ソフトウェア信頼性研究 —モデリングアプローチ—

Researches on Software Reliability:
Modeling Approaches

山田 茂 Shigeru YAMADA
田村慶信 Yoshinobu TAMURA

井上真二 Shinji INOUE

アブストラクト 品質の高いソフトウェアを開発できるようにプロジェクトを確実に成功させるためには、品質を重視したプロジェクトマネジメント（品質指向ソフトウェアマネジメント）を徹底し、その技術の持続的向上を図る必要がある。本稿では、品質指向ソフトウェアマネジメントの立場から、実際のソフトウェアプロジェクトに対する実用性を考慮して、ソフトウェア信頼性工学における計測・評価技術とマネジメント技術に関する四つのトピックスについて、その具体的適用例を挙げながら解説する。

キーワード 品質指向ソフトウェアマネジメント、ソフトウェア信頼性モデル、オープンソースソフトウェア、2 変量モデル、チェンジポイント、ディープラーニング

Abstract Quality-oriented software management technologies and their sustainable improvements are the key factors for ensuring the success of software projects and developing high-quality software. Focusing on the four topics related to software reliability measurement/assessment and management technologies, we explain their technologies from the viewpoint of quality-oriented software management with some examples of applications to actual software project data.

Key words Quality-Oriented Software Management, Software Reliability Modeling, Open-Source Software, Bivariate Modeling, Change Point, Deep Learning.

1. はじめに

高品質ソフトウェア開発を目指すためには、ソフトウェアの開発プロジェクトを科学的管理の下で推進し、開発プロセスと開発成果物（ドキュメントとソースコード）に対する評価尺度として、何よりも QCD（Quality, Cost, Delivery：品質，コスト，納期）指標を持つことが重要である。これは、経営管理要素の中でも品質を中核として、短納期とコスト削減を意識しながら、顧客ニーズに応えるソフトウェア製品の開発ができるように、PDCA（Plan-Do-Check-Act）サイクルの中で開発プロジェクトをマネジメント（制御と管理）していくという考え方である。

したがって、品質の高いソフトウェア製品を開発できるようにプロジェクトを確実に成功させるためには、ソフトウェアプロジェクトの QCD に関わる問題を含む開発プロセスを持続的に

図1 品質指向ソフトウェアマネジメントとプロセス改善

改善していくことが重要である。そのためには、開発スケジュールを詳細に計画し、開発プロセスの状態の是非を判定できる定量的基準を持った上で、プロセスの成熟度を向上させていく必要がある（図1）。企業の実践的研究者を含む筆者らの共同研究グループでは、ソフトウェア品質の持続的向上という課題に取り組む中で、ソフトウェア信頼性工学の効果的基盤技術を開発し、有効なプロジェクトマネジメント技術を導出して、その効果の評価と検証を行ってきた。これらの研究から品質指向ソフトウェアマネジメント⁽¹⁾という考え方を提唱し、品質を重視したプロジェクトマネジメントを徹底しながら、その技術の持続的向上を図る重要性を強調してきた。すなわち、開発プロセスの計測・制御とソフトウェア製品品質の予測・評価をプロジェク

山田 茂 正員：シニア会員 鳥取大学大学院工学研究科

E-mail yamada@tottori-u.ac.jp

井上真二 正員 関西大学総合情報学部

E-mail ino@kansai-u.ac.jp

田村慶信 正員 東京都市大学知識工学部

E-mail tamuray@tcu.ac.jp

Shigeru YAMADA, Senior Member (Graduate School of Engineering, Tottori University, Tottori-shi, 680-8552 Japan), Shinji INOUE, Member (Faculty of Informatics, Kansai University, Takatsuki-shi, 569-1095 Japan), and Yoshinobu TAMURA, Member (Faculty of Knowledge Engineering, Tokyo City University, Tokyo, 158-8557 Japan).

電子情報通信学会 基礎・境界サイエティ

Fundamentals Review Vol.12 No.1 pp.38-50 2018 年 7 月

©電子情報通信学会 2018

トマネジメントの中で実践しながら、マネジメント技術の獲得・定着・改善のサイクルを絶えず回していく中で、プロダクトとプロセスの品質の向上が図れるのである。また、ソフトウェア開発は、人的要因が絡み、その出来栄である品質/信頼性に影響が及んでいるプロジェクトチームにより進行していくことも忘れてはならない。

本稿では、上述した中でもソフトウェア信頼性工学の基盤技術のうち、ソフトウェア信頼性のモデリングアプローチに焦点を当て、「品質が作り込まれたことを根拠をもって説明する」ためのソフトウェア信頼性モデル開発について議論する。品質指向ソフトウェアマネジメントでは、ソフトウェア信頼性モデルから得られた各種の評価尺度に基づいて品質/信頼性を定量的に評価することが、品質に基づいたプロジェクトマネジメントを実施する上で重要であり、品質指向ソフトウェアマネジメントの定着化を図るためにも重要課題である。そのためには、推定・予測精度が良く現場への適用性・汎用性の高いソフトウェア信頼性モデルが必要となる。本稿で取り上げたのは、以下の四つのトピックスである。

- (1) ソフトウェア開発形態の一つであるオープンソースソフトウェア (OSS) プロジェクトに対するソフトウェア信頼性モデルを用いた品質評価法と最適リリース (出荷) 問題。
- (2) OSS 信頼性モデリングの考え方とディープラーニングによる信頼性要因分析。
- (3) ソフトウェア信頼性モデリングにおける考慮すべき信頼性要因として、ソフトウェアの動作時間 (開発プロセス中のテスト期間あるいはユーザの運用時間) をカレンダー時間とテスト労力の関数として見た 2 変量ソフトウェア信頼性モデル。
- (4) ソフトウェア信頼性モデルによるテスト工程での定量的信頼性評価の推定・予測精度に影響するチェンジポイント (変化点) を考慮したソフトウェア信頼性モデル。

2. OSS プロジェクトに対する品質評価法と最適リリース問題

近年注目されているオープンソースソフトウェア (OSS) の開発プロジェクトでは、ユーザが OSS を使用している間に何らかの不具合が発生すると、バグトラッキングシステムへ障害内容が登録される。その情報を開発者が確認し、ソースコードを改良することにより、新たなバージョンの OSS が公開されるといふ開発サイクルによって OSS プロジェクトの開発は進められる (図 2)。つまり、OSS プロジェクトでは、従来の開発形態のような、フォールト (いわゆるバグと同義) の発見・修正・除去をテストチームにより行い、品質確認と共に品質/信頼性を評価するためのテスト工程が存在しない⁽²⁾。OSS 開発は、主にネットワーク上で複数の開発者が共同開発を行っており、開発速度が迅速であるという利点を持つ一方で、有効な品質管理手法が確立されておらず、OSS の信頼性に関する課題は多い。そこで本

図 2 OSS プロジェクトにおける開発プロセス

章では、ソフトウェア信頼性評価モデルの一つである対数型ポアソン実行時間モデルを用いて、管理図による OSS 開発プロセスにおける品質の安定性の評価と開発期間の見積りが可能であることを示す。

2.1 対数型ポアソン実行時間モデル

本節では、非同次ポアソン過程^{(3), (4)} (nonhomogeneous Poisson process, 以下 NHPP と略す) に基づくソフトウェア信頼度成長モデルによる、OSS プロジェクトの信頼性評価について議論する。検出可能フォールト数が無限であると仮定された NHPP に基づく、対数型ポアソン実行時間モデル^{(3), (5)}の平均値関数 $\mu(t)$ (時間区間 $(0, t]$ における総期待フォールト発見数)、及び強度関数 $h_{\mu}(t)$ (単位開発時間当りに検出されるフォールト数) は、それぞれ次式で与えられる。

$$\mu(t) = \frac{1}{\theta} \ln(\lambda_0 \theta t + 1) \quad (\lambda_0 > 0, \theta > 0), \quad (1)$$

$$h_{\mu}(t) \equiv \frac{d\mu(t)}{dt} = \frac{\lambda_0}{\lambda_0 \theta t + 1} = \lambda_0 \exp[-\theta \mu(t)]. \quad (2)$$

