独立行政法人情報処理推進機構委託

# 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評価技術の開発とその実装」
# 成果報告書

平成 26 年 2 月

国立大学法人広島大学

本報告書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」の公募による採択を受けて国立大学法人広島大学大学院工学研究院情報部門（研究責任者土肥正）が実施した研究の成果をとりまとめたものである.

目次

# 研究成果概要

## 1. 背景

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「定量化」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」する取組みは, 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性モデルは, 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評価するのに用いられてきた. このようなフォールトの計数データだけを用いた「バグモデル」や「ソフトウェア信頼度成長モデル」は, 取り扱いが非常に簡便である. また, モデルから得られるソフトウェアの信頼度は「ある規定の条件下で規定の期間中にソフトウェア障害が発生しない確率」と定義され, 正確な信頼度の見積もりができるのであれば開発者にとって非常に有益な情報となるため, いくつかの民間企業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を利用しているため, 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の検証が難しい」, 「管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」と言う重大な欠陥をもつ.

一方, 現実的には, テスト網羅度や欠陥密度のような原因と結果が明確な可観測情報に基づいて, 主観的な信頼性を算出する手法がとられることも多い. 網羅度や欠陥密度は信頼性評価尺度のひとつではあるが, これは当該ソフトウェアの信頼性を正確に表現しているものではない. つまり, 網羅度や欠陥密度を導出したとしても, 確率・統計理論を用いない限り, 網羅度・欠陥密度と定量的ソフトウェア信頼性の因果関係において妥当性を検証することが難しい.

従来のソフトウェアの定量的信頼性評価では, テスト工程で発見されたフォールトの個数情報のみから「残っているバグの個数」などを推定する試みがなされている. これは, 「発見されたフォールト個数」という非常に簡単な情報であるため, ソフトウェアの種類に係わらず, あらゆるソフトウェア開発に対して適用されてきた.

本研究では, このような抽象化・汎用化に対する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, より詳細なデータ(統計量)を扱えるモデルへと拡張を行う. 開発現場で獲得し得る情報水準に応じて信頼性評価の方法を分類する次世代のソフトウェア信頼性評価技術の体系化を目指す. 最も重要なポイントは「信頼性評価の妥当性検証を可能にすること」と「管理技術, 開発手法へフィードバック可能な情報を与えること」である. これを実現する本質は「信頼性と可観測な要因の因果関係を明らかすること」である. このような目的に従い, 本研究では, 各々の現場で獲得可能な情報水準に基づいた信頼性評価体系を提案する.

## 2. 研究内容

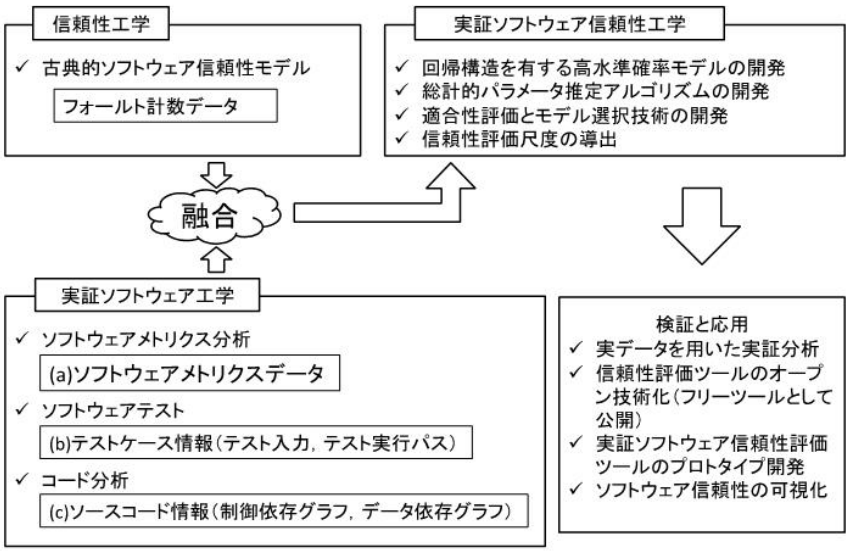


**図 1:研究課題の概念図**

図1に本研究の概念図を示す. 本研究では, 各々の現場で獲得可能な情報水準に基づいた信頼性評価体系を提案する. 具体的には, 3 種類のデータ(a)ソフトウェアメトリクス, (b)テストケースの情報(テスト入力, テスト実行パスなど), (c)ソースコードの情報(制御依存グラフ, データ依存グラフなど)と, テスト工程で観測されるフォールト計数データに対する関係をモデル化することで, 各々のデータを有機的に利用するソフトウェア信頼性モデルおよびモデルに基づいた各種信頼性評価尺度の導出に関する方法論を開発する. すなわち, (a)開発規模, テスト労力, 網羅度のような既に要約されたメトリクスデータだけが利用可能であれば, 回帰構造を確率モデルに組み込むことで, 信頼性評価モデルを記述することができる. 一方, (b)テストケースの詳細な情報が利用可能であれば, テストケースの距離空間をテスト入力と実行パスから定義し, ミクロなフォールト位置情報とマクロなフォールト検出事象をモデル化すれば良い. さらに, (c)ソースコードを直接利用できるのであれば, ソースコードから要約されたメトリクスを抽出する前に, ソースコードから静的に分析して得られる依存グラフとフォールトの作り込み・発見現象をモデル化した信頼性評価が可能である.

最終的に, 本研究で開発された信頼性評価技術をツール上に実装し, それをフリーウェアとして公開することで信頼性評価技術の実務への応用を啓蒙する. 具体的に, Excel インタフェースを利用して手軽に信頼性評価を行うことができる MSRATS と, 統計解析ソフト R 上で駆動する Rsrat の2種類の信頼性評価ツールを開発する.

本研究課題では次の研究目標を設定した.

- 研究目標1「メトリクスを扱う信頼性評価モデルの構築」:(a)ソフトウェアメトリクス, (b)テストケース情報, (c)ソースコード情報に基づいたソフトウェア信頼性評価のための確率モデルの解析を行う. 現在までに行われてきた研究と新しい成果を組み合わせることで最良モデル

（最も良いカーネルの構造など）の選定を行う.

- 研究目標2「パラメータ推定アルゴリズムの開発」：各々のモデル(a), (b), (c)で必要とされる情報(データ)から未知パラメータを効率良く推定するためのアルゴリズムを開発し，その安定性解析を実施する.
- 研究目標3「データを用いたモデルの有効性検証」：実際の事例に基づいた実証分析を通じて，提案技術の有効性を定量的に評価する．また，一般に公開されているベンチマークデータや開発データを再調査し，実証分析に利用可能なデータを分類・整理する.
- 研究目標4「モデルによる信頼性評価を実装したツールの開発」：Excel，R の拡張機能として，提案したモデルおよび推定アルゴリズムを実装する.

## 3. 研究成果

### 研究目標１「メトリクスを扱う信頼性評価モデルの構築」

#### (1) 一般化線形モデル構築

ソフトウェアメトリクスとは、ソフトウェア自身あるいはその開発を特徴づける定量的な指標である．一般に，ソフトウェアメトリクスは，コード行数，ソースの複雑度といった設計段階のソフトウェアに関する情報のデザインメトリクス，テスト工程において計測される，投入するテストケース数やカバレッジなどソフトウェアのテストの進捗具合やフォールトの検出に寄与するような情報であるテストメトリクスの2種類に分類できる．デザインメトリクス，テストメトリクスはそれぞれ，フォールト検出に与える影響が異なることを考慮して，デザインメトリクス，テストメトリクスを包括的に扱うモデルを開発した.

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的に実施され，各テスト毎で検出されたフォールトは次のテストまでに修正される.
- モジュール$i$において単一のフォールトが$k$番目のテスト区間で検出されるフォールト検出確率$p_{i,k}$ $\left(0 \le p_{i,k} \le 1\right)$ はモジュール$i$において$k$番目のテスト区間に関するテストメトリクスベクトル$x_{i,k} = \left(x_{i,k,1}, \dots, x_{i,k,r_i}\right)$ を用いて

$$l_p(p_i, k) = \alpha_{0,i} + \alpha_i x_{i,k}$$

となる．ここで$l_p(\cdot)$はフォールト検出確率のリンク関数である.

- 各モジュール$i = 1, \dots, j$ に含まれる総フォールト数（初期フォールト数）$M_1, \dots, M_j$はモジュール固有のデザインメトリクス$s_i$に依存した平均$\xi_i$のポアソン分布に従い，その平均は以下で与えられる.

$$l_\xi(\xi_i) = \beta_0 + \beta_{s_i}.$$

ここで，$l_\xi(\cdot)$は総フォールト数のリンク関数である.

いま,

$$l_p(p) = \mathrm{logit}(p) = \log\frac{p}{1-p},\ \ l_\xi(\xi) = \log(\xi)$$

とすると，モジュール$i$の$k$番目のテスト区間までに検出されるフォールト数$Y_{i,k}$は以下の確率関数で与えられる．

$$\begin{aligned}&P\left(Y_{i,k_i} = y_{i,k}\right)\\&= \sum_{m_i=y_{i,k}}^{\infty} P\left(Y_{i,k_i} = y_{i,k} | M_i = m_i\right) P(M_i = m_i)\\&= \exp\left(-\xi_i\lambda_{i,k}\right)\frac{\left(\xi_i\lambda_{i,k}\right)^{y_{i,k}}}{y_{i,k}!}.\end{aligned}$$

ここで，$\lambda_{i,n_i} = 1 - \prod_{k=1}^{n_i}\left(1 - p_{i,k}\right)$ である．

このモデルは，モジュール$i$に含まれる総フォールト数に対するポアソン回帰と$k$番目のテスト区間におけるフォールト検出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．これにより一般化線形の考えを取り入れたソフトウェア信頼性モデルを構築することができる．

**(2) カーネル法の適用**

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表現することができる．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．元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すことができることが多く，線形回帰などの線形モデルの適用に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をカーネル関数で表す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とと，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分析できることも大きな利点となっている．

いま，一般化線形ソフトウェア信頼性モデルに対してカーネル法を適用する．この場合，先に示した線形の関係式を次のような関数で置き換える．

$$l_p\left(p_{i,k}\right) = \sum_{u=1}^{k}\mathcal{K}\left(x_{i,u}, x_{i,k}\right),\ \ l_\xi(\xi_i) = \sum_{u=1}^{k}\mathcal{K}(s_i, s_m).$$

ここで，$\mathcal{K}(x,y)$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．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．通常，数値ベクトルに対してはガウスカーネルおよび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うカーネル関数は上述のものとは大きく異なる．例えば，文字列に関しては部分文字列の頻度によってベクトル化するカーネルが考えられている．また，二つの文字列$x, y$の距離$D(x,y)$が定義された場合，これを用いたカーネルを用いることができる．これにより，テストデータ入力やプログラムソースなどの構造データをメトリクスの抽出なしに扱うことができる．

## 研究目標２「パラメータ推定アルゴリズムの開発」

### (1)一般化線形ソフトウェア信頼性モデルに対する推定アルゴリズムの構築

一般化線形ソフトウェア信頼性モデルに対する推定手法を開発した．一般化線形ソフトウェア信頼性モデルは従来のソフトウェア信頼性モデルと比較してパラメータ数が多いため，効率的な推定を行う目的でEMアルゴリズムの適用を行った．

EMアルゴリズムはNHPPモデルでも用いている手法であり，未観測データを含むモデルパラメータを推定する一般的な推定の枠組みである．一般的には，未観測データの期待値を計算するE-stepと完全データ（観測データと未観測データ）の期待対数尤度を最大にするM-stepの二つの手続きを繰り返し，これらのステップで観測データに対する最尤推定値を計算することができる．

ここで考える一般化線形モデルの場合，未観測データを各モジュールの総フォールト数$N_1, \dots, N_j$とし，モジュール$m$のフォールト検出時刻列を$T_{m,1} < T_{m,2} < \cdots < T_{m,N_m}$とする．このとき，完全対数尤度は

$$\log\mathcal{L}\left(\beta_0, \beta, \alpha_{0,1}, \alpha_1, \dots, \alpha_m | \mathcal{D}, \mathcal{U}\right)$$
$$= \sum_{m=1}^{j} N_m \left(\beta_0 + \beta_{S_j}\right) - \sum_{m=1}^{j} \exp\left(\beta_0 + \beta_{S_j}\right) + \sum_{m=1}^{j}\sum_{k=1}^{N_m} \log\left(\lambda_{m,k} - \lambda_{m,k-1}\right)$$

となる．つまり，最初の二項が$N_1, \dots, N_m$を従属変数としたポアソン回帰となり，$\lambda_{m,k}$に関する項は$N_m - \sum_{u=1}^{k-1} y_u$を従属変数としたロジスティック回帰となる．一方で，総フォールト数の期待値は与えられたパラメータ$\beta_0, \beta, \alpha_{0,m}, \alpha_m$に対して

$$\mathrm{E}[N_m] = \sum_{k=1}^{K_m} y_{m,k} + \exp\left(\beta_0 + \beta_{S_m}\right) \prod_{k=1}^{K_m} \frac{\exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)}{1 + \exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)}$$

として計算できる．上記の式を利用して，最終的に通常のポアソン回帰およびロジスティック回帰を利用した効率的な推定アルゴリズムを開発した．

一方，一般化線形ソフトウェア信頼性モデルでは要因数に応じた回帰係数の選択（要因選択）が必要となる．要因選択手法には，AICを用いた手法を適用した．これは最尤法における真のモデルとのバイアスをパラメータ数によって近似した指標であり，AICを最小にするモデルが最良のモデルとなる．

### (2)カーネル法に対するパラメータ推定および近似手法の適用

カーネル法で高次元空間へ写像した要因データは元のデータ数と同じか，それ以上の数となるため，一般的な最尤法の適用では過適合の問題が生じる．そこで，カーネル法を適用したモデルに対しては正則化を行った．正則化を用いた最尤法は罰則付き最尤法とも呼ばれ，罰則項$P(\lambda)$を用いて次の罰則付き尤度を最大にするようなパラメータ推定を行う．

$$\log \mathcal{L}(\theta | D) + \lambda P(\theta).$$

罰則項は，平滑化などモデルに必要な事前知識を用いて設定されることが多いが，汎用的に用いられる罰則項としてはパラメータベクトルのL2ノルムの二乗を用いた．一方で，罰則の強さを表すパラメータ（正則パラメータあるいは罰則パラメータ）$\lambda$には，ABIC(Akaike Bayesian Information Criterion)を用いた決定手法を適用した．つまり，ABICを最小にするような正則化パラメータを用いた．

また，カーネル法では回帰係数の数が多いことから計算時間が通常の一般化線形ソフトウェア信頼性モデルに比べてかかる可能性がある．そこで，次の二点の工夫を考えた．

**高速な数値計算ライブラリの利用**

一般化線形ソフトウェア信頼性モデルのパラメータ推定では，そのEMアルゴリズムの中でポアソン回帰およびロジスティック回帰に関するパラメータを重み付き最小二乗法で解く．この計算コストを低減するため重み付き最小二乗法に対して高速な数値計算ライブラリを適用した．

**ブートストラップサンプリングの利用**

もう一つの工夫としては，データ増加を直接的に軽減する手法としてサンプリング（ブートストラップサンプリング）の活用を行った．ブートスラップサンプリングとは取得したデータから再サンプリングすることで擬似的に同じデータセットを作成する手法の総称である．ここでは，各モジュールに対して静的メトリクスとして得られたカーネル値に対するデータからランダムにモジュールを限定した場合のサンプルを擬似的に生成し，この擬似的なデータに対して推定と予測を行った．

**研究目標３「データを用いたモデルの有効性検証」**

**(1)適合性評価**

一般化線形ソフトウェア信頼性モデルのデータへの適合性を調査した．特に，従来の手法である単純なNHPPモデルと，静的メトリクスだけをポアソン回帰として利用するソフトウェア信頼性モデル，動的なテストメトリクスをロジスティック回帰として利用するソフトウェア信頼性モデルとの比較を行った．適合性評価にはAICを用いた．検証で用いるデータはApache Software Foundation (ASF)で管理されているTomcatプロジェクトおよびLenyaプロジェクトを対象とした．また，バージョンの違いによる影響も見るため，Tomcatについては幾つかのバージョンに対するデータを収集した．さらに，それぞれのプロジェクト毎の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た．静的メトリクスには各プロジェクトのモジュールにおけるコード行数や複雑度などをソースコードから直接算出した．一方，動的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Bugzillaでのコメント数などをそれぞれのモジュールに対して計測した．

表1はTomcat5に対してNHPPモデルのみ(NHPP)，ロジスティック回帰によるソフトウェア信頼性モデル(Logit)，ポアソン回帰によるソフトウェア信頼性モデル(Poi)，一般化線形ソフトウェア信頼性モデル(GLSRM)をそれぞれ適用したときのAICの値を示している．

表 1: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 |

表から，AICに関してNHPP ≥ Poi ≥ Logit ≥ GLSRMの関係が得られる．つまり，適合性の観点からはGLSRMが最も良いことがわかった．また，静的なメトリクスを扱うポアソン回帰によるソフトウェア信頼性モデルと，動的なメトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデルを比較すると，動的なメトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデルの方が適合性が高い．これは初期の総フォールト数を決めるための情報よりも，観測されたバグ報告数から得られる情報の方が適合性に対して強い影響を及ぼしていることを示唆している．

### (2) テストケースの入力情報を用いた信頼性評価

検証(2)では，テストケースの入力情報を用いたソフトウェア信頼性評価手法の有効性を検証した．モデルにはカーネル法を適用したロジスティック回帰によるソフトウェア信頼性モデルを用いた．特に，文字列に対するカーネルを応用して，テストケースの入力情報とフォールト検出確率を関連させた手法を適用する．検証に使用するデータは，SIR:Software-artifact Infrastructure Repository(http://sir.unl.edu)で提供されている文字列変換プログラムのreplaceのソースコードとテストケースを利用して生成した．replaceではあらかじめ5,542個のテストケースが準備されており，人工的にバグを埋め込んだプログラムに対して，5,542個のテストケースを利用して疑似的なテストデータを生成した．

カーネル法を適用したロジスティック回帰によるソフトウェア信頼性モデルでは，テストケースに対する類似度（内積，カーネル）を求める必要がある．ここでは，類似度にテストケースの編集距離に基づいた手法を適用する．さらに，投入されるテストケース数はテスト進捗状況によって変わるため，検証では次の状況を考えた．

- Case1:50個のテストケース投入が完了した時点で，50個のテストケース入力から得られるカーネルでフォールト検出確率を表す．

パラメータ推定にはL2ノルムの二乗による罰則付き最尤法とABICによる正則化（罰則）パラメータの決定を行った．

図2はCase1において，ABICの基準により決定された正則化パラメータを用いた場合にモデルから算出した累積バグ数の平均値（平均値関数，赤線）と実際に観測された累積バグ数（緑線）を表している．

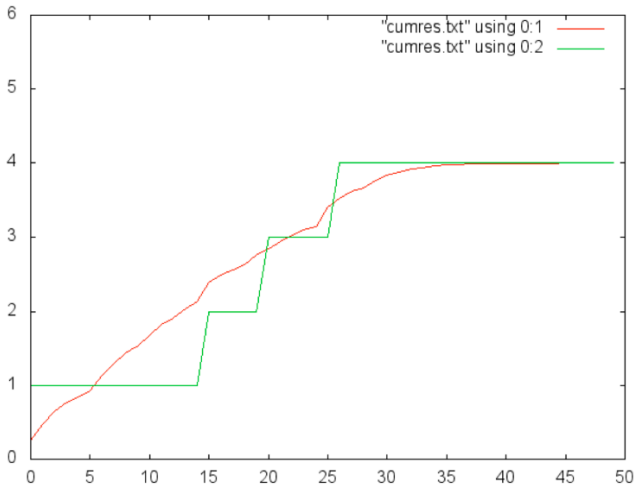


**図 2：累積バグ数と推定した平均値関数（Case1）**

この図から，テストケース入力を用いたカーネル法を用いても，与えられたデータにフィットするモデルを得ることができたことが確認できる．

### (3) ソースコード情報を用いた信頼性評価

ソースコード情報を用いたソフトウェア信頼性評価手法の有効性を検証した．ここでは静的メトリクスに対してポアソン回帰を用いたソフトウェア信頼性モデルを基にして，カーネル関数を用いたソースコード情報の取り扱いについて調査した．

検証(3)には，検証(1)で言及したASFのTomcat6プロジェクトにおけるCatalinaモジュール中のソースコード（31ファイル）を用いた．また，コードクローン検出ツールCCFinderが出力する二つのプログラム間の共通するクローンセット数によって類似度を定義した．また，ここでの検証ではテスト進捗によるバグ検出数のデータとして，SVNリポジトリに記録されている各ファイルの修正履歴からファイルの修正(modify)の回数を記録した．検証(2)と同様にパラメータ推定は，L2ノルムの二乗を用いた罰則項を用い，正則化（罰則）パラメータλと決定にはABICを用いた．

図3はカーネル法を適用したポアソン回帰によるソフトウェア信頼性モデルを用いた推定結果である．

| | NHPP | | | Kernel (CCFinder) | | | No. faults |
|---|---|---|---|---|---|---|---|
| | Total faults | Residual faults | FFP | Total faults | Residual faults | FFP | |
| ApplicationContext.java | 19.0429 | 2.0429 | 0.1297 | 19.0455 | 2.0455 | 0.1293 | 1 |
| ApplicationContextFacade.java | 8.1241 | 1.1241 | 0.3250 | 8.1261 | 1.1261 | 0.3243 | 24 |
| ApplicationDispatcher.java | 17.4777 | 0.4777 | 0.6202 | 17.4779 | 0.4779 | 0.6201 | 11 |
| ApplicationFilterChain.java | 11.4338 | 0.4338 | 0.6481 | 11.4340 | 0.4340 | 0.6479 | 8 |
| ApplicationFilterConfig.java | 12.0952 | 1.0952 | 0.3345 | 12.0965 | 1.0965 | 0.3340 | 4 |
| ApplicationFilterFactory.java | 11.0313 | 0.0313 | 0.9692 | 11.0313 | 0.0313 | 0.9692 | 14 |
| ApplicationHttpRequest.java | 9.6243 | 2.6243 | 0.0725 | 9.6377 | 2.6377 | 0.0715 | 3 |
| ApplicationHttpResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 3 |
| ApplicationRequest.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| ApplicationResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| AprLifecycleListener.java | 36.8628 | 4.8628 | 0.0077 | 36.8708 | 4.8708 | 0.0077 | 2 |
| Constants.java | 6.3282 | 0.3282 | 0.7202 | 109.1404 | 103.1404 | 0.0000 | 0 |
| ContainerBase.java | 10.6579 | 0.6579 | 0.5179 | 10.6584 | 0.6584 | 0.5177 | 118 |
| DummyRequest.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| DummyResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 3 |
| JasperListener.java | 6.6377 | 0.6377 | 0.5285 | 109.1538 | 103.1538 | 0.0000 | 0 |
| JreMemoryLeakPreventionListener.java | 317.3526 | 301.3526 | 0.0000 | 109.7239 | 93.7239 | 0.0000 | 0 |
| NamingContextListener.java | 14.9339 | 0.9339 | 0.3930 | 14.9344 | 0.9344 | 0.3928 | 88 |
| StandardContext.java | 58.6706 | 3.6706 | 0.0255 | 58.6726 | 3.6726 | 0.0254 | 294 |
| StandardContextValve.java | 10.5598 | 0.5598 | 0.5713 | 10.5602 | 0.5602 | 0.5711 | 2 |
| StandardEngine.java | 15.8002 | 4.8002 | 0.0082 | 15.8276 | 4.8276 | 0.0080 | 6 |
| StandardEngineValve.java | 6.5147 | 0.5147 | 0.5977 | 109.1494 | 103.1494 | 0.0000 | 0 |
| StandardHost.java | 16.4417 | 2.4417 | 0.0870 | 16.4463 | 2.4463 | 0.0866 | 40 |
| StandardHostValve.java | 12.7507 | 0.7507 | 0.4721 | 12.7512 | 0.7512 | 0.4718 | 4 |
| StandardPipeline.java | 7.0484 | 0.0484 | 0.9528 | 7.0484 | 0.0484 | 0.9528 | 4 |
| StandardServer.java | 14.6352 | 1.6352 | 0.1949 | 14.6372 | 1.6372 | 0.1945 | 13 |
| StandardService.java | 11.3825 | 0.3825 | 0.6822 | 11.3826 | 0.3826 | 0.6821 | 15 |
| StandardThreadExecutor.java | 21.6689 | 8.6689 | 0.0002 | 21.7568 | 8.7568 | 0.0002 | 6 |
| StandardWrapper.java | 27.4471 | 0.4471 | 0.6395 | 27.4471 | 0.4471 | 0.6395 | 25 |
| StandardWrapperFacade.java | 6.6081 | 0.6081 | 0.5444 | 109.1528 | 103.1528 | 0.0000 | 0 |
| StandardWrapperValve.java | 12.2168 | 0.2168 | 0.8051 | 12.2169 | 0.2169 | 0.8050 | 45 |

**図 3：ソースコード情報を用いた信頼性評価結果.**

この図ではNHPPモデルで評価した際の各ファイルの総バグ(修正)数，残存バグ(修正)数，およびFFP(fault-free probability)と，カーネル法を適用したポアソン回帰によるソフトウェア信頼性モデルで評価した際の各ファイルの総バグ(修正)数，残存バグ(修正)数，およびFFPを示している．また，このときの正則化パラメータは$\lambda = 0.0$となっている．正則化パラメータが0の場合NHPPモデルの結果と同じになることが解析的にわかっているため，NHPPモデルによる評価と大きく変わらない結果となっている．その一方で，幾つかのファイルにおいて推定値が大きく変化しており(赤字で表示)，ソースコード情報が何らかの影響を与える結果となった．

## 研究目標４「モデルによる信頼性評価を実装したツールの開発」

評価モデルと推定アルゴリズムを実装したツールの開発を行った．開発したツールはExcelとRによるインタフェースをもつ．また，作業は推定アルゴリズムとインタフェースの実装に分割される．推定アルゴリズムの実装はCで行い，共通ライブラリ化することで，Excel，Rの双方から利用できるようにした．本報告書では，Microsoft ExcelのAddInとして作成したツールをMSRATS (Metrics-based Software Reliability Assessment Tool on Spreadsheet)，統計処理ツールRのパッケージとして開発したツールをRsrat (Software Reliability Assessment Tool on R)と呼称する．

### (1) MSRATS の概要

MSRATSはC#を用いたExcel AddInとして開発され，後述するRsratが大量のデータ解析を行う研究者向けであるのに対して，ユーザの使いやすさに主眼をおいた設計となっている．MSRATSは次の特徴を持つ．

- Excelのスプレッドシートから時刻データあるいは個数データの入力
- 11種類の典型的なNHPPモデル，動的メトリクスを扱うロジスティック回帰によるソフトウェア

信頼性モデル，静的メトリクスを扱うポアソン回帰によるソフトウェア信頼性モデル，一般化線形ソフトウェア信頼性モデルのパラメータ推定および信頼性尺度計算の自動化

- Excelのグラフ描画機能を利用した信頼度関数などのグラフ描画

次のソフトウェア信頼性評価尺度をモデルに基づいて算出することができる．

- 予測総バグ数：単一のソフトウェアモジュール（プログラム単位orシステム単位）に内在する総バグ数の予測値．
- 予測残存バグ数：単一のソフトウェアモジュール（プログラム単位orシステム単位）に現時点で残存しているバグ数の予測値．
- ソフトウェア信頼度関数：一定期間中にバグが発見されない確率．
- FFP(fault-free probability)：現在のモジュールにバグがない確率．
- 各種MTTF(mean time to failure)，累積MTTF，瞬間MTTF，条件付きMTTF：バグ発見までの平均時間．
- Median，Betenlife：信頼度が0.5，0.1になるまでの時間．

### (2)Rツールの概要

RsratはR言語を用いた統計処理ソフトウェアRのパッケージとして開発した．ユーザがR言語を用いて拡張することができるため，実験的なデータ解析や大量のデータ解析を行う研究者に向いている．Rsratは次の特徴を持つ．

- Rのデータフレーム形式を利用した時刻データあるいは個数データの入力
- 11種類の典型的なNHPPモデル，動的メトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデル，静的メトリクスを扱うポアソン回帰によるソフトウェア信頼性モデル，一般化線形ソフトウェア信頼性モデルのパラメータ推定および信頼性尺度計算の自動化
- R言語による拡張が可能

算出されるソフトウェア信頼性評価尺度および扱えるモデルはMSRATSと同じである．また，カーネル法の適用に関してもカーネル値を要因として入力し，正則化最尤法を適用する．

## 4. まとめ

本研究では,【モデル構築】,【推定アルゴリズム開発】,【ツール開発】,【実証分析】という一連の研究・開発作業を通じて, 従来までのソフトウェア信頼性評価技術の問題点を克服し, さらに開発管理情報の有効的な利用を前提とした次世代ソフトウェア信頼性評価技術を開発した. 信頼性評価尺度の推定精度の向上ならびに推定労力を軽減するためのツール化を実現し, その効果を実際のプロジェクトデータを用いて検証した.

本研究で提案した一般化線形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. つまり, メトリクス情報は適合性に大きく貢献することがわかった. さらに, 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)が, ソフトウェアそのもののメトリクス(コード行数や複雑性)よりも強く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が確認できた.

カーネル法を適用した手法では, これまでに定量化することが難しかった情報を信頼性評価に適用するための基本的枠組みを整備することができた. カーネル関数によって, テストケース入力情報やソースコード上の文字列情報を直接的にフォールトの検出確率に関連づけたモデル化はこれまでに報告されておらず, その意味で, カーネル回帰に基づいた信頼性評価技術は画期的な方法と言える. また, 通常の確率・統計モデルにおいて任意パラメータを推定し, 最適なモデルを選択した場合と比較しても遜色のない成果を与えることができた. 一方で, ここで行った検証からはテスト情報やソースコード情報を活用することで特段適合性の高いモデルを得ることはできなかった. テスト入力情報の違いや距離の定義に関する違い, ソースコードの静的解析において同定可能な類似性や構造を特徴づけるパラメータにより結果が異なるため, 使用するカーネルを包括的に調査し, テスト入力情報やソースコードに適したカーネルの構築が課題として考えられる. また, このようなアプローチは開発の各段階に参加する技術者の信頼性評価に関するコンセンサスをとる上で重要な技術であり, 今後急速に発展する新しい研究分野であると言える.

ソフトウェア信頼性評価はテストの進捗状況把握と出荷判定の際に行われるべき活動である. レビューなどの静的テストやテストケースを投入しながらの動的テストの現場では, 一体どれくらいのレビューやテストケース準備を行えば良いかの理論的な指針がほとんどない. そこで, 通常は納期とコストを考慮し, テスト計画に合致するようテスト作業を進めることが一般的である. 還元すれば, 定量的な信頼性評価は実質的には行われておらず, フォールトが出尽くしたかどうかを判定する成長曲線の飽和状態だけを観測することでテスト進捗状況把握や出荷判定を行っているケースがほとんどであろう. 本研究で開発したツールを活用することで, テスト作業者がテストの進捗状況を手軽に把握することができるようになり, また開発管理の立場から製品の出荷判定の根拠となる説明資料として定量的信頼性評価結果を用いることが考えられる. 同時に, ユーザや市場に対するソフトウェア製品の品質に関する説明責任を果たす意味で, ソフトウェアの定量的信頼性を確保することは極めて重要な社会的意義を持つものと考える.

# 1 研究の背景および目的

## 1.1 背景

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「定量化」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」する取り組みは，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性モデルは，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評価するのに用いられてきた．このようなフォールトの計数データだけを用いた「バグモデル」や「ソフトウェア信頼度成長モデル」は，取り扱いが非常に簡便である．また，モデルから得られるソフトウェアの信頼度は「ある規定の条件下で規定の期間中にソフトウェア障害が発生しない確率」と定義され，正確な信頼度の見積もりができるのであれば開発者にとって非常に有益な情報となるため，いくつかの民間企業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を利用しているため，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の検証が難しい」，「管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」と言う重大な欠陥をもつ．

一方，現実的には，テスト網羅度や欠陥密度のような原因と結果が明確な可観測情報に基づいて，主観的な信頼性を算出する手法がとられることも多い．網羅度や欠陥密度は信頼性評価尺度のひとつではあるが，これは当該ソフトウェアの信頼性を正確に表現しているものではない．つまり，網羅度や欠陥密度を導出したとしても，確率・統計理論を用いない限り，網羅度・欠陥密度と定量的ソフトウェア信頼性の因果関係において妥当性を検証することが難しい．

従来のソフトウェアの定量的信頼性評価では，テスト工程で発見されたフォールトの個数情報のみから「残っているバグの個数」などを推定する試みがなされている．これは，「発見されたフォールト個数」という非常に簡単な情報であるため，ソフトウェアの種類にかかわらず，あらゆるソフトウェア開発に対して適用されてきた経緯があり，抽象化されたソフトウェア信頼性モデルの一つの利点であると考えられる．一方で，従来のソフトウェア信頼性技術の啓蒙活動が必ずしも成功していないひとつの原因は，実際の開発現場において「信頼度を左右する要因を明らかにしたい」と言う要求分析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．つまり，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った．このため，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の代名詞であり，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も少なくない．

本研究では，このような抽象化・汎用化に対する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，より詳細なデータ（統計量）を扱えるモデルへと拡張を行う．開発現場で解決すべき課題は，「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作成（出荷）できるか」であり，その第一歩として，設計手法・テスト技法と結果として得られたソフトウェアの信頼度が統計的にどのような因果関係を持つかを分析する．加えて，本研究課題では，新たな「分析手法」を提案することに主眼を置いている．これまでのメトリクス計測を対象とした研究では，ある特定種別

のソフトウェアでの「分析結果」が重要になることが多く，その点が本研究課題とメトリクスを分析する研究との大きな違いとなる．換言すれば，分析手法を特定の産業だけで使うのではなく，ソフトウェア信頼度成長モデルが持つような，エンタープライズ系，組込み系を通じて利用できる「汎用性」を有し，メトリクスの情報を利用することのできるモデルを構築することで，広く産業界に普及する統一的手法の確立が必要となる．

ソフトウェア信頼性理論が提唱されて 40 年以上が経過した現在においても，ソフトウェア内に残存するフォールトが存在しないことを理論的に証明する手段がないため，定量的ソフトウェア信頼性評価は未解決の課題であると言える．その理想に少しでも近づくためには，ソフトウェア工学で長年醸成された実証ソフトウェア工学の知見と，高水準な確率・統計理論を駆使したソフトウェア信頼性評価技術を融合する新しい技術の確立が必要とされる．「銀の弾丸はない」ようにソフトウェア信頼性評価にも王道はなく，開発現場で得られる可観測な情報を有機的に活用することで，高精度かつ合理的な信頼性評価を実施することが高信頼化ソフトウェアの開発という理想に近づくための近道となる．

## 1.2 研究課題

本研究では，開発現場で獲得し得る情報水準に応じて信頼性評価の方法を分類する次世代のソフトウェア信頼性評価技術の体系化を目指す．最も重要なポイントは「信頼性評価の妥当性検証を可能にすること」と「管理技術，開発手法へフィードバック可能な情報を与えること」である．これを実現する本質は「信頼性と可観測な要因の因果関係を明らかすること」である．つまり，開発情報とソフトウェア信頼度の相関をモデル化することで確率・統計に基づいた要因分析と，確率・統計に基づいた従来のソフトウェア信頼度の見積もりが同時に可能となる．しかし，一般的には，開発情報に関するメトリクスの計測にはコストがかかるため，本研究では，各々の現場で獲得可能な情報水準に基づいた信頼性評価体系を提案する．具体的には，3 種類のデータ(a)ソフトウェアメトリクス，(b)テストケースの情報（テスト入力，テスト実行パスなど），(c)ソースコードの情報（制御依存グラフ，データ依存グラフなど）と，テスト工程で観測されるフォールト計数データに対する関係をモデル化することで，各々のデータを有機的に利用するソフトウェア信頼性モデルおよびモデルに基づいた各種信頼性評価尺度の導出に関する方法論を開発する．

フォールト計数データのみに基づいた従来のソフトウェア信頼性評価技術と本研究で提案するソフトウェア信頼性技術の大きな違いは，開発情報の利用法にある．すなわち，(a)開発規模，テスト労力，網羅度のような既に要約されたメトリクスデータだけが利用可能であれば，回帰構造を確率モデルに組み込むことで，信頼性評価モデルを記述することができる．一方，(b)テストケースの詳細な情報が利用可能であれば，テストケースの距離空間をテスト入力と実行パスから定義し，ミクロなフォールト位置情報とマクロなフォールト検出事象をモデル化すればよい．さらに，(c)ソースコードを直接利用できるのであれば，ソースコードから要約されたメトリクスを抽出する前に，ソースコードから静的に分析して得られる依存グラフとフォールトの作り込み・発見現象をモデル化した信頼性評価が可能である．

このように，開発現場で成し得る情報計測の様々な階層に対応した信頼性評価技術を実装し，信頼性と要因の因果関係を明らかにすることで，ここで開発される信頼性評価技術を利用する開発者は信頼性評価のために何をなすべきかを容易に知ることができ，管理・開発フェーズに結果をフィードバックできるというメリットがある．

