特集：
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スタイルとその変遷

2.1

藤枝和宏
北陸先端科学技術大学院大学
fujieda@jaist.ac.jp

■はじめに

オープンソースソフトウェアの開発スタイルについて述べた文書としては，Eric S. Raymond による「伽藍とバザール（The Cathedral and the Bazaar）」[1] がよく知られている．この文書では，伽藍とバザールの 2 つの開発スタイルを紹介している．1 人か 2，3 人の開発者の手元で開発を進め，一定の品質に達したところで次のバージョンをリリースするのが伽藍モデル．開発中のバージョンを品質にこだわらず次々とリリースし，なるべく多くの人を開発にかかわらせようとするのがバザールモデルである．

実は，この文書が発表される以前から，多くのプロジェクトが，後に Raymond によってバザールモデルと名づけられるスタイルを採用しており，そのいくつかでは，より進んだ開発スタイルがとられていた．Raymond による伽藍とバザールという分類は，バザールモデルを特徴付けるためには必要だったのかもしれない．しかし，発表された 1997 年当時のオープンソースソフトウェアの開発スタイルを説明するものではなかった．

そこで本稿では，「伽藍とバザール」では述べられていなかった，1990 年代初頭にバザールモデルが現れ始めてから，現在に至るまでのオープンソースソフトウェアの開発スタイルの変遷を，「ソースコードの公開方法」と「開発チームの運営方法」の 2 つの観点で説明する．

■ソースコードの公開方法

バザールモデル以降，オープンソースソフトウェアの開発スタイルは大きく変遷を遂げていくことになるが，実は基本的なところは昔からほとんど変わっていない．まず，特定の個人かグループが自分（たち）の開発したソフトウェアのソースコードをインターネット上に広く公開する．そのソフトウェアに興味を持ったユーザの一部が，バグ報告や要望だけでなく，公開されているソースコードを元にバグフィックスや改良を行った結果を開発者にフィードバックする．開発者はその結果を利用しながら開発を進めていく．

バザールモデルの登場で大きく変化したことの 1 つは，ユーザがバグフィックスや改良に利用するソースコードが，開発者がリリースしたものから開発途中のものに変わっていったことである．

－リリース版のソースコードを公開する

初期のオープンソースソフトウェアでは，ユーザに公開されていたソースコードは，開発者が世に出すとふさわしいと判断してリリースしたものだけだった．そのソースコードをもとにユーザが行ったバグフィックスや改良結果は，開発者が次のバージョンをリリースするまで公開されない．開発者が重大だと判断したバグの修正は，リリース版のソースコードに対する差分（パッチ）として公開されることはあったが，その頻度もそう高いものではなかった．

この方式では開発状況がユーザには分からないため，バグを発見したり改良案を思いついたりしたユーザが，すでに開発者やほかのユーザが行っているかもしれないと考え，自らバグフィックスや改良を行わないことがあった．また，パッチの公開やバージョンアップが長引くと，ユーザが重要だと判断したバグの修正や改良のためのパッチを「非公式パッチ」と称して独自に公開してしまうことがあった．バージョンアップが長く止まってしまったソフトウェアでは，非公式パッチが数多く公開されて収拾がつかなくなることもあった．

－頻繁にリリースする

それに対して，Linus Torvalds は 1991 年に始めた Linux カーネルの開発において，開発中のソースコードを自分の手元にあまり長くとどめずに，新しいバージョンとして次々とリリースするアプローチをとった．ユーザは Linus の手元にあるものに近いソースコードを参照できるため，ほかのユーザや Linus 自身によるものとの重複を恐れずにバグフィックスや改良に取り組めるようになり，非公式パッチの流通も抑えることができた．

