ユーザ視点に基づいたソフトウェアアップデート計画に関する一考察

岡村 寛之
広島大学大学院工学研究院
okamu @ rel.hiroshima-u.ac.jp

中原 良太
広島大学工学部第二類
（電気・電子・システム・情報系）

土肥 正
広島大学大学院工学研究院
dohi @ rel.hiroshima-u.ac.jp

要旨

本研究では，ユーザ視点からのソフトウェアの運用保守において「いつソフトウェアをアップデートするか」と言う問題を考える．特に，ソフトウェアアップデートのアクティビティを残存バグ数から数理モデルで表現し，ある種の決定問題として定式化および最適解の導出手法を議論する．さらに，実データを用いた例を通じて，最適なソフトウェアアップデート計画の特徴について議論する．

1 はじめに

情報システムの社会における重要度は年々増加しており，システムの停止により多大なコストが発生することがある．近年では OpenStack などの仮想化基盤ソフトウェアを用いたクラウドシステムが企業を中心に盛んに用いられているため，そのような基盤ソフトウェアの信頼性や可用性の確保が重要な課題となっている．

一般にソフトウェアの信頼性は，テスト工程におけるバグの発見および除去により改善されることが知られている．仮に，ソフトウェアに内在するバグや仕様上の欠陥が完全に取り除けた場合，そのソフトウェアの障害発生確率は 0 となる．しかしながら，現実問題としてバグや仕様上の欠陥を完全に取り除くことは不可能である．また，ソフトウェア開発におけるテスト費用や納期の制約により，不完全な状態でソフトウェアをリリースすることも現実には起こりえる．そのため，ベンダー（開発者）視点では，ソフトウェアをリリースした後，バグフィックスやセキュリティアップデートを提供するなどの「保守作業」が実際的に製品としてのソフトウェアの信頼性に大きく寄与している．

一方で，ユーザ視点でソフトウェアシステムの運用に注目した場合，システムの停止をできる限り少なくし，システムの可用性を高めることが望ましい．一般的に，システム停止の要因として，ソフトウェアに内在するバグが引き起こすシステム障害によるものが注目されるが，現実的に保守によるシステムも可用性を低下する要因として認識する必要がある．先に述べたように，ソフトウェアの保守作業はリリース後に提供されるバグフィクスやセキュリティアップデートなどを適用するソフトウェアのアップデート作業であり，それらを行うことでソフトウェアに内在するバグや脆弱性を取り除き，システムの障害確率を低下させることができる．しかしながら，ソフトウェアのアップデートでは再起動などの一時的なシステム停止を要する場合が多い．アプリケーションソフトウェアなどの上位層では，そのような保守による停止は大きな問題とならないが，仮想化基盤ソフトウェアなどでは，その上で稼働するすべてのアプリケーションを一旦停止する必要があるため，その影響を無視することはできない．

本論文では，ユーザ視点からシステム運用時において可用性の観点から最適な保守計画を考えるための数理モデルについて考察する．ソフトウェアの保守では二つの視点から二つの問題が存在することに注意すべきである．一つは，ベンダー視点で，ソフトウェアのバグフィックスはセキュリティアップデートを「いつリリースするか」

と言う問題である．これはパッチマネジメントの問題として，いくつかの文献において議論されている．文献 [1] では，ベンダー視点で最適なパッチマネジメントを行うための数理モデルの考察を行っている．そこでは，リスクとパッチリリース費用のトレードオフを考慮して，最適なパッチ配布スケジュール問題を取り扱っている．もう一つは，ユーザ視点から，「いつアップデートを行うか」と言う問題である．Luo ら [2] はオープンソースなど開発リポジトリからユーザがいつでも最新版を入手できる状況における最適なソフトウェアアップデート計画のためのモデルを考察している．一方で，上記の問題はベンダーが支配的な社会的なゲームととらえることも可能であり，実際，Cavusoglu ら [3] はベンダーとユーザによるゲームとして，単純な仮定の下でパッチマネジメントの議論を行っている．

本論では，Luo ら [2] と同様にユーザ視点からいつアップデートを行うかと言う問題を考える．Luo ら [2] は任意時刻において，いつでも最新のものが利用できるという状況で議論を行っているが，一般的に，プロプライエタリなソフトウェアにおいては，ベンダーから提供された最新版を利用できるだけであり，誰もが開発版を利用できるわけではない．換言すると，最新版はある固定されたタイミングで（離散的に）利用可能である．この状況を考慮すると，Luo ら [2] とは異なったソフトウェアアップデート計画のための数理モデルが考えられる．

