モジュールOSSの特性を考慮したソフトウェア信頼度成長モデルに関する一考察

山口大学大学院・創成科学研究科 宮本　翔一郎（Shoichiro Miyamoto）†
†Graduate School of Sciences and Technology for Innovation, Yamaguchi University
山口大学大学院・創成科学研究科 周　蕾（Lei Zhou）††
††Graduate School of Sciences and Technology for Innovation, Yamaguchi University
山口大学大学院・創成科学研究科 田村 慶信（Yoshinobu Tamura）†††
†††Graduate School of Sciences and Technology for Innovation, Yamaguchi University
鳥取大学・名誉教授 山田 茂（Shigeru Yamada）††††
††††Emeritus Professor, Tottori University

1　はじめに

ソフトウェアのソースコードを変更した際，ソフトウェアの取りうる状態の数は変化する．また，ソフトウェア内に潜在するフォールト数が大きく変動するとフォールトの発見確率も同様に変動することが知られている．このとき，フォールトの発見確率を活用するソフトウェア信頼性評価モデルの精度が低下することから，フォールトの発見確率の変動を考慮したモデルの開発が必要とされている．これまでの研究において，複数のリリースを考慮したソフトウェア信頼度成長モデル（SRGM）が複数の研究者によって提案されてきた [2–4]．特に，ソフトウェアの新たなバージョンがリリースされた時刻等をチェンジポイントとして扱い，推定精度の向上を試みるモデルが提案されてきた [5–8]．他方，近年の OSS においては頻繁に新たなバージョンがリリースされている．特に，開発が活発な OSS であるほどバージョンのリリース数も増加する．このとき，新たなバージョンがリリースされるたびにチェンジポイントを生成するとモデルがオーバーフィッティングを引き起こしやすくなり，従来のチェンジポイントを用いたモデルの適用が困難となる．また，OSS を利用している既存システムへの影響を懸念して古いバージョンを利用し続けるユーザも存在する．この傾向は OSS をシステムの一部として活用する，モジュール OSS において顕著に見られる．これらのユーザは最新のソフトウェアを活用しないことから OSS のデバッグに寄与しにくくなると考えられる．しかしながら，これらのユーザがソフトウェアのデバッグ活動に与えている影響について分析した研究例はない．したがって，ソフトウェアの信頼性に与える影響の解明が必要とされている．

ソフトウェアのバージョンに関する情報は，一般的にソースコードの一部として記録されている．これらの情報はリポジトリマイニングを行うことで収集が可能である．また，ソフトウェアのバージョニングに用いられる番号は主にセマンティックバージョニングと呼ばれる規則にしたがって割当が行われている．バージョン番号を活用することで，ソフトウェアの変更内容を意味づけて解析することが可能である．本論文では OSS を活用しているソフトウェアに対してリポジトリマイニングを行い，バージョン番号の変更履歴に基づくデータを収集する．また，収集したデータに対し，セマンティックバージョニングの観点からソフトウェアアップデートによる SRGM への影響を分析する．さらに，バージョン別ダウンロード数データとリポジトリ上の採用状況を比較する．

2　ソフトウェアのバージョン管理

2.1　セマンティックバージョニング

ソフトウェアのバージョンの形式は近年まで標準的な規格が策定されてこなかった．そのため，開発者がそれぞれ独自に設定した数字やアルファベットを用いた独自仕様が用いられていた．しかしながら，これらの書式ではバージョン番号からどの程度の規模の変更が行われたのか把握することが困難であった．特に，多数の依存性を抱えているシステムの場合はバージョン管理に膨大な労力を有する課題が存在していた．そこで，バージョン管理を完結にする観点から，今日では 2009 年に提案されたセマンティックバージョニングがほとんどのソフトウェアで使用されている．セマンティックバージョニングは当時のソフトウェア開発で慣習化していたバージョニングの書式を標準規格としてまとめたものである．セマンティックバージョニングのバージョン番号は $X.Y.Z$ の形式で表される．それぞれのアルファベットは次のような意味を持つ．

- メジャーバージョン X: 後方互換性のない変更を行う場合に更新する．
- マイナーバージョン Y: 後方互換性を保つ機能改修や廃止を行う場合に更新する．
- パッチバージョン Z: 後方互換性を保つフォールトの修正を行う場合に更新する．

上位のバージョン番号が変更されたとき，下位の番号は 0 にリセットされる．また，Major version が 0 のときは開発中を意味し，後方互換性のない変更を保証しない．開発者はこれらのバージョン番号を活用し，柔軟なバージョン管理を行うことが可能となる．例として，パッチバージョンのアップデートは後方互換性に影響を与える可能性が小さいことから，積極的に行うといったポリシーを定めることで，ソフトウェアの安定性と品質を重視した運用を行うことが可能となる．

