# 次世代ソフトウェア信頼性評価技術の開発に向けて

土肥　正[†]　　岡村　寛之[†]

本研究では次世代のソフトウェア信頼性評価技術として，ソフトウェアメトリクスを利用した評価モデルの構築を行う．具体的には，従来のソフトウェア信頼度成長モデルに対して，メトリクスを説明変数とした回帰構造を取り入れる．また，信頼性尺度に強く寄与するメトリクスの分析ツールの開発を行い，数値実験を通じてその有効性を検証する．

# Towards development of the next-generation software reliability assessment technology

Tadashi Dohi[†], Hiroyuki Okamura[†]

This paper proposes a next-generation framework on software reliability assessment by incorporating software metrics as explanatory variables. More specifically, we apply the regression-based approaches to the existing software reliability growth models, and develop software reliability assessment tools to evaluate the software reliability quantitatively and to identify the significant metrics which strongly affect some reliability measures. It is shown through numerical experiments that our methods are quite effective and useful.

## 1. はじめに

ソフトウェアの高信頼化は近年の大きな課題である．一般的に，ソフトウェアの信頼性確保には，(i) 開発プロセスの管理技術の向上，(ii) 開発技術自体の向上，が大きく起因している．それと同時に，プロセスと製品の信頼性を定量的に評価し，プロジェクト管理技術および開発手法にフィードバックする取組は，多様化するソフトウェアに対して高信頼性を保証するための重要な技術である．70 年代初頭から研究が開始されたソフトウェア信頼度成長モデル（あるいはバグモデルとも呼ばれる）は，ソフトウェアテストで発見されるフォールトの計数過程を統計的に推論することで，発見フォールト数の飽和状態を監視し，ソフトウェア信頼度などの定量的評価尺度を算出するために用いられてきた．このようなフォールトの計数データだけを用いたソフトウェア信頼度成長モデルは，取扱いが非常に簡便であるため，現在においても尚実務において利用されている．しかしながら，従来のモデルでは，テスト労力やソフトウェア種別などの情報を明示的に利用せず，フォールト計数データのみに依拠しているため，それらの要因と定量化された信頼性の因果関係がブラックボックス化している．そのため「得られた数値の妥当性検証が難しい」ことや「管

【脚注】
† 広島大学　Hiroshima University

理技術や開発手法への明示的なフィードバックが難しい」ことが欠陥として指摘されている．

従来のソフトウェア信頼性技術の啓蒙活動が必ずしも産業界において成功していない要因の一つは，「信頼度を左右する要因を明らかにしたい」と言う開発現場における要求を念頭においたモデル開発が行われていなかった点にある．これは，モデルの抽象度が非常に高いため，信頼度を左右する要因を根本的に特定することが不可能である点に起因する．ソフトウェア工学分野では，モデルのあてはめによるソフトウェア信頼性評価技術は役に立たない技術として捉えられている向きがあり，現在主流となっている各種メトリクス計測に基づいた実証的ソフトウェア工学と独立した領域と見なされることが多い．

本研究では，従来のソフトウェア信頼度成長モデルの利点を活かしながら，より詳細なプロジェクトデータが扱えるモデルへの拡張を試みる．開発現場で解決すべき課題の一つは「どのような設計・テストをしたらフォールトのないソフトウェアが開発できるか」であり，その第一歩として，設計段階やテスト段階において計測されるメトリクスを有効に活用できるソフトウェア信頼度成長モデルを開発し，モデルから得られるソフトウェアの信頼性尺度が統計的にどのようなメトリクスに起因するかについて分析する．なお，本研究は独立行政法人情報処理推進機構技術本部ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の支援のもと行われた「次世代信頼性評価技術の開発とその実装」[1] における成果の一部をまとめたものである．

## 2. ソフトウェア信頼度成長モデル

本研究では，従来のモデルにおいてどのようにしてメトリクス情報を取り扱うかを議論し，その統計的な分析手法を提示する．そのため，ここでは基礎となるソフトウェア信頼度成長モデルの数理的な構造について述べる．ソフトウェア信頼度成長モデルはこれまでに様々なものが提案されているが，ここでは非同次ポアソン過程（NHPP: Non-Homogeneous Poisson Process）に基づいたモデルを包括的に扱う枠組みを紹介する．NHPP によるソフトウェアフォールト検出過程のモデリングの起源は Goel and Okumoto(1979)[6] であり，以来，過去 35 年間に渡って数多くのモデルが提案されてきた．NHPP モデルでは，時刻 $t$ までに発見されたバグ数の累積数を $N(t)$ とすると，$N(t)$ を以下の確率関数で定義する．

$$P(N(t)=k)=\frac{H(t)^k}{k!}e^{-H(t)}. \tag{1}$$

上述のポアソン型の確率関数で表現される確率過程は一般にポアソン過程と呼ばれ，さらに $H(t)$ が時刻に依存する（非定常である）ため，非同次ポアソン過程（NHPP）と呼ばれる．ここで $H(t)$ は時刻 $t$ までに発見される累積バグ数の「期待値」を表し，平均値関数と呼ばれる．

