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

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 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

近年，開発工数の短縮やソフトウェアの高機能化を目的としてオープンソースソフトウェア（OSS）を個人や企業のプロジェクトに利用することが一般的になりつつある．Synopsys 社の調査では 2022 年時点でプログラムのソースコード中 97%が OSS のソースコードであると指摘しており，OSS が社会的なインフラと化していることが分かる [1]．一方，88%のソースコードに 2 年以上メンテナンスされていない OSS のコードが使用されているとも指摘しており，OSS を選定する際に脆弱性および信頼性を検証することが重要となっている [1]．

これまでに，ソフトウェアの信頼性を検証するため，様々な研究者によってソフトウェア信頼性に関する研究が行われてきた [2–4]．ソフトウェア信頼度成長モデル（SRGM）はその研究のうちの 1 つであり，ソフトウェア開発工程の見積りやテスト工程において利用されてきた．しかしながら，これらの SRGM は，組織で実施されているウォーターフォールモデル型開発を想定しており，残存フォールト数を有限と仮定したモデルが多い [5,6]．一方，OSS は開発サイクルが一般的であり，ソフトウェア内に潜在するフォールト数は変動する．また，OSS におけるフォールトはバグトラッキングシステムへの報告によって認知される．フォールトの報告数は OSS の利用者数に影響される傾向があり，ソフトウェアの利用者数が増加した場合，報告フォールト数も同様に増加しやすくなる．このような傾向から，従来の SRGM は OSS に適用し難くなっており，OSS に適した SRGM が必要とされている．

これまで，OSS に適した SRGM として，対数型ポアソン実行時間モデルが提案されている [7]．このモデルではソフトウェアに潜在する発見可能なフォールト数が無限であることを前提に設計されており，従来の SRGM に比べて OSS のフォールト成長曲線の推移に適合しやすくなっている．しかしながら，OSS の成長曲線に対して対数型ポアソン実行時間モデルだけで包括することは困難であり，全ての OSS プロジェクトに適用できるとは限らない．

対数型ポアソン実行時間モデルの他には，深層学習を用いたフォールト数の予測がいくつかの研究者から提案されている [8–10]．この手法ではバグトラッキングシステムに登録された大規模フォールトデータを説明変数とし，発生するフォールト数を予測している．しかしながら，前述した OSS の特性から，OSS の報告フォールト数は外的な要因に左右されやすく，フォールト数の予測は容易でない．

本論文では，OSS のダウンロード数をディストリビュータから取得し，OSS のダウンロード数とフォールト数の関係性について検証する．また，ダウンロード数を考慮した，新たな信頼度評価法を提案する．さらに，深層学習を利用したソフトウェア信頼性評価モデルについて議論する．

2　SRGM

SRGM とは，ソフトウェアの時間的な要素を考慮して構築されるソフトウェア信頼性評価における解析モデルの 1 つであり，ソフトウェアの潜在フォールト数の把握や故障発生時間間隔の推定に用いられている．代表的な SRGM モデルとして指数形 SRGM が知られている．指数形 SRGM は，単位時間辺りに発見されるフォールト数は，残存するフォールト数に比例するといった仮定に基づき構築されている．指数形 SRGM の平均値関数は，次のように表される．

$$H(t) = a(1 - \exp(-bt)). \tag{1}$$

ここで，a はソフトウェアの総フォールト数，b はフォールト 1 個あたりのフォールト発見率を表す．式 (1) の平均値関数をもつ指数形 SRGM はソフトウェア内の総フォールト数が一定であると仮定されており，ソフトウェアの総フォールト数が変動しないようなソフトウェアに対して有用である．一方，OSS は開発サイクルが一般的であり，ソフトウェア内の総フォールト数がアップデートによって変動しやすい傾向にある．そこで，ソフトウェア内の潜在フォールト数を無限と仮定した対数型ポアソン実行時間モデルが提案されている．対数型ポアソン実行時間モデルの平均値関数は，次のように表される．

$$L(t) = \frac{1}{\theta} \log(\lambda\theta t + 1). \tag{2}$$

ここで，λ は初期故障強度，θ はソフトウェア故障 1 個当りの故障強度の減少率を表す．対数型ポアソン実行時間モデルはソフトウェア内の潜在フォールト数が変動する OSS に対して有用である．しかしながら，ソフトウェアの利用者数が急激に増加した場合，報告フォールト数も同様に増大する傾向にあり，対数型ポアソン実行時間モデルに適合することが困難になる．ソフトウェアの利用者数を考慮した SRGM が必要であると考える．

