JSDSS 日本ソーシャルデータサイエンス論文誌
第 1 巻　第 1 号（2017 年 3 月）

原著論文

オープンソース・ソフトウェア製品の運用モデルおよび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]

概要：オープンソース・ソフトウェア(OSS)はコミュニティによって開発され，ソースコードや開発プロセスが公開されているソストウェアである．ライセンス費用などが不要なこともあり，近年，商用サービスで利用される場合が増加している．OSS を採用しても，運用作業のためのコストがかかるが，プロプライエタリ・ソフトウェアに比べて運用コストの算出が難しい．本稿では，OSS の利用実態に合わせた運用コストモデルを新たに提案する．また，採用実績の多い Wordpress および Moodle を取り上げ，これまでのリリース情報に基づき運用コストの比較を行うとともに，OSS の運用戦略について論じる．

キーワード：オープンソース・ソフトウェア(OSS)，OSS の運用コスト，コミュニティ分析

1. はじめに

オープンソース･ソフトウェア (OSS) はソースコーソや開発プロセスを公開することで，誰でもソフトウェア開発プロジェクトに参画することが可能なソフトウェアである．品質が高く，コストの安いソフトウェアとして期待されている．OSS はライセンスに関する手続きや費用が不要であるため，近年，商用サービスにも広く応用されるようになってきた．特に，不特定多数の仮想マシン上に実装する必要のあるクラウドサービスでは，コスト面で有利であると考えられる．

他方，OSS を採用したシステムを構築する場合であっても，運用に関するコストがかかる．例えば，セキュリティのパッチを適用したり，サポートが切れるタイミングでソフトウェアの入れ替えを行うことが必要となる．商用ソフトウェアの場合には，サポートも含めて提供されるが，OSS の場合にはサポートを行うベンダーを見つけるのが難しい場合もある．このため，運用作業を内部で実施する場合も多く，OSS のサポートに関しては見積りしづらいというデメリットがある．商用ソフトウェアの場合にはサポートの条件や体制などが決まっているため，システムの運用ライフサイクルに関する見積もりがしやすい．

上記の問題に対して，筆者らは文献 [5] においてサポートに求められる要件を整理した．更に，サポート作業中，最もコストのかかるアップデート作業に関して，OSS を取り上げて更新の頻度やそのポリシーをもとに運用コストを試算し比較した．また，結果を比較して OSS の効率的な運用方法について論じた．

本稿では，上記のケーススタディを基に OSS の運用モデルを一般化し，上記ケーススタディで用いた例題に適用しモデルの評価を行う．

本稿の構成は次の通りである．まず 2 章で関連研究について述べた後，3 章で運用モデルの提案を行う．4 章でケーススタディとして Wordpress および Moodle を取り上げる．5 章で最適化戦略について述べ，6 章で考察を行う．7 章でまとめと今後の課題について述べる．

2. 関連研究

2.1 OSS のビジネスモデル・パタン

OSS のサポートは，その母体となる OSS コミュニティの動向に左右される．継続的に OSS がサポートされるためには，コミュニティが活性化し，開発プロジェクトにリソースが提供されることが必要である．従って，OSS のビジネスモデルについての考察が必要になる．

OSS ビジネスモデルのパタンは既にいくつか提案されている．例えば，文献 [8] では，サポートに関連するビジネスモデルのパタンとして次のようなものがあげられている．

(1) **DUAL LICENSE**
OSS および商用の 2 つのライセンスを設ける（製品は分けない）

(2) **DUAL PRODUCT**
ライセンスだけでなく，OSS 製品とクローズド製品を分離する

(3) **IP MODULARITY**
知的財産権を保有する部分と，オープンな部分を明確に分ける

OSS で商用システムを運用する場合，複数のビジネスモ

†1 室蘭工業大学（連絡先：kuwata@mmm.muroran-it.ac.jp）
†2 群馬大学
†3 国立情報学研究所
投稿：2016 年 11 月 30 日
採択：2017 年 2 月 10 日

デル・パタンの OSS を組み合わせる必要がある．本稿で取り上げた基盤ソフトウェアは OS ディストリビューションとして商用サポートが行なわれている．他方，AP は商用サポートの段階にない．運用モデルの構築にあたって，ビジネスモデル・パタンの中で商用サポート有無を意識してモデル化を行うこととした．

2.2 ソースコードレポジトリ解析による OSS の品質評価

近年，IEEE や ACM 主催のソースコード・リポジトリのマイニングに関するカンファレンスが開催されるなど，ソフトウェア工学の分野で OSS の分析が注目されている．