これまでに数多くの NHPP によるソフトウェア信頼度成長モデルが提案されているが，本質的な違いは平均値関数 $H(t)$ にどのような関数を仮定するかという点に集約される．一方，Langberg and Singpurwalla(1985)[8] は，以下の仮定に基づいたモデル化により，ほとんどすべての NHPP モデルが包括的に表現できることを示した．

**仮定 A:** テスト実施前にプログラム中に含まれる総バグ数は有限であり，平均 $a$ のポアソン分布に従う．

**仮定 B:** それぞれのバグは互いに独立に発見される．

**仮定 C:** 各バグの発見時刻は確率分布関数 $F(t)$ をもつ非負の確率変数によって表される．

このとき，時刻 $t$ までに発見される累積バグ数の確率関数は以下のようになる．

$$P(N(t)=k)=\frac{(aF(t))^k}{k!}e^{-aF(t)}. \tag{2}$$

これは平均値関数が $\mathrm{E}[N(t)]=aF(t)$ の形をもつ NHPP に一致する．$F(t)$ に 0 から 1 まで単調に増加する確率分布関数を代入することで，上述の枠組みは初期残存バグ数が有限個の NHPP モデルを任意に表現することができる．さらに，テスト時間が $(0,t_1],(t_1,t_2],\ldots,(t_{K-1},t_K]$ のように離散的に与えられる場合であっても，あるバグが $n-1$ 番目のテスト日までに発見されない条件のもとで $n$ 番目のテストで発見される条件付き確率（バグ発見確率）を $p_n=(F(t_n)-F(t_{n-1}))/(1-F(t_{n-1}))$ とすると，

$$P(N_n=k)=\frac{(aq_n)^k}{k!}e^{-aq_n} \tag{3}$$

のように表現することもできる．ここで $q_n=1-\Pi_{k=1}^{n}(1-p_k)$ である．

上述の仮定に従えば，ソフトウェア信頼度成長モデルは次の二つの要因によって決定されることが自明である．

(1) ソフトウェアバグ総数の平均値 ($a$)

(2) 単一のバグ発見時刻に関する確率分布 ($F(t)$)

上述の仮定 (1) と (2) は，それぞれ，テスト前にプログラム全体に残存する総バグ数とバグの見つかり具合を表現するパラメータが存在すると言い換えることができ，事実「バグの見つかり具合」の違いで様々な NHPP モデルがこれまでに提案されてきた．表 1 に代表的な NHPP モデルと仮定されるバグ発見確率の対応関係を示す．

表 1　代表的な NHPP モデルとバグ発見確率の対応.

| モデル | バグ発見確率分布 | 文献（既存モデル名） |
|---|---|---|
| Exponential | 指数分布 | 指数形 SRGM，Goel and Okumoto model [6] |
| Gamma | ガンマ分布 | 遅延 S 字形 SRGM [13,14] |
| Pareto | 第二種パレート分布 | 修正 Duane model [2,9] |
| Truncated Normal | 切断正規分布 | [12] |
| Log-Normal | 対数正規分布 | [3] |
| Truncated Logistic | 切断ロジスティック分布 | 習熟 S 字形 SRGM [10] |
| Log-Logistic | 対数ロジスティック分布 | Log-Logistic model [7] |
| Truncated Extreme-Value Max | 切断 Gumbel 分布 | 修正 Gompertz model [11, 15] |
| Log Extreme-Value Max | Frechet 型極値分布 | [11] |
| Truncated Extreme-Value Min | Gompertz 分布（最小値） | [11] |
| Log Extreme-Value Min | Weibull 分布 | Goel model [5,11] |

## 3. メトリクス情報を利用したソフトウェア信頼度成長モデル

### 3.1. 一般化線形ソフトウェア信頼度成長モデル

先に示したように，従来のソフトウェア信頼度成長モデルは，総バグ数とバグ発見確率の二つの要因から構成され，これらはソフトウェアテストにおいて観測された発見バグ数から統計的に直接推定される．しかしながら，総バグ数やバグ発見確率はソフトウェアの規模や種類などソフトウェア固有の特徴量に強く関連しており，そのような情報を含んだメトリクス情報を活用することで，信頼度推定の精度向上や，バグ総数などに寄与する要因分析を行うことが可能になる．

