数理解析研究所講究録
第1912巻 2014年 44-52



# オープンソースソリューションの最適メンテナンス時刻推定のためのAIRアプリケーションの開発

山口大学大学院・理工学研究科 足立 翔人 (Shoto Adachi) †
山口大学大学院・理工学研究科 田村 慶信 (Yoshinobu Tamura) †
†Graduate School of Science and Engineering, Yamaguchi University
鳥取大学大学院・工学研究科 山田 茂 (Shigeru Yamada) ‡
‡Graduate School of Engineering, Tottori University

## 1 はじめに

OS, サーバ, およびアプリケーションレベルまで, 様々な種類のオープンソースソフトウェア (open source software, 以下 OSS と略す) が世界中で開発・公開されている. また, 企業主体の下で開発され, 運用・保守までをサービスとして提供するビジネスモデルも普及している. このように, 低コスト, 保守・運用が容易といった観点から, OSS を利用したソフトウェアサービスの提供に注目が集まっている. しかしながら, ソフトウェアの設計図にあたるソースコードが世界中に公開されているため, 悪意のあるサイト攻撃や情報流出の標的になり易く, なかなか導入に踏み切れないという問題も抱えている.

特に, 最近では, この OSS を利用したソフトウェア開発事例として, 複数のオープンソースコンポーネントを結合したオープンソースソリューションの利用が拡大している. オープンソースソリューションの一例として, OSS である Apache HTTP Server [1], Apache Tomcat [2], PostgreSQL [3] といったサーバソフトウェアを組み合わせ, その基盤上に Java 言語で開発するようなシステムが挙げられる. このようなシステムの場合, 各オープンソースコンポーネントの相互運用性の確保が重要となる. 一方, オープンソースソリューションにはいくつかの問題点もある. 1つは, オープンソースコンポーネントの規模が比較的大きいために, システム全体としても大規模化する傾向である. また, 複数の OSS から構成されているため, 品質の面で全般的に問題となることが多く, 1つの主要コンポーネントが全体のオープンソースソリューションとしての機能自体を崩壊させる危険性も含んでいることである.

従来から, ソフトウェア製品の開発プロセスにおけるテスト進捗管理や出荷品質の把握のための信頼性評価を行うアプローチとして, 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. その代表的なモデルの1つとして, ソフトウェア信頼度成長モデルがある [4].

本論文では, こうした大規模オープンソースソリューション開発におけるテスト工程を対象とした確率微分方程式モデルを構築する. また, オープンソースソリューションに対する最適メンテナンス問題について議論する. さらに, 実際の OSS のソフトウェアフォールト発見数データに対する数値例を示すことにより, 大規模オープンソースソリューションの信頼性評価法について議論するとともに, 本手法の適用可能性について考察する. これにより, 大規模オープンソースソリューションに対して, 信頼性評価に関する何らかの評価指標を与えることができるものと考える. 特に, 提案された信頼性評価手法を数理モデルに関する知識が無くとも利用できるように, Flex を用いて信頼性評価ツールを設計・開発し, 実際のフォールトデータに対するツールの実行例を示すとともに, ツールの適用可能性について議論する.

## 2 確率微分方程式モデル

まず，時刻 $t$=0 でオープンソースソリューションのテスト工程が開始され，任意の時刻 $t$ におけるソフトウェア内の発見フォールト数 $\{N(t), t \geq 0\}$ は以下の常微分方程式によって記述されるものと仮定する．

$$\frac{dN(t)}{dt} = b(t)\{a - N(t)\}. \tag{1}$$

ここで，$b(t)(> 0)$ は時刻 $t$ におけるフォールト発見率を，$a$ はオープンソースソリューションに潜在する総フォールト数を表す．

一般的に，オープンソースソリューションのテスト工程の初期段階においては，各オープンソースコンポーネント間の結合状態や整合性が不安定であり，オープンソースソリューション全体としての機能が十分に発揮できない状況が想定される．このように，大規模オープンソースソリューションの特徴を考えた場合，そのフォールト発見事象は，テスト工程の初期段階において不規則な状態となり，時間の経過とともに安定していくと考えられる．こうした不規則性を，標準化された Gauss 型白色雑音によって近似的に表現する．また，Wiener 過程の分散の性質から，時間の経過とともに Gauss 型白色雑音が増大していく傾向があるが，本論文では，オープンソースソリューションの特徴を考慮するために，Gauss 型白色雑音の変化量を表す $\mu(t)$ を導入する．

