オープンソースソフトウェアに対する
最適バージョンアップ時期推定のためのソフトウェアツール

田村 慶信	肌附 康司	山田 茂	木村 光宏
広島工業大学情報学部	鳥取大学大学院工学研究科	鳥取大学工学部	法政大学工学部
情報工学科	社会開発システム工学専攻	社会開発システム工学科	経営工学科
tam@cc.it-hiroshima.ac.jp	b02t7041z@edu.tottori-u.ac.jp	yamada@sse.tottori-u.ac.jp	kim@k.hosei.ac.jp

概　要

オープンソースソフトウェア (Open Source Software, 以下 OSS と略す) は，セキュリティ向上や開発・保守コストの削減のために，近年 EU 加盟国を中心に欧米において，数多くの企業，自治体，教育機関，政府関係機関などで支持されている．しかしながら，国内では市場に占める割合も小さく，開発環境の整備も今一歩の状態である．特に，サポート体制や品質・信頼性の問題は，その普及を妨げる大きな要因として考えられている．

本論文では，OSS の品質上の問題を改善するための信頼性評価ツールを開発した．特に，本ツールの開発では，フリーのグラフ生成ライブラリである JFreeChart を使用した．これにより，グラフ描画に必要なデータセットを引き渡すだけで高機能なグラフを生成することが可能となり，開発労力を大幅に削減することが可能となる．さらに，オープンソースソフトウェアのリリース候補版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きくかかわるだけでなく，OSS の保守労力の増大に深く関係する．したがって，リリース候補版の信頼性を評価するとともに，最適なバージョンアップ時期を推定することは OSS 開発において重要な段階となる．本論文では，最適なバージョンアップ時期の推定機能を既存のソフトウェアツールに組み込む．

本ツールが，OSS の信頼性に関する評価指標を提示し，より高品質な各種 OSS の開発に結びつくものと考える．

1　はじめに

オープンソースプロジェクトの下で開発されているオープンソースソフトウェア (Open Source Software, 以下 OSS と略す) は，近年 EU 加盟国を中心に欧米においても，数多くの企業，自治体，教育機関，政府関係機関などで支持されている．

一方，OSS の利用に関しては，未だに多くの不安が残されている．まず第 1 に，システム導入後のサポートおよび品質上の問題といった利用者側の一般的な不安がある．第 2 に，OSS は本当にビジネスになるのか，オープンソースのソフトウェアを事業化することによって自社製のソフトウェア商品までが市場を失うことにならないか，といった開発者側の不安がある [1]．特に，サポートや品質上の問題については，OSS の普及を妨げる大きな要因として考えられている．OSS が急速に普及し始めている現在，OSS の信頼性に関するなんらかの計測可能な評価指標を提示することが重要であると思われる．また，OSS に対する現在の研究動向としては，設計工程，開発手法，セキュリティなどを対象とした文献はいくつか提案されているが [2,3]，動的解析に基づいた信頼性評価に関する研究はほとんど行われていないのが現状である．さらに，OSS の発見フォールト数に対する傾向分析に基づく事例研究としての文献がいくつか提案されているが [4,5]，これらは OSS のもつ特有の開発形態を考慮した信頼性評価手法を提案したものではない．

本論文では，OSS の品質上の問題を改善するための信頼性評価ツールを開発する．特に，本ツールの開発では，フリーのグラフ生成ライブラリである JFreeChart [6] を使用する．JFreeChart により，グラフ描画に必要なデータセットを引き渡すだけで高機能なグラフを生成することが可能となり，開発労力を大幅に削減することが可能となる．さらに，実際に公開されている OSS に対して，本信頼性評価ツールを適用する．

また，オープンソースソフトウェアのリリース候補版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きくかかわるだけでなく，OSS の保守労力の増大に深く関係する．したがって，リリース候補版の信頼性を評価するとともに，最適なバージョンアップ時期を推定することは OSS 開発において重要な段階となる．本論文では，最適なバージョンアップ時期の推定機能を既存のソフトウェアツール [7] に組み込む．

本ツールが，OSS の信頼性に関する評価指標を提示し，より高品質な各種 OSS の開発に結びつくものと考える．

2 各コンポーネントに対する信頼性評価

OSS において，システム全体の信頼性に対する各コンポーネントの影響を考えた場合，コンポーネントの規模，フォールト報告者のスキル，フォールト修正の状態，コンポーネントの開発時間，コンポーネント間のパスの数，コンポーネント間の入出力データ量といった様々な要因を考慮する必要がある [8]．こうした複雑な状態を適当な仮定の下で物理的な意味からモデル化することは困難である．したがって，本論文では，各コンポーネント間の内部状態をブラックボックスとして捉えるためにニューラルネットワーク [9] を適用する．すなわち，入力と出力の関係から，その内部構造をニューラルネットワークにより学習させることによって，各コンポーネントがシステム全体の信頼性に与える影響度合いを推定する．これにより，バグトラッキングシステム上から採取されたデータのみに基づいた信頼性評価が可能となることから，実利用上においても容易に適用できるものと考える．本論文においては簡単のために 3 層ニューラルネットワークを適用する．

