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

広島大学大学院工学研究科

教授　土肥正（研究責任者）
准教授 岡村寛之（研究分担者）
助教　肖霄（研究分担者, H25.9月まで）
RA　羅超, 鄭俊俊, 易志鵬,
斉藤靖洋 (研究分担者)

HIROSHIMA UNIVERSITY

1. 背景・課題

- **ソフトウェアの信頼性を作り込む（確保する）ことと同程度に，信頼性を定量的に評価することは困難な問題**
  - 多くのソフトウェアの障害は決定論的 (deterministic)
  - ソフトウェアにフォールトが含まれないことを証明することは困難
- **広義のソフトウェア信頼**： ソフトウェアシステムが，定められた環境の下で規定の期間中，健全に動作する能力または度合い
- **狭義のソフトウェア信頼**： ソフトウェアシステムが，定められた環境の下で規定の期間中，健全に動作する確率（**ソフトウェア信頼度**）
- **伝統的ソフトウェア工学における信頼性評価技術**：テスト環境において観測された情報に基づいて，ユーザもしくはマーケットにおいてソフトウェアフォールトに起因する障害が発生しない確率を推定

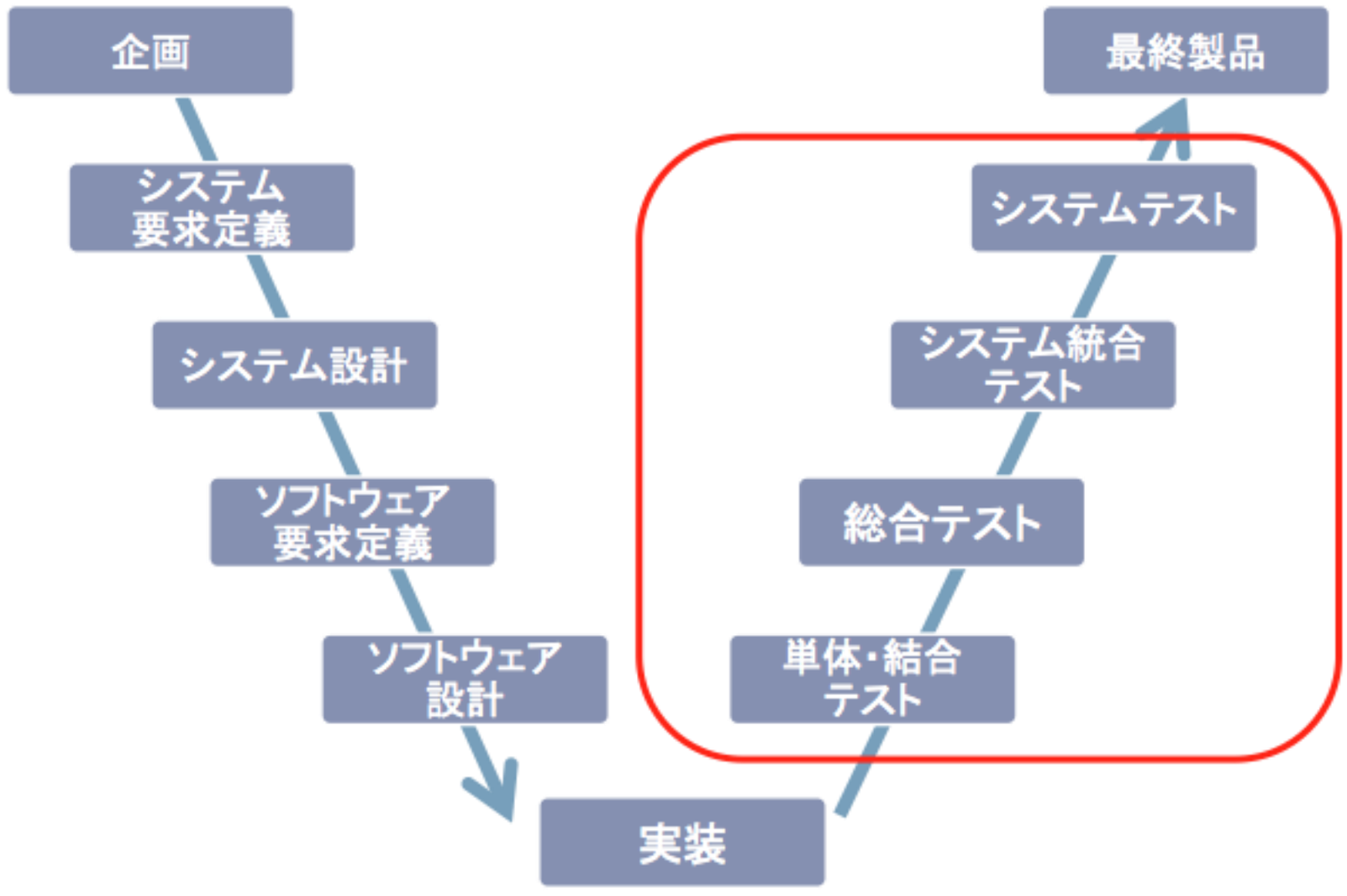
ソフトウェア信頼性評価

- **信頼性評価の意義**
  - システム開発プロジェクト（システムライフサイクル）におけるプロダクト品質に関する管理項目
  - 目に見えないシステム品質である「信頼性」を見える化（定量化）
  - 品質保証のためのベストプラクティス

- **定量的な信頼性評価の効果**
  - 定量的な品質測定（当たり前品質としての ``信頼性の役割’’ ）
  - システム出荷判定・受け入れ判定
  - 資源（人・金・物）配分の管理： 効率の良い開発労力（人員）の配分
  - 保守プロセス計画： 保守（パッチなど）のリリース計画

テスト工程におけるソフトウェアの品質測定

ソフトウェア製品の信頼性を定量化



HIROSHIMA UNIVERSITY

2. 研究目標

- **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**
  - 信頼性評価の妥当性検証を可能にすること
  - 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
- **実証ソフトウェア信頼性工学 (Empirical Software Reliability Engineering)**
- **信頼性と可観測な要因の因果関係の解明**
  - 開発情報とソフトウェア信頼度の相関のモデル化
  - 確率・統計に基づいた要因分析
  - 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
- **ソフトウェア信頼性評価技術の啓蒙**
  - モデルと推定手続きの標準化
  - ツール開発

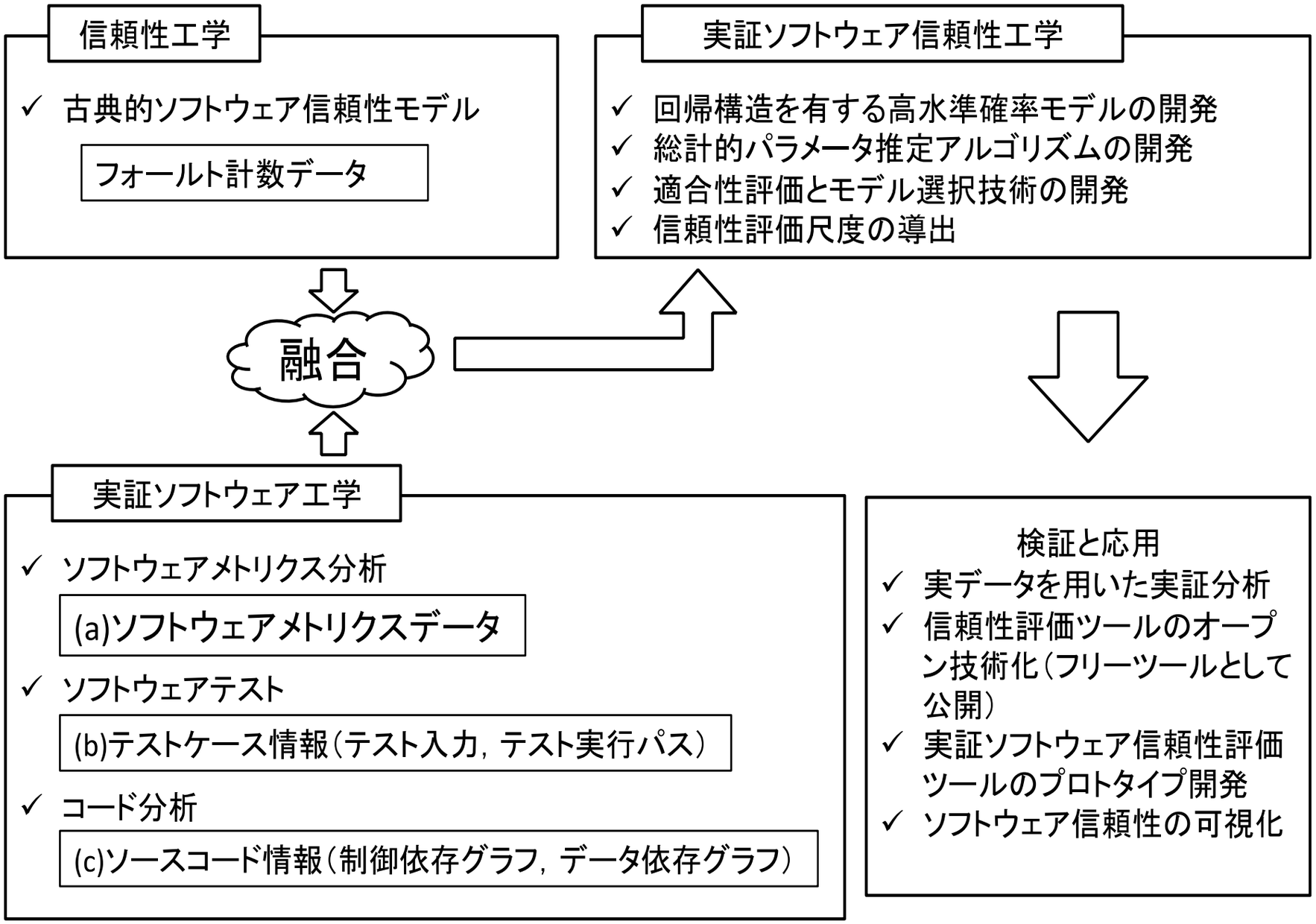
委託研究における研究課題

- **開発現場で成し得る情報計測の様々な階層に対応して，開発情報を信頼性評価に如何に利用するか？**
  - 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が利用可能な場合
  - テストケースの詳細な情報が利用可能な場合
  - メトリクスを抽出することなく，ソースコードの静的分析結果が利用可能な場合
- **基本的なアイデア**
  - メトリクスの回帰構造を組込んだ高水準確率モデルと推定アルゴリズムの開発
  - 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
  - 依存グラフとフォールトの作り込み・発見現象のモデル化