2.2　開発におけるバージョン管理

OSS にはバージョンが複数あることから，使用の際にはバージョンを指定する必要がある．このとき，バージョンコントロールにはバージョン指定子が用いられる．指定子の表記方法はプログラミング言語等によって異なるものが使用されており，Python においては Python Enhancement Proposals（PEP）で PEP508 として標準化された指定子が利用されている [10]．例として，Microsoft 社が公開する OSS の 1 つである everyonecancode における構成ファイルの一部を図 1 に示す

```
...
ujson==5.9.0
urllib3==2.2.1
uvicorn==0.29.0
...
```

図 1： everyonecancode における構成ファイルの一部．

図 1 のようにパッケージの使用する外部依存パッケージおよびバージョンが記録されている．これらの構成ファイルの変更履歴をソフトウェアリポジトリ上から追跡することでソフトウェアの採用状況を把握することが可能である．本論文ではこの特徴を活用し，ソフトウェアリポジトリに対してリポジトリマイニングを実施し，採用状況に関するデータを収集する．

3　データ収集

3.1　分析対象

ダウンロード数と採用数の関連性について調査するため，GitHub の公開リポジトリから次の条件で採用状況を取得した [1]．本論文では，OSS のダウンロード傾向について把握するため，2024 年 3 月現在，Python の OSS として PyPI 上で，最もダウンロード数の多い urllib3 を選定した [12,13]．また，urllib3 のメジャーバージョンのうち，ソフトウェアリポジトリ上に利用実態のデータが多く存在するメジャーバージョン 2 系を収集対象として選択した

3.2　リポジトリマイニングによるデータの収集

次の手順で urllib3 のバージョン 2.0.2 から 2.2.1 の利用データを収集した

1. ファイル名が “requirements.txt” かつ内容で urllib3 のバージョン 2 を指定子で設定しているファイルを検索する．
2. 取得したリポジトリ，プロジェクト名，ファイルパスをもとにコミットを検索し，日時およびバージョンを抽出する．

各指定子が本論文で分析可能な数のデータ量であることを調べるため，それぞれの指定子でリポジトリ上におけるクエリ実行結果のデータ数を取得したところ表 1 のようになった．

表 1： 各指定子におけるクエリ実行結果のデータ数．

Specifier	==	>=	<=	≈=	===
Count	68.4k	402	32	288	35

表 1 より，指定子 “==” 以外の各検索結果は指定子 “==” の 1%未満と小さく，バージョンアップデートにおいて及ぼしている影響が少ないと考えられることから指定子 “==” のみに対して分析を実施した．抽出したデータをセマンティックバージョニングの観点から，メジャーバージョン，マイナーバージョン，およびパッチバージョンに分類した．また，urllib3 を新規で採用したソフトウェアと，バージョンの更新を 1 回以上行ったソフトウェアの 2 つに分類し，新規採用による採用傾向と更新採用による傾向について分析を行った．さらに，OSS のリポジトリ上における採用状況とバージョン別ダウンロード数の傾向を比較し，ダウンロード数とリポジトリ上の採用数で利用状況の差異について分析した．

4　ソフトウェアアップデートによる利用者数の変動

4.1　メジャーバージョンアップデート

メジャーバージョンアップが行われた際の移行先のバージョンを集計した結果を表 2 に示す．また，メジャーバージョンがリリースされてから更新されるまでにかかった日数を図 2 に示す．さらに，図 3 に，各リポジトリで最初に採用されたバージョンを集計した結果を示す． メジャーバージョン 1 から 2 に更新する場合は，新たなメジャーバージョンがリリースされてすぐに採用を行うユーザが多いことが分かる．一方，メジャーバージョン 2 から urllib3 を新たに採用するユーザの場合は，リリース直後の利用を控える傾向にあることが分かる．

これらの結果から，新たなメジャーバージョンがリリースされると，古いメジャーバージョンを使用していたユーザはリリース後すぐに更新を行う一方，新たに OSS を利用するユーザはリリース直後の利用を控える傾向にあることが分かる．また，新たに OSS を利用するユーザのほうが更新するユーザよりも多く存在している．したがって，新たなメジャーバージョンを多くのユーザが利用するようになるまで

表 2： 各バージョンにおける最新リリース期間の採用数.

Version	更新採用	新規採用
2.0.0	-	6
2.0.1	6	19
2.0.2	16	3187
2.0.3	454	4096
2.0.4	351	5498
2.0.5	459	1516
2.0.6	72	2184
2.0.7	200	6153
2.1.0	404	11851
2.2.0	607	2812
2.2.1	140	5593
2.0	-	1
2.1	-	1

表 3： 各バージョンにおける採用数と平均値.