まず，$w_{ij}^1(i = 1,2,\cdots,I; j = 1,2,\cdots,J)$ を入力層と中間層の結合係数，また $w_{jk}^2(j = 1,2,\cdots,J; k = 1,2,\cdots,K)$ は中間層と出力層の結合係数とする．さらに，$x_i(i = 1,2,\cdots,I)$ は正規化された入力データを表し，本論文では，フォールト報告者により致命的であると判断されたフォールト数，特定の OS において発見されたフォールト数，システムの内部構造に習熟した修正者のフォールト修正数，システムの内部構造に習熟した発見者のフォールト発見数とした．ここで，入力層，中間層，出力層におけるユニットの数を，各々I 個，J 個，および K 個とする．また，各々の層のユニットを示すインデックスを i, j, および k とする．ここで，各々の層のユニットの出力を h_j, y_k とすると，

$$h_j = f\left(\sum_{i=1}^{I} w_{ij}^1 x_i\right), \tag{1}$$

$$y_k = f\left(\sum_{j=1}^{J} w_{jk}^2 h_j\right), \tag{2}$$

となる．但し，$f(\cdot)$ はシグモイド型関数であり，

$$f(x) = \frac{1}{1+e^{-\alpha x}}, \tag{3}$$

として表される．ここで，α はゲインと呼ばれる定数である．ネットワークの学習を行うために，誤差逆伝播法を用いる．ニューラルネットワークの出力層における値を $y_k(k = 1,2,\cdots,K)$ とし，教師パターンを $d_k(k = 1,2,\cdots,K)$ とすると，式 (2) の y_k の評価は次式で与えられる．

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2. \tag{4}$$

ここで，教師パターン $d_k(k = 1,2,\cdots,K)$ には，各コンポーネントにおける累積発見フォールト数データの正規化された値を採用する．すなわち，各コンポーネントにおける累積フォールト発見数データに基づいて，各コンポーネントの重み係数とそれに影響を及ぼす要因との間の結合状態の特徴をニューラルネットワークの結合係数に蓄積させ，ある時点における各コンポーネントの重要度の推定・予測が可能なモデルを考える [7].

3 システム全体に対する信頼性評価

従来から, ソフトウェアの信頼性を定量的に評価する手法として，ソフトウェア故障の発生現象を不確定事象として捉えて確率・統計論的に取り扱う方法がとられている．その 1 つが，SRGM である [10]．中でも非同次ポアソン過程（nonhomogeneous Poisson process，以下 NHPP と略す）モデルは，実利用上極めて有効でありモデルの簡潔性が高いゆえにその適用性も高く，実際のソフトウェア信頼性評価に広く応用されている．この NHPP モデルは，所定の時間区間内に発見されるフォールト数や発生するソフトウェア故障数を観測して，これらの個数を数え上げる計数過程 $\{N(t), t \geq 0\}$ を導入し，以下の式で与えられる確率変数すなわちポアソン過程を仮定する SRGM である [10].

$$\Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!}\exp[-H(t)] \quad (n = 0,1,2,\cdots). \tag{5}$$

ここで，$\Pr\{\cdot\}$ は確率を表し，$H(t)$ は時間区間 $(0,t]$ において発見される総期待フォールト数，すなわち $N(t)$ の期待値を表し，NHPP の平均値関数と呼ばれる．

3.1 システム全体に対する信頼性評価

本論文では，検出可能フォールト数が無限であると仮定された非同次ポアソン過程に基づく対数型ポアソン実行時間モデルを適用する [10].

このモデルでは，時間区間 $(0,t]$ で発見される総期待フォールト数を表す平均値関数 $\mu(t)$ は，

$$\begin{aligned}\mu(t) &= \frac{1}{\theta - P}\ln[\lambda_0(\theta - P)t + 1] \\ & \quad subject\ to\ \ (P-\theta) < \frac{1}{\lambda_0 t} \\ & \quad (0<\theta,\ 0<\lambda_0,\ 0<P<1), \end{aligned} \tag{6}$$

により与えられる．ここで，パラメータ λ_0 は初期故障強度，パラメータ θ はソフトウェア故障 1 個当りの故障強度の減少率を表す．また，パラメータ P はシステム全体に及ぼすコンポーネントの影響率を表す．これは，各コンポーネントに対してニューラルネットワークにより推定されたパラメータ y_i の平均値 $P = \sum_{i=1}^{n} y_i/n$ により表されるものとする．ここで，n はコンポーネント数を表す．さらに，モデルに含まれる未知パラメータの推定方法として最尤法を適用する．上記の NHPP モデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

特に，OSS の開発環境について考えた場合，ソフトウェア内に潜在するフォールト数が有限であると仮定されたモデルよりもむしろ無限回のソフトウェア故障が発生すると仮定されたモデルを適用する方が現実的である．すなわち，OSS の開発では常にフォールト修正やバージョンアップが繰り返されており，使用頻度や人気の高い OSS になるほどフォールト報告が頻繁に行われている．この運用形態は OSS の開発プロジェクトが無意味なものとみなされ解散するまで続けられる．したがって，1 つの企業組織内において，ある特定の使用目的に限定されたソフトウェアの開発を対象としている従来の SRGM では，OSS の信頼度成長現象を十分に包括できないものと考える [11]．本モデルにおけるパラメータ P により，OSS 開発の活動状態を定量的に表すことが可能となる．すなわち，OSS 開発において重要なのは，バグトラッキングシステムへのフォールト登録数やユーザ数の状態によって信頼度成長曲線が大きく変化することであり，こうした OSS 開発の活動状況を把握するための指標として有用であると考える．

3.2 モデルパラメータの推定

モデルに含まれる未知パラメータ λ_0 および θ の推定法として最尤法を適用する．このとき，一定の時刻 t_k までに発見された累積フォールト数 y_k に関する K 組のフォールト発見数データ $(t_k,\ y_k)(k = 1,\ 2,\ \cdots,\ K)$ が観測されているとすると，平均値関数 $\mu(t)$ をもつ NHPP モデルの対数尤度関数は，

$$\begin{aligned}\ln L &= \sum_{k=1}^{K}(y_k - y_{k-1})\ln\left[\mu(t_K) - \mu(t_{k-1})\right] \\ &- \mu(t_K) - \sum_{k=1}^{K}\ln\left[(y_k - y_{k-1})!\right], \end{aligned} \tag{7}$$