Contd.-

- **理論的枠組み**
  - 「初期潜在フォールト数」と「フォールト発見確率」の2つのパラメータに回帰構造を導入した一般化線形モデルの開発
  - 一般化線形モデルに対して最尤推定原理に基づいたパラメータ推定アルゴリズム (EM アルゴリズム）の開発
  - テストケースの距離空間 (Levenshtein 距離) の表現
  - ソースコード上の文字列情報の抽出
  - カーネル回帰の導入
- **ツール化における基本方針（2種類のツール開発）**
  - 汎用的な表計算ソフトウェア（Microsoft Excel) の拡張機能（AddIn）として開発
  - 統計ソフト R の機能を活用することで安定した統計処理計算を実現

委託研究課題の俯瞰



研究課題の概念図

HIROSHIMA UNIVERSITY

3. 研究成果

一般化線形モデルとカーネル推定法の構築（中間目標1）

$p_{i,k}$ ：i 番目のモジュールにおいて時刻 k でフォールトを発見する確率

ξ_i ：i 番目のモジュールにおいてテスト開始前にソフトウェア内に潜在する総フォールト数の平均値

$Y_{i,k}$ ：i 番目のモジュールにおいて時刻 k で発見される累積総フォールト数

$$\mathrm{P}(Y_{i,k} = y_{i,k}) = \frac{(\xi_i(1 - \prod_{k=1}^{n_i} \overline{p}_{i,k}))^{y_{i,k}}}{y_{i,k}!} \exp(-\xi_i(1 - \prod_{k=1}^{n_i} \overline{p}_{i,k}))$$

(1) メトリクスを活用したソフトウェア信頼性評価

- 総フォールト数の平均に対する回帰表現（Poisson 回帰）：静的メトリクス

$$\xi_i = \exp \beta_0 + \boldsymbol{\beta} \boldsymbol{s}_l$$

- フォールト発見確率に対する回帰表現（Logistic 回帰）：動的メトリクス

$$p_{i,k} = \frac{1}{1 + \exp(\alpha_{0,i} + \boldsymbol{\alpha}_i \boldsymbol{x}_{i,k})}$$

$\boldsymbol{s}_l$ ： ソフトウェアデザインメトリクスト（テスト開始以前に潜在する初期フォールト数に関与するメトリクスデータ）

$\boldsymbol{x}_{i,k}$ ： ソフトウェアテストメトリクス（デバッグに関与するメトリクスデータ）

- Poisson 回帰とLogistic 回帰を組合せた**統合モデル**：静的メトリクスと動的メトリクスの両方

(2) カーネル法に基づいたソフトウェア信頼性評価

一般化線形モデルをカーネル回帰に拡張

一般化線形モデルと同じ確率関数形を用いて, 回帰式にカーネルを適用

$$\log \xi_i = \gamma_0 + \sum_{l=1}^{M} \gamma_l K(m_i, m_l)$$

$$\log \frac{p_{i,k}}{1 - p_{i,k}} = \psi_{i,0} + \sum_{l=1}_{D_i} \psi_{i,k} K(t_{i,k}, t_{i,l})$$

M : モジュール数

D_i : モジュール i のテストケース数

$K(m_i, m_j)$: モジュール i とモジュール j に対するカーネル

$K(t_{i,k}, t_{i,l})$: モジュール i のテストケース k と l に対するカーネル

Contd.-

具体的なカーネル表現

- **テストケース情報**
  - テスト入力に基づいたテストケース間の距離
    - Levenshtein 距離
  - テスト入力とフォールト発見確率の関連を記述
- **ソースコード情報**
  - ソースコードの文字列に基づいたモジュール間の距離
    - コードクローンセット
  - モジュールの特性とフォールト数の関連を記述

統計的推定アルゴリズム（中間目標2）

一般化線形モデルに対する最尤推定（EMアルゴリズム）

- Step0 : 未観測データの決定 $\boldsymbol{M} = (M_1, \ldots, M_j)$
- Step1: パラメータの更新（収束するまで繰り返す）

for i=1 ; j

$$(\beta_0, \boldsymbol{\beta}) \leftarrow \mathrm{PR}(\boldsymbol{s}_i \ldots \boldsymbol{s}_j, \boldsymbol{M})$$

$$\boldsymbol{R_i} = M_i - \boldsymbol{y_i}$$

$$(\alpha_{0,i}, \boldsymbol{\alpha}_i) \leftarrow \mathrm{LR}(\boldsymbol{y_i}, \boldsymbol{R_i}, \boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,n_i})$$

$$\xi_i \leftarrow \exp(\beta_0 + \boldsymbol{\beta s}_i)$$

$$M_i \leftarrow y_{i,n_i} + \xi_i \left(1 - \prod_{k=1}^{n_i} \frac{1}{1 + \exp(\alpha_{0,i} + \boldsymbol{\alpha}_i \boldsymbol{x}_{i,k})} \right)$$

end

PR() : Poisson回帰モジュール
LR() : Logistic回帰モジュール
$y_{i,ni}$: n_i 番目のテスト期間においてモジュールiで検出された累積フォールト数

カーネル法を適用した場合の推定

カーネル法における問題点
・パラメータ数（自由度）とデータ数が同じ
・正則化法（罰則付き最尤法）
・正則化パラメータはABIC最小化により決定

$$\mathrm{PLLF}(\boldsymbol{\theta}) = \mathrm{LLF}(\boldsymbol{\theta}) - \lambda \boldsymbol{\theta}^T \boldsymbol{K} \boldsymbol{\theta}$$

$$\mathrm{ABIC} = -2 \max \mathrm{PLLF}(\boldsymbol{\theta}) - \frac{2}{|\boldsymbol{\theta}|} \log 2\pi + \log \det \boldsymbol{I}(\boldsymbol{\theta}) + 2|\boldsymbol{\theta}|$$

$\boldsymbol{\theta}$ ：パラメータベクトル
$\boldsymbol{K}$ ：正定値行列
λ ：正則化パラメータ

$\mathrm{PLLF}(\boldsymbol{\theta})$：罰則付き対数尤度関数
$\mathrm{LLF}(\boldsymbol{\theta})$：対数尤度関数
$\boldsymbol{I}(\boldsymbol{\theta})$ ：フィッシャー情報行列

カーネル法に対する推定アルゴリズム

LR, PR を罰則付き最尤法へ置き換える

- Step0 : 未観測データの決定 $\boldsymbol{M} = (M_1, \ldots, M_j)$
- Step1: パラメータの更新（収束するまで繰り返す）

for *i=1 ; j*

$$(\beta_0, \boldsymbol{\beta}) \leftarrow \mathrm{PR}(\boldsymbol{s}_i \ldots \boldsymbol{s}_j, \boldsymbol{M})$$

$$\boldsymbol{R_i} = M_i - \boldsymbol{y_i}$$

$$(\alpha_{0,i}, \boldsymbol{\alpha}_i) \leftarrow \mathrm{LR}(\boldsymbol{y_i}, \boldsymbol{R_i}, \boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,n_i})$$

$$\xi_i \leftarrow \exp(\beta_0 + \boldsymbol{\beta s}_i)$$

$$M_i \leftarrow y_{i,n_i} + \xi_i \left(1 - \prod_{k=1}^{n_i} \frac{1}{1 + \exp(\alpha_{0,i} + \boldsymbol{\alpha}_i \boldsymbol{x}_{i,k})} \right)$$

end

近似アルゴリズム

高速化に対する工夫

PR, LR のPoisson回帰，Logistic回帰モジュールにおいて，
重み付き最小二乗法を繰り返し解く必要がある

→ LAPACK (Linear Algebra Package) における重み付き最小二乗関数を利用することで計算そのものを高速化した.
（内部では，QR分解，特異値分解を使い分けた演算を行っている）

ファイル単位あるいはテストケース単位のデータを扱うとデータ数が増加

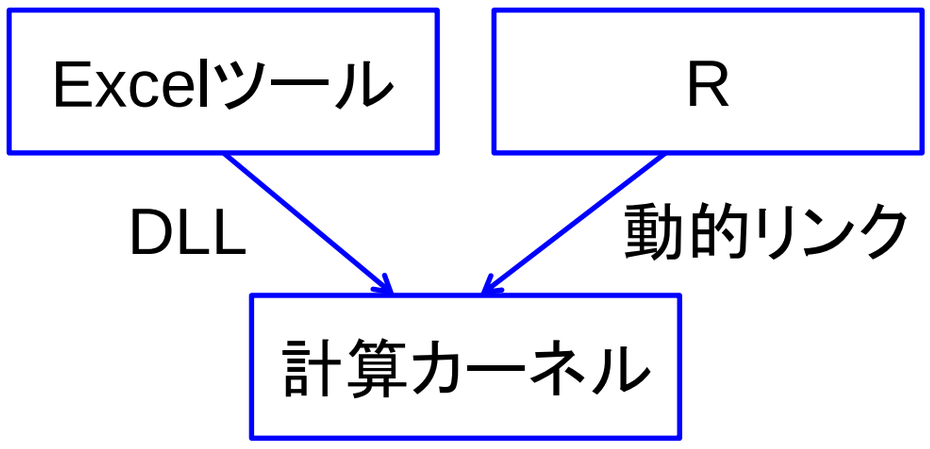
→ データと要因をランダムにサンプリング（ブートストラップ）して
部分的なデータセットを作成し，部分的なデータセットに対して
推定を行う（ランダムフォレスト）.
（LAPACKの利用で十分な高速化が行えたので試験的な機能として実装）