バージョン	リリース日	新規	更新	平均新規採用数	平均更新採用数	リリース期間
2.0.2	2023-5-4	1596	578	46.24	16.74	34.51
2.0.3	2023-6-7	2529	704	59.92	16.68	42.20
2.0.4	2023-7-20	3481	1161	55.56	18.53	62.65
2.0.5	2023-9-20	733	379	59.05	30.53	12.41
2.0.6	2023-10-3	1064	1003	70.85	66.79	15.01
2.0.7	2023-10-18	2872	1408	107.24	52.57	26.77
2.1.0	2023-11-13	9432	1833	120.74	23.46	78.11
2.2.0	2024-1-31	1748	762	92.35	41.13	18.52
2.2.1	2024-2-18	16360	2346	135.87	19.48	120.40

には時間を要することがリポジトリ上のデータより分かる．この傾向がダウンロード数においても同様の傾向が見られるか分析する．図 4 にメジャーバージョンのダウンロード数の推移を示す．バージョン 2 が利用された後もバージョン 1 ダウンロード数は減少していない．また，バージョン 2 のダウンロード数はバージョン 1 と同様に増加傾向にあることが分かる．ダウンロード数からも，新たなメジャーバージョンが利用されるまでに時間を要することが分かる．

4.2 マイナーバージョンアップデート

表 3 に各バージョンの公開日および各バージョンにおいて最新リリース期間に新規採用された件数および更新採用された件数を示す．ただし，最新リリース期間は，各バージョンの最新バージョンが公開されてから次の最新バージョンが公開されるまでの期間とする．また，平均採用数を次のように求める．

$$n = \frac{a}{t}. \tag{1}$$

ここで，n は平均採用数，a は採用数，t は最新リリース期間である．表 3 より，新規採用のソフトウェアと更新採用を行ったソフトウェアの割合は一定でないことが分かる．特に，バージョン 2.0.7 から 2.1.0

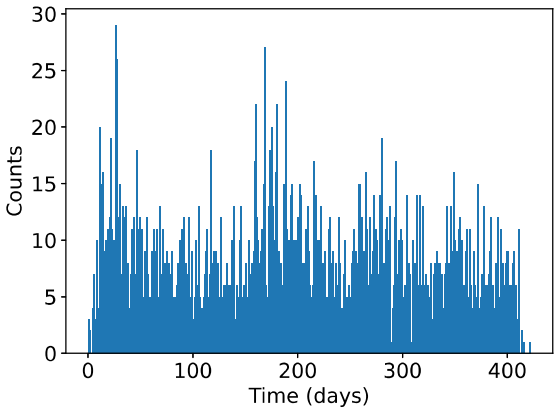


図 2： メジャーバージョンにおける
更新採用数の推移.

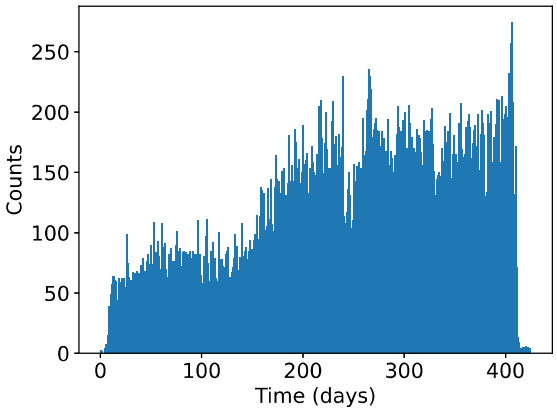


図 3： メジャーバージョンにおける
新規採用数の推移.