ソースコード･リポジトリとしては, SourceForge, GitHub, Google Code などが挙げられる．特に，近年人気のある GitHub [9] の分析に関しては多くの論文が発表されている．例えば，Tsay ら[7]は GitHub 上でソースコードの変更提案 (pull-request) に関連する議論の流れについてインタビューを通じて分析を行っている．また，Biazzini ら [1] はソースコードの分岐 (fork) に着目してプロジェクトの状態を可視化する方法を提案している．

GitHub Archive [3] は 2011 年以降の GitHub 上での 20 種類以上のイベントを時系列列データとして提供しており，GitHub で行われたソフトウェア変更の過程を追いかけることが可能である．

他方，Kalliamvakou ら [4] は，GitHub の分析に関する限界を指摘している．例えば，ソースコード・リポジトリは必ずしもソフトウェア開発やその議論の場として使われておらず, 成果物の格納場所として使われている場合がある．また， GitHub 全体の commit 数の中央値は 6 であり，多くのプロジェクトでは開発成果を頻繁に更新していないことを指摘している．

このように，一般的にはソースコード・リポジトリのマイニングだけからプロジェクトの状態を推定し運用性の評価を実施することは難しい．そこで本稿では，サポートポリシーの明確になっているソフトウェアを取り上げてそのリリース履歴から運用コストの評価を行うアプローチを採用することとした．

3. 運用モデルの提案

3.1 運用作業モデル

本稿では，OSS を利用して構築するシステムの導入および運用に関して，以下の作業モデルを提案する．図 1 に導入から運用中までの作業の流れを示す．

(1) **初期導入**
システムの構築作業，初期設定作業など運用開始前の作業

(2) **マイナー更新**
セキュリティや不具合修正などの軽微なソフトウェアの更新作業

(3) **メジャー更新**
バージョンアップや機能向上などを含むソフトウェアの更新作業

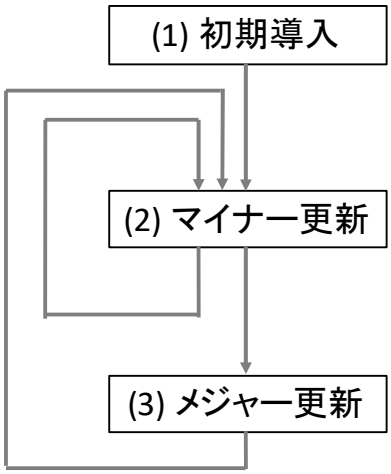


図 1　導入から運用までの作業

(2) マイナー更新はセキュリティや不具合対応など，システム運用に必須な作業である．特に, セキュリティパッチは緊急性があり，必要に応じて都度実施する必要がある．他方，(3) メジャー更新は機能向上の必要性がなければ不要である．一般にマイナー更新に比べ手間がかかる．しかし, リリース後時間が経過することで, サポート対象外になり, (2) マイナー更新の情報が提供されなくなるため，サポート切れを防ぐ目的でメジャー更新が必要になる．

3.2 ソフトウェアサポートのモデル

サポートの観点から，本稿では導入する OSS を以下の 2 つの部分に分けて議論する．

(A) **基盤ソフトウェア（基盤）**
OS ディストリビューション (CentOS や Ubuntu 等) に含まれるソフトウェアを示す．この部分は，OS ディストリビュータによってサポートされる．
パッケージシステムを使うことで容易に更新作業が可能である．

(B) **アプリケーション** (AP)
(A) のみで構成できない場合には，(A) 上に AP を導入する必要がある．AP の配布方法，導入方法および更新ポリシーは提供元により異なる．

一般に導入の前提条件として，基盤のバージョンなどが指定されるため，(A) との依存関係が生じる．例えば，データベースを利用する AP の場合，利用可能な DBMS のバージョンが指定される．OS ディストリビューションの特定のバージョンに含まれる DBMS のバージョンは決まっ

ているため，利用可能な OS ディストリビューションのバージョンに制限が発生する．

3.3 コストモデル

運用作業モデルおよびソフトウェアサポートのモデルを用いて, 導入および運用に関わるコストモデルを構築する．

運用コストは作業にかかわるコストを合計することで計算可能である．総運用コストCは，基盤コストC_i，AP コストC_aから計算できる．

$$C = C_i + C_a \tag{1}$$

$$C_i = C_{i,init} + \sum_{n=1}^{N_{i,major}} C_{i,major}(n) + \sum_{n=1}^{N_{i,minor}} C_{i,minor}(n) \tag{2}$$

$$C_a = C_{a,init} + \sum_{n=1}^{N_{a,major}} C_{a,major}(n) + \sum_{n=1}^{N_{a,minor}} C_{a,minor}(n) \tag{3}$$