本論文は以下の構成となる．2 節では，ソフトウェア運用と保守を単純化した数理モデルについて，モデルの仮定とふるまいを述べる．特に，ソフトウェアに内在する残存バグ数によって，ソフトウェア保守作業がモデル化されることについて言及する．3 節では，モデルにもとづいて，費用の観点から最適なソフトウェアアップデート計画を導出するための手順について説明する．4 節では，実際のソフトウェアデータを用いた数値例を通じて最適アップデート計画の特徴を示す．

2　モデルの記述

ユーザは時刻 $T_0 = 0$ から，あるソフトウェアの利用を開始し，時刻 $T_L(> T_0)$ までの利用を行うものとする．ベンダーはその期間 $[T_0, T_L]$ において N 回のソフトウェアリリースを行う．ソフトウェアリリースはバグフィクスのみであり，N 回のリリースの時刻を $T_1 < \cdots < T_N$ として表現する．ただし，$T_0 \geq T_1$ および $T_N \geq T_L$ である．ユーザは期間 $[T_0, T_L]$ における任意時刻でソフトウェアのアップデートを何回でも行うことができる．アップデートによってソフトウェアをリリースされている最新の版にすることができる．簡単のため，ここではデグレードはないものとする．

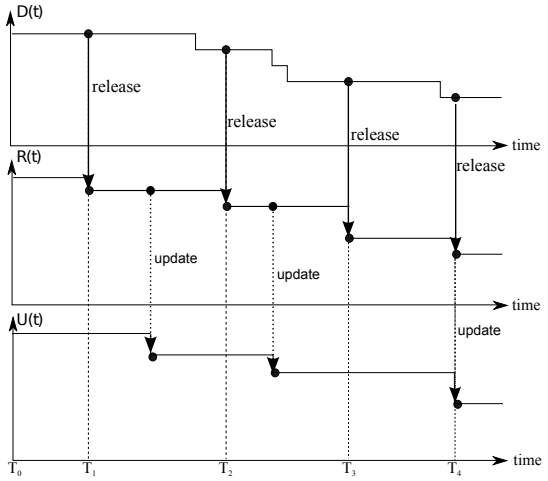

図 1. リリースおよびアップデートによる残存フォールト数のふるまい．

ソフトウェアのリリースやアップデートを残存バグ数によってモデル化する．いま，次の三つの異なる環境における残存バグ数を定義する．

- $D(t)$: 開発者が管理しているソフトウェア（開発版）に内在する残存バグ数を表す確率過程
- $R(t)$: リリースされたソフトウェア（リリース版）の残存バグ数を表す確率過程
- $U(t)$: ユーザが利用しているソフトウェア（ユーザ版）の残存バグ数を表す確率過程

開発版の残存バグ数は，バグ報告などにより継続的に減少していく．一方，リリース版は開発者が最新版をリリースするタイミングのみで残存バグ数が減少し，$D(T_i) = R(T_i),\ i = 1, \ldots, N$ の関係が成り立つ．また，ユーザ版も同様に，アップデートのみで残存バグ数が減少しする．例えば，時刻 t でユーザがアップデートを行ったとすると $R(t) = U(t)$ の関係が成り立つ (図 1 を参照)．

システムの障害はユーザ版における残存バグ数に比例した発生率 $\mu U(t)$ で発生するものと仮定する．ここで，

μ は単位時間単位バグあたりの障害発生率を表す．一方で，障害からの復旧について次の二つの方法を考える．

- 方法 1：障害が発生した場合，システム再起動で利用環境を変化させることにより障害を回避することを試みる．
- 方法 2：より最新のソフトウェアがリリースされている場合は緊急アップデートを行う．リリースされていない場合は方法 1 と同じ．

システム障害はソフトウェアに内在するバグによって引き起こされるが，再現性のある障害と再現性のない一時的な障害に分類できる．方法 1 は，メモリリークなど特定が難しいバグによる再現性のない一時的な障害に対応した回避方法である．実際,「再起動したら動くようになった」と言うのは運用においてもよく観測される事象である．一方，方法 2 は再現性のある障害の一部に効果のある手法であり，観測された再現性のある障害が最新版で修正されていれば本質的な回避が行える．ただし，モデルの簡単化のため，いずれの方法も，どのようにして障害を回避するかと言う詳細については議論しない．つまり，もし行いたい作業が再現性のある障害で，何度やっても行えない場合はその作業を諦めるような回避も含んでいることに注意する．

