人工知能学会
第20回知識流通ネットワーク研究会
SIG-KSN-020-07

オープンソース・ソフトウエア・コミュニティの コストモデルと運用最適化に関する考察

○桑田 喜隆[†1] 石坂 徹[†1] 横山 重俊[†2†3] 合田 憲人[†3]
[†1] 室蘭工業大学
[†2] 群馬大学
[†3] 国立情報学研究所

A study on a cost model of OSS community and the optimization of operation cost

Yoshitaka Kuwata[†1], Toru Ishizuka[†1], Shigetoshi Yokoyama[†2†3], and Kento Aida[†3]
[†1] Muroran Institute of Technology, Japan
[†2] Gumma University, Japan
[†3] National Institute for Informatics, Japan

概要

オープンソース・ソフトウェア(OSS)はOSSコミュニティによって開発され維持管理されるソフトウェアである．筆者らはこれまで利用者の立場から運用コストの最適化について論じた．本稿では，開発者側のコストを含めたコストモデルを提案するとともに，OSSコミュニティ全体の最適化に関して考察を行う．

Abstract

Open Source Software (OSS) is developed and maintained by OSS communities. In the previous paper, we studied optimization of operation-cost for users of OSS products. In this paper, we carry out simulations with Wordpress and Moodle in order to evaluate release-cost for OSS development projects. We discuss the optimization of both operation-cost and release-cost.

1. はじめに

オープンソース・ソフトウェア製品(以下, OSS)は OSS コミュニティによって開発, 維持管理がされるソフトウェアである. 基盤ソフトウェア(OS)からミドルウェア製品, アプリケーションソフトウェアまで幅広く開発されている．ソースコードが公開されているため，自分達のニーズに合わせて変更することが可能である．また，変更内容を開発側に還元することで, 開発プロジェクトに参加することが可能である．

他方, OSS を組み合わせてシステムの運用を継続して行く場合, 利用しているソフトウェアの維持保守が大きな課題となる．一般に OSS は開発の速度が速く, 新しい版のリリースにあたり大きな変更が行われることも多い. 運用中のシステムのアップデートを行うことは, コストがかかりリスクを伴うため, 必ずしも最新の版を利用できない場合もある. 旧版を継続して使い続けることも可能であるが, いずれは維持保守が中止される. ソフトウェアに脆弱性が発見されても修正されないため, 古い版を使い続けることには問題がある．従って，OSS であっても計画的にアップデートを実施して行くことが必須であると考える．

筆者らは，これまで[1]において利用者の立場から OSS の運用コストのモデルを構築し，その最適化の方法について論じた. そこでは，開発プロジェクト側の維持保守にかかるコストや，ポリシーに関しては考慮せず，利用者として提供さているソフトウェアをなるべく低コストで運用して行く方法に関して論じた. OSS は利用者側からのフィードバックをもとに開発を行うべきであり，本来の OSS プロジェク

1 Yoshitaka Kuwata
室蘭工業大学
北海道室蘭市水元町２７−１
kuwata@mmm.muroran-it.ac.jp

トの趣旨とは異なる.

そこで本稿では，OSS コミュニティ側から開発プロジェクトに関わる維持管理ポリシーおよびそれに伴うコストについて論じる. 更に, 開発プロジェクトのコストと利用者側のコストのトレードオフに関して論じる.

2. OSS に関する関係モデル

まず，OSS に関するステークホルダおよびそれらの関係を整理する. 図 1 に本稿で前提とする OSS の関係モデルを示す.

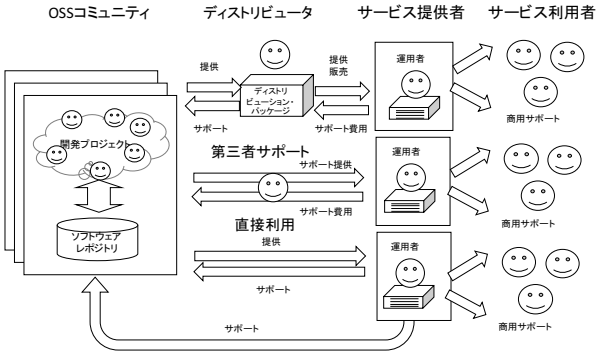

図 1 OSS の関係モデル

OSS はその歴史や規模などによって組織の形態や関係するステークホルダが大きく異なる. ここでは商用で利用されているソフトウェアを開発している，比較的大きな OSS コミュニティを対象としてモデル化している.

以下に，図 1 に示した関係モデルの主なステークホルダに関して説明する.