ソフトウェアメトリクスは，コード行数やソースの複雑度といった開発するソフトウェアに関するプロダクトメトリクスと，開発プロセスに関連したプロセスメトリクスに大きく分類することができる．ソフトウェア信頼性評価においては，ソフトウェアそのものの特徴量を表すプロダクトメトリクスと，ソフトウェアテストにおいて観測され得るプロセスメトリクスが特に重要であると考えられる．一般に，これら二つのメトリクスはそれぞれバグに与える影響が異なるため，その特徴を十分に考慮した分析手法を提案する必要がある．

いま，$j$ 個のモジュールからなるソフトウェアを考え，以下の仮定を設ける．

- ソフトウェアテストは逐次的かつ離散時刻上で実施され，各テストで発見されたバグは次のテストまでに修正される．
- モジュール $i$ において，一つのバグが $k$ 番目のテストで発見されるバグ発見確率 $p_{i,k}(0 \le p_{i,k} \le 1)$ を，モジュール $i$ の $k$ 番目のテストに関連したプロセスメトリクス $\boldsymbol{x}_{i,k}$ を用いて

$$\mathrm{logit}(p_{i,k}) = a_{0,i} + \boldsymbol{a}_i \boldsymbol{x}_{i,k} \tag{4}$$

のように表す．ここで logit はロジット関数であり，

$$\mathrm{logit}(p) = \log \frac{p}{1-p} \tag{5}$$

で与えられる．また，$a_{0,i}$ および $\boldsymbol{a}_i$ は回帰係数と呼ばれ，プロセスメトリクスがバグ発見確率に与える影響度を表す．

- 各モジュール $i = 1,\ldots,j$ に含まれる総バグ数 $M_1,\ldots,M_j$ はモジュール固有のプロダクトメトリクス $\boldsymbol{s}_i$ に依存した平均 $v_i$ のポアソン分布に従い，その平均値は以下で与えられる．

$$\log(v_i) = b_0 + \boldsymbol{b}\boldsymbol{s}_i. \tag{6}$$

ここで，$b_0$, $\boldsymbol{b}$ はプロダクトメトリクスが総バグ数に与える影響度を表す回帰係数である．

上記の仮定のもとで，モジュール $i$ の総バグ数が平均 $\exp(v_i)$ のポアソン分布に従うため，モジュール $i$ の $k$ 番目のテストまでに発見されるバグ数 $Y_{i,k}$ は以下の確率関数で与えられる．

$$\begin{aligned} &P(Y_{i,k} = y_{i,k}) \\ &= \sum_{m=y_{i,k}}^{\infty} P(Y_{i,k} = y_{i,k} | M_i = m) P(M_i = m) \\ &= \exp(-v_i q_{i,k}) \frac{(v_i q_{i,k})^{y_{i,k}}}{y_{i,k}!}. \end{aligned} \tag{7}$$

ここで，$q_{i,n} = 1 - \Pi_{k=1}^{n}(1 - p_{i,k})$ である．このモデルでは，従来のソフトウェア信頼度成長モデルに対して，総バグ数に対するポアソン回帰と $k$ 番目のテストにおけるバグ発見確率に対するロジスティック回帰を同時に適用していることに注目すべきである．本稿では上述のモデルを一般化線形ソフトウェア信頼度成長モデルと呼ぶ．この

モデルでは各モジュールの各テスト期間で発見されたバグ数，プロセスメトリクス，プロダクトメトリクスに関するデータから回帰係数 $a_{0,i}$,$\boldsymbol{a}_i$,$b_0$,$\boldsymbol{b}$ の推定を行う．また，上の仮定におけるロジット関数や対数関数はメトリクスからの影響度（線形の関数で仮定されている）から，実際の総バグ数やバグ発見確率への変換を規定しており，リンク関数と呼ばれる．これらは変更することが可能であり，プロビット関数や恒等関数を適用することもできる．

### 3.2. カーネル法の適用

一般化線形モデルに基づいたモデル化は，基本的にメトリクスとバグ数あるいはバグ発見確率に与える影響度が線形であることを仮定している．しかしながら，実際問題として，これらが線形の関係にあるかどうかを事前に判断するのは難しい．一方，適切なメトリクスを選択する必要があるため，任意の非線形構造を仮定することは有効ではない．これらの問題を解決する試みとして，ここではカーネル法の適用を考える．カーネル法とは，パターン認識などでよく用いられる手法の一つであり，線形回帰やサポートベクターマシンなどと組み合わせることで，非線形で表される因果関係を記述することができる．原理的には，データを高次元の特徴空間へ写像し，その高次元の特徴空間でデータの分析を行うものである．もとの空間において非線形な関係も，高次元空間では線形の関係で表せることが多く，線形回帰などの線形モデルを適用してより効率よく分析を行うことができる．また，高次元空間での座標を決める代わりにデータ間の内積にカーネル関数を用いることで，高次元化に伴う計算量の増大を抑えることができる．このため，文字列やグラフなどの構造データに対して，カーネル関数による内積を定義することで，様々な特徴量を分析できることも大きな利点となっている．