3　利用者数とフォールト数の関係

3.1　ライブラリの選定

OSS の利用者数が増加した場合，フォールト数が増大する傾向にある．しかしながら，OSS は改変，再配布が自由とされている場合が多く，正確な利用者数を把握することは困難である．一方，利用者の多くはディストリビュータを介して OSS をダウンロードしており，ディストリビュータのダウンロード数を調べることでソフトウェア利用者数と同等の指標を得ることができる．本論文では，次の条件にてダウンロード数を収集した．

- Node.js のビルド系ライブラリのうち，主要なビルドツールを選定した．
- GitHub の Issue に基づき報告フォールトを取得した．
- npm よりパッケージのダウンロード数を取得した．

以上の条件から，本論文では，Nx，SWC，webpack，および Vite の 4 つを選定した [11–14] .

3.2　累積報告フォールト数と経過時間の関係

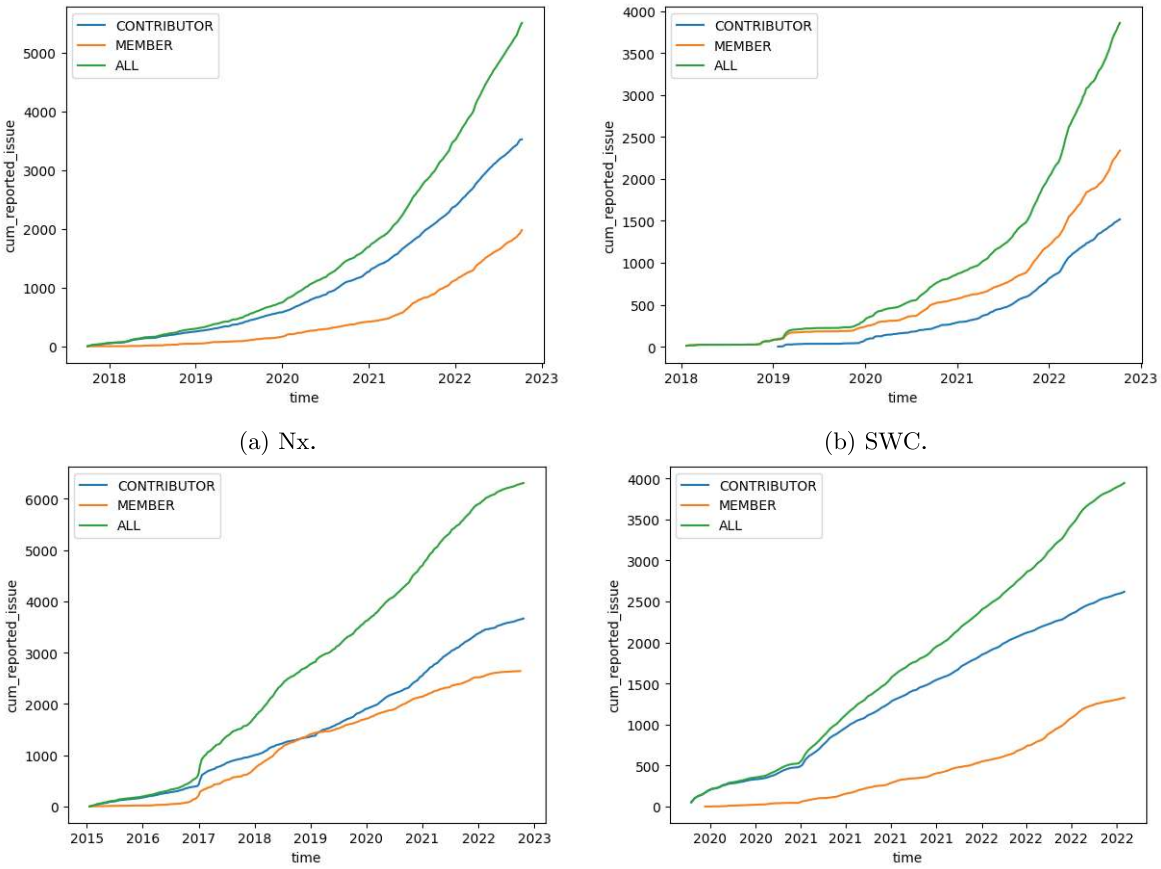
収集したデータセットのうち，報告者別にフォールトの分類を行った．フォールトのうち，利用者によって報告されたフォールトおよび開発メンバーによって報告されたフォールトの 2 つに分類した．図 1 に累積発見フォールト数と時間の関係を示す．