また、式 (2) から、

$$\ln h_{\mu}(t) = \ln \lambda_0 - \theta \mu(t), \quad (3)$$

の関係を導く。ここで、 λ_0 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。ソフトウェア故障とは、ソフトウェアが期待どおりに動作しないことと定義される。また、ソフトウェア信頼性とは、ソフトウェア故障を引き起こすことなく動作することができる性質や度合いであり、これを確率で表現したものがソフトウェア信頼度である。図 3 は、実際の OSS プロジェクト (Android 1.5 NDK, Release 1) のバグトラッキングシステムのデータを用いてモデルパラメータ λ_0 及び θ を推定した上で、式 (1) の平均値関数の推定結果を示したものである。

2.2 管理図法

管理図とは、品質や製造工程が統計的に安定状態にあるかを判断するために使用するグラフである。ここでは、管理図の考え方を用いて、OSS 開発プロセスにおける品質の安定性を評価する。式 (3) において $Y = \ln h_{\mu}(t)$ と置くと、推定された Y は次式となる。

$$\hat{Y} = \ln \hat{\lambda}_0 - \hat{\theta} \hat{\mu}(t_G). \quad (4)$$

図3 対数型ポアソン実行時間モデルの平均値関数

Yの観測値

図4 Android プロジェクトに対する管理図

式(4)の t_G は所定のテスト時刻である。式(4)は、プロジェクトが良好に進行し、フォールトの検出により安定的に信頼度成長が確保されているとき、強度関数の対数値は、総検出フォールト数の増加とともに、線形的に減少することを示している^{(1)・(3)}。したがって、式(4)において、 $t = t_G$ で推定された対数値 $Y_G = \ln h_\mu(t_G)$ に対する $100(1 - \alpha)\%$ 信頼区間は、回帰式の分散分析の結果を用いて次式により与えられる。

$$(\ln \hat{\lambda}_0 - \hat{\theta} y_G) \pm t \left(n - 2, 1 - \frac{\alpha}{2} \right) \sqrt{V(\hat{Y}_G)}. \quad (5)$$

ここで、 y_G は $t = t_G$ のときの総発見フォールト数、 $t(k, p)$ は自由度 k の t 分布の $100p\%$ 点、 $V(\hat{Y}_G)$ は $\hat{Y}_G$ の分散である。式(4)を中心線(CL)、式(5)を上方及び下方管理限界線(UCL及びLCL)として用いることで、管理図を作成できる。図4は、前出のAndroid 1.5 NDKプロジェクトデータを用いて作成した管理図である。OSS開発プロジェクトが進行するにつれて、単位開発時間当りに検出されるフォールト数に基づいたLCLを超えることもあったが、プロジェクトを通じて品質が安定化傾向を示していることが分かる。

2.3 OSS 最適リリース問題

開発中のOSSの開発を終了して、実際の運用段階に移行するのに最適な開発期間を求める問題をOSSの最適リリース(出荷)問題⁽³⁾とする。OSSプロジェクトが目標フォールト発見率(ソフトウェア故障強度) λ_F に到達する時間 τ_F は、次式で与えられる。

$$\tau_F = \frac{1/\lambda_F - 1/\lambda_0}{\theta}. \quad (6)$$

次に、総期待ソフトウェアコストの定式化を行う。 c_1 を開発段階のフォールト1個当りの修正コスト、 c_2 を運用段階のフォールト1個当りの修正コスト、 c_3 を単位開発時間当りのコスト、 c_4 を機会損失コスト係数と定義する。また、 T をリリース時刻、 T_U をメジャーバージョンアップまでの運用時間とする。このとき、総期待ソフトウェアコスト $C(T)$ ($T > 0$)は、

$$C(T) = c_1 \mu(T) + c_2 \{\mu(T_U) - \mu(T)\} + c_3 T + c_4 T^2, \quad (7)$$

で与えることを考える。また、式(7)の $C(T)$ を最小化する時間 $T = T_0$ は次式で与えられる。

$$T_0 = \frac{-(2c_4 + c_3 \lambda_0 \theta) + \sqrt{(2c_4 - c_3 \lambda_0 \theta)^2 + 8c_4 \lambda_0^2 \theta (c_2 - c_1)}}{4c_4 \lambda_0 \theta}. \quad (8)$$

更に、信頼性要因として目標フォールト発見率を取り上げ、コストを最小化するにあたり、 T_0 におけるフォールト発見率 λ_p を目標フォールト発見率 λ_F 以下にする要件も付加する。すなわち、OSS最適リリース問題は、

$$\begin{cases} \text{Minimize } C(T) & \text{subject to } \lambda_p \leq \lambda_F \\ \text{for } c_2 > c_1 > 0, c_3 > 0, c_4 > 0, \end{cases} \quad (9)$$

として定式化される。 $h_\mu(0) > c_3/(c_2 - c_1)$ のとき、式(9)を満たす最適リリース時刻 T^* は、次式で与えられる。

$$T^* = \max\{T_0, \tau_F\}. \quad (10)$$

95%信頼限界での目標ソフトウェア故障強度 λ_F に到達する時間 τ_F 、並びに式(8)の T_0 を推定して、表1のように信頼性要件とコスト要因を考慮した最適リリース時刻 T^* を算出することができた。

3. オープンソースソフトウェアの信頼性

現在、2.で議論したような様々なOSSが利活用されている。コスト削減、短納期、標準化などの恩恵を受けることができることから、OSSを導入する組織も増えている。一方で、OSSに関しては、ライセンス上の問題や品質上の問題などから、その導入に踏み切れない組織も多い。ここでは、OSSの品質上の問題について議論する。

まず、OSSと一般的なプロプライエタリソフトウェアとの違いとしては、開発形態が異なる点が挙げられる。従来のソフトウェア開発形態としてよく知られているのがウォーターフォールモデルである。これは、要求仕様定義・設計・コーディング・テスト・運用保守という流れで開発が進められており、明確なテスト工程が存在していた。一方、OSSの場合においては、例えば、フォールト管理やコード品質分析、ユニットテスト、ビルド問題の分析などを目的とした支援ツールとして、Redmine, Jira,

表 1 Android プロジェクトの最適リリース時刻

Android 1.5 NDK,Release 1

λ_0

8.596096

θ

0.00038

前提条件 c1=1, c3=10, c4=1.0, Tu=500, λF=6.5						前提条件 c1=1, c2=40, c3=10, Tu=500, λF=6.5					
λ0	c3/(c2-c1)	c2	T0	tF	T*	λ0	c3/(c2-c1)	c4	T0	tF	T*
8.596096	20	1.5	104.5333	104.5333	104.5333	8.596096	0.526316	1	116.2022	104.5333	116.2022
8.596096	10	2	104.5333	104.5333	104.5333	8.596096	0.526316	1.5	59.03165	104.5333	104.5333
8.596096	5	3	0.04612	104.5333	104.5333	8.596096	0.526316	2	35.06446	104.5333	104.5333
8.596096	2.5	5	11.56111	104.5333	104.5333	8.596096	0.526316	2.5	22.93487	104.5333	104.5333
8.596096	1.111111	10	30.18238	104.5333	104.5333	8.596096	0.526316	3	16.02387	104.5333	104.5333
8.596096	0.714286	15	47.08825	104.5333	104.5333	8.596096	0.526316	3.5	11.74503	104.5333	104.5333
8.596096	0.526316	20	62.68027	104.5333	104.5333	8.596096	0.526316	4	8.927041	104.5333	104.5333
8.596096	0.416667	25	77.22413	104.5333	104.5333	8.596096	0.526316	4.5	6.980441	104.5333	104.5333
8.596096	0.344828	30	90.90651	104.5333	104.5333	8.596096	0.526316	5	5.583783	104.5333	104.5333
8.596096	0.294118	35	103.8645	104.5333	104.5333	8.596096	0.526316	5.5	4.550313	104.5333	104.5333
8.596096	0.25641	40	116.2022	104.5333	116.2022	8.596096	0.526316	6	3.765799	104.5333	104.5333
8.596096	0.227273	45	128.001	104.5333	128.001						
8.596096	0.204082	50	139.3259	104.5333	139.3259						

YouTrack, TeamCity, 及び Jenkins が知られている。しかしながら、OSS の品質については、従来のテスト工程のように、体系的かつ組織付けて品質を測定・評価するための明確なテスト工程が存在していないのが現状である。