(1) OSS コミュニティ

OSS プロジェクトの運営組織. 複数の開発プロジェクトを主管している場合がある.

例：Linux Foundation, Apache Software Foundation

(2) 開発プロジェクト

OSS コミュニティの下で, ソフトウェア製品の開発を行う組織（グループ）

例 : Linux kernel project[2], Apache Hadoop Project[3]

(3) ソフトウェア・レポジトリ

開発中のソフトウェアを格納する場所としてソフトウェア・レポジトリが利用される. ソフトウェアのバージョン管理の機能を持ち，任意の時点のソースコードを取り出したり，差分の管理を行うことができる.

例：Github, sourceforge

(4) ディストリビュータ

複数の OSS 製品をまとめてすぐに利用可能なパッケージ化して提供する者

例：Redhat linux[4], Ubuntu Linux[5]

ディストリビュータで扱かっていない OSS 製品では，開発プロジェクトでパッケージを配布している場合もある.

(5) サービス提供者

OSS を利用してシステムを構築し, サービスを提供する者. 商用サービスおよび組織内サービスの場合を含む.

(6) サービス利用者

上記サービス提供者からサービスを受ける者. OSS を利用しているかどうかは，意識しない場合が多い.

3. OSS のリリース管理

リリース管理は, 開発中のソフトウェアを正式版として公開し, 古いソフトウェアの更新を中止するタイミングを決定する作業である. OSS のリリース管理方法は，OSS コミュニティや開発プロジェクトごとに異なるが, ここでは一般的なリリース管理について述べる.

3.1 バージョン, リビジョン, パッチ番号の付与

ソフトウェアを公開する際に, そのソフトウェアを識別するための，番号や名称，日付等を付与する.

一般的には, 前後関係が分かるように番号が使われることが多い. 以下に付与方法の例を示す.

(1) バージョン番号

機能や性能を改善した版を出す場合，バージョン番号を変更する. 細かなバージョンの表記のためには，メジャーバージョンとマイナーバージョンを利用する. 著しく機能を変更した場合には，メジャーバージョン（1 桁目の数字）を，それ以下の場合にはマイナーバージョン（2 桁目の数字）を変更する.

例： Linux Kernel 2.6
Moodle 3.1

マイナーバージョンが一桁(9)より大きくなる場合にも，メジャーバージョンを上げない場合が多い. 例えば, バージョン 3.9 の次は, バージョン 4.0 ではなく，バージョン 3.10 を付与する.

(2) リビジョン番号

バージョンより細かな管理の必要な場合に

は，バージョン番号より細かい単位としてリビジョン番号を利用する．
例：Linux Kernel 2.6.10
Moodle 3.1.2

(3) パッチ番号

不具合修正などはリビジョン番号より細かな単位として，パッチ番号を用いる．
例：Linux Kernel 2.6.10-23

3.2 ソフトウェアの維持保守ポリシー

[1]で議論した通り，新しい版を公開した後，古いソフトウェアがいつまで保守されるかは，利用者側コストに大きな影響を与えるため，OSS 採用判断の重要なポイントの一つである．

図２に，ソフトウェアのリリース管理例を示す．

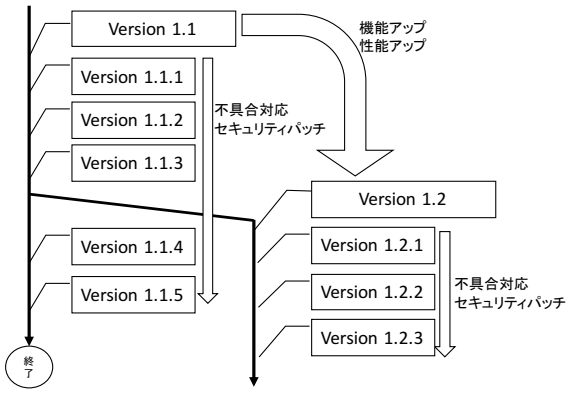

図 2 ソフトウェアのバージョン管理例

Version 1.1 の公式リリース（以下，公開）後，ソフトウェアの不具合対応やセキュリティパッチとして Version 1.1.1 から Version 1.1.3 を公開する．同時に次の版の開発が行われ，準備ができた段階で Version 1.2 を公開する．

Version 1.2 の公開後，ソフトウェアの不具合対応やセキュリティパッチとして Version 1.2.1 から Version1.2.3 を公開するが，Version 1.1 系列のソフトウェアに対しても同様の修正を行い，Version 1.1.4 および Version 1.1.5 を公開する．

古い版に対して対応作業を継続して行くと版を重ねるごとに作業のためのコストが増加するため，古い版の更新はいずれ中止することが必要になる．図では Version 1.1.5 で Version 1.1 系列のサポート終了としている．