3 最適ソフトウェアアップデート計画

最適なソフトウェアアップデート計画を考えるため，次の費用パラメータを考える．

- c_u: 一回のアップデートに要する費用
- c_e: 一回の緊急アップデートに要する費用
- c_f: 一回のシステム再起動による障害回避に要する費用

先に述べたように，ソフトウェアアップデートではシステム停止を伴う場合があるため，c_u はシステム停止による機会損失とアップデートそのものの労力による費用を含む．また，c_f には再起動によるシステム停止による機会損失に加えて，作業を諦めることによる回避のペナルティ費用も含む．上記の費用を，アップデートによるシステムの停止時間，緊急アップデートによるシステムの停止時間，システム再起動によるシステム停止時間として，それらの停止時間を最小化する，本質的に可用性を最大化する問題も取り扱うことができる．

上記の費用構造のもとでは，数学的に総費用を最小にする観点から，ユーザは必ず開発者が最新版をリリースしたと同時にアップデートを実行するかしないかを決定し，それ以外の時刻ではアップデート（緊急アップデートを除く）を行わないことがわかる．つまり，アップデート計画問題は，最新版のリリース時刻列 $T_1, T_2, \ldots, T_N$ でアップデートを行う／行わないと言う行動を決定するマルコフ決定過程により記述できる．

費用を最小にする最適なアップデート計画を求めるために，開発者が x 番目のリリースを行った時刻 T_x において，ユーザが $y(< x)$ 番目にリリースされたソフトウェアを利用している状況を考える．このとき，次の二つの費用を定義する．

- $W_P(x|y)$: x 番目にリリースされたソフトウェアへのアップデートを行い，それ以降は最適なアップデート計画に従ったときの時刻 T_L までの総期待費用．
- $W_{\overline{P}}(x|y)$: x 番目にリリースされたソフトウェアへのアップデートを行わずに，それ以降は最適なアップデート計画に従ったときの時刻 T_L までの総期待費用．

障害回避を方法 1 によって行う場合，最適性原理から $y = 0, \ldots, N$, $x = y+1, \ldots, N$ に対して以下の最適性方程式を得る．

$$V(x|y) = \min\{W_P(x|y), W_{\overline{P}}(x|y)\}, \tag{1}$$

$$W_P(x|y) = c_u + c_f\mu\Delta T_{x+1}\xi_x + V(x+1|x), \tag{2}$$

$$W_{\overline{P}}(x|y) = c_f\mu\Delta T_{x+1}\xi_y + V(x+1|y). \tag{3}$$

ここで $\Delta T_{x+1} = T_{x+1} - T_x$ および $\xi_x = \mathrm{E}[R(T_x)]$ である．さらに，$T_{N+1} = T_L$ および $V(N+1|\cdot) = 0$ であることに注意する．

一方，障害回避を方法 2 によって行う場合，最適性原理から $y = 0, \ldots, N$, $x = y+1, \ldots, N$ に対して以下の最適性方程式を得る．

$$V(x|y) = \min\{W_P(x|y), W_{\overline{P}}(x|y)\}, \tag{4}$$

$$W_P(x|y) = c_u + c_f\mu\Delta T_{x+1}\xi_x + V(x+1|x), \tag{5}$$

$$\begin{aligned} W_{\overline{P}}(x|y) = {} & \int_0^{\Delta T_{x+1}} \left(c_e + c_f\mu\left(\Delta T_{x+1} - t\right)\xi_x\right) dF_y(t) \\ & + V(x+1|x)F_y(\Delta T_{x+1}) \\ & + V(x+1|y)(1 - F_y(\Delta T_{x+1})). \end{aligned} \tag{6}$$

表 1. IE9 のバグ発見数データ.

デバッグ期間（月）	その期間でのバグ発見数
2	11
2	7
2	8
2	3
2	4
2	5
2	13
1	2
1	4
1	5
1	3
1	3
1	13
1	9
1	2
1	11
1	19
1	18
1	11
1	10
1	9

ここで $F_y(t) = 1 - e^{-\xi_y t}$ である.

上記の二つの最適性方程式はいずれも $y = N, x = N$ から後ろ向きに $V(x|y)$ を算出することができ，得られた $V(x|y)$ をもとにして，リリース時刻列 $T_1, \ldots, T_N$ でアップデートを行うかどうかの最適な行動を導出できる.

4 数値例

ここでは，Microsoft 社の Web ブラウザである Internet Explorer 9 (IE9) の 28 ヶ月間（2011/6/14 から 2013/10/8 まで）のバグ修正レポート [1] から収集したデータを用いた例を示す.