一般にカーネル法では，もとの線形モデルにカーネル関数を直接適用する．いま，一般化線形ソフトウェア信頼度成長モデルに対してカーネル法を適用すると，先に示した関係式を次のような関数で置き換えることができる

$$\mathrm{logit}(p_{i,k}) = \sum_{u=1}^{k} c_u \mathcal{K}(\boldsymbol{x}_{i,u}, \boldsymbol{x}_{i,k}) \tag{8}$$

$$\log(v_i) = \sum_{m=1}^{j} h_m \mathcal{K}(\boldsymbol{s}_i, \boldsymbol{s}_m). \tag{9}$$

ここで，$c_u$, $h_m$ は回帰係数，$\mathcal{K}(x, y)$ はカーネル関数であり，$x$ と $y$ の類似度（内積）を表す関数である（付録参照）．カーネル関数は類似度の性質を満たしていればどのような関数でも適用することができるため，通常，数値ベクトルに対してはガウスカーネルや多項式カーネルがよく用いられる．また，文字列やグラフなどの構造データを扱えるカーネル関数を導入することで，ソースコードやその他の設計ドキュメントからメトリクスを抽出することなく，モデルを構築することも可能である．例えば，二つの文字列 $x$, $y$ の何らかの距離 $D(x, y)$ が定義できる場合，これを用いたカーネルとして以下のようなものが構成できる．

$$\mathcal{K}(x, y) = \exp\left(-\frac{D(x, y)}{\sigma}\right). \tag{10}$$

本論文における数値例ではカーネル法の適用を行っていないが，具体的に，文字列スペクトルカーネル [16] とここで記述したモデルを用いてプログラムソースコードから直接，総バグ数を推定する試みを行っている．詳細は [1] を参照のこと．

## 4. 実験による検証

### 4.1. 検証ツール

一般化線形ソフトウェア信頼度成長モデルを利用する標準的な手順は次のようになる．

(1) データの収集：テスト中盤以降において，これまでのバグ発見数，テストに関するプロセスメトリクス（工数，カバレッジなど），各モジュール単位のプロダクトメトリクス（コード行数, 複雑度など）を収集する．

(2) モデルパラメータの推定：収集したバグ数に関するデータとメトリクスデータを用いて，一般化線形ソフトウェア信頼度成長モデルのパラメータ推定を行う．推定手法には最尤法あるいは罰則付き最尤法を用いる．また，モデルの適合性尺度をもとにしたメトリクスの取捨選択を行う．

(3) 信頼性評価：推定したモデルパラメータを用いることで，モジュール単位の期待残存バグ数やフォールトフリー確率などの信頼性尺度を算出する．

最終的には，定量的な信頼性尺度をもとにして，開発中のソフトウェアに対してリリースの判定やテスト戦略の変更などのアクションを行う．また，メトリクスと定量的な信頼性尺度の関連が明確になるため，この分析結果を次期以降の類似プロジェクトに反映させることもできる．

本研究では，上記の (2) と (3) を支援するツールの開発を行った[1]．開発したツールは，推定に関する統計的な計算を行う DLL(Dynamic Link Library) をコアとし