上記のことから，ソフトウェア故障強度 $b(t)$ に不規則性を導入すると，式 (1) は，

$$\frac{dN(t)}{dt} = \{b(t) + \sigma\gamma(t)\mu(t)\}\{a - N(t)\}, \tag{2}$$

となる．ここで，$\sigma(> 0)$ は定数パラメータであり，$\gamma(t)$ は解過程の Markov 性を保証するために標準化された Gauss 型白色雑音である．さらに，$\mu(t)$ はオープンソースソリューション全体の安定性を示す成長関数を表す．式 (2) を，以下の Itô 型の確率微分方程式 [5,6] に拡張して考える．

$$dN(t) = \left\{b(t) - \frac{1}{2}\sigma^2\mu(t)^2\right\}\{a - N(t)\}dt + \sigma\mu(t)\{a - N(t)\}d\omega(t). \tag{3}$$

式 (3) の確率微分方程式を初期条件 $N(0) = 0$ の下で Itô の公式を用いて変換すると，

$$N(t) = a\left\{1 - \exp\left(-\int_0^t b(s)ds - \sigma\mu(t)\omega(t)\right)\right\}, \tag{4}$$

となる．

本論文では，ソフトウェア故障強度関数 $b(t) \equiv b_1(t)$，$b(t) \equiv b_2(t)$ およびオープンソースソリューション全体の安定性を示す成長関数 $\mu(t)$ は，次式を満たすものとする．

$$\int_0^t b_1(s)ds = (1 - \exp[-\alpha t]), \tag{5}$$

$$\int_0^t b_2(s)ds = \{1 - (1 + \alpha t)\exp[-\alpha t]\}, \tag{6}$$

$$\mu(t) = \exp[-\beta t]. \tag{7}$$

ここで，$\alpha$ はソフトウェア故障強度の加速係数を，$\beta$ は安定性に関する成長係数を表す．

式 (4) で定義された $\omega(t)$ の性質は，次の通りである．

(1) $\omega(t)$ は Gauss 過程である．

(2) $\omega(t)$ の平均および分散は，それぞれ

$$\mathrm{E}[\omega(t)] = 0, \mathrm{Var}[\omega(t)] = \sigma^2 t, \tag{8}$$

により与えられる．

(3) $\omega(t)$ は定常独立増分をもつ．

(4) $\Pr[\omega(0)=0]=1$.

## 3 ソフトウェア信頼性評価尺度

任意の時刻 $t$ における発見フォールト数の期待値 $\mathrm{E}[N(t)]$ および $\mathrm{Var}[N(t)]$ は，ソフトウェア信頼性を評価する上で重要な尺度となる．これらは，Wiener 過程 $\omega(t)$ の密度関数，

$$f(\omega(t)) = \frac{1}{\sqrt{2\pi t}} \exp\left\{-\frac{\omega(t)^2}{2t}\right\}, \tag{9}$$

より，提案モデルを用いた場合の，テスト時刻 $t$ までに発見される総フォールト数の期待値は，式 (4) より，

$$\mathrm{E}[N(t)] = a\left\{1 - \exp\left(-\int_0^t b(s)ds + \frac{\sigma^2\mu(t)^2}{2}t\right)\right\}, \tag{10}$$

となる．同様に，テスト時刻 $t$ までに発見される総フォールト数の分散は，

$$\mathrm{Var}[N(t)] = \mathrm{E}[\{N(t) - \mathrm{E}[N(t)]\}^2], \tag{11}$$

となる．

また，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．テスト時刻 $t$ における瞬間 MTBF(instantaneous MTBF: $\mathrm{MTBF_I}$) および累積 MTBF(cumulative MTBF: $\mathrm{MTBF_C}$) は，以下のように導出できる [7]．本論文では，任意の時刻 $t$ における瞬間的なフォールト発見間隔の平均を意味する瞬間 MTBF を計算の簡単化のために，

$$MTBF_I(t) = \frac{dt}{\mathrm{E}[dN(t)]}, \tag{12}$$

で近似的に計算する．瞬間 MTBF と同様に，本論文では，テスト開始時点から考えたときの発見フォールト 1 個当たりに要する発見時間の平均を意味する累積 MTBF を簡単化のために，

$$MTBF_C(t) = \frac{t}{\mathrm{E}[N(t)]}, \tag{13}$$

と定義する．

さらに，残存フォールト数の期待値もソフトウェア信頼性を評価する上で重要な尺度である．テスト工程の任意の時刻 $t$ におけるソフトウェア内の残存フォールト数の期待値 $\mathrm{E}[M(t)]$ は，

$$\mathrm{E}[M(t)] \quad = \quad \mathrm{E}[N(\infty) - N(t)], \tag{14}$$

により表すことができる．

不完全デバッグ率は，オープンソースソリューションの安定性に関する時間変化を把握するのに有益な尺度である．また，不完全デバッグ率が小さな値を取ることは，それだけオープンソースソリューションが安定しており，信頼度が向上したと判断できることになる．テスト時刻 $t$ における不完全デバッグ率は，以下のように導出できる．

$$\Pr[N(t+\Delta t) \le x | N(t) = x] = \Phi\left[\frac{\log\dfrac{\displaystyle\int_0^t b(s)ds}{\displaystyle\int_0^{t+\Delta t} b(s)ds}}{\sigma\mu(t+\Delta t)\sqrt{\Delta t}}\right]. \tag{15}$$