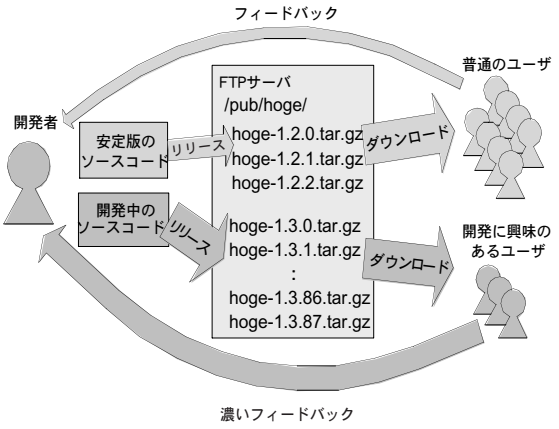

図 -1　頻繁にリリースする開発スタイル

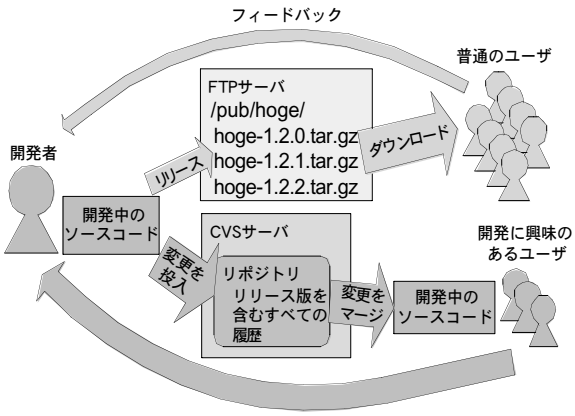

図 -2　匿名 CVS サーバを利用するスタイル

ただし，これをそのまま実践すると，ほとんどのバージョンが多数のバグを含んだままリリースされることになるため，どのバージョンが日常の利用に耐えるのかが，普通のユーザには分からなくなってしまう．そこで，Linux カーネルの開発では，安定版と開発版の 2 つのバージョンを用意し，前者のバージョンアップは無難なものにとどめ，後者でこの方式を用いている．さらに，両者の区別を明確にするために，バージョン番号の 2 桁目（マイナーバージョン）に，安定版では偶数を，開発版では奇数を用いている．この方式の概観を図 -1 に示す．

このスタイルは次第に多くのソフトウェアで採用されるようになり，現在でもいくつかのソフトウェアで採用されている．

－スナップショットを公開する

前述の方式の亜種として，開発中のソースコードを新たなバージョンとしてリリースするのではなく，スナップショットとして公開するソフトウェアも現れてきた．スナップショットを公開する際には，いつのものかを識別するためにバージョン番号として日付を用いたり，バージョン番号を「current」として常に最新のものだけを公開したりした．スナップショットを公開するタイミングは，最初は開発者が決めているソフトウェアが多かったが，毎日あるいは毎週といった定期的なタイミングで，ある程度自動的に行うようにするソフトウェアも増えていった．

－匿名 CVS サーバで公開する

CVS（Concurrent Versions System）[2] はソフトウェアや文書の変更を管理するバージョン管理システムであり，1990 年にバージョン 1.0 がリリースされた．CVS を利用すると，ファイルを多数含むソフトウェアのバージョン管理が容易に行えるため，オープンソースソフトウェアの開発者たちは，次第にこれを用いて開発を進めるようになっていった．

当時の CVS では，ネットワークを介して複数人で共同開発を行う際には，変更履歴を格納したリポジトリを NFS を用いて各ホストで共有する必要があったため，CVS を用いた共同作業は基本的に LAN の中でしか行えなかった．この制約は，1995 年にリリースされた CVS バージョン 1.5 に，サーバ・クライアント機能が導入されたことで解消された．このバージョンでは，CVS 内部の処理をサーバとクライアントに分割して，インターネットを介した分散共同作業を可能にしていた．

このバージョンがリリースされて以降，ソースコードの共有方式として，匿名 CVS サーバが次第に用いられるようになっていった．これは，不特定多数のユーザが読み出し可能な CVS サーバ（匿名 CVS サーバ）を用意して，開発者が用いている CVS のリポジトリをインターネット上に公開するものである．この方法を用いると，ユーザは CVS クライアントを用いて，リポジトリから最新のソースコードを取得したり，取得済みのソースコードに開発者の加えた変更を反映させたりできる（図 -2）．