図 1 より，Nx および SWC においてはフォールト発見数が指数関数的に増大しており，特定の値に収束する傾向がないことが分かる．また，利用者と開発者の報告フォールト数の傾向は，類似していることが確認できる．

webpack では，ソフトウェアリリース後，利用者および開発者ともに，累積フォールト発見数は単調増加したのち，2022 年付近から累積フォールト発見数の伸びが鈍化していることが分かる．利用者および開発者の傾向は類似していないものの，累積フォールト発見数の鈍化は共通して発生していることが分かる．

Vite では，利用者および開発者ともに累積フォールト発見数の伸びが鈍化しない傾向にあることが分かる．Vite は 4 つのパッケージの中でも最も新しいパッケージであり，今後 Nx および SWC と同様に指数関数的なフォールト数の増化が起きる可能性もあると想定できる．

全てのパッケージにおいて，累積フォールト発見数が収束する傾向がほとんど観測されず，OSS の累積フォールト数と時間の関係においては，ソフトウェアの安定性を評価することが難しいことが分かる．また，一般的なソフトウェアに適用される SRGM と類似する曲線が存在しないことが分かる．



(a) Nx.　(b) SWC.

(c) webpack.　(d) vite.

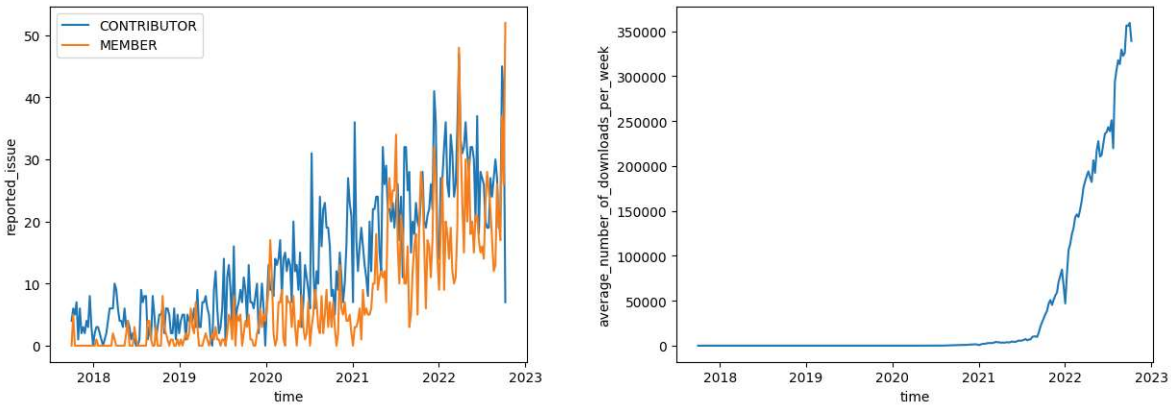
図 1： 累積報告フォールトと経過時間の関係．

3.2.1　報告フォールト数とダウンロード数の関係

図 2~5 にそれぞれ，Nx，SWC，webpack，および Vite における報告フォールト数，ダウンロード数，および時間の関係を示す．Nx においてはダウンロード数は 2021 年頃から増え始めており，報告フォールト数も同様に増加していることが分かる．また，2021 年初期のダウンロードと 2022 年末期にかけてダウンロード数は膨大な数になっている一方，報告フォールト数はそれほど伸びていないことが分かる．

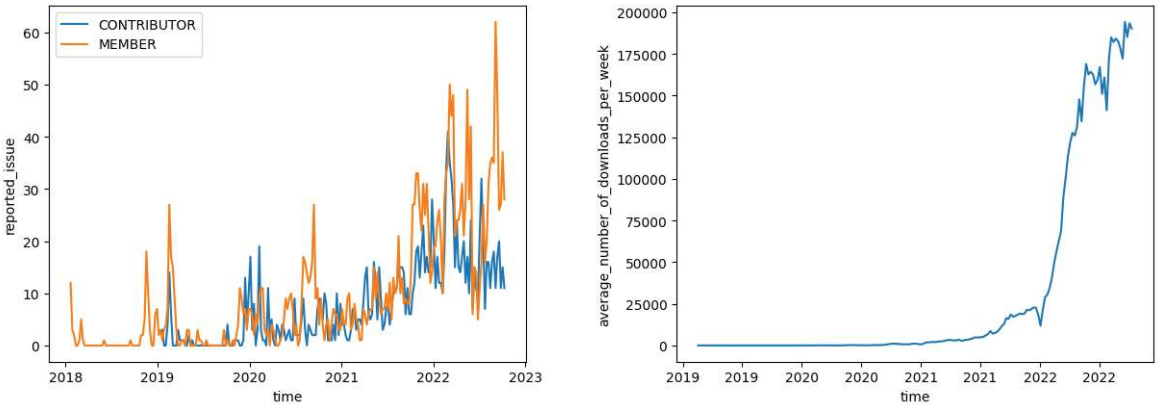
また，SWC においても同様の傾向が見られる．webpack ではダウンロード数がリリース当初から徐々に伸びていることが分かる．さらに，リリースの初期を除き，Nx および SWC で見られたような急激なフォールト数の増加は起きていないことが分かる．Vite では，webpack の初期のような傾向が観測されていることが分かる．特に，Nx および SWC のダウンロード数が急激に増加した 2021 年付近とも同様にフォールト数が増大していることが分かる．