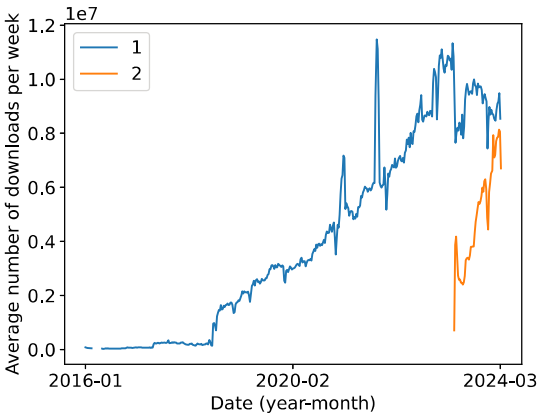
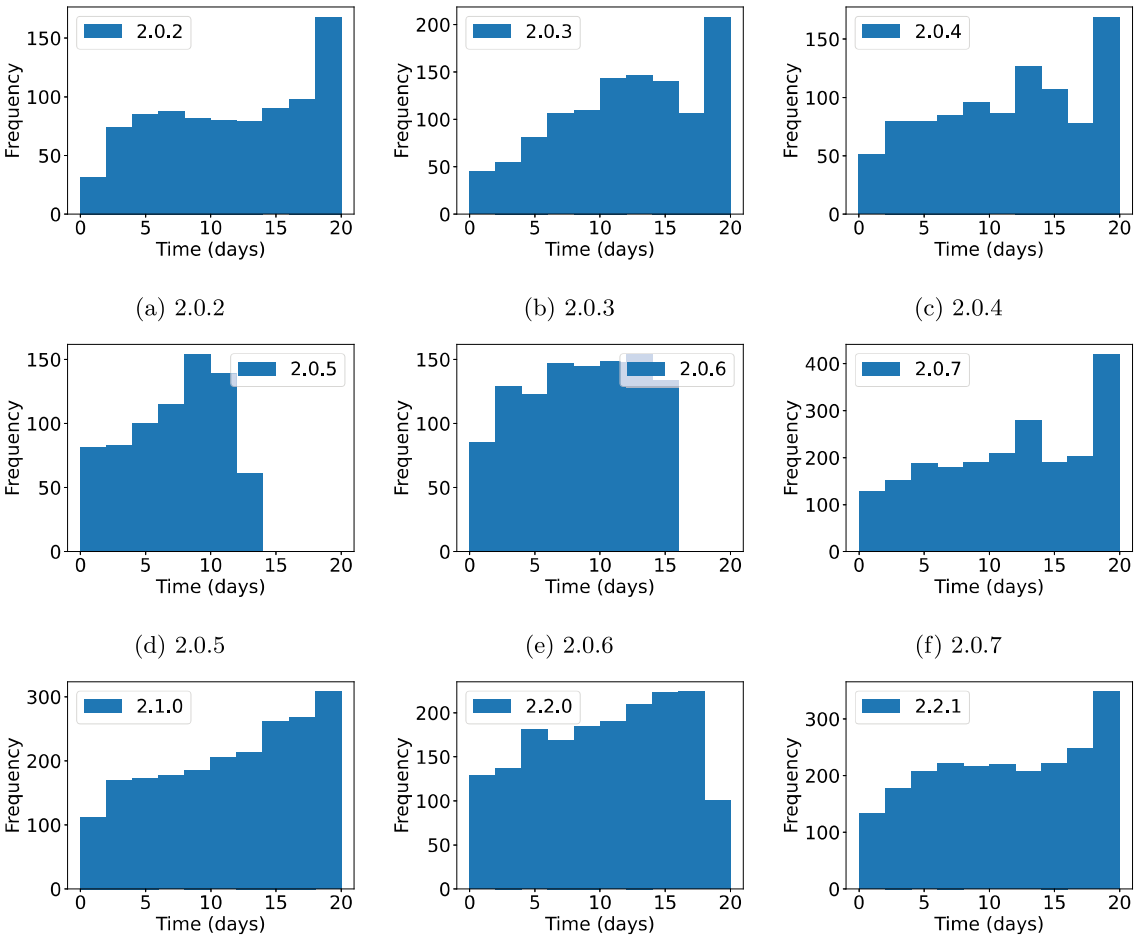


図 4： 各メジャーバージョンのダウンロード数の推移.

にマイナーアップデートが行われた際には更新のソフトウェアの比率が約 4 分の 1 から 2 分の 1 程度まで大きく変動している．したがって，新規採用と更新採用に傾向の違いが見られる．また，新規採用において平均採用数はバージョンを経るにつれて増加傾向にあることが分かる．更新採用の平均採用数は公開日数が長くなるにつれて低下する傾向にあることが分かる．

採用状況の分布について分析する．図 5 に新規で urllib3 を使用するユーザによる採用状況，図 6 にバージョン更新で urllib3 を使用するユーザによる採用状況を示す．新規採用においてはリリース直後の採用数は少なく，時間経過とともに採用数が増加する傾向にあることが分かる．一方，更新採用においてはマイナーバージョンおよびパッチバージョンを問わずリリース直後に最も大きな採用数を示し，減少する傾向にあることが分かる．新規採用と更新採用では逆の傾向を示すことが分かる．また，2.1.0, および 2.2.0 にマイナーアップデートされたとき更新採用においては，パッチバージョンに比べてリリース直後にアップデートするユーザの数が少ない．アップデートによる影響を考慮してすぐの更新を見合わせたと考えられる．さらに，マイナーアップデート後も古いマイナーバージョンにアップデートするユーザが存在することが確認できる．マイナーバージョンアップデートはメジャーバージョンの同様に更新後すぐの採用を見合わせるユーザが存在することが確認できる．