ここで，$C_{i,major}(n)$はn回目の基盤のメジャー更新コストを，$N_{i,major}$は基盤の総メジャー更新回数を示す．その他の係数については表 1 に示す．

表 1　コストモデルの係数

		(1) 初期導入	(2) メジャー更新	(3) マイナー更新
(a) 基盤	コスト	$C_{i,init}$	$C_{i,major}(n)$	$C_{i,minor}(n)$
	回数	1	$N_{i,major}$	$N_{i,minor}$
(b) AP	コスト	$C_{a,init}$	$C_{a,major}(n)$	$C_{a,minor}(n)$
	回数	1	$N_{a,major}$	$N_{a,minor}$

ここで，単純化のためメジャーおよびマイナー更新コストは各回全て同一であると仮定すると次式が成り立つ．

$$\begin{aligned} C_{i,major} &\equiv C_{i,major}(1) = \cdots = C_{i,major}(N_{i,major}) \\ C_{i,minor} &\equiv C_{i,minor}(1) = \cdots = C_{i,minor}(N_{i,minor}) \\ C_{a,major} &\equiv C_{a,major}(1) = \cdots = C_{a,major}(N_{a,major}) \\ C_{a,minor} &\equiv C_{a,minor}(1) = \cdots = C_{a,minor}(N_{a,minor}) \end{aligned} \tag{4}$$

式 (4) および式 (2), (3) を利用すると，C_iおよびC_aは次式で表せる．

$$C_i = C_{i,init} + C_{i,major} \cdot N_{i,major} + C_{i,minor} \cdot N_{i,minor} \tag{5}$$

$$C_a = C_{a,init} + C_{a,major} \cdot N_{a,major} + C_{a,minor} \cdot N_{a,minor} \tag{6}$$

即ち，初期導入コストと，各更新作業のコストにそれらの実施回数を乗じたものとの和となる．

具体的なコスト指標として，作業手順の数や作業時間が考えられるが，作業環境や作業者のスキルなどに依存するという課題がある．このため，比較のためには作業環境やスキルなどを統一して考える必要がある．

4. ケーススタディ

本節ではケーススタディとして，Wordpress および Moodle のアップデート状況を述べる．

4.1 Wordpress

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．

表 2 に 2005 年以降の Wordpress の更新履歴とサポート期間を示す．表では，メジャーリリースごとにリリース日から最終更新のあった日までをサポート期間として計算した．また，更新回数はマイナーな更新回数を示している．サポートが継続されている版については，執筆時点である 2016 年 11 月 11 日までの情報を掲載している．

表 2　Wordpress の更新履歴とサポート期間

版	リリース日	最終更新日	更新回数	サポート期間（日）	備考
1.5	2005/2/17	2005/8/14	2	178	
2.0	2005/12/26	2006/7/29	4	215	
2.0.5	2006/10/27	2007/8/5	6	282	
2.1	2007/1/22	2007/4/3	3	71	
2.2	2007/5/16	2007/9/8	3	115	
2.3	2007/9/24	2008/2/5	3	134	
2.5	2008/3/29	2008/4/25	1	27	
2.6	2008/7/15	2008/11/25	5	133	
2.7	2008/12/10	2009/2/10	1	62	
2.8	2009/6/10	2009/11/12	6	155	
2.9	2009/12/18	2010/2/15	4	59	
3.0	2010/6/17	2011/4/26	6	313	
3.1	2011/2/23	2011/6/29	4	126	
3.2	2011/7/4	2011/7/12	1	8	
3.3	2011/12/12	2012/6/27	3	198	
3.4	2012/6/13	2012/9/6	2	85	
3.5	2012/12/11	2013/6/21	2	192	
3.6	2013/8/1	2013/9/11	1	41	
3.7	2013/10/24	2016/9/7	16	1049	現行
3.8	2013/12/12	2016/9/7	16	1000	現行
3.9	2014/4/16	2016/9/7	14	875	現行
4.0	2014/9/4	2016/9/7	13	734	現行
4.1	2014/12/17	2016/9/7	13	630	現行
4.2	2015/4/23	2016/9/7	10	503	現行
4.3	2015/8/18	2016/9/7	6	386	現行
4.4	2015/12/8	2016/9/7	5	274	現行
4.5	2016/4/12	2016/9/7	4	148	現行
4.6	2016/8/16	2016/9/10	1	25	現行