ここで，$x$ は時刻 $t$ における累積フォールト発見数であり，微小時間区間 $\Delta t(t>0)$ において，$x$ を超過しない確率として定義する．また，$\Phi$ は標準正規分布関数を表し，

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \exp\left(-\frac{y^2}{2}\right)dy, \tag{16}$$

で定義されるものである．

## 4 最適メンテナンス問題

オープンソースソリューションを運用する際における総ソフトウェアコストを定式化するために，以下のパラメータを定義する [8,9].

$c_1$ : 単位時間当りの運用コスト ($c_1$>0),
$c_2$ : フォールト 1 個当りの修正コスト ($c_2$>0),
$c_3$ : 運用段階におけるフォールト 1 個当りの保守コスト ($c_3$>$c_2$).

ここで，$c_2$ はバグトラッキングシステム上に登録されたフォールトを対象とし，$c_3$ は実際のオープンソースソリューションの運用環境に起因するフォールトを対象とする．このとき，総ソフトウェアコストを以下のように定義する．

$$C(t) = c_1 t + c_2 \mathrm{E}[N(t)] + c_3\{a - \mathrm{E}[N(t)]\}. \tag{17}$$

式 (17) を最小にする時刻 $t^*$ が，オープンソースソリューションの最適メンテナンス時刻となる．

## 5 AIR アプリケーションの開発

### 5.1 要求仕様定義

本ツールの要求仕様を以下に示す．

1. 本ツールの信頼性評価に使用するデータは，実際のオープンソースソリューション開発のテスト工程から採取された実測データを用いる．
2. 実測データに基づいて信頼性評価を行い，各推定結果をグラフで表示する．
3. 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確率微分方程式モデルを用いる．さらに，提案された確率微分方程式モデルに含まれる未知パラメータには最尤法を適用する．
4. 信頼性評価尺度として，累積 MTBF，不完全デバッグ率，残存フォールト数などを適用する．
5. 総ソフトウェアコストをグラフ表示するとともに，最適メンテナンス時刻を推定する．
6. 推定結果をグラフ表示する．
7. ツールの操作には GUI を使用し，マウスを用いて行う．
8. ツールの開発言語に Flex [10] を使用する．

### 5.2 実行手順

本ツールを用いたオープンソースソリューションに対する信頼性評価ツールの実行手順を以下に示す．

1. オープンソースソリューション開発のテスト工程から得られた累積フォールト発見数データを採取する．
2. フォールトデータと未知パラメータの初期値を CSV フォーマットで入力する．
3. 最尤法により未知パラメータを推定する．
4. 種々の信頼性評価尺度をグラフ表示する．
5. 総ソフトウェアコストをグラフ表示する．

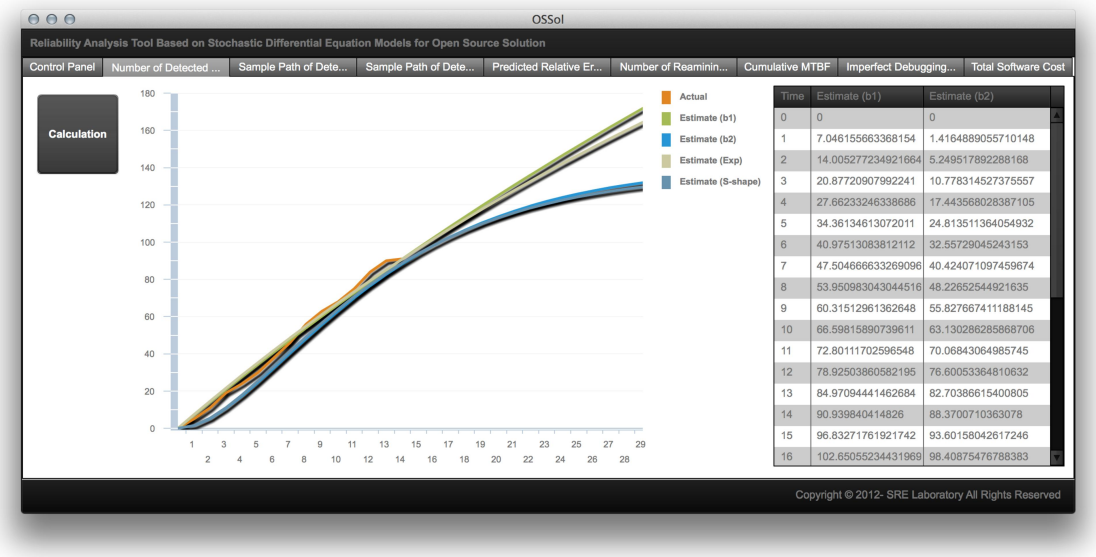


図 1： 推定された累積フォールト発見数の期待値.