他方，信頼性評価技術は秘匿性のある固有技術というよりも，全ての人が共通して利用できるオープン技術として捉えられるべきである．そのため本研究では，上述の新しい評価技術をフリーツールとして実装し，産業界に無料で提供する．これにより，信頼性評価技術の啓蒙と信頼性評価技術のさらなる向上に繋がるものと考えている．

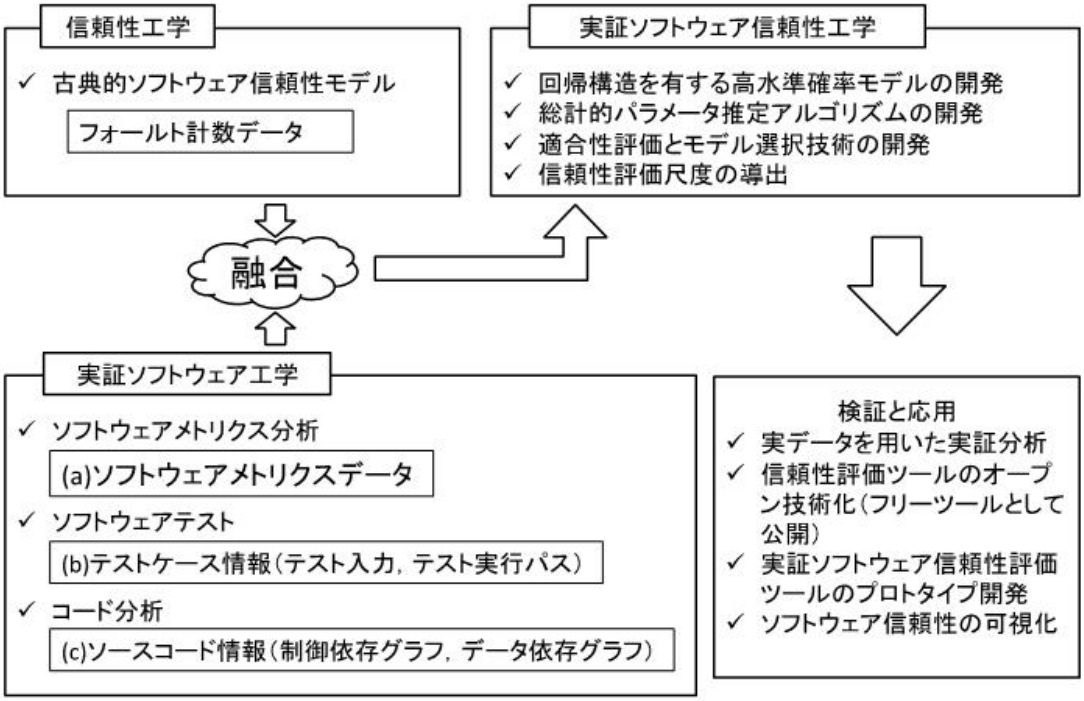


図 1-1：研究課題の概念図

## 1.3 研究の意義

ソフトウェアの定量的信頼性を正確に評価することは，ソフトウェア自身に高い信頼性を作り込むことと同程度に困難である．ソフトウェア信頼性工学が提唱され，40 年以上の歳月が経過した現在においても，ソフトウェアの定量的信頼性評価は未解決の問題である．一方，テスト工程においてフォールト検出過程を記述するために，数多くのソフトウェア信頼性モデルが提案されている．一見，簡単なデータフィッティングに見えるソフトウェア信頼性モデルでも，統計的推定，仮説検定，モデル選択，予測評価，信頼度計測という一連の作業を行わなければならず，開発現場で容易に利用可能な信頼性評価技術の啓蒙活動があまり行われていないのが現状である．

国立大学法人広島大学で開発した表計算ソフト上で駆動する信頼性評価ツール SRATS(Software Reliability Assessment Tool on Spreadsheet)は，ホームページ上から無料でダウンロードでき，一般的に業務で利用する Microsoft Excel 上で動作できる手軽さから国内における多くの企業の開発現場で使用されている．また，同大学ではメトリクス情報による信頼性評価を行うツール（M-SRAT，PI-SRAST）の提供も行っている．数理モデルに基づいた信頼性評価をツールとして実装することは，ユーザが難解な理論を理解する負担を軽減することができるため，ソフト

ウェア信頼性評価を啓蒙する上でも，簡単に利用できる信頼性評価ツールの普及が今後益々重要視されるものと考えられる．

現在，このようなツールは統合テスト，システムテストにおいて主に利用されているが，開発工程で観測される多種多様な情報を定量的な信頼性評価に有機的に活用するという試みはあまりなされていない．本研究では，3 種類のデータ(a)ソフトウェアメトリクス，(b)テストケースの情報（テスト入力，テスト実行パス），(c)ソースコードの情報（制御依存グラフ，データ依存グラフ）と，テスト工程で観測されるフォールト計数データに対する関係をモデル化することで，3 種類のデータを有機的に利用するソフトウェア信頼性モデルおよびモデルに基づいた各種信頼性評価尺度の導出に関する方法論を開発する．(a)はM-SRAT，PI-SRASTで実装されたメトリクスベース・ソフトウェア信頼性評価技術を回帰分析の観点から大幅に拡張し，効率の良いパラメータ推定アルゴリズムを開発することで計算コストの削減と推定精度の向上を目指す．(b)は，ソフトウェアテストケースの距離空間を定義し，実行したテストケース間の「距離」を計測することでテストとフォールトの位置情報を関係づけ，それをソフトウェア信頼性評価に融合する試みである．テスト実績を信頼性評価に反映するという意味でこれまでにない新しい信頼性評価技術であり，学術的な価値も高い．(c)は，ソースコードの構造データに着目する．ソースコードはソフトウェアを構成する細粒度の情報であるが，ソースコードを直接利用する信頼性評価は，過去 40 年間の信頼性工学の歴史上，誰も発案しなかった方法である．さらに，ソースコードの構造データと信頼性の関係を明らかにすることで，実証ソフトウェア工学で得られている経験的な知見，例えば，ソースコード上のクローン（類似または一致した部分）がソフトウェア障害の原因となり得るという実証的知見などを説明可能な枠組みとして期待される．

上述の(a)，(b)，(c)で列挙した各々の信頼性評価技術は異なる情報量の下で相互補完的に利用される技術であり，獲得できる情報に応じてそれぞれをツール化することは非常に有益である．加えて，SRATS の例にあるように，開発現場においては日常的に利用するソフトウェア（Microsoft Excel）の拡張機能（AddIn）として開発することは，本研究で体系化する信頼性評価技術を産業界に広く普及させ，エンタープライズ系，組込み系に関わらず多くのソフトウェア開発現場で，高い精度のソフトウェア信頼度推定と信頼性を向上させるための要因特定を実施するための重要な施策である．加えて，実証ソフトウェア工学分野では，統計ソフトウェア R を用いることが多く，R のパッケージとして作成することは，実証ソフトウェア工学との両輪としてさらなる研究発展につながることが期待される．

このようなアイデアを実現することによって，従来から行われてきたフォールトの計数情報だけに基づいた信頼性評価技術を飛躍的に改善することが可能である．加えて，開発現場で信頼性評価が広く普及することと，実証ソフトウェア工学からの学術的な発展がツール提供によって可能になることは，次世代信頼性評価における基盤技術を我が国から発信することにつながるものと考えている．

## 1.4 期待される効果

本研究によって期待される成果の中で，「定量的な信頼度とその要因の因果関係が明らかになる」ことが最も重要な点としてあげられる．これは，得られる定量的な信頼性の妥当性を議論する上でも重要である．これまでブラックボックス化されていたソフトウェア信頼度が，適切なモデル化によって主たる要因が明らかになることは算出される尺度の「信用」に大きく貢献する．一方で，管理や開発側にとっての大きな利点は，信頼度を左右する要因が特定されることによって，どこをテストすれば良いのか，どの設計を見直せば良いのか，どこに人員を重点的に割り当てれば良いのかなど，開発工程へのフィードバックが明確になる点にある．さらには，そのようなノウハウを蓄積することで，類似したプロジェクトあるいは未踏のプロジェクトに対しても高信頼化に重点をおいた管理・計画を行うことが可能となる．

ソフトウェアの信頼度と要因の因果関係を明らかにする試みは，これまでにも多く行われており，その結論は，ソフトウェアの種別や開発規模などにより主要因が異なるというものであった．本研究で特に強調したい点は，固有ソフトウェア種別に対する要因特定を目的とするのではなく，どのような種別に対しても適用可能な「普遍的なアプリケーション」としてのモデルと推定手法を提供することにある．これは，今後どのような利用形態が現れるか予測不可能なソフトウェア分野において，高い信頼性を確保するために重要な点である．

さらには，それらを既存ツールの拡張機能として提供することで，現場における信頼性管理の「キラーアプリケーション」として普及させることができる．それによって，それぞれの企業で定量的な信頼性とその主要因を継続的に管理することが容易になり，改善活動へとつながる．このスパイラルは，国内企業におけるソフトウェア品質を飛躍的に高める効果が期待される．

## 1.5 関連研究

ソフトウェアの開発工程において信頼性を作り込むためには，要求仕様，設計，実装，テストの段階で様々な工夫を凝らす必要がある．図 1-2 は，ソフトウェア工学における諸領域とソフトウェア信頼性の相互関係を表した概念図である．ソフトウェア製品の開発に規定の開発モデルを当てはめることで，開発プロジェクトないし開発プロセスを管理することを目的とするならば，プロジェクトマネジメント(PM)の技法やプロセス管理の理論を適用することが望ましい．しかしながら，ソフトウェアの信頼性特性はシステムごとに大きく異なっており，既存の開発モデルに従って開発水準の向上を実現することで製品の信頼性が確保される保証はない．

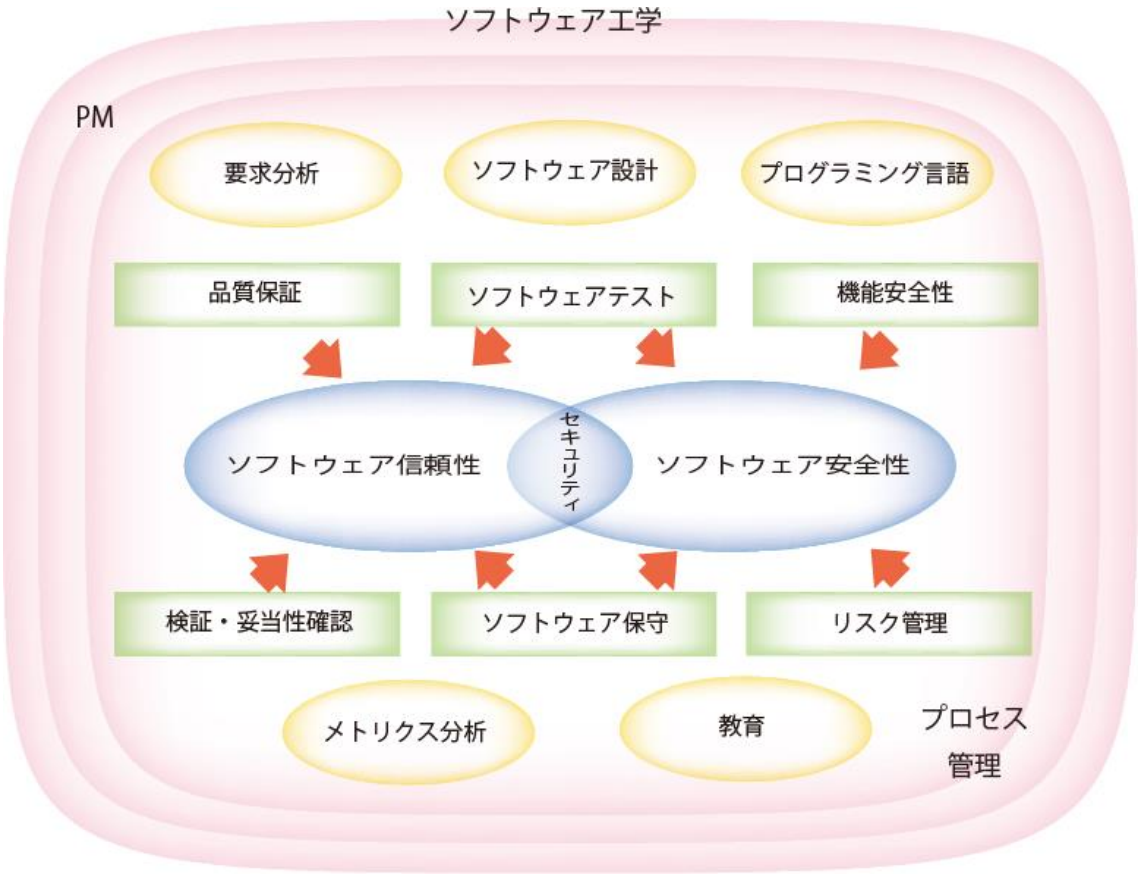


**図 1-2：ソフトウェア工学における諸領域とソフトウェア信頼性の相互関係**

すなわち，ソフトウェアの開発工程で信頼性を作り込むためには，信頼性に特化した要求分析・設計・言語選択・メトリクス分析・教育体系を意識しながら，品質保証・テスト・機能安全・検証妥当性確認・保守・リスク管理の観点から信頼性確保の問題に取り組む必要がある．換言すれば，バグを作り込むことを避けるために可能な限り高度な開発技術を駆使することにより，各フェーズにおけるベストプラクティスを実践する以外の王道はないと考えられている．反面，上記の事実は，極端に高い信頼性を確保するためには極端に高いコストが必要であることを示唆しており，コストの投入額に見合った信頼性を作り込み，保証することができるかが大きな問題となっている．

しかしながら，ハードウェア製品と異なり，ソフトウェア製品の信頼性を定量的に評価することは容易ではない．ハードウェア製品の場合，それが量産品であるならば，事後的に報告された不良品の件数に応じて信頼性を評価することは原理的に可能である．一方，ソフトウェア製品は完全に同一のものを複製することが可能であり，ひとつの生産工程においてひとつの製品しか開発されないため，ハードウェア製品と同様な評価をすることは困難である．しかも，要求仕様から設計，設計から実装の段階では，開発担当者の開発能力に大きく依存する形で開発作業が進められ，完全に自動化することは困難な状況にある．さらに，ソフトウェアテストにおいて全ての論理的誤りを検出・除去することは事実上不可能であり，仮にできたとしても非現実的な長さでかつ天文学的に大きなテスト労力を必要とする．

この意味において，ソフトウェア信頼性評価はソフトウェア信頼性工学における中心的な課題のひとつであり，1970 年代初頭から活発な研究が継続的に行われている．ソフトウェア信頼性の定量的評価尺度として「ソフトウェア信頼度」がある．これは，「ソフトウェアが規定の環境で規定の期間中意図する機能を実現することができる確率」として定義されており，ソフトウェア信頼度を導出するためには何らかの確率モデルを構築する必要がある．これまでに中心的な役割を演じてきたのは，所謂，「ソフトウェア信頼性モデル（もしくは，単にバグモデル）」であった．実際に 70 年代初頭から，ソフトウェアの信頼性を定量的に予測する目的で，膨大な数の数理モデルが提案されてきた．

一方，数理モデルをベースとしたソフトウェア信頼性評価技術は，当初の絶大な期待感とは裏腹に，産業界ではほとんど用いられていない技術の代表例として認識されているのが現状である．この理由としては，無意味なモデルの開発競争や誤った統計技法の氾濫などの学術サイドの問題や，ソフトウェア信頼性モデルの効用と限界を理解することなく誤った認識の上で信頼性評価を志向していた実務サイドの問題が挙げられる．信頼性モデルに基づいたソフトウェア信頼性評価において最も実務家サイドから受け入れられなかった点は，多くのソフトウェア信頼性モデルによる評価はテストで報告されたバグ情報（検出バグ数などの統計情報）だけに依存しており，要求分析・設計・コーディングスキルだけでなくどのようなテストが行われてきたかのテスト情報さえも考慮しておらず，開発での労力や工夫が全く信頼性評価には反映されていないという批判であろう．

上述の問題に対しては，様々な解決方法が考えられる．すなわち，開発段階における各フェーズで観測されるメトリクス情報をフォールト検出時間分布（故障時間分布）上で表現することで，複数の情報をソフトウェア信頼性モデルに取り込むことは可能である．ただし，このような方法はテスト工程で得られた各種メトリクスに関しては有効であるものの，要求仕様書や設計書の情報を活用するための方法論は確立されていない．

# 2 実施内容

## 2.1 研究アプローチ

### 2.1.1 研究の全体像

本研究内容は,【モデル構築】,【推定アルゴリズム開発】,【ツール開発】,【実証分析】からなる.

【モデル構築】

ソフトウェアメトリクスとフォールト計数過程を結びつけるモデルの構築を行う. 特に, 従来のソフトウェア信頼度成長モデルと, 実証ソフトウェア工学でも頻繁に用いられる標準的な回帰手法(Cox, ロジスティック, ポアソン回帰)を融合したモデルの構築を目指す. 一般的に, 良いモデルではモデルの「簡潔さ」すなわち「汎化能力」が重要である. そのため本研究では, 特殊な状況や複雑なモデルを考えることを避け, 従来からある基本的なソフトウェア信頼度成長モデルと一般化線形回帰を自然な形で融合したモデル化の枠組みを体系化する. この手法は, ソフトウェア開発形態や環境毎にモデルを考える必要がなく, 汎用的にどのようなソフトウェア開発形態や環境に対しても適用可能であるという利点をもつ.

さらに, 本研究ではテストケース情報(テスト入力, テスト実行パス)とソースコード情報(制御依存グラフ, データ依存グラフ)の構造データを扱う. そのため, 上述の回帰構造に対して文字列データおよびグラフデータを扱えるカーネル法の適用を行い, 入力するデータの構造に関する汎用化も目指す.

特に, 広島大学が既発表しているCox回帰, ロジスティック回帰, ポアソン回帰と信頼度成長モデルを融合したメトリクスベースの信頼性評価モデルを「簡潔さ」の観点から再構築する. 具体的には, 一般化線形モデルの枠組みによるモデルの統一化を検討する. 一般化線形モデルは, リンク関数, 確率分布で構成される線形回帰モデルを拡張した考え方であり, ロジスティック回帰, ポアソン回帰を特別な場合として含む. ここでは, Cox 回帰の一部も一般化線形モデルと結びつけることで, 一般化線形モデルと信頼度成長モデルを融合する枠組みの構築を行う. これによってモデルの汎用性と汎化能力を確保することができる. また後述するように, 推定アルゴリズムに対する統一化や, 評価尺度算出アルゴリズムの統一化が可能となり, ツール開発時の省力化に貢献する.

次に, 一般化線形モデルで構造データを扱えるようにするためにカーネル法を適用したモデルの構築を行う. カーネル法は元の特徴空間を高次元空間に写像することで, 線形モデルで非線形データを扱えるように工夫された手法であり, SVM(Support Vector Machine), PCA(Principal Component Analysis)にも適用されている. さらに, 文字列やグラフといった構造データも扱うことができるため, 本研究で考えるテストケース情報やソースコード情報もカーネル法を経由することで, 一般化線形モデルによるメトリクスベースの信頼性評価モデルと同様な枠組みで扱うことができる. ただし, 単純な回帰と構造が異なるため, カーネル法の適用条件を十分に検討した上でモデルの構築を行わなければならない.

【推定アルゴリズム開発】

上述するモデルはメトリクスあるいは構造データを扱うため，従来のソフトウェア信頼性モデルよりもパラメータ推定が難しくなる．また，原理的には回帰モデルと同じ構造を適用するが，確率モデルの構造自体が標準的な回帰モデルと異なるため，標準的な回帰手法で用いられる推定アルゴリズムを直接適用することができない．そこで，効率的なパラメータ推定アルゴリズムを新しく開発する必要がある．特に，テストケース情報，ソースコード情報を直接扱う場合，膨大なデータ量となるためスケーラブルなパラメータ推定アルゴリズムが必要となる．本研究では，これらの問題に対し，EM (Expectation-Maximization) アルゴリズム，階層ベイズ法の適用を考え，数十万行数からなる標準的な規模のオープンソースプロジェクトに対して適用可能なアルゴリズム開発を目指す．

一般化線形モデルによるメトリクスベースの信頼性評価モデルは，従来の Cox 回帰，ロジスティック回帰，ポアソン回帰で開発されたEMアルゴリズムを改良し，一般化線形モデルへ適用できるよう拡張する．また，要因選択や交互作用に関して分析する手法（ステップワイズ法など）の検討とそのアルゴリズムの詳細化を行う．一方，カーネル法を適用したモデルでは，一般的に縮小推定が用いられる．そのため，罰則付き最尤法や階層ベイズと MAP(Maximum A Posterior)推定の適用を検討する．さらに，カーネル法によって，テストケース情報やソースコード情報を取り扱うためには数千から数十万規模のデータを扱う必要があるため，推定アルゴリズムのスケーラビリティについても十分に検討する必要がある．その問題に対しては，損失関数を直接的に最小化する手法だけでなく，SVM で行われているような凸二次計画法の利用についても検討する．

【ツール開発】

産業界での利用を前提とし，上述したモデルおよび推定アルゴリズムを実装したツールの開発を行う．特に，開発現場での利用形態を考慮すると，新たなツールとして構築するよりも，現存するソフトウェアの拡張機能として提供する方が使用に対する敷居が低い．本研究では，2 種類の既存ソフトウェア（Microsoft Excel，R）の拡張機能として，信頼性評価ツールの提供を行う．

ツール開発は，推定アルゴリズムを実装する作業グループとExcel，R によるインタフェースを実装する作業グループに分かれる．実装は C 言語で行い，共通ライブラリ化することで，Excel，R の双方から利用できるようにする．Excel では，VBA（Visual Basic Application）で記述された既存ツールである SRATS を元として，C#による再記述を行う．また，R に対しては新規パッケージとして構築し，R のパッケージを管理するサイトであるCRAN(Comprehensive R Archive Network)へ投稿可能な標準的な記述で実装する．開発は，一般化線形モデルに関する機能を先に開発し，カーネル法に関する機能を追加する形でインクリメンタルな開発を行う．

【実証分析】

開発したツールを利用して，実際の開発で計測されたテストデータおよび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実証分析を行う．ツール開発において，一般化線形モデルとカーネル法に関する機能がインクリメンタルに開発されるため，実証分析においても，一般化線形モデルに関する機能を用いた実証分析を先に行い，その後，カーネル法に関する機能を用いた実証分析を行う．また，一般化線形モデルとカーネル法では利用するデータ（メトリクス）が異なるので，それぞれの実証分析前にデータ整理を行う必要がある．

(a)のソフトウェアメトリクスを利用するモデルでは，メトリクス情報として，テストケース数，テスト労

力，テストカバレッジ情報，バグレポート枚数，ソフトウェア複雑性メトリクス，コード行数を想定している．これらの要因分析により，信頼性に対する主要因の分析などを行い，モデルの有効性と妥当性について検証する．

(b)のテストケース情報を利用するモデルでは，予め準備されたテストベッドに対して，各テストケースの入力とテスト実行パスを計測する．その後，テストケース間の入力と実行パスから計算される「距離」をカーネルとすることで，テストケース情報とフォールト検出の相関を一般化線形モデルで表現し，各テストケースのフォールト検出確率などを推定することができる．特に検証では，3 種類の実行パス間距離（C0 カバレッジによる距離，ステートメントの実行回数による距離，C1 カバレッジによる距離）を導入し，カバレッジとフォールト検出能力の関係を明らかにする．

(c)のソースコード情報を利用するモデルでは，ソースコードから PDG(Program Dependence Graph)を生成し，グラフ構造データに対してたたみ込みカーネルを適用することで，(a)の一般化線形モデルによるソフトウェア信頼性モデルと同様な枠組みで信頼性とソースコードの因果関係を分析する．また，より簡便な手法として，大阪大学大学院で開発されたコードクローン検出ツール CCFinder を利用する手法についても検証する．そこでは，CCFinder によって解析されたトークン列と文字列を扱う p-スペクトラムカーネルの導入，さらに，クローン箇所に関する情報を直接カーネルに代入し，クローンとフォールト発生の関係に関する検証も行う．

### 2.1.2 研究目標

#### (1) 想定する仮説など

本研究課題は，「定量的なソフトウェア信頼性と可観測なソフトウェア情報の因果関係を明らかにする」ことにより，より高い精度での信頼性評価が可能になるという仮説に基づいている．同時に，その情報はソフトウェア開発者に対して信頼性評価のために何をなすべきかという情報を提供することも想定している．特に，可観測なソフトウェア情報として，3 種類のデータ(a)ソフトウェアメトリクス，(b)テストケースの情報（テスト入力，テスト実行パス），(c)ソースコードの情報（制御依存グラフ，データ依存グラフ）が，テスト工程で観測されるフォールト計数データに大きく寄与していることを想定している．

#### (2) 到達目標の設定

本研究の目的は「信頼性評価の妥当性検証」および「信頼性の側面から管理技術，開発手法へフィードバック可能な情報の提供」ができる信頼性評価体系の構築である．その本質は「定量的なソフトウェア信頼性と可観測なソフトウェア情報の因果関係を明らかにする」ことを意味している．これを実現するためには「信頼性と可観測な要因の因果関係を明らかする」必要がある．つまり，開発情報とソフトウェア信頼度の相関をモデル化することで，確率・統計に基づいた要因分析と従来のソフトウェア信頼度の見積もりが同時に可能になるという仮説の下で，各々の現場で獲得可能な情報水準に基づいた信頼性評価体系を提供する．具体的には，3 種類のデータ(a)ソフトウェアメトリクス，(b)テストケースの情報（テスト入力，テスト実行パス），(c)ソースコードの情報（制御依存グラフ，データ依存グラフ）と，テスト工程で観測されるフォールト計数データに対する関係をモデル化することで，3 種類のデータを有機的かつ個別に利用するソフトウェア信頼性モデルおよびモデルに基づいた各種信頼性評価尺度の導出に関する方法論を開発する．開発現場で

成し得る情報計測の様々な階層に対応した信頼性評価技術を実装し，信頼性と要因の因果関係を明らかにすることで，当該信頼性評価技術を利用する開発者は信頼性評価のために何をなすべきかを容易に知ることができ，管理・開発フェーズに結果をフィードバックできるというメリットがある．

### (3) 到達目標に向けた研究目標の設定

上述の到達目標である「現場で獲得可能な情報水準に基づいた信頼性評価体系」の確立に向けて次のような研究目標を設定する。

研究目標1「メトリクスを扱う信頼性評価モデルの構築」:(a)ソフトウェアメトリクス，(b)テストケース情報，(c)ソースコード情報に基づいたソフトウェア信頼性評価のための確率モデルの解析を行う．現在までに行われてきた研究と新しい成果を組み合わせることで最良モデル（最も良いカーネルの構造など）の選定を行う．

研究目標2「パラメータ推定アルゴリズムの開発」:各々のモデル(a)，(b)，(c)で必要とされる情報(データ)から未知パラメータを効率良く推定するためのアルゴリズムを開発し，その安定性解析を実施する．

研究目標3「データを用いたモデルの有効性検証」:実際の事例に基づいた実証分析を通じて，提案技術の有効性を定量的に評価する．また，一般に公開されているベンチマークデータや開発データを再調査し，実証分析に利用可能なデータを分類・整理する．

研究目標4「モデルによる信頼性評価を実装したツールの開発」:Excel，R の拡張機能として，提案したモデルおよび推定アルゴリズムを実装する．

## 2.2 研究の活動実績・経緯

### 2.2.1 研究活動の実績

本研究活動の実績を図 2-1 に示す.

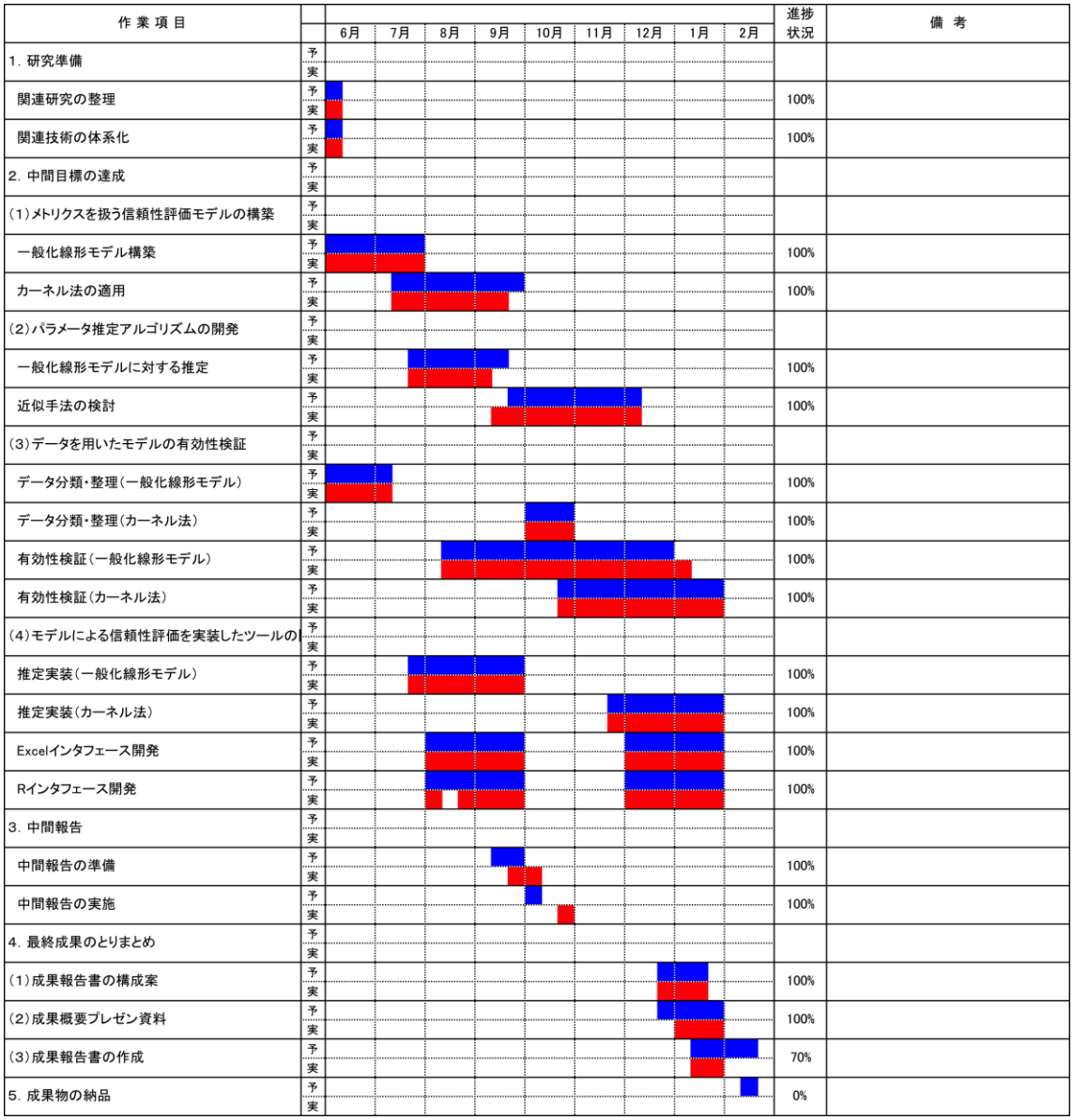


図 2-1. 研究活動の実績

### 2.2.2 研究の活動実績と経緯

作業グループごとに独立して研究・開発作業を行い，約 2 週間に一度の割合で定例ミーティングを開催し，研究内容に関する討論と進捗状況の管理を行った．以下に，月ごとの研究活動実績を整理する．

【６月】

過去に論文として発表された関連研究を 1970 年代まで遡って調査を行い，主として，ソフトウェア信頼度成長モデルに関する体系的分類とソフトウェア信頼度成長モデルを統一的に扱うモデルの分類を行った．一般化線形モデルは候補となるリンク関数の調査を理論統計学の文献を調査した．また、一般化線形モデルで利用可能なデータとしてオープンソースプロジェクトを対象とし，Apache Software Foundation(ASF)の Tomcat, Ant, log4j のリポジトリから，バージョン毎のソースコードの取得を行い，それらのメトリクス（コード行数，複雑度，など）を計測した．

【７月】

中間目標である「一般化線形モデル構築」について，一般化線形モデルで利用できるリンク関数の調査を行った．また中間目標「カーネル法の適用」について，一般化線形モデルに適用できるカーネル関数について調査を開始した．中間目標「データ分類・整理（一般化線形モデル)」について， Apache Software Foundation(ASF)の Tomcat, Ant, log4j のリポジトリから，バージョン毎のソースコードの取得を行った．中間目標「Excel インタフェース開発」,「R インタフェース開発」の作業開始に向けて準備作業の確認を行った．

【８月】

中間目標「カーネル法の適用」について，グラフのカーネルについてはカーネルを第一候補とした．中間目標「一般化線形モデルに対する推定」について，交互作用の問題については,実験計画法の適用を考慮することとし,変数選択には通常の変数増減法を適用し，実証分析の結果を見ながら検討した．

【９月】

「推定実装（一般化線形モデル)」,「Excel インタフェース開発」について，一般化線形モデルの推定モジュールのテストを行った．また，Excel インタフェースからの結合テストを開始した.「R インタフェース開発」について，R の関数の実装を八割程度完了させた．

【１０月】

Excel，R ともに，有効性検証を通じてテストを行った． 「有効性検証（カーネル法)」について，一般化線形モデルでカーネルを適用した場合の予備実験を行い，一般化線形モデルにおける結果と比較した．近似手法において，GPGPU による高速化は汎用性が低いため，並列化を行う際には CPU のコアを利用した手法を検討した．動的メトリクスとしてカーネル手法を適用するデータの収集に擬似的にテストを行いテストに関する詳細なメトリクスを計測した．

【１１月】

カーネル実装については，カーネル関数による類似度などの計算を，信頼性推定を行うRやExcelアプリケーションと独立させることで，カーネル関数の自由度をもたせることができるような設計にした.「推定実装（カーネル法)」について，推定アルゴリズムを実装完了させた．カーネル法を適用した手法のツールインターフェイスに関しての方針を検討した.

【１２月】

「有効性検証（カーネル法)」について，テストケース（テストパス）に対する文字列カーネルの適用を行った「推定実装（カーネル法)」について， $L_2$罰則項を適用したポアソン回帰モデルおよびロジスティック回帰の実装およびテストを完了させた.「Excelインタフェース開発」,「R インタフェース開発」について，罰則付き最尤推定を扱うインタフェースの開発を完了させた.「有効性検証（一般化線形モデル)」について，R のツールに基づいた一般化線形モデルを検証した.「推定実装（カーネル法)」について，フィッシャー情報行列の計算およびABIC（の近似）を用いたパラメータ推定アルゴリズムを実装した.

【１月】

「有効性検証（一般化線形モデル)」について，一部のデータに対する適合性検証を修正した.「Excelインタフェース開発」,「Rインタフェース開発」について，Rの機能を利用してブースティング（ランダムフォレスト）を実装した． Excelインタフェースのグラフ描画に関して実際の実行を通じて確認を行い，修正した.「Excelインタフェース開発」について，グラフ描画機能を実装した.

### 2.2.3 学会参加状況

研究期間を通じて，国内外で開催された研究会や国際会議に積極的に参加し，研究動向調査や技術動向調査を実施した．以下では学会などの参加状況について述べる.