これらのグラフより，ダウンロード数が急激に増加した場合，報告フォールト数が急激に増加することが分かる．また，利用者の報告フォールト発見数と開発者の報告フォールト数は同様のトレンドを示しやすい傾向にあり，利用者の報告するフォールトが開発者のフォールト発見に役立てられていると考えられる．



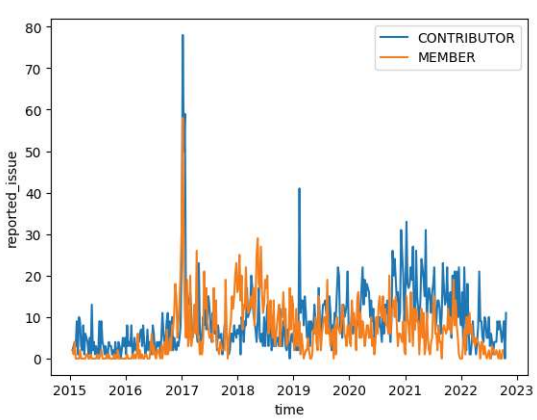
(a) 報告フォールト数と時間の関係.　(b) ダウンロード数と時間の関係.

図 2： Nx における報告フォールト数とダウンロード数の関係.

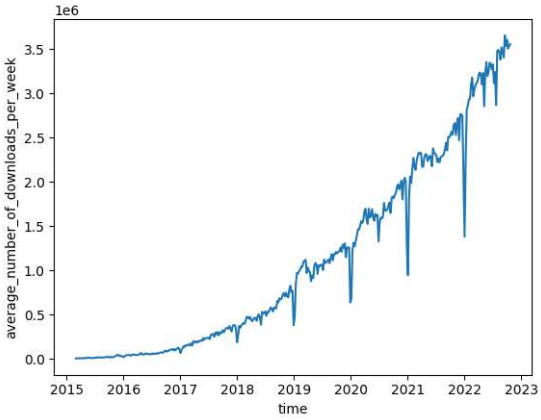


(a) 報告フォールト数と時間の関係.　(b) ダウンロード数と時間の関係.

図 3： SWC における報告フォールト数とダウンロード数の関係.

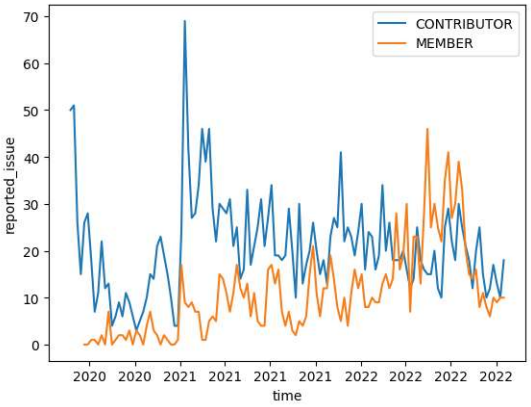


(a) 報告フォールト数と時間の関係.

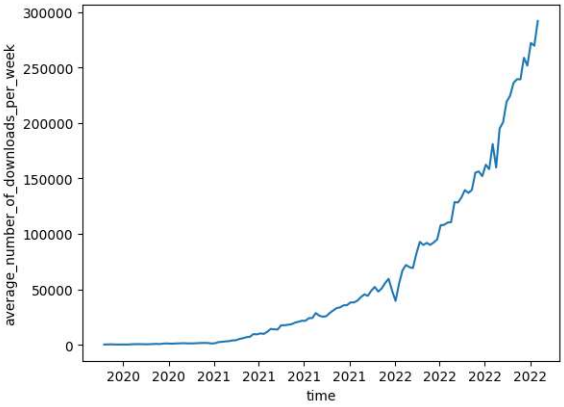


(b) ダウンロード数と時間の関係.

図 4： webpack における報告フォールト数とダウンロード数の関係.



(a) 報告フォールト数と時間の関係.



(b) ダウンロード数と時間の関係.

図 5： Vite における報告フォールト数とダウンロード数の関係.