となる．ここで，式 (6) の平均値関数 $\mu(t)$ をもつ対数型ポアソン実行時間モデルに含まれる未知パラメータ λ_0 および θ の最尤推定値を求めるために，λ_0 および θ について式 (7) を偏微分する．このとき，

$$\frac{\partial \ln L}{\partial \lambda_0} = \frac{\partial \ln L}{\partial \theta} = 0, \tag{8}$$

とおいて数値的に解くことにより，最尤推定値 $\hat{\lambda}$ および $\hat{\theta}$ を得る．具体的には，$\phi = \lambda_0(\theta - P)$ とおくと，式 (7) より，

$$\begin{aligned}\frac{\partial \ln L}{\partial \phi} &= \sum_{k=1}^{K}(y_k - y_{k-1}) \\ &\cdot \left\{\frac{\left(\frac{t_k}{\phi t_k + 1}\right) - \left(\frac{t_{k-1}}{\phi t_{k-1}+1}\right)}{\ln[\phi t_k + 1] - \ln[\phi t_{k-1} + 1]}\right\} \\ &- \frac{y_n t_n}{(\phi t_n + 1)\ln(\phi t_n + 1)} = 0, \end{aligned} \tag{9}$$

となる．ゆえに，式 (9) から求めた ϕ を使って，$\lambda_0 = \phi/(\theta - P)$ と，

$$y_n = \frac{1}{\theta - P}\ln(\phi t_n + 1), \tag{10}$$

の関係から，モデルパラメータ λ_0 および θ を求めることができる．

3.3 ソフトウェア信頼性評価尺度

式 (6) の平均値関数をもつ NHPP モデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

瞬間フォールト発見率は強度関数により表すことができる．これは，単位時間当りに発見されるフォールト数

として定義される．瞬間フォールト発見率は，式 (6) から以下のように導出できる．

$$\mu_d(t) = \frac{d\mu(t)}{dt}. \tag{11}$$

平均ソフトウェア故障発生時間間隔（mean time between software failures: MTBF）は，ソフトウェア 故障の発生頻度を表すのに有益な尺度である．また，MTBF が大きな値を取ることは，それだけフォールトが発見し難くなり，ソフトウェア信頼性が向上したと判断できることになる．任意の時刻 t における累積 MTBF（cumulative MTBF: $\mathrm{MTBF_C}$）は，以下のように導出できる．運用開始時点から考えたときの発見フォールト 1 個当りに要する発見時間の平均を意味する累積 MTBF は，

$$MTBF_C(t) = \frac{t}{\mu(t)}, \tag{12}$$

により表すことができる．

3.4 モデルの適合性評価

モデルの実測データに対する適合性を評価するために予測相対誤差を適用する．

予測相対誤差（predicted relative error）は，任意の時刻 t_k において推定したときの，フォールト報告終了時刻 t_K までに発見される累積発見フォールト数の予測値と実測値との相対誤差である．時刻 t_k における予測相対誤差を $PRE_k[t_k]$ とすると，

$$PRE_k[t_k] \quad = \quad \frac{\hat{y}(t_k;t_K) - y_K}{y_K}, \tag{13}$$

により計算される．$\hat{y}(t_k;t_K)$ は，時刻 t_k までの実測データを用いたフォールト報告終了時刻 t_K での総期待発見フォールト数の推定値，y_K は，フォールト報告終了時刻 t_K までに発見された総フォールト数の実測値である．

4 最適バージョンアップ問題

4.1 最適バージョンアップ時刻

OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定することは，リリース後の信頼性維持や進捗度管理に役立つと考えられる．本論文では，OSS のバージョンアップ時期の推定方法 [12,13] について議論する．

バージョンアップには大きく分けてマイナーバージョンアップとメジャーバージョンアップがある．しかしながら，どの程度の改訂で区別されるという明確な基準がある訳ではない．マイナーバージョンアップは，旧版にいくつかの細かい機能が追加されたり，性能が若干向上した場合に実施される．ただし，機能の不具合やセキュリティ上の脆弱性を修復するような，クリティカルなフォールトがいくつか発見され緊急に修正を施す必要がある場合に実施される改訂はマイナーバージョンアップではなく，バグフィックスと呼ばれている．一方，メジャーバージョンアップは，OSS 自体の機能が大きく変更されたり，大型の新機能の追加や，性能が劇的に向上した場合に実施される．OSS におけるマイナーバージョンアップは，不定期かつ頻繁に実施されていることから，その推定時期を予測することは困難であり，たとえマイナーバージョンアップ時期が推定できたとしても，その有益性は小さいと考えられる．したがって，本論文では，メジャーバージョンアップを対象とし，前回のバージョンアップ期間に基づいて，次期バージョンアップ時期を予測するための手法について考察する．

4.2 総開発労力の定式化

OSS の開発に伴う総開発労力を定式化し，総開発労力を最小にする時刻を最適バージョンアップ時刻と定義することにより，バージョンアップ後の信頼性維持や進捗度管理に役立つものと考える．まず，総開発労力を定式化するために，以下のパラメータを定義する．

m_1: フォールト 1 個当りの修正労力 $\quad (m_1 > 0)$,

m_2: フォールト 1 個当りの保守労力 $\quad (m_2 > m_1)$,

m_3: 単位時間当りの保守労力 $\quad (m_3 > m_1)$.