【会議名：ソフトウェア・シンポジウム2013(SS 2013)】

場所：長良川国際会議場，岐阜市

主催：ソフトウェア技術者協会

日程：2013年7月7日～9日

参加者：岡村，肖

SS 2013 への参加を通じてソフトウェア工学における我が国における学会・産業界の取組について概観することができた．特に，ソフトウェアテスト，品質評価，定量化手法に関する最新の研究成果について学び，その成果を委託研究に応用できるかどうかについて詳細に検討した．具体的に，[02_研究論文]や[14_研究論文]に見られるように，オペレーションズ・リサーチの代表的な方法である数理計画法の適用方法に関するアイデア，[12_研究論文]で提案されているテスト実行パスから生成される決定表と仕様から作成する決定表の抽出方法，[10_研究論文]で提唱されているメトリクスの計測方法，[11_経験論文]

で提案されたテストケースの生成法を信頼性評価に組み込むアイデアなどは，次世代信頼性評価に適用することが可能であり，今後はその実装に関する可能性について検討した．

【会議名：The 37th Annual International Computers, Software & Applications Conference(COMPSAC 2013)】
場所：京都テルサ，京都市
主催：IEEE CS/IPSJ
日程：2013年7月23日～25日
参加者：土肥，岡村，羅
ソフトウェア工学分野において最も伝統のある国際会議であるIEEE COMPSACに参加し，ソフトウェアテスト，定量化手法に関する最新の研究成果を学び，それらを委託研究に応用できるかどうかについて調査した．COMPSAC 2013への参加を通じてソフトウェア工学における世界的な取組みについて概観することができた．特に，ソフトウェアテスト，定量化手法に関する最新の研究成果について学び，その成果を委託研究に応用できるかどうかについて詳細に検討した．具体的に，(i)ソフトウェア・アーキテクチャに関する多くの研究発表においてモデル化技法や可視化技術の進展について整理した，(ii)ソフトウェアテスト分野におけるテストオラクルの生成法やコード類似性の計測法，フォールト検出確率に基づいたテストケースの選択法について詳細に理解することができた．上記の知見はソフトウェア信頼性評価分野にも深く関連する事項であり，メトリクスベース信頼性評価への適用可能性の検討に大いに役立った．

【会議名：ソフトウェアエンジニアリング・シンポジウム2013(SES 2013)】
場所：東洋大学，東京都
主催：情報処理学会
日程：2013年9月9日～11日
参加者：土肥，岡村，肖
ソフトウェア工学分野において国内の研究者が一堂に会するSESで最新研究を調査し，それらを委託研究に応用できるかどうかについて検討した．SES 2013への参加を通じて，ソフトウェアテストや定量化手法に関する最新の研究成果について学び，その成果を委託研究に応用できるかどうかについて参加者三名それぞれの視点から分析し，詳細に検討した．具体的にはリポジトリの利用や，それらを用いた代表的なメトリクスの抽出，コードクローンの手法の進展について整理した，この知見は「有効性検証（カーネル法)」に反映させることができた．

【会議名：The Joint Conference of the 23rd International Workshop on Software Measurement(IWSM) and the 8th International Conference on Software Process and Product Measurement(Mensura)(IWSM/MENSURA 2013)】
場所：METU Culture and Convention Center, Ankara, Turkey
主催：IEEE
日程：2013年10月23日～25日

参加者：土肥，岡村

ソフトウェアメトリクスに関する国際会議において最新の研究を調査し，それらを委託研究に応用できるかどうかについて検討した．IWSM/MENSURA 2013 への参加を通じて，ソフトウェアメトリクスの最新研究成果について学び，その成果を委託研究に応用できるかどうかについて参加者二名それぞれの視点から分析し，詳細に検討した．具体的にはファンクションポイントの利用や問題点について整理した，この知見は「有効性検証（一般化線形モデル)」に反映させることができた．

【会議名：The 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)】

場所：The Westin Pasadena, LosAngeles, CA, USA

主催：IEEE

日程：2013 年 11 月 4 日～7 日

参加者：土肥，羅

ソフトウェア信頼性工学における最高水準の国際会議であり、最新の研究を調査し，それらを委託研究に応用できるかどうかについて検討した．また、委託研究の成果の一部を論文として発表した.ISSRE 2013 への参加を通じて,ソフトウェアメトリクスの活用方法,信頼性評価におけるツール化の実情に関する最新研究成果について学び，その成果を委託研究に応用できるかどうかについて参加者二名それぞれの視点から分析し，詳細に検討した．具体的には計測メトリクスの種類やリポジトリデータの活用に関する問題点について整理した，この知見は「有効性検証（一般化線形モデル)」に反映させることができた．

【会議名：The 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)】

場所：Crowne Plaza Cabana, SanJose, CA, USA

主催：IEEE/ACM

日程：2013 年 11 月 11 日～15 日

参加者：土肥，岡村

自動ソフトウェア工学に関する国際会議において最新の研究および関連するツールを調査し，それらを委託研究に応用できるかどうかについて検討した．ASE2013 への参加を通じて，現在のソフトウェア工学を支援するツールの最新研究成果について学び，その成果を委託研究に応用できるかどうかについて参加者二名それぞれの視点から分析し，詳細に検討した．特に，ソフトウェアマイニング手法は本研究課題におけるカーネル法と類似しており，ここで得られた知見は「有効性検証（カーネル法)」に反映させることができた．

【会議名：The 19th Pacific Rim International Symposium on Dependable Computing(PRDC 2013)】

場所：Marriot Pinnacle Downtown, Vancouver, British  Columbia, Canada

主催：IEEE

日程：2013 年 12 月 2 日～4 日

参加者：土肥，岡村

ディペンダブルコンピューティングに関する国際会議に参加し，それらを委託研究に応用できるかどうかについて検討した．PRDC 2013 への参加を通じて，ディペンダブルコンピューティングの最新研究成果について学び，その成果を委託研究に応用できるかどうかについて参加者二名それぞれの視点から分析し，詳細に検討した．特に，国外の研究者から一般化線形モデルに関する意見を収集した．ここで得られた知見は「有効性検証（一般化線形モデル)」に反映させることができた．

【会議名：電子情報通信学会信頼性研究会】

場所：機械振興会館，東京都

主催：電子情報通信学会

日程：2013 年 12 月 13 日

参加者：土肥，斎藤

国内における信頼性工学に関する主要研究会であり、ソフトウェア信頼性だけでなく他分野において研究されている最新の信頼性評価技術を調査し，それらを委託研究に応用できるかどうかについて参加者二名それぞれの視点から検討した．また、委託研究の成果の一部を論文として発表した．委託研究成果の一部であるテスト情報を考慮したソフトウェア信頼性評価では多くの質問が寄せられ，特にテスト入力空間における距離の定義には他にも様々なものが考えられることが指摘され,研究内容の改善に役立つものと考えられる．この知見は「有効性検証（カーネル法)」に反映させることができた．またその他の分野で議論されている様々な信頼性評価尺度をソフトウェア信頼性に適用する試みは興味深く，これらの知見を「推定実装（カーネル法)」に取り入れることを検討した．

【会議名：The 15th IEEE International Symposium on High Assurance Systems Engineering(HASE 2013)】

場所：Newport Beachside ResortHotel, Miami, Florida, USA

主催：IEEE

日程：2014 年 1 月 9 日～11 日

参加者：土肥，岡村

高アシュアランスシステムに関する国際会議に参加し，それらを委託研究に応用できるかどうかについて検討した．HASE 2014 への参加を通じて，狭義の信頼性だけでなく高いユーザビリティや性能を保持しながら信頼性を高めるいくつかの工夫について学び，その成果を委託研究に応用できるかどうかについて参加者二名それぞれの視点から分析した．特に，国外の研究者からポアソン回帰モデルに関する意見を収集した．ここで得られた知見は「有効性検証（一般化線形モデル)」に反映させることができた．

### 2.2.4 研究成果の公表

研究を推進してゆく過程の中で予備実験や他のモデルの検討・比較などを行ってきており，それらの部分的な成果を外部に向けて公表してきた．その成果を以下に列挙する．

(1) D. KuwaandT. Dohi, “Generalized logit-based software reliability modeling with metrics data, ”*Proceedings of The 37th Annual International Computer Software and Applications Conference(COMPSAC 2013)*, pp. 246--255, IEEE CPS, 2013.
【発表内容】委託研究で採用したロジスティック回帰に基づいたNHPPモデルのパラメータの自由度を増やすことで異なるソフトウェア信頼性モデルを開発することができたので,適合性評価や予測評価に関する性能を比較した.

(2) S. Ikemoto, T. DohiandH. Okamura, “Estimating software reliability with static project data in incremental development processes, ”*Proceedings of 2013 Joint Conference of the 23rd International Workshop on Software Measurement (IWSM-2013) and the 8th International Conference on Software Process and Product Measurement (MENSURA-2013)*, pp. 219--224, IEEE CPS, 2013.
【発表内容】委託研究で採用した回帰に基づいたNHPPモデルと異なるソフトウェア信頼性モデルをインクリメンタル開発データに適用し，モデルの適合性評価を行った.

(3) S. Ikemoto, T. DohiandH. Okamura, “Quantifying software test process and product reliability simultaneously, ” *Proceedings of The 24th International Symposium on Software Reliability Engineering(ISSRE 2013)*, pp. 108--117, IEEE CPS, 2013.
【発表内容】委託研究で採用した回帰に基づいたNHPPモデルと異なるソフトウェア信頼性モデルに対して，テスト環境の変化に応じて変調する新しいモデルを提案した.

(4) D. KuwaandT. Dohi, “Generalized Cox proportional hazards regression-based software reliability modeling with metricsdata, ” *Proceedings of The 19th IEEE Pacific Rim International Symposium on Dependable Computing(PRDC2013)*, pp. 328--337, IEEE CPS, 2013.
【発表内容】委託研究で採用しなかったCox回帰に基づいたNHPPモデルのパラメータの自由度を増やすことで異なるソフトウェア信頼性モデルを開発することができたので，適合性評価や予測評価に関する性能を比較した.

(5) H. OkamuraandT. Dohi, “A novel framework of software reliability evaluation with software reliability growth models and software metrics, ” *Proceedings of The 15th IEEE International Symposium on High Assurance Systems Engineering(HASE 2014)*, pp. 97--104, IEEE CPS, 2014.
【発表内容】ポアソン回帰に基づいたソフトウェア信頼性モデルの基本的な性質を調べるために，適合性評価や予測評価に関する性能を比較した.

(6) 岡村寛之，土肥正，“ソフトウェアメトリクスとフォールト記録による高精度フォールト予測，”電子情報通信学会技術研究報告(信頼性研究会)，vol. 113, no. 162, pp. 13--18, 紋別，7月26日，2013.
【発表内容】メトリクスデータの取り扱いとフォールトデータの取り扱いを工夫すること

で，ソフトウェア信頼性評価の精度を向上できる可能性について論じた．

(7) 久和大祐，土肥正，“メトリクスデータを活用した一般化 Cox 回帰に基づくソフトウェア信頼性モデリング，”電子情報通信学会技術研究報告(信頼性研究会)，vol．113，no．162，pp．19--24，紋別，7 月 26 日，2013．
【発表内容】委託研究で採用しなかった Cox 回帰に基づいた NHPP モデルのパラメータの自由度を増やすことで異なるソフトウェア信頼性モデルを開発することができたので，適合性評価や予測評価に関する性能を比較した．

(8) 岡村寛之，竹腰祐輝，土肥正，“テストケース入力情報を用いた細粒度ソフトウェア信頼性推定，”電子情報通信学会技術研究報告(信頼性研究会)，vol．113，no．348，pp．13--18，東京，12 月 13 日，2013．
【発表内容】テストケースの入力情報からテストケースの距離を定義する方法をいくつか提案し，その精度比較を行った．

## 2.3 研究実施体制


図 2-2：実施体制

表 2-1：研究責任者のプロフィール

| | |
|---|---|
| (ふりがな)<br>氏名 | どひただし<br>土肥正 |
| 生年月日 | 1965 年 6 月 18 日 |
| 所属機関 | 国立大学法人広島大学 |
| 所属<br>(部署名) | 大学院工学研究院情報部門<br>（ディペンダブルシステム論研究室) |
| 役職 | 教授 |
| 住所 | 〒739-8527 東広島市鏡山 1-4-1 |
| TEL | (082)424-7698 |
| E-mail | dohi@rel. hiroshima-u. ac. jp |

| 【学歴（大学卒業以降)】 | 【職歴】 |
|---|---|
| 1989:広島大学工学部第二類卒業<br>1991:広島大学大学院工学研究科博士課程前期修了<br>1992:広島大学大学院工学研究科博士課程後期中途退学 | 1992:広島大学工学部助手<br>1992:UBC(カナダ)客員研究員<br>1996:広島大学工学部助教授<br>2000:デューク大（米国）客員研究員<br>2002:広島大学大学院工学研究科教授<br>2010:広島大学大学院工学研究院教授 |

【研究実績】

ソフトウェア信頼性工学,ディペンダブルコンピューティングの研究に従事.学術雑誌掲載論文 200 編，査読付国際会議論文 250 編，著書 40 冊. ISSRE2011(広島)，ATC2012(福岡)など，内外で開催された 10 の国際会議における実行委員長を務める. PRDC2012 プログラム委員長，COMPSAC2013 トラックチェア，IEEE 主催国際会議 ISSRE，DSN，PRDC，SERE や ACM 主催国際会議 RACS，SAC などのプログラム委員を歴任.

【主な論文・著書】

"Enhancing performance of random testing through Markov chain Monte Carlo methods, "*IEEE Transactions on Computers*, vol. 62, no. 1, pp. 186--192, 2013(with B. Zhou, and H. Okamura).

"Wavelet shrinkage estimation for NHPP-based software reliability models, "*IEEE Transactions on Reliability*, vol. 62, no. 1, pp. 211—225, 2013(with X. Xiao).

"Towards quantitative software reliability assessment in incremental development processes , " *Proceedings of 33rd International Conference on Software Engineering(ICSE-2011)*, pp. 41--50, 2011(with T. Fujii, and T. Fujiwara, ).

# 3 研究成果

## 3.1 研究目標１「メトリクスを扱う信頼性評価モデルの構築」

### 3.1.1 当初の想定

#### (1)研究内容

ソフトウェアメトリクス，テストケース情報，ソースコード情報に基づいたソフトウェア信頼性評価のための確率モデルの構築を行う．具体的には，現在まで行ってきた研究と新しい成果を融合し信頼性評価のための最良モデルの選定を行う．ここでは「一般化線形モデル構築」と「カーネル法の適用」の二つの作業項目からなり，「一般化線形モデル構築」では，これまでのポアソン，ロジスティック，Cox回帰に基づいたモデルを一般化線形モデルの観点から統合化する．「カーネル法の適用」では，統合化した一般化線形モデルの回帰式に対してカーネル関数を適用する．カーネル関数は代表的なものでも幾つかの種類があるため，データ構造に適したカーネルを定性的な観点から絞り込む作業を行う．

#### (2)想定する課題と解決策

「一般化線形モデル構築」において定式化は容易に行えることが予想されるが，後続する研究目標である「推定アルゴリズム開発」において，効率的な推定が行えるように，一般化線形モデルの枠組みの自然な拡張をソフトウェア信頼性モデルに適用できるように工夫する．一方，「カーネル法の適用」でも，推定アルゴリズムにおける効率性を考慮する必要がある．そのため，「メトリクスを扱う信頼性評価モデルの構築」と「パラメータ推定アルゴリズムの開発」の期間をオーバーラップさせ，互いの成果を反映できるようにスケジュールを調整する．

### 3.1.2 研究プロセスと成果

#### (1)一般化線形モデル構築

一般化線形ソフトウェア信頼性モデル構築の準備として，既存のソフトウェア信頼性モデルおよび一般化線形モデル（回帰モデル）について言及し，その後に一般化線形ソフトウェア信頼性モデルのアイデアおよび定式化を行う．

##### (i)ソフトウェア信頼性モデル

ソフトウェア信頼度成長モデル（SRGM:Software Reliability Growth Model），もしくは単にソフトウェア信頼性モデル（SRM:Software Reliability Model）は本来ソフトウェアシステムテストにおける障害発生状況から運用時における信頼度を評価するためのモデルとして開発された．しかしながら，ソフトウェアテストフェーズ全体において発見されるバグが時間とともに収束していく振る舞いをうまく表現できることから，テストアクティビティの定量化・計測手段として用いられることも多い．ここでは，ソフトウェア信頼性モデルとして，代表的かつ数理的にも簡単に扱うことができる非定常ポアソン過程によ

るモデル化の原理とその利用方法を紹介する．

非定常ポアソン過程（NHPP:non-homogeneous Poisson process）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)[5]であり，過去30年にわたって数多くのモデルが提案されている．NHPPモデルでは，時刻$t$までに発見されたバグ数を$N(t)$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，非定常ポアソン過程（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代入するかという点に集約される．1970-1980年代に提案された様々なモデルは$H(t)$を確定的な微分方程式によって構成していた．例えば，Goel and Okumoto[5]によるモデル（以下，G-Oモデル）は平均値関数が次の微分方程式から得られている．

$$\frac{dH(t)}{dt}=b\big(a-H(t)\big). \tag{2}$$

ここで，パラメータ$a$はテスト開始前に潜在する総期待バグ数，$b$は単一バグの発見率あるいはソフトウェアデバッグ率を表し，現在のバグ発見率が残存バグ数に比例するという仮定から導出されている．この微分方程式を$H(t)=0$の初期条件の下で解くと，G-Oモデルの平均値関数

$$H(t)=a(b-e^{-bt}) \tag{3}$$

が得られる．その他多くのNHPPによるソフトウェア信頼性モデルも，そのほとんどが上記のような平均値関数に関する微分方程式から導かれる．

しかしながら，上述の議論は確率過程の平均的な振る舞いのみに着目し，平均的な振る舞いに対して仮定を設定するという，確率論的なモデリングの観点からは奇妙な解析となっている．現在では，NHPPモデルは次のような確率論的解釈が与えられ，NHPPモデルと他のモデルとの確率・統計論的な位置づけが既に明確になっている[7]．

いま，ソフトウェアテストにおけるバグ発見過程に対して次の仮定を設ける．

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$M$は有限であり，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検出される．

仮定C： 各バグの発見時刻は同一の確率分布関数$F(t)$をもつ非負の確率変数によって表される．

各バグの発見時刻は同一の確率分布関数$F(t)$をもつ非負の確率変数によって表される．

$$P(N(t)=k|M=n)=\binom{n}{k}F(t)^k\big(1-F(t)\big)^{n-k}. \tag{4}$$

総バグ数$M$が平均$\omega$のポアソン分布であるとき，累積発見バグ数に対する確率関数は

$$P(N(t)=k)=\frac{\big(\omega F(t)\big)^k}{k!}e^{-\omega F(t)} \tag{5}$$

となり，これは平均値関数$\omega F(t)$のNHPPの確率関数に一致する．バグ発見の時刻分布$F(t)$は任意であるため，この式は$\lim_{t\to\infty}H(t)=\omega<\infty$となる全てのNHPPモデルを包括する．

この解釈に従えば，ソフトウェア信頼性モデルは以下の2つの要因によって支配されていると言える．

- ソフトウェアバグの総数の平均値：$\omega$
- 単一のバグ発見時刻に関する確率分布：$F(t)$

つまり，これまでに多くのモデルが提案されているが，それらの違いは「バグ発見時刻分布の違い」のみに集約される．

ソフトウェア信頼性モデルを用いた信頼性評価は次の手順で行うことが推奨されている．

I. **ソフトウェアバグデータの採取**：テスト段階において発見されたバグに関するデータをどのような形式で利用するかを決定することは，データ解析手順に大きく影響を与える．代表的なデータ形式として，バグが発見・修正された時刻あるいはそれらの時間間隔に関するデータ（時刻データ）と時間間隔内に発見・修正されたバグ数に関するデータ（個数データ）がある．

II. **候補となるモデルの選択**：データおよびテスト環境に依存して，いくつかのモデルを選択する．文献[9, 18]では，モデルの選択基準として，予測妥当性，有効性，仮定の質，適用性，簡潔性をあげている．つまり，モデルの利用者が何らかの意図をもってモデルを選択する必要がある．

III. **モデルパラメータの推定**：バグ数データから選択されたモデルのパラメータを推定する．代表的な推定手法は最尤法あるいはベイズ法である．最尤法では，データが観測される確率を最大にするようなパラメータ値（最尤推定値）を算出する．数学的には尤度方程式と呼ばれる非線形方程式を解くことと等価であり，実際には何らかの数値解法を用いて算出する．また，ベイズ法はパラメータを確率変数と見なし，パラメータに対する事前分布と観測データの尤度を用いて，パラメータ分布を更新（事後分

布と呼ばれる）することでパラメータ推定を行う手法である．一般的に，ベイズ推定は汎化誤差の観点で最適な推定値を与えるが，実際の計算におけるコストが高い．

IV. **統計的検定に基づいたモデル決定と信頼性評価**：パラメータ推定された各モデルがデータをうまく表現しているかどうかをチェックする．つまり，データが想定したモデルからの標本であるかどうかを統計的に検定する．カイ二乗適合度検定やコルモゴロフ・スミルノフ検定などの適合度や，情報量基準と呼ばれる指標によって判断を行う．

上述した手順III，IVについては次節で述べる．ここでは手順IIに関して，標準的なモデルを与える．

**指数分布モデル**

指数分布モデルはGoel and Okumoto[5]によって提案されたモデルそのものであり，テスト時間経過に伴ってバグが偶発的に発見される現象を表している．平均値関数は以下で与えられる．

$$H(t) = \omega F(t) = \omega\left(1 - e^{-\beta t}\right),\ 0 \le t < \infty \tag{6}$$

**ガンマ分布モデル**

ガンマ分布モデルはバグ発見時刻分布が形状パラメータ$\alpha$，尺度パラメータ$\beta$のガンマ分布によって与えられるモデルである．つまり，その平均値関数は

$$H(t) = \omega F(t) = \omega \int_0^t \frac{\beta(\alpha+1)x^{\alpha}e^{-\beta x}}{\Gamma(\alpha)},\ 0 \le t < \infty \tag{7}$$

となる．特に$\alpha = 2$と固定したとき，Yamada et al. [15]による遅延S字モデルになる．

**パレート分布モデル**

パレート分布モデルはバグ発見時刻分布が第二種パレート分布（ロマックス分布）に従うモデルであり，平均値関数は

$$H(t) = \omega F(t) = \omega \frac{\beta}{(t+\beta)},\ 0 \le t < \infty \tag{8}$$

となる．これは文献[8]で議論された修正Duaneモデルと同じである．

**正規分布モデル**

バグ発見時刻分布が正規分布に従うモデルは2種類考えられる．一般的に正規分布に従う確率変数は$(-\infty, \infty)$の値を取り得るが，バグ発見時刻分布は$[0, \infty)$の範囲であるため，これを修正する必要がある．修正には「分布の切断」と「確率変数の変換」の2種類が考えられ

る．

分布の切断は，確率変数が$[0,\infty)$の領域に収まるように条件をつける手法である．具体的に，平均$\mu$，分散$\sigma^2$の正規分布の分布関数を

$$\Phi(t;\mu,\sigma^2)=\int_{-\infty}^{t}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{\sigma^2}}dx,\quad -\infty\le t<\infty \tag{9}$$

と表すと，$t=0$で切断された正規分布は

$$F(t)=1-\frac{1-\Phi(t;\mu,\sigma^2)}{1-\Phi(0;\mu,\sigma^2)},\quad 0\le t<\infty \tag{10}$$

となる．一方，確率変数の変換では，領域$(-\infty,\infty)$を$[0,\infty)$に写す関数を用いる．具体的に，元の正規確率変数$X$ではなく$e^X$に対する分布をバグ発見時刻分布とする．確率変数$X$が正規分布に従う場合，$e^X$は対数正規分布に従う．対数正規分布の確率分布関数は

$$F(t)=\Phi(\log t;\mu,\sigma^2),\quad 0\le t<\infty \tag{11}$$

となる．これらのバグ発見時刻分布を用いると切断正規分布モデルの平均値関数は

$$H(t)=\omega\left(1-\frac{1-\Phi(t;\mu,\sigma^2)}{1-\Phi(0;\mu,\sigma^2)}\right),\quad 0\le t<\infty \tag{12}$$

で与えられ，対数正規分布モデルの平均値関数は

$$H(t)=\omega F(t)=\omega\Phi(\log t;\mu,\sigma^2),\quad 0\le t<\infty \tag{13}$$

で与えられる．

**ロジスティック分布モデル**

ロジスティック分布とは累積分布関数がロジスティック曲線を描く分布である．正規分布とほぼ同じ形を取るが裾が若干厚い特徴を持つ．正規分布と同じく$(-\infty,\infty)$で定義される分布であるため，ロジスティック分布によるソフトウェア信頼性モデルの表現として，切断ロジスティック分布モデルと対数ロジスティック分布モデルがある．位置パラメータ$\theta$，尺度パラメータ$\xi$のロジスティック分布の累積分布関数は

$$\Psi(t;\mu,\psi)=\frac{\exp((t-\mu)/\psi)}{1+\exp((t-\mu)/\psi)},\quad -\infty<t<\infty \tag{14}$$

である．そのため，切断ロジスティック分布モデルは

$$H(t) = \omega F(t) = \omega\left(1 - \frac{1-\Psi(t;\mu,\psi)}{1-\Psi(0;\mu,\psi)}\right),\ 0 \le t < \infty \tag{15}$$

で与えられる．これは文献[10]で提案された習熟S字形信頼度成長モデルと呼ばれ，上記の平均値関数はロジスティック曲線を描く．一方，対数ロジスティック分布モデルの平均値関数は

$$H(t) = \omega F(t) = \omega\Psi(\log t;\mu,\psi),\ 0 \le t < \infty \tag{16}$$

となり，文献[6]で提案されている．

**極値分布モデル**

極値分布とは確率変数列の最大値あるいは最小値に対する分布として定義される．一般的には3種類（Gumbel, Frechet, Weibull）の分布型があるが，ここではGumbel分布（Gumbel型の最大極値分布）を基本に議論を展開する．

Gumbel分布は，正規分布やロジスティック分布と同様に$(-\infty,\infty)$で定義される分布であるため，ソフトウェア信頼性モデルの表現としても切断型，対数型が考えられる．さらに，最大値，最小値に対する分布を考慮することで，合計4種類の異なる平均値関数を持つモデルが考えられる．

位置パラメータ$\theta$，尺度パラメータ$\xi$の Gumbel 分布の累積分布関数は

$$\Xi(t;\mu,\xi) = \exp\left(-\exp\left(-\frac{t-\mu}{\xi}\right)\right),\ -\infty \le t < \infty \tag{17}$$

である．そのため，切断 Gumbel 分布（切断最大極値分布）モデルの平均値関数は

$$H(t) = \omega F(t) = \omega\left(1 - \frac{1-\Xi(t;\mu,\xi)}{1-\Xi(0;\mu,\xi)}\right),\ 0 \le t < \infty, \tag{18}$$

対数 Gumbel 分布（対数最大極値分布）モデルの平均値関数は

$$H(t) = \omega F(t) = \omega\Xi(\log t;\mu,\xi),\ 0 \le t < \infty \tag{19}$$

となる．対数Gumbel（最大値）分布はFrechet型の極値分布となることに注意する．一方で，最小値はGumbel確率変数$X$に対して$-X$で表現することができるため，切断最小極小値分布モデルの平均値関数は以下で与えられる．

$$H(t) = \omega F(t) = \omega\frac{\Xi(-t;\mu,\xi)}{\Xi(0;\mu,\xi)},\ 0 \le t < \infty. \tag{20}$$

また，対数最小極値分布モデルの平均値関数は

$$H(t) = \omega F(t) = \omega\big(1 - \Xi(-\log t; \mu, \xi)\big),\ \ 0 \le t < \infty \qquad (21)$$

となり，これはワイブル型極値分布（ワイブル分布）をバグ発見時刻分布としたモデル[4]になる．

以下の表 3-1-1 に標準的な NHPP モデルをまとめる．

**表 3-1-1：NHPP モデルとバグ発見時刻分布の対応．**

| モデル名 | バグ発見時刻分布 | 文献（既存モデル名） |
|---|---|---|
| exp | 指数分布 | 指数形SRGM<br>Goel and Okumoto model[5] |
| gamma | ガンマ分布 | 遅延S字形SRGM[15]<br>Gamma-type NHPP model[16] |
| pareto | 第二種パレート分布 | 修正Duane model[8]<br>Littlewood NHPP model[1] |
| tnorm | 切断正規分布 | [12] |
| norm | 対数正規分布 | [2] |
| tlogis | 切断ロジスティック分布 | 習熟 S 字形 SRGM[10] |
| llogis | 対数ロジスティック分布 | Log-Logistic model[6] |
| txvmax | 切断 Gumbel 分布 | 修正 Gompertz model[11, 17] |
| lxvmax | Frechet 型極値分布 | [11] |
| txvmin | Gompertz 分布（最小値） | [11] |
| lxvmin | Weibull 分布 | 一般化指数形SRGM<br>Goel model[4,11] |

**（ⅱ）一般化線形モデル**

一般化線形モデル(GLM:generalized linear model)とは，正規分布による線形回帰（重回帰）モデルを一般化したモデルの枠組みであり，正規分布以外の分布や，多様なリンク関数を用いて独立変数と従属変数の因果関係を表すモデルである．特に，分布が指数分布族の場合，後述する回帰係数の推定アルゴリズムがリンク関数によらず，繰り返し最小二乗法を適用できる点が大きな利点となっている．

いま，確率変数（従属変数）$Y_1, \ldots, Y_N$とそれらに対応する確率変数（独立変数）ベクトル$X_1, \ldots, X_N$の観測値が与えられるとする．ここで，$X_i = \big(X_{i,1}, \ldots, X_{i,L}\big)$であり，$X_{i,l}$は$Y_i$に対する要因$l$の値を表す．このとき，$Y$と$X$の関係を表すモデルとして次の仮定を考える．

- $Y_1, \ldots, Y_N$は指数分布族に属する同一の分布に従う．ただし，パラメータが異なる．
- 従属変数の平均は$g(E[Y_i]) = \beta x_i$で与えられる．$x_i$は$X_i$の観測値であり，$g(\cdot)$はリン

ク関数と呼ばれる．

上記の枠組みは$Y_1, \ldots, Y_N$の分布とリンク関数を選ぶことにより，既存の回帰モデルに帰着させることができる．

分布を正規分布，リンク関数を$g(\mu) = \mu$とすると，重回帰モデルに帰着される．また，分布をポアソン分布，リンク関数を$g(\mu) = \log(\mu)$とするとポアソン回帰，分布をベルヌーイ分布，リンク関数をロジット関数$g(\mu) = \log \mu/(1+\mu)$とすると，ロジスティック回帰へそれぞれ帰着される．

**(iii)一般化線形ソフトウェア信頼性モデル**

前述のソフトウェア信頼性モデルはソフトウェアメトリクスを直接扱うことができない．一方，一般化線形モデルは要因との因果関係を表すことができる．そこでこれらを融合したモデルの構築を行う．

ソフトウェアメトリクスとは、ソフトウェア自身あるいはその開発を特徴づける定量的な指標である．一般に，ソフトウェアメトリクスは，コード行数，ソースの複雑度といった設計段階のソフトウェアに関する情報のデザインメトリクス，テスト工程において計測される，投入するテストケース数やカバレッジなどソフトウェアのテストの進捗具合やフォールトの検出に寄与するような情報であるテストメトリクスの2種類に分類できる．デザインメトリクス，テストメトリクスはそれぞれ，フォールト検出に与える影響が異なることを考慮して，デザインメトリクス，テストメトリクスを包括的に扱うモデルを開発する．

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的に実施され，各テスト毎で検出されたフォールトは次のテストまでに修正される．
- モジュール$i$において単一のフォールトが$k$番目のテスト区間で検出されるフォールト検出確率$p_{i,k}$ $\left(0 \le p_{i,k} \le 1\right)$ はモジュール$i$において$k$番目のテスト区間に関するテストメトリクスベクトル$\boldsymbol{x}_{i,k} = \left(x_{i,k,1}, \ldots, x_{i,k,r_i}\right)$を用いて

$$l_p(p_i, k) = \alpha_{0,i} + \alpha_i \boldsymbol{x}_{i,k} \tag{22}$$

となる．ここで$l_p(\cdot)$はフォールト検出確率のリンク関数である．

- 各モジュール$i = 1, \ldots, j$に含まれる総フォールト数（初期フォールト数）$M_1, \ldots, M_j$はモジュール固有のデザインメトリクス$\boldsymbol{s}_i$に依存した平均$\xi_i$のポアソン分布に従い，その平均は以下で与えられる．

$$l_\xi(\xi_i) = \beta_0 + \boldsymbol{\beta}_{s_i}. \tag{23}$$

ここで，$l_\xi(\cdot)$は総フォールト数のリンク関数である．

フォールト検出確率のリンク関数としては，ロジット関数，プロビット関数および complementary log-log関数が適用できる．また，総フォールト数のリンク関数としては対数関数や一次関数が適用できる．いま，

$$l_p(p) = \mathrm{logit}(p) = \log\frac{p}{1-p},\quad l_\xi(\xi) = \log(\xi) \tag{24}$$

とすると，モジュール$i$において$k$番目のテスト区間のフォールト検出確率は

$$p_{i,k} = \frac{\exp(\alpha_{0,i}+\alpha_i x_{i,k})}{1+\exp(\alpha_{0,i}+\alpha_i x_{i,k})} \tag{25}$$

となる．さらに，モジュール$i$の初期フォールト数は平均$\exp(\xi_i)$のポアソン分布に従うため，モジュール$i$の$k$番目のテスト区間までに検出されるフォールト数$Y_{i,k}$は以下の確率関数で与えられる．

$$\begin{aligned} &P(Y_{i,k_i} = y_{i,k}) \\ &= \sum_{m_i=y_{i,k}}^{\infty} P(Y_{i,k_i} = y_{i,k} | M_i = m_i) P(M_i = m_i) \\ &= \exp(-\xi_i\lambda_{i,k})\frac{(\xi_i\lambda_{i,k})^{y_{i,k}}}{y_{i,k}!}. \end{aligned} \tag{26}$$

ここで，$\lambda_{i,n_i} = 1 - \prod_{k=1}^{n_i}(1-p_{i,k})$である．

このモデルは，モジュール$i$に含まれる総フォールト数に対するポアソン回帰と$k$番目のテスト区間におけるフォールト検出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．

**(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，リンク関数を使うことで，メトリクスとフォールト数あるいはフォールト検出確率の関係が線形であることを仮定している．しかしながら，実際の問題としては本当に線形関係で記述できるかどうか判断するのは難しい．また，ソフトウェアに関するメトリクスを必要とする一方で，ソースコードの詳細な情報を有効に使うことができていない．そこで，これらの問題を解決するため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表現することができる．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．元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すことができることが多く，線形回帰などの線形モデルの適用により効率良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を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とと，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．

一般にカーネル法の適用は，もとの線形モデルに対してカーネル関数を適用することで得られる．いま，一般化線形ソフトウェア信頼性モデルに対してカーネル法を適用する．この場合，先に示した線形の関係式を次のような関数で置き換えることで得られる．

$$l_p(p_{i,k}) = \sum_{u=1}^{k} \mathcal{K}(x_{i,u}, x_{i,k}),\ \ l_\xi(\xi_i) = \sum_{u=1}^{k} \mathcal{K}(s_i, s_m). \tag{27}$$

ここで，$\mathcal{K}(x,y)$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．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．通常，数値ベクトルに対しては次のガウスカーネルおよび多項式カーネルがよく用いられる．

$$\mathcal{K}(x,y) = \exp\left(-\frac{\|x-y\|^2}{\sigma}\right) \tag{28}$$

$$\mathcal{K}(x,y) = (xy + c)^d. \tag{29}$$

これらは，用いる高次元空間の次元数が異なるため，データに対して適したカーネル関数が存在することに注意する．

また，文字列やグラフなどの構造データを扱うカーネル関数は上述のものとは大きく異なる．例えば，文字列に関しては部分文字列の頻度によってベクトル化するカーネルが考えられている．また，二つの文字列$x,y$の距離$D(x,y)$が定義された場合，これを用いたカーネルとして以下のものを用いることができる．

$$\mathcal{K}(x,y) = \exp\left(-\frac{D(x,y)}{\sigma}\right). \tag{30}$$

### 3.1.3 課題と問題点

一般化線形ソフトウェア信頼性モデルは，動的／静的なソフトウェアメトリクスと従来の信頼性モデルを自然な形で融合したモデルである．また，次節で言及するが，その大きな利点は，効率的な推定アルゴリズムが構築できる点にある．しかしながら，最も大きな要因である「どのようなメトリクスを用いれば良いか」については，現時点でも大きな課題として残っている．カーネル法の適用は，特定のソフトウェアメトリクスを抽出しない代わりに，メトリクスの源泉であるソースコードやテストケース情報を直接利用することを試みることで「どのようなメトリクスを採用するか」という問題に対する一つの解を与えた．しかしながら，カーネル法においても類似度（あるいは距離）の定義によりその性能が大きくゆらぐ．端的な例を挙げれば，整数値0を入力したときと，整数値1以上を入力した時で大きく動作の異なるプログラムがあった場合，0と1の距離，1と2の距離をユークリッド距離で定義することは直感的にも間違っていることは明らかである．このような場合，0と1は「プログラムの振る舞い」の上で大きな距離があるため，それらを考慮した距離（メトリック空間）を定義する必要がある．信頼性評価における「距離の効果」に関する議論はこの研究を通じて始まったばかりであり，その点を検証実験などを通じて明らか

にしていく必要がある.