筆者らの研究グループでは、こうした OSS が注目され始めた 2004 年頃から^{(6)・(7)}、オープンソースプロジェクトのバグトラッキングシステム上におけるフォールトデータを適用した様々な研究成果を公表してきた⁽²⁾。本章からは、こうしたオープンソースプロジェクトの下で開発されている OSS に対する信頼性評価と品質改善のための手法として、ソフトウェア信頼性モデルに基づくパラメトリックアプローチと機械学習に基づくノンパラメトリックアプローチについて紹介する。

4. オープンソースソフトウェアに対するモデリングアプローチ

まず、ここではモデリングアプローチとして、OSS に対する確率微分方程式に基づく信頼性評価モデルの適用例とともに、ハザードレートモデルに基づく信頼性評価法について紹介する。本手法により、オープンシステム構築の際における最適な OSS の選択や、メンテナンス時期の予測支援を行うことが期待される。

4.1 クラウドオープンソースソフトウェアに対するジャンプ拡散過程モデル

クラウド OSS のフォールト検出過程を、その値が連続的に変動していく確率過程でモデル化する。まず、時刻 $t = 0$ で OSS の運用が開始され、任意の時刻 t における検出フォールト数を $\{N(t), t \geq 0\}$ と定義する。ここで、クラウドの運用段階におけるフォールト発見事象が、ログインするユーザ数やサービスアプリケーション数の変化などに起因するプロビジョニングプロセスにより不規則に変動するものと仮定すると、クラウド OSS に対するジャンプ拡散過程モデルは以下のように導出される⁽⁸⁾。

$$N(t) = D(t) \left[1 - \exp \left\{ - \int_0^t b(t) ds - \sigma c(t) W(t) + \sum_{i=1}^{M_t(\lambda)} \log V_i \right\} \right]. \quad (11)$$

このとき、 $D(t)$ は要求仕様の変化を考慮した場合における OSS 内に潜在する総フォールト数を表すものとする。また、 $b(t) (> 0)$ は時刻 t におけるフォールト発見率を、 $c(t)$ は運用環境がクラウド OSS に与える影響度合いを表す。更に、 $W(t)$ は Wiener 過程であり、形式的には白色雑音の時間積分 $\int_0^t \gamma(s) ds$ で定義されるものである。 $\sigma (> 0)$ は定数パラメータを表す。 $M_t(\lambda)$ は、 $W(t)$ とは独立な強度パラメータ λ をもつポアソン過程であり、時刻 t までにジャンプが発生した回数を表す。 λ はジャンプ事象が生じる確率的な頻度である。また、 V_i は i 回目のジャンプ幅を表す独立な確率変数である。

本モデルにおいて、フォールト発見率として、以下の $b(t) \equiv b_1(t)$, $b(t) \equiv b_2(t)$ を考える。

$$\int_0^t b_1(s) ds = (1 - \exp[-bt]), \quad (12)$$

$$\int_0^t b_2(s) ds = (1 - (1 + bt)\exp[-bt]), \quad (13)$$

$$c(t) = \exp(-E[p_k]t). \quad (14)$$

ここで、 b はフォールト 1 個当りのフォールト発見率を表す。

クラウド環境が障害なく継続的に利用できる状態かどうかを確認することは、運用段階における保守コストの削減にもつながることから非常に重要となる。継続した利用可能時間の割合を評価するための尺度として稼働率を以下のように定義する。

$$PER_c(t) = \frac{t - MTTR_c \cdot N(t)}{t}. \quad (15)$$

ここで、 $MTTR_c$ はクラウド OSS の平均不動作時間、すなわち平均修復時間を表し、その修復時間は修復率 λ を持つ指数分布に従うものと仮定する。

実際のクラウド OSS のオープンソースプロジェクトである OpenStack⁽⁹⁾ におけるバグトラッキングシステム上に登録されたフォールトデータを適用した数値例を示す。 $b(t) \equiv b_1(t)$ 及び $b(t) \equiv b_2(t)$ に対する推定された稼働率のサンプルパスが図 5 及び図 6 である。図 5 及び図 6 から、推定されたネットワークトラフィックの変化に応じて、稼働率が不規則に変動している様子が確認できる。図 5 及び図 6 は、推定されたパラメータを用いて擬似的に発生させたサンプルパスであるが、可視化することにより、雑音の発生頻度を容易に把握することが可能となる。

図 5 推定された稼働率のサンプルパス ($b(t) \equiv b_1(t)$) の場合)

図 6 推定された稼働率のサンプルパス ($b(t) \equiv b_2(t)$) の場合)

4.2 組込みオープンソースソフトウェアに対するハザードレートモデル

確率変数 X_k ($k = 1, 2, \dots$) により, $(k-1)$ 番目と k 番目のソフトウェア故障発生時間間隔を表すものとする, X_k に対するハザードレートは,

$$z_k(x) = w_k(x)\{N - (k-1)\} \quad (16)$$

$$(k = 1, 2, \dots, \alpha; \phi > 0),$$

$$w_k(x) = \phi e^{-p_k} \quad (17)$$

$$(N > 0, \phi > 0, -1 < p_k < 1),$$

により表すことができるものと仮定する. ここで, 各諸量を次のように定義する.

$z_k(x)$: 組込みソフトウェア全体に対するハザードレート,

$w_k(x)$: ネットワークトラヒックを考慮した固有フォールト 1 個当りのハザードレート,

N : ソフトウェアコンポーネント内に潜在する総固有フォールト数,

ϕ : 固有フォールト 1 個当りのハザードレート,

p_k : $k-1$ に対する k のネットワークトラヒックの変化率.

式 (16) は, 組込みソフトウェア内に潜在する総固有フォールト及びネットワークトラヒックの変化に起因するフォールトによるソフトウェア故障発生現象を, ネットワークトラヒックの変化率を表す p_k により陽に記述するものである. 特に, p_k が負の値の場合には信頼度退化を示し, p_k の値が正の場合には信頼度成長を表す⁽¹⁰⁾.

式 (16) のハザードレートモデルから, X_k の平均値すなわち k 番目のソフトウェア故障に対する平均ソフトウェア故障時間間隔 (Mean Time between Software Failures, 以下 MTBF と略す) は,

$$E[X_k] = \frac{1}{w_k(x)(N - k + 1)}, \quad (18)$$

により与えられる.

特に, 組込み OSS が障害を引き起こすことなく継続的に利用できる状態かどうかをハードウェアとともにシステムテスト工程で確認することは, リリース後における保守コストの削減にもつながることから非常に重要となる. 継続した利用時間を評価するための尺度として稼働率を以下のように定義する.

$$PER_m(l) = \frac{\sum_{k=1}^l E[X_k] - MTTR_m \cdot l}{\sum_{k=1}^l E[X_k]}. \quad (19)$$

ここで, $MTTR_m$ は l 番目のソフトウェア故障発生に対する組込み OSS の不動作時間, すなわち平均修復時間を表す. 平均修復時間 $MTTR_m$ は修復率 λ を持つ指数分布に従うものと仮定することにより, 稼働率を以下のように与えることができる.

$$PER_m(l) = \frac{\sum_{k=1}^l E[X_k] - \frac{l}{\lambda}}{\sum_{k=1}^l E[X_k]}. \quad (20)$$

一例として, 携帯電話用 OS として開発・公開されている Android⁽¹¹⁾ 上で BusyBox⁽¹²⁾ が動作するシステムを構築する環境を考える. テスト工程を想定するために, 実際の Android 及び BusyBox のオープンソースプロジェクトにおけるバグトラッキングシステム上に登録されたフォールトデータを適用した数値例を示す. Android はスマートフォン用 OS として知られ, BusyBox はテレビ, オーディオ, ブロードバンドルータ, 小形サーバなど, 家電製品を代表とした様々な組込み製品に利用されている.

推定された MTBF を図 7 に示す. 図 7 から, フォールトが発見されるたびに MTBF の値が増加していく, すなわち信頼度成長が起こっている様子が確認できる. 特に, ネットワークトラヒックが MTBF に与える影響は大きいと言える. 更に, 推定された稼働率を図 8 に示す. 図 8 から, 稼働率が, ネットワークトラヒックの影響を受けつつ増加する様子が確認できる.