よって，以下のような期待開発労力が得られる．

$$E_1(t) = m_1 \cdot \mu(t). \tag{14}$$

一方，メジャーバージョンアップ後の保守労力は以下のように定式化できる．

$$E_2(t) = m_2\{\mu(t_0) - \mu(t)\} + m_3 t. \tag{15}$$

ここで，t_0 は前回のバージョンアップ期間を表す．

さらに，時間区間 $(0, t_0]$ を越えた場合において，コンポーネントを新規に開発する際には，整合性を確認するためのペナルティ労力が課せられるものとする．本論文では，ペナルティ関数を以下のように定義する．

$$G(t) = (1-c)\exp\left[\frac{t-t_0}{v}\right], \tag{16}$$

ここで，$c(>0)$ は，t_0 を越えて開発されたシステム全体に対する新規コンポーネントの割合，$v(>0)$ は，過去のメジャーバージョンアップ回数を表す．

したがって，総期待開発労力は，式 (14)，式 (15) および式 (16) より，

$$E(t) = E_1(t_i) + E_2(t) + G(t) \tag{17}$$

のように表すことができる．この式 (17) を最小にする時刻 t^* が，OSS の最適メジャーバージョンアップ時刻となる．

5 OSS に対する信頼性評価ツールの開発

本ツールは，オブジェクト指向での分析・設計フェーズにおける表記法である Unified Modeling Language（以下 UML と略す）のアクティビティ図およびシーケンス図を用いて設計し，オブジェクト指向型言語の 1 つである Java を用いて開発した．

5.1 要求仕様定義

本ツールの要求仕様を以下に示す．

1. OSS に対する信頼性評価ツールとして，バグトラッキングシステムから採取された実測データを用いて信頼性評価を行い，各推定結果をグラフ表示する．なお，信頼性評価に必要な全てのデータは，1 つの CSV（Comma Separated Values）形式のファイルに記述されるものとする．
2. システム全体を構成する複数のコンポーネントの重要度を推定するために，ニューラルネットワークを採用する．システム全体に対する信頼性評価のために適用するモデルは，一般化対数型ポアソン実行時間モデルを採用する．
3. 適用モデルに含まれる未知パラメータの推定，実測データに対するモデルの適合性評価，信頼性評価を行う．
4. ニューラルネットワークに基づき各コンポーネントの重要度を推定し，各コンポーネントの重要度を表示する．
5. 実測データと推定値との適合性比較規準としては，予測相対誤差を用いる．
6. ソフトウェア信頼性評価尺度として，総期待発見フォールト数，瞬間フォールト発見率，累積 MTBF を用いる．
7. すべてのグラフは，同時に表示することが可能であるとする．
8. 以上の操作は，GUI によりマウスボタンのクリックにより行う．
9. プログラム記述言語として，オブジェクト指向型言語の 1 つである Java を用いて開発する．
10. 開発および保守労力削減のために，フリーのグラフ生成ライブラリである JFreeChart [6] を使用する．
11. 総期待開発労力をグラフ表示する．推定された総期待開発労力に基づき最適バージョンアップ時刻を導出する．

5.2 信頼性評価手順

以下に，OSS に対するソフトウェア信頼性評価手順を示す．

1. 信頼性評価の対象となる OSS を選択し，バグトラッキングシステムからフォールトデータを採取する．
2. 得られたデータから，ツールで読み込むために必要な CSV 形式のデータファイルを作成する．
3. ニューラルネットワークにより各コンポーネントの重要度を推定する．モデルに含まれる未知パラメータを最尤法により推定する．
4. 信頼性評価尺度として，総期待発見フォールト数，瞬間フォールト発見率，および累積 MTBF を推定し，結果をグラフ表示する．また，予測相対誤差の推定結果をグラフ表示する．
5. 総期待開発労力の推定結果をグラフ表示するとともに，最適バージョンアップ時刻を推定する．

5.3 信頼性評価の実行例

本論文で提案された信頼性評価手法ならびに最適バージョンアップ問題の適用例を示すために，Fedora プロジェクト[1]で開発が進められている Fedora Core Linux の Kernel を取り上げる [14]．本論文では，Fedora Core 7（FC7）のリリース候補版である Test 1 のリリース以降における信頼性評価結果を示す．評価版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きく関係すると同時に，OSS

[1]Fedora は Red Hat Inc. の登録商標である．

表 1： Fedora Core 7 におけるリリース候補版のスケジュール

Date	Event
1 February 2007	Test1 Release
29 February 2007	Test2 Release
27 March 2007	Test3 Release
24 April 2007	Test4 Release
31 May 2007	Fedora 7 General Availability

の保守コストや保守労力の増大に関係することから，リリース候補版の信頼性評価は正式版リリースへ向けての重要な段階となる．FC7 のリリーススケジュールを表 1 に示す．

ソフトウェアの信頼性を評価するために，所定の時間区間内に発見されるフォールト数またはソフトウェア故障発生数のデータに基づいた手法がとられているという歴史的経緯 [10] から，本論文におけるニューラルネットワークの出力データと教師信号には，各コンポーネントに対する累積フォールト発見数データを適用する．これにより，各コンポーネントの相互作用の状態をブラックボックスとして捉え，実際の OSS 開発環境においても適用可能性の高い手法として利用できるものと考える．