ツール開発 (中間目標4)

ツール開発規模とアーキテクチャ

計算カーネル
・C++による開発（約 6,500 LOC）
・NHPPモデルに対する既存のEMアルゴリズム
・GLM（一般化線形モデル）に対する重み付き最小二乗法
・GLMに対する罰則付き最尤法
・一般化線形モデルを適用した信頼性モデルに対する
　EMアルゴリズム（罰則あり／なし）

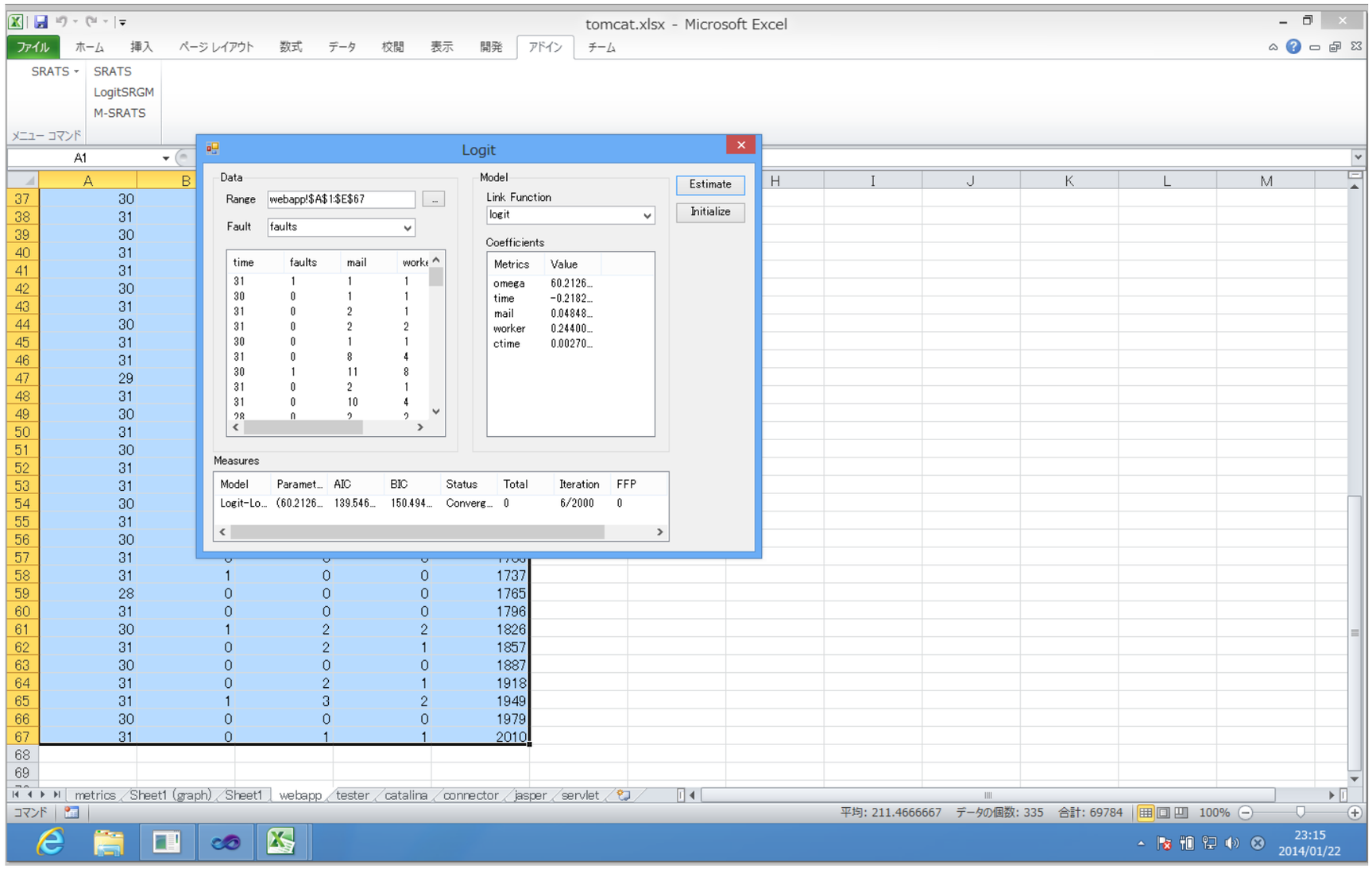
Excelインタフェース
・C#による開発（約 5,500 LOC）
・ユーザインタフェースとグラフ描画



Rインタフェース
・R言語による開発（約 1,000 LOC）
・ユーザインタフェース
・ランダムフォレスト，ABIC最適化をR言語により実装

ツール開発 (中間目標4)

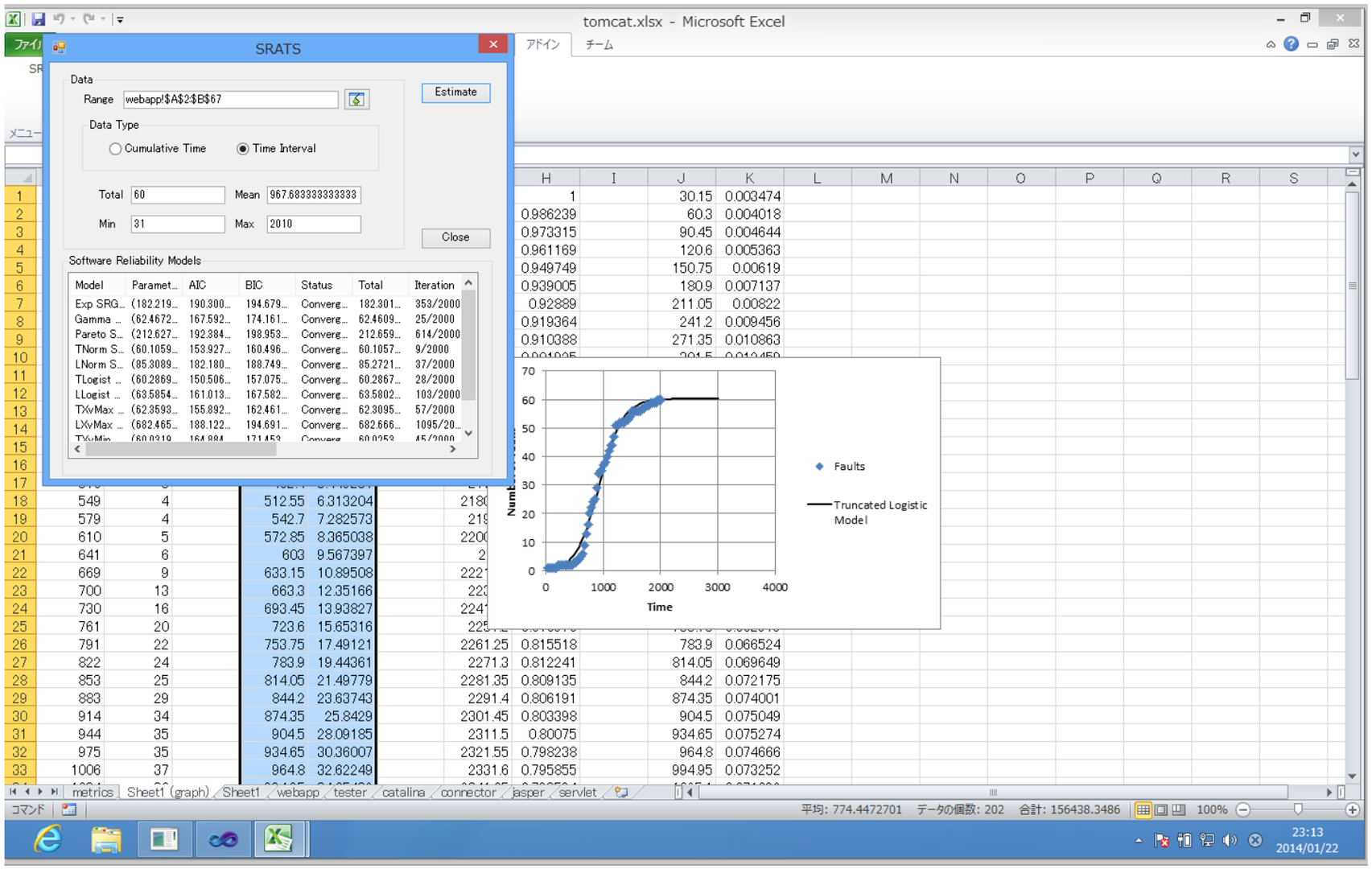
Excel ツールの起動画面（Logistic回帰）



HIROSHIMA UNIVERSITY

Contd.-

Excel ツールによる評価結果の出力例



推定ツール（Rパッケージ）～推定結果

```
Status: Convergence

Maximum LLF: -696.7107

AIC: 1421.421

SRMS for each modules:
$webapps

Truncated Extreme-Value Max SRGM: 44.36196 857.4068 485.4593
Total Faults: 65.29736
Residual Faults: 0.2973602

$tester

Log-Normal SRGM: 26.07633 10.23931 1.213098
Total Faults: 26.06588
Residual Faults: 25.06588

$catalina

Log-Normal SRGM: 287.4735 7.00063 0.4843549
Total Faults: 275.379
Residual Faults: 3.378957

$connectors

Log Extreme-Value Max SRGM: 718.1013 9.917643 2.562109
Total Faults: 713.2256
Residual Faults: 624.2256

$jasper

Log-Logistic SRGM: 48.52472 6.848826 0.2637445
Total Faults: 74.42234
Residual Faults: 0.4223376

$servlets

Log Extreme-Value Max SRGM: 27.67908 6.689805 0.3768252
Total Faults: 57.65213
Residual Faults: 0.6521339

 警告メッセージ:
1: In dpois(y, mu, log = TRUE) : non-integer x = 65.477900
2: In dpois(y, mu, log = TRUE) : non-integer x = 275.175342
3: In dpois(y, mu, log = TRUE) : non-integer x = 95.608481
4: In dpois(y, mu, log = TRUE) : non-integer x = 74.695674
5: In dpois(y, mu, log = TRUE) : non-integer x = 58.681056
>
```