(a) 2.0.2 (b) 2.0.3 (c) 2.0.4

(d) 2.0.5 (e) 2.0.6 (f) 2.0.7

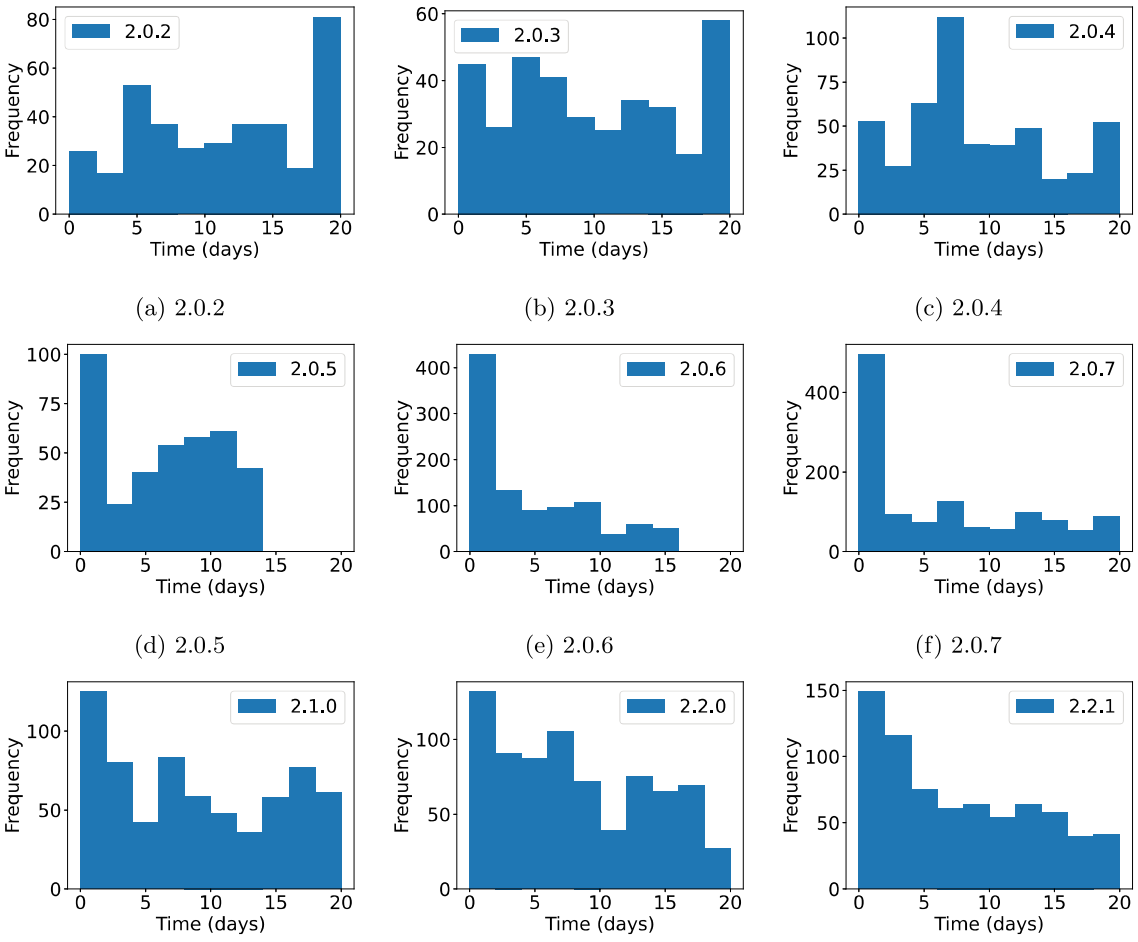
(g) 2.1.0 (h) 2.2.0 (i) 2.2.1

図 5： 各バージョンにおける新規採用数の推移.

図 7 に新規採用における各バージョンにおける採用状況の推移を示す．また，図 8 に更新採用における各バージョンにおける採用状況の推移を示す．マイナーアップデート後も，古いマイナーバージョンを使用し続けるユーザが新規および更新の両方に存在する．また，メジャーバージョンに比べて新たなマイナーバージョンは早く採用されることが分かる．図 9 に新規採用および更新採用の日別合計値を示す．また，図 10 にバージョンごとの日別ダウンロード数を示す．ダウンロード数からも，リポジトリ上の採用状況と同じ傾向がみられるセマンティックバージョニングの観点より，マイナーアップデートは後方互換性が概ね保証されている．したがって，メジャーバージョンに比べてユーザにとって採用しやすいと考えられる．一方，バージョンの更新に時間がかかっていることから一時的に更新を見合わせているユーザも存在すると考えられる．

4.3 パッチバージョンアップデート

パッチバージョンの採用傾向について分析する．図 5 および 6 より，パッチバージョンアップデートにおいてはリリース後すぐに実施されていることが分かる．また，図 7~9 より，パッチアップデート後に古いパッチバージョンは利用されなくなっていくことが分かる．図 10 より，ダウンロード数からも同様の傾向が分かる．これらの結果より，ユーザは新たなパッチバージョンの利用には積極的な傾向が観測される．また，古くなったパッチバージョンは利用しなくなる傾向にあることが分かる．これは，パッチ



(a) 2.0.2　(b) 2.0.3　(c) 2.0.4

(d) 2.0.5　(e) 2.0.6　(f) 2.0.7

(g) 2.1.0　(h) 2.2.0　(i) 2.2.1

図 6： 各バージョンにおける更新採用数の推移.