4　提案手法

4.1　ソフトウェア推定実行時間

前節より，OSS の累積フォールト数はダウンロード数の影響を受けて信頼度成長曲線が一般的な SRGM と異なる形状を示すことが確認された．また，ソフトウェアの報告フォールト数がダウンロード数の影響を受けることも分かった．OSS に累積フォールト発見数を使用した信頼性評価を行う場合，これらの要素を念頭においたモデルの設計が必要になると考える．

一般的なソフトウェア開発では，テスト工程において人員が大幅に変動しないため，ソフトウェアのテスト時間は単純な経過時間に従う傾向にある．一方，仮にテスト担当者の人員が大きく変動した場合，テスト目標の達成も早くなると考えられる．本論文では，この特徴に着目し，ソフトウェアのダウンロード数をテスト担当者の数とみなしてソフトウェアの推定実行時間を求めることでダウンロード数を考慮した OSS 信頼性評価法について議論する．まず，i 番目の集計回数を以下のように定義する．

$$T_i = \sum_{k=1}^{i} \Delta t_k N_k. \tag{3}$$

ここで，i は i 番目の単位時間を表す．また，Δt_k は k 番目と $k-1$ 番目における経過時間を表す．さらに，N_k は k 番目におけるソフトウェアのダウンロード数を表す．式 (3) により，ダウンロード数を考慮したソフトウェア推定実行時間が求められる．ダウンロード数の増加によってフォールト発見数が増大した場合，ソフトウェア推定実行時間も同様に増加するため信頼度成長曲線の傾きが補正される．

4.2　深層学習モデル

提案したソフトウェア推定実行時間をフォールトデータに適用し，深層学習モデルを構築して信頼性評価を行った．フォールトデータは表 1 のようにして前処理を行った．表 1 のうち “bugs ” を目的変数に，残りの変数を説明変数とし標準化した．また，説明変数としてダウンロード数が有用であるか検討を行うため．ダウンロード数を説明変数として用いる場合と用いない場合の 2 つに分け，それぞれ学習を行った．図 6 に，学習に用いたモデルの構造を示す．さらに，深層学習に基づく報告フォールト数の推定結果を図 7 に示す．最適化方法には adam を使用した [15].

5　数値例

5.1　推定実行時間を用いた信頼度成長曲線

図 7 から，推定実行時間を適用した結果，利用者，開発者および全ての利用者において対数型ポアソン実行時間モデルに近い形状を示した．図 7 より，webpack および vite において累積フォールト数の増加率は著しく減少しておりソフトウェアの信頼度が高い状態になっていると考えられる．また，Nx および SWC においては累積フォールト発見数の増加率が上昇傾向にあり，ソフトウェアの信頼度が高いとは言えない状態にあることが分かる．さらに，全てのパッケージにおいてリリース初期に大半のフォールトが発見されていることが分かる．

表 1： 前処理方法

項目	方法
author_association, locked	Count encoding
reactions,reported_issue, downloads	Frequency encoding
user_login	One-Hot encoding

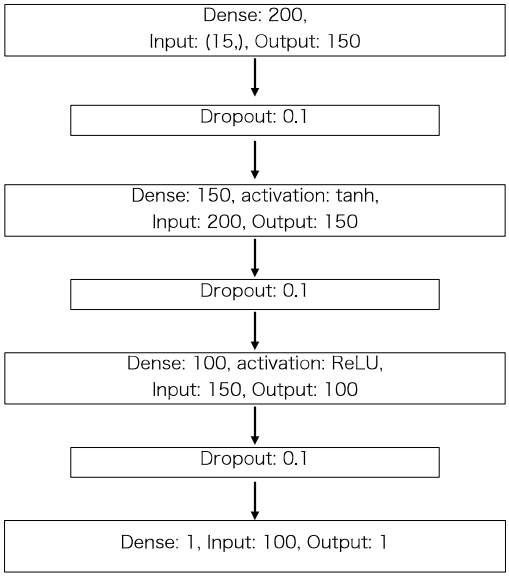


図 6： 深層学習モデルの構造.