まず，ニューラルネットワークにより推定された各 Kernel コンポーネントに対する推定結果および式 (6) におけるモデルパラメータの推定結果を図 1 に示す．Fedora Core を構成するコンポーネント数は小規模なものを合わせると数百と非常に多いことから，本論文ではシステムの主要コンポーネントとして Kernel を取り上げることとする．この結果から，Kernel コンポーネントの信頼性に関する重要度が最大であり，Kernel-module-thinkpad，Kernel-pcmcia-cs，Kenrel-utils，および Kernel-xen-2.6 のコンポーネントの重要度が最低であることが分かる．これは，Kernel コンポーネントの成熟度が高く，Kernel-module-thinkpad，Kernel-pcmcia-cs，Kenrel-utils，および Kernel-xen-2.6 のコンポーネントの成熟度が低いことを意味する．また，推定された式 (6) における累積フォールト発見数の期待値の推定値 $\widehat{\mu}(t)$ を図 2 に示す．さらに，式 (11) の瞬間フォールト発見率の推定値 $\widehat{\mu_d}(t)$，式 (12) における累積 MTBF の推定値 $\widehat{MTBF_C}(t)$ の推定結果を図 3 および図 4 に示す．これらの結果から，時間の経過とともに平均故障発生時間間隔が小さくなり，今後もソフトウェア故障が頻繁に発生することが確認できる．また，予測相対誤差の推定結果を図 5 に示す．図 5 から，実測値と推定値との誤差

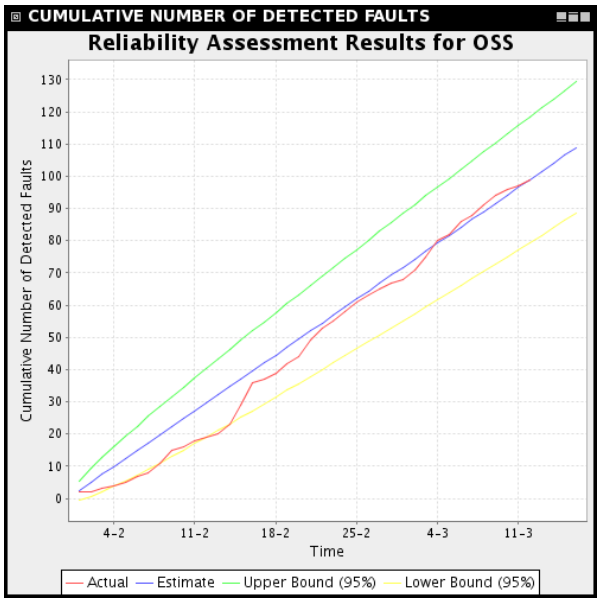

図 2： 推定された累積フォールト発見数の期待値，$\widehat{\mu}(t)$.

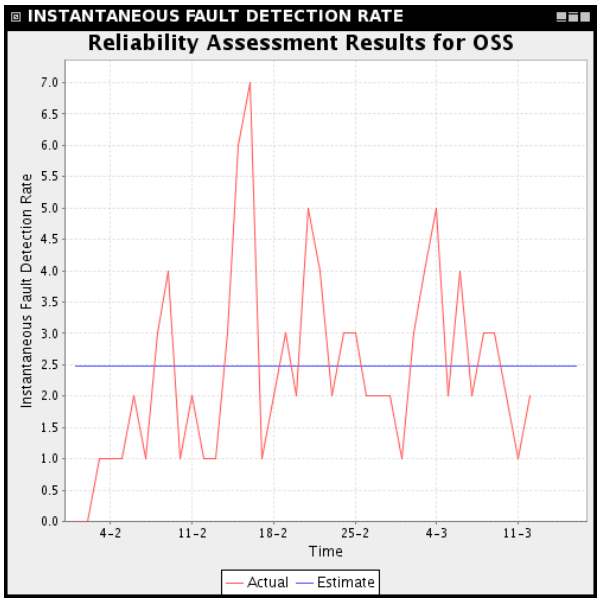

図 3： 推定された瞬間フォールト発見率，$\widehat{\mu_d}(t)$.

は，2 月 16 日以降において安定していることが確認できる．

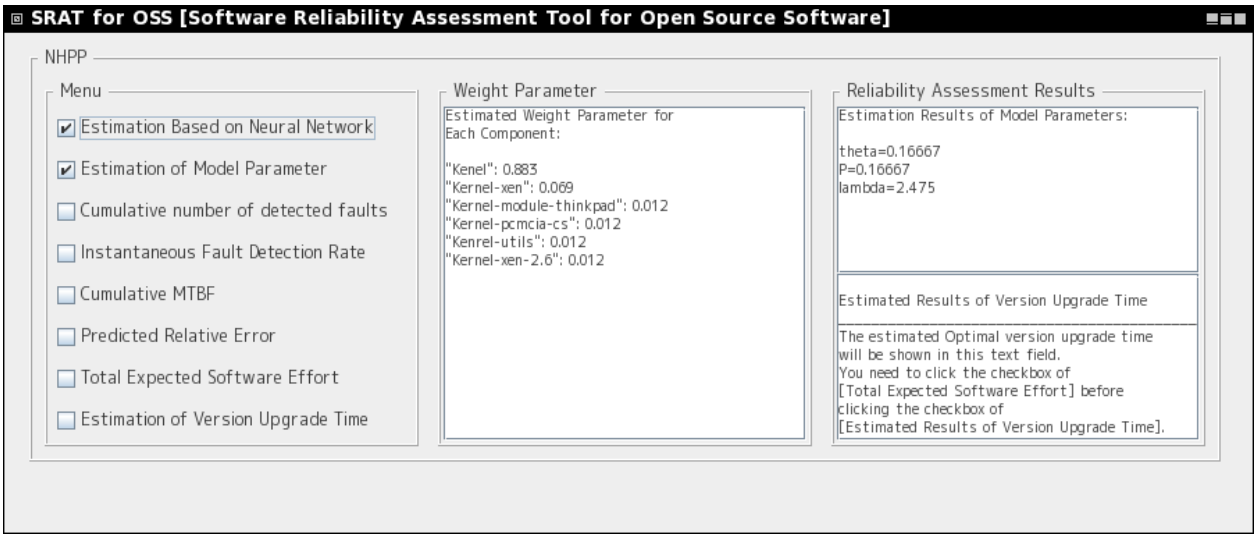

図 1： 推定された各コンポーネントに対する重要度とモデルパラメータの推定結果.