バージョンアップデートがフォールトの修正のみを目的としており，パッチバージョンが新しいフォールトが少なくなることが要因の一つとして考えられる．また，後方互換性も基本的に保証されていることも要因の一つとして考えられる．

5　提案手法

前節より，各それぞれのアップデートにおいて，次のことが分かる．

- メジャーバージョン: リリース初期に利用者が減少する．また，古いメジャーバージョンを利用し続けるユーザが一定数存在する．
- マイナーバージョン: リリース後，ユーザは新たなマイナーバージョンに更新する．また，一定数のユーザが古いマイナーバージョンを利用し続ける
- パッチバージョン: リリース後，ユーザは新たなパッチバージョンをすぐに使用する

特に，メジャーバージョンは多くのユーザがリリース直後の利用を控える傾向にあることが確認される．これはセマンティックバージョニングのルールより，後方互換性がメジャーバージョンアップにおいて保証されていないことから古いメジャーバージョンを利用し続けることが原因として考えられる．この

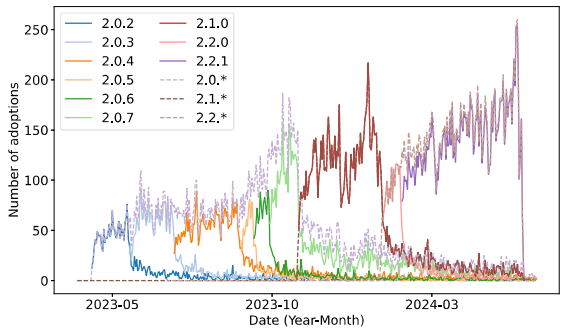


図 7： バージョン別の新規採用数の推移.

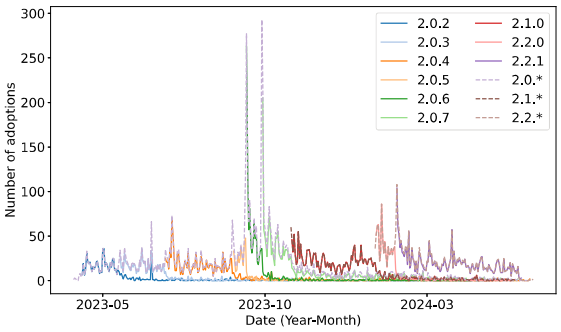


図 8： バージョン別の更新採用数の推移.

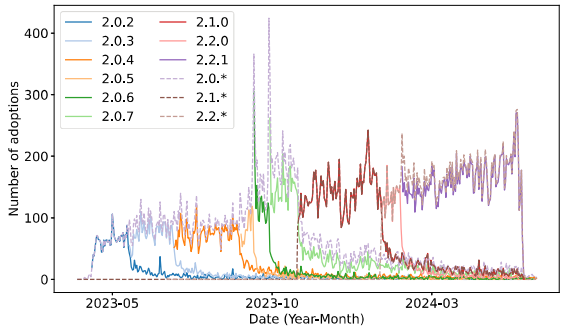


図 9： バージョン別の採用数の推移.

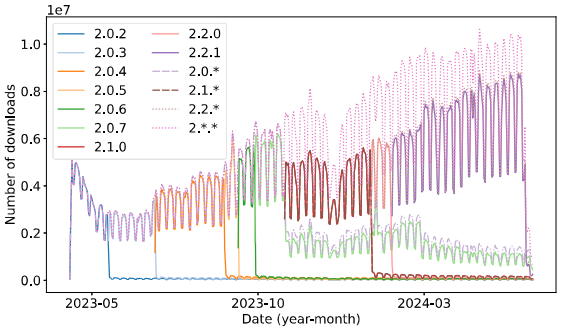
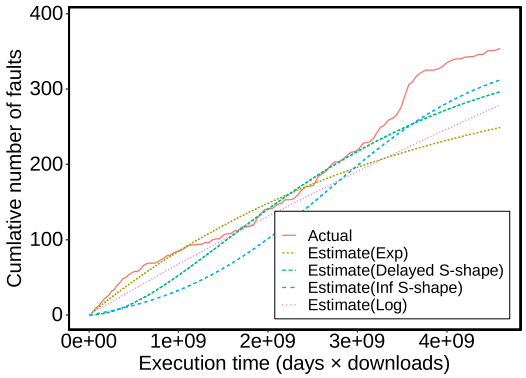


図 10： バージョン別のダウンロード数の推移.