R を用いる利点：

✓大量のデータ解析に対して有利（R言語によりユーザが手軽にプログラミング可能）
✓回帰モデルにおける既存の変数選択機能やブートストラップ機能などが適用可能

R を用いる欠点：

✓学術領域以外では一般的ではない
✓表計算ソフトのような簡便さがないため，初学者が気軽に利用するための敷居が高い（SRATS の成功例）

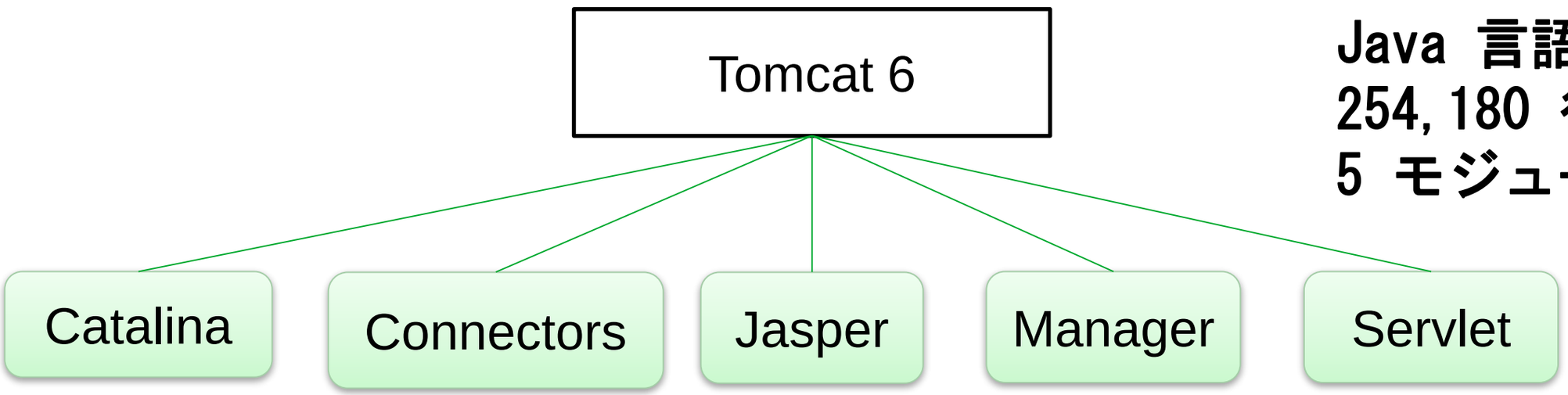
ツールの基本的な機能

- **一般化線形モデル**
  - NHPP, Poisson回帰, Logistic回帰, 統合モデル
  - 最尤法・罰則付き最尤法によるパラメータ推定
  - 情報量基準 (AIC) によるモデル選択
  - 信頼性評価(平均値関数, 強度関数, FFP)尺度の算出
  - 信頼性評価尺度の描画機能(Excelツール)
- **カーネル回帰モデル**
  - カーネル値をデータとして入力(カーネル値の計算は外部ツールに依存)
  - 罰則付き最尤法によるパラメータ推定
  - 情報量基準 (ABIC) による正則化パラメータの推定(Rツール)
  - 信頼性評価(平均値関数, 強度関数, FFP)尺度の算出

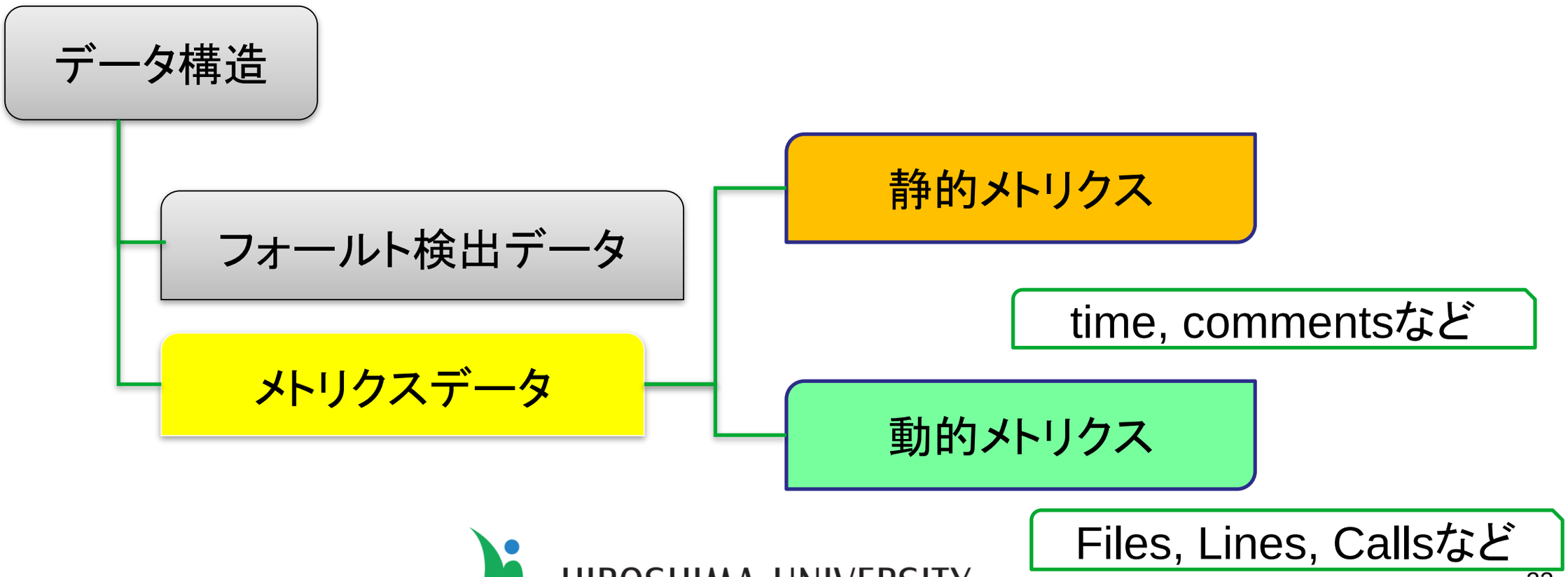
有効性を検証するための実証分析 (中間目標 3)

- **実験 I:** Tomcat 6
  - ✓ 一般化線形モデルにおいて, ソフトウェアメトリクスの情報が信頼性評価に与える影響について評価
  - ✓ 実際のオープンソースプロジェクトデータを対象に, NHPP モデル, Poisson 回帰モデル, Logistic 回帰モデル, 統合モデルを比較
- **実験II:** ``replace'' モジュール
  - ✓ テストケースの入力と実行パスの関係がソフトウェア信頼性評価に与える影響について評価
- **実験 III**: ``catalina'' モジュール(Tomcat 6)
  - ✓ ソースコード分析においてファイル間の類似度をコードクローンによって計測し, 信頼性評価に適用
  - ✓ CCFinder (大阪大学)をチェーンツールとして活用し, コードクローンセットを特定

実験 I



Java 言語
254,180 行
5 モジュール



Tomcat 6 の評価結果

AIC							Poireg0	
	model			NHPP	Logit	Metrics(A)	NHPP	Logit
Catalina	Truncated Extreme-Value			489.2961	388.6588	2,3,4	Metrics(B)	Metrics(B)
Connectors	Log Extreme-Value Max			277.7447	201.743	2,3,4	1,7,10	1,2,6,7
Jasper	Truncated Extreme-Value Max			293.5129	247.2723	1,2		
Manager	Log Extreme-Value Min			163.1457	125.5798	2,4		
Servlet	Log Extreme-Value Max			181.5431	181.5806	2,4		
				1405.243	1144.835		1403.823	1144.835

Metrics												
A	1	2	3	4								
	time	comments	votes	ctime								
B	1	2	3	4	5	6	7	8	9	10	11	12
	Files	Lines	Statements	Branches	Calls	Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Avg Depth	Avg Complexity

ABIC	Poireg1		Poireg2	
	NHPP	Logit	NHPP	Logit
lambda	9.45E-18	1.9E-08	2.32E-16	1.13E-07
abic	1303.163	1211.307	1366.362	1260.081