## 3.2 研究目標２「パラメータ推定アルゴリズムの開発」

### 3.2.1 当初の想定

#### (1)研究内容

構築したモデルパラメータをソフトウェアメトリクス，テストケース情報，ソースコード情報から推定するアルゴリズムの開発を行う．ここでは「一般化線形モデルに対する推定」と「近似手法の検討」の二つの作業項目からなる．「一般化線形モデルに対する推定」では，統合化したポアソン，ロジスティック，Cox回帰に基づいたモデルに対するパラメータ推定をEM（Expectation-Maximization）アルゴリズムにより効率的に推定する手法を開発する．また，ステップワイズ法による要因選択に対する効率的なアルゴリズムの検討も行う．さらに，カーネル関数を用いたモデルに対する推定として，縮小推定を適用した拡張も行う．「近似手法の検討」では，カーネル関数に基づいたモデルのパラメータ推定についての改良を行う．テストケース情報やソースコード情報は数千から数十万規模のデータを扱うことになるため，縮小推定を適用した手法で対応できない可能性がある．そのため，凸二次計画法を利用した近似解法について検討を行う．

#### (2)想定する課題と解決策

最も難しい点はカーネル法を適用した場合に，扱うデータ規模が大きくなるため推定アルゴリズムのスケーラビリティを維持する点である．解決策としては「高いスケーラビリティを持つ数理計画法の適用」，「並列化が容易に行える手法の開発」の二つのアプローチを考える．

### 3.2.2 研究プロセスと成果

#### (1)一般化線形モデルに対する推定

##### (i)ソフトウェア信頼性モデルに対する推定

ここでは，ソフトウェア信頼性モデルを用いてデータからソフトウェア信頼性を評価する上で重要となる統計手法についての概略を述べる．先に示した，一般的な信頼性評価手順を再掲する．

I. **ソフトウェアバグデータの採取**：テスト段階において発見されたバグに関するデータをどのような形式で利用するかを決定することは，データ解析手順に大きく影響を与える．代表的なデータ形式として，バグが発見・修正された時刻あるいはそれらの時間間隔に関するデータ（時刻データ）と時間間隔内に発見・修正されたバグ数に関するデータ（個数データ）がある．

II. **候補となるモデルの選択**：データおよびテスト環境に依存して，いくつかのモデルを選択する．文献[9, 18]では，モデルの選択基準として，予測妥当性，有

効性，仮定の質，適用性，簡潔性をあげている．つまり，モデルの利用者が何らかの意図をもってモデルを選択する必要がある．

III. **モデルパラメータの推定**：バグ数データから選択されたモデルのパラメータを推定する．代表的な推定手法は最尤法あるいはベイズ法である．最尤法では，データが観測される確率を最大にするようなパラメータ値（最尤推定値）を算出する．数学的には尤度方程式と呼ばれる非線形方程式を解くことと等価であり，実際には何らかの数値解法を用いて算出する．また，ベイズ法はパラメータを確率変数と見なし，パラメータに対する事前分布と観測データの尤度を用いて，パラメータ分布を更新（事後分布と呼ばれる）することでパラメータ推定を行う手法である．一般的に，ベイズ推定は汎化誤差の観点で最適な推定値を与えるが，実際の計算におけるコストが高い．

IV. **統計的検定に基づいたモデル決定と信頼性評価**：パラメータ推定された各モデルがデータをうまく表現しているかどうかをチェックする．つまり，データが想定したモデルからの標本であるかどうかを統計的に検定する．カイ二乗適合度検定やコルモゴロフ・スミルノフ検定などの適合度や，情報量基準と呼ばれる指標によって判断を行う

手順III，IVに関しては数理モデルや統計に関する知見を必要とするため，実務家にとって決して簡単ではない．そのため，幾つかの書籍では推定を簡便にした手法が紹介されることが多いが，その場合，統計的に合理的な多くの性質を失ってしまうばかりか，モデル化における根本的な仮定を逸脱することもある（モデルを適用範囲外へ利用してしまっている）．現在では，上記の統計的取り扱いをパッケージングしたツール（例えば，SRATS: http://www.rel.hiroshima-u.ac.jp/okamu/SRATS/）が開発されており，それらのツールを用いた評価が実際の運用において推奨される．ここでは手順IIIと手順IVについて言及する．

一般的に，ソフトウェア信頼性モデルに用いられるパラメータ推定手法は，最小二乗法，最尤法，ベイズ法の3種類が提案されている．本研究では最も一般的に利用される最尤法の適用を考える．

最尤法とは，確率モデルに対する汎用的なパラメータ推定手法であり，観測データが実際に生成される確率を最大にするようにパラメータを決定する手法である．最尤法による推定量（最尤推定量）は漸近正規性や漸近有効性などの統計的に多くの良い性質を持つため，他の推定量に比べて利用範囲の広い推定量である．

具体的に，NHPPモデルにおけるパラメータ推定について言及する．いま表3-2-1のような実データが与えられているものとする．

**表 3-2-1：テスト工程におけるバグ数のデータ形式.**

| 時間間隔 | 発見バグ数 |
|---|---|
| $[0, t_1)$ | $y_1$ |
| $[t_1, t_2)$ | $y_2$ |
| … | … |
| $[t_{K-1}, t_K)$ | $y_K$ |

このとき，平均値関数$H(t;\theta)$（$\theta$はパラメータ）に対する対数尤度関数

$$\mathrm{LLF}(\theta) = \sum_{i=1}^{K} y_i \log\{H(t_i;\theta) - H(t_{i-1};\theta)\} - H(t_K;\theta) - \sum_{i=1}^{K} \log y_i! \qquad (31)$$

を定義する．最尤法における推定値は$\mathrm{LLF}(\theta)$を最大にする値によって与えられる．上記を最大にする値は解析的に導出できないため，数値的に算出され，よく知られているのは以下に示すニュートン法である．

最尤法における推定値は$\mathrm{LLF}(\theta)$を最大にする値によって与えられるため，スコア関数

$$u(\theta) = \frac{\partial}{\partial\theta}\mathrm{LLF}(\theta) \qquad (32)$$

を定義すると，1 階の最適性条件から下記の方程式（尤度方程式）を満たすパラメータを見つける問題に帰着される．

$$u(\theta) = 0 \qquad (33)$$

いま，スコア関数を最尤推定値まわりでテイラー展開することによって

$$u(\theta) \approx u(\theta_0) + \frac{\partial}{\partial\theta}u(\theta)|_{\theta=\theta_0}(\theta - \theta_0) \qquad (34)$$

を得る．ここでスコア関数の1階微分はヘッセ行列$H_e$と呼ばれる．式(33)とヘッセ行列を用いて，上式は

$$\theta \approx \theta_0 - H_e^{-1}(\theta_0)u(\theta_0) \qquad (35)$$

となる．暫定的なパラメータとして，次の更新式を得る．

$$\theta \coloneqq \theta' - H_e^{-1}(\theta')u(\theta'). \qquad (36)$$

これは，暫定的なパラメータ$\theta'$が最尤推定値に近い場合，式(36)の更新式を用いることによって対数尤度を最大にする最尤推定値を算出できることを意味する．実際には，適当に与えられた暫定値に対して更新式を繰り返し適用する．この手法はニュートン法と呼ばれ，多くの問題に対して汎用的に用いることができる．また，ヘッセ行列の逆行列を厳密に算出するのではなく，近似的に$H_e^{-1}$を算出する手法は準ニュートン法と呼ばれる．

しかしながら，NHPPに対する対数尤度関数は非線形性の強い関数であると同時に，多くのパラメータには非負の制約が課せられることが多い．そのため，局所的な収束性しか持

たないニュートン法で最尤推定値を算出することは，モデルパラメータの数が多ければ多いほど困難な作業になる．

近年，EM (Expectation-Maximization)アルゴリズムを用いたパラメータ推定法が提案されている[13]．これは，ニュートン法が持つ収束性の悪さを克服するための，ソフトウェア信頼性モデルにおける有効なパラメータ推定アルゴリズムとなっている．

いま，バグ発見時刻に関する時刻データ$S_1 < S_2 < \cdots < S_N$が利用可能な状況を考える．ここで，$N$はソフトウェアに潜在する総バグ数であり，仮定Aからパラメータ$\omega$のポアソン分布に従うものとする．このとき，対数尤度関数は

$$\mathrm{LLF}(\omega,\theta) = N\log\omega - \omega + \sum_{i=1}^{N}\log f(S_i;\theta) \tag{37}$$

となる．ここで$\theta$はバグ発見時刻分布に関するパラメータ集合である．1階の最適性条件から，尤度方程式

$$\omega = N, \tag{38}$$

$$\sum_{i=1}^{N}\frac{\partial}{\partial\theta}\log f(S_i;\theta) = 0 \tag{39}$$

を得る．

EMアルゴリズムは不完全なデータを用いた推定に関するアルゴリズムである．確率密度関数$f(\cdot\,;\,\theta)$に従う観測不可能な確率変数$X$と観測可能な確率変数$Y = u(X)$が与えられているものとする．いま，$Y$に関する観測値$y$が与えられると，EMアルゴリズムを用いてパラメータ$\theta$を推定することができる．このとき，EMアルゴリズムにおける1ステップは，不完全な観測値$y$が与えられたもとでの対数尤度の期待値を最大にする$\theta$を求めることに対応する．すなわち，パラメータの更新式は

$$\theta := \mathrm{argmax}\{\mathrm{E}[\log f(X;\theta)|u(X) = y;\theta']\} \tag{40}$$

によって与えられる．ここで$\theta'$は暫定的なパラメータであり，$\mathrm{E}[\cdot\,;\,\theta]$は$\theta$を用いたときの期待演算子を示す．

ソフトウェア信頼性モデルにおけるパラメータ推定において，データ$\mathcal{D} = (s_1,\ldots,s_K,T)$が与えられているものとする．ここで，$T$は現在のテスト経過時刻であり，$(s_K,T]$においてバグが発見されていないことを意味する．先のデータと比較すると，ここで与えられたデータは$N$個あるうちの$K$個のバグに関するデータしか記述されていないことがわかる．すなわち，$K+1$個目以降のバグ発見時刻がわかっていないという意味で不完全データとみなすことができる．この意味において，EMアルゴリズムの1ステップを次のように構築することができる．

$$\omega = \mathrm{E}[N|\mathcal{D};\omega',\theta'] \tag{41}$$

$$\mathrm{E}\left[\sum_{i=1}^{N}\frac{\partial}{\partial\theta}\log f(S_i;\theta)|\mathcal{D};\omega',\theta'\right]=0. \tag{42}$$

ここで,上記の期待値は以下の式を用いて算出することができる(詳細は文献[13]を参照).

$$\mathrm{E}\left[\sum_{i=1}^{N}h(S_i)|\mathcal{D};\omega',\theta'\right]=\sum_{i=1}^{N}h(S_i)+\omega'\int_{T}^{\infty}h(u)f(u;\theta')du. \tag{43}$$

ここで$h(\cdot)$は任意の可測関数である

具体的には,上記を基本式として,各NHPPモデルに対して推定アルゴリズムを構成する.例えば,指数形ソフトウェア信頼性モデルの場合,バグ発見時刻分布が密度関数$f(t;\beta)=\beta e^{-\beta t}$で与えられる.式(39)から,完全データが与えられたもとで$\beta$に対する最尤推定値は簡単に

$$\beta=\frac{N}{\sum_{i=1}^{N}S_i} \tag{44}$$

となるため,バグ発見時刻データ$\mathcal{D}=(s_1,\ldots,s_K,T)$が与えられたもとで,式(41),(42),(43)を用いて

$$\omega:=\mathrm{E}\left[N|\mathcal{D};\omega',\beta'\right]:=K+\omega'e^{-\beta' T}, \tag{45}$$

$$\beta:=\frac{\mathrm{E}[N|\mathcal{D};\omega',\beta']}{\mathrm{E}[\sum_{i=1}^{N}S_i|\mathcal{D};\omega',\beta']}:=\frac{K+\omega'e^{-\beta'T}}{U+\omega'(T+1/\beta')e^{-\beta'T}} \tag{46}$$

を得る.ここで$U=\sum_{i=1}^{K}S_i$である.上記の更新式を繰り返し適用することで指数形ソフトウェア信頼性モデルのパラメータ推定を行うことができる.これらの更新式を用いる場合は$K,\ U,\ T$という3つの統計量がわかれば十分であることから,従来の手法に比べて計算手続きが極端に簡素化されていることがわかる.

EMアルゴリズムによるNHPPモデルのパラメータ推定は取り扱うデータが変化しても,不完全データに関する取り扱いを変えることで対応できるというメリットがある.ここでは,バグの厳密な発見時刻がわからないグループデータ(個数データ)に対するEMアルゴリズムを紹介する.

個数データは時間間隔とその間の発見バグ数からなるデータで,実際のソフトウェア開発では厳密なバグ発見時刻がわからないことが多いため,個数データによってパラメータ推定を行うことは,より現実的な状況を想定している.いま,観測された個数データを$\mathcal{I}=\{(t_1,y_1),(t_2,y_2),\ldots,(t_K,y_K)\}$とする.ここで$(t_i,y_i)$は時間間隔$[t_i,y_i)$で$y_i$個のバグが発見されたことを示す.個数データを取り扱う場合,先の完全データと比較して(a)未発見のバグの発見時刻がわからない,(b)発見されたバグの厳密な発見時刻がわからない,という2つの意味において不完全データとして考えることができる.これは,時間データを取り扱

う場合と比較して，不完全データが与えられた元での期待値の計算が異なることを意味する．つまり，式(43)の期待値に関する式を

$$\mathrm{E}\left[\sum_{i=1}^{N} h(S_i)|\mathcal{I};\omega',\theta'\right] = \sum_{i=1}^{K} \frac{y_i \int_{t_{i-1}}^{t_i} h(u)f(u;\theta')du}{\int_{t_{i-1}}^{t_i} f(u;\theta')du} + \omega' \int_{t_K}^{\infty} h(u)f(u;\theta')du \quad (47)$$

と変更することで，個数データに関するEMアルゴリズムを構築することができる．例として，指数形ソフトウェア信頼性モデルの場合は以下のアルゴリズムとなる．

$$\omega := \mathrm{E}\left[N|\mathcal{I};\omega',\beta'\right] := \sum_{i=1}^{K} y_i + \omega' e^{-\beta' t_K}, \quad (48)$$

$$\beta := \frac{\mathrm{E}[N|\mathcal{I};\omega',\beta']}{\mathrm{E}[\sum_{i=1}^{N} S_i|\mathcal{I};\omega',\beta']} := \frac{\sum_{i=1}^{K} y_i + \omega' e^{-\beta' t_K}}{\sum_{i=1}^{K} y_i T_i + \omega'\left(t_K + \frac{1}{\beta'}\right)e^{-\beta' t_K}}. \quad (49)$$

ここで，

$$\tau_i := \frac{(t_i + 1/\beta')e^{-\beta' t_{i-1}} - (t_i + 1/\beta')e^{-\beta' t_i}}{e^{-\beta' t_{i-1}} - e^{-\beta' t_i}}. \quad (50)$$

次に適切なモデルを選択するための基準について述べる．前節で述べた確率的なモデルの解釈が与えられる以前，NHPPモデルに基づいたモデル化では，ある特有のデバッグに関するシナリオを設けることによって，平均値関数が満たすべき微分方程式を仮定し，NHPPモデルを特徴づけていた．つまり，モデルを理解するための物理的な解釈は，平均値関数によって表現された決定論的シナリオを理解することと同義であった．しかしながら，このようなモデル化はモデル評価の際に「仮定の質」（仮定の妥当性）を評価しなければならない．これは，経験則あるいは主観的な判断に委ねられるものであり，仮定と現象の因果関係がはっきりしない状況で仮定の妥当性を検証した上で適当なソフトウェア信頼性モデルを選択することは現実的に不可能である．そこで，「どのモデルが良いか」と言うモデル選択の問題よりも，「モデルがデータにあっているのか」を検査する「適合性検定」と呼ばれる統計的な技術が発展してきた．

その一方で，確率論的な解釈が与えられることにより，NHPPを用いたモデルにおけるモデル選択が「バグ発見時刻分布の選択問題」へ置き換わることで，従来の統計における「モデル選択」の議論が適用することができ，ソフトウェア信頼性モデルにおける大きな発展につながった．ここでは，統計的観点からのモデル選択問題について説明する．

前節で述べたように，ソフトウェア信頼性モデルではモデルの違いがバグ発見時刻分布の違いに集約される．そのため，単にフォールト検出時間がどのような分布に従っているかを統計的に検証することでモデル選択を行うことができる．これは，モデルの妥当性が完全に確率分布の適合度評価の問題に帰着されるという意味において非常に有効である．

比較的新しい統計学においては，情報量基準とよばれる評価規範によってモデルを評価することが頻繁に行われる．これは真のモデル（分布）とデータに基づいて構成されたモデル（分布）の「距離」を定義することに対応している．通常，分布の距離として

Kullback-Leibler（K-L）情報量を用いる．これは真のモデルを$p(x)$，データに基づいて構成されたモデルを$p'(x)$とした時

$$I = \int p(x)\log\frac{p(x)}{p'(x)}dx = \int p(x)\log p(x)dx - \int p(x)\log p'(x)dx \tag{51}$$

として定義される．上式の第1項は真のモデルのみからなる量であるため，第2項を最大にするモデルがK-L情報量の意味で最も良いモデルとなる．しかしながら，厳密には真のモデルが明らかである状況でなければ正確な値を算出することができない．つまり，式(51)の第2項の正確な見積もりが必要となる．これを用いれば，2つあるモデルのどちらがより真のモデルに近いかどうかを評価することができる．この見積もりの根拠を提供するツールが情報量規準であり，よく知られたものではAIC（Akaike's Information Criterion）[3]やBIC（Bayesian Information Criterion）[14]がある．実際にAICやBICは

$$\mathrm{AIC} = -2 \times (\text{最大対数尤度} - \text{モデルのパラメータ数}), \tag{52}$$
$$\mathrm{BIC} = -2 \times (\text{最大対数尤度}) + (\text{モデルのパラメータ数})\log(\text{データ数}) \tag{53}$$

として算出され，式の形状から考えると，必要以上にデータに適合することを防ぐために，モデルパラメータの数（モデルの次元）が評価に影響していることがわかる．AICやBICなどの情報量規準を用いると，検定に関する問題が情報量規準の大小比較に帰着される．つまり，ソフトウェア信頼性モデルのフレームワークにおいては，最尤推定を行った各モデルに対して，AICあるいはBICを計算し，最も値の小さいモデルを採用することで検定を行う手間が軽減できる．ただし，近年の研究ではソフトウェア信頼度成長モデルは，ある時刻より先のサンプルが得られない特殊な環境としてモデル化されているため，BICで用いている近似（ラプラス近似）の仮定が成立していない．上記のBICの計算式に関しては今後，理論的な見直しが必要であると思われる

**(ii)一般化線形ソフトウェア信頼性モデルに対する推定アルゴリズムの構築**

本研究では一般化線形ソフトウェア信頼性モデルに対する推定手法を開発した．一般化線形ソフトウェア信頼性モデルは従来のソフトウェア信頼性モデルと比較してパラメータ数が多いため，効率的な推定を行う目的でEMアルゴリズムの適用を考える．

EMアルゴリズムは先に示したようにNHPPモデルでも用いている手法であり，未観測データを含むモデルパラメータを推定する一般的な推定の枠組みである．一般的には，未観測データの期待値を計算するE-stepと完全データ（観測データと未観測データ）の期待対数尤度を最大にするM-stepの二つの手続きを繰り返し，これらのステップで観測データに対する最尤推定値を計算することができる．

ここで考える一般化線形モデルの場合，未観測データを各モジュールの総フォールト数$N_1, \ldots, N_j$とし，モジュール$m$のフォールト検出時刻列を$T_{m,1} < T_{m,2} < \cdots < T_{m,N_m}$とする．このとき，完全対数尤度は

$$\log\mathcal{L}(\beta_0, \beta, \alpha_{0,1}, \alpha_1, \dots, \alpha_m|\mathcal{D}, \mathcal{U})$$
$$= \sum_{m=1}^{j} N_m \left(\beta_0 + \beta_{S_j}\right) - \sum_{m=1}^{j} \exp\left(\beta_0 + \beta_{S_j}\right) + \sum_{m=1}^{j}\sum_{k=1}^{N_m} \log\left(\lambda_{m,k} - \lambda_{m,k-1}\right) \quad (54)$$

となる．つまり，最初の二項が$N_1, \dots, N_m$を従属変数としたポアソン回帰となり，$\lambda_{m,k}$に関する項は$N_m - \sum_{u=1}^{k-1} y_u$を従属変数としたロジスティック回帰となる．一方で，総フォールト数の期待値は与えられたパラメータ$\beta_0, \beta, \alpha_{0,m}, \alpha_m$に対して

$$\mathrm{E}[N_m] = \sum_{k=1}^{K_m} y_{m,k} + \exp\left(\beta_0 + \beta_{S_m}\right) \prod_{k=1}^{K_m} \frac{\exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)}{1+\exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)} \quad (55)$$

として計算できる．最終的に，推定アルゴリズムは次のようになる．

- Step 1 : Compute $\omega_m \leftarrow \exp\left(\beta_0 + \beta_{S_m}\right),\ m = 1, \dots, j.$
- Step 2 : Compute

$$p_{m,k} \leftarrow \frac{\exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)}{1+\exp\left(\alpha_{0,m} + \alpha_m x_{k,m}\right)},\ m = 1, \dots, j, k = 1, \dots, K_m. \quad (56)$$

Step 3 : For $i = 1, \dots, j,$ compute

$$Z_m \leftarrow \sum_{k=1}^{K_m} y_{m,k} + \omega_m \prod_{k=1}^{K_m} \left(1 - p_{m,k}\right) \quad (57)$$

- Step 4 : $\left(\beta_0, \beta\right) \leftarrow \mathrm{PR}\left(s_1, \dots, s_j; Z_1, \dots, Z_j\right)$
- Step 5 : $\left(\alpha_{0,m}, \alpha_m\right) \leftarrow \mathrm{LR}\left(x_{m,1}, \dots, x_{m,K_m}; Z_m, \dots, Z_m - \sum_{u=1}^{K_m-1} y_u\right).$

ここで，PR()とLR()はそれぞれ通常のポアソン回帰およびロジスティック回帰による回帰係数の推定を表す．上記のStep1からStep5を繰り返すことで，回帰係数に対する最尤推定値が得られる．

一方，一般化線形ソフトウェア信頼性モデルでは要因数に応じた回帰係数が必要となる．一般的に，パラメータ数を多くすればするほどデータへの適合は良くなるが，モデルの汎化能力を高めることはできないことが知られている．このような問題に対処するためには不要な要因を削除して要因を選別する必要がある．これは要因選択と呼ばれ，回帰を基にしたモデルでは一般的に行われる手続きである．

要因選択手法には，Devianceを使う手法や検定による手法があるが，ここではAICを用いた手法を用いる．AICの定義式を再掲する．

$$\mathrm{AIC} = -2（最大対数尤度）+ 2（パラメータ数）. \tag{58}$$

これは最尤法における真のモデルとのバイアスをパラメータ数によって近似した指標であり，AICを最小にするモデルが最良のモデルとなる．このことから，AICを最小にする要因の組み合わせを求めることで，データへの過適合を解消し汎化能力の高いモデルを得ることができる．

具体的な手続きでは，全ての要因の組み合わせを総当たり的に調べることは不可能であるため変数増減法を用いる．変数増加法では，加えることによってAICを最も低くする要因から優先的に選択する手続きを，AICが増加するまで行う．これによって，AICを最も小さくする要因の組み合わせを近似的に求める．本研究でも，AICをもとにした変数増減法を適用する．

**(2) カーネル法に対するパラメータ推定および近似手法の適用**

カーネル法は元のデータを高次元空間に写像した上での線形モデルによる分析であるため，カーネル関数で得られたデータを新たな要因データとした推定を行うことになる．しかしながら，カーネル法で高次元空間へ写像した要因データは元のデータ数と同じか，それ以上の数となるため，一般的な最尤法の適用では過適合の問題が生じる．一方で，要因がデータ間の内積で与えられているため，先に述べたような要因選択を行うことができない．そこで，カーネル法を適用したモデルでは正則化（regularization）が行われることが多い．正則化を用いた最尤法は罰則付き最尤法などとも呼ばれ，罰則項$P(\lambda)$を用いて次の罰則付き尤度を最大にするようなパラメータ推定を行う．

$$\log \mathcal{L}(\theta|D) + \lambda P(\theta). \tag{59}$$

罰則項は，平滑化などモデルに必要な事前知識を用いて設定されることが多いが，汎用的に用いられる罰則項としては次のようなものがある．

- パラメータベクトルのL1ノルム：

$$P(\theta) = \sum_{k=1}^{K} |\theta_K| . \tag{60}$$

- パラメータベクトルの L2 ノルムの二乗：

$$P(\theta) = \sum_{k=1}^{K} {\theta_K}^2 . \tag{61}$$

一方で，罰則の強さを表すパラメータ（正則パラメータあるいは罰則パラメータ）$\lambda$を決める必要がある．$\lambda$が小さすぎると最尤推定値に近づくため過適合の問題が生じる．一方で，罰則項が強すぎると全てのパラメータが一定の値へと近づき，真のモデルと大きくかけ離れてしまう．正則化パラメータの決定は一般的な統計においても難しい問題であり，クロスバリデーションなどの手法が提案されている．ここでは，ABIC（Akaike Bayesian

Information Criterion）を用いた決定手法を考える．ABICは主に階層ベイズモデルのパラメータ決定に使われる手法である．いま，パラメータの事前分布を$p(\theta;\lambda)$とする．ここで$\lambda$は事前分布を支配するパラメータであり，超パラメータと呼ばれる．階層ベイズモデルでは元のモデルパラメータ$\lambda$と事前分布で混合したモデルを考える．つまり，

$$p(x;\lambda) = \int p(x;\theta)p(\theta;\lambda)d\theta\ . \quad (62)$$

となり，新たにλをパラメータとするモデルが得られる．このモデルに対するAICがABICと呼ばれる指標である．混合モデルに対するAICであるため，AICが小さいほどデータへの過適合を除いた最良のモデルになる．正則化最尤法は一般的に適切な分布を与えることで，階層ベイズモデルに帰着される．その意味で，ABICを最小にするような正則化パラメータを決めることでAICの観点から最良のモデル選択を行うことができる．しかしながら，正則化最尤法に対する厳密なABICは算出が難しいため，実用面では次の近似的なABICを算出する．

$$\text{ABIC} \approx -2\text{罰則付き対数尤度の最大値} + \log\det I_F \quad (63)$$

ここで$I_F$はフィッシャー情報行列であり，ここではパラメータ数がモデル決定に影響しないためパラメータ数に関係する項は省いてある．上記を最小にするような$\lambda$を見つけることで，正則化パラメータの決定が行える．
また，カーネル法では回帰係数の数が多いことから計算時間が通常の一般化線形ソフトウェア信頼性モデルに比べてかかる可能性がある．そこで，次の二点の工夫を考える．

**高速な数値計算ライブラリの利用**

一般化線形ソフトウェア信頼性モデルのパラメータ推定では，そのEMアルゴリズムの中でポアソン回帰およびロジスティック回帰に関するパラメータを重み付き最小二乗法で解く必要がある．カーネル法を適用した場合，特に，これらの回帰係数がデータ数と同等あるいはそれ以上となるため，この計算に多くの時間が費やされる．そこで高速化に対するの一つの工夫として，重み付き最小二乗法に対して高速な数値計算ライブラリを適用することとした．具体的には，LAPACK（Linear Algebra Package）におけるdggglm関数の利用することとした．dggglmは重み付き最小二乗法を解く高速かつ安定な（悪条件にも対応できる）ライブラリであり，内部的にはQR分解を利用して最小二乗問題の解を得ている．次の図に関数の概要該当ライブラリの仕様を示す．

```
* DGGGLM solves a general Gauss-Markov linear model (GLM) problem:
*
*          minimize || y ||_2   subject to   d = A*x + B*y
*              x
*
*  where A is an N-by-M matrix, B is an N-by-P matrix, and d is a
*  given N-vector. It is assumed that M <= N <= M+P, and
*
*             rank(A) = M    and    rank( A B ) = N.
*
*  Under these assumptions, the constrained equation is always
*  consistent, and there is a unique solution x and a minimal 2-norm
*  solution y, which is obtained using a generalized QR factorization
*  of the matrices (A, B) given by
*
*     A = Q*(R),   B = Q*T*Z.
*           (0)
*
*  In particular, if matrix B is square nonsingular, then the problem
*  GLM is equivalent to the following weighted linear least squares
*  problem
*
*               minimize || inv(B)*(d-A*x) ||_2
*                   x
*
*  where inv(B) denotes the inverse of B.
```

図 3-2-1：dgggｌm 関数の概要(http://www.netlib.no/netlib/lapack/double/dggglm.f).

**ブートストラップサンプリングの利用**

もう一つの工夫としては，データ増加を直接的に軽減する手法としてサンプリング（ブートストラップサンプリング）の活用を検討した．ブートストラップサンプリングとは取得したデータから再サンプリングすることで擬似的に同じデータセットを作成する手法の総称である．ここでは，各モジュールに対して静的メトリクスとして得られたカーネル値に対するデータからランダムにモジュールを限定した場合のサンプルを擬似的に生成し，この擬似的なデータに対して推定と予測を行う．これは元のデータに対するサブセットとなるため，この作業を十分な数のデータセット（例えば100から1000セット）に対して行いそれらの算術平均によってパラメータの推定と予測を行う．

### 3.2.3 課題と問題点

推定における課題は，データに対するスケーラビリティの確保と正則化パラメータの推定である．データに対するスケーラビリティ確保は当初の想定される課題でもあった．高速な数値計算ライブラリを用いることで実用的なサイズの推定を行えることは確認できたが，一つのプロジェクト全てのファイル単位を扱うような場合では，より高速かつ省メモリな方法が必要となる．一方，今回の研究を通じて，正則化パラメータの決定が単純に回帰係数を推定するよりも，数倍あるいは数十倍の計算時間を必要とすることがわかった．現在行っている手法はABIC最小化であるが，これにはフィッシャー情報行列の導出が必要とされる．しかしながら，フィッシャー情報行列は非常に誤差に対して敏感であり，正定値行列になることは原理的に知られているが，かなり高精度推定を行わないとこの条件がくずれる．そのためABICの計算に使うフィッシャー情報行列の算出には，対処療法的に複数の初期値から何度も検証という手続きを導入した．この計算にコストがかかっており，正則化パラメータを決定する手法の検討も含めて今後対応する必要がある．

## 3.3 研究目標３「データを用いたモデルの有効性検証」

### 3.3.1 検証実験の概要

#### (1)検証内容

実際の開発で計測されたテストデータおよび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の分析を行い，それまでに試作したツールを用いてインタフェースや推定アルゴリズムの効率性に関する検証を行う．ここでは主として，「有効性検証（一般化線形モデル）」と「有効性検証（カーネル法）」の作業項目からなる．

「有効性検証（一般化線形モデル）」では，メトリクス情報として，テストケース数，テスト労力，テストカバレッジ情報，バグレポート枚数，ソフトウェア複雑性メトリクス，コード行数を実際のソフトウェア開発からのデータとして利用した．また，モデルを使って信頼性に寄与する要因の分析を行い，モデルの有効性と妥当性について検証した．さらに，この検証では，ユーザインタフェースおよび推定アルゴリズムに関するテスト工程も兼ねており，使いやすさや計算時間などの測定や開発へのフィードバックを行うこととした．

「有効性検証（カーネル法）」では，テストケース情報として予め準備されたテストベッドにおける各テストケースの入力とテスト実行パスの計測からテストカバレッジを算出し，テストカバレッジとフォールト検出能力の関係を検証する．さらに，ソースコードの情報としては文字列やPDG (Program Dependence Graph)などの構造データを基にしたたたみ込みカーネルの適用を行う．さらに，「パラメータ推定アルゴリズムの開発」で作成した近似手法の性能テストおよびツールのインタフェースのテストも兼ねる．

#### (2)想定される課題と解決策

ソフトウェアメトリクスを含めいくつかのデータは過去の開発プロジェクトから採取す

ることが難しいため，オープンソースプロジェクトのソースやテストベンチマークとして公開されているプログラムをターゲットとしてテストに関する情報の採取を行う．また，ソースコードの情報には，既存のツール（CCFinder など）を使い，労力の低減を図る．

### 3.3.2 検証実験の計画と環境

本研究で提案する新たなモデル「一般化線形ソフトウェア信頼性モデル」と「カーネル法を適用した一般化線形ソフトウェア信頼性モデル」に関する性能評価として次の三つの実験を行った．これらは，実験(i)「適合性評価」が一般化線形ソフトウェア信頼性モデルの有効性検証，実験(ii)「ソースコード情報を用いた信頼性評価」と(iii)「テストケースの入力情報を用いた信頼性評価」がカーネル法を適用した一般化ソフトウェア信頼性モデルの有効性検証に対応する．

#### (i)適合性評価

一般化線形ソフトウェア信頼性モデルのデータへの適合性を調査する．特に，従来の手法である単純なNHPPモデルと，静的メトリクスだけをポアソン回帰として利用するソフトウェア信頼性モデル，動的なテストメトリクスをロジスティック回帰として利用するソフトウェア信頼性モデルとの比較を行う．特にここでは，AICを用いた評価を行うことで，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．

実験で用いるデータはApache Software Foundation (ASF)で管理されているTomcatプロジェクトおよびLenyaプロジェクトを対象とした．また，バージョンの違いによる影響も見るため，Tomcatについては幾つかのバージョンに対するデータを収集する．さらに，プロジェクト毎の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た．各プロジェクトのモジュール構成，バグレポート収集期間は次の通りである．

Tomcat

Javaサーブレットを提供する．主としてJava言語で作成されており，Apache Web Serverと組み合わせて利用されることが多い．Bugzillaの管理では以下のプログラム群（モジュール）で構成される．

- 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

対象とする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）と見なして分析する．それぞれのバージョンにおけるバグ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．

Lenya

Java/XMLベースのコンテンツマネジメントシステムである．Bugzillaで管理されている．モジュールは以下の通りとなる．webapps(manager):管理用ウェブアプリケーション

- AccessControl:サイトに対するアクセスコントロール
- FromEditor:WYSIWYGのエディタ
- KupuIntegration:WYSIWYG XHTMLエディタKupuとの連係
- LuceneIntegration:検索エンジンLuceneとの連係
- NavigationFramework:メニューなどのナビゲーションアイテムの自動生成
- SiteManagement:Lenyaコア
- TinyMCEIntegration:エディタTinyMCEとの連係
- UsecaseFramework:JXテンプレートとJavaを用いてユースケースを実装するフレームワーク
- Workow:XMLによるワークフローエンジン

バグデータの収集期間は2003/5から2013/1となっている．

本研究で提案した一般化線形ソフトウェア信頼性モデルでは，静的メトリクスと総フォールト数の因果関係，動的なテストメトリクスとフォールト検出確率の因果関係をそれぞれポアソン回帰モデル，ロジスティック回帰モデルで関連づけている．そのため，実データの分析においても静的メトリクスと，動的メトリクスの収集が必要となる．

実際に静的メトリクスは各プロジェクトのモジュールにおけるソースコードから直接算出した．算出にはSource Monitorを用いて対応するモジュールが格納されているディレクトリ全体に対して次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
- AvgStmts/Method(St/Me):単一メソッドあたりのステートメント数
- MaxComplexity(MCx):最大複雑度
- MaxDepth(MDp):最大のネストの深さ
- AvgDepth(ADp):平均のネストの深さ
- AvgComplexity(ACx):平均複雑度

一方，動的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各プロジェクトにおける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）

表3-3-1から表3-3-4は実際に計測した各プロジェクトの静的メトリクスを表している．また，図3-3-1から図3-3-12はTomcat5からTomcat7およびLenyaにおける累積のバグ報告数，コメント数，投票数を表している．従来のNHPPを用いた分析では累積のバグ報告数に対する収束具合だけから残存バグ数や信頼度（一定期間中にバグが検出される確率）の算出を行っていた．一方，本研究で提案した一般化線形ソフトウェア信頼性モデルでは，静的メトリクスと総フォールト数の因果関係，動的なテストメトリクスとフォールト検出確率の因果関係をそれぞれポアソン回帰モデル，ロジスティック回帰モデルで関連づけている点が大きな特徴となっている．

表 3-3-1:Tomcat5 の静的メトリクス.

| Module | St | MCx | CI | MeCI |
|---|---|---|---|---|
| webapps | 13784 | 52 | 175 | 5.96 |
| catalina | 53367 | 92 | 612 | 10.87 |
| connectors | 34580 | 205 | 409 | 11.14 |
| jasper | 19017 | 93 | 223 | 8.99 |
| servlets | 4451 | 26 | 139 | 6.3 |

表 3-3-2:Tomcat6 の静的メトリクス.

| Module | Files | Lines | Statements | Branches | Calls | Comments | Classes | Methods/Class | Avg Stmts/Method | Max Complexity | Avg Depth | Avg Complexity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| catalina | 574 | 167671 | 60064 | 19.4 | 32442 | 31.5 | 719 | 10.6 | 5.78 | 93 | 2.24 | 2.89 |
| connectors | 22 | 11956 | 4231 | 21.1 | 2272 | 25 | 43 | 14.3 | 5.1 | 31 | 2.12 | 2.64 |
| jasper | 113 | 45829 | 20681 | 19.8 | 11984 | 27.2 | 245 | 8.95 | 6.98 | 94 | 2.6 | 3.3 |
| manager | 16 | 6832 | 2908 | 21.6 | 2155 | 21 | 23 | 6.48 | 16.57 | 37 | 2.57 | 5.74 |
| servlet | 114 | 21892 | 2719 | 8.8 | 1107 | 47.8 | 106 | 6.08 | 1.85 | 26 | 1.53 | 1.59 |

表 3-3-3:Tomcat7 の静的メトリクス.

| Module | Files | Lines | Statements | % Branches | Calls | % Comments | Classes | Methods/Class | Avg Stmts/Method | Max Complexity | Max Depth | Avg Depth | Avg Complexity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Catalina | 619 | 188585 | 68583 | 19.5 | 37977 | 30.2 | 784 | 11.27 | 5.65 | 296 | 9+ | 2.25 | 2.93 |
| Connectors | 22 | 12951 | 4539 | 20.8 | 2540 | 23.1 | 38 | 17.37 | 5.13 | 296 | 9+ | 2.19 | 3.12 |
| Jasper | 123 | 47715 | 21410 | 19.8 | 12266 | 26.7 | 237 | 9.48 | 7.02 | 100 | 9+ | 2.61 | 3.29 |
| Manager | 17 | 7350 | 3091 | 20.9 | 2259 | 20.1 | 14 | 13.36 | 13.78 | 35 | 9+ | 2.54 | 5.12 |
| Servlet & JSP API | 141 | 23407 | 3305 | 8.1 | 1211 | 50.2 | 115 | 7.1 | 1.63 | 25 | 9+ | 1.45 | 1.59 |

表 3-3-4:Lenya の静的メトリクス.

| Module | Files | Lines | Statements | % Branches | Calls | % Comments | Classes | Methods/Class | Avg Stmts/Method | Max Complexity | Max Depth | Avg Depth | Avg Complexity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Access Control | 63 | 8970 | 3631 | 11.9 | 2153 | 34.5 | 57 | 6.88 | 5.88 | 24 | 7 | 1.77 | 2.18 |
| Form Editor | 66 | 9271 | 4302 | 20.5 | 2010 | 17.2 | 18 | 31.83 | 6.63 | 103 | 9+ | 2.57 | 4.37 |
| Kupu Integration | 357 | 20964 | 9592 | 13.2 | 4686 | 12.7 | 9 | 78.11 | 12.48 | 35 | 9+ | 1.97 | 3.22 |
| Lucene Integration | 67 | 6419 | 2181 | 15.3 | 1026 | 31.3 | 38 | 6.26 | 5.47 | 25 | 8 | 1.98 | 2.63 |
| Navigation Framework | 15 | 799 | 369 | 10.8 | 205 | 26.5 | 9 | 4.11 | 5.3 | 12 | 8 | 1.61 | 2.45 |
| Site Management | 87 | 6045 | 1936 | 12.7 | 1379 | 19.4 | 33 | 5.15 | 7.19 | 12 | 7 | 1.83 | 2.61 |
| TinyMCE Integration | 26 | 986 | 302 | 13.6 | 136 | 26.3 | 2 | 21 | 5.19 | 18 | 6 | 1.63 | 3.04 |
| Usecase Framework | 52 | 5847 | 2358 | 13.9 | 1187 | 34.6 | 33 | 9.18 | 4.99 | 21 | 8 | 1.88 | 2.23 |
| Workflow | 44 | 4252 | 1637 | 12.2 | 941 | 30.2 | 31 | 4.94 | 6.51 | 19 | 8 | 1.82 | 2.36 |

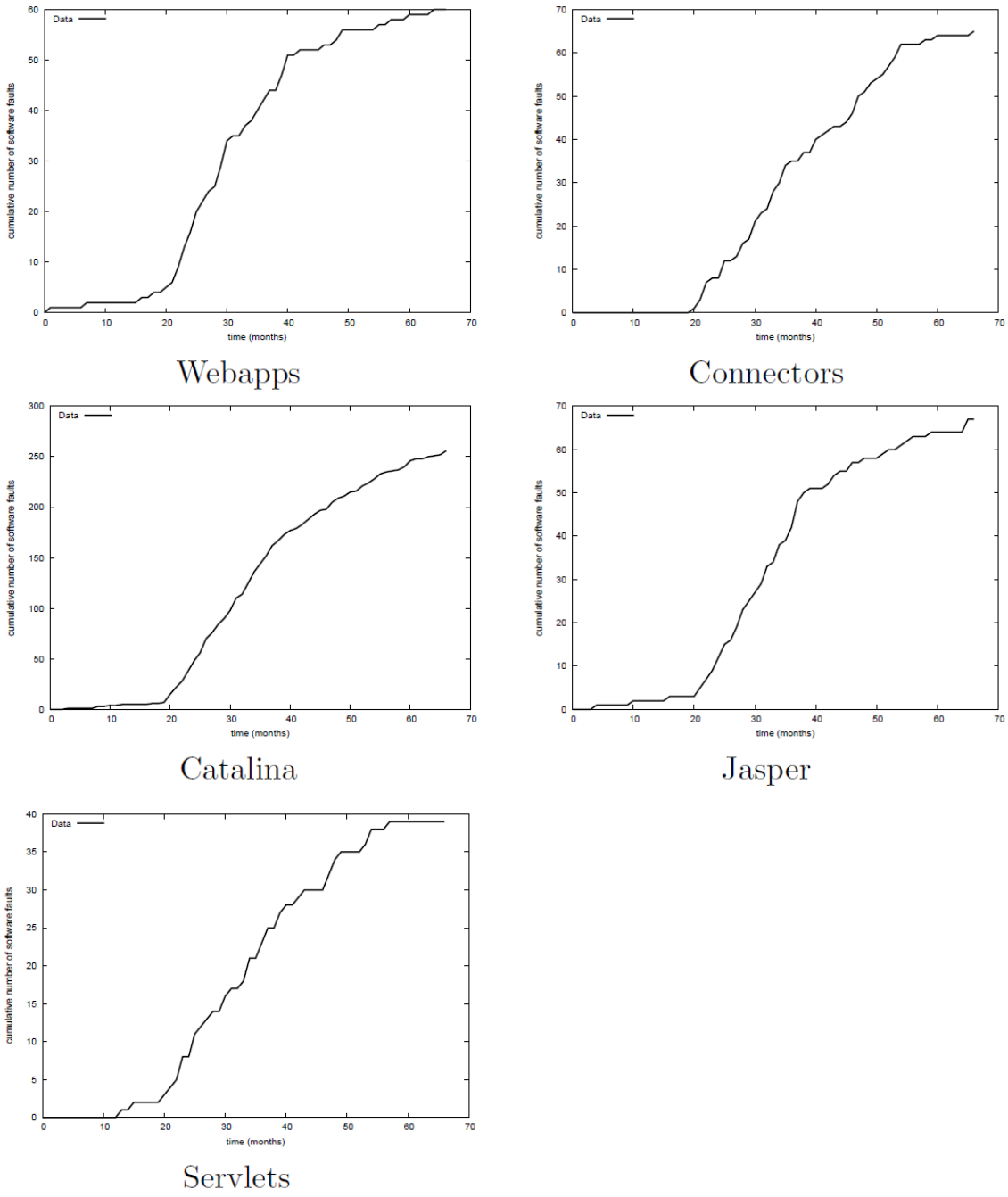


図 3-3-1:Tomcat5 における各モジュールの累積バグ報告数.