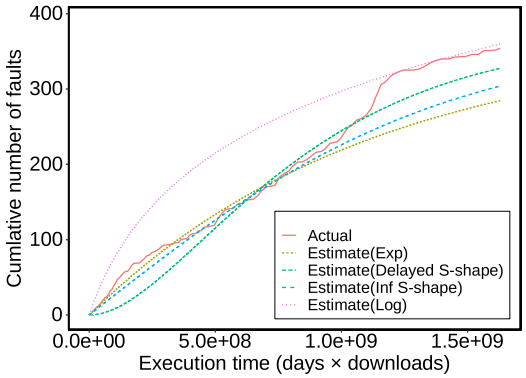
とき，メジャーバージョンのリリース時は最新メジャーバージョンを利用するユーザ数が減少することから，OSS のソースコードに対するデバッグアクティビティは減少する．一方，セマンティックバージョニングの規則よりメジャーバージョン更新時に後方互換性が保証されない変更を伴うことから潜在フォールト数は著しく増加する．このとき，メジャーバージョンリリース時はデバッグ能力の減退とフォールトが混合したデバッグ能力の変動過程が起きると考えられる．また，潜在フォールトが著しく増加した時フォールト数の発見確率も著しく増加し，新たなソフトウェアがリリースされた時と同様とみなせる．そこで，メジャーバージョンでソフトウェア信頼度成長曲線を分割し，実行時間の計算にメジャーバージョンのダウンロード数を使用する手法を提案する．提案手法により，特定のメジャーバージョンのダウンロード数を活用することで推定精度の向上が期待できる．

6　数値例

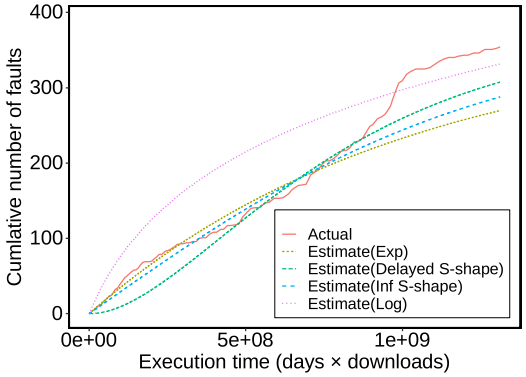
提案手法を urllib3 のバージョン 2 に適用し，モデルを構築した．SRGM の平均値関数として指数形 SRGM, 遅延 S 字形 SRGM, 習熟 S 字型 SRGM, および対数型ポアソン実行時間モデルを適用した [14–17]．また，テスト時間としてユーザ数を考慮した実行時間を適用した [18]．実行時間の算出に全てのバージョンのダウンロード数を用いた場合，メジャーバージョンのダウンロード数のみを用いた場合，および最新バージョンのみのダウンロード数を用いた場合の推定結果を図 11 に示す．評価指標として MAE, MAPE および AIC による推定結果を表 4 に示す．評価指標は多くの場合においてメジャーバージョンのダウンロード数を使用した際に最良の結果を示している．特に，メジャーバージョンを使用したときの習熟 S 字形モデルの性能は非常に高くなっている．また，最新版のダウンロード数を用いたモデルとメジャーバージョンのダウンロード数を用いたモデルは大きく推定結果が異ならないことが確認できる．これは



(a) 全てのバージョンのダウンロード数を用いた場合．



(b) メジャーバージョンのダウンロード数を用いた場合．



(c) 最新パッチバージョンのダウンロード数を用いた場合．

図 11： 推定結果．

パッチバージョンのアップデートにユーザが積極的なことから，最新版のダウンロード数とメジャーバージョンのダウンロード数が大きく異ならなかったことが要因として考えられる．

7 おわりに

本論文では，ソフトウェアリポジトリより urllib3 の利用状況を調査し，ソフトウェアダウンロード数との関係性について分析を行った．分析の結果，次のような結果が得られた．

- 新たにリリースされたパッチバージョンの更新や採用には積極的な傾向が確認できた．
- 新たにリリースされたマイナーバージョンの更新や採用にも積極的な傾向が確認できた．一方，パッチバージョンの傾向と比較すると消極的であった．
- 新たにリリースされたメジャーバージョンの採用は消極的な傾向が確認できた．
- リポジトリ上の利用状況はダウンロード数と関連性を持つことが確認できた．

また，分析結果に基づいて，バージョン別信頼度成長曲線を推定した．さらに，バージョン別信頼度成長曲線において実行時間の計算に使用するバージョンの範囲について分析を行った．その結果，メジャーバージョンのダウンロード数を利用して推定するモデルは，全てのバージョンを実行時間に利用するモデ

表 4： 推定結果の比較.