図 7 推定された MTBF

図 8 推定された稼働率

5. オープンソースソフトウェアに対するディープラーニングに基づくフォールト識別法

バグトラッキングシステム上には、様々なフォールト情報が登録されている。登録されたフォールトの重要度を自動で識別することが可能となれば、OSS の品質改善に役立つものとする。本章では、バグトラッキングシステム上のデータをそのまま利用し、フォールト重要度に影響を与える要因を考慮せず識別可能なアプローチとして、ディープラーニングに基づく手法について平易に紹介する。本章におけるフィードフォワードニューラルネットワークによるディープラーニングの構造を図 9 に示す。図 9 において、 $z_l (l = 1, 2, \dots, L)$ 及び $z_m (m = 1, 2, \dots, M)$ は特徴抽出のための結合ユニットを意味する。また、 $o_n (n = 1, 2, \dots, N)$ は、圧縮された特徴量を表す。ディープラーニングに関しては、幾つかのアルゴリズムや応用事例が提案されているが^{(13), (14)}、ここでは、オープンソースプロジェクトにおけるバグトラッキングシステム上におけるフォールトデータを分析するために、ディープニューラルネットワークを採用する。

特徴抽出のための結合ユニットにおけるパラメータに対する情報量として以下のようなデータを適用する。また、目的変数に該当するフォールトレベルの種類を表 2 に示す。このとき、8 種類の説明変数に対して、二つの目的変数が最終的な特徴量として抽出されるように学習され、各入力データは出現頻度のような数値データに変換される⁽¹⁵⁾。

- バグトラッキングシステム上へ登録された日付

図 9 ディープラーニングの構造

表 2 バグトラッキングシステム上に登録されたフォールトレベル

Index Number	Fault Level
1	Trivial
1	Enhancement
1	Minor
1	Normal
1	Blocker
2	Major
2	Critical

表 3 90%の学習データを利用した場合におけるニューラルネットワークとディープラーニングに基づく推定結果に対する認識率の比較結果

Estimation Method	Recognition Rate
Neural Network	20.120%
Deep Learning	79.479%

- ソフトウェアプロダクト名
- ソフトウェアコンポーネント名
- ソフトウェアバージョン名及びバージョン番号
- フォールト報告者のニックネーム
- フォールト修正者のニックネーム
- フォールトの状態
- オペレーティングシステム

認識率の推定結果として、90%の学習データを利用して、ニューラルネットワークとディープラーニングに基づく推定結果に対する表 2 に示したフォールトレベルの認識率の比較結果を表 3 に示す。表 3 から、ニューラルネットワークよりもディープラーニングに基づくフォールト識別法の方が認識率が高いことが確認できる。

特に、ディープラーニングに基づく手法は、その特性値を事前に抽出する必要がないため、バグトラッキングシステム上へ組み込むことにより、現存するデータをそのまま利用して認識することが可能となる。

6. 2 変量ソフトウェア信頼性モデル

ソフトウェア故障は、フォールトが作り込まれたプログラムパスをテスト用の入出力データであるテストケースによって実行されない限り観測されず、作り込まれたフォールトの発見・除去には至らないソフトウェア特有の故障メカニズムがある。ソフトウェア信頼度成長モデリングでは、テストの実施に伴い観測されるソフトウェア故障発生時間間隔やソフトウェア故障発生数の確率的な性質に基づきながら、テスト時間だけに依存した信頼度成長過程を数理的に記述する場合がほとんどである。しかしながら、上述した故障発生メカニズムを陽に反映し、より現実的なソフトウェア信頼度成長過程を記述するためには、従来のテスト時間要因だけでなく、テスト網羅度などのテスト労力要因も同時に考慮した数理モデルの開発が必要である^{(16)~(18)}。

本章では、従来のソフトウェア信頼度成長要因としての時間要因を、テスト時間要因と、観測可能なテスト労力要因に分けながら、それらに同時に依存したソフトウェア信頼度成長過程を記述する二つのタイプの2変量ソフトウェア信頼性モデルについて議論する。

6.1 ソフトウェア故障発生現象の記述

ある時刻までに発生した総ソフトウェア故障数に着目する場合、その確率的現象の時間的変化を考慮しながら、計数過程（確率過程）を用いて、ソフトウェア故障発生現象を記述する。計数過程としては、実データへの適合性やモデル構造の単純性からNHPPが経験的に広く適用されている。ここでは、ソフトウェア信頼度成長要因をテスト時間要因とテスト労力要因から成る2変量ソフトウェア信頼性モデルを構築するにあたり、以下の仮定を設ける。

- (A1) ソフトウェア故障が観測された場合、直ちにその原因となるフォールトは修正・除去される。
- (A2) ソフトウェア故障は独立かつ同一の2変量確率分布 $F(s, u) = \Pr\{S \leq s, U \leq u\}$ に従って観測される。
- (A3) テスト開始前に潜在する総フォールト数は非負の離散型確率変数 N_0 で表される。

ここで、 S 及び U は、それぞれ、テスト時間及びそれまでに費やしたテスト労力を表す確率変数である。いま、 $\{N(s, u), s \geq 0, u \geq 0\}$ を、テスト時間 s 及びそれまでに費やしたテスト労力 u によって観測された総発見フォールト数を表す二次元確率過程とすると、上記の仮定から、

$$\begin{aligned} \Pr\{N(s, u) = m \mid N_0 = n\} \\ = \binom{n}{m} \{F(s, u)\}^m \{1 - F(s, u)\}^{n-m}, \end{aligned} \quad (21)$$

を得る。式 (21) から、二次元確率過程 $\{N(s, u), s \geq 0, u \geq 0\}$ は、

$$\begin{aligned} \Pr\{N(s, u) = m\} \\ = \sum_n \{F(s, u)\}^m \{1 - F(s, u)\}^{n-m} \Pr\{N_0 = n\} \\ (m = 0, 1, 2, \dots), \end{aligned} \quad (22)$$

として与えられる。例えば、初期潜在フォールト数 N_0 がパラメータ $\omega (> 0)$ のポアソン分布に従う場合、式 (22) は、

$$\Pr\{N(s, u) = m\} = \frac{\{\omega F(s, u)\}^m}{m!} \exp[-\omega F(s, u)], \quad (23)$$

と整理され、平均値関数が $H(s, u) \equiv \omega F(s, u)$ である二次元NHPPと本質的に等価となる。

6.2 具体的なモデルの構築 (1)

一つ目のモデル（以降、「モデル1」と呼ぶ）では、ソフトウェア複雑性と信頼性との関係を考慮した2変量モデルについて議論する。考慮する複雑性尺度として比較的容易に観測できるコード行数を取り上げ、初期潜在フォールト数 N_0 に対して以下の仮定を与える。

- (B1) 評価対象とするソフトウェアは K コード行数 (LOC) で構成される。
- (B2) それぞれのコードには確率 λ でフォールトが作り込まれている。
- (B3) 作り込まれたフォールトに起因するソフトウェア故障はそれぞれ独立に発生する。

上記の仮定から、初期潜在フォールト数 N_0 の確率関数を、

$$\begin{aligned} \Pr\{N_0 = n\} = \binom{K}{n} \lambda^n (1 - \lambda)^{K-n} \\ (0 < \lambda < 1; n = 0, 1, 2, \dots, K), \end{aligned} \quad (24)$$

と与える。このとき、二次元確率過程 $\{N(s, u), s \geq 0, u \geq 0\}$ は、式 (22) から、

$$\begin{aligned} \Pr\{N(s, u) = m\} \\ = \binom{K}{m} \{\lambda F(s, u)\}^m \{1 - \lambda F(s, u)\}^{K-m}, \end{aligned} \quad (25)$$

に従うことになる。式 (25) から、ソフトウェア信頼性の定量的な評価に有用な種々の信頼性評価尺度を導出することができる。例えば、 $N(s, u)$ の期待値 $E[N(s, u)]$ は、定義から、

$$E[N(s, u)] = K \lambda F(s, u), \quad (26)$$

と導出できる。また、ソフトウェアの信頼性を定量的に与えるソフトウェア運用信頼度⁽¹⁶⁾は、テストが時刻 s_e まで実施され、それまでに費やしたテスト労力が u_e であるとき、その後の時間区間 $(s_e, s_e + \eta) (s_e \geq 0, \eta \geq 0)$ において、ソフトウェア故障が発生しない確率として定義され、その信頼度関数は、

$$R(\eta \mid s_e, u_e)$$

$$\begin{aligned}
&= \sum_k \Pr\{N(s_e + \eta, u_e) = k \mid N(s_e, u_e) = k\} \\
&\quad \Pr\{N(s_e, u_e) = k\} \\
&= [1 - \lambda\{F(s_e + \eta, u_e) - F(s_e, u_e)\}]^K, \quad (27)
\end{aligned}$$