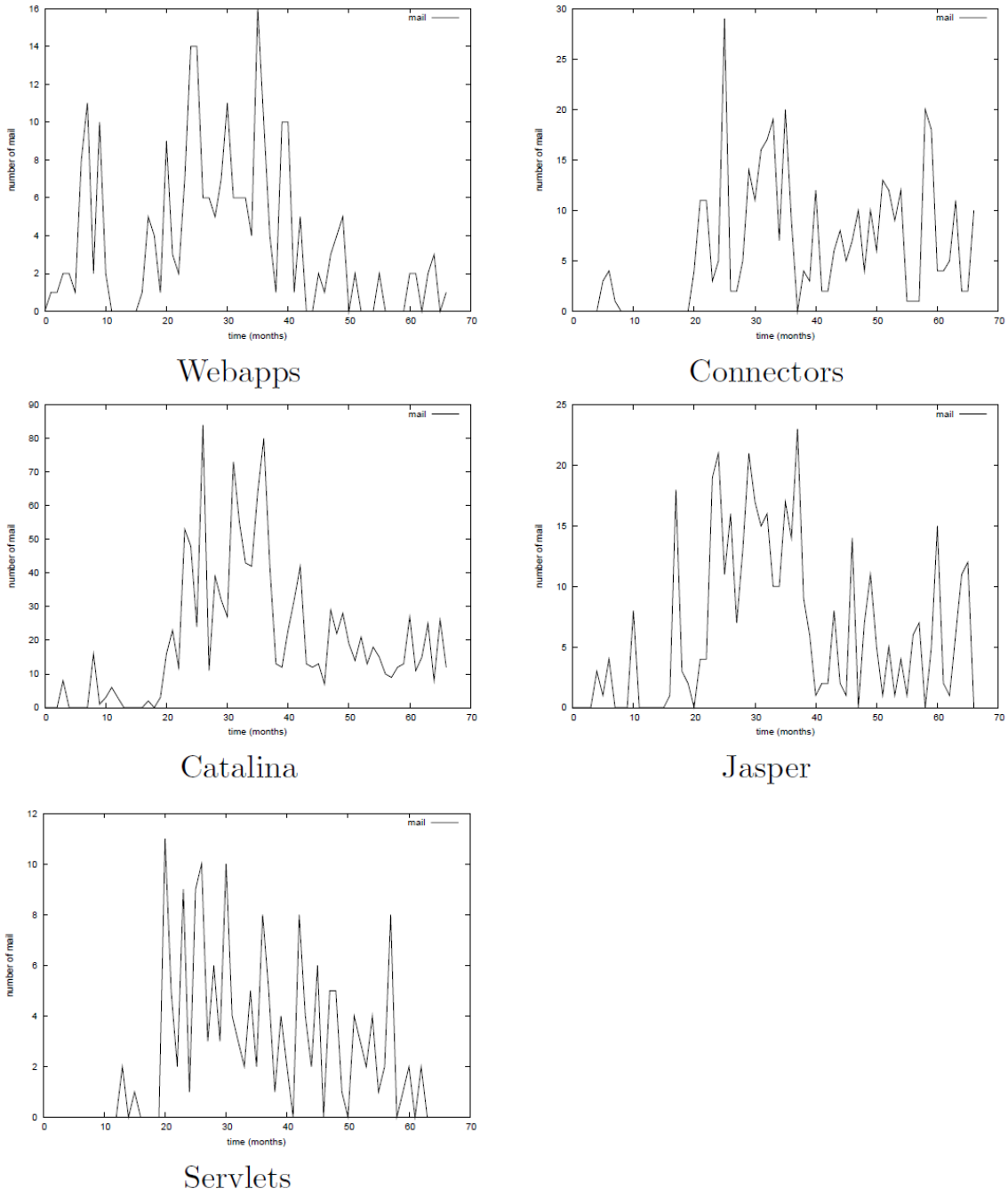


図 3-3-2:Tomcat5 における各モジュールのコメント数.

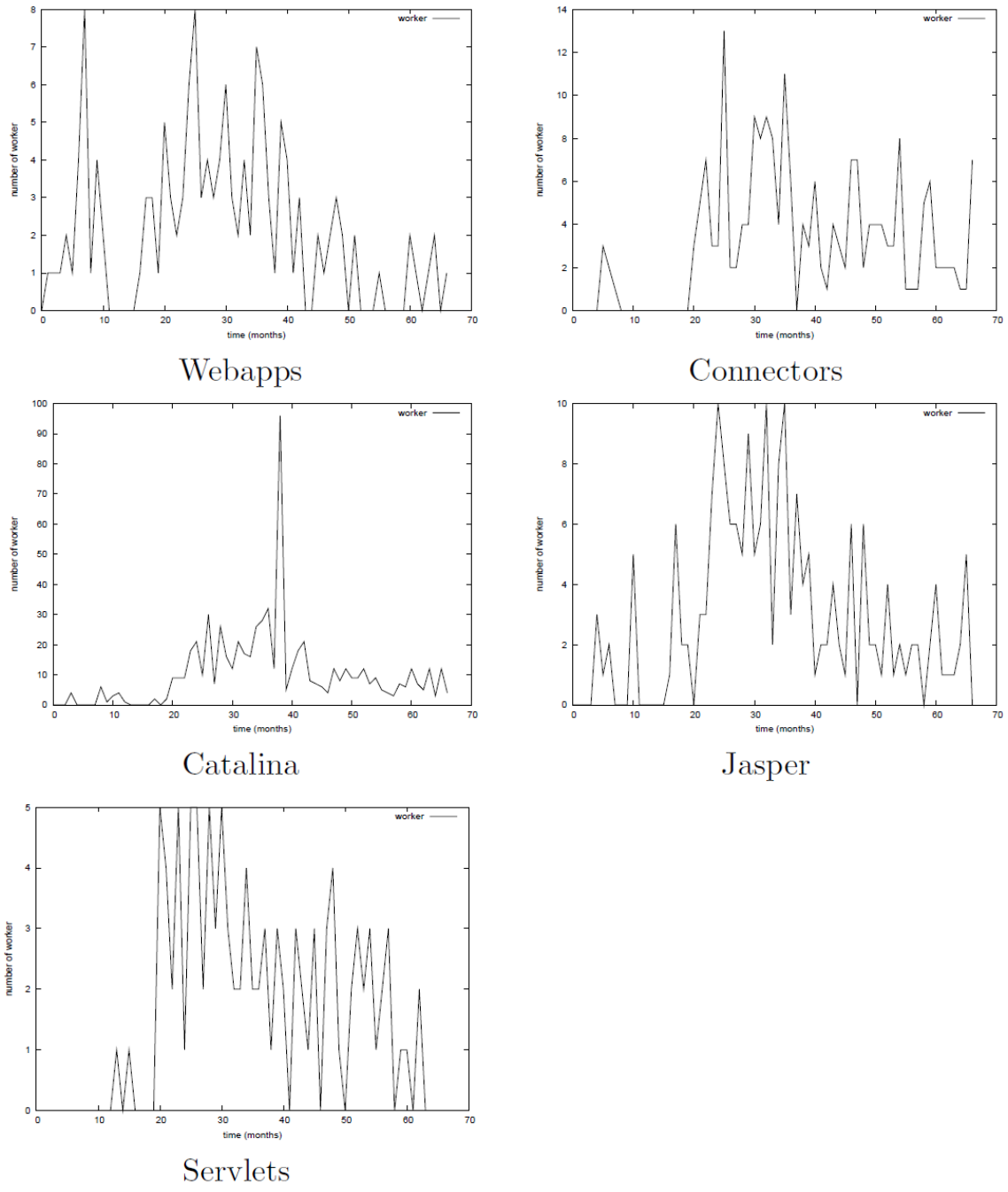


図 3-3-3:Tomcat5 における各モジュールのコメントを投稿した人数.

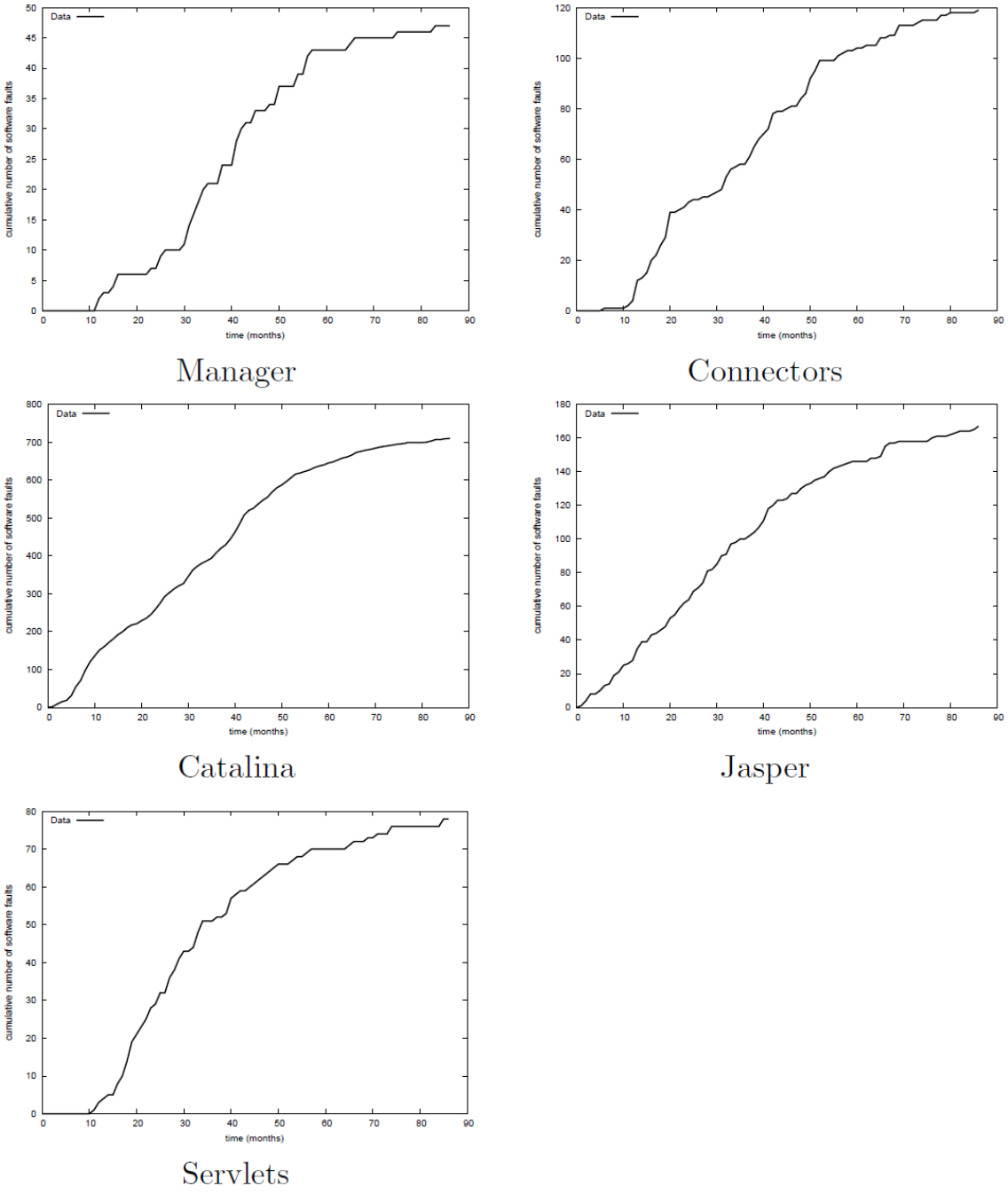


図 3-3-4:Tomcat6 における各モジュールの累積バグ報告数.

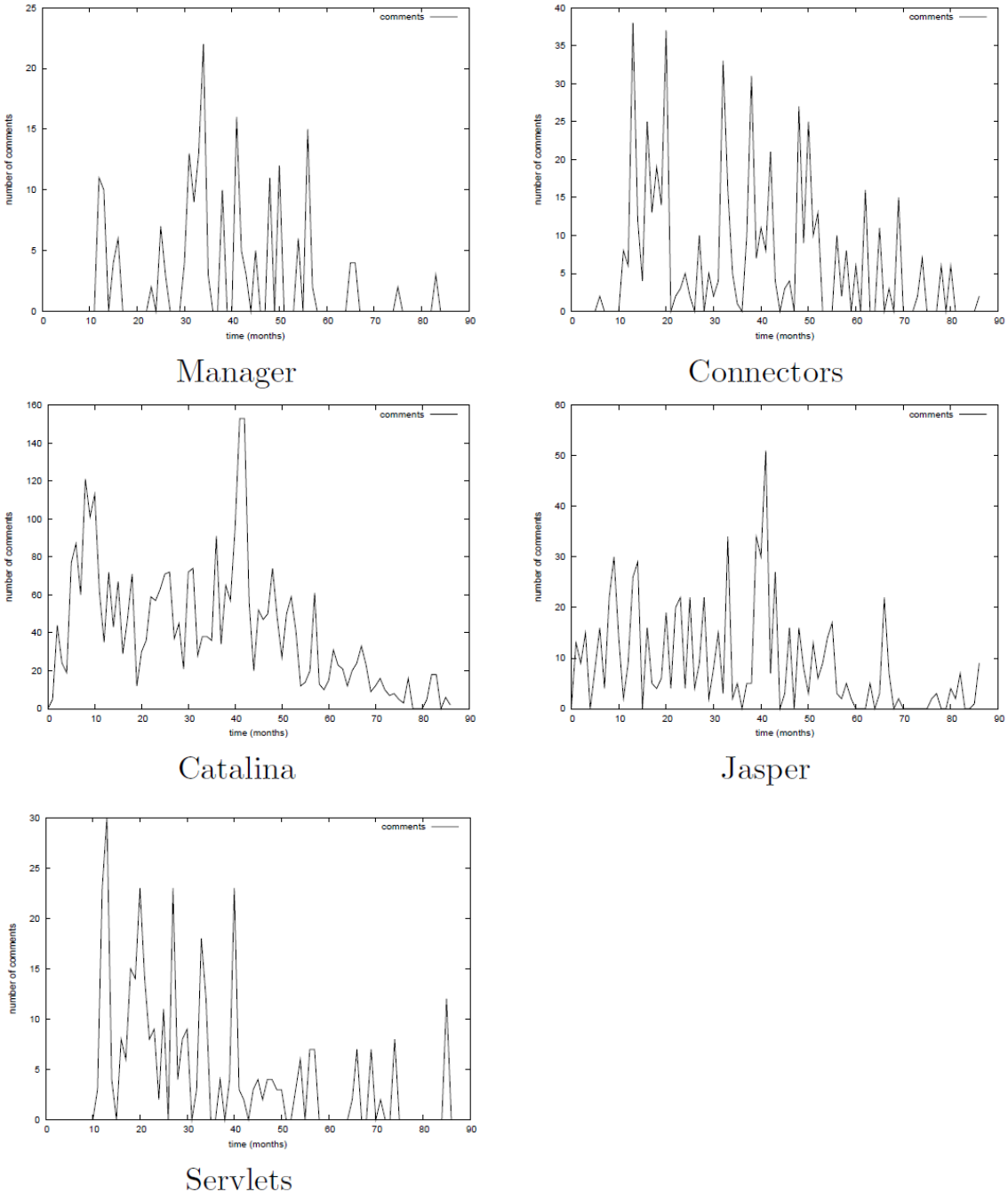


図 3-3-5:Tomcat6 における各モジュールのコメント数.

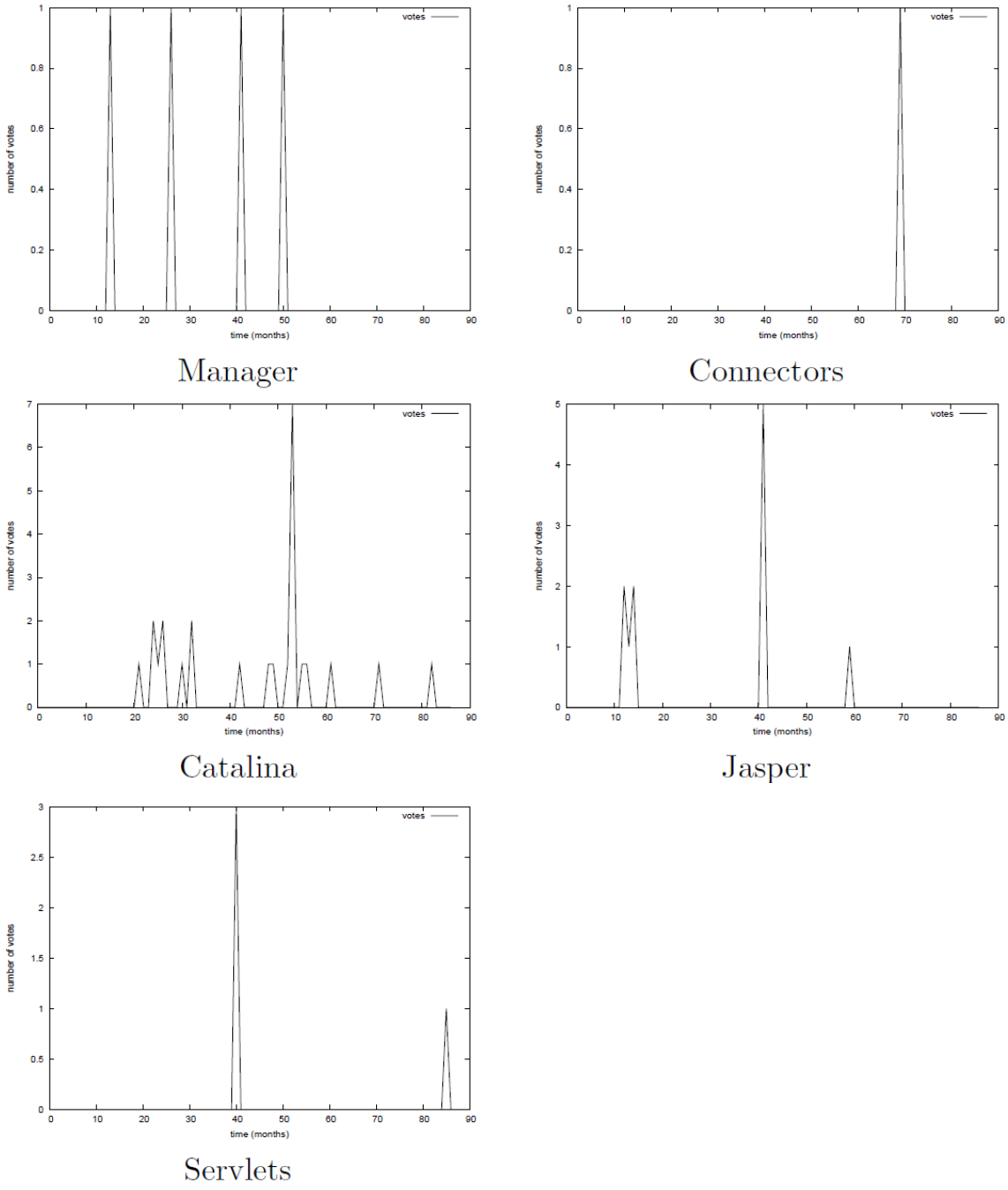


図 3-3-6:Tomcat6 における各モジュールの投票数.

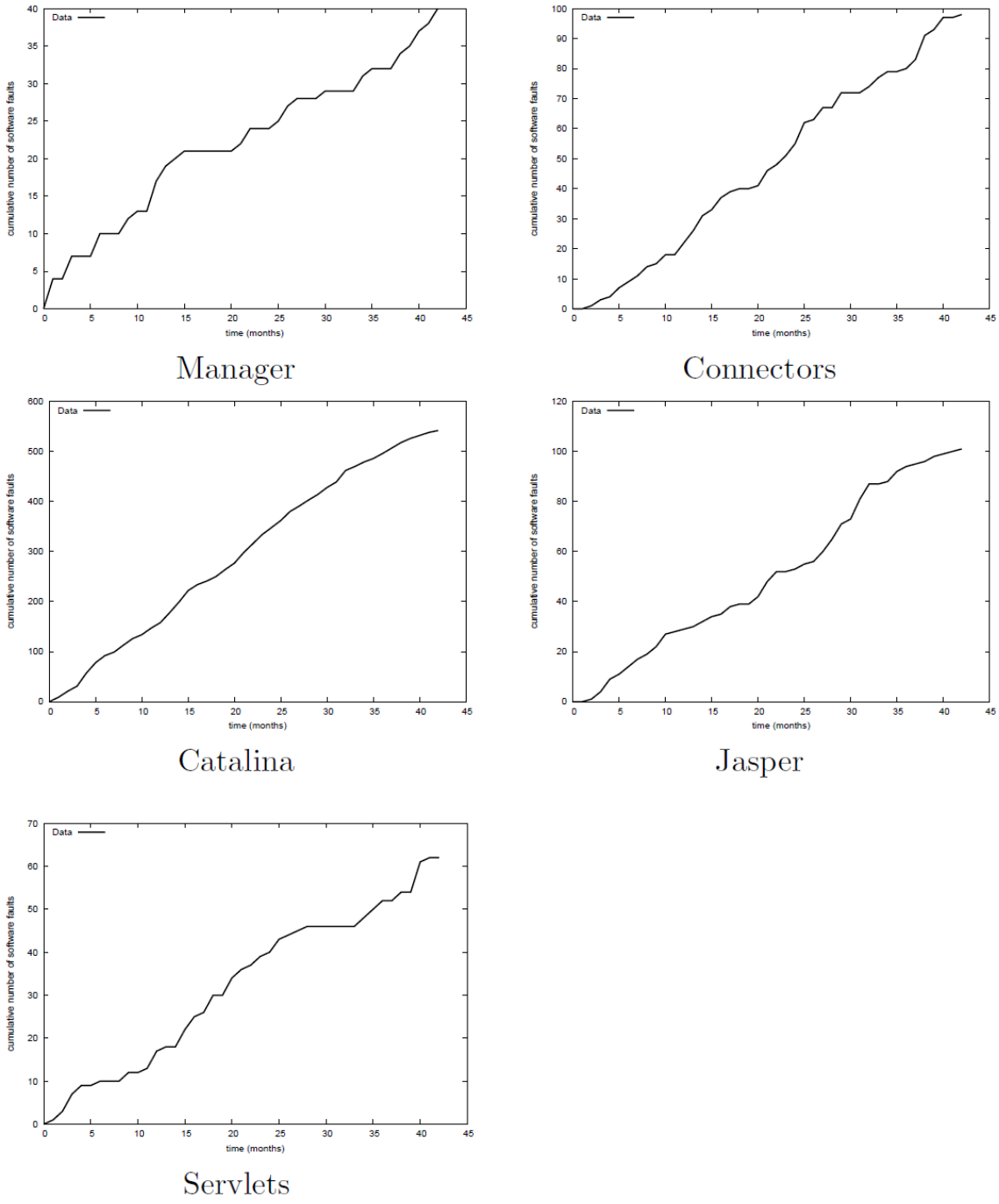


図 3-3-7:Tomcat7 における各モジュールの累積バグ報告数.

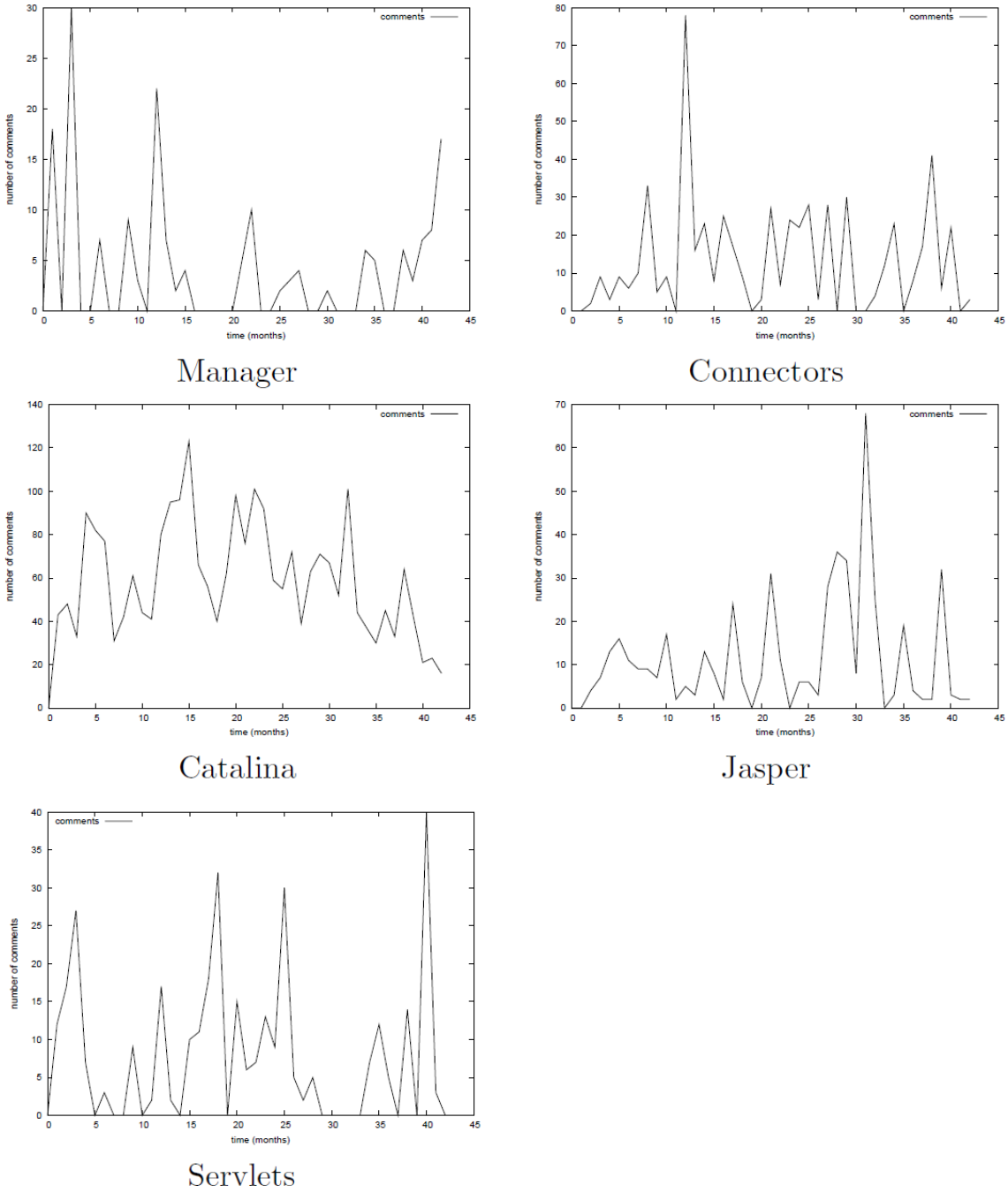


図 3-3-8:Tomcat7 における各モジュールのコメント数.

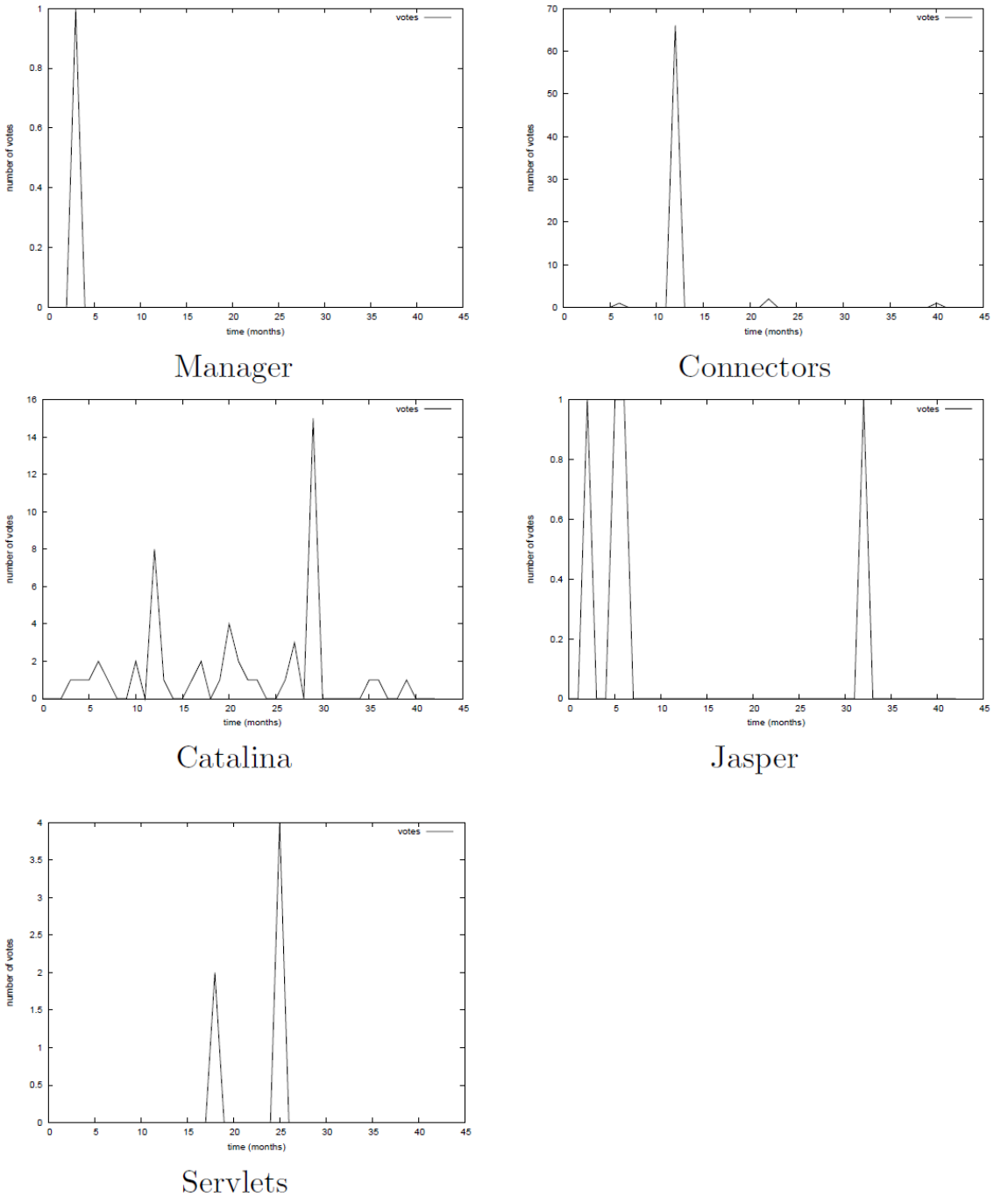


図 3-3-9:Tomcat7 における各モジュールの投票数.