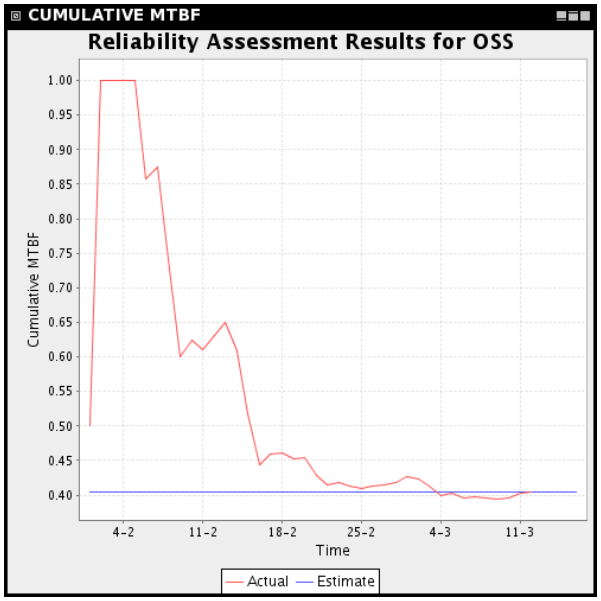

図 4： 推定された累積 MTBF，$\widehat{MTBF}_C(t)$.

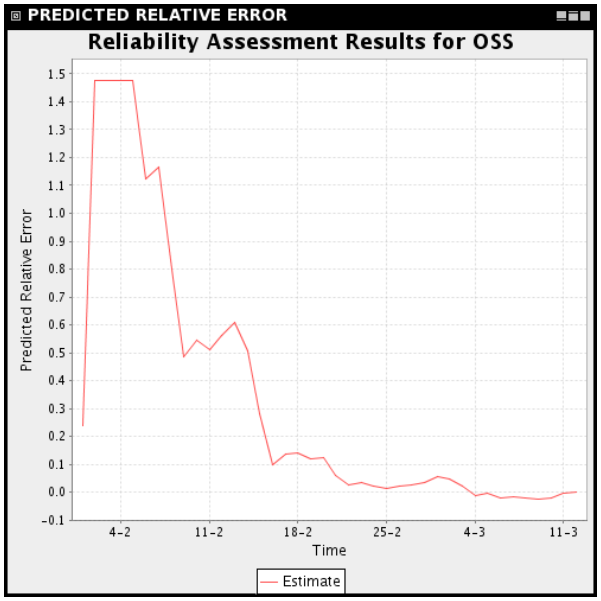

図 5： 推定された予測相対誤差.

5.4 最適バージョンアップ時刻推定の実行例

5.4.1 Fedora Core

まず，最適メジャーバージョンアップ時期推定のツールの実行例として，Fedora Core を取り上げる．ここでは一例として，FC6 までの過去のデータに基づき，FC7 への次期バージョンアップ時刻を推定する．前バージョンである FC6 の評価版リリースから正式リリースまでの期間は 92 日であったことから，t_0 を 92 と仮定し，2007 年 2 月 1 日以降における式 (17) の推定された総期待開発労力を図 6 に示す．さらに，図 6 から推定された最適バージョンアップ時刻の推定結果を図 7 に示す．図 6 および図 7 から，総期待開発労力を最小にする最適バージョンアップ時刻は FC 7 の評価版がリリースされてから

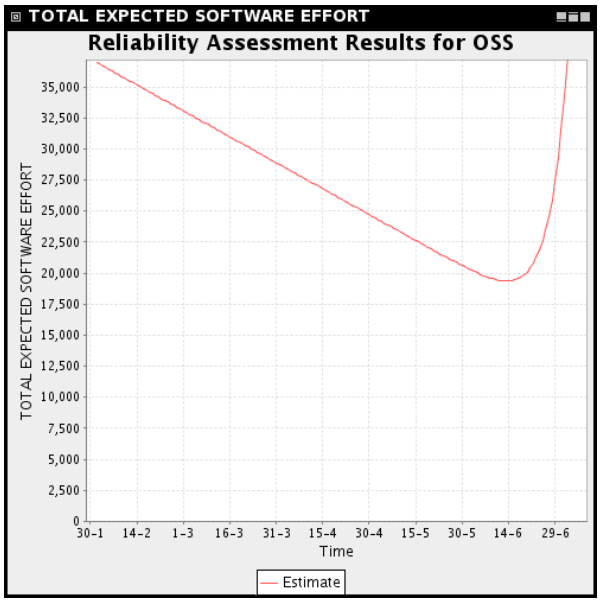

図 6： 推定された総期待開発労力.

表 2： Firefox 2 におけるリリース候補版のスケジュール

Date	Event
29 March 2006	Alpha 1 Release
12 July 2006	Beta 1 Release
26 September 2006	RC 1 Release
6 October 2006	RC 2 Release
24 October 2006	Final Release

6 月 14 日となり，そのときの総期待開発労力は 19359.0 人・日であることが確認できる．

なお，実際の FC7 は，2007 年 2 月 1 日に評価版がリリースされ，120 日後の 5 月 31 日に正式版がリリースされている．このことから，総期待開発労力を最小にするという観点から考えた場合，2 週間早い正式版リリースであったことが読み取れる．

5.4.2 Firefox

同様に，最適メジャーバージョンアップ時期推定の数値例として，Web ブラウザの Firefox [15] を取り上げる．ここでは一例として，Firefox 1.5 までの過去のデータに基づき，Firefox 2 への次期バージョンアップ時刻を推定する．Firefox 2 のリリーススケジュールを表 2 に示す．