4. ケーススタディ

ケーススタディとして，実際の OSS 開発プロジェクトの維持コストを試算した．
試算にあたり，以下を仮定した．

- ソフトウェアの公開ごとにコストがかかる．
- 簡単化のため，公開に関するコストは同一とし，公開回数のみを測定してコストとする．
- テスト用にベータ版や公開候補(Release Candidate,RC)版を公開しているプロジェクトでは，それらもコストとして考慮する．

Wordpress[6,7]プロジェクトおよび Moodle[8,9]プロジェクトのリリース履歴をホームページより調査した．以下に結果を述べる

4.1 Wordpress のケース

Wordpress は人気の高い Web ベースのコンテンツ管理システムである．PHP 言語で記述されており，Linux の Apache HTTP サーバ上で動作する．バックエンドとしてデータベースシステムが必要になる．

図３に 2005 年から 2016 年までの各年の Wordpress の公開回数の時系列変化を示す．

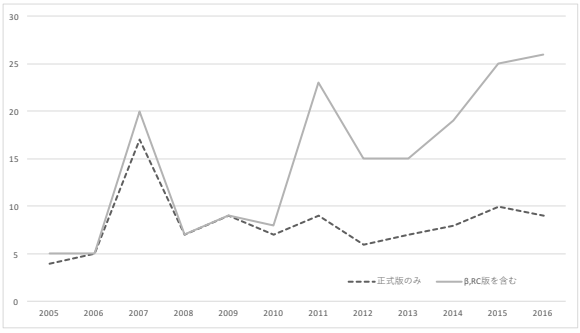

図 3 Wordpress の公開回数の時系列変化

Wordpress は最新版のみをサポートする方針であるため，不具合対応を行うバージョンは常に１個のみである．このため，正式版の公開回数を見ると，10 回/年以下と経年で増加していないことが分かる．但し，2007 年は例外的に公開回数が多い．

また，Wordpress が広く普及してきた 2011 年以降は，正式公開前にベータ版や RC 版の公開を行うようにして品質の確保を行っている．このため，それらを含めた公開回数は，2011 年以降で，それ以前に比べて 2 ないし 2.5 倍に増加していることが分かる．

4.2 Moodle のケース

Moodle は最も利用者の多い学習管理システム(Learning Management System)の一つである．ソフトウェア構成は Wordpress と類似している．PHP 言語で記述されており，Linux の Apache HTTP サーバ上で動作する．商用製品を含めて複数のデータベースがサポートされている．

図 4 に 2005 年から 2016 年までの各年の Moodle の公開回数の時系列変化を示す．

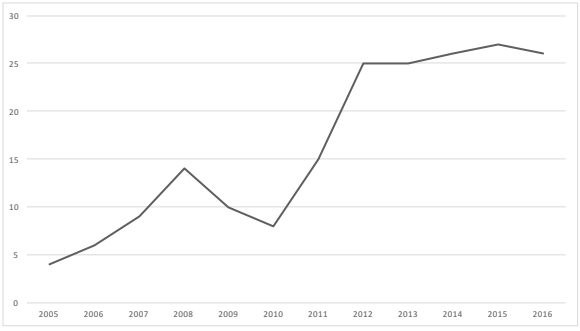

図 4 Moodle の公開回数の時系列変化

Moodle は公開後 2 年間程度のサポートを継続しているため，Wordpress に比べて公開回数が全体に多くなっている． 特に，2012 年以降はサポートを手厚く実施したため，公開回数も 25 回/年程度に増加している．

Wordpress, Moodle とも 2011 年から，年間 20 前後の公開を行っている．これは，ソフトウェアの普及が進み，公開前のテストなどが求められるようになってきたため，コストが増大したと考えられる．

5. サポートコスト・シミュレーション

本章では，Wordpress 及び Moodle の公開ポリシーの評価のため，両ソフトウェアの実際の公開履歴データをもとにサポートコスト・シミュレーションを行う．

本シミュレーションでは，現実ではサポートの中止された版のサポートが，仮に継続されていた場合に，どれだけのコストがかかったかを推定する．このため，以下の仮定を置いた．

- ある版で修正された不具合やセキュリティパッチは，それ以前の版にも必要である．（実際には，機能アップの途中で作り込まれた不具合である可能性もある．）
- ある版の実際の 1 年間の公開数だけ，サポートされなかった版に対しても修正が必要である．但し，機能検証が目的のベータ版および RC 版は除く．