5.2 フォールト数の推定

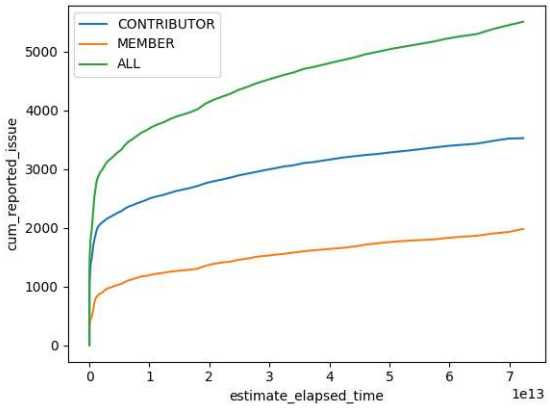
図 8 に，各パッケージにおける発見フォールト数の推定結果を示す．図 8 より，Nx，SWC，および vite においてフォールト数の推定が精度良く行えたことが分かる．また，説明変数におけるダウンロード数の有無は推定結果に大きな差異をもたらしていなかった．一方，webpack においては推定結果がテストデータよりも高くなっており，推定結果が良好でないことが確認できる．また，webpack でもその他 3 つのパッケージと同様，説明変数におけるダウンロード数の有無は推定結果に大きな差異をもたらさなかった．

図 9 に，各パッケージにおける累積フォールト発見数の推定結果を示す．図 9 より，Nx，SWC，および Vite においては精度良く累積フォールト発見数の推定が行えていることが分かる．一方，webpack ではテストデータから乖離した曲線が示されており，推定結果が良好でないことが分かる．webpack においては，推定実行時間が他のライブラリに比べて膨大なことが挙げられる．webpack は 4 つのライブラリの中で最もダウンロード数が多く，推定実行時間の変化が大きくなっている．また，信頼性の向上によって報告フォールト数そのものも少なくなっており，深層学習が他のライブラリに比べて困難であったと考えられる．

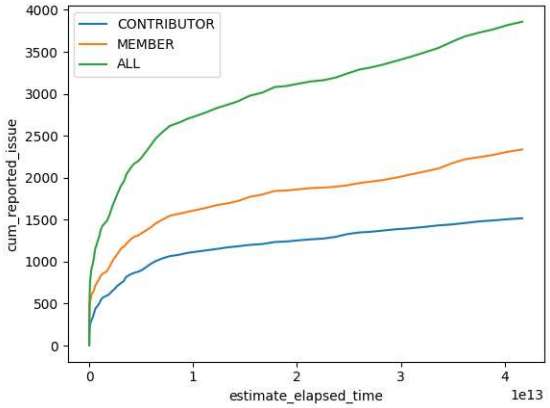
6 おわりに

本論文では，OSS の利用者数が SRGM に与える影響について着目し，利用者数を考慮した信頼性評価法を提案した．利用者数と報告フォールト数の関係について調査した結果，利用者数が増加した場合フォールト数も同様に増加することが分かった．また，利用者数の増加率に比べて報告フォールト数の増加率は小さく，フォールトが発見されるについてソフトウェアの信頼度が増加していることが確認された．

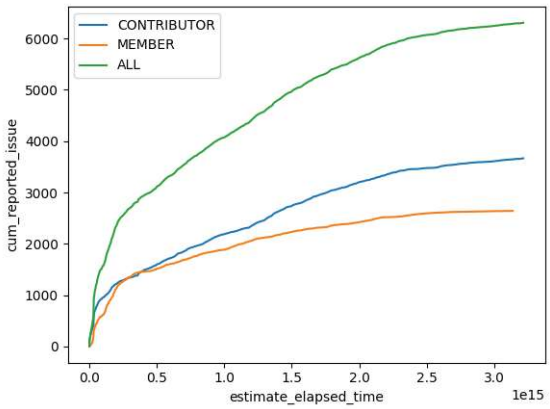
また，利用者数を考慮した信頼性評価法について提案した．提案手法を OSS に適用した結果，利用者数が急激に変動する OSS において適切な信頼性評価を行うことに成功した．提案手法は利用者数に応じて実行時間が調整されるため，利用者数の変動する環境においても妥当な信頼性評価を行うことがで



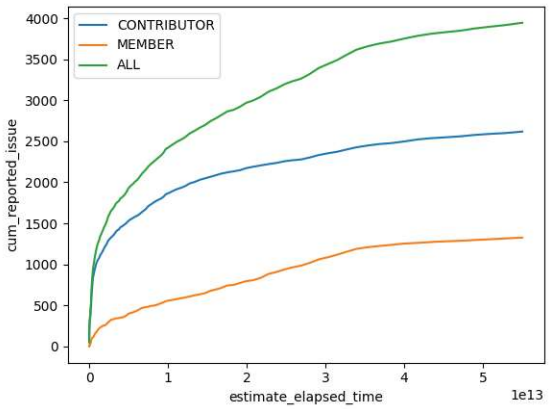
(a) Nx.



(b) SWC.