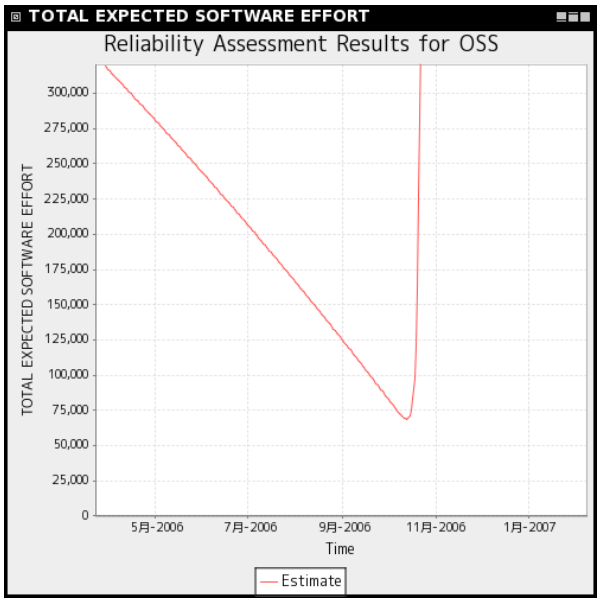

図 8： 推定された総期待開発労力.

Firefox 2 の Alpha 1 がリリースされた 2006 年 3 月 29 日以降における式 (17) の推定された総期待開発労力を図 8 に示す．さらに，図 8 から推定された最適バージョンアップ時刻の推定結果を図 9 に示す．図 8 および図 9 から，総期待開発労力を最小にする最適バージョンアップ時刻は Firefox の Alpha 1 リリース後の 10 月 14 日となり，そのときの総期待開発労力は 68905.1 人・日であることが確認できる．

なお，実際の Firefox 2 は，2006 年 10 月 24 日に正式版がリリースされている．このことから，総期待開発労力を最小にするという観点から考えた場合，10 日遅い正式版リリースであったことが読み取れる．

6 おわりに

オープンソースという開発スタイルは，今後，何らかの形で市場で大きな流れを作っていくものと考えられる [1]．この流れを阻害する大きな要因として，サポートや品質上の問題が挙げられる．

本論文では，こうした問題を解決するために，OSS に対する信頼性評価法を提案するとともに，本手法の数理モデルとしての中身を理解していなくとも容易に使用できるように OSS に対する信頼性評価ツールとして実装した．特に，ニューラルネットワークと SRGM を融合することにより，各コンポーネント間の相互作用を包括した信頼性評価法について議論した．また，実際にバグ

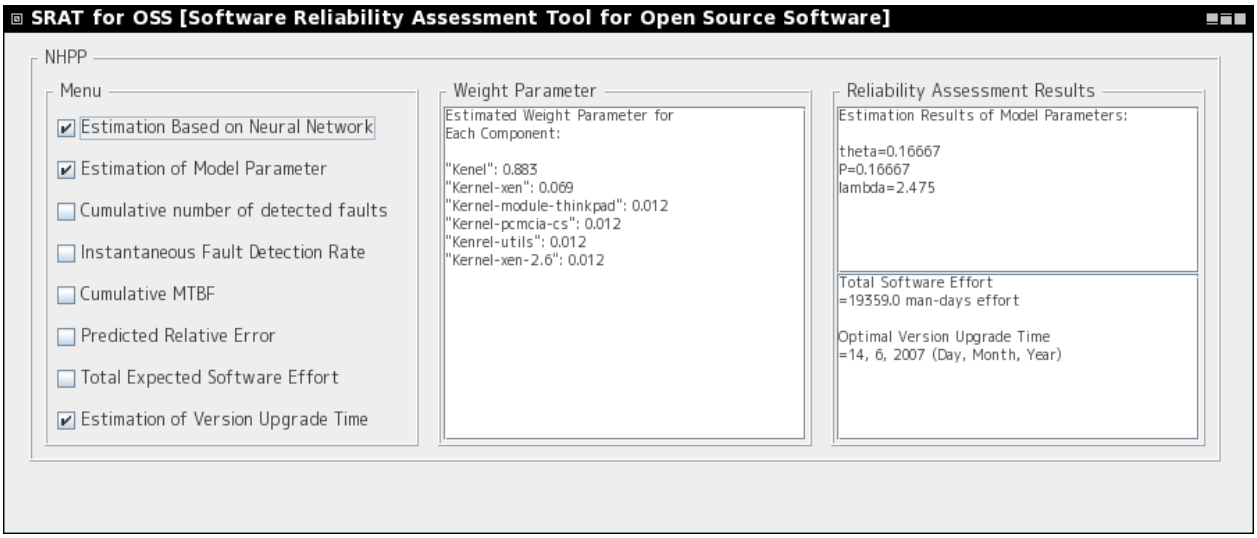

図 7： 最適バージョンアップ時刻および総期待開発労力の推定結果（FC7）.