Contd.-

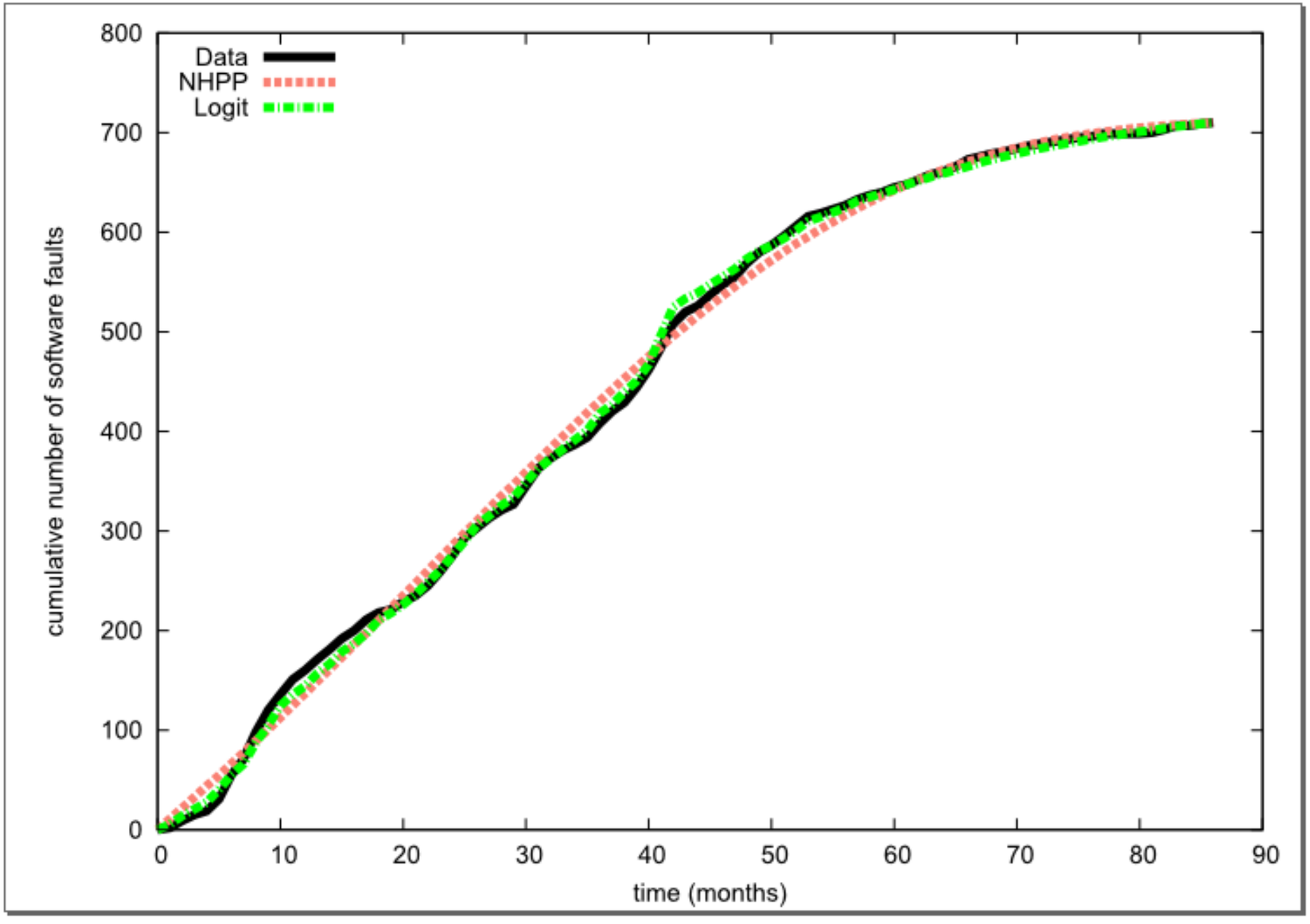
Total Faults			Poireg0		Poireg1		Poireg2	
	NHPP	Logit	NHPP	Logit	NHPP	Logit	NHPP	Logit
Catalina	713.553	723.7668	713.656	723.7766	713.5463	723.7766	713.5463	723.7766
Connectors	167.4882	121.3126	169.305	121.3148	167.4239	121.3148	167.4239	121.3148
Jasper	177.2828	312.0477	176.8449	311.3198	177.2896	311.3198	177.2896	311.3198
Manager	47.17135	47.36607	47.1899	47.36623	47.17084	47.36623	47.17084	47.36623
Servlet	87.00858	78.44292	86.00088	78.44301	87.00584	78.44301	87.00584	78.44301

Residual Faults			Poireg0		Poireg1		Poireg2	
	NHPP	Logit	NHPP	Logit	NHPP	Logit	NHPP	Logit
Catalina	3.553001	13.76683	3.655974	13.77662	3.546294	13.77662	3.546294	13.77662
Connectors	48.48816	2.312623	50.30504	2.314787	48.42395	2.314787	48.42395	2.314787
Jasper	10.28278	145.0477	9.844939	144.3198	10.28958	144.3198	10.28958	144.3198
Manager	0.171347	0.366068	0.189897	0.36623	0.170836	0.36623	0.170836	0.36623
Servlet	9.008584	0.442918	8.000884	0.44301	9.005845	0.44301	9.005845	0.44301

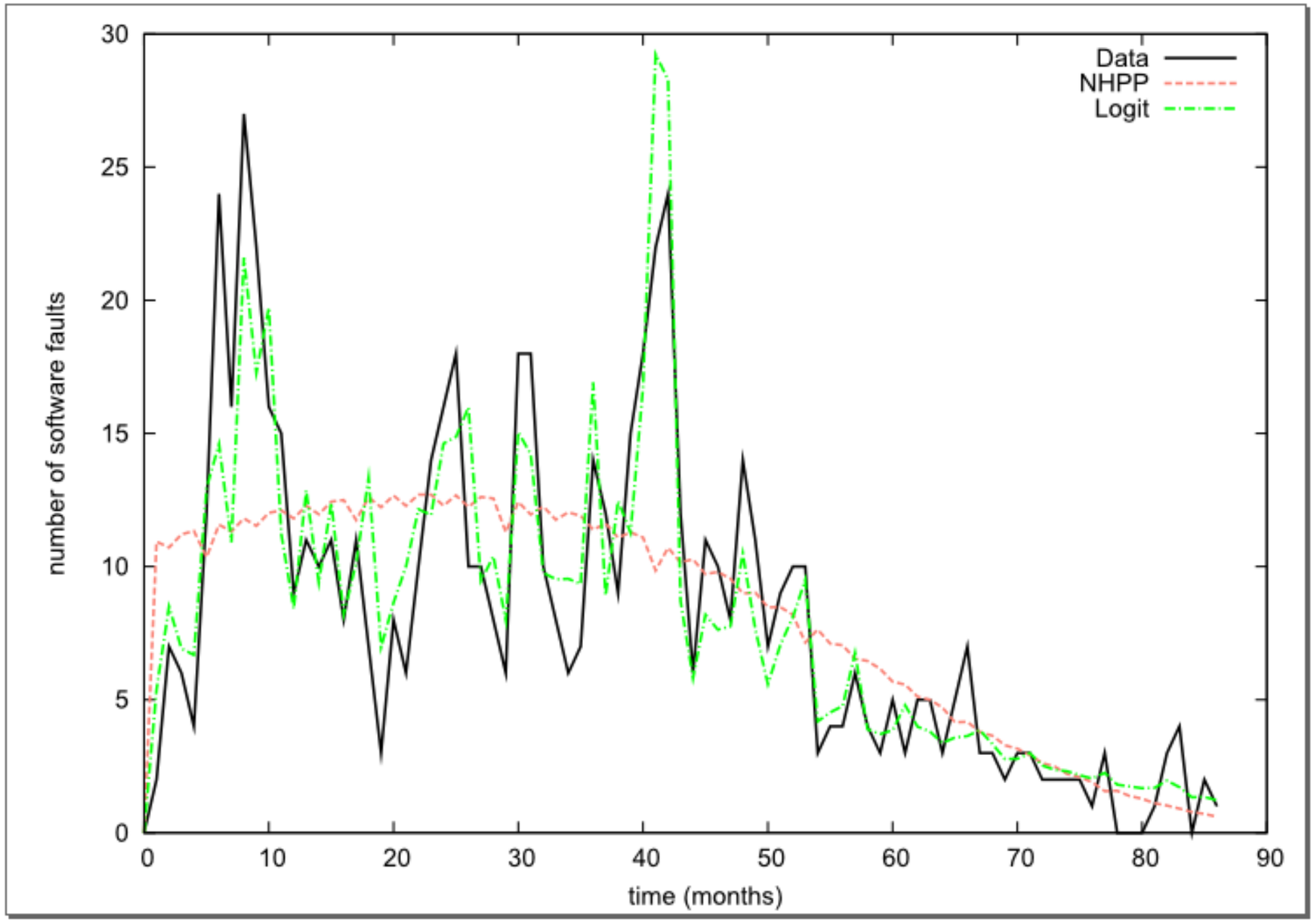
Real Data	
	Total Fault
Catalina	710
Connectors	119
Jasper	167
Manager	47
Servlet	78

FFP			Poireg0		Poireg1		Poireg2	
	NHPP	Logit	NHPP	Logit	NHPP	Logit	NHPP	Logit
Catalina	0.028639	1.05E-06	0.025836	1.04E-06	0.028831	1.04E-06	0.028831	1.04E-06
Connectors	8.75E-22	0.099001	1.42E-22	0.098787	9.33E-22	0.098787	9.33E-22	0.098787
Jasper	3.42E-05	1.02E-63	5.3E-05	2.1E-63	3.4E-05	2.1E-63	3.4E-05	2.1E-63
Manager	0.842529	0.693456	0.827044	0.693343	0.84296	0.693343	0.84296	0.693343
Servlet	0.000122	0.64216	0.000335	0.642101	0.000123	0.642101	0.000123	0.642101