分析を行った 2005 年以降, 開発が活発に行われており，頻繁に新版がリリースされている．リリースの履歴 [12, 13] を調べると 3.6 版まではサポート期間が 1 年以下であ

り，運用中は継続的にメジャーバージョンアップをおこなう必要があったことがわかる．2013 年にリリースされた 3.7 版以降は自動アップデート機能が実装され, 版を上げることなくセキュリティパッチを適用し継続利用が可能になった．

ミドルウェア（PHP およびデータベース）に依存関係があり，版ごとに要件が決められている．表 3 に依存関係を示す．

表 3　Wordpress のミドルウェア要件

Wordpress	PHP	MySQL	備考
1.5-2.2	不明	3.23	CentOS 4
2.3-2.8	4.2	4.0	CentOS 5
2.9-3.1	4.3	4.1.2	CentOS 5
3.2-4.6	5.2.4	5.0	CentOS 6

2.3 版,2.9 版,および 3.2 版でミドルウェア要件が変更になっている．これに伴い，CentOS 提供のミドルウェアを利用するケースでは，2.3 版，3.2 版で OS 更新が必要となる．

例えば，2007 年 9 月に CentOS5 上にリリース直後の 2.3 版を導入した場合，3.1 版まで Wordpress 本体のみのアップデートで継続利用可能であったことが分かる. 3.1 版の最終更新は 2011 年 6 月であるので, 33 ヶ月の継続利用となる．

4.2 Moodle

Moodle は最も利用者の多い学習管理システムの一つである．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．

2.0 版，2.1 版，2.5 版，および 2.7 版でミドルウェア要件が変更になっている．これに伴い，CentOS 提供のミドルウェアを利用するケースでは，2.0 版，2.7 版で OS の入れ替えが必要となる．

例えば，2011 年 7 月に CentOS6 上にリリース直後の 2.1 版を導入した場合，2.6 版まで Moodle のみのアップデート (OS 更新なし)で継続利用可能であったことが分かる．2.6 版の最終更新は 2015 年 5 月であるので，46 ヶ月の継続利用となる．

1.9 版からはスクリプトが用意されて更新が容易に実施できるようになっている．但し，Moodle は移行対象の版が決まっており，数世代前の版からの更新しかサポートされない点に注意が必要である．

2005 年以降の Moodle の更新履歴とサポート期間[10,11]を表 4 に示す．表記方法は Wordpress の場合と同じである．

Moodle は分析を行った 2005 年以降，定期的に新版のリリースが行なわれている. Wordpress に比べてサポート期間が長いが，これは新版リリース後もそれ以前の版のサポートが継続されているためである．

表 4　Moodle の更新履歴とサポート期間

版	リリース日	最終更新日	更新回数	サポート期間（日）	備考
1.5	2005/6/5	2006/5/21	4	350	
1.6	2006/6/19	2009/1/28	9	954	
1.7	2006/11/7	2009/1/28	7	813	
1.8	2007/3/30	2010/12/3	14	1344	
1.9	2008/3/3	2012/7/6	19	1586	
2.0	2010/11/24	2012/3/14	9	476	
2.1	2011/7/1	2013/1/14	10	563	
2.2	2011/12/5	2013/7/8	11	581	
2.3	2012/6/25	2014/1/13	11	567	
2.4	2012/12/3	2014/7/14	11	588	
2.5	2013/5/14	2014/11/10	9	545	
2.6	2013/11/18	2015/5/11	11	539	
2.7 LTS	2014/5/12	2016/9/12	16	854	現行
2.8	2014/11/10	2016/5/9	12	546	
2.9	2015/5/11	2016/9/12	8	490	現行
3.0	2015/11/16	2016/9/12	6	301	現行
3.1 LTS	2016/5/23	2016/9/12	2	112	現行

Moodle プロジェクトはサポートポリシーを明示しており，LTS (Long Term Support) 版はリリース後 3 年，それ以外の版は 12 ヶ月（不具合修正）または 18 ヶ月（セキュリティ修正）間サポートが継続される．

表 5 に Moodle のミドルウェア要件を示す．ここでは，データベースとして MySQL および PostgreSQL のみを示している．

表 5　Moodle のミドルウェア要件

Moodle	PHP	MySQL	PostgreSQL	備考
1.5-1.9	4.3	4.1	-	CentOS 4
2.0	5.2.8	5.0.25	8.3	CentOS 6
2.1-2.4	5.3.2	5.0.25	8.3	CentOS 6
2.5-2.6	5.3.3	5.1.33	8.3	CentOS 6
2.7-3.1	5.4.4	5.5.32	9.1	CentOS 7