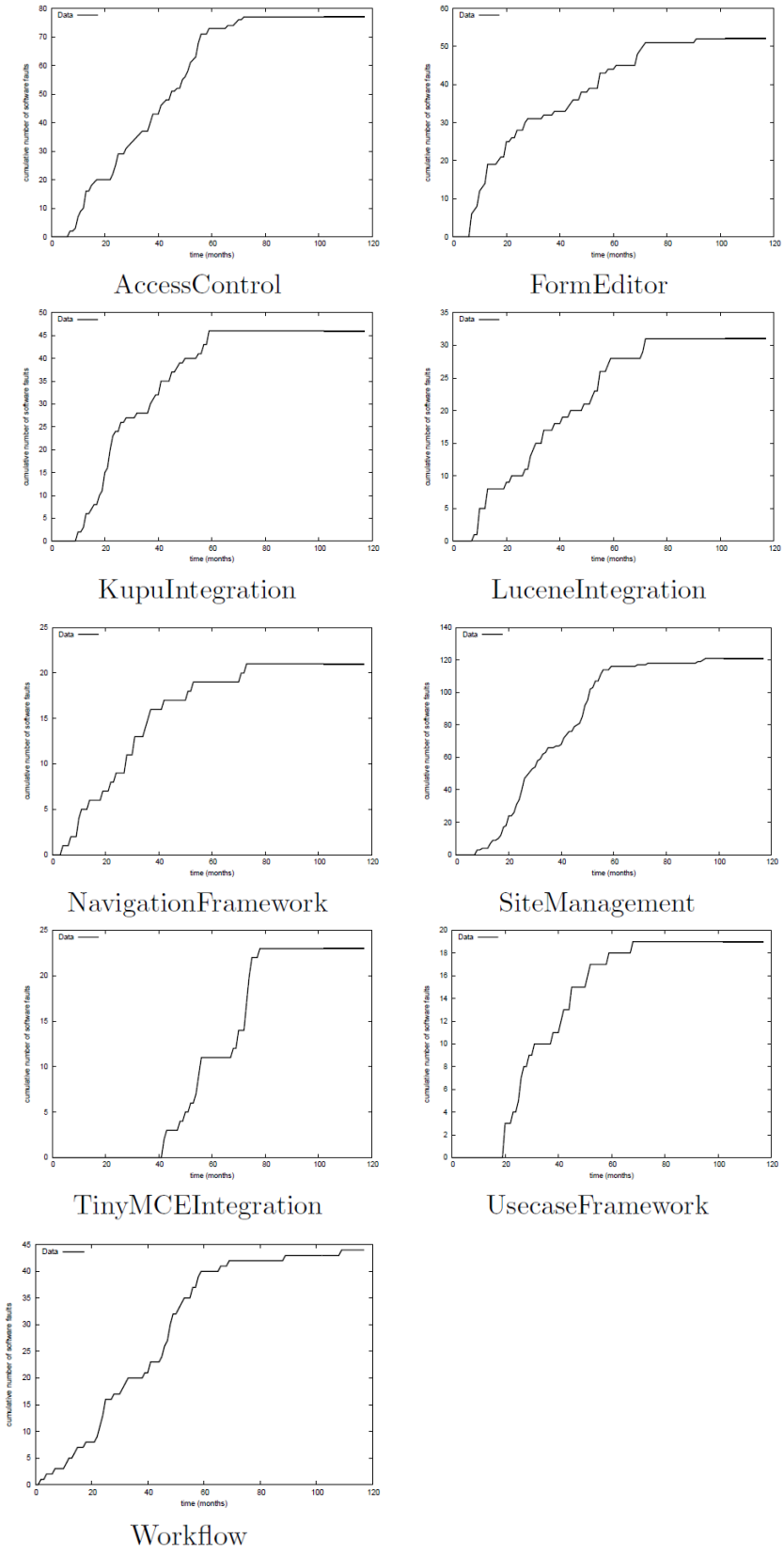


図 3-3-10:Lenya における各モジュールの累積バグ報告数.

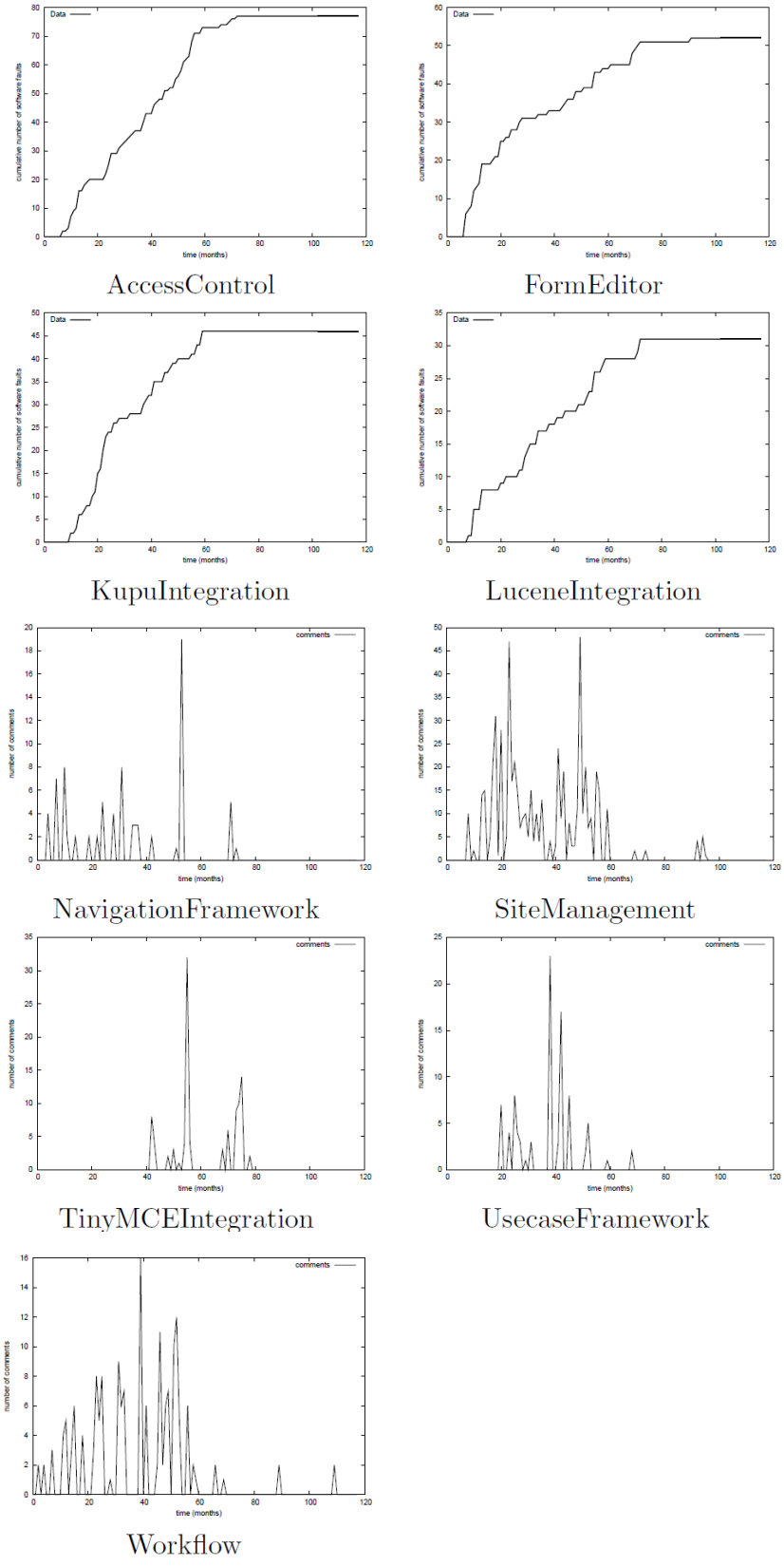


図 3-3-11:Lenya における各モジュールのコメント数.

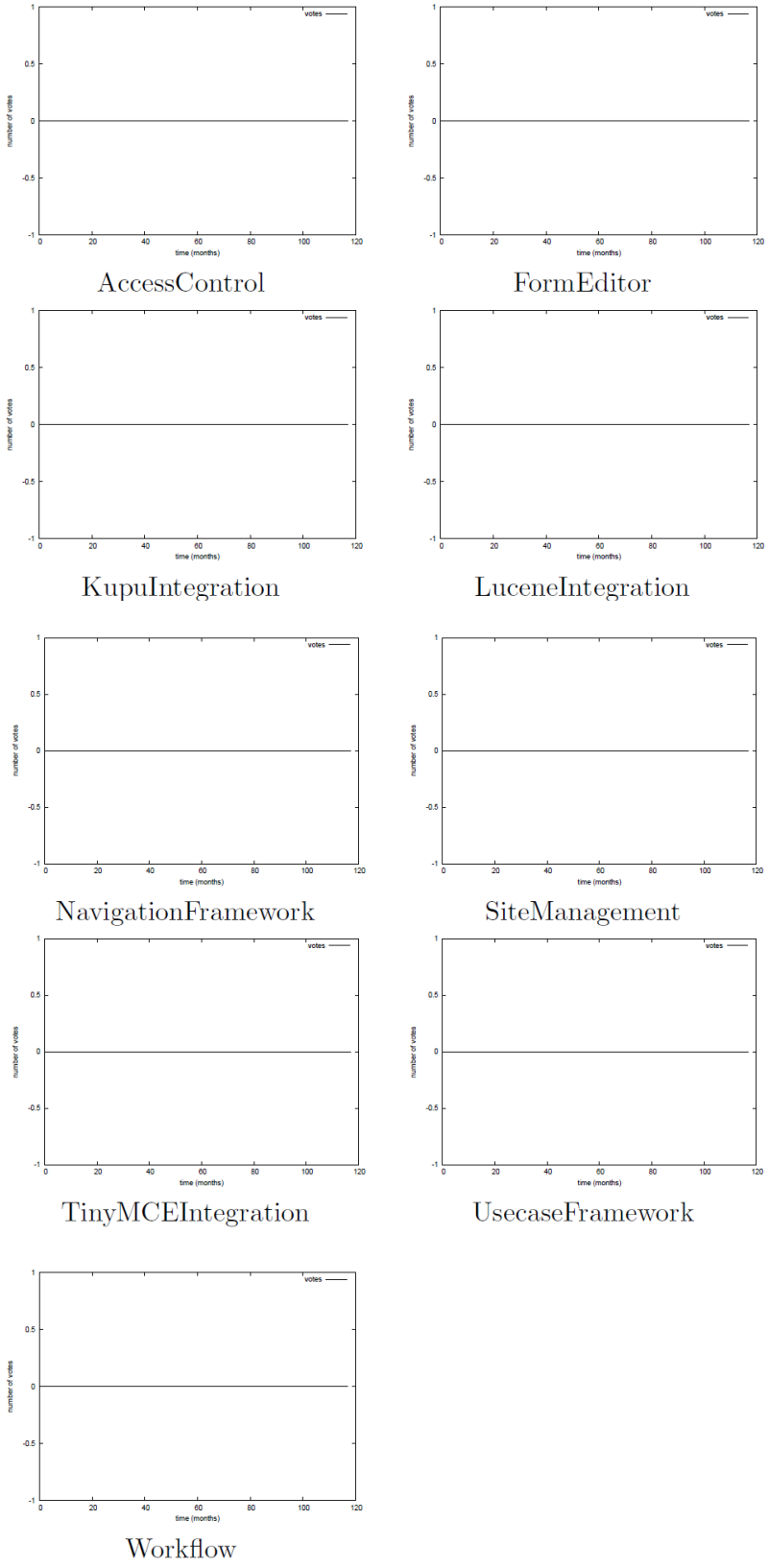


図 3-3-12:Lenya における各モジュールの投票数.

実験は，まずモジュール毎で独立に(1)NHPPモデルの推定，(2)ロジスティック回帰によるソフトウェア信頼性モデルの推定を行った．(1)ではSRATSに含まれる11種類のモデル全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，ロジスティック回帰によるソフトウェア信頼性モデルでは，先に述べた変数増減法による要因選択を行って最小のAICを示す要因の組み合わせを求めている．さらに，各プロジェクトのモジュールに対する静的メトリクスを用いて，(3)ポアソン回帰によるソフトウェア信頼性モデルの推定と(4)一般化線形ソフトウェア信頼性モデルの推定を行った．ポアソン回帰によるソフトウェア信頼性モデルの推定では，事前に各モジュールに適用するNHPPモデルを選ぶ必要がある．これは，(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ロジスティック回帰によるソフトウェア信頼性モデルで使用する要因を選ぶ必要がある．これも(2)で単一モジュールに対して選択された要因を用いることとした．加えて，ポアソン回帰によるソフトウェア信頼性モデルおよび一般化線形ソフトウェア信頼性モデルでは，静的メトリクスに対する要因選択を行う必要がある．これも，先に述べた変数増減法により最小のAICを示す要因の組み合わせを選んだ．この静的メトリクスに対する変数増減法を行う場合，各モジュールのNHPPモデルとロジスティック回帰の動的メトリクスを固定することに注意する．つまり，一般化線形ソフトウェア信頼性モデルでは静的メトリクスと動的メトリクスを含めて全ての組み合わせを対象として，AICを最小にする組み合わせを見つける必要があるが，計算量が膨大になるため，近似的に，(a)動的メトリクスの組み合わせを決定する，(b)(a)で決定した動的メトリクスに対して静的メトリクスの組み合わせを決定するという二段階の手順を適用した．

**(ii)テストケースの入力情報を用いた信頼性評価**

実験(ii)では，テストケースの入力情報を用いたソフトウェア信頼性評価手法の有効性を検証する．テストパス情報（カバレッジ情報など）は非常に有用であるが，実際の現場における応用を鑑みると，テストパスとほぼ同等の情報量を持つテストケース入力情報の方が扱いやすいため，実験(ii)ではテストケースの入力情報に着目した．モデルにはカーネル法を適用したロジスティック回帰によるソフトウェア信頼性モデルを適用する．通常のロジスティック回帰によるソフトウェア信頼性モデルでは数値化された動的メトリクスのみしか扱えないのに対して，カーネル法では構造データ（文字列やグラフ構造）を直接入力値とすることができる．その一例として，ここではテストケースの入力情報に着目する．一般的にテストケースとは，ソフトウェアへの入力と期待される結果から構成される．例えば，システムテストにおけるテストケースは「電気ポットの電源を入れ給湯ボタンを押す．お湯が一定量以上あった場合はお湯が出る．」のような文字列で表現されることが多い．そこで，文字列に対するカーネルを応用して，テストケースの入力情報とフォールト検出確率を関連させた手法を試行する．

検証に使用するデータは，SIR:Software-artifact Infrastructure Repository(http://sir.unl.edu)で提供されているプログラムとテストケースを利用して生成する．SIRで提供されているプログラムはソフトウェアテストケース生成アルゴリズムなどのバグ検出能力を検証するためのベンチマークとなっており，正しく動作するプログラム，フォールトが埋め込まれたプログラム，テストベッド（テストケースの集合）

からなる．具体的にここでは，replaceと呼ばれる文字列置換のプログラムを対象とした．ReplaceはC言語で記述されたプログラムであり，いくつかのバグが埋め込まれたバージョンが準備されている．また，入力はコマンドラインの標準入力から，置換元のテキスト，置換パターンを入力する．その入力に対して，合計5,542個のテストケースが準備されている．これらの，検証用のプログラムに対して，以下の手順で疑似的なテストデータを生成した．

- 5,542個のテストケースをランダムに並び替える．
- 12個のバグを埋めこんだプログラムを作成する．
- テストケースを先に生成した順番で投入する．
- バグを埋め込んだプログラムの出力と正しく動作するプログラムの出力を比較する．
- 比較した出力が異なっていたら，予め埋め込んだバグを一つずつ除去していき，その出力が正しく動作するプログラムと同じになるまでバグの除去を繰り返す．
- 比較した出力が同じになる，または，バグを除去したプログラムの出力が正しいプログラムと同じになったら，次のテストケースを投入し，上記の手順を繰り返す．

このとき，入力したテストケースの入力（この場合は，置換元のテキストと置換パターン）と，そのテストケースで除去されたバグの個数を記録することで，疑似的にソフトウェアテストのデータを作成することができる．

図3-3-13は投入するテストケース数を増やしていったときの累積バグ数の変化を示している．また，図3-3-14は最初の10個のテストケース入力である．図3-3-14で見るようにこれらは，単純な文字列となっていることに注意する．

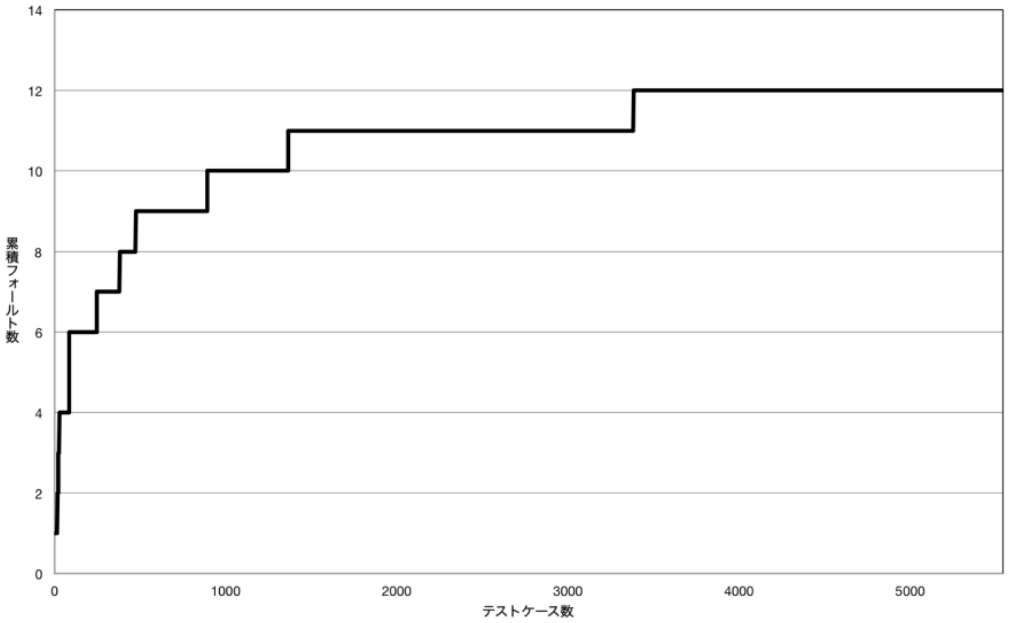


図 3-3-13:replace プログラムのテストにおける累積バグ数のふるまい.

| | |
|---|---|
| #1 | '-?' 'a&' < \|abcd\| -a \|abcd\| \n |
| #2 | ' ' '@%@&' < \|abcd\|  \|abcd\| \n |
| #3 | ' ' 'NEW' < \|abcd\|  \|abcd\| \n |
| #4 | ' ' 'NEW' < |
| #5 | ' ' 'rY NCDt+32Ilu>dr~s^1Q{0*{RLN>Muz' < y\n2?GhetmK\n y \n |
| #6 | ' ' <\nBpF\n\n |
| #7 | ' *' '@%&a' < \|abcd\|  \|abcd\| \n |
| #8 | ' *' 'a&' < \|abcd\|  \|abcd\| \n |
| #9 | ' *' 'a&' < \|abcd\|  \|abcd\| \n |
| #10 | ' *' 'a&' < |

図 3-3-14:replace プログラムのテストケースの一部.

カーネル法を適用したロジスティック回帰によるソフトウェア信頼性モデルでは，テストケースに対する何らかの類似度（内積，カーネル）を求める必要がある．ここでは，類似度にテストケースの編集距離に基づいた手法を適用する．編集距離はLevenshtein距離とも呼ばれ，二つの文字列の一方を他方へ変更するときに，何回の文字列挿入と削除を行うかを数え，それを二つの文字列間と距離として定義する．図3-3-15は最初の10個のテストケース入力に対する編集距離の行列を表している．

| | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| #1 | 0 | 7 | 7 | 25 | 50 | 24 | 8 | 5 | 4 | 22 |
| #2 | 7 | 0 | 4 | 23 | 47 | 24 | 5 | 6 | 5 | 23 |
| #3 | 7 | 4 | 0 | 19 | 46 | 23 | 7 | 6 | 5 | 23 |
| #4 | 25 | 23 | 19 | 0 | 49 | 10 | 26 | 25 | 22 | 4 |
| #5 | 50 | 47 | 46 | 49 | 0 | 47 | 48 | 48 | 48 | 51 |
| #6 | 24 | 24 | 23 | 10 | 47 | 0 | 27 | 25 | 22 | 11 |
| #7 | 8 | 5 | 7 | 26 | 48 | 27 | 0 | 3 | 6 | 24 |
| #8 | 5 | 6 | 6 | 25 | 48 | 25 | 3 | 0 | 3 | 21 |
| #9 | 4 | 5 | 5 | 22 | 48 | 22 | 6 | 3 | 0 | 18 |
| #10 | 22 | 23 | 23 | 4 | 51 | 11 | 24 | 21 | 18 | 0 |

**図 3-3-15:最初の 10 個のテストケース間の距離行列.**

いま，テストケース$x$とテストケース$y$の編集距離を$D(x;y)$とすると，類似度（内積，カーネル）を次の式で与えることにする．

$$\kappa(x,y) = \exp(-D(x,y)). \tag{64}$$

さらに，投入されるテストケース数はテスト進捗状況によって変わるため，検証では次の二つの場合を考えた．

- Case1:50個のテストケース投入が完了した時点で，50個のテストケース入力から得られるカーネルでフォールト検出確率を表す．つまり，フォールト検出確率は次の式で与えられる．

$$\log\frac{p_i}{1-p_i} = \gamma_0 + \sum_{k=1}^{50}\gamma_k\kappa(x_k, x_i). \quad (65)$$

ここで，$p_i$は$i$番目のテストケースのフォールト検出確率，$x_i$は$i$番目のテストケースを表している．

- Case2:100個のテストケース投入が完了した時点で，100個のテストケース入力から得られるカーネルでフォールト検出確率を表す．フォールト検出確率は次の式で与えられる．

$$\log\frac{p_i}{1-p_i} = \gamma_0 + \sum_{k=1}^{100}\gamma_k\kappa(x_k, x_i). \quad (66)$$

パラメータ推定において，総フォールト数に関するパラメータと，カーネルに対する回帰係数$\gamma_0, \ldots, \gamma_K$を推定する必要がある．しかしながら，先に述べたようにカーネル法の適用ではデータ数よりもパラメータ数が多くなることがある．そのため，正則化によりデータへの過適合を防ぐ．ここでは，汎用的に適用可能なL2ノルムの二乗による罰則項を用いる．また，正則化（罰則）パラメータλはABICを最小にするように決定する．

**(iii)ソースコード情報を用いた信頼性評価**

ソースコード情報を用いたソフトウェア信頼性評価手法の有効性を検証する．先のテストケース入力に対するソフトウェア信頼性モデルと同様に，カーネル法を適用することでソースコードの文字列情報を直接扱うことができる．特に，ここでは静的メトリクスに対してポアソン回帰を用いたソフトウェア信頼性モデルを基にして，カーネル関数を用いたソースコード情報の取り扱いについて調査する．

実験には，(i)で言及したASFのTomcat6プロジェクトにおけるCatalinaモジュール中のソースコード（31ファイル）を用いる．ソースコード情報を利用するためには，カーネル（類似度）の定義が必要となる．類似度には様々なものが考えられるが，ここではコードクローン検出ツールCCFinderが出力する二つのプログラム間の共通するクローンセット数を用いる．クローンセットとはソースコード中でコピーされた部分的なコードであり，これらを多く共有するプログラムほど同じ機能に関連する，つまり，類似度が高いものと仮定した．図3-3-16はCCFinderが出力したCatalinaモジュール内のプログラム間のクローンセット数である．

| Matrix | ApplicationCon | ApplicationCon | ApplicationDisp | ApplicationFilte | ApplicationFilte | ApplicationFilte | ApplicationHttp |
|---|---|---|---|---|---|---|---|
| ApplicationCon | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ApplicationCon | 0 | 24 | 0 | 0 | 0 | 0 | 0 |
| ApplicationDisp | 0 | 0 | 10 | 0 | 0 | 0 | 0 |
| ApplicationFilte | 0 | 0 | 0 | 8 | 0 | 0 | 0 |
| ApplicationFilte | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| ApplicationFilte | 0 | 0 | 0 | 0 | 0 | 14 | 0 |
| ApplicationHttp | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| ApplicationHttp | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ApplicationReq | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ApplicationRes | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AprLifecycleLis | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Constants.java | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ContainerBase | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DummyReques | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DummyRespon | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JasperListener | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JreMemoryLea | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| NamingContext | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardConte | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardConte | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardEngine | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardEngine | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardHost.j | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardHostV | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardPipelin | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StandardServe | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

図 3-3-16:Tomcat6 の Catalina モジュールにおける共有するコードクローンセット数.

具体的に，ファイル$j$ $(j = 1, \ldots, 31)$，に対する総フォールトとして次の回帰式を利用することになる．

$$\log \omega_j = \gamma_0 + \sum_{k=1}^{31} \gamma_i c_{i,j} / M. \quad (67)$$

ここで$c_{i,j}$はファイル$i$とファイル$j$に共通するコードクローンセット数，$M$は全てのファイル間において最大のコードクローンセット数である．

また，ここでの検証ではテスト進捗によるバグ検出数のデータが必要となる．実験(i)では，ASF Bugzilla に報告されているバグの累積数をデータとして利用した．しかしながら，これらのデータから細かいプログラム単位でのバグ数を数えることは現実的に（実用的にも）難しいと考えられる．そこで，ここでの検証ではSVNリポジトリに記録されている各ファイルの修正履歴からファイルの修正(modify)の回数を記録することとした．パラメータ推定は，実験(ii)と同様にL2ノルムの二乗を用いた罰則項を用い，正則化（罰則）パラメータλと決定にはABICを用いた．

### 3.3.3 結果と分析

(i)適合性評価

表3-3-5から表3-3-8は各プロジェクトに対してNHPPモデルのみ(NHPP)，ロジスティック回帰によるソフトウェア信頼性モデル(Logit)，ポアソン回帰によるソフトウェア信頼性モデル(Poi)，一般化線形ソフトウェア信頼性モデル(GLSRM)をそれぞれ適用したときのAICの値を示している．

表 3-3-5: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 3-3-6: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 3-3-7: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 3-3-8:Lenya に対する AIC.

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| Access Control | 218.76 | 179.56 | – | – |
| Form Editor | 196.62 | 137.79 | | |
| Kupu Integration | 152.79 | 121.29 | | |
| Lucene Integration | 142.98 | 95.48 | | |
| Navigation Framework | 103.06 | 84.49 | | |
| Site Management | 272.55 | 233.99 | – | – |
| TinyMCE Integration | 100.06 | 85.92 | – | – |
| Usecase Framework | 88.06 | 85.67 | – | – |
| Workflow | 167.69 | 146.29 | – | – |
| Total (project) | 1442.56 | 1170.48 | 1440.44 | 1168.48 |

表から，いずれの場合でも

$$\text{NHPP} \geq \text{Poi} \geq \text{Logit} \geq \text{GLSRM} \tag{68}$$

の関係が得られる．つまり，適合性の観点からはGLSRMが最も良いことがわかった．また，静的なメトリクスを扱うポアソン回帰によるソフトウェア信頼性モデルと，動的なメトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデルを比較すると，動的なメトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデルの方が適合性が高い．これは初期の総フォールト数を決めるための情報よりも，観測されたバグ報告数から得られる情報の方が適合性に対して強い影響を及ぼしていることを示唆している．ここでは，モジュール数が比較的少ないため，このような結果になったものと考えられる．つまり，モジュール数が多い状況では，静的メトリクスからの情報の影響がより大きくなるものと予想される．

また表3-3-9から表3-3-12は，各プロジェクトにおけるモジュールに対してAICにより選択されたNHPPモデル，動的メトリクス，静的メトリクスの種類を表している．

表 3-3-9: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 | | |

表 3-3-10: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 3-3-11: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 | | |

表 3-3-12:Lenya において選択されたモデルとメトリクス．

| Module | NHPP | Logit | Poi | GLSRM |
|---|---|---|---|---|
| Access Control | Truncated Extreme-Value Min SRGM | comments, ctime | Ln, St, Br, Ca, St/Me, MCx | Fl, St, Br, Ca, Cm, Cl, MCx |
| Form Editor | Log Extreme-Value Max SRGM | comments | | |
| Kupu Integration | Log-Normal SRGM | comments, ctime | | |
| Lucene Integration | Truncated Extreme-Value Min SRGM | comments, ctime | | |
| Navigation Framework | Log Extreme-Value Min SRGM | comments, ctime | | |
| Site Management | Gamma SRGM | comments, ctime | | |
| TinyMCE Integration | Truncated Extreme-Value Min SRGM | comments, ctime | | |
| Usecase Framework | Log Extreme-Value Max SRGM | time, comments, ctime | | |
| Workflow | Truncated Logistic SRGM | comments | | |

選択されたメトリクスを見ると，静的メトリクスでは最大複雑度(MCx)が比較的多く，動的メトリクスではコメント数が多くのデータで選択される結果となった．これは，静的メトリクスでは従来から重要とされてきたコード行数や複雑度が大きく影響することを意味している．また，コメント数などのプロジェクトのアクティビティを表す指標は，フォールト検出確率を決定するのに利用可能であることがわかる．

(ii)テストケースの入力情報を用いた信頼性評価

図 3-3-17 と図 3-3-18 は Case1 と Case2 において，ABIC の基準により決定された正則化パラメータを用いた場合にモデルから算出した累積バグ数の平均値（平均値関数，赤線）と実際に観測された累積バグ数（緑線）を表している．

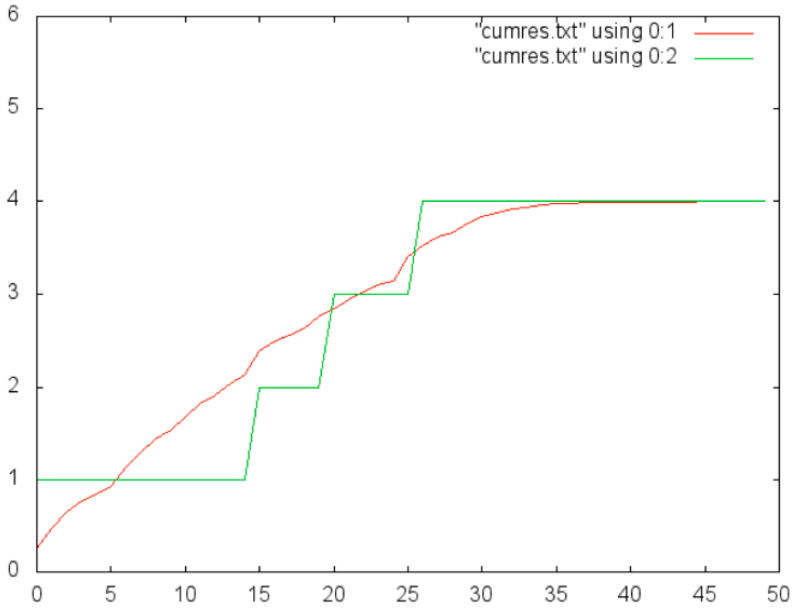


図 3-3-17:累積バグ数と推定した平均値関数(Case1)．

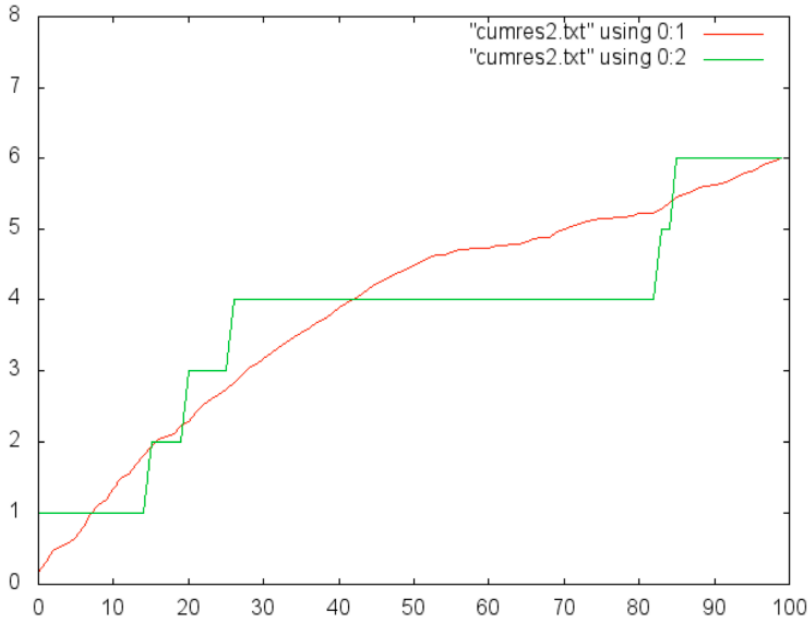


図 3-3-18:累積バグ数と推定した平均値関数(Case2)．

また，推定された正則化パラメータλと対応するABICおよび，その時のバグ数に関する予想値は次の通りである．

Case1：

- 総期待バグ数:4.0
- 期待残存バグ数:0.005
- ABIC:-136.40
- λ:981.24

Case2：

- 総期待バグ数:7.6
- 期待残存バグ数:1.6
- ABIC:15.26
- λ:1037.5

図3-3-17と図3-3-18から，テストケース入力を用いたカーネル法を用いても，与えられたデータにフィットするモデルを得ることができる．つまり，メトリクスの要因選択という組み合わせ最適化を必要とする問題を扱うことなく，ほぼ同等なデータへの高い適合性が得られることが分かった．一方で，テストケース入力という粒度の細かい情報を用いることによる成果は，この例に関しては残念ながら見受けられなかった．つまり，単純に数値ベクトルとして要約された動的メトリクスを用いた信頼性モデルでも同等の性能を示す可能性があることから，期待したほどの大きな精度の向上は見受けられなかった．

### (iii) ソースコード情報を用いた信頼性評価

表3-3-13はカーネル法を適用したポアソン回帰によるソフトウェア信頼性モデルを用いた推定結果である．

表 3-3-13: ソースコード情報を用いた信頼性評価結果.

| | NHPP | | | Kernel (CCFinder) | | | No. faults |
|---|---|---|---|---|---|---|---|
| | Total faults | Residual faults | FFP | Total faults | Residual faults | FFP | |
| ApplicationContext.java | 19.0429 | 2.0429 | 0.1297 | 19.0455 | 2.0455 | 0.1293 | 1 |
| ApplicationContextFacade.java | 8.1241 | 1.1241 | 0.3250 | 8.1261 | 1.1261 | 0.3243 | 24 |
| ApplicationDispatcher.java | 17.4777 | 0.4777 | 0.6202 | 17.4779 | 0.4779 | 0.6201 | 11 |
| ApplicationFilterChain.java | 11.4338 | 0.4338 | 0.6481 | 11.4340 | 0.4340 | 0.6479 | 8 |
| ApplicationFilterConfig.java | 12.0952 | 1.0952 | 0.3345 | 12.0965 | 1.0965 | 0.3340 | 4 |
| ApplicationFilterFactory.java | 11.0313 | 0.0313 | 0.9692 | 11.0313 | 0.0313 | 0.9692 | 14 |
| ApplicationHttpRequest.java | 9.6243 | 2.6243 | 0.0725 | 9.6377 | 2.6377 | 0.0715 | 3 |
| ApplicationHttpResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 3 |
| ApplicationRequest.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| ApplicationResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| AprLifecycleListener.java | 36.8628 | 4.8628 | 0.0077 | 36.8708 | 4.8708 | 0.0077 | 2 |
| Constants.java | 6.3282 | 0.3282 | 0.7202 | 109.1404 | 103.1404 | 0.0000 | 0 |
| ContainerBase.java | 10.6579 | 0.6579 | 0.5179 | 10.6584 | 0.6584 | 0.5177 | 118 |
| DummyRequest.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 1 |
| DummyResponse.java | 5.9751 | 0.9751 | 0.3772 | 5.9775 | 0.9775 | 0.3762 | 3 |
| JasperListener.java | 6.6377 | 0.6377 | 0.5285 | 109.1538 | 103.1538 | 0.0000 | 0 |
| JreMemoryLeakPreventionListener.java | 317.3526 | 301.3526 | 0.0000 | 109.7239 | 93.7239 | 0.0000 | 0 |
| NamingContextListener.java | 14.9339 | 0.9339 | 0.3930 | 14.9344 | 0.9344 | 0.3928 | 88 |
| StandardContext.java | 58.6706 | 3.6706 | 0.0255 | 58.6726 | 3.6726 | 0.0254 | 294 |
| StandardContextValve.java | 10.5598 | 0.5598 | 0.5713 | 10.5602 | 0.5602 | 0.5711 | 2 |
| StandardEngine.java | 15.8002 | 4.8002 | 0.0082 | 15.8276 | 4.8276 | 0.0080 | 6 |
| StandardEngineValve.java | 6.5147 | 0.5147 | 0.5977 | 109.1494 | 103.1494 | 0.0000 | 0 |
| StandardHost.java | 16.4417 | 2.4417 | 0.0870 | 16.4463 | 2.4463 | 0.0866 | 40 |
| StandardHostValve.java | 12.7507 | 0.7507 | 0.4721 | 12.7512 | 0.7512 | 0.4718 | 4 |
| StandardPipeline.java | 7.0484 | 0.0484 | 0.9528 | 7.0484 | 0.0484 | 0.9528 | 4 |
| StandardServer.java | 14.6352 | 1.6352 | 0.1949 | 14.6372 | 1.6372 | 0.1945 | 13 |
| StandardService.java | 11.3825 | 0.3825 | 0.6822 | 11.3826 | 0.3826 | 0.6821 | 15 |
| StandardThreadExecutor.java | 21.6689 | 8.6689 | 0.0002 | 21.7568 | 8.7568 | 0.0002 | 6 |
| StandardWrapper.java | 27.4471 | 0.4471 | 0.6395 | 27.4471 | 0.4471 | 0.6395 | 25 |
| StandardWrapperFacade.java | 6.6081 | 0.6081 | 0.5444 | 109.1528 | 103.1528 | 0.0000 | 0 |
| StandardWrapperValve.java | 12.2168 | 0.2168 | 0.8051 | 12.2169 | 0.2169 | 0.8050 | 45 |

この表ではNHPPモデルで評価した際の各ファイルの総バグ（修正）数，残存バグ（修正）数，およびFFP（fault-free probability)と，カーネル法を適用したポアソン回帰によるソフトウェア信頼性モデルで評価した際の各ファイルの総バグ（修正）数，残存バグ（修正）数，およびFFPを示している．また，このときの正則化パラメータは$\lambda = 0.0$となっている．正則化パラメータが0の場合NHPPモデルの結果と同じになることが解析的にわかっているため，NHPPモデルによる評価と大きく変わらない結果となっている．その一方で，幾つかのファイルにおいて推定値が大きく変化しており（赤字で表示），ソースコード情報が何らかの影響を与える結果となっている．

## 3.3.4 課題と問題点

一般化線形ソフトウェア信頼性モデルに関しては，当初想定していた以上のデータ適合性を示すこととなり，本研究における大きな成果となった．その一つの要因としては，リポジトリデータベースにおけるコメント数をプロジェクトのアクティビティに関するメトリクスとして利用した点がある．一方で，カーネル法の適用に関しては，メトリクスの選択をする必要がないという大きな利点は確認できたものの，計算コストが一般化線形ソフ

トウェア信頼性モデルの数倍から数十倍かかるため，実用化のためにはより効率的な計算アルゴリズムおよび実装を行う必要があることが判明した．ただし，汎化能力については将来の利用において有望視されるべきものであるため，手法も含めて，今後の数理的な改善が望まれる．

## 3.4 研究目標４「モデルによる信頼性評価を実装したツールの開発」

### 3.4.1 ツール概要

評価モデルと推定アルゴリズムを実装したツールの開発を行う．開発するツールはExcelとRによるインタフェースをもつ．また，作業は推定アルゴリズムとインタフェースの実装に分割される．推定アルゴリズムの実装はCで行い，共通ライブラリ化することで，Excel，Rの双方から利用できるようにする．Excelによるインタフェースでは，VBA(Visual Basic Application)で記述された既存ツールであるSRATSを元として，C#による再記述を行う．また，Rは新規パッケージとして構築し，Rのパッケージを管理するサイトであるCRAN (Comprehensive R Archive Network)へ投稿可能な標準的な記述で実装する．
推定アルゴリズムはC言語で共通ライブラリ化することで，開発コストおよびパフォーマンスの確保を行う．また，Excelでは既存ツールを参考にすることで開発作業の省力化を目指している．また，ツール普及の観点から，CRANへ投稿可能とすることで世界中からのダウンロードを可能とする．
本報告書では，Microsoft ExcelのAddInとして作成したツールをMSRATS(Metrics-based Software Reliability Assessment Tool on Spreadsheet)，統計処理ツールRのパッケージとして開発したツールをRsrat(Software Reliability Assessment Tool on R)と呼称する．

#### (1)MSRATS の概要

MSRATSはC#を用いたExcel AddInとして開発され，後述するRsratが大量のデータ解析を行う研究者向けであるのに対して，ユーザの使いやすさに主眼をおいた設計となっている．MSRATSは次の特徴を持つ．

- Excelのスプレッドシートからのデータ入力
- 時刻データ・個数データの両方に対応するデータ入力形式
- パラメータ推定および信頼性尺度計算の自動化
- Excelのグラフ描画機能を利用した信頼度関数などのグラフ描画

次のソフトウェア信頼性評価尺度をモデルに基づいて算出することができる．

- 予測総バグ数：単一のソフトウェアモジュール（プログラム単位orシステム単位）に内在する総バグ数の予測値.
- 予測残存バグ数：単一のソフトウェアモジュール（プログラム単位orシステム単位）に現時点で残存しているバグ数の予測値.
- ソフトウェア信頼度関数：一定期間中にバグが発見されない確率.
- FFP(fault-free probability)：現在のモジュールにバグがない確率.
- 各種MTTF(mean time to failure), 累積MTTF, 瞬間MTTF, 条件付きMTTF：バグ発見までの平均時間.
- Median, Betenlife：信頼度が0.5, 0.1になるまでの時間.