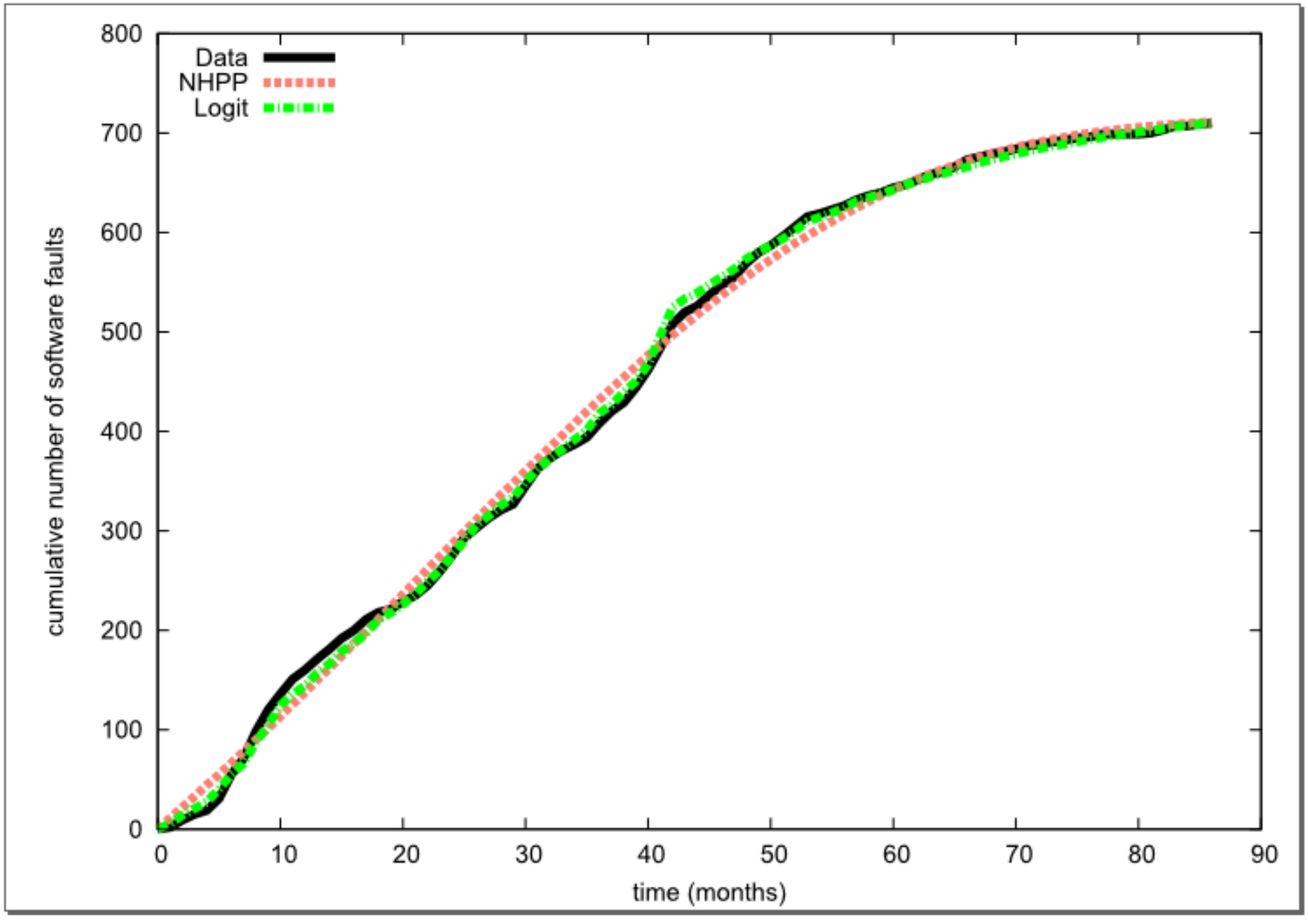
期待累積フォールト数（平均値関数）の推定結果
Tomcat 6 (Catalina): NHPP v.s. Logistic 回帰モデル



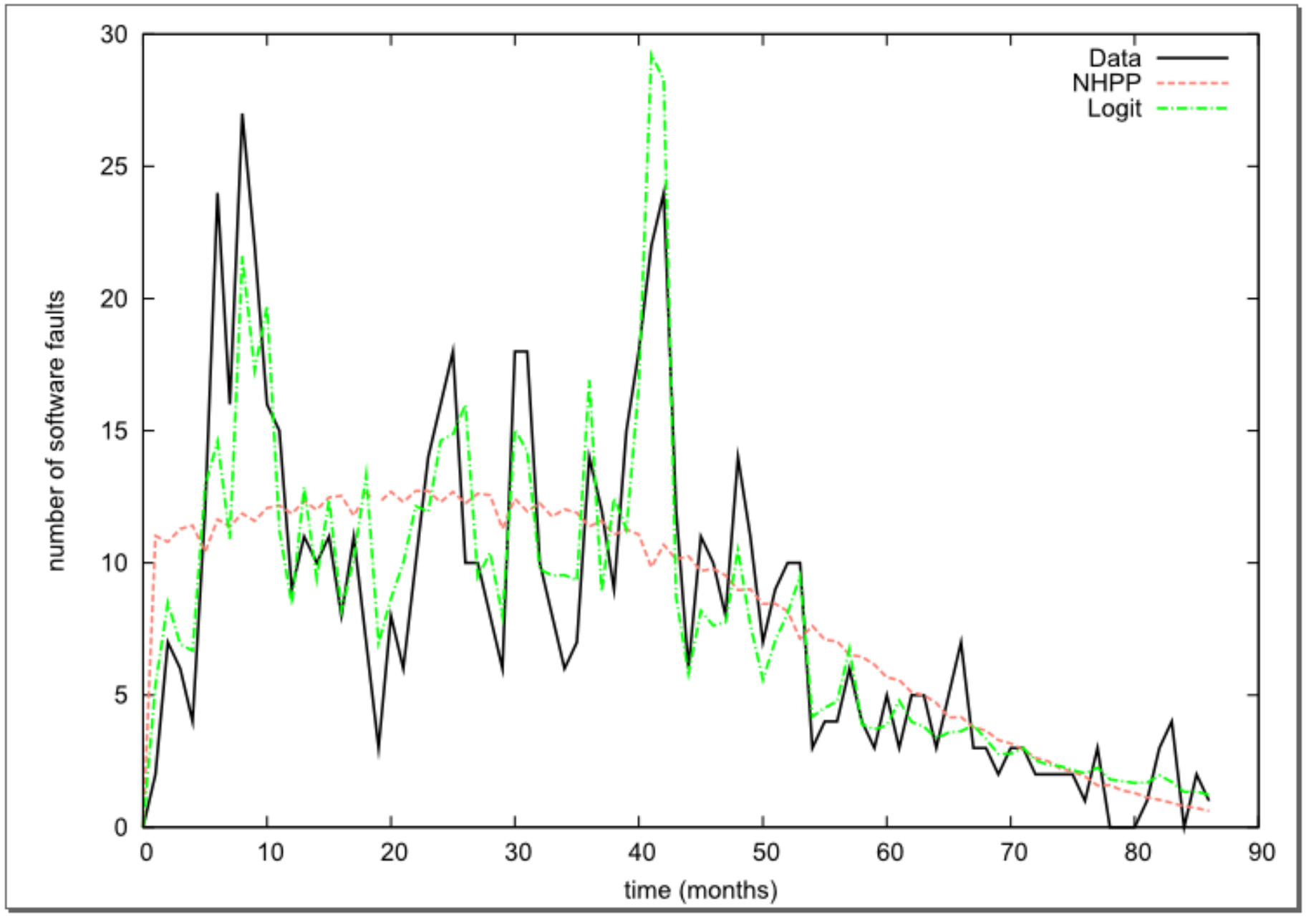
期待フォールト数(強度値関数)の推定結果
Tomcat 6 (Catalina): NHPP v.s. Logistic 回帰モデル