5. ライフサイクル最適化戦略

机上の運用シミュレーションによってアップグレードポリシーによる運用コストの比較を行った．

【前提条件】

- 基盤として，CentOS の公式リリース版に含まれるソフトウェアを利用して AP を動作させる．
- 運用中のシステムのセキュリティを担保するため，基盤および AP の更新は必ず実施する．

【計算方法】

- 2005 年から執筆時まで運用した場合の導入および更

新回数を机上シミュレーションによって算出する．
(1) 基盤および AP の初期導入
(2) 基盤および AP のメジャー更新
(3) 基盤および AP のマイナー更新

- 運用作業経験者へのインタビュー結果から，本稿では，運用コスト(作業時間)として表 6 に示す値を採用した．ただし，基盤のマイナー更新 (3a) は，パッケージ管理システムによって自動的に行うことができるため，本稿では更新コストを 0 として計算から除外した．

表 6　コストの試算に採用した係数

コスト（時間）	(1) 初期導入	(2) メジャー更新	(3) マイナー更新
(a) 基盤	16	16	0
(b) AP	16	8	2

【比較した更新戦略】

(S1) **即時更新戦略**

AP リリース後，直ちに最新版を導入する．この更新戦略では，最新の機能を利用することが可能である．

(S2) **遅延更新戦略**

更新コストを低減するため，サポートされる限り AP を継続利用する．

(S2-1) 最新版に移行する．

(S2-2) サポート中の次版に移行する．

5.1 Wordpress

表 7 に Wordpress の運用シミュレーション結果を示す．戦略ごとの更新時の版の移行の詳細については付録 A に示した．

表 7　Wordpress の運用シミュレーション結果

戦略	(1) 初期導入	(2)メジャー更新	(3)マイナー更新	合計
(S1) 即時	2 (32)	29 (248)	67 (134)	98(414)
(S2-1) 遅延	2(32)	20(178)	64(128)	86(336)
(S2-2) 遅延	2(32)	20(178)	64(128)	86(336)

※数値は，作業回数（コスト）を示す

(S1) と (S2) の戦略を比較すると，アップデートを遅延することで，78 時間（約 19%）のコスト削減が可能であることが分かる．

3.6 版まで，最新版のみがサポート対象であるため，Wordpress のケースでは更新戦略による違いがない．3.7 版以降の更新を継続するか，3.7 版を継続利用するかの違いだけである．また戦略 (S2-1) と (S2-2) は全く同じ結果である．

図 2 に運用コストの経年変化を示す．

平均して年間 35 時間程度の作業が必要である．特に，基盤のメジャー更新の必要な 2007 および 2011 年に，運用コストが増大していることが分かる．

また，2012 年までは必ず新版へのアップデートが必要であったため，戦略によらず運用コストが同じになっている．2013 年以降 AP のメジャーアップデートが必須ではなくなったため，戦略 (S2) の運用コストが年間 20 時間以下に減少していることが分かる．

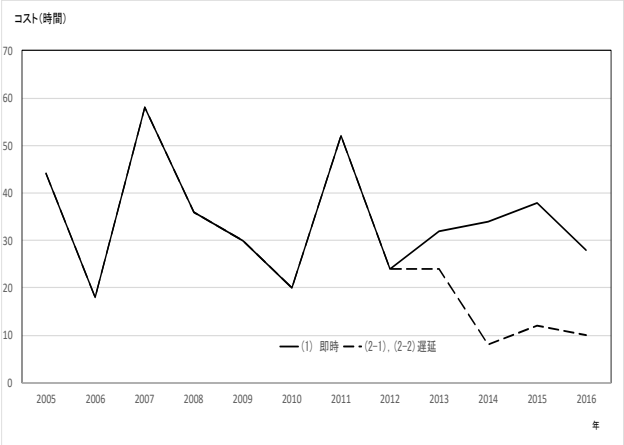


図 2　Wordpress の運用コストの変化

5.2 Moodle

表 8 に Moodle の運用シミュレーション結果を示す．戦略ごとの更新時の版の移行の詳細については付録 B に示した．

表 8　Moodle の運用シミュレーション結果

戦略	(1) 初期導入	(2) メジャー更新	(3) マイナー更新	合計
(S1) 即時	2 (32)	19 (168)	59 (118)	80 (318)
(S2-1) 遅延	2 (32)	10 (96)	59 (118)	71 (246)
(S2-2) 遅延	2 (32)	13 (120)	51 (102)	65 (254)