表 1 は収集した月ごとのバグ発見数データを示している．これを用いて，開発版における残存バグ数を表す確率過程の推定を行う．いま，開発版における累積バグ発見個数を表す確率過程を $N(t)$ を次の非同次ポアソン過程（NHPP）と仮定する.

$$P(N(t) = n) = \frac{(\omega G(t))^n}{n!} e^{-\omega G(t)}. \tag{7}$$

ここで，$\mathrm{E}[N(t)] = \omega G(t)$ は平均値関数と呼ばれ，時刻 t で発見されるバグの期待値を表す．実際，式 (7) は NHPP に基づいたソフトウェア信頼度成長モデル（以下，NHPP モデル）を表しており，リリース判定などで利用されている [4]．また，ω はソフトウェアに内在する初期バグ数の期待値，$G(t)$ は個々のバグの発見時刻に対する累積確率分布関数を表している．いま，開発版の残存バグ数 $D(t)$ に対して $N(t) + D(t)$ がソフトウェアに内在する総バグ数（初期バグ数）であるため,

$$P(D(t) = n) = \frac{(\omega \overline{G}(t))^n}{n!} e^{-\omega \overline{G}(t)} \tag{8}$$

であることが示される．ここで $\overline{G}(t) = 1 - G(t)$ であり，$G(0) = 1$ から $G(\infty) = 0$ まで単調に減少する関数である．つまり，データから NHPP モデルを推定することで，開発版の残存バグ数の平均値を推定することができる.

NHPP モデルに対しては，伝統的に最尤法および情報量規準によるモデル選択が行われる．NHPP モデルでは，バグ発見時刻分布 $G(t)$ にどのような確率分布を適用するかで様々なモデルが表される．つまり，具体的な手順は，まず，候補となるいくつかの NHPP モデルに対して最尤推定によるパラメータ推定を行い，次に，以下の赤池情報量規準（AIC）[5] を算出し，最も AIC が小さくなるモデルを最良のモデルとして選ぶ.

$$\mathrm{AIC} = -2(\text{最大対数尤度}) + 2(\text{自由パラメータ数}). \tag{9}$$

上述のデータに対して，これらの推定作業を SRATS [2] で行った．SRATS では 11 種類の候補となるモデルに対する最尤法と情報量規準の算出を行う．表 2 は SRATS によって得られた各モデルの AIC を示している．この結果から AIC を最小にする切断最小値分布モデルを利用し，以下の平均残存バグ数を推定した.

$$\mathrm{E}[D(t)] = \omega H(-t)/H(0) \tag{10}$$

$$H(t) = \exp\left(-\exp\left(-\frac{t-\beta}{\alpha}\right)\right), \tag{11}$$

$$\omega = 1817.9, \quad \alpha = 14.99, \quad \beta = 60.28. \tag{12}$$

また，図 2 に推定した残存バグ数の平均値を示す.

[1] マイクロソフトセキュリティ情報
https://technet.microsoft.com/ja-jp/security/bulletin

表 2. 推定結果.

NHPP モデル	AIC
指数分布モデル	170.09
ガンマ分布モデル	155.31
パレート分布モデル	178.58
切断正規分布モデル	134.82
対数正規分布モデル	167.85
切断ロジスティック分布モデル	133.61
対数ロジスティック分布モデル	154.85
切断最大値分布モデル	135.93
対数最大値分布モデル	177.82
切断最小値分布モデル	133.42
対数最小値分布モデル	154.45

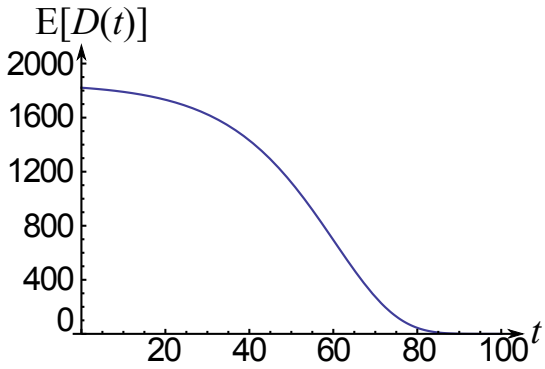

図 2. 推定された期待残存バグ数のふるまい.

残存バグ数の減り方が最適なアップデート計画に与える影響をみるため，ユーザが IE9 を利用する期間として次の三つの期間に対する最適なアップデート計画をそれぞれ算出する.