(a) 全てのバージョンのダウンロード数を用いた場合.

	Exp	Delay	Inf	Log
MAPE	0.19	1.47	1.15	0.29
MAE	31.33	25.69	38.31	31.31
AIC	616.24	608.10	562.81	542.34

(b) メジャーバージョンのダウンロード数を用いた場合.

	Exp	Delay	Inf	Log
MAPE	0.14	1.23	0.18	0.31
MAE	22.16	22.40	19.35	52.02
AIC	545.45	570.65	523.44	606.12

(c) 最新パッチバージョンのダウンロード数を用いた場合.

	Exp	Delay	Inf	Log
MAPE	0.14	1.06	0.16	0.30
MAE	25.77	25.13	22.47	50.88
AIC	590.74	587.27	559.63	654.22

ルおよび最新バージョンを実行時間に利用するモデルと比較し，多くの既存モデルにおいて最良の推定結果を示した．また，習熟 S 字形モデルは最良の推定結果を示した．提案モデルはアップデートが頻繁に発生する OSS においてより高精度な信頼性評価を実現できる．また，提案手法はメジャーバージョンごとに信頼度成長モデルを適用するため，OSS のメジャーバージョンの選定に有用である．今後は，ダウンロード数の少ない OSS に本手法を適用して影響について分析する．また，セマンティックバージョニングに従わない場合の推定精度に及ぼす影響について調査する．

謝辞

本研究の一部は，JSPS 科研費基盤研究（C）（課題番号 23K11066）の援助を受けたことを付記する．

参考文献

[1] GitHub, “GitHub,” https://github.com/.

[2] A. Tandon, A.G. Aggarwal, and N. Nijhawan, “An NHPP SRGM with Change Point and Multiple Releases,” *International Journal of Information Systems in the Service Sector*, Vol. 8, pp. 57–68, 2016.

[3] N. Nijhawan and A.G. Aggarwal, “On development of change point based generalized SRGM for software with multiple releases,” Proceedings of 2015 4th International Conference on Reliability, pp. 1–6, 2015.

[4] A.G. Aggarwal, V. Dhaka, N. Nijhawan, and A. Tandon, “*Reliability Growth Analysis for Multi-release Open Source Software Systems with Change Point*,” System Performance and Management Analytics, pp.125–137, 2018.

[5] J. Zhao, H.W. Liu, G. Cui, and X.-Z. Yang, "Software reliability growth model with change-point and environmental function," *Journal of Systems and Software*, Vol. 79, No. 11, pp. 1578–1587, 2006.

[6] N. Zhang, G. Cui, and H. Liu, "A stochastic software reliability growth model with learning and change-point," Proceedings of 2012 World Automation Congress, pp. 399–403, 2012.

[7] Z. Mengmeng and P. Hoang, "A multi-release software reliability modeling for open source software incorporating dependent fault detection process," *Annals of Operations Research*, Vol. 269, No. 1–2, pp. 773–790, 2017.

[8] S. Inoue and S. Yamada, "Change-point modeling for software reliability assessment depending on two-types of reliability growth factors," Proceedings of 2010 IEEE International Conference on Industrial Engineering and Engineering Management, pp. 616–620, 2010.

[9] T.P. Werner, "Semantic Versioning 1.0.0-beta," https://semver.org/spec/v1.0.0-beta.html, 2009.

[10] Python Enhancement Proposals, "PEP 508 – Dependency specification for Python Software Packages," https://peps.python.org/pep-0508/, 2015.

[11] Microsoft, "everyonecancode," https://github.com/microsoft/everyonecancode.

[12] Python Software Foundation, "PyPI," https://pypi.org/.

[13] urllib3, "urllib3," https://urllib3.readthedocs.io/en/stable/.

[14] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE transactions on Reliability*, Vol. 28, No. 3, pp. 206–211, 1979.

[15] S. Yamada, M. Ohba, and S. Osaki, "S-shaped software reliability growth models and their applications," *IEEE Transactions on Reliability*, Vol. 33, No. 4, pp. 289–292, 1984.

[16] M. Ohba, "Inflection S-shaped software reliability growth model," *Stochastic Models in Reliability Theory: Proceedings of a Symposium Held in Nagoya, Japan, April 23–24, 1984*, pp. 144–162, Springer, 1984.

[17] J. D. Musa, A. Iannino, and K. Okumoto, "*Software reliability: measurement, prediction, application*," McGraw-Hill, Inc., 1987.

[18] 宮本翔一郎, 田村慶信, 山田茂, 「深層学習に基づくパブリックソフトウェアリポジトリの信頼性評価法」, 京都大学数理解析研究所講究録「確率的環境下での数理的意思決定とその周辺」研究集会講究録, No. 2242, pp. 69-80, 2023.