と求められる。

6.3 具体的なモデルの構築 (2)

二つ目のモデル (以降, 「モデル 2」と呼ぶ) では, 従来のソフトウェア信頼度成長要因としてのテスト時間を, コブ・ダグラス型効用関数を用いて,

$$t \equiv s^\alpha u^{1-\alpha} \quad (0 \leq \alpha \leq 1), \quad (28)$$

と拡張するアプローチを考える。ここで, α はソフトウェア信頼度成長過程に与える影響度を表すパラメータである。1 変量ソフトウェア信頼性モデルの中でも比較的単純なモデルの一つであるワイブル型モデル⁽¹⁹⁾に適用した場合, 式 (23) の 2 変量 NHPP の平均値関数は, 次のように与えられる。

$$\begin{aligned}
H(s, u) \equiv \gamma(s, u) &= \left(\frac{s^\alpha u^{1-\alpha}}{\rho} \right)^\beta \\
&\quad (0 < \beta < 1, \rho > 0). \quad (29)
\end{aligned}$$

式 (29) において, β は信頼度成長パラメータ, ρ は尺度パラメータである。式 (23) 及び式 (29) から, ソフトウェア信頼性評価尺度としての $N(s, u)$ の期待値 $E[N(s, u)]$ は, 二次元 NHPP の性質から,

$$E[N(s, u)] = \gamma(s, u), \quad (30)$$

と求められ, また, 運用ソフトウェア信頼度関数は,

$$R(\eta \mid s_e, u_e) = \exp[-\{\gamma(s_e + \eta, u_e) - \gamma(s_e, u_e)\}], \quad (31)$$

と求められる。

6.4 モデルパラメータの推定

実際のテスト工程から N 組のフォールト発見数データ: $(s_i, u_i, y_i) (i = 0, 1, 2, \dots, N; s_0 = 0 < s_1 < s_2 < \dots < s_N; u_0 = 0 < u_1 < u_2 < \dots < u_N)$ が得られたものとする。ここで, y_i はテスト時刻 s_i までにテスト労力 u_i を費やして発見された総フォールト数を表す。

モデル 1 では, 式 (25) の確率的性質に基づき, 最尤法を用いて推定する。上記のフォールト発見数データに対する尤度関数は, マルコフ性から,

$$\begin{aligned}
l &= \Pr\{N(s_1, u_1) = y_1, N(s_2, u_2) = y_2, \\
&\quad \dots, N(s_N, u_N) = y_N\} \\
&= \prod_{i=1}^N \Pr\{N(s_i, u_i) = y_i \mid N(s_{i-1}, u_{i-1}) = y_{i-1}\}
\end{aligned}$$

$$\begin{aligned}
&\cdot \Pr\{N(s_1, u_1) = y_1\} \\
&= \prod_{i=1}^N \left(\frac{K - y_{i-1}}{y_i - y_{i-1}} \right) \{z(\Phi_{i-1}, \Phi_i)\}^{y_i - y_{i-1}} \\
&\quad \{1 - z(\Phi_{i-1}, \Phi_i)\}^{K - y_i}, \quad (32)
\end{aligned}$$

と導出される。ここで,

$$z(\Phi_{i-1}, \Phi_i) = \frac{\lambda\{F(s_i, u_i) - F(s_{i-1}, u_{i-1})\}}{1 - \lambda F(s_{i-1}, u_{i-1})}, \quad (33)$$

である。式 (32) 及び式 (33) から, 対数尤度関数を導出し, 各モデルパラメータに関する同次方程式を数値的に求めることで, 最尤推定値が得られる。

モデル 2 では, 式 (29) の構造に着目し, 最小二乗法を用いたアプローチを試みる。式 (29) の両辺に自然対数をとると,

$$\log \gamma(s, u) = -\beta \log \rho + \alpha \beta \log s + (1 - \alpha) \beta \log u, \quad (34)$$

となり, 重回帰式:

$$Y_i = a_0 + a_1 K_i + a_2 L_i + \epsilon_i, \quad (35)$$

を得る。ここで, 式 (35) において, それぞれ,

$$\begin{cases} Y_i = \log y_i, \\ K_i = \log s_i, \\ L_i = \log u_i, \\ a_0 = -\beta \log \rho, \\ a_1 = \alpha \beta, \\ a_2 = (1 - \alpha) \beta, \end{cases} \quad (36)$$

であり, ϵ_i は等分散性を仮定した実測データと回帰直線との残差を表す誤差項である。これにより, 偏差二乗和:

$$\begin{aligned}
S(a_0, a_1, a_2) &= \sum_{i=1}^N \epsilon_i^2 \\
&= \sum_{i=1}^N \{Y_i - (a_0 + a_1 K_i + a_2 L_i)\}^2, \quad (37)
\end{aligned}$$

から, 各パラメータに対する同次連立方程式を解くことで, 最終的に求めたいパラメータは, それぞれ,

$$\begin{cases} \hat{\alpha} = \frac{\hat{a}_1}{\hat{a}_1 + \hat{a}_2}, \\ \hat{\beta} = \hat{a}_1 + \hat{a}_2, \\ \hat{\rho} = \exp \left[-\frac{\hat{a}_0}{\hat{a}_1 + \hat{a}_2} \right], \end{cases} \quad (38)$$

のように推定される。

6.5 実際データによる適合性評価

実測データを用いて, 今回議論したモデルに基づいた信頼性解析例及び従来モデルとの適合性比較結果を示す。適用したデータは, それぞれ, DS1 と呼ぶ 24 組のデータ: $(s_k, u_k, y_k) (k = 1, 2, \dots, 24; s_{24} = 24 \text{ (週)}, u_{24} =$

図 10 推定された総期待フォールト発見数 $\hat{\gamma}(s, u)$ の挙動 (モデル 2, DS1)

表 4 MSE に基づいた適合性比較

	DS1	DS2
OUR MODEL 2	54.7535	75.0829
OUR MODEL 1	1.96356	3.09947
ONE-D WEIBULL	1425.24*	205.688*
TCD	111.838	49.9692
EXPO	603.198	73.0885
DELAYED-S	150.005	36.6406

0.9095, $y_{24} = 296$, $K = 1.972 \times 10^5$ (LOC)) 及び DS2 と呼ぶ 22 組のデータ: (s_k, u_k, y_k) ($k = 1, 2, \dots, 22$; $s_{22} = 22$ (週), $u_{22} = 0.9198$, $y_{24} = 212$, $K = 1.630 \times 10^5$ (LOC)) であり, テスト時間要因として週 (カレンダー時間), テスト労力要因として C0 (命令網羅) カバレッジを適用する.

図 10 に, 式 (29) のモデル 2 に基づき, DS1 に対して推定された総期待フォールト発見数 $\hat{\gamma}(s, u)$ の挙動を表す. 図 10 において, 点線はテスト時間とテスト労力要因から構成される二次元信頼度成長要因の下で観測された累積フォールト発見数の挙動を表し, 曲面は推定されたモデルの挙動を示す. 図 10 から, たとえ長い時間テストを実施したとしても, テスト網羅度が上昇しない限りソフトウェア故障は追加的に観測されにくく, 除去されるフォールト数はほとんど増加しない状況や, 一方, 短いテスト時間において高いテスト網羅度を効率的に達成できた場合, それに応じてソフトウェア内に潜在するフォールトが多く検出できる状況など, 1 変数モデルでは表現することができなかったより現実的なソフトウェア信頼度成長過程を描いていることが分かる.