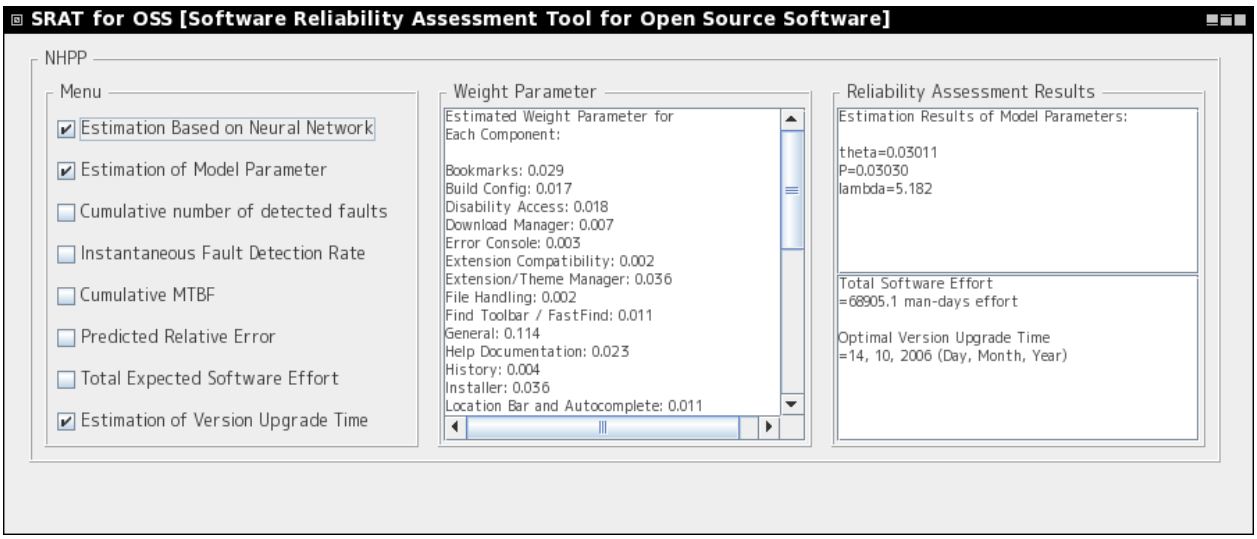

図 9： 最適バージョンアップ時刻および総期待開発労力の推定結果（Firefox 2）.

トラッキングシステムから採取されたフォールト発見数データに対する本ツールの実行例を示した.

さらに，OSS の開発において，ある程度目安となるような最適バージョンアップ時期の推定方法について議論するとともに，総期待開発労力を定式化し，既存の信頼性評価ツールに対して最適バージョンアップ時期推定機能として組み込んだ．本論文において提案されたツールによって，OSS のバージョンアップ後の信頼性維持や進捗度管理に役立つものと考える.

OSS の開発は世界中に分散する誰もが開発に参加できる環境である一方，その信頼性向上に関する取り組みは，フォールト報告に基づいて修正作業を行うのみといったのが現状である．OSS では信頼性を動的かつ定量的に評価するという試みが行われていなかったことから，本論文において新たに提案された信頼性評価法を OSS に適用することによって，信頼性に関する評価指標を提示することが可能となるとともに，より高品質な OSS の開発に結びつくものと考える．特に，ある程度目安となる

ような適切なバージョンアップ時期を推定することは，リリース後の信頼性維持，総開発労力の削減，高信頼性に伴う OSS の人気度の向上などにつながるものと考えられる．

謝辞

本研究の一部は，文部科学省科学研究費基盤研究 (C)（課題番号 18510124），若手研究 (B)（課題番号 17700039），および財団法人サタケ技術振興財団の奨学寄付金の援助を受けたことを付記する．

参考文献

[1] ソフトウェア情報センター研究会報告書, オープンソースソフトウェアの利用状況調査／導入検討ガイドラインの公表について, 東京, 2004.

[2] A. MacCormack, J. Rusnak, and C.Y. Baldwin, “Exploring the structure of complex software designs: An empirical study of open source and proprietary code,” *Management Science*, vol. 52, no. 7, 2006, pp. 1015–1030.

[3] G. Kuk, “Strategic interaction and knowledge sharing in the KDE developer mailing list,” *Management Science*, vol. 52, no. 7, 2006, pp. 1031–1042.

[4] Y. Zhoum, J. Davis, “Open source software reliability model: an empirical approach,” *Proceedings of the workshop on Open Source Software Engineering (WOSSE)*, vol. 30, no. 4, 2005, pp. 67–72.

[5] P. Li, M. Shaw, J. Herbsleb, B. Ray and P. Santhanam, “Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems,” *Proceedings of 12th International Symposium on the Foundations of Software Engineering (FSE-12)*, 2004, pp. 263–272.

[6] JFreeChart, http://www.jfree.org/jfreechart/

[7] Y. Tamura, K. Hadatsuki, S. Yamada, and M. Kimura, “Development of a user-oriented reliability assessment tool for open source software (in Japanese),” *Proceedings of the Linux Conference 2006*, vol. 4. [Online]. Available: http://lc.linux.or.jp/lc2006/

[8] 田村慶信, 山田茂, 木村光宏, “オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察,” 電子情報通信学会論文誌，vol. J88–A，no. 7，pp. 840–847, 2005.

[9] E. D. Karnin, “A simple procedure for pruning back-propagation trained neural networks,” *IEEE Trans. Neural Networks.*, vol. 1, pp. 239–242, 1990.

[10] 山田 茂, ソフトウェア信頼性モデル—基礎と応用—, 日科技連出版社, 東京, 1994.

[11] Y. Tamura and S. Yamada, “A Method of User-oriented Reliability Assessment for Open Source Software and Its Applications,” *Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, Oct. 8–11, 2006, pp. 2185–2190.

[12] S. Yamada, H. Narihira, and S. Osaki, “Optimal release policies for a software system with a scheduled software delivery time,” *Intern. J, Systems Science*, vol. 15, no. 8, Aug. 1984, pp. 905–914.

[13] S. Yamada and S. Osaki, “Optimal software release policies with simultaneous cost and reliability requirements,” *European J. Operational Research*, vol. 31, no. 1, July 1987, pp. 46–51.

[14] Fedora Project, sponsored by Red Hat. [Online]. Available: http://fedora.redhat.com/

[15] Mozilla.org, Mozilla Foundation. [Online]. Available: http://www.mozilla.org/