【脚注】

1 http://www.rel.hiroshima-u.ac.jp/msrat/

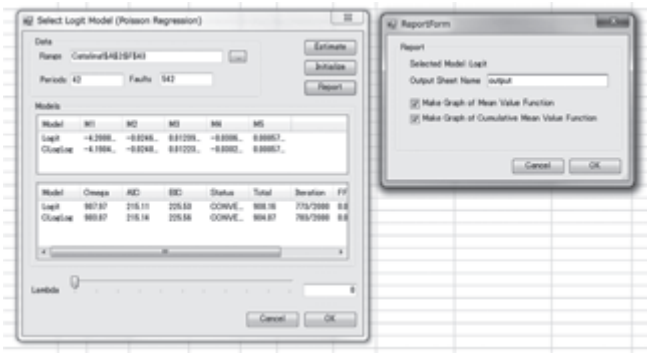

図 1　信頼性評価ツールの画面

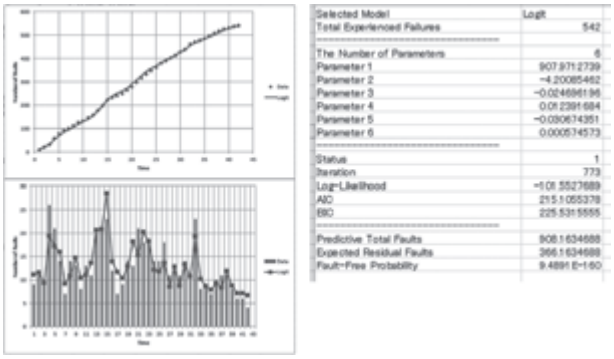

図 2　出力例

て，Microsoft Excel AddIn によるインタフェースおよび統計ツール R からの利用が可能となっている．図 1 に Microsoft Excel AddIn インタフェースの画面を示す．Microsoft Excel ではシート上で管理しているバグ発見数およびメトリクスデータを選択することで容易に推定が行える．また，結果として各種信頼性尺度や推定したモデルによる予測グラフなどの出力を行うことができる（図 2 参照）

## 4.2. 一般化線形ソフトウェア信頼度成長モデルの検証

オープンソースプロジェクトにおけるデータを対象としたモデルの有効性分析を行う．ここでは，一般化線形ソフトウェア信頼度成長モデルの適合性評価を行う．特に，従来の手法であるメトリクスデータを用いない単純な NHPP モデルに対して，プロダクトメトリクスだけを利用する場合，プロセスメトリクスだけを利用する場合，両方のメトリクスを利用する場合の比較を行う．AIC(Akaike Information Criterion)[4] を用いた適合性評価を行うことで，単純なデータとの誤差比較ではなく，背後にある（真の）モデルを適切に表現できるかどうかの検証を行う．AIC は以下のように定義されるモデル選択基準であり，値が小さいほどデータをよく説明しているモデルであることを示している．

$$\mathrm{AIC} = -2(\text{最大対数尤度} - \text{モデルのパラメータ数}). \tag{11}$$

検証に用いるデータは Apache Software Foundation (ASF) で管理されている Tomcat プロジェクトを対象とした．Tomcat のバージョンは 5, 6, 7 であり，メジャーバージョンアップについては別アプリケーション（別プロジェクト）と見なして分析している．それぞれのバージョンに対してバグトラッキングシステム（ASF Bugzilla）からバグレポートを収集し，バグ数の計測を行った．データの収集期間は Tomcat5 が 2004/5 から 2009/1，Tomcat6 が 2006/1 から 2013/11，Tomcat7 が 2010/6 から 2013/11 となっている．また，モジュール構成は次の通りである．

- webapps (manager): 管理用ウェブアプリケーション
- catalina: Servlet container のコア
- connectors: Tomcat と Apache の連携
- jasper: JSP ページコンパイラとランタイムエンジン
- servlets: Servlet プログラム群

プロダクトメトリクスは上記のモジュールに対応するソースコード群から直接計測した．計測には Source Monitor[2] を用いて，対応するソースコードが格納されているディレクトリ全体に対して以下のメトリクスを算出した．

- Files (Fl): ファイル数
- Lines (Ln): コード行数
- Statements (St): ステートメント数
- % Branches (Br): 分岐の出現率
- Calls (Ca): メソッドの呼び出し回数
- % Comments (Cm): コメントの出現率
- Classes (Cl): クラス数
- Methods/Class (Me/Cl): 単一クラスあたりのメソッド数
- Avg Stmts/Method (St/Me): 単一メソッドあたりのステートメント数
- Max Complexity (MCx): 最大複雑度
- Max Depth (MDp): 最大のネストの深さ
- Avg Depth (ADp): 平均のネストの深さ
- Avg Complexity (ACx): 平均複雑度

一方，プロセスメトリクスに関してはプロジェクトのアクティビティを表す指標として考えられる以下のメトリクスをそれぞれのモジュールに対して計測した．

- time: 日数
- ctime: 累積日数
- comments: Bugzilla 上での該当モジュールに対するコメント数
- votes: Bugzilla 上での該当モジュールに対する投票回数
- wokers: コメントを投稿した人の数（Tomcat5 のみ）

【脚注】

2　http://www.campwoodsw.com/sourcemonitor.html

実験では，まずモジュール毎で独立に (1)NHPP モデルの推定，(2) プロセスメトリクスのみを考慮したモデルの推定を行った．(1) では表 1 に示した代表的な 11 種類のモデルすべてに対してパラメータを推定し，最小の AIC を示すモデルを選択した．また，プロセスメトリクスのみを用いたモデルでは，変数増減法による変数選択を行って最小の AIC を示すメトリクスの組み合わせを求めた．次に，モジュールのプロダクトメトリクスを用いて，(3) プロダクトメトリクスのみを考慮したモデルの推定と (4) 一般化線形ソフトウェア信頼度成長モデルの推定を行った．(3) の推定では，事前に各モジュールに適合した NHPP モデルを選ぶ必要がある．これには，(1) の手順で選ばれたモデルと同じモデルを採用した．同様に (4) では，各モジュールに適用するプロセスメトリクスを選ぶ必要がある．これには (2) で選択されたメトリクスを用いることとした．さらに，(3) と (4) ではプロダクトメトリクスの選択を行う必要がある．これも，変数増減法により最小の AIC を示す組み合わせを選んだ．