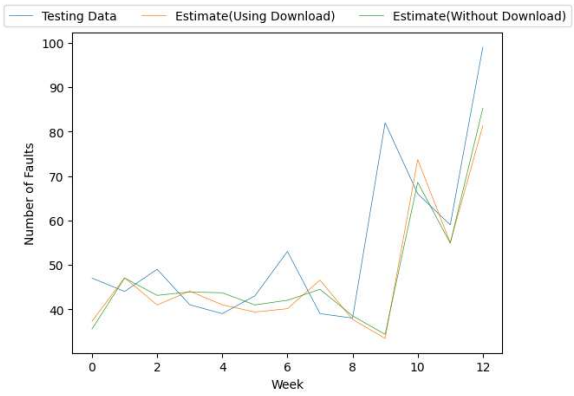


(c) webpack.

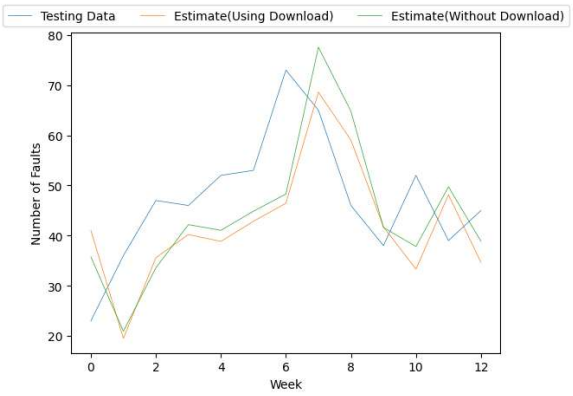


(d) vite.

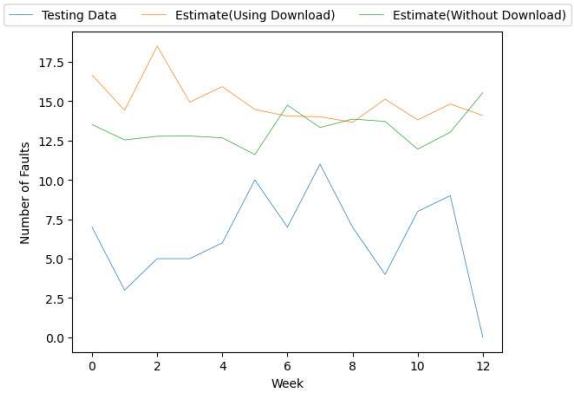
図 7： 深層学習に基づく累積フォールト報告数の推定結果.



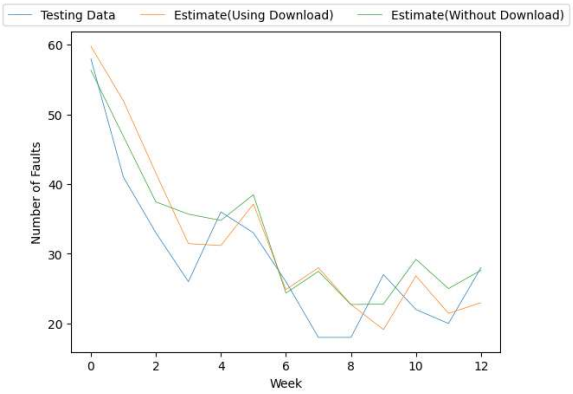
(a) Nx.



(b) SWC.

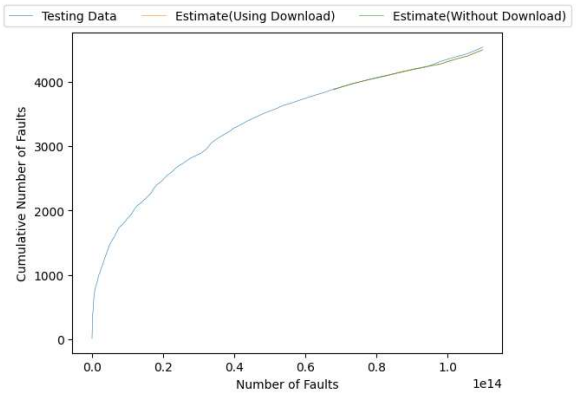


(c) webpack.

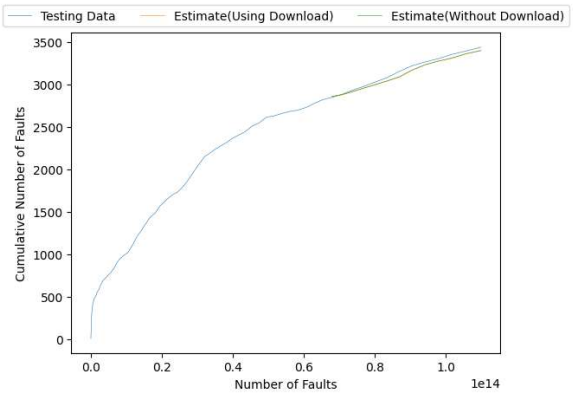


(d) vite.