※数値は，作業回数（コスト）を示す．

(S1) と (S2) の戦略を比較すると，アップデートを遅延することで，72 時間（約 22%）のコスト削減が可能であることが分かる．

戦略によって AP の更新作業回数に違いが出た．戦略 (2-1) 遅延が最も作業回数が少なくコストが低い．また，戦略によらず AP のマイナー更新回数に大きな違いはない．運用に当たっては不具合修正やセキュリティパッチは必須の作業であることが分かる．

図 3 に運用コストの経年変化を示す．

Wordpress に比べて更新が少ないため全体の運用コストは低いが，基盤のメジャー更新時にコストが増大していることが分かる．

遅延戦略 (S2-1) および (S2-2) は基盤のメジャー更新時期が異なるため，年ごとのコストに大きなばらつきが生じているが，ライフタイムでのコストはほとんど同じである．

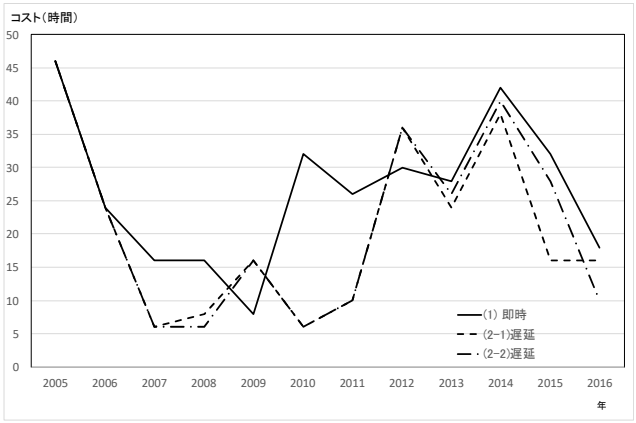


図 3　Moodle の運用コストの時系列変化

6. 考察

5 節で示した運用シミュレーション結果，得られた知見および考察を以下に述べる．

6.1 更新戦略に関する考察

(1) サポートされている期間は同じ版の利用を続けるとコストが下がる．

更新作業を減らすために,なるべく更新を避けて同じ版を利用することがコスト削減につながる．

(2) 更新時に，新しい版に一気に更新することで，更新回数を減らせる．

新機能が必要でない場合には,更新時にまとめて最新版に更新した方が，変更コストが少ない．但し，Moodle の様に移行制限がある AP については注意が必要である．

6.2 AP の特性に関する考察

AP を比較することで，以下の知見が得られた．

(1) サポート期間の長い AP の方が,更新回数が少なくコストが低いことが期待される．

初期の Wordpress より Moodle のほうがサポート期間が長く，更新回数も少なくてすむ．同一機能の AP であれば，サポート期間の長いものを選択することでコスト低減が可能となる．近年，LTS (Long Term Support) 版を出すプロジェクトがあるが，本シミュレーションからも更新コスト削減に寄与することが期待される．

(2) AP のマイナー更新回数は，AP の更新に依存し，戦略に依らず必要になる．

品質の良い AP を利用するとマイナー更新が少なくなることが期待される．分野によっては更新頻度が異なる可能性がある．例えば,セキュリティの脅威に曝される WEB サーバは脆弱性が発見された場合に更新が必須である．文献 [2] によると，リリースごとに OSS の品質に顕著な向上はないものの，CVE に上げられる脆弱性は減少する傾向にあるとの分析が述べられている．

6.3 コストモデルの評価

3 節で提案したコストモデルに従って，シミュレーションを行った．以下の点に関しては，さらに検討が必要であると考える．

(1) コストの係数

簡単化のために更新コストを同一としたが，ソフトウェアの仕組みが異なり，AP ごとに更新にかかる手間が異なる可能性がある．またインタビュー結果から表 6 の値を決めているが，作業者によって異なる可能性がある．このため，複数の条件下で実際の更新作業にかかる時間の測定を行うことが望ましい．また，商用ソフトのコストと比較することも必要であると考える．

(2) 他の運用コストの考慮

本論文では，ソフトウェアの導入および更新作業のみをコストとしてカウントした．しかし，実際の運用においては，以下の作業が発生し，運用に大きな影響を与えると考える．

- ソフトウェアの学習
- ユーザ対応
- バックアップ・リストア作業

ただし，これらのコストは商用ソフトの場合でも必要であるため，OSS 特有のコストではない．より一般的な運用モデルの評価の中で検討する必要があると考える．

7. まとめと今後の課題

本稿では，OSS 製品の利用実態にあわせた運用コストモデルを提案した．また，OSS のサポートポリシーに基づき，運用コストを最適化するための方法について論じた．具体的なソフトウェアパッケージを取り上げ，12 年間の運用を行なった場合のシミュレーションを通じて，バージョンアップの戦略を論じた．