表 2 から表 4 は NHPP モデルのみ（NHPP），プロセスメトリクスのみ（Logit），プロダクトメトリクスのみ（Poi），一般化線形ソフトウェア信頼度成長モデル（GLSRM）それぞれを適用したときの AIC の値を示している．表から，いずれの場合でも，AIC の大小関係において

$$\mathrm{NHPP} \geq \mathrm{Poi} \geq \mathrm{Logit} \geq \mathrm{GLSRM} \qquad (12)$$

の関係が得られた．つまり，適合性の観点からは一般化線形ソフトウェア信頼度成長モデルが最も高い適合性を示すことがわかった．また，プロダクトメトリクスのみの場合とプロセスメトリクスのみの場合を比較すると，プロセスメトリクスを利用する方がモデルの適合性が高いことがわかった．これは初期の総バグ数に影響する情報よりも，テスト中に観測された発見バグ数から得られる情報の方が信頼性評価により寄与することを示唆している．この一つの理由として，今回の実験ではモジュール数が比較的少ないため，プロダクトメトリクスから得られる情報量が相対的に少なかったことが考えられる．よりモジュール数が多い状況では，プロダクトメトリクスからの情報も信頼性評価に大きく影響を与えるものと予想される．

また表 5 から表 7 は，選択された NHPP モデル，プロセスメトリクス，プロダクトメトリクスの種類を表している．選択されたメトリクスをみると，プロダクトメトリクスでは最大複雑度（MCx）が比較的多く選ばれており，プロセスメトリクスではコメント数が多く選択される結果となった．これは，プロダクトメトリクスとして従来から重要視されてきたコード行数や複雑度などの因子が信頼性評価にも大きく影響することを意味している．また，コメント数などプロジェクトのアクティビティを直接表す指標の利用は，バグ発見確率をより正確に同定するために大きく寄与することも確認できた．

表 2　Tomcat5 に対する AIC.

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | 268.96 | 271.04 | - | - |
| connectors | 145.10 | 128.20 | - | - |
| jasper | 159.96 | 147.70 | - | - |
| servlets | 116.99 | 102.08 | - | - |
| webapps | 150.51 | 137.69 | - | - |
| Total (project) | 841.52 | 786.71 | 838.87 | 784.58 |

表 3　Tomcat6 に対する AIC.

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | 489.30 | 388.66 | - | - |
| connectors | 277.74 | 201.74 | - | - |
| jasper | 293.51 | 247.27 | - | - |
| manager | 163.15 | 125.58 | - | - |
| servlets | 181.54 | 181.58 | - | - |
| Total (project) | 1405.24 | 1144.83 | 1403.82 | 1144.84 |

表 4　Tomcat7 に対する AIC.

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | 247.91 | 210.87 | - | - |
| connectors | 168.82 | 129.44 | - | - |
| jasper | 167.34 | 137.57 | - | - |
| manager | 117.07 | 90.49 | - | - |
| servlets | 148.33 | 110.57 | - | - |
| Total (project) | 849.47 | 678.94 | 847.40 | 675.86 |

表 5　Tomcat5 において選択されたモデルとメトリクス.

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | Truncated Extreme-Value Max SRGM | comments, ctime | St, MCx | St, MCx |
| connectors | Log Extreme-Value Max SRGM | worker, ctime | | |
| jasper | Log-Logistic SRGM | worker, ctime | | |
| servlets | Gamma SRGM | comments, ctime | | |
| webapps | Truncated Logistic SRGM | comments, ctime | | |

## 5. まとめと今後の課題

実験の結果から，本研究で提案した一般化線形ソフトウェア信頼度成長モデルが従来モデルと比較してかなり高い適合性能力を有することが示された．AIC などの情報量規準はモデルの自由度を考慮したモデル選択基準であり，値が小さいものが最良モデルとなる．よって，扱う統計情報の量が多くなることでモデルの自由度が増えたとしても，メトリクス情報の計測は適合性評価結果に貢献できていることが分かる．また，プロダクトメトリクスとプロセスメトリクスの効果を比較した場合，明らかに後者の方が適合性に関する貢献度が高いことも分かった．コード行数，ファイル数，クラス数などのソフトウェアの規模に関するメトリクスやソースコードの複雑性は，ソフトウェアの信頼性評価に全く影響を与えないわけではないが，少なくともテストの進捗状況に応じて変化するバグ発見の挙動に直接的に効果を与えることは稀である．このことは実証ソフトウェア工学でこれまで提案されてきたプロダクトメトリクス計測よりも，テスト時間やバグフィックスのために費やされた労力（コメント数，メール交換数など）の方がむしろ信頼度成長現象を説明するための要因となっていることを意味している．