- 期間 1：0ヶ月から 24ヶ月目までの 24ヶ月間（$T_0 = 0, T_L = 24$）.
- 期間 2：32ヶ月から 56ヶ月目までの 24ヶ月間（$T_0 = 32, T_L = 56$）.
- 期間 3：64ヶ月から 88ヶ月目までの 24ヶ月間（$T_0 = 64, T_L = 88$）.

また，IE9 は，一ヶ月あるいは二ヶ月毎に定期的にアップデート用の修正プログラムがリリースされているため，上記のいずれの期間においても $\Delta T_x = 1, x = 1, \ldots, 23$ とした．また，単位時間単一バグ当たりの障害発生率を $\mu = 0.1$，アップデート一回当たりに要する費用を $c_u = 1.0$，障害を再起動による回避に要する費用を $c_f = 1.0$ とした．また，緊急アップデートに要する費用は $c_e = 1.5, 2.0, 2.5$ の三つの場合を考える.

表 3 は，方法 1 と方法 2 それぞれの障害回避方法のもとで，最適なアップデート計画を行った時の最小総期待費用を表している．表中の括弧内の値は対応する期間で行う計画されたアップデート回数を表している．各期間における総期待費用を比較すると，残存バグ数が少なく且つその期間で発見されるバグ数が少ない期間（期間 3）が最も少ない費用であることがわかる．また，方法 1 でアップデートを行う回数に着目すると，期間 2 が最も多く，その期間で発見されるバグ数とアップデート回数に強い関連があることがわかる．一方で，方法 2 に着目すると，緊急アップデートに要する費用 c_e が一回のアップデートと再起動による障害回避に要する費用の和 $c_u + c_f$ より小さい場合，計画されたアップデート回数が極端に少ないことが読み取れる．これは，アップデートのほとんどを緊急アップデートで対応することを意味している．方法 2 では，緊急アップデートを行う機会があるため，方法 1 ほど発見バグ数とアップデート回数の間に強い相関が見られなかった．また，方法 1 と方法 2 の総期待費用を比較すると，c_e が $c_u + c_f$ より小さい場合にだけ，方法 2 の総期待費用が方法 1 の総期待費用より低くなっている．つまり，緊急アップデートは，通常のアップデートと障害回避に要する費用が緊急アップデートに要する費用より小さい場合だけ効果的であり，その他の場合は緊急的にアップデートする必要性がないことも読み取れる.

表 3. 最小総期待費用.

		期間 1	期間 2	期間 3
方法 1		4273(10)	3155(23)	435(19)
方法 2	$c_e = 1.5$	4267(0)	3144(0)	427(4)
	$c_e = 2.0$	4279(23)	3155(23)	434(19)
	$c_e = 2.5$	4279(23)	3155(23)	435(21)

5 まとめと今後の課題

本研究では，ユーザ視点からソフトウェアアップデートに関する数理モデルを構築し，最適なアップデート計

[2] http://www.srat-app.com/SRATS/

画を導出するための問題設計および最適解の導出を行った．また，数値例では実データに基づいた分析を行い，最適アップデート計画に対する特徴について議論した．

本研究では，いくつかの重要な点をモデル化のために簡単化している．一つ目は，デグレードの問題であり，実際のアップデート作業では，アップデートすることにより引き起こされる障害が存在する．ユーザ視点で，すぐにアップデートを適用しない理由の一つとしてあげられるため，実際の計画においては重要な要因となり得る．もう一つは，アップデート内容に関する問題である．例えば，重大なセキュリティ上のアップデートであれば，緊急的かつ優先的に対応する必要がある．しかしながら，ここで行ったモデル化はこれらの要因を考えないこととしており，今後の課題として，これらの要因を考慮した上で，アップデート計画を行うための意思決定モデルの構築を行う予定である．

参考文献

[1] H. Okamura, M. Tokuzane and T. Dohi, “Optimal security patch release timing under non-homogenous vulnerability-discovery processes”, Proc. Int. Symp. on Software Reliab. Eng. (ISSRE’09), 120–128, 2009.

[2] C. Luo, H. Okamura and T. Dohi, “Optimal planning for open-source software updates”, in submission.

[3] H. Cavusoglu, H. Cavusoglu, and J. Zhang, “Security patch management: Share the burden or share the damage?” *Management Science*, vol. 54, no. 4, pp. 657–670, 2008.

[4] 山田，ソフトウェア信頼性モデル，日科技連，1994.

[5] H. Aiakike, Information theory and an extension of the maximum likelihood principle, Proc. 2nd Int. Symp. on Information Theory (PN. Petrov and F. Csaki, eds.), 267–281, 1973.