匿名 CVS が利用されるようになってからは，単に最新の開発状況を公開するためだけに，開発者がソースコードを頻繁にリリースすることは，あまり行われなくなっていった．ちなみに「伽藍とバザール」が発表されたのは，この後のことだ．

－ SourceForge を利用する

匿名 CVS サーバで公開するアプローチは，有効ではあるものの誰もが実践できるものではなかった．「頻繁にリリースする」とか「スナップショットを公開する」といったアプローチは，従来からソースコードの公開に利用している FTP サーバや WWW サーバ上で，そのまま実践できるのに対して，匿名 CVS サーバを導入するには，そのためのサーバを新たに設置しなければならないからだ．

1993 年にリリースされ，一部のオープンソースソフトウェアの開発で用いられて効果を発揮していた，バグ追跡システム GNATS[3] の導入も，同様の理由であまり進まなかった．GNATS はユーザからのバグレポートの受け付けと，受け付けたレポートの管理や検索を支援する．これを利用すると，ユーザによるバグレポートは定型フォームをもとに行われるため，開発者にとって必要

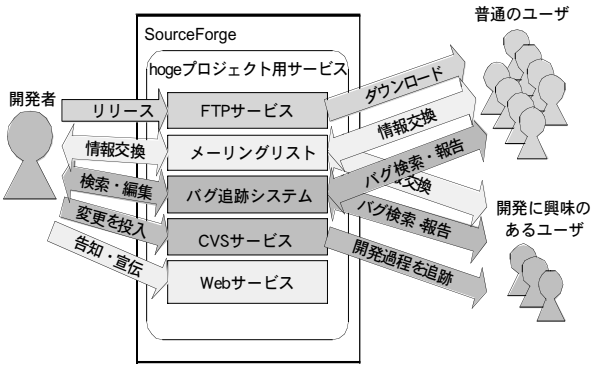

図 -3　SourceForge を用いた開発スタイル

な情報が欠落するのを防ぐことができる．また，ユーザが受け付け済みのバグを確認できることから，同じバグに関する報告が何度も開発者に寄せられるのを防げたりする．

オープンソースソフトウェアの開発者が，技術的あるいは金銭的な理由で，有用であることが知られている技術を導入できない問題は，2000 年 1 月に登場した SourceForge によって解決された．

SourceForge[4] は，オープンソースソフトウェアの開発に必要な環境を，開発者に無償で提供するサービスであり，オープンソースソフトウェアのコミュニティを支援する OSDN（Open Source Development Network）によって運営されている．SourceForge では，匿名 ftp サーバ，WWW サーバ，CVS サーバ，メーリングリストサーバ，バグ追跡システムなど，オープンソースソフトウェアの開発に必要なものを一通り提供しているだけでなく，ほとんどのサービスは Web ブラウザ上で利用可能になっている．SourceForge を用いた開発スタイルの概観を図 -3 に示す．

SourceForge が登場してからは，多くのオープンソースソフトウェアの開発者が，SourceForge 上の CVS サーバを利用して開発を進めるようになり，その過程を格納したリポジトリは，匿名 CVS サーバや Web ベースのインタフェースを通じて，広くインターネット上に公開されるようになった．

■開発チームの運営方法

最初に述べたように，オープンソースソフトウェアの開発は，バグ報告や要望だけでなく，バグフィックスや改良を行った結果を送ってくれるユーザの助けを借りながら進められる．このユーザは「コントリビュータ（Contributor）」と呼ばれる．ソースコードの公開方法の工夫により，熱心なコントリビュータを多く獲得することに成功したソフトウェアでは，開発者とコントリビュータが，インターネットによって緩やかに結び付けられた，1 つの開発チームとして機能するようになった．この開発チームの運営方法もバザールモデルの登場以来，いくつかの変遷を遂げてきている．