一般化線形ソフトウェア信頼度成長モデルの実務的利点として，複数のモジュールを有するシステムのバグ情報並びにメトリクス情報を直接扱うことで，各モジュール単位で信頼性評価を行うことが可能であることが挙げられる．従来まで，モジュールテストにおける信頼性評価では，各々のモジュールに対して対応するバグデータに信頼性評価モデルを独立に適用する方法がとられてきた．このような方法では，回帰構造を導入したとしてもプロダクトメトリクスの情報を矛盾なく信頼性評価に組込むことが困難であった．本研究で提案した一般化線形ソフトウェア信頼度成長モデルは，オープンソースに代表されるようなリポジトリデータベースで開発管理されるソフトウェアに対しても，現実的かつ有効な評価スキームを提供している．

設計情報や開発管理情報がメトリクスの形で集約されているような場合，すなわち，定量的なソフトウェアの開発管理が徹底している現場においては，提案ツールがそのまま利用できる．計測可能でかつ入手可能なメトリクス情報があれば，とりあえずそれを全て利用することでより正確な信頼性評価が行える．しかしながら，メトリクス情報の計測と管理にはコストがかかることも事実である．どのようなメトリクスを収集すべきかについての実証的知見がなければ，本研究で開発された信頼性評価技術を有効に活用することは難しい．その意味で，これまで実証ソフトウェア工学において積み重ねられた研究成果や企業で培われた独自のノウハウをメトリクス選択に生かすことが肝要であると考えられる．一方で，考えられ得る種類のメトリクス情報が獲得できたとしても，適切な選択を行うためには膨大な計算量が必要となる．例えば三つのモジュールそれぞれに対して 2 種類のプロダクトメトリクスと 3 種類のプロセスメトリクスが観測されている場合でも，適切な要因を選択するためには 21952 通りの組み合わせを試す必要がある．このような問題に対処するため，変数選択手法の強化が課題となる．一つの可能性としては，ここで紹介したカーネル法と現在のデータマイニング技術を融合することで，高いスケーラビリティを持ったアルゴリズムを開発することである．一方，信頼性評価技術を業務の一環として恒

表 6　Tomcat6 において選択されたモデルとメトリクス．

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | Truncated Extreme-Value | comments, votes, ctime | Fl,Cl,MCx | Fl,Ln,Cm,Cl |
| connectors | Log Extreme-Value Max | comments, votes, ctime | | |
| jasper | Truncated Extreme-Value Max | time, comments | | |
| manager | Log Extreme-Value Min | comments, ctime | | |
| servlets | Log Extreme-Value Max | comments, ctime | | |

表 7　Tomcat7 において選択されたモデルとメトリクス．

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| catalina | Truncated Extreme-Value Min | comments, votes | Fl, Ca, Cm | St, Cm |
| connectors | Log-Normal | comments, votes | | |
| jasper | Truncated Extreme-Value Min | comments | | |
| manager | Gamma | comments, votes | | |
| servlets | Exponential | comments, votes, ctime | | |

常的に利用するためには，バグトラッキングシステムやリポジトリデータベースと連動するツールが必要不可欠である．バグトラッキングなどのデータ形式は標準化されていない場合が多く，このようなデータの標準化が，データマイニング技術と融合した信頼性評価実務において今後益々必要になると考えられる．

## 謝辞

本研究は，独立行政法人情報処理推進機構技術本部ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の支援のもと行われたものである．

## A カーネル法

カーネル法の基本的なアイデアは，非線形の関係を線形で表すために，データに対して非線形変換を事前に適用することから始まる．例えば，真の関係として $Y = X^2$ で表される入力値 $X$ と出力値 $Y$ のデータ $\{(x_1, y_1),\ldots,(x_n, y_n)\}$ に対する分析を考える．この場合，このデータに対して $Y = aX + b$ の関係を仮定した線形回帰では明らかに真の関係を捉えることができない．しかしながら，元のデータに対して非線形変換を行った新たなデータ $\{(x_1^2, y_1),\ldots,(x_n^2, y_n)\}$ を考えると，もとの関係が非線形であるにもかかわらず，$Y = aX + b$ の関係式を仮定した線形モデルによる分析で真のモデル構造を同定することができる．