また，比較のため以下の条件でシミュレーションを行なった．

(1) 現行サポートポリシー
　実際に開発プロジェクトで行った場合
(2) 永年サポート
　公開した全ての版のサポートを継続する
(3) ５年サポート
　公開後５年間サポートを継続する
(4) ３年サポート
　公開後３年間サポートを継続する
(5) 最新サポート
　最新版のみをサポートする

5.1 Wordpress のケース

Wordpress の公開回数シミュレーション結果を図５に示す．

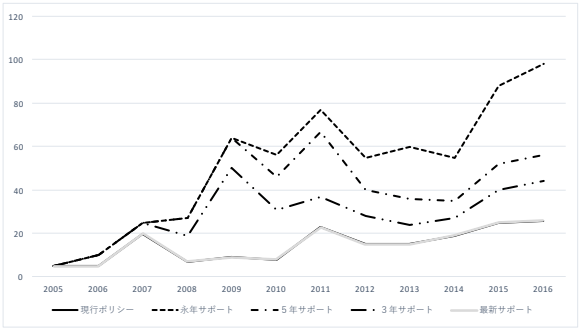

図５Wordpress の公開回数シミュレーション結果

永年サポートは版が累積するため，年を追うごとに公開回数が増加する．5 年サポート，3 年サポートは現行の公開回数に比べ，それぞれ 2.6 倍，1.9 倍になっている．また，現行ポリシーは最新サポートと同じ結果である．

表１に Wordpress のシミュレーション結果である公開回数の平均を示す．

表１ Wordpress の年平均公開回数

ポリシー	平均公開回数
現行ポリシー	14.8
永年サポート	51.7
５年サポート	38.6
３年サポート	28.3
最新サポート	14.8

5.2 Moodle のケース

Moodle の公開回数シミュレーション結果を図 6 に示す.

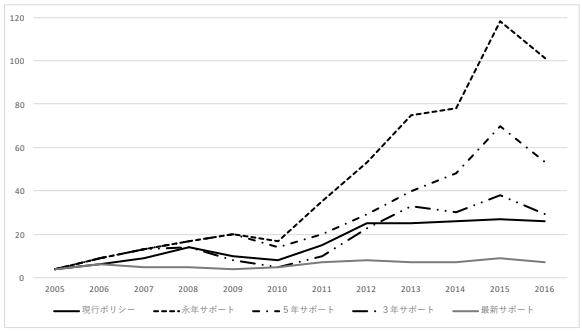

図 6 Moodle の公開回数シミュレーション結果

永年サポートは年を追うごとに公開回数が著しく増加する．2015 年以降は年間 100 回以上になり，現実的ではない．5 年サポートは現行の公開回数に比べ，1.7 倍であるが，3 年サポートは現行ポリシーとほとんど同じになっている.

最新サポートは現行ポリシーより公開回数が少なくすみ，平均 6 回程度である.

表 2 に Moodle の公開回数平均を示す.

表 2 Moodle の年平均公開回数

ポリシー	平均公開回数
現行ポリシー	16.3
永年サポート	45.0
5 年サポート	28.1
3 年サポート	18.0
最新サポート	6.2

6. 考察

6.1 長期のサポート戦略

OSS 開発プロジェクトは新しい機能を開発し，継続的にソフトウェアを公開することで，ソフトウェア製品としての魅力を維持している.年に数度の公開を行うことを目標としている開発プロジェクトも多い．しかし，人的なリソースの制限もあるため,プロジェクトを継続して行くためには,メンテナンスコストの増加は避けるべきである.

他方,利用者側はアップデートのコストを避けるため,可能な限り長くサポートを継続してもらいたい．そこで，提供側で可能な戦略として，サポート期間をあらかじめ設定し，期間終了までに利用者にバージョンアップを促すことが現実的であると考えられる.

6.2 サポートポリシーの最適化

近年，長期サポート版(Long Term Support, LTS)を導入しているプロジェクトがある.前章で示した通り,Moodle は LTS を導入しており,コストの上昇を抑える効果が出ていると考えられる.

Wordpress で LTS サポートを行った場合の,公開回数シミュレーション結果を図 7 に示す.