提案した運用コストモデルには限界があるものの，運用コストの概要を知るために有用であることが分かった．OSS コミュニティによるサポート期間が長いパッケージを利用すると，セキュリティのリスクを下げつつ，安定的な運用を行えることが分かった．

筆者らは計算機環境をアーカイブに保存することで，研究環境の保存と再現を行う仕組みを提案中である．保存した計算機環境を再現する場合にも本稿で述べたコストモデルの適用が可能であると考える．例えば，再現後の計算機環境のパッケージ選択やバージョンアップ計画などにも適用可能である．

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．

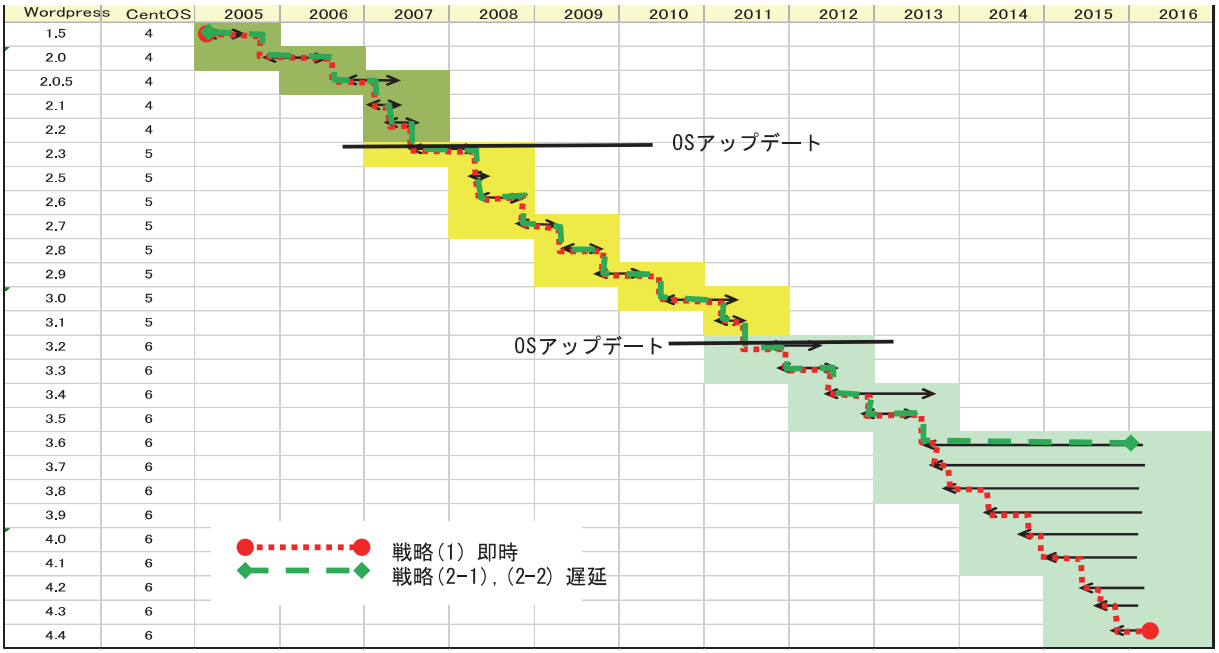
参考文献

[1] Biazzini, M. and Baudry, B., “May the Fork be with You: Novel Metrics to Analyze Collaboration on GitHub,” *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*,