期待累積フォールト数（平均値関数）の推定結果
Tomcat 6 (Catalina): Poisson 回帰モデルP v.s. 統合モデル



期待フォールト数(強度関数)の推定結果
Tomcat 6 (Catalina): Poisson 回帰モデルP v.s. 統合モデル



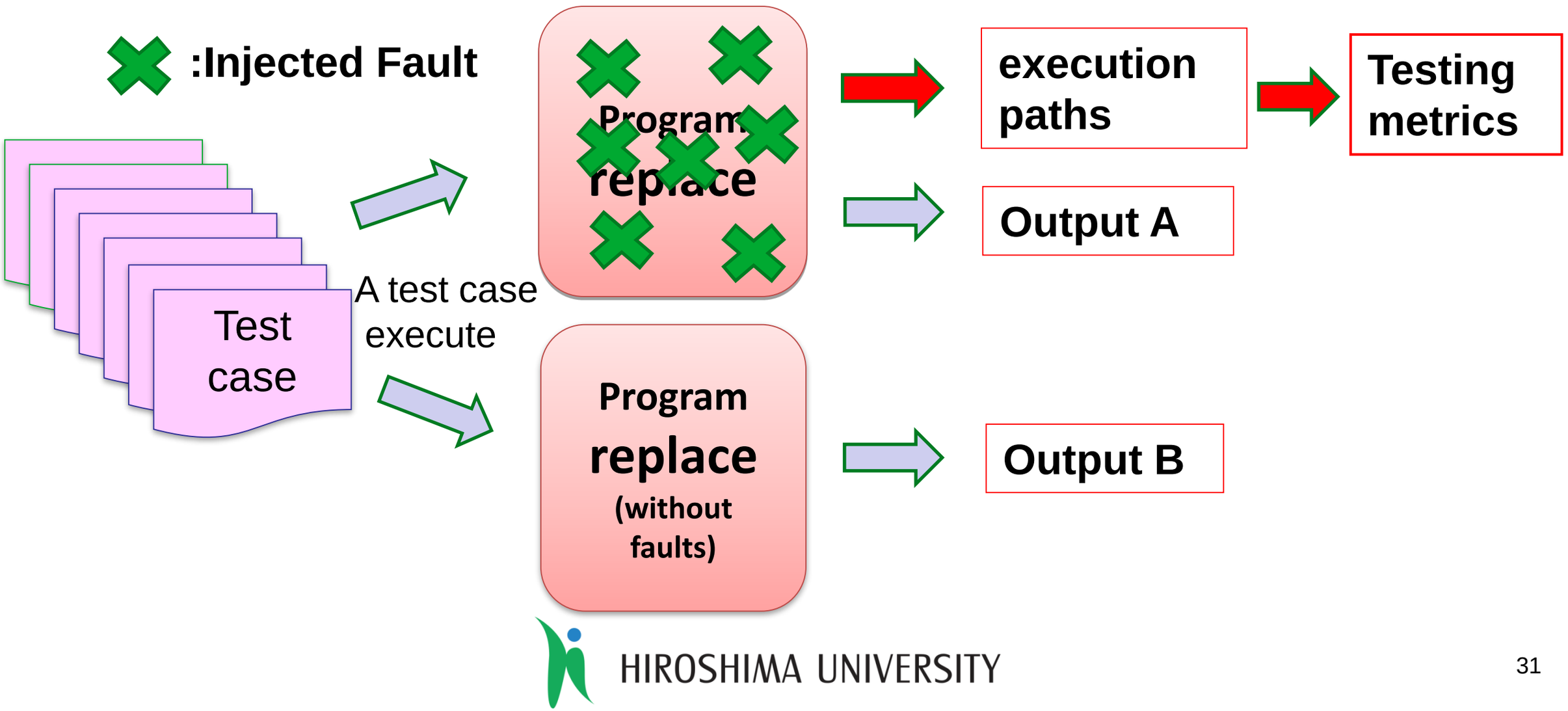
実験Ｉから得られた知見

- 静的メトリクスと動的メトリクスを両方含む統合モデルは適合性の観点から他のモデルに劣ることはないが，変数選択を行うメリットは大きい
- 適合性評価の観点からは静的メトリクスよりも動的メトリクスの推定精度への寄与が高い
- 要因選択や回帰係数の値から，ソフトウェア信頼度推定に影響を与える（構造上もしくは管理上の）要因を特定することが可能となり，その結果をプロセス改善に役立てることが出来る
- モジュール単位での信頼性評価を実施するために有効なソフトウェア信頼性評価技術であり，オープンソースプロジェクトデータや各種リポジトデータを活用するための標準モデル （ツール）となり得る

実験 II

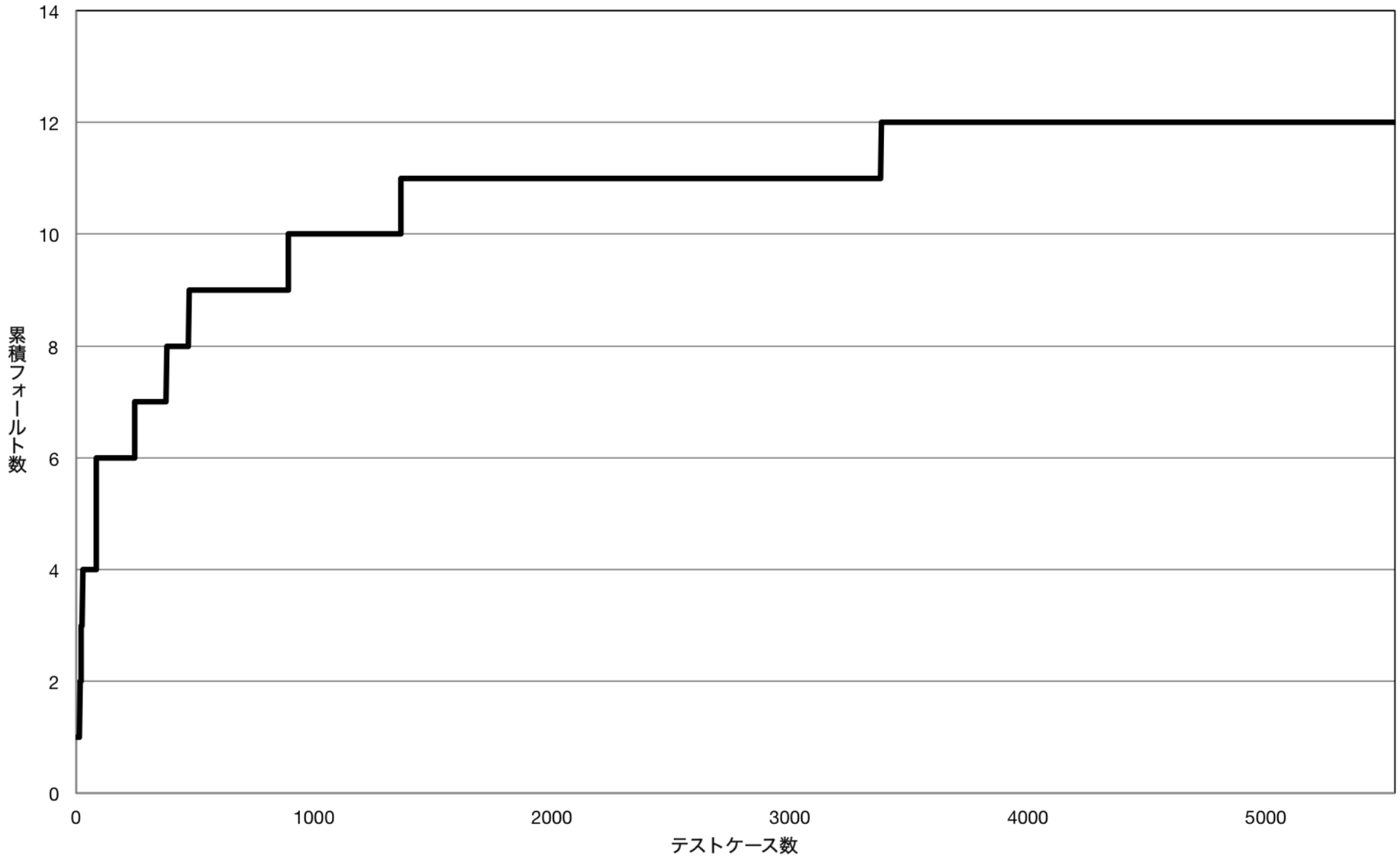
対象： SIR (Software-artifact Infrastructure Repository) で公開されているプロジェクトのパターンマッチングプログラム“replace” (C 言語, 212 ステートメント)

実験手順（フォールト・インジェクション実験）：12個のフォールトを予め埋め込んでおき, 5,542 テストケースを実行



HIROSHIMA UNIVERSITY

フォールト検出のふるまい



テストケース例

#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&' <

テストケース間の距離とカーネル値

- テストケースの距離
  - テスト入力文字列に対する編集距離（Levenshtein距離）

距離の例

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

$$K(t_{i,k}, t_{i,l}) = \exp\left(-\frac{D(t_{i,k}, t_{i,l})}{s}\right)$$

$D(t_{i,k}, t_{i,l})$:テストケース k と l の距離

カーネル Logistic 回帰に上記のカーネルを適用

実験方法と推定値

Case 1: 最初の50個のテストケースの距離とテスト結果を使ってパラメータを推定し，50個以降の予測やデータへの適合を見る.

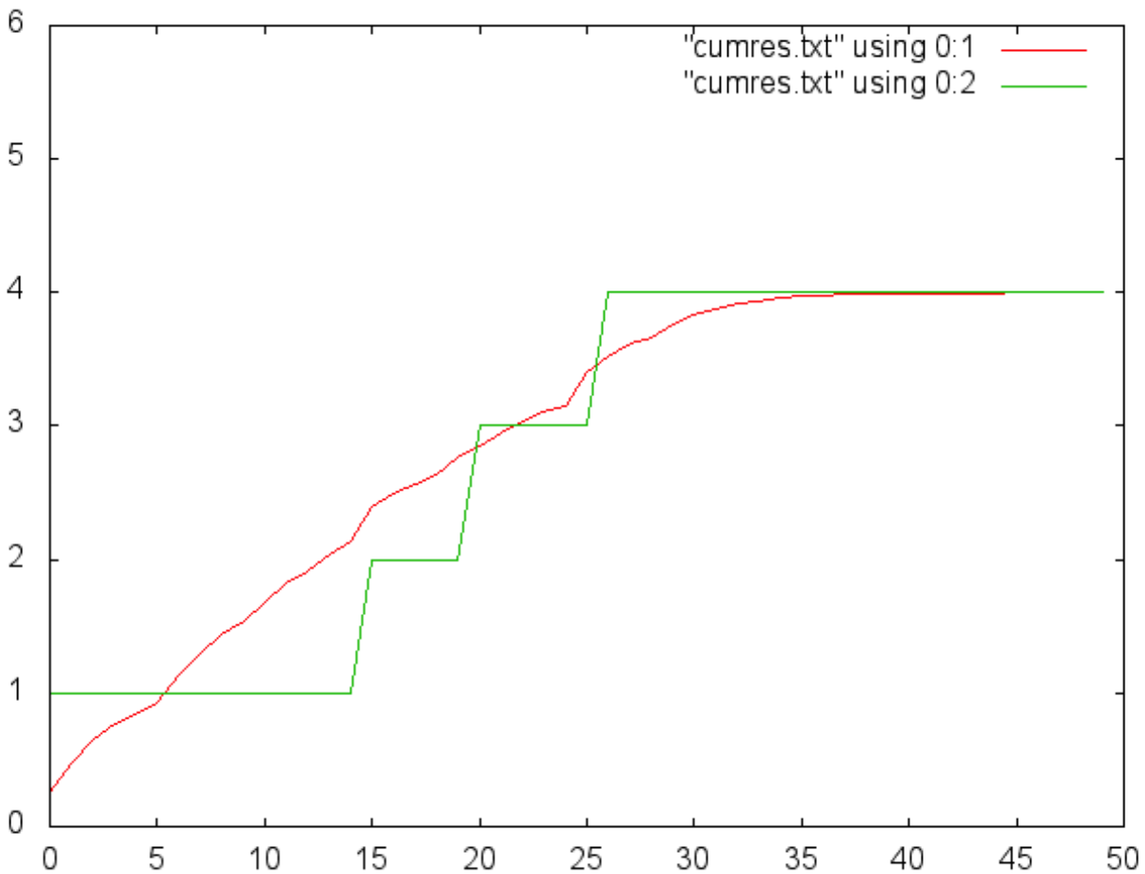
総期待フォールト数: 4.0
期待残存フォールト数: 0.005
ABIC: -136.40
λ: 981.24

Case 2: 最初の100個のテストケースの距離とテスト結果を使ってパラメータを推定し，100個以降の予測やデータへの適合を見る.