次に, 従来の代表的な数種類の 1 変数 SRGM モデルとの適合性比較を行う. 比較対象とするモデルは, 1 変数ワイブル型 (ONE-D WEIBULL), テスト網羅度依存型 (TCD), 指数形 (EXPO), 及び遅延 S 字形 (DELAYED-S) ソフトウェア信頼度成長モデル^{(1), (19)}を取り上げる. また, 適合性比較では, 適用した実データとの適合性を検証するために平均偏差二乗和 (MSE), 更に, フォールト発見数の予測妥当性を検証するために予測相対誤差 (PRE) に基づいて検証する. 表 4 に MSE に基づいた適合性比較結果を示す. MSE は, 小さい値を示すモデルの方が, 実データとの適合性が高いモデルと判定でき, 今回提案したモデルの有効性が伺える. なお, 表中の*印は, 理論的に適切なパラメータ推定値が得られてない状況下での結果であ

ることを示す. 特に, モデル 2 は高い適合性を示している. また, PRE に関する適合性比較結果は, 紙面の都合上割愛するが, モデル 2 は, テスト終了時におけるフォールト発見数をテスト早期において, 他のモデルよりも比較的高い精度で予測できることも検証されている⁽¹⁷⁾. ただし, 一般的に, モデルパラメータの数が多いモデルが MSE や PRE が良くなることが知られているが, この点の検証については, パラメータの推定に用いた確率的性質がモデルによって異なるため, 今後更なる検証が必要であることも付記しておく.

7. チェンジポイントを考慮したソフトウェア信頼性モデル

多くのソフトウェア信頼性モデルは, ソフトウェア故障発生現象やソフトウェアフォールト発見事象に関する確率・統計的性質はテスト期間を通じて本質的に変化しないとの基本的仮定に基づいて構築されてきた. しかし, 実際のテスト工程では, 開発管理面の要因 (納期までに目標とする信頼度を達成するための活動など) や固有技術面からの要因 (フォールト発見難易度, フォールトの独立性, モジュール毎のフォールト密度の違い, フォールト検出ターゲットの変更など) の変化によって, 観測されるソフトウェア信頼度成長過程の時間的挙動も著しく変化することが報告されている⁽²⁰⁾. また, このような変化が観測される時刻はチェンジポイント (変化点) と呼ばれ, この影響を考慮しながら, 実際に観測されるソフトウェア信頼度成長過程を記述することで, 信頼性評価や予測精度の向上が大いに期待できる^{(21)~(25)}. 本章では, ソフトウェア信頼性モデルの中でも NHPP に基づいて, チェンジポイントの影響を考慮した場合の信頼性モデリング手法⁽²⁶⁾について議論する.

7.1 NHPP モデル構築枠組み

テスト工程において検出可能なフォールト数が有限の場合を仮定したモデルの多くは, 以下の三つの基本的仮定に基づいて構築することができる.

- (B1) ソフトウェア故障が発生した場合, その原因となるフォールトは, 直ちにかつ完全に修正・除去される.
- (B2) 各ソフトウェア故障は, それぞれ, 独立かつ時間に関してランダムに発生して, 発生時刻は, それぞれ, 同一の確率分布 $F(t) \equiv \Pr\{T \leq t\} = \int_0^t f(x)dx$ に従う非負の確率変数によって与えられる. ここで, $f(t)$ は確率密度関数を表す.
- (B3) テスト開始前にソフトウェア内に潜在する総フォールト数 (初期潜在フォールト数) $N_0(> 0)$ は, ある確率分布に従う確率変数とする.

いま, $\{N(t), t \geq 0\}$ を任意のテスト時刻 t までに発見された総フォールト数を表す確率過程とする. このとき, (B1) ~ (B3) から, テスト時刻 t までに m 個のフォールトが発見される確率は, 次のように定式化される.

$$\Pr\{N(t) = m\}$$

図 11 チェンジポイントを考慮したソフトウェア故障発生現象に関する確率諸量

$$= \sum_n \binom{n}{m} \{F(t)\}^m \{1 - F(t)\}^{n-m} \Pr\{N_0 = n\} \quad (m = 0, 1, 2, \dots). \quad (39)$$

よく知られた結果として、(B3) で定義した初期潜在フォールト数 N_0 が平均 $\omega (> 0)$ のポアソン分布に従うと仮定した場合、式 (39) は、

$$\begin{aligned} \Pr\{N(t) = m\} &= \sum_n \binom{n}{m} \{F(t)\}^m \{1 - F(t)\}^{n-m} \frac{\omega^n}{n!} \exp[-\omega] \\ &= \exp[-\omega] \frac{\{\omega F(t)\}^m}{m!} \sum_n \frac{\{\omega(1 - F(t))\}^{n-m}}{(n-m)!} \\ &= \frac{\{\omega F(t)\}^m}{m!} \exp[-\omega F(t)] \quad (m = 0, 1, 2, \dots), \end{aligned} \quad (40)$$

となり、平均値関数 $E[N(t)] = \omega F(t)$ を持つ NHPP と本質的に等価となる。したがって、当該モデリング枠組みに基づいて、ソフトウェア故障発生現象若しくはフォールト発見事象が NHPP に従うモデルを構築する場合、式 (40) から、ある適切なソフトウェア故障発生時間分布 $F(t)$ を与えることで具体的に NHPP モデルを構築することができる。

7.2 チェンジポイントモデリング

前述した基本的仮定 (B2) を拡張しながら、ソフトウェア信頼度成長過程に影響を与える要因の変化が複数回観測される場合を想定したモデルの構築枠組みを議論する。まず、図 11 に、本研究において取り扱うソフトウェア故障発生現象に関する確率諸量を定義する。図 11 では、チェンジポイント発生ごとにテスト時間を区切り、 $i (i = 1, 2, \dots)$ 番目のテスト時間区間における $j (j = 1, 2, \dots, n_i)$ 番目のソフトウェア故障発生時刻を $X_{i,j}$ として、ある一般分布 $\Pr\{X_{i,j} \leq t\} = K_{i-1}(t)$ に従うものと仮定する。また、 $i (i = 1, 2, \dots)$ 番目のテスト時間区間における $j - 1$ 番目から j 番目のソフトウェア故障発生時間間隔を $S_{i,j}$ と定義する。なお、チェンジポイント前後における上述した確率諸量は、ある関係があるものと仮定して、これらをそれぞれ次のように与える。

$$X_{i+1,j} = \alpha_i(X_{i,j}), \quad (41)$$

$$S_{i+1,j} = \alpha_i(S_{i,j}), \quad (42)$$

$$K_{i+1}(t) = K_i(\alpha_i^{-1}(t)). \quad (43)$$

式 (41)–式 (43) において、 $\alpha_i(\cdot)$ は i 番目のチェンジポイント発生前後の確率諸量の変化を関係付ける関数であり、これをテスト環境関数 (testing-environment function) と呼ぶことにする。特に、本研究では、次のような関数を一例として考える⁽²⁷⁾: $\alpha_i(t) = \alpha_i t$ ($t \geq 0, \alpha_i > 0$)。ここで、 $\alpha_i (> 0) (i = 1, 2, \dots)$ は i 番目のチェンジポイントに関するテスト環境係数であり、ソフトウェア信頼度成長過程に影響を与える要因の変化がソフトウェア故障発生現象に関する確率諸量に与える相対的な影響度を表す。なお、この関数は、 i 番目のチェンジポイント発生によるソフトウェア故障発生現象の変化を定量的に把握する上で単純かつ有用な関数であると考えられる。

まず、1 番目のチェンジポイント発生に関して NHPP の平均値関数を導出する。いま、テスト開始から 1 番目のチェンジポイント発生までに n_1 個のソフトウェア故障が観測され、それらのソフトウェア故障発生時刻が $0 < x_{1,1} < x_{1,2} < \dots < x_{1,n_1} \leq \tau_1$ であるとする。なお、 τ_1 は 1 番目のチェンジポイントを表す。このとき、1 番目のチェンジポイント発生後 1 番目のソフトウェア故障が発生するまでの時間を表す確率変数 $X_{2,1}$ (若しくは $S_{2,1}$) の確率分布: $K_1(t) \equiv \Pr\{S_{2,1} \leq t\}$ の余関数: $\bar{K}_1(t) \equiv 1 - K_1(t)$ は次のように与えることができる。

$$\begin{aligned} \bar{K}_1(t) &\equiv \Pr\{S_{2,1} > t\} \\ &= \frac{\Pr\{S_{1,n_1+1} > \tau_1 - x_{1,n_1} + t/\alpha_1\}}{\Pr\{S_{1,n_1+1} > \tau_1 - x_{1,n_1}\}} \\ &= \frac{\exp[-\{M_1(\tau_1 + t/\alpha_1) - M_1(x_{1,n_1})\}]}{\exp[-\{M_1(\tau_1) - M_1(x_{1,n_1})\}]} \quad (44) \end{aligned}$$