MSRATS で扱えるソフトウェア信頼性モデルは次の通り.

- 11種類の典型的なNHPPモデル
- 動的メトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデル
- 静的メトリクスを扱うポアソン回帰によるソフトウェア信頼性モデル
- 一般化線形ソフトウェア信頼性モデル

なお，カーネル法を適用したモデルについては，カーネル値を要因として入力し，正則化最尤法を適用することで実現できる．MSRATSで扱うデータは時刻データ，個数データおよびそれらを両方表現できるデータ（一般化個数データ）となっている．時刻データはバグの発見時刻を記録したデータであり，スプレッドシート上では１コラムのデータとなる．個数データは一定期間中に発見されたバグの個数を記録したデータであり，スプレットシート上では２コラムのデータとなる．左のコラムは観測した時間，右のコラムは対応する区間で発見されたバグの個数となる．一般化個数データは個数データに対して観測区間の終わりでバグの発見があったか無かったを指定できる形式であり，３コラムで表現される．最初のコラムは個数データと同じで観測した時間，２番目のコラムも個数データと同じでバグの個数，３番目のコラムは対応する観測区間の終わりでちょうどバグが発見される場合に1，それ以外のときに0を入力する．一般化個数データは時刻データ，個数データを両方表現することができる.

次に，主に利用するWindowsフォームを説明する.

SRATSフォーム（図3-4-1）

SRATSフォームでは11種類のNHPPモデルを同時に取り扱う．データ入力はRangeと記載されているテキストボックスにシート範囲を記入する．これはテキストボックス横のボタンを押すとセル選択で参照できるようになっている．データ選択後は，Estimateボタンを押すと，自動的にデータ形式を判断し，全てのモデルパラメータ推定を行う．結果は下のリストに表示される．詳細はリストでモデルを選択し，Reportを押すことでグラフ描画などの詳細な信頼性評価尺度が出力される．

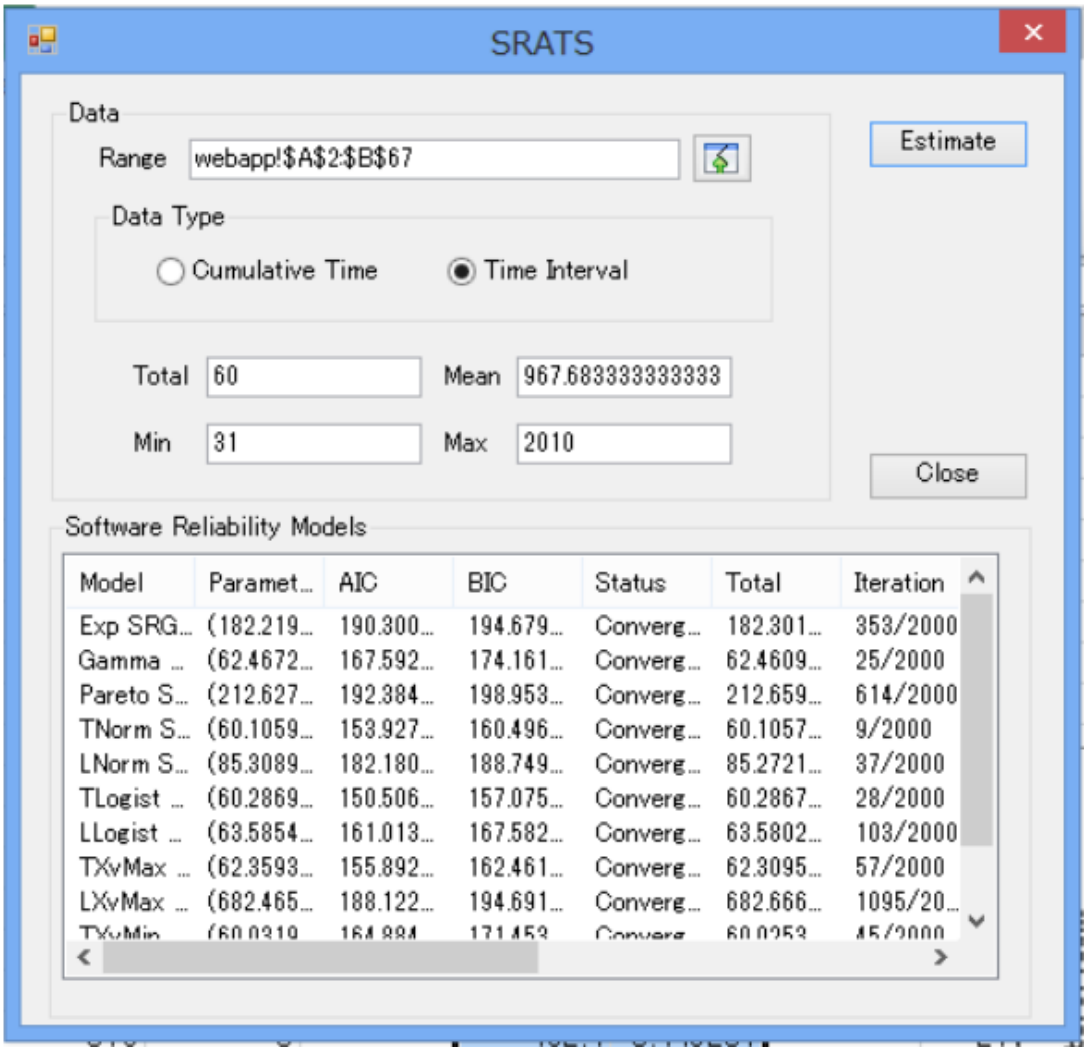


図 3-4-1:SRATS フォーム.

LSRATSフォーム（図3-4-2）

LSRATSフォームでは，ロジスティック回帰によるソフトウェア信頼性モデルを取り扱う．SRATSと同様にデータをRangeで指定する．ロジスティック回帰によるソフトウェア信頼性モデルでは，どの列がバグ個数データか示す必要がある．これはRangeの下にあるドロップダウンリストでデータ選択後に行う．データの設定が完了したのちにEstimateボタンで各メトリクスの回帰係数および総フォールト数のパラメータを推定する．結果は下のリストに表示される．

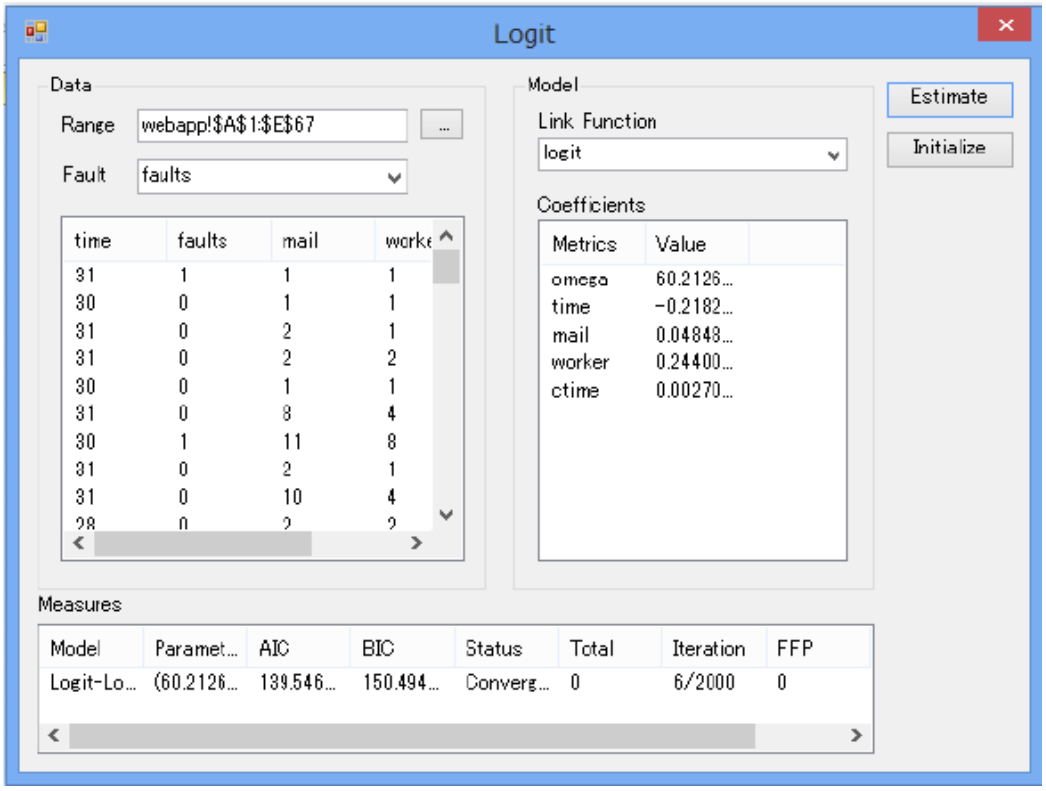


図 3-4-2:LSRATS フォーム.

MSRATSフォーム（図3-4-3）

MSRATSフォームはポアソン回帰によるソフトウェア信頼性モデルを取り扱う．MSRATSフォームでは先に，各モジュールとそれらの静的なメトリクスをMetricsテキストボックスで指定する．指定後に中央のリストにモジュールとメトリクス情報が表示される．リストでモジュールをダブルクリックすることでSRATSあるいはLSRATSフォームが表示され，モジュール毎のバグデータを登録し，モジュール毎のモデルパラメータ推定を行うことができる．全てのモジュールでモデルを指定した後，MSRATS上のEstimateボタンを押すことで，モジュールの静的メトリクスを考慮した推定を行うことができる．

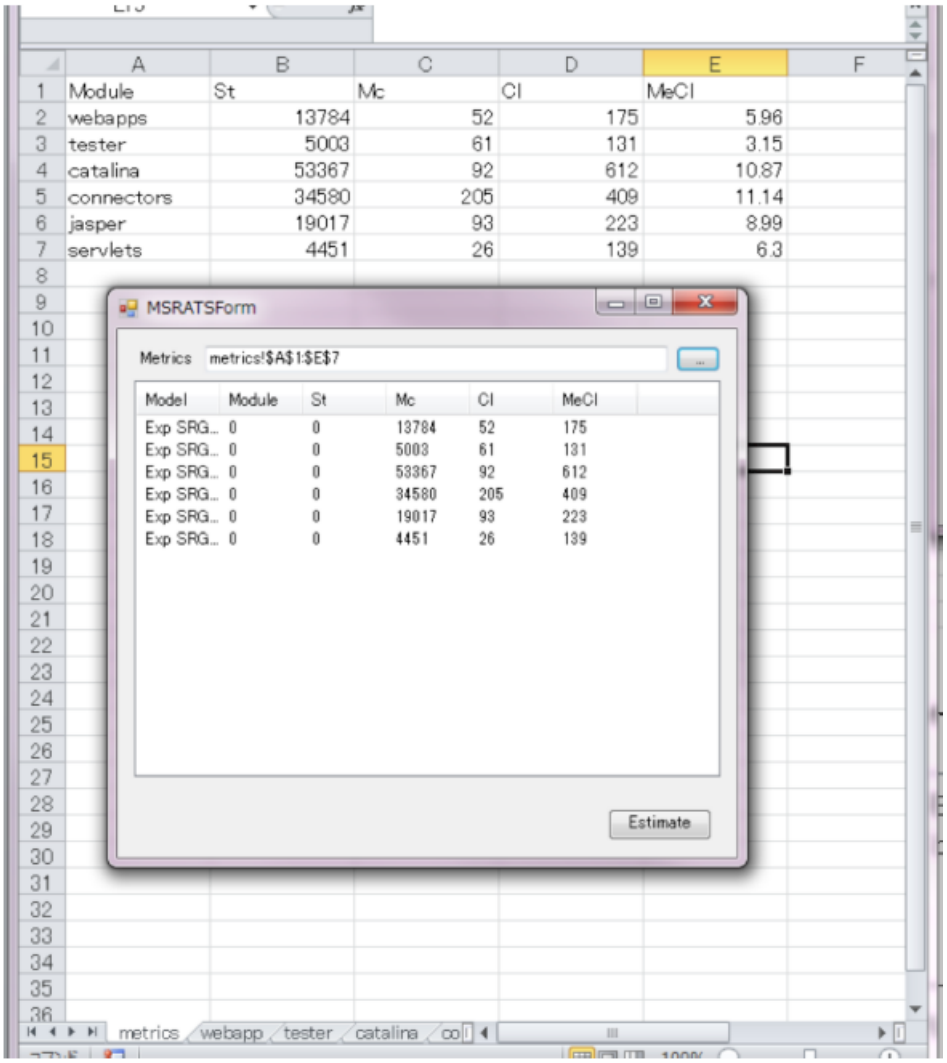


図3-4-3:MSRATSフォーム.

### (2)Rツールの概要

RsratはR言語を用いた統計処理ソフトウェアRのパッケージとして開発された．MSRATSと比較すると修得が難しいが，ユーザがR言語を用いて拡張することができるため，実験的なデータ解析や大量のデータ解析を行う研究者に向いている．Rsratは次の特徴を持つ．

- Rのデータフレーム形式を利用したデータ入力
- 時刻データ・個数データの両方に対応するデータ入力形式
- パラメータ推定および信頼性尺度計算の自動化
- R言語による拡張が可能

算出されるソフトウェア信頼性評価尺度および扱えるモデルはMSRATSと同じである．また，カーネル法を適用に関してもカーネル値を要因として入力し，正則化最尤法を適用する点でMSRATSと同じである．Rsratで扱えるデータ形式に関してもMSRATSと同様であり，時刻データ，個数データおよび一般化個数データである．
主に利用する関数は次のものがある．

srat.nhpp

NHPPモデルに対するRの関数．データをRのデータフレームで与え，バグ個数と時間間隔に対応するコラムをformulaで指定する．デフォルトでは11種類のNHPPモデル全てに対して推定を行い，それぞれのモデルで評価した信頼性尺度や推定されたパラメータをリストとして返す．

srat.logit

ロジスティック回帰によるソフトウェア信頼性モデルを扱うRの関数．データをRのデータフレームで与え，バグ個数とその他の推定に利用する動的メトリクスに対応するコラムをformulaで指定する．信頼性尺度や推定されたパラメータをリストとして返す．また，結果に対してstep関数を適用することでステップワイズによる変数増減法を適用することができる．さらにcontrolにlist形式でlambdaに0以外の値を指定することで，それを正則化パラメータとしたL2正則化最尤法を実行する．

srat.poireg

ポアソン回帰によるソフトウェア信頼性モデルを扱うRの関数．各モジュールのメトリクスデータをRのデータフレームで与える．各モジュールで利用するモデルはsrm.list引数で指定する．これはsrat.nhppまたはsrat.logitの戻り値をlist形式で与える．モジュールに対するモデルとメトリクスの対応は，メトリクスデータの行の名前とsrm.listにおけるリストの名前を照合してマッチングする．また，推定に利用する静的メトリクスに対応するコラムをformulaで指定する．静的メトリクスに対する回帰係数やその他プロジェクト全体に対する評価結果をリストとして返す．また，個別のモジュールに対する推定・評価結果

は戻り値となるリストの中のsrmに各モジュール名のリストとして返す．結果に対してstep関数を適用することで，静的メトリクスに対する変数増減法を実行できる．さらにcontrolにlist形式でlambdaに0以外の値を指定することで，それを正則化パラメータとした静的メトリクスに対するL2正則化最尤法を実行する．

### 3.4.2 ツール設計

開発したツールは，図3-4-4で示すようにそれぞれのツールから推定を行うための計算ライブラリを呼び出している（動的リンクによる呼び出し）．

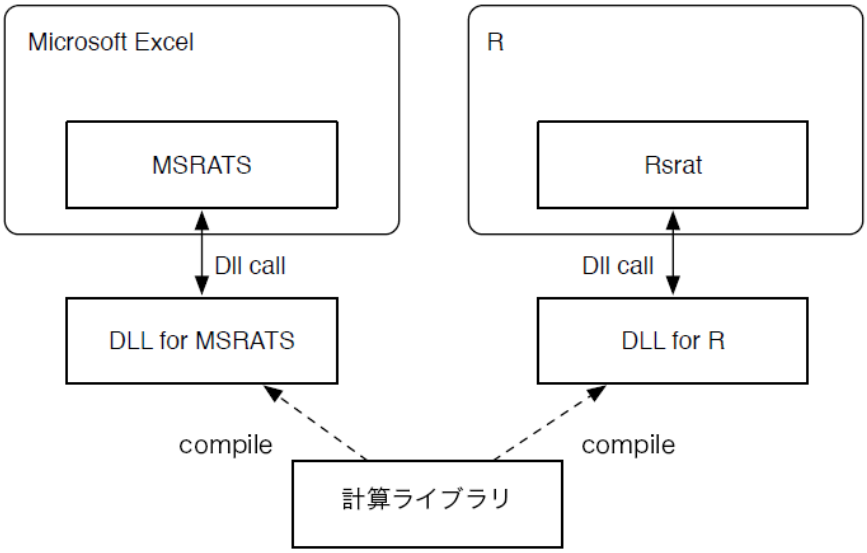


図 3-4-4: ツールの依存関係.

#### (1) 計算ライブラリ

計算ライブラリはC++による開発を行い，50ファイル，約6,500LOCの規模となる．ライブラリは六つのモジュールから構成され，それぞれ以下のように要約される．

common

汎用的な数値計算プログラムをまとめたモジュールであり，二分法，数値積分，ガンマ関数，ガンマ分布，正規分布，$q$分位点の計算を行う．

scivec

ベクトル・行列のデータ形式の定義と基本的な演算を行う．

srm

NHPPモデルによる信頼性尺度の計算およびデータからのパラメータ推定を行う．

glm

一般化線形モデルにおける定義と，最尤法および正則化最尤法による回帰係数の推定を行う．いずれの推定でも重み付き最小二乗法を利用する．

logit

動的メトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデルの定義，パラメータ推定（最尤法，正則化最尤法），および信頼性尺度の計算を行う．

poireg

静的メトリクスを扱うポアソン回帰によるソフトウェア信頼性モデルの定義およびパラメータ推定（最尤法，正則化最尤法）を行う．

各モジュールの関連を図3-24に示す．ベクトル・行列演算や数値計算を別モジュールとすることでソースコードのメンテナンス性を高めている．

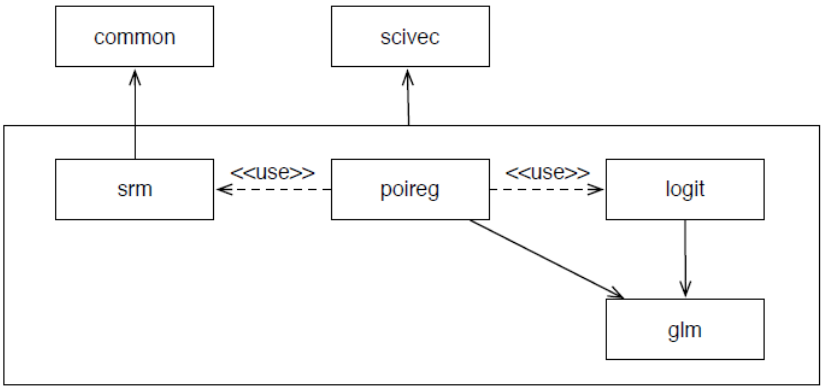


図3-24:計算ライブラリにおけるモジュール間の依存関係.

(2)MSRATS

MSRATSはC#による開発であり34クラスで約5,500LOCの規模となる．基本的な設計はMVC(Model-View-Controller)による設計であり，以下のモジュールからなる．

Common

DLL用のデータ構造の定義とExcel固有の操作とのインタフェースを提供する．

Forms

SRATS，LSRATS，MSRATSフォームなどのユーザインタフェースおよび，各フォームとモデルを制御するコントロールクラスを提供する．

Data

スプレッドシートからバグデータを読み込むなどのデータ入出力を行う.

Models

NHPP モデル, 動的メトリクスを扱うロジスティック回帰によるソフトウェア信頼性モデル, 静的メトリクスを扱うポアソン回帰によるソフトウェア信頼性モデル, それぞれの定義, パラメータ推定（最尤法, 正則化最尤法）および信頼性尺度の計算を行う. 計算ライブラリを動的に call する.

図3-4-6にクラス図を示す. Excel固有の命令や, DLLとのインタフェースなどをまとめることでソースの移植性を高めている.

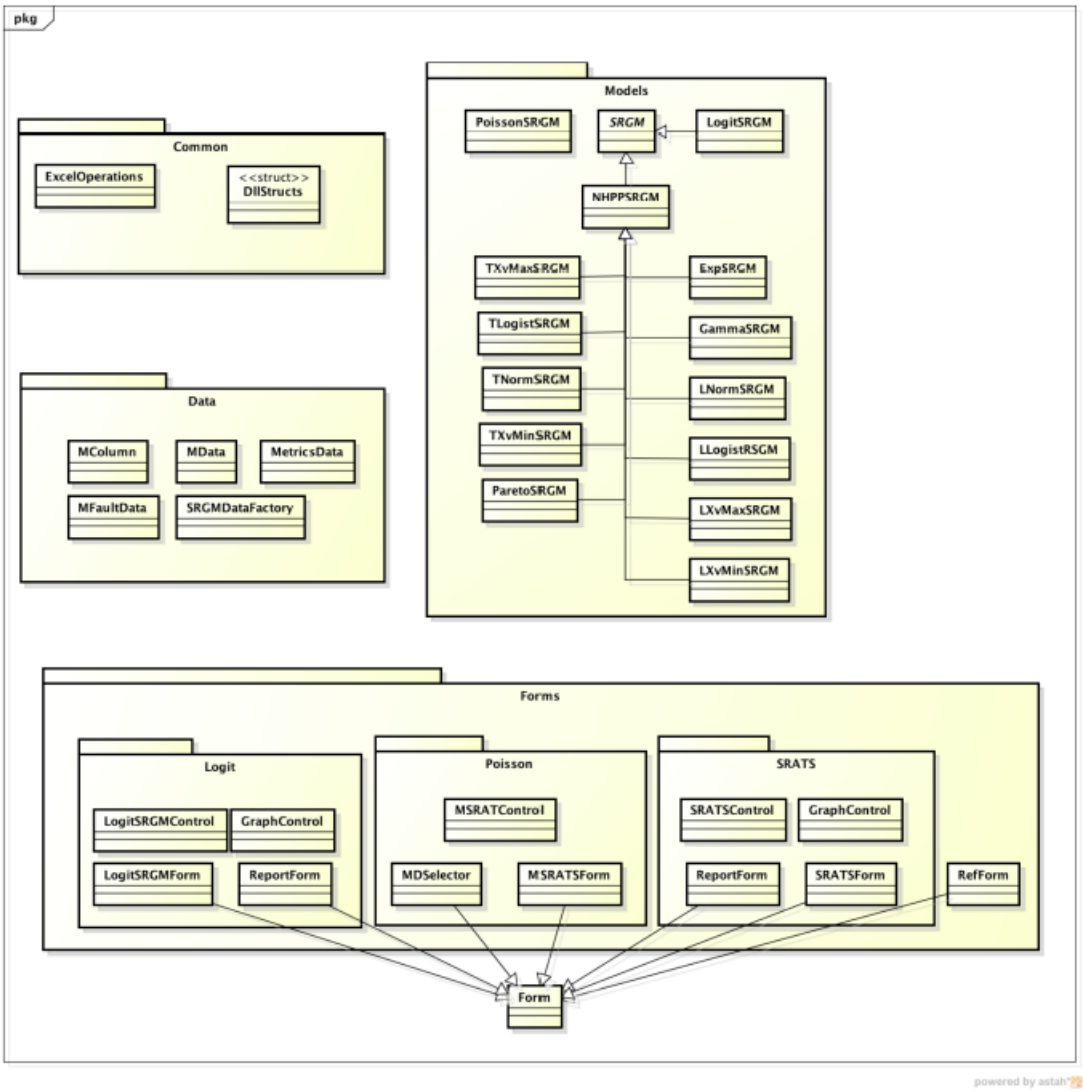


図 3-4-6:MSRATS におけるクラス図.

(3)Rsrat

RsratはRで用いる関数を定義するためのR言語による部分と, C++で記述された計算ライ

ブラリとのインタフェース部分からなる．全体で約1,500LOCの規模となる．R言語ファイルは以下のものからなる．

Rsrat.common.R
　全体に共通するデータ構造や関数の定義を行う．

Rsrat.nhpp.R
　NHPPモデルに対する関数srat.nhpp関係の各種関数を提供する．

Rsrat.logit.R
　　ロジスティック回帰によるソフトウェア信頼性モデルに対する関数srat.logit関係の各種関数を提供する．

Rsrat.poireg.R
　　ポアソン回帰によるソフトウェア信頼性モデルに対する関数srat.poireg関係の各種関数を提供する．

Rsrat.bs.R
　ソフトウェア信頼性モデルのブートストラップに対する機能を提供する．

### 3.4.3 用例

(1) MSRATS の使用例
　ここでは MSRATS の一連の操作方法を説明する．基本的な操作の流れは

- MSRATS の起動
- コンポーネント情報を入力
- 各コンポーネントのフォールトデータの選択
- パラメータの推定＆モデル選択
- レポート出力

となる．

起動

MSRATSはExcel上のアドインとして動作するため，データを選択した状態からメニューバーあるいはアドインタブから「M-SRATS」を選択するとメインフォームが表示される（図3-4-7）．

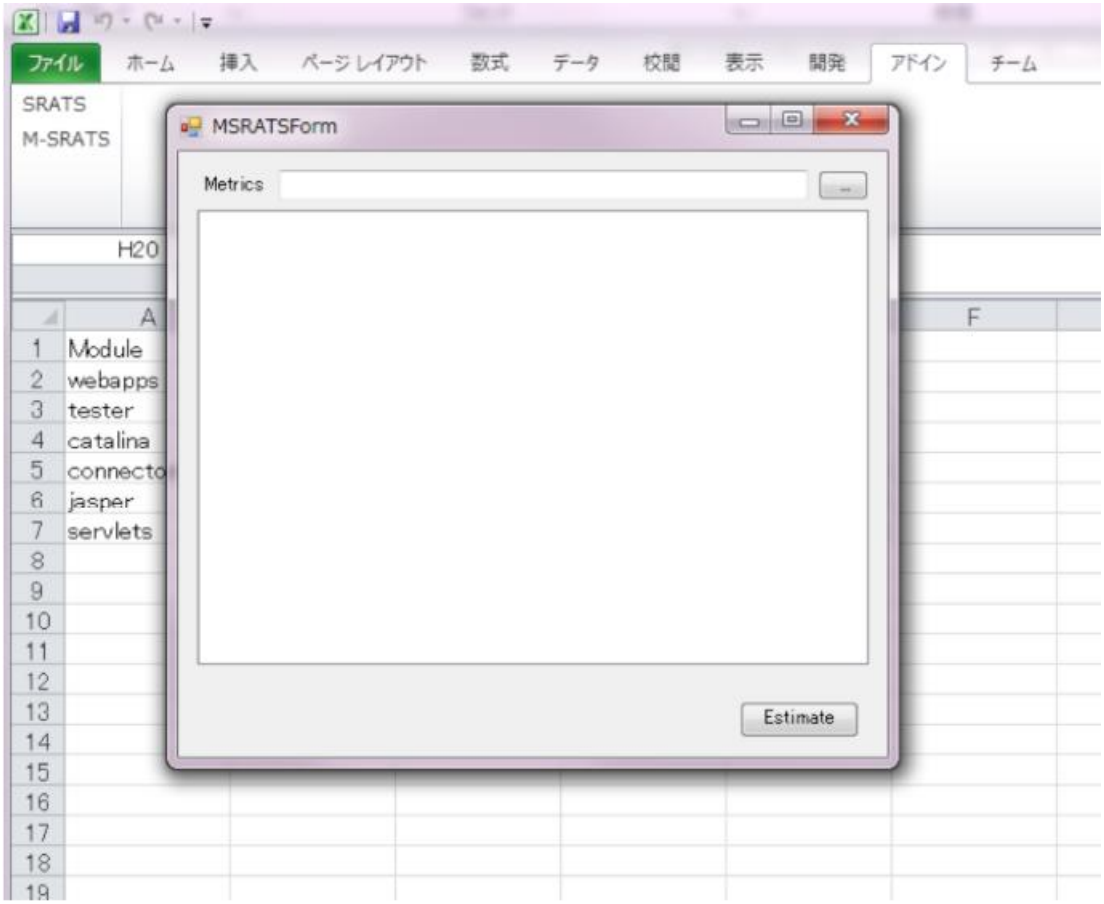


図 3-4-7:MSRATS 起動フォーム.

コンポーネント情報入力

「Metrics」のボタンを押し，ソフトウェア中の各モジュールのメトリクス情報を選択して入力する（図3-4-8）．

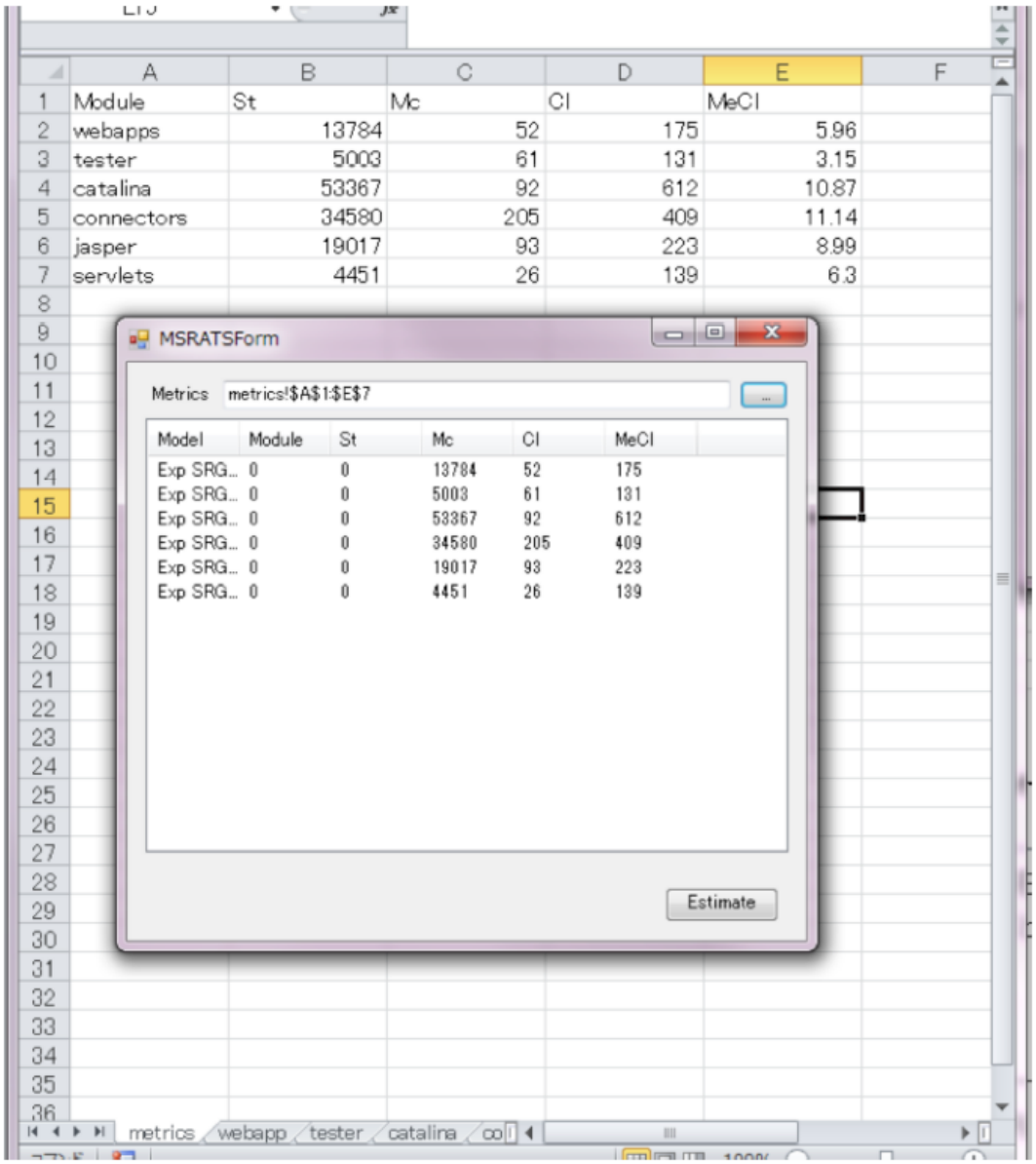


図 3-4-8:メトリクス選択後.

データ選択

一つのモジュールをダブルクリックすると対応するモジュールのフォールトデータ入力（SRATS or LSRATSのフォーム）と推定画面が表示される（図3-4-9）.