このように，得られたデータに対する非線形変換を行うことは，非線形な関係を分析するのに有効な手段である．一方で，どのような非線形変換が適切かという問題がある．これを組み合わせにより解決しようと試みると組み合わせ爆発が起こるため，計算上の困難性が生じる．カーネル法はこの計算上の問題に対する一つの有効な手段を与える．

いま，$\Phi$ を任意のデータが存在する空間から（高次元の）あるベクトル空間への非線形写像とする．このとき，先の議論は $\Phi$ によって元のデータを非線形変換し，適当なベクトル空間において線形モデルで分析することに対応する．一般に線形モデルを用いた分析では，二つのデータに対する内積が定義されている必要がある．ここで，$<x, y>$ をデータ $x$ と $y$ の内積として表記すると，あるベクトル空間においては $<\Phi(x), \Phi(y)>$ が計測できれば，線形モデルによる分析が可能であることを意味している．カーネル法におけるアイデアの一つは，非線形写像 $\Phi$ を規定することなく，内積を表わすカーネル関数

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = <\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y})> \qquad (13)$$

を直接定義することにある．これはカーネルトリックと呼ばれ，これにより高次元ベクトル空間における内積の計算をすることなく，より少ない計算量で線形モデルによる分析が可能となる．一般に，実数値関数を考えるとき，$\mathcal{K}$ は以下を満たすような関数（正定値カーネルと呼ばれる）であればどのようなものを選んでも良い．

・対称性：

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathcal{K}(\boldsymbol{x}_j, \boldsymbol{x}_i) \quad \text{for } i, j = 1, \ldots, n. \qquad (14)$$

・正値性：対称性のもと，以下の行列 $\boldsymbol{G}$ が半正定値

$$\boldsymbol{G} = \begin{pmatrix} \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}. \qquad (15)$$

【参考文献】

[1] 2013 年度ソフトウェア工学分野の先導的研究支援事業「次世代ソフトウェア信頼性技術の開発とその実装」成果報告書 . 独立行政法人情報処理推進機構技術本部ソフトウェア高信頼化センター（http://www.ipa.go.jp/files/000038549.pdf）.

[2] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood. Evaluation of competing software reliability predictions. IEEE Transactions on Software Engineering, SE-12:950-967, 1986.

[3] J. A. Achcar, D. K. Dey, and M. Niverthi. A Bayesian approach using nonhomogeneous Poisson processes for software reliability models. In A. P. Basu, K. S. Basu, and S. Mukhopadhyay, editors, Frontiers in Reliability, pages 1-18. World Scientic, Singapore, 1998.

[4] H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, Proc. 2nd Int'l Sympo. on Information Theory, pages 267-281. Akademiai Kiado, 1973.

[5] A. L. Goel. Software reliability models: Assumptions, limitations and applicability. IEEE Transactions on Software Engineering, SE-11:1411-1423, 1985.

[6] A. L. Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability, R-28:206-211, 1979.

[7] S. S. Gokhale and K. S. Trivedi. Log-logistic software reliability growth model. In Proc. 3rd IEEE Int'l. High-Assurance Systems Eng. Symp. (HASE-1998), pages 34-41. IEEE CS Press, 1998.

[8] N. Langberg and N. D. Singpurwalla. Unication of some software reliability models. SIAM Journal on Scientific Computing, 6(3):781-790, 1985.

[9] B. Littlewood. Rationale for a modied Duane model. IEEE Transactions on Reliability, R-33(2):157-159, 1984.

[10] M. Ohba. Inflection S-shaped software reliability growth model. In S. Osaki and Y. Hatoyama, editors, Stochastic Models in Reliability Theory, pages 144-165. Springer-Verlag, Berlin, 1984.

[11] K. Ohishi, H. Okamura, and T. Dohi. Gompertz software reliability model: estimation algorithm and empirical validation. Journal of Systems and Software, 82:535-543, 2009.

[12] H. Okamura, T. Dohi, and S. Osaki. Software reliability growth models with normal failure time distributions. Reliability Engineering and System Safety, 116:135-141, 2013.

[13] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. IEEE Transactions on Reliability, R-32:475-478, 1983.

[14] M. Zhao and M. Xie. On maximum likelihood estimation for a general non-homogeneous Poisson process. Scandinavian Journal of Statistics, 23:597-607, 1996.

[15] 山田 . ゴンペルツ曲線を用いた確率的ソフトウェア信頼度成長モデル . 情処学論 , 33(7):964-969, 1992.

[16] 福水 . カーネル法入門 . 多変量データの統計科学 8. 朝倉書店 , 2010.