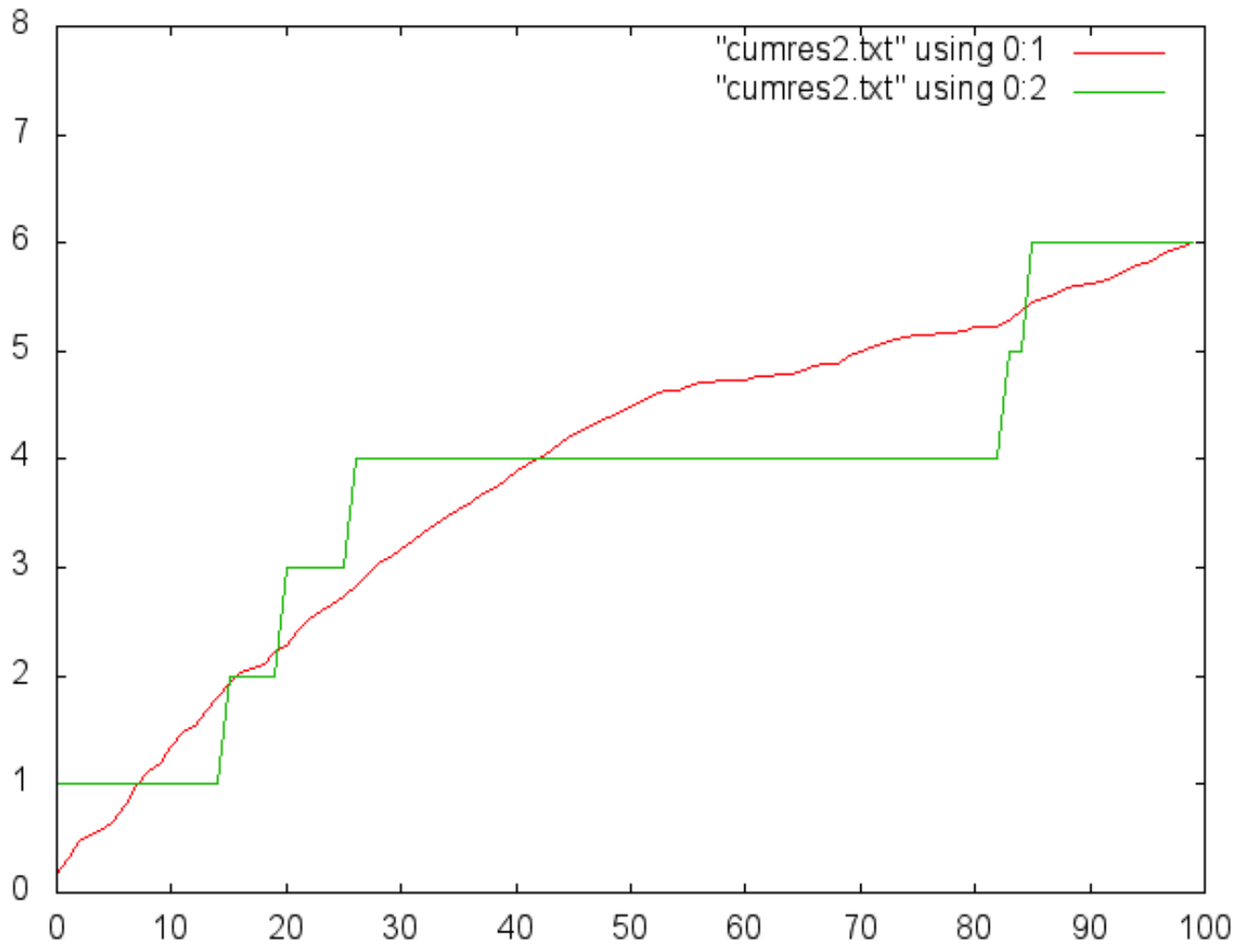
総期待フォールト数: 7.6
期待残存フォールト数: 1.6
ABIC: 15.26
λ: 1037.5

累積フォールト数

Case 1



Case 2



実験 II から得られた知見

- データへの適合性が高い

  モデル選択や変数選択を行わなくても比較的高い適合性のモデルが得られる
- 残存フォールト数など将来の予測に関しては従来の NHPP 同様に，観測されているデータから類推できる程度の予測性能を保証（これまでの累積数の増え具合から予想できる）
- 将来のテストケース情報を与えることにより高い精度の予測が行える可能性がある
- テスト入力情報を信頼性評価に組込むための基本的枠組みを提供

実験 III

対象：Tomcat 6 （オープンソース）Catalina モジュール内
の Java ファイル（ファイル数 31）

類似度の計算手順：
(1) CCFinder（大阪大学）を用いてコードクローンセットの特定
(2) 2つのファイル間に共通するコードクローンセット数を表したマトリクス表現
(3) Poisson 回帰モデルへの適用（ファイル j の総フォールト数 ω_j の回帰式)

$$\log \omega_j = \beta_0 + \sum_{i=1}^{31} \beta_i \exp(-m_{i,j})$$

ここで, m_{ij} はCCFinder から得られたコードクローンセットのマトリクスの(i,j) 要素

モジュール間の類似度マトリクスの例

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.ja	1	0	0	0	0	0	0
StandardHostV	0	0	0	0	0	0	0
StandardPipelir	0	0	0	0	0	0	0
StandardServe	0	0	0	0	0	0	0

評価結果の比較

	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

実験 III から得られた知見

- 多くのモジュールで総フォールト数，総期待残存フォールト数，FFPにあまり変化は見られなかった
- いくつかのファイルでは大きな変化があった．変化があったモジュールはいずれも，他のファイルとコードクローンセットを共有しないファイルであり，コードクローンが信頼性評価へ何らかの影響を与えることが確認できた．
- 静的ソースコード解析の結果を信頼性評価に組込むための基本的枠組みを提供

4. 結論と今後の課題

研究成果の概要

- **ソフトウェアメトリクス情報を活用した信頼性評価技術の確立**
  - 一般化線形モデル（Poisson 回帰, Logistic 回帰, 統合モデル）の開発
  - EM アルゴリズムの開発と実装
  - Excel （表計算ツール）とR（統計解析ツール）に基づいたツールの開発
- **テストケースの入力の距離を信頼性評価技術に適用する試み**
  - テストケースの距離情報とカーネル回帰による信頼性評価技術の開発
  - Excel とRに基づいたツールの開発
- **ソースコード分析結果を信頼性評価技術に適用する試み**
  - コードクローンによるファイル間類似度を考慮した信頼性評価技術の開発
  - Excel とRに基づいたツールの開発

Contd.-

今後の課題

- **ソフトウェアメトリクス情報を活用した信頼性評価技術のさらなる進展**
  - 変数（要因, メトリクス）選択機能の強化と全自動化
  - リポジトリデータのマイニング技術と大規模データの取扱い
  - 実際のバグトラッキングシステムとの連動
- **テストケースの実行パスの距離を考慮した信頼性評価技術の提案**
  - テストケースの距離情報と他のカーネル回帰表現を検討
  - テストケース距離情報の信頼性評価における有効性のさらなる検証
- **ソースコード分析結果を信頼性評価技術に適用する試み**
  - 各種静的ソースコード解析ツールとの連動
  - コードクローン情報の信頼性評価における有効性のさらなる検証

Contd.-

- **開発ツールの公開とメインテナンス**
  - 広島大学のディペンダブルシステム論研究室のホームページ上で開発ツールを公開
  - 日本語と英語によるマニュアルを準備し, ツールの啓蒙活動を実施
  - ツールのバグ修正や新しい機能追加等のメインテナンスを継続的に行う
- **実用化・汎用化に向けて**
  - 大規模データの取扱い(組合せ爆発の回避, データの前処理労力の軽減)
  - 信頼性評価結果のプロセス改善への施策
- **上流工程におけるソフトウェア信頼性評価技術の体系化とツール支援**
  - ソースコード開発の前段階において, 運用プロファイル, 設計情報, エラー伝搬シナリオに基づいて定量的信頼性評価を行う技術の体系化
  - 形式手法, モデルチェッキング, FMEA の融合

付録

- D. Kuwa and T. 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.
- S. Ikemoto, T. Dohi and H. 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.
- S. Ikemoto, T. Dohi and H. 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.
- D. Kuwa and T. Dohi, “Generalized Cox proportional hazards regression-based software reliability modeling with metrics data,” *Proceedings of The 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2013)*, pp. 328--337, IEEE CPS, 2013.
- H. Okamura and T. 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.
- 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.
- 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.
- 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.