ここで、 $M_1(t) (\equiv \omega K_0(t))$ は、テスト開始から i 番目のチェンジポイントまでの任意のテスト時間 $t \in (0, \tau_1]$ までに発見された総期待フォールト数若しくは NHPP の平均値関数を表す。式 (44) より、1 番目のチェンジポイント後の任意のテスト時刻 $t \in (\tau_1, \tau_2]$ までに発見される総期待フォールト数は、

$$\begin{aligned} M_2(t) &= -\log \Pr\{S_{2,1} > t - \tau_1\} \\ &= -\log \bar{K}_1(t - \tau_1) \\ &= M_1\left(\tau_1 + \frac{t - \tau_1}{\alpha_1}\right) - M_1(\tau_1), \end{aligned} \quad (45)$$

のように導出できる。同様に、 $i (i \geq 2)$ 番目のチェンジポイント後から 1 番目のソフトウェア故障発生時間 $S_{i,1}$ の確率分

布関数 $K_i(t)$ は、最終的に次のように導出することができる。

$$\begin{aligned} & \Pr\{S_{i,1} > t\} \\ &= \exp \left[- \left\{ M_1 \left(\frac{\sum_{k=1}^i A_{k,i} + t}{A_i} \right) \right. \right. \\ & \quad \left. \left. - M_1 \left(\frac{\sum_{k=1}^{i-1} A_{k,i} + (x_{i,n} - \tau_{i-1})}{A_{i-1}} \right) \right\} \right] \\ & \quad / \left[\exp \left[- \left\{ M_1 \left(\frac{\sum_{k=1}^{i-1} A_{k,i-1} + (\tau_i - \tau_{i-1})}{A_{i-1}} \right) \right. \right. \right. \\ & \quad \left. \left. - M_1 \left(\frac{\sum_{k=1}^{i-1} A_{k,i-1} + (x_{i,n} - \tau_{i-1})}{A_{i-1}} \right) \right\} \right] \right]. \quad (46) \end{aligned}$$

ここで、

$$\begin{cases} A_{k,i} = \alpha_k \cdot \alpha_{k+1} \cdots \alpha_i (\tau_k - \tau_{k-1}), \\ A_i = \alpha_1 \cdot \alpha_2 \cdots \alpha_i, \end{cases} \quad (47)$$

である。これにより、時間区間 $(\tau_i, t]$ において発見された総期待フォールト数は、

$$\begin{aligned} M_{i+1}(t) &= M_1 \left(\frac{\sum_{k=1}^{i+1} A_{k,i} + (t - \tau_i)}{A_i} \right) \\ & \quad - M_1 \left(\frac{\sum_{k=1}^{i-1} A_{k,i-1} + (\tau_i - \tau_{i-1})}{A_{i-1}} \right), \end{aligned} \quad (48)$$

のように導出できる。

以上の議論から、テスト時刻 t までにチェンジポイントが $i (i = 1, 2, \dots)$ 回発生した場合を考慮した総期待発見フォールト数 $\Lambda(t)$ は最終的に、

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = M_1(t) & (0 \leq t \leq \tau_1), \\ \Lambda_2(t) = M_1(\tau_1) + M_2(t) & (t > \tau_1), \\ \vdots \\ \Lambda_i(t) = M_1(\tau_1) + M_2(\tau_2) + \cdots \\ \quad \cdots + M_{i-1}(\tau_{i-1}) + M_i(t) \\ \quad = M_1 \left(\frac{\sum_{k=1}^i A_{k,i-1} + (t - \tau_{i-1})}{A_{i-1}} \right) \\ \quad (t > \tau_{i-1}). \end{cases} \quad (49)$$

と求められる。式 (49) から、複数回のチェンジポイントを考慮した NHPP モデルの平均値関数は、1 番目のチェンジポイントにおけるソフトウェア故障発生時間分布 $K_0(t)$ 若しくは NHPP の平均値関数 $M_1(t)$ を適切に与えることによって構築

できる。

7.3 ソフトウェア信頼性解析例

実際に観測されたデータ： $(t_i, y_i) (i = 1, 2, \dots, 26; t_{26} = 26$ (週), $y_{26} = 34; \tau_1 = 17, \tau_2 = 21)$ を用いながら、今回のモデリング枠組みに基づいたソフトウェア信頼性解析例を示す。このデータは、ある Windows バージョンソフトウェアのテスト工程において得られたフォールト発見数データであり、観測された 2 回のチェンジポイントはテスト人員の変更とテスト技術者の変更を行った時刻である。

本章で議論したモデリング枠組みを用いてモデルを構築するためには、1 番目のチェンジポイント発生前におけるソフトウェア故障発生時間分布を適切に仮定する必要がある。例えば、1 番目のチェンジポイント発生前にソフトウェア故障発生時間分布がパラメータ $(2, b)$ のガンマ分布に従うと仮定したとき、1 番目のチェンジポイント発生前における NHPP の平均値関数は遅延 S 字形 SRGM： $\Lambda_1(t) = \omega \{1 - (1 + bt) \exp[-bt]\}$ ($a > 0, b > 0$)⁽¹⁾ に従い、2 回のチェンジポイントの影響を考慮した場合の平均値関数は、式 (49) から、

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega \{1 - (1 + bt) \exp[-bt]\} & (0 < t \leq \tau_1), \\ \Lambda_2(t) = \omega \{1 - (1 + b(\tau_1 + \frac{t - \tau_1}{\alpha_1})) \exp[-b(\tau_1 + \frac{t - \tau_1}{\alpha_1})]\} & (\tau_1 < t \leq \tau_2), \\ \Lambda_3(t) = \omega \{1 - (1 + b(\tau_1 + \frac{\alpha_2(\tau_2 - \tau_1) + (t - \tau_2)}{\alpha_1 \alpha_2})) \exp[-b(\tau_1 + \frac{\alpha_2(\tau_2 - \tau_1) + (t - \tau_2)}{\alpha_1 \alpha_2})]\} & (\tau_2 < t), \end{cases} \quad (50)$$

と導出される。式 (50) に含まれるパラメータを最尤法に基づいて推定できる。図 12 に、推定された式 (50) の平均値関数を示す。ここで、 $\hat{\omega} = 34.038$, $\hat{b} = 0.172$, $\hat{\alpha}_1 = 0.669$, 及び $\hat{\alpha}_2 = 0.249$ であった。図 12 から、推定された総期待発見

図 12 推定された平均値関数と 90%信頼限界 ($\tau_1 = 17, \tau_2 = 21$, and $\Lambda_1(t) = \omega \{1 - (1 + bt) \exp[-bt]\}$).

フォールト数の時間的挙動がチェンジポイントで実測データの挙動に沿いながら変化していることが分かる。また、推定されたテスト環境係数から、チェンジポイント前より後の方がソフトウェア故障発生時間間隔は相対的に狭くなっていることも分かる。なお、期待残存フォールト数やソフトウェア信頼度関数⁽¹⁾など、信頼性評価に有用な信頼性評価尺度も図 12 から直ちに推定できる。また、今回適用したデータを含めた数組の実測データを適用して、チェンジポイントの影響を考慮していない従来モデルとの適合性比較を行った結果、議論したモデルの実測データに対する適合性が向上したことも確認できた⁽²⁶⁾。

8. おわりに

本解説では、品質指向ソフトウェアマネジメントの立場から、実際のソフトウェアプロジェクトに対する実用性を考慮して、ソフトウェア信頼性工学における計測・評価技術とマネジメント技術に関する四つのトピックスを具体的適用例を挙げながら議論した。近年、ソフトウェア開発技術の進展と開発形態の変革が著しく、それにふさわしいソフトウェア工学 (software engineering) の分野でのソフトウェア生産技術、及びソフトウェア信頼性工学の中でもソフトウェア信頼性モデルに関する研究成果も数多く公表されている。

特に、ソフトウェア信頼性モデリングについては、開発プロダクトとしての成果物についてはもちろんのこと、開発プロセスあるいはフィールドにおける品質/信頼性要因を的確に取り込んで、より適合性・妥当性の高いソフトウェア信頼性モデルに改善したり、新たに構築したりする必要があるだろう。