－メインテナの採用

Linus は Linux カーネルの開発を始めてすぐに，貢献度の高いコントリビュータに開発の一部をまかせて，彼自身はなるべく全体の舵取りとリリース管理だけを行うようにしていた．この開発の一部をまかされたコントリビュータを「メインテナ（Maintainer）」という．Linux カーネルでは，ソースコードのパッケージに含まれている MAINTAINERS というファイルに，メインテナの名前，担当分野や連絡先などが列挙されている．

－コアメンバの採用

いくつかのオープンソースソフトウェアでは，熱心なコントリビュータに開発の一部を任せるだけでなく，全体の舵取りやリリース管理などについても，開発者と同じ権限を与えていた．この最初の開発者と同じ権限を持ち，開発チームの最上位に位置する人たちを，最初の開発者も含めて「コアメンバ（Core Member）」あるいは「コアチーム（Core Team）」という．

この形態は，最初の開発者がコントリビュータに明示的に権限を与える場合よりも，最初の開発者が開発をやめてしまい，やむなく主要なコントリビュータでコアチームを形成し，開発を継続するかたちで生じることが多かった．この例としては，386BSD の後を受けて始められた FreeBSD[5] や，NCSA httpd の後を受けて始められた Apache[6] がある．

－コミッタの採用

CVS1.5 で導入されたサーバ・クライアント機能は，ソースコードの公開方式に大きな影響を与えたが，開発チームの運営方法にも大きな影響を与えている．

CVS1.5 以前は，コントリビュータがメインテナやコアメンバになったとしても，最終的に開発中のソースコードを変更できるのは，それを持っている開発者だけだった．CVS1.5 以降では，開発者のリポジトリのソースコードに，インターネットを介して変更を加えることが可能になったため，開発者はコアメンバやメインテナにリポジトリの変更権限を与え，彼らが自分の手を煩わせることなく，その役割を果たせるようにした．

CVS1.5 以降では，開発者がメインテナやコアメンバの役割をコントリビュータに与えずに，リポジトリの変更権限だけを与えてしまうことも行われるようになった．CVS のリポジトリに変更を加えるコマンドが commit であったことから，彼らは「コミッタ（Committer）」と呼ばれる．コミッタは，リポジトリ内のソースコードに何か変更を加える際には，事前に開発者やほかのメンバの了承を得た上で行う．

コミッタの採用は主に開発者の手間を省く目的で行われているが，コントリビュータに開発チームの一員としての自覚をうながす効果もある．

	Mozilla	Apache	Samba
Core	Driver	PMC	Samba Team
Maintainer	Module Owner	Committer	
Committer	Committer		
Contributor	Contributor	Developer	Contributor

図 -4　チーム構成の実例

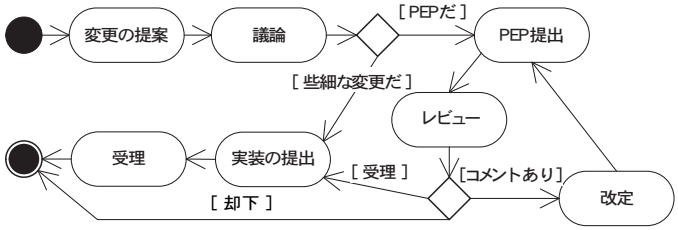

図 -5　PEP に関するワークフロー

－チーム構成の実例

これまで述べてきた，オープンソースソフトウェアの開発チームを構成する構成員を，権限の強い順に並べると，コアメンバ，メインテナ，コミッタ，コントリビュータとなる．実際には，この 4 つの構成員がすべて存在するケースはまれである．また，各階層の呼び名もソフトウェアによってまちまちだ．

たとえば Mozilla では，コアメンバはドライバ（Driver），メインテナはモジュールオーナ（Module Owner）と呼ばれている [7]．Apache では，コアメンバに相当する PMC（Project Management Committee）が設けられており，その下にコミッタ，コントリビュータに相当するデベロッパ（Developer）と続く．Samba にはコアメンバに相当する Samba Team しかない [8]．これらのチーム構成をまとめたものを図 -4 に示す．

