを通じた知識創造プロセスにインプリケーションを与えるかもしれない。ニュートーキョーによる業務システムのオープンソース化は，当初のコスト削減だけでなく，業界ネットワーク構成の新しいモデルを作り出している。

このような多様な視点からのオープンソースソフトウェアを巡るプロセスの整理は，これからの課題である。