文 献

- (1) 山田 茂, 福島利彦, 品質指向ソフトウェアマネジメント ~ 高品質ソフトウェア開発のためのプロジェクトマネジメント ~, 森北出版, 東京, 2007.
- (2) S. Yamada and Y. Tamura, OSS Reliability Measurement and Assessment, Springer International Publishing, Switzerland, 2016.
- (3) 山田 茂, ソフトウェア信頼性の基礎 — モデリングアプローチ, 共立出版, 東京, 2011.
- (4) S. Yamada, Software Reliability Modeling: Fundamentals and Applications, Springer-Verlag, Tokyo/Heidelberg, 2014.
- (5) K. Okumoto, “A statistical method for software quality control,” IEEE Trans. Softw. Eng., vol. SE-11, no. 12, pp. 1424–1430, 1985.
- (6) 田村慶信, 山田 茂, 木村光宏, “オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察,” 信学論 (A), vol. J88-A, no. 7, pp. 840–847, July 2005.
- (7) Y. Tamura and S. Yamada, “Comparison of software reliability assessment methods for open source software,” Proc. 11th International Conference on Parallel and Distributed Systems, vol. II, pp. 488–492, Fukuoka, Japan, July 20–22, 2005.
- (8) Y. Tamura and S. Yamada, “Reliability computing and management considering the network traffic for a cloud computing,” Ann. Oper. Res., Springer US, vol. 244, no. 1, pp. 163–176, Feb. 2016.
- (9) The OpenStack project, “OpenStack,” <https://www.openstack.org/>
- (10) Y. Tamura and S. Yamada, “AIR application for reliability analysis considering debugging process and network traffic in mobile clouds,” Journal of Simulation Modelling Practice and Theory, vol. 50, pp. 165–175, Jan. 2015.
- (11) Open Handset Alliance, “Android open source project,” <https://source.android.com/>
- (12) E. Andersen, BUSYBOX, <https://busybox.net/>
- (13) H.P. Martinez, Y. Bengio, and G.N. Yannakakis, “Learning deep physiological models of affect,” IEEE Computational Intelligence Magazine, vol. 8, no. 2, pp. 20–33, 2013.
- (14) C. Szegedy, W. Liu, Y. Jia, et al., “Going deeper with convolution,” Proc. 2015 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9, 2015.
- (15) Y. Tamura and S. Yamada, “Software reliability analysis considering the fault detection trends for big data on cloud computing,” Industrial Engineering Management Science and Applications 2015, Lect. Notes in Electrical Engineering, vol. 349, pp. 1021–1030, Springer-Verlag Berlin Heidelberg, 2015.
- (16) 石井智隆, 土肥 正, “二次元 NHPP に基づいたテスト労力依存型ソフトウェア信頼性モデル,” 信学論 (D), vol. J89-D, no. 8, pp. 1684–1694, Aug. 2006.
- (17) 井上真二, 山田 茂, “2 変量ワイブル型ソフトウェア信頼度成長モデルとその適合性評価,” 情処学論, vol. 49, no. 8, pp. 2851–2861, May 2008.
- (18) S. Inoue and S. Yamada, “Two-dimensional software reliability growth models,” in Reliability Modeling with Applications, S. Nakamura, C.H. Qian, and M. Chen, eds, pp. 179–194, World Scientific, Singapore, Feb. 2014.
- (19) L.H. Crow, “On tracking reliability growth,” Proc. 1975 Annual Reliability and Maintainability Symposium, pp. 438–443, 1975.
- (20) M. Zhao, “Change-point problems in software and hardware reliability,” Commun. Stat. — Theory Method, vol. 22, no. 3, pp. 757–768, 1993.
- (21) C.Y. Huang, “Performance analysis of software reliability growth models with testing-effort and change-point,” J. Syst. Softw., vol. 76, no. 2, pp. 181–194, 2005.
- (22) C.Y. Huang, “Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency,” J. Syst. Softw., vol. 77, no. 2, pp. 139–155, 2005.
- (23) C.Y. Huang and M.R. Lyu, “Estimation and analysis of some generalized multiple change-point software reliability models,” IEEE Trans. Reliab., vol. 60, no. 2, pp. 498–514, 2011.
- (24) J. Zhao, H.W. Liu, G. Cui, and X.Z. Yang, “Software reliability growth model with change-point and environmental function,” J. Syst. Softw., vol. 79, no. 11, pp. 1578–1587, 2006.
- (25) F.Z. Zou, “A change-point perspective on the software failure process,” Software Testing and Verification, vol. 13, no. 2, pp. 85–93, 2003.
- (26) S. Inoue and S. Yamada, “Software reliability assessment with multiple changes of testing-environment,” IEICE Trans. Fundamentals, vol. E98-A, no. 10, pp. 2031–2041, Oct. 2015.
- (27) 岡村寛之, 土肥 正, 尾崎俊治, “運用段階におけるソフトウェア製品の信頼性評価法 — 加速寿命試験モデルの提案 —,” 信学論 (A), vol. J83-A, no. 3, pp. 294–301, March 2000.

(R 研究会提案, 平成 29 年 12 月 4 日受付)

山田 茂 (正員：シニア会員)

昭 50 広島大・工・経営卒。昭 53 同大学院工学研究科修士課程了。昭 52-55 日本電装 (現デンソー) 株式会社品質保証部勤務。昭 58 同大学院工学研究科博士課程後期了。昭 58 岡山理科大・助手・専任講師。昭 63 広島大・工・第二类 (電気系)・助教授。平 5 鳥取大・工・社会開発システム・教授。平 30 同大学定年退職。現在、鳥取大学名誉教授/特任教授。工博。ソフトウェア信頼性工学、品質管理工学、プロジェクトマネジメントに関する研究に従事。情報処理学会平 3 年度 Best Author 賞、第 8 回電気通信普及財団賞テレコムシステム技術賞、日本信頼性学会 1998 年度高木賞、International Leadership Award in Reliability Engg. Research (ICQRIT/SREQOM, 2003) 受賞、Best Paper Award (International Computer Symposium, 2004) 受賞、プロジェクトマネジメント学会平 17 年度論文賞、平 15 年度鳥取大学研究業績賞、Leadership Award in Appreciation for Outstanding Contribution to ISSAT International Conference on Reliability and Quality in Design (2007, 2004), Outstanding Paper Award (2008 IEEE International Conference on Industrial Engineering and Engineering Management), International Leadership and Pioneering Research Award in Software Reliability Engineering (2009), 平 21 年度鳥取大学研究功績賞、Exceptional International Leadership and Contribution Award in Software Reliability (ICRITO '2010), プロジェクトマネジメント学会プロジェクトマネジメント功労賞 (2014), 日本オペレーションズ・リサーチ学会普及賞 (2017), Research Award for Outstanding Contribution to Software Reliability Engineering (ISSAT, 2017) 各受賞。情報処理学会、日本オペレーションズ・リサーチ学会、日本経営工学会、日本品質管理学会、プロジェクトマネジメント学会、日本信頼性学会、IEEE 各会員。

井上真二 (正員)

平 13 鳥取大・工・社会開発システム卒。平 18 同大学院工学研究科博士後期課程了。平 18 鳥取同大学院工学研究科助教、平 29 関西大・総合情報・准教授、現在に至る。博士 (工学)。ソフトウェア信頼性工学、品質管理工学、プロジェクトマネジメントに関する研究に従事。2004 International Computer Symposium — Best Paper Award, IEEE Reliability Society Japan Chapter — 2006 Outstanding Young Scientist Award, 2008 IEEE International Conference on Industrial Engineering and Engineering Management — Outstanding Paper Award 各受賞。日本オペレーションズ・リサーチ学会、日本品質管理学会、日本経営工学会、IEEE 各会員。

田村慶信 (正員)

平 10 鳥取大・工・社会開発システム卒。平 12 同大学院工学研究科博士前期課程了。平 15 同大学院工学研究科博士後期課程了。同年鳥取環境大・環境情報・情報システム・助手、平 18 広島工大・情報・情報・講師、平 20 同准教授、平 21 山口同大学院理工学研究科准教授、平 29 東京都市大・知識工・経営システム・教授、現在に至る。博士 (工学)。オープンソースソフトウェア、信頼性、ビッグデータ、及びクラウドなどに関する研究に従事。IEEE Reliability Society Japan Chapter Award (2010), Research Leadership Award in Area of Reliability (ICRITO 2010), Best Conference Paper Award (IEEE IEEM 2012) 各受賞。日本オペレーションズ・リサーチ学会、プロジェクトマネジメント学会、日本信頼性学会、IEEE 各会員。