－意思決定の方法

オープンソースソフトウェアが，1 人かせいぜい数人の開発者で開発が行われているうちは問題にならなかったのだが，開発チームの人数が増えてくると，ソフトウェア全体に影響を及ぼすような大きな変更に関する意思決定が難しくなるケースが出てきた．

最初の開発者か，その開発者が承認した正当な後継者がいるオープンソースソフトウェアであれば，最終的な判断をその人に委ねることができる．その人を「やさしい独裁者」と呼び，この方式を「やさしい独裁者モデル」と呼ぶ [9]．多くのオープンソースソフトウェアで用いられているのは，この方式である．

やさしい独裁者を持たない Apache プロジェクトでは，意思決定を 2 種類の投票を用いて行っている．1 つはコミッタ以上の構成員で行われて，反対が 1 つもなく 3 票以上の賛成で可決する consensus approval．もう 1 つは，コントリビュータも含め全員で行われて，賛成が反対よりも多く，賛成票にコミッタ以上の票が 3 つ以上含まれている場合に可決する majority approval である．

これらの方式を利用している場合においても，オープンソースソフトウェアの開発では，なるべく参加者間で議論を重ねて合意を形成しようとする傾向がある．そこで，議論を円滑に進めるためと，その結果を保存するという 2 つの目的で，Python や Perl6 では，大きな変更を加えようとする参加者に，一定のルールに従ってその変更について記述した文書を事前に公開することを求めている．この文書をもとに参加者間で議論を進めながら文書を何度か改訂し，コアメンバがその文書の内容を了承した後に，実際の変更が行われる．

この文書は Python では PEP（Python Enhancement Proposal）[10]，Perl6 では RFC（Request for Comments）[11] と呼ばれる．Python の PEP に関するワークフローを図 -5 に示す．いずれも，インターネットの標準規格を決める際に用いられている RFC にならったものである．RFC については参考文献 12）の 4 章を参照してほしい．

■まとめ

かなり駆け足だが，オープンソースソフトウェアの開発スタイルの変遷を「ソースコードの公開方法」と「開発チームの運営方法」の 2 つの視点で追ってみた．本稿では触れていない 1990 年以前の動向や，オープンソースソフトウェアを取り巻く環境の変遷については，参考文献 12）にさまざまな立場の人が，それぞれの視点で述べたものがあるので参照してほしい．

「伽藍とバザール」を読んで，オープンソースソフトウェアの開発は，どんどんリリースして，ユーザのフィードバックをどんどん取り込みさえすればうまくいくものだと誤解してしまった人は多いかもしれない．本稿が，その誤解を解く助けになれば幸いである．

参考文献

1）Raymond, E. S. 著，山形浩生訳：伽藍とバザール，http://cruel.org/freeware/cathedral.html
2）Concurrent Versions System: The Open Standard for Version Control, http://www.cvshome.org/
3）GNATS: http://www.gnu.org/software/gnats/
4）SourceForge.net: http://sourceforge.net/ :http://sourceforge.jp/（日本語版）
5）FreeBSD プロジェクトについて：http://www.freebsd.org/doc/ja_JP.eucJP/books/handbook/history.html
6）About the Apache HTTP Server Project: http://httpd.apache.org/ABOUT_APACHE.html
7）Hacking Mozilla: http://www.mozilla.org/hacking/
8）The Samba Team: http://www.jp.samba.org/samba/team.html
9）Raymond, E. S. 著，山形浩生訳：ノウアスフィアの開墾，http://cruel.org/freeware/noosphere.html
10）PEP Purpose and Guidelines, http://www.python.org/peps/pep-0001.html
11）About Perl6 RFCs, http://dev.perl.org/rfc/meta/
12）DiBona, C. 他編著，倉骨彰訳：オープンソースソフトウェア：彼らはいかにしてビジネススタンダードになったか，オライリー・ジャパン，http://www.oreilly.co.jp/BOOK/osp/OpenSource_Web_Version/Web_version000106.html

（平成 14 年 11 月 5 日受付）