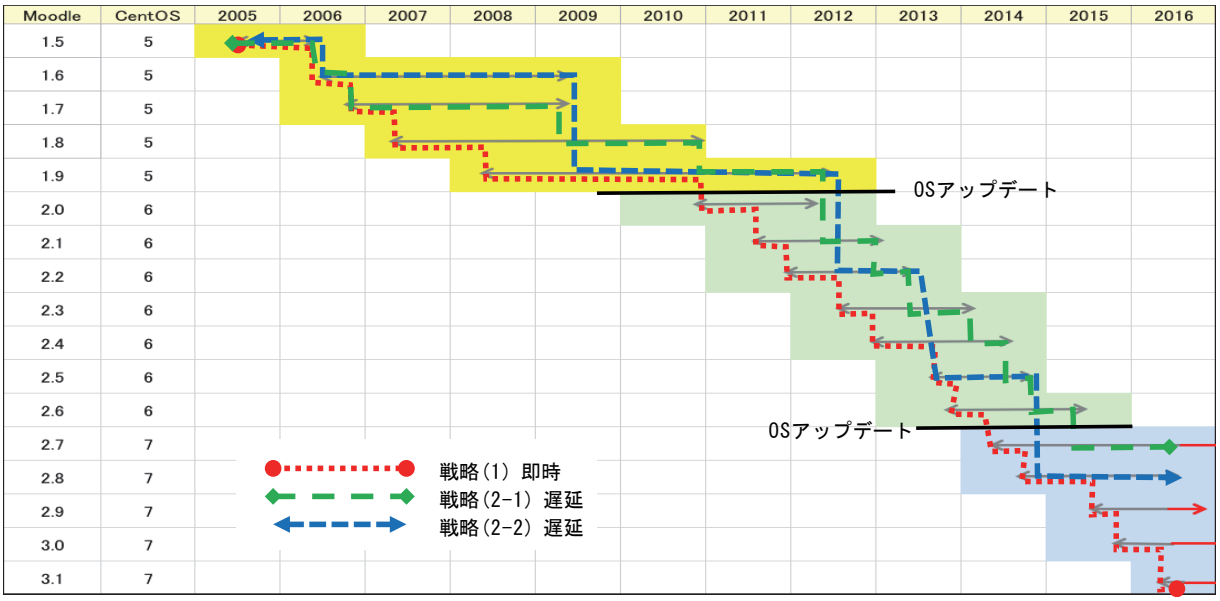
ACM, Hyderabad, India, pp. 37-43, (2014).
[2] Edwards. N. and Chen, L., “An Historical Examination of Open Source Releases and Their Vulnerabilities,” *Proceedings of 2012 ACM Conference on Computer and Communications Security* (*CCS '12*). ACM, New York, NY, USA, pp.183-194 (2012).
[3] Ilya, G., GitHub Archive, https://githubarchive.org/ （2016/11/11 閲覧）.
[4]Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M. and Damian, D., “The Promises and Perils of Mining GitHub,” *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, Hyderabad, India, pp. 92-101 (2014).
[5] 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, (2016).
[6] Technowalker Inc, Moodle システム要件表, http://www.twalker.co.jp/moodle/youken.html（2016/11/11 閲覧）
[7] Tsay, J., Dabbish, L. and Herbsleb, J., “Let's Talk about It: Evaluating Contributions through Discussion in GitHub,” *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM, Hong Kong, China, pp. 144-154 (2014).
[8] Weiss, M., “The Business of Open Source: Missing Patterns,” *Proceedings of the 20th European Conference on Pattern Languages of Programs* (*EuroPLoP '15*). ACM, New York, NY, USA, Article 13, 11pages, (2015).
[9] GitHub Inc. GitHub, https://github.com/ （2016/11/11 閲覧）
[10] Moodle プロジェクト, https://moodle.org/ （2016/11/11 閲覧）
[11] Moodle プロジェクト, リリース一覧 https://docs.moodle.org/dev/Releases （2016/11/11 閲覧）
[12] 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 （2016/11/11 閲覧）
[13] WordPress.org, Releases Category Archive, https://wordpress.org/news/category/releases/ （2016/11/11 閲覧）

付録

A.　Wordpress の運用シミュレーション結果



B.　Moodle の運用シミュレーション結果



日本ソーシャルデータサイエンス論文誌
第 1 巻　第 1 号（2017 年 3 月）

A Study on an Operation Model of OSS Products and Optimization

Yoshitaka KUWATA[†1] Toru ISHIZAKA[†1] Shigetoshi YOKOYAMA[†2†3] Kento AIDA[†3]

Abstract: Open Source Software (OSS) 's are software products developed by community. The source code and development process of OSS are open to public. Because the license fee is not required, many commercial IT service make use of OSS. Even one decide to adopt OSS, though, the operation costs of OSS are required. It is harder to predict operation cost of OSS than those of proprietary software. In this paper, we propose a cost model based on practical use of OSS. We applied the model to two of the most popular OSSs, Wordpress and Moodle, and verified the feasibility of the model. We also discuss operation strategies of OSS to reduce operation cost.

Keywords: Open Source Software(OSS), Running Cost of OSS, Community analysis

†1 Muroran Institute of Technology (Corresponding Author: kuwata@mmm.muroran-it.ac.jp)
†2 Gunma University
†3 National Institute for Informatics
Submitted: 30/11/2016
Accepted: 10/02/2017