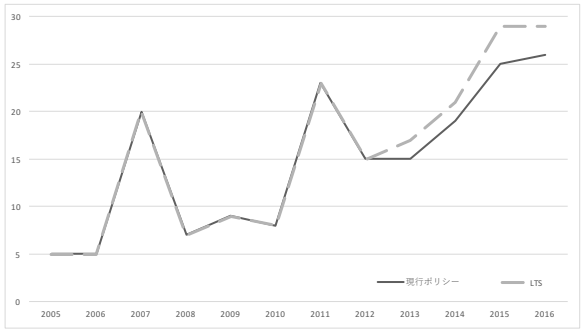

図 7 Wordpress で LTS サポートを行った場合の公開回数シミュレーション結果

上記のシミュレーションでは，3.4 版(2012 年)および 4.0 版(2014 年)を LTS として，公開後 3 年間サポートすることを仮定した.
現行ポリシーに比べ,公開回数が年 4-5 回増加するが,増加割合は一定である．表 1 の 3 年サポートポリシーに比べ公開回数増加が少なく,実現可能なコスト範囲内であると考える.

6.3 利用者コストとサポート

[1]で筆者らは長期にわたりサポートの継続する OSS 製品を採用することでアップデートに関わるコストを下げることができることを示した．他方，本論文ので示した通り，長期サポートを継続すると開発プロジェクトで維持管理のためのリソースを消費し,新しいソフトウェア開発に影響を与える可能性があることがわかった．また，LTS など安定版を長期にわたり提供し続けることでコストを抑えつつ運用コストの低減が可能であることが分かった.

OSS の性質上，ライセンスが認める範囲内で,開発プロジェクト以外の第三者が長期の維持保守を実施する形態も可能である.

開発プロジェクトによる維持保守が終了し

た後まで,安定した版を有償で維持保守を提供するサービスは,利用者の利益にもつながるし,受益者負担の観点からも理にかなう場合もあると考える．但し，ソフトウェアの性質にも依存する．

7. 関連研究

本稿では開発プロジェクトのコストの計算のために，その成果物の一つである「公開」の回数を代替指標として用いた．容易に得られる指標であるため，集計が容易であるが，プロジェクトのコストを正確に反映していない面もある．例えば，大規模な改修を含むメジャー公開とセキュリティパッチ公開を同じ１回として集計している．公開の種類によって重み付けを変えるなどの手法も考えられるが,重みのつけ方などに限界がある．

より詳細なコスト計算を行うためには,開発プロセスの詳細なモデル化が必要になると考える．一つの手法として，ソフトウェア・レポジトリの更新情報を使うことで,より詳細な開発モデルの作成が可能である．

例えば,文献[10]ではレポジトリ上のソフトウェア公開履歴を分析し,開発プロジェクトの分類やリリースサイクルを指標として設定したが,および公開パタンがプロジェクトの発展でどのように変化するかを調査している．開発の詳細モデルとコスト分析を組み合わせて,より詳細なモデルを構築することが可能になると考える．

8. まとめと今後の課題

本稿では，OSS コミュニティ側で OSS 製品にサポート関するコストについての評価を実施した．コストモデルに基づきシミュレーションを実施することで,公開ポリシーの評価が可能であることを示した．

また,評価を実施したWordpress及びMoodleの製品リリース方法は,コストとサポートの有効性のバランスが良いことが分かった．

課題として,前回提案した利用者コストモデルと今回の OSS コミュニティのコストモデルの統合化を実施し,最適化する戦略を検討することがあげられる．

なお，本研究は，国立情報学研究所の平成28 年度公募型共同研究テーマ「複数計算機から構成される研究用計算機環境の自動再構築に関する研究」の研究成果である．

A. 参考文献

1. 桑田喜隆,石坂徹,横山重俊,合田憲人, OSS 製品のサポートポリシーの分析に基づくシステムライフサイクルの最適化に関する考察，第19 回人工知能学会知識流通ネットワーク研究会(SIG-KSN-20), 2016 年 9 月 20 日
2. Linux Kernel Projects, https://kernelnewbies.org/KernelProjects, (2017/2/18 参照)
3. Apache HTTP Server Projects, https://httpd.apache.org/, (2017/2/18 参照)
4. RedHat, Inc., Red Hat Enterprise Linux Life Cycle, https://access.redhat.com/support/policy/updates/errata, (2017/2/18 参照)
5. Ubuntu Linux, https://ubuntu.com/, (2017/2/18 参照)
6. WordPress プロジェクト, WordPress Codex 日本語版, https://wpdocs.osdn.jp/WordPress_バージョン一覧, (2017/2/16 参照)
7. WordPress.org, Releases Category Archive, https://wordpress.org/news/category/releases/ , (2017/2/16 参照)
8. Moodle プロジェクト，https://moodle.org/, (2017/2/16 参照)
9. Moodle プロジェクト，リリース一覧 https://docs.moodle.org/dev/Releases (2017/2/16 参照)
10. Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel. Observations on patterns of development in open source software projects. In Proceedings of the fifth workshop on Open source software engineering. ACM, New York, NY, USA, 2005., Pp.1-5.