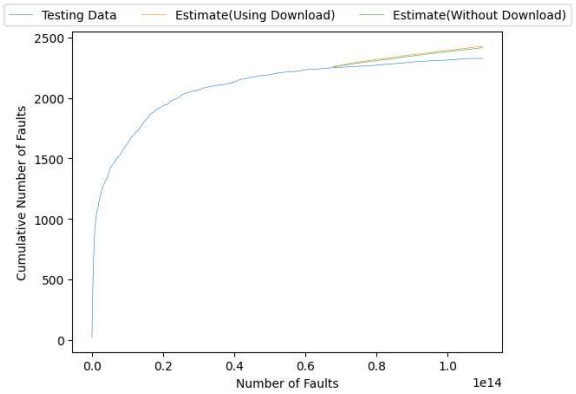
図 8： 各パッケージにおける報告フォールト数の推定結果.



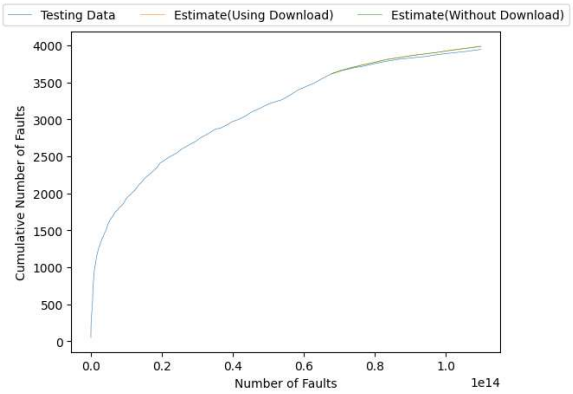
(a) Nx.



(b) SWC.



(c) webpack.



(d) vite.

図 9： 各パッケージにおける累積フォールト発見数の推定結果.

きる．

さらに，深層学習を用いた OSS 信頼性評価法を提案した．提案手法は精度の良い推定を実現しており，OSS のフォールト推定に有用である．また，説明変数においてダウンロード数の有無がもたらす提案手法への影響について検証した．その結果，ダウンロード数を使用してもモデルの精度に有意な差はなく，有用性を確認できなかった．

今後の展望として，深層学習に基づく OSS 信頼性評価法において，報告フォールト数と推定実行時間の 2 つを推定して示したい．また，説明変数として有用性を確認できなかったダウンロード数について再度検討を行いたい．

謝辞

本研究の一部は，JSPS 科研費基盤研究（C）（課題番号 20K11799）の援助を受けたことを付記する．

参考文献

[1] Synopsys, “2022 Open source security and risk analysis report,” https://www.synopsys.com/, 2022.

[2] S. Yamada and Y. Tamura, *OSS reliability measurement and assessment*, Springer Series in Reliability Engineering, Springer, 2016.

[3] M.R. Lyu, ed., *Handbook of software reliability engineering*, IEEE Computer Society Press Los Alamitos, CA, 1996.

[4] S. Yamada, *Software reliability Modeling: Fundamentals and Applications*, Springer-Verlag, Tokyo/Heidelberg, 2014.

[5] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application*, McGraw-Hill, New York, 1987.

[6] P. K. Kapur, H. Pham, A. Gupta, and P. C. Jha, *Software reliability assessment with OR applications*, Springer-Verlag, London, 2011.

[7] K. Okumoto, “A statistical method for software quality control,” *IEEE Transactions on Software Engineering*, vol. SE–11, no. 12, pp. 1424–1430, 1985.

[8] Y. Tamura and S. Yamada, “AI approach to fault big data analysis and reliability assessment for open-source software,” in *System Reliability Management*, pp. 1–17, CRC Press, 2018.

[9] N. Karunanithi, D. Whitley, and Y. K. Malaiya, “Using neural networks in reliability prediction,” *IEEE Software*, vol. 9, no. 4, pp. 53–59, 1992.

[10] T. Dohi, Y. Nishio, and S. Osaki, “Optimal software release scheduling based on artificial neural networks,” *Annals of Software engineering*, vol. 8, no. 1, pp. 167–185, 1999.

[11] Nx, “Smart, fast extensible build system,” https://nx.dev/, 2022.

[12] SWC, “Rust-based platform for the web,” https://swc.rs/, 2022.

[13] webpack, “webpack,” https://webpack.js.org/, 2022.

[14] vite, “Vite — next generation frontend tooling,” https://vitejs.dev/, 2022.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the International Conference on Learning Representations*, pp. 1–15, 2015.