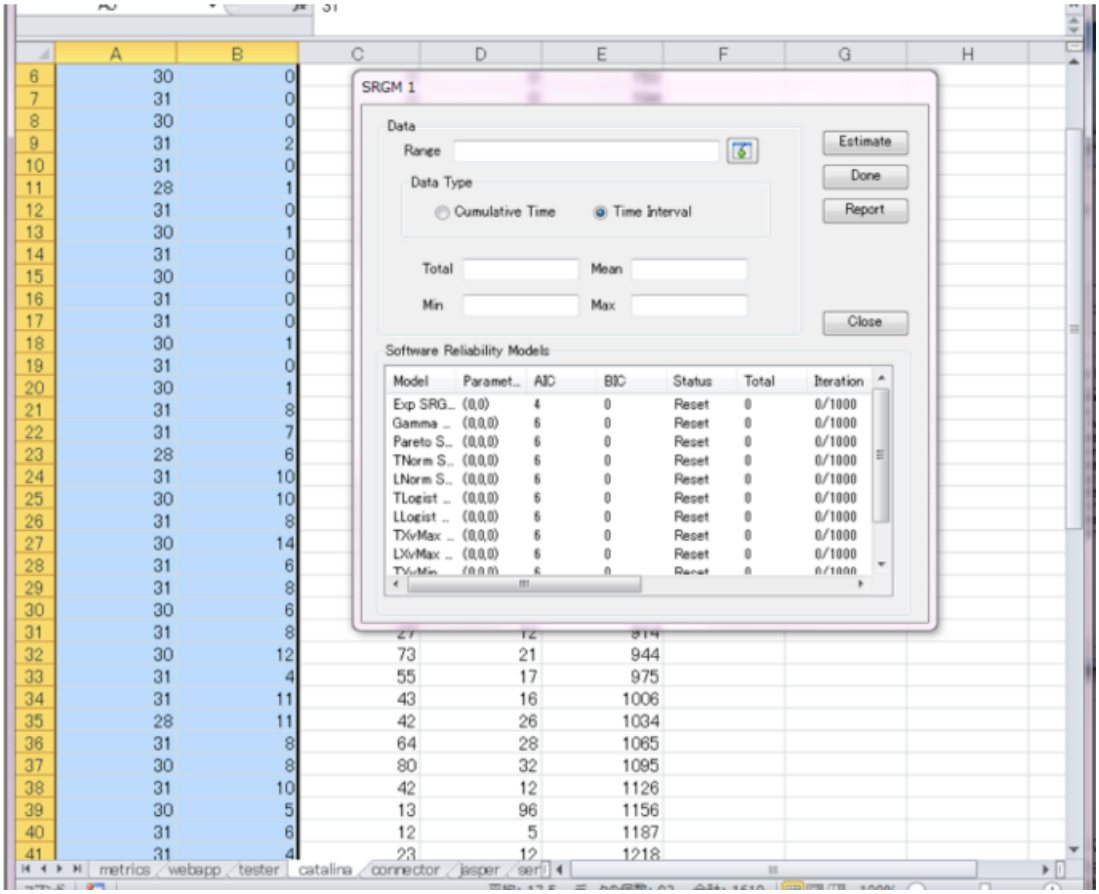


**図 3-4-9:モジュールの SRATS フォーム（推定前）.**

データ形式は先に紹介した三つデータ形式を指定する．MSRATSのフォーム上で，第１列目が時間間隔の場合はTime Interval，第１列目が累積時間の場合はCumulative Timeを選ぶようにする．MSRATS内では，時刻・個数・一般化個数データの判別は選択したデータの列数で判断する．つまり，1列だけ選んだ場合は時刻データ，2列選んだ場合は個数データ，3列選んだ場合は一般化個数データとして処理をする．

単一モジュールに対するパラメータ推定

単一モジュールに対してデータ選択をした後にEstimateボタンを押すことで選択しているモジュールに関するデータだけからモデルの推定を行う．結果は下のSoftware Reliability Modelsにリスト表示される（図3-4-10）．

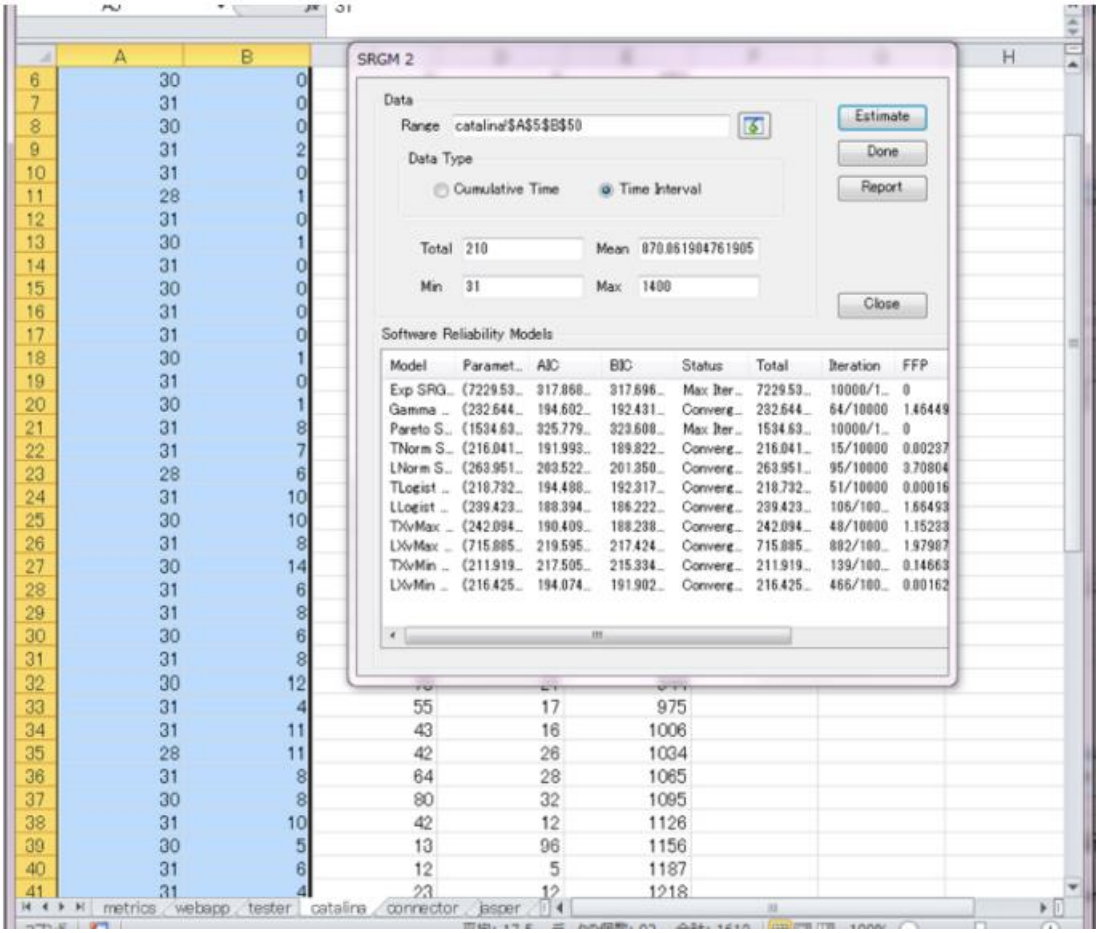


**図 3-4-10:モジュールの SRATS フォーム（推定後）.**

このとき，推定に用いる初期値や収束判定は自動で行う．モデルリストから特定のモデルを選択し，Reportボタンを押すことで信頼性評価尺度をシートへ出力し，グラフの描画を行うことができる．また，一つのモデルを選択しDoneボタンを押すと該当するモジュールに対するモデルを選び，MSRATSフォームへ戻る．このとき，選ばれたモデルがポアソン回帰によるソフトウェア信頼性評価で使われる．同様な作業はLSRATSフォームでも行え，この場合はロジスティック回帰によるソフトウェア信頼性モデルが選ばれたことになる．

一般化線形ソフトウェア信頼性モデルのパラメータ推定

全てのモジュールに対するモデルを選択した後に，ポアソン回帰によるソフトウェア信頼性モデルで静的メトリクスの考慮を行う．推定は MSRATS フォームの Estimate ボタンを押すことで各モジュールの回帰係数などの推定が行える．推定結果（信頼性評価）は各モジュール単位で行うため，MSARTS フォームでモジュールを選択（ダブルクリック）して SRATS あるいは LSRATS フォームを呼び出し，そこから信頼性評価尺度の出力やグラフ描画を行う（図 3-4-11）．

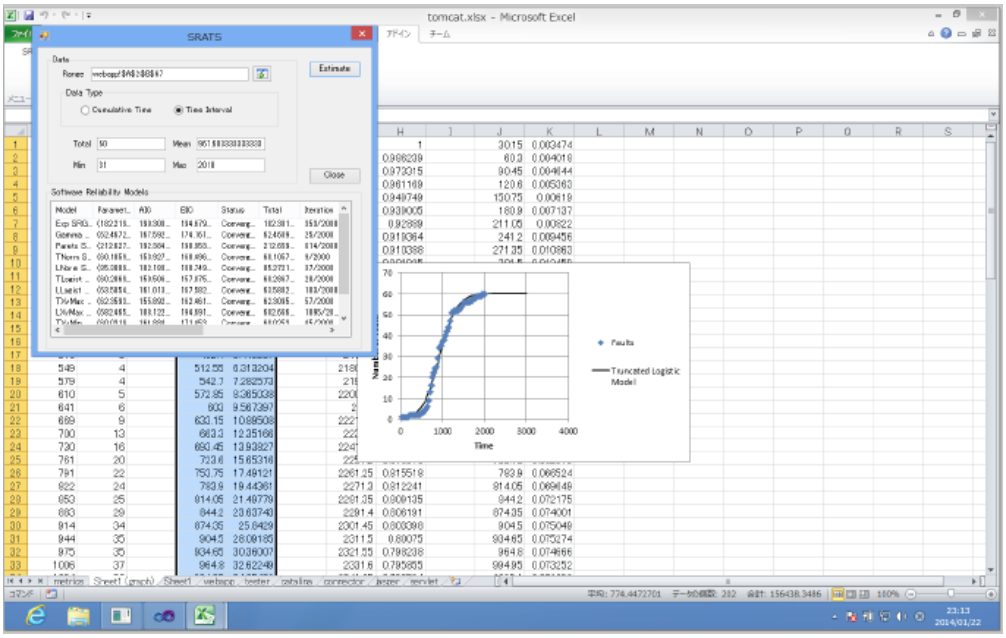


図 3-4-11:MSRATS からの出力.

### (2)Rsrat の使用例

ここではRsratを利用した一般化線形ソフトウェア信頼性モデルの推定と評価を行う．Rsratでは次の手順で行うこととなる．

(i)各モジュールに対してロジスティック回帰によるソフトウェア信頼性モデルの推定を行う．

(ii)(i)の結果を用いてポアソン回帰によるソフトウェア信頼性モデルの推定と評価を行う．

### (i)ロジスティック回帰によるソフトウェア信頼性モデルの推定

下のような動的なメトリクスとバグ個数のデータをcsvファイルで用意する．ただし，1行目はラベルとする．

| days | faults | coverage | work |
|---|---|---|---|
| 1 | 2 | 0.1 | 2.0 |
| 2 | 10 | 0.2 | 5.0 |
| 3 | 1 | 0.7 | 3.0 |
| 4 | 0 | 0.9 | 1.0 |

Rでcsvデータを読み込む．

```
dat.module1<-read.csv(file="test.csv")
```

次にsrat.logitを使って推定を行う．例えば以下のような入力をする．

```
result.module1<-srat.logit(formula=faults~coverage+work,data=dat.module1)
```

formulaはR標準の回帰関数であるlmなどで利用されるものと同じ書き方になっている．変数選択（要因選択）はRの機能で提供されているstepを使うことができる．つまり次のコマンドを入力する．

```
result.module1<-step(result.module1)
```

これを全てのモジュールについて実行する．

(ii)ポアソン回帰によるソフトウェア信頼性モデルの推定

ポアソン回帰の関数srat.poiregへ(i)の結果を渡すため，モジュール名と対応したリストにセットする．

```
result.list<-list(module1=result.module1,…,module10=result.module10)
```

次に，静的メトリクスを次のようなcsvファイルで用意し，R上に読み込む．

| module | LOC | Complexity | Files | Classes | Methods | Comments |
|---|---|---|---|---|---|---|
| module1 | 10304 | 1.1 | 229 | 12 | 19 | 113 |
| module2 | 12994 | 0.9 | 102 | 23 | 59 | 1034 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| module10 | 8484 | 1.5 | 45 | 11 | 53 | 19 |

このとき，行のラベル名がモジュール名と一致するように注意する．

```
metrics<-read.csv(file="metrics.csv",row.names=1)
```

以上の準備のもとで以下のコマンドを実行する．

```
result<-srat.poireg(formula=~.,metrics=metrics,srm.list=result.list)
```

上記は，ピリオドを使うことで，LOC,Complexity,…,Commmentsまでの全てのメトリクスを使うことを示している．ロジスティック回帰によるソフトウェア信頼性モデルと同様に

```
result<-step(result)
```

とすることで，Rの機能を用いて変数増減法を行うことができる．結果はリスト形式になっており，例えば各モジュールの総バグ数は

```
result$total.faults
```

とすることで得られる．また，各モジュール個別に適用したモデルの結果がresult$srmsにリスト形式で保存されており，残存バグ数やFFPなどを得ることができる．

### 3.4.4 実用化へ向けた課題と問題点

(1)MSRATS の課題

MSRATSについては，今後より使いやすいインタフェースへの改善が課題としてあげられる．MSRATSに先行するSRATSが普及した大きな理由の一つはシンプルな機能，つまり，11種類のモデルによる推定を，「エクセルシートでデータを選択」，「ボタン一つで推定」と

いう「操作数の少なさ」があると考えている．一方で，MSRATSは親しみがあるExcelのインタフェースを利用しているものの，メトリクスデータの選択，モジュール毎のデータ選択，モジュール毎のモデルパラメータ推定，モジュール毎の最良モデルの選択，全てのモジュールの推定という操作を行う必要があり，数理を専門としない実務家にとってかなり敷居が高い．開発ツールをあらゆる現場における実用に供するためには，現場における典型的なデータの調査を行い，「データを入力」，「推定」という二つのステップだけを実行することが理想であるため，そのための理論の改善か実装の改善が必要となる．

(2)Rsrat の課題

Rsratは研究者をターゲットとしているため，現時点での完成度は非常に高いものと考えている．一方で，試験的な機能をユーザ（研究者）が容易に拡張できるように，R言語ファイルで定義した関数や計算ライブラリとのインタフェースをより洗練したものにする必要があると考えている．

# 4 考察

## 4.1 研究により判明した効果や問題点等

### 4.1.1　研究・開発単位での効果と問題点・課題

本研究では,【モデル構築】,【推定アルゴリズム開発】,【ツール開発】,【実証分析】という一連の研究・開発作業を通じて，従来までのソフトウェア信頼性評価技術における予測精度に関する問題点を克服し，さらに開発管理情報の有効的な利用を前提とした次世代ソフトウェア信頼性評価技術を開発した．信頼性評価尺度の推定精度の向上ならびに推定労力を軽減するためのツール化を実現し，その効果を実際のプロジェクトデータを用いて検証した．以下では，本研究で到達することのできた新しい技術とそれらの効用について概説し，研究の過程において判明した問題点について述べる．

**一般化線形モデル**

【一般化線形モデルの効果】

前章で述べた実証分析の結果から，本研究で提案した一般化線形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 や ABIC の情報量基準はモデルの自由度を考慮したモデル選択基準であるため，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．ポアソン回帰に静的メトリクスデータを活用した場合と，ロジスティック回帰に動的メトリクスデータを用いた場合を比較すれば，明らかに後者の方が適合性に関する貢献度が高いことが分かった．本研究で提案したポアソン回帰とロジスティック回帰を組み合わせた統合モデルが最適モデルであることは既に述べたが,この主な理由は動的メトリクスデータの貢献によるものである．無論，変数選択を行って寄与率の高いメトリクスを抽出しているため，全ての組み合わせを列挙してはいないが，静的メトリクスと動的メトリクスの有用性については議論を待つまでもない．これより，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フォールト検出過程の挙動に直接的に効果を与えることはない．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静的メトリクス計測技術よりも，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．ソフトウェアに内在する初期期待フォールト数や残存フォールト数，FFP の推定結果についても，最も適合性の高いモデルに基づいた信頼性評価尺度が最も信憑性のある定量的尺度であるとの前提に立てば，モデルや観測メトリクスの種類に応じて推定結果がかなり異なることが理解できる．

一般化線形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フォールト情報並びにメトリクス情報を一度に全て扱うことで，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，各モジュールの単体テストにおける

信頼性評価では，各々のフォールトデータに信頼性評価モデルを独立に適用するという方法がとられてきた．このような方法では，モジュール相互の関連性を全く無視しており，しかも回帰構造を導入することで静的メトリクスの情報を矛盾なく信頼性評価に組み込むことが困難である．本研究で提案した一般化線形モデルは，オープンソースに代表されるように，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最も現実的かつ有効な評価スキームを提供していると言える．世界に先駆けて最も一般的かつ汎用的なソフトウェア信頼性評価技術とそのツールを開発した意義は高いものと考えられる．

【一般化線形モデルに関する問題点と課題】

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，提案ツールはほとんどそのまま利用することが可能である．計測可能でかつ入手可能なメトリクス情報があれば，それを全て利用することで正確な信頼性評価が行えることは事実である．反面，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り，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無ければ，ここ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，実証ソフトウェア工学において長年積み重ねられた実証的成果や企業で培われた独自のノウハウを生かすことが肝要である．考えられ得る全ての種類のメトリクス情報が獲得できたとしても，それを適切に選択し最良モデルを構築するためにはさらに労力が必要である．モジュールごとに観測された複数の静的メトリクスと動的メトリクスの組み合わせを全て考慮し最適なモデル選択を行うためには，膨大な計算量が必要とされる．例えば 3 つのモジュールを有し，各々に対して 2 種類の静的メトリクスと 3 種類の動的メトリクスが観測されているとすれば 21952 通りの組み合わせが存在する．モジュール数が増え，かつ観測メトリクスの数が増えるにつれてその組み合わせ数は天文学的な数になることは自明である．本研究で適用した変数選択法は統計ソフト R に内装されている機能をそのまま用いているため，厳密な意味での最適なメトリクスの組み合わせを求めているわけではない．換言すれば，あるソフトウェアに特有なメトリクスの最適な組み合わせが存在する可能性も無いわけではない．このようなビッグデータの取り扱いは重要な課題であり，本研究で開発した Excel ツールもその部分がマニュアル設定となっている．変数選択機能の強化とそれに関連したデータマイニング技術の開発が今後の重要な課題となる．

また，信頼性評価ツールを業務の一環として恒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の連動が必要不可欠である．世界中に広く存在するこのような管理ツールのあらゆるデータ構造に対応し，データベースからの出力に前処理を行うことなく解析できるようなツールの開発は大変困難であることが想定される．しかしながら，ソフトウェアの信頼性評価を不可欠な品質管理項目として位置づけるならば，このようなデータベースとの連携は避けては通れない問題である．これを解決する方法として，データベースに格納されている情報の種類の分類や必要なデータの長さを定義し,標準化されたデータ形式の下で信頼性評価を行う枠組みについて検討する必要がある．

**カーネル法**

【カーネル法の効果】

ソフトウェアメトリクスの形で集約された統計情報は有効である反面，その計測作業自体にコストがかかり，産業界において常に利用されているとは限らない．また現実問題として，メトリクス計測を通じて工程管理をしている現場は極端に少ないというのが現状であり，一般化線形モデルの適用範囲は限定的であることも事実である．ソフトウェアのテストによって信頼性は向上するので，テスト技術者はテスト成果に基づいた信頼性評価にのみ信頼をおく傾向にある．一方で，開発者は成果物であるソースコードの中身をレビューすることなく信頼性評価を行うことは不可能であると考える向きがあり，仕様書や詳細設計書の誤りを除くことに加えて，ソースコードの分析結果から品質を評価する傾向にある．このような異なるパラダイム上で定量的な信頼性評価を行うためには，テストケースの入出力やソースコードから，直接，定量的信頼性評価を実施する方法が長い間切望されてきた．本研究で行ったように，カーネル関数にテストケース入力情報やソースコード上の文字列情報を表現し，フォールトの検出確率に関連づけたモデル化をした例はこれまでに報告されていない．その意味で，ここで新しく提案されたカーネル回帰に基づいた信頼性評価技術は，定量化することが困難であった情報を定量化するという意味で画期的な方法と言える．テスト入力情報を活用した信頼性モデルは大量のテストケースを投入するシステムテスト工程において威力を発揮する信頼性評価技術であるが，単体テストにおいても適用可能である．ソースコード情報を扱うモデルは，結合テストを実施する前の単体テストにおいて用いられるべき方法として位置づけられる．

本研究では，テスト入力の距離を編集距離として定義し，ロジスティック回帰モデルに応用することを提案した．また，ソースコード情報を活用するモデルではコードクローンに着目したファイル間の類似性を計測し，これを信頼性評価に適用するというアイデアを提案した．いずれのモデルも，通常の確率・統計モデルにおいて任意パラメータを推定し，最適なモデルを選択した場合と比較しても遜色のない成果を与えることができるものの，テスト情報やソースコード情報を活用することで特段適合性の高いモデルが構築できるわけではなかった．しかしながら，テスト入力情報の違いや距離の定義に関する違い，ソースコードの静的解析において同定可能な類似性や構造を特徴づけるパラメータが異なれば，信頼性評価に関する適合性評価の結果も全く異なるものになる．また，カーネル関数の種類やパラメータ推定法の種類が数多く存在するため，これらの性能を包括的に調査しなければモデルの優位性を結論づけることは難しい．本研究で得られた知見としては，これまでに定量化することが難しかった情報を信頼性評価に適用するための基本的枠組みが整備され，その実用化に向けた可能性について言及できたことである．このようなアプローチは開発の各段階に参加する技術者の信頼性評価に関するコンセンサスをとる上で重要な技術であり，今後急速に発展する新しい研究分野であると言える．

【カーネル法に関する問題点と課題】

テスト情報やソースコード情報を活用するというアイデアは画期的であるという反面，類似度（あるいは距離）の定義によりその性能が大きく変化するという問題がある．信頼性評価に用いられるべき各種属性の距離の性質についてまだ詳細がわかっておらず，ソフ

トウェア工学における新たな問題として今後研究を行う必要がある．また，カーネル法の適用においてメトリクスの選択をする必要がないという大きな利点はあるものの，計算コストが一般化線形ソフトウェア信頼性モデルの数倍から数十倍かかるため，実用化のためにはより効率的な計算アルゴリズムおよび実装を行う必要があることは既に指摘した通りである．現在入手可能で，かつ最高速で演算を実行するパッケージを組み込んだツールを開発していているが，膨大なテストケースや大規模なソースコードを取り扱うためには，並列計算や高パフォーマンス計算の技術を援用した計算アルゴリズムをさらに開発する必要がある．

**ツール開発**

【ツール開発の効果】

本研究で開発したツールの効果として，まず Excel インタフェースに基づいた MSRATS は表計算ソフトウェア上に必要な情報を整理するだけで，手軽に信頼性評価が行えるという利点がある．これは広島大学で従前までに開発していた SRATS の拡張ツールとして捉えることが可能であり，フリーウェアとして提供するため，今後は産業界における信頼性評価のための標準ツールとして認められることを期待している．一方，統計解析ソフト R 上で駆動する Rsrat は，R 言語でコマンド入力する必要はあるものの，拡張性に優れており，しかも R の持つ様々な統計ツールと連動することが可能であり，世界中の研究者によって活用されることが期待される．これまでに世界中で公開されてきた信頼性評価のためのフリーツールは数が少なく，有料のものを合わせてもその性能にかなりの問題があった．ツールとして必要な信頼性評価尺度の精度保証，モデル選択の自動化，評価結果を出力するための実行可能性と実時間制約など，これまでのツールには欠けていた問題点を克服することができ，しかも高水準確率モデルに基づいた性能を有している点に注目すべきである．今後は，産業界での応用と研究用ツールとしての二つの側面からの利用が考えられるため，ツールのマニュアル整備や機能追加を計画している．

【ツール開発に関する問題点と課題】

MSRATS については，今後より使いやすいインタフェースへの改善が課題としてあげられる．MSRATS に先行する SRATS が普及した大きな理由の一つはシンプルな機能，つまり，11 種類のモデルによる推定を，「エクセルシートでデータを選択」，「ボタン一つで推定」できるという「操作数の少なさ」にあると考えている．一方で，MSRATS は親しみがある Excel のインタフェースを利用しているものの，メトリクスデータの選択，モジュール毎のデータ選択，モジュール毎のモデルパラメータ推定，モジュール毎の最良モデルの選択，全てのモジュールの推定という操作を行う必要があり，実務家が自由に使いこなすためにはかなり敷居が高い．実用に供するためには，現場における典型的なデータの調査を行い，「データを入力」，「推定」という二つのステップが理想であるため，そのための理論の改善や実装の改善が必要となる．Rsrat は研究者をターゲットとしているため，現時点での完成度は非常に高いものと考えている．一方で，試験的な機能をユーザ（研究者）が容易に

拡張できるように，R 言語ファイルで定義した関数や計算ライブラリとのインタフェースをより洗練したものにする必要があると考えている．

### 4.1.2 今後の研究への展開

本研究の成果と問題点を踏まえて，今後のソフトウェア信頼性評価技術の研究動向について述べる．先にも述べたように，ソフトウェアメトリクス情報に基づいた一般化線形モデルの開発とそのツール化は，現時点における信頼性評価技術の最高到達点にあると言える．メトリクスデータの抽出法やリポジトリデータベースとの連動などの課題は残されているものの，基本的な枠組みは全て開発されたものと考えられる．今後の理論的な研究動向として，ソフトウェアフォールトデータが持つ非定常性や非線形性を考慮して，さらに複雑なモデル化技術を構築することが考えられる.過去 40 年の当該研究分野の歴史において NHPP モデル以上に適合性が高く，かつ汎用的なモデルは存在しない．一方で，ポアソン過程が持つ平均値と分散値の値が常に等しいという dispersion の問題は理論的に重要な課題である．これまでに提案されたモデルの中で，非定常ガンマ過程に基づくソフトウェア信頼性モデルがその問題点を解決する唯一つの方法であり，一般化線形モデルと非定常ガンマ過程を融合することによる新しい評価モデルの開発が期待できる．ただし，もはや累積フォールト検出数に関する確率関数が解析的に陽な形で求められないので，漸近理論に基づいた何らかの近似手続きが必要となるものと考えられる．

カーネル法に基づく信頼性評価技術は，メトリクスの形で抽出することのできない情報をカーネル関数上に表現することで信頼性評価を行う点が斬新的であった．今後，カーネル関数の種類，テスト入力情報やソースコード情報の適切な選定を通じて，より効果的かつ理論的な正当性の高い信頼性評価技術が開発されるものと考えられる．例えば，テストケース間の類似性やファイルの類似性を測る尺度は他にも数多く存在するため，推定アルゴリズムの検討も加えて，今後も継続して研究を行うべき課題である．このモデル化技法の基本的な枠組みも NHPP に基づいているため,先に述べた非定常ガンマ過程の適用について検討することが考えられる．また，テストツールやソースコード分析の出力結果をそのまま入力とし，ソフトウェア信頼性評価尺度を出力結果として返すような信頼性評価技術の開発は，テスト現場で日々格闘するテスト技術者にとっては有益なツールとなり得る．

本研究を通じて，パラメータの推定法として最尤推定法を用いてきた．反面，最尤推定法の効用と限界については良く知られており，より理論統計学的に洗練した推定アルゴリズムであるベイズ推定の活用が考えられる．ベイズ推定はパラメータの事前情報を与える必要がある反面，ベイズリスクを最小にする理論的な妥当性と多くの漸近的に有益な性質を有することから，先端的学習理論における最も重要な要素技術として定着しつつある．マルコフ連鎖モンテカルロ法や変分ベイズ法など，計算コストを削減するためのアルゴリズムも考案されているが，本研究で取り扱ったような回帰構造を有する信頼性評価問題に適用した事例はまだない．ベイズ法はソフトウェア信頼性評価技術に事前情報を組み込むことが可能であり，開発のノウハウや知識を事前情報の形でパラメータ化することに成功すれば，開発経験に関する情報を有効に活用したソフトウェア信頼性評価技術を提案することができるものと期待している．さらに，最尤推定法とは異なる頻度論的アプローチとして，ソフトウェア信頼性モデルの具体的なパラメトリックな形状を仮定することなく信

頼性評価を行うノンパラメトリック法を検討することが考えられる．一般に，回帰ベースの方法はモデルの具体的な形状だけに結果が依存しないという意味でセミパラメトリックモデルに分類される．本研究で開発した信頼性評価技術の論点は開発・管理情報の有効な活用にあったが，未踏ソフトウェアの開発や特殊な開発形態を採用している現場では，フォールト数データくらいしか計測できないような状況もあり得る．そのような場合，ノンパラメトリック最尤推定法やカーネル強度関数と呼ばれる強度を持つ NHPP モデルの活用が有効であるように思われる．

## 4.2 今後の課題

### 4.2.1 研究成果の産業界への寄与

本委託研究の成果を実際の企業活動に適用する場合の効用と問題点について述べる．ここでは企業における開発現場で利用することを念頭に設計された MSRATS を例に,研究成果の具体的な活用法について言及する．

【従来の信頼性評価ツールと比べてどのように優れた信頼性評価が可能になるか？】

ソフトウェアの定量的信頼性を評価する際に既存の有償ツールや他の無償ツールと比較して，各テスト工程で観測されたテスト実績や開発データに基づいて製品の信頼性を高い精度で推定することが可能である．現在までに世界中で入手可能な信頼性評価ツールは，有料ツールと無償ツールの種類に拘わらずフォールト検出数データなど極めて限定された統計情報しか取り扱えない．一方で，MSRATS を用いることで，ソフトウェアメトリクス，テスト入出力情報，ソースコード情報からソフトウェアの信頼性を特徴づける情報を抽出し，極めて精度の高い定量的信頼性評価を実現することが可能となった．また，従来のツールでは多くのパラメトリックなモデルから最良モデルを選択し，現在のプロジェクトに最も適合する信頼性評価モデルを決定するという複雑な手続きを踏んでいたが，本研究でベースとなっている回帰構造に基づいた方法はセミパラメトリックモデルであり，最良モデルを選択する労力はほとんど必要とされない．さらに，Excel インタフェースに基づいた表計算ソフトウェア上に必要な情報を整理するだけで，手軽に信頼性評価が行えるという利点は，信頼性評価を低いコストで手軽に行えることに繋がる．

【企業のソフトウェア開発者は、どのようなデータを収集し、何をする必要があるか？】

まず，企業の開発現場のどのような状況において信頼性評価を行うべきかを特定する必要がある．信頼性評価結果をテスト進捗状況の把握に用いる場合には，現在までに行われたテストが十分であったかどうかの管理指標として機能する．一方，ソフトウェアを市場もしくはユーザにリリースすべきかの出荷判定に用いる場合には，製品の最終品質を定量化するための指針となる．メトリクス情報を扱う一般化線形モデルは単体テストとシステムテストの両方に活用することが可能である．テスト入力情報を扱うカーネル法に基づいたモデルではプログラムの正誤判定を規定するテストケース（入力値と期待出力の組）の距離を評価するため，内部構造に依存しないテストを実施するシステムテストで用いられるべき信頼性評価技術である．また，主に単体テストのレビュー工程においてソースコー

ドの静的解析がなされることから，ソースコード解析に基づいた信頼性評価は単体テストの段階で行われるべきであろう．具体的なソースコード解析の一例として，本研究では CCFinder に基づいた類似性尺度を用いたが,対象とするプログラムコードを信頼性評価の観点から特徴づける評価指標であれば他の種類のコード解析手法を適用することも可能である．収集すべきデータの種類に関しては，ソフトウェアの種別や開発現場で測定可能な定量的データの種類に依存するため，どのような情報が最も信頼性評価に寄与するかを前もって知ることは難しい．しかしながら，本研究で行った実証分析では，複雑性メトリクスやボリュームメトリクスなどの静的メトリクスよりもむしろ，テスト進捗に応じて観測される累積テストケース数やバグ修正の際に行ったコミュニケーション回数などの動的メトリクスの方がソフトウェアの信頼性推定に大きく寄与することが分かった．回帰モデルを導入した利点として，信頼性を特徴づける観測データの定性的な性質を事前に分析することなく，直接モデルに代入した上で適切な変数選択を行うことで，信頼性評価に必要とされるデータは取捨選択することが可能である．よって，開発工程で計測可能なデータは多ければ多いほど良く，データ計測と管理に要するコストとのトレードオフを考慮した上で，どのようなデータを計測すべきかを検討すべきであろう．

【ソフトウェア信頼性がどのように可視化できるのか？】

本研究で開発したツールを用いることで，テスト工程で観測される総期待累積発見フォールト数，各テスト時間単位で発見される期待フォールト数（ソフトウェア強度），ソフトウェア MTBF(Mean Time Between Failures)，累積 MTBF，FFP(Fault Free Probability)などの定量的信頼性評価尺度を推定することができる．総期待累積発見フォールト数やソフトウェア強度はあとどれくらいのフォールトが発見されるかを推定するために用いられ，MTBF はフォールトが発見されるまでの時間尺度の平均的な長さを推定するために用いられる．また，FFP はテストを打ち切った際にフォールトが製品内に含まれていない確率であるため，現在テストを行っているソフトウェアにフォールトが含まれているかどうかを判定するために用いられる定量的評価尺度である．確率モデルを適用する最大の利点は，このような確率論的評価尺度を理論的に算出できることにある．このような確率論的評価指標はデータから最も合理的なモデルを通して可視化する他に求める手立てはなく，確率論的信頼性評価の最大の効用と言える．MSRATS では上述の基本的なソフトウェア信頼性評価尺度を自動的に計算する機能が備わっているが，モデルパラメータの推定値も同時に出力しているためその他の信頼性評価尺度を求めることも原理的には可能である．

一方，本研究で取り扱ったモデルは全て回帰構造が導入されており，回帰係数の推定結果を注意深く分析することで，設計上の問題点や開発管理上の問題点の所在を統計的に推論することができる．例えば，設計メトリクスや開発メトリクスを用いて信頼性評価を行う場合，どのメトリックが製品の信頼性に影響を及ぼすかを統計的に判定することが可能である．コード行数やソフトウェア複雑性メトリクスが信頼性に対して負の相関を持つ場合には現行の開発システムの設計を見直す必要があることを示唆しており，戻り工程の位置を決定したり次回の開発に役立てることができる．また，テストケース数やテスト能力指標の欠乏がフォールスアラームとして信頼性評価に影響している場合には，追加テストを実施したりテスト計画を早急に見直すという施策に繋がる．回帰係数の相対的な値だけ

でこのような施策の重要性を判断することは困難であるので，回帰分析に統計的な意味があるかどうかを判定する仮説検定を導入することで，定量的信頼性評価の観点から評価結果の開発現場へのフィードバックを行うことができる．

【企業におけるソフトウェア開発にどのように活用できるか？】

先にも述べたように，信頼性評価はテストの進捗状況把握と出荷判定の際に行われるべき活動である．レビューなどの静的テストやテストケースを投入しながらの動的テストの現場では，一体どれくらいのレビューやテストケース準備を行えば良いかの理論的な指針がほとんどない．そこで，通常は納期とコストを考慮し，テスト計画に合致するようテスト作業を進めることが一般的である．還元すれば，定量的な信頼性評価は実質的には行われておらず，フォールトが出尽くしたかどうかを判定する成長曲線の飽和状態だけを観測することでテスト進捗状況把握や出荷判定を行っているケースがほとんどであろう．MSRATS のような簡易なツールを活用することで，テスト作業者がテストの進捗状況を手軽に把握することができるようになり，また開発管理の立場から製品の出荷判定の根拠となる説明資料として定量的信頼性評価結果を用いることが考えられる．同時に，ユーザや市場に対するソフトウェア製品の品質に関する説明責任を果たす意味で，ソフトウェアの定量的信頼性を確保することは極めて重要な社会的意義を持つものと考える．

### 4.2.2 研究成果の産業界への展開に向けて

実際のソフトウェア開発に信頼性評価の技術を啓蒙するための最も重要な要素として，信頼性評価の費用対効果について言及しなければならない．開発労力やテスト労力が大きければ大きいほど製品の信頼性は向上することは自明であり，その意味で信頼性とコストはトレードオフの関係にある．開発システムを市場やユーザにリリースした後で不具合やシステム障害が発生したとしても，迅速なパッチリリースや保全によって対処することが可能であれば，信頼性評価に莫大なコストをかけることもなく，かつ厳密な定量的評価も必要ないという見方もある．しかしながら，システムの品質向上に対して真摯に取り組む姿勢を放棄する経済活動は産業界における競争的状況からは淘汰されるべきであり，最大限の自助努力を行うことで「安心・安全」を最高の価値基準とする高度情報化社会に貢献すべきではなかろうか。信頼性評価ツールだけでなく高品質な製品を開発するために役立つツールを自前で開発するリスクを少しでも削減し，産業界全体でその効果を共有できるという意味で，本委託研究で研究・開発された技術とツールを世に問う意義は極めて高いと考えている．今後は，信頼性評価結果をより具体的に設計・開発現場にフィードバックするために，実際の開発プロセスに適用することを予定している．

## 参考文献


[1] 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.

[2] J. A. Achcar, D. K. Dey, and M. Niverthi. A Bayesian approach using nonhomoge-neous 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.

[3] 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.

[4] A. L. Goel. Software reliability models: Assumptions, limitations and applicability. *IEEETransactions on Software Engineering*, SE-11:1411-1423, 1985.

[5] 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.

[6] 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.

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

[8] Littlewood. Rationale for a modi_ed duane model. *IEEE Transactions on Reliability*, R-33(2):157-159, 1984.

[9] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability, Measurement, Pre-diction, Application*. McGraw-Hill, New York, 1987.

[10] M. Ohba. Inection 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 model with normal distribution and its parameter estimation. In *Proceedings of the International Conference on Quality, Reliability, Maintenance and Safety Engieering (ICQR2MSE)*, pages 424-429, 2011.

[13] H. Okamura, Y.Watanabe, and T. Dohi. An iterative scheme for maximum likelihood estimation in software reliability modeling. In *Proc. 14th Int'l Symp. on Software Reliab. Eng.*, pages 246-256, 2003

[14] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461-464, 1978.

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

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

[17] 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.

[18] 山田．ソフトウェア信頼性モデル．日科技連，東京, 1994.