## 5.3 ツールの実行例

本論文では，Apache HTTP Server，Apache Tomcat，MySQL を組み合わせ，JSP（Java Server Pages）技術を利用した大規模オープンソースソリューションを構築する場合を考える．一例として，大規模オープンソースソリューションのテスト工程を想定するために，実際の Apache HTTP Server，Apache Tomcat，MySQL のオープンソースプロジェクトにおけるバグトラッキングシステム上に登録されたフォールトデータを適用した数値例を示す．

累積フォールト発見数の期待値の推定結果を図 1 に示す．また，推定された既存の確率微分方程式モデルに対するフォールト発見数のサンプルパスを図 2 に示す．さらに，推定された提案モデルに対するフォールト発見数のサンプルパスを図 3 に示す．図 3 から，テスト工程初期段階において不規則性が大きい様子が確認できる．次に，予測相対誤差の推定結果を図 4 に示す．ここで，横軸は進捗率を，縦軸は予測相対誤差を表す．図 4 から，テスト工程開始時点以降において予測相対誤差が小さくなり，オープンソースソリューションが安定している様子が確認できる．また，不完全デバッグ率の推定結果を図 5 に示す．図 5 から，信頼度成長が起こるとともにデバッグ作業が安定している様子が確認できる．

最適メンテナンス時刻の推定結果を図 6 に示す．図 6 から，$b_2(t)$ の場合において，運用開始時点から約 30 週目においてメンテナンスを行えば良いことが確認できる．

また，実測データに対するモデルの適合性を評価するために，赤池情報量規準 (Akaike's Information Criterion, 以下 AIC と略す) を適用する．提案モデルと既存モデルとの実測データに対する適合性評価結果を表 1 に示す．AIC は以下により与えられる．

$$\mathrm{AIC} = -2 \cdot LLF + 2 \cdot N. \tag{18}$$

ここで，$LLF$ は最大対数尤度，$N$ は自由パラメータ数を表す．表 1 から，提案モデルの適合性が良いことが確認できる．

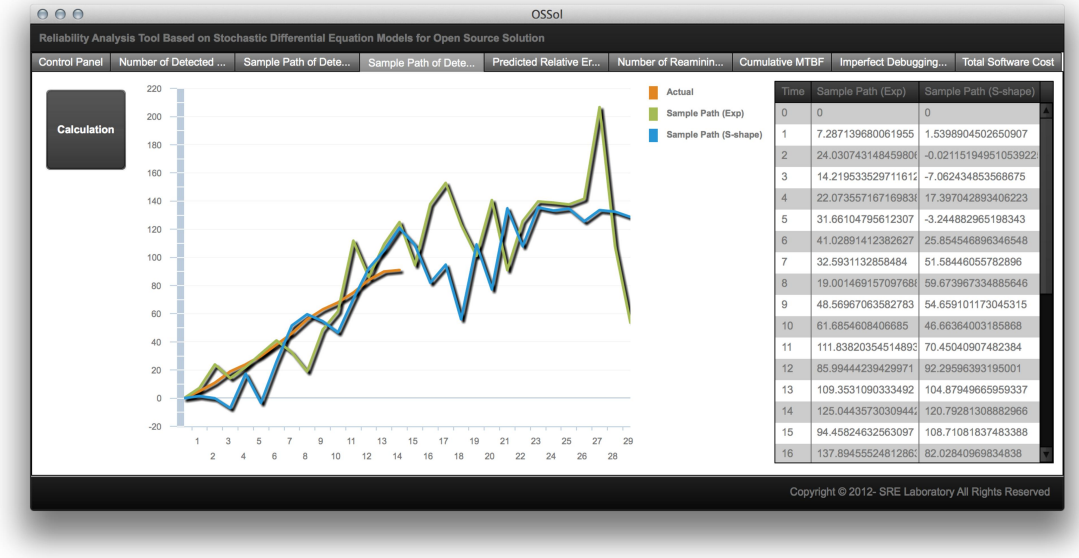


図 2： 推定された既存の確率微分方程式モデルに対するフォールト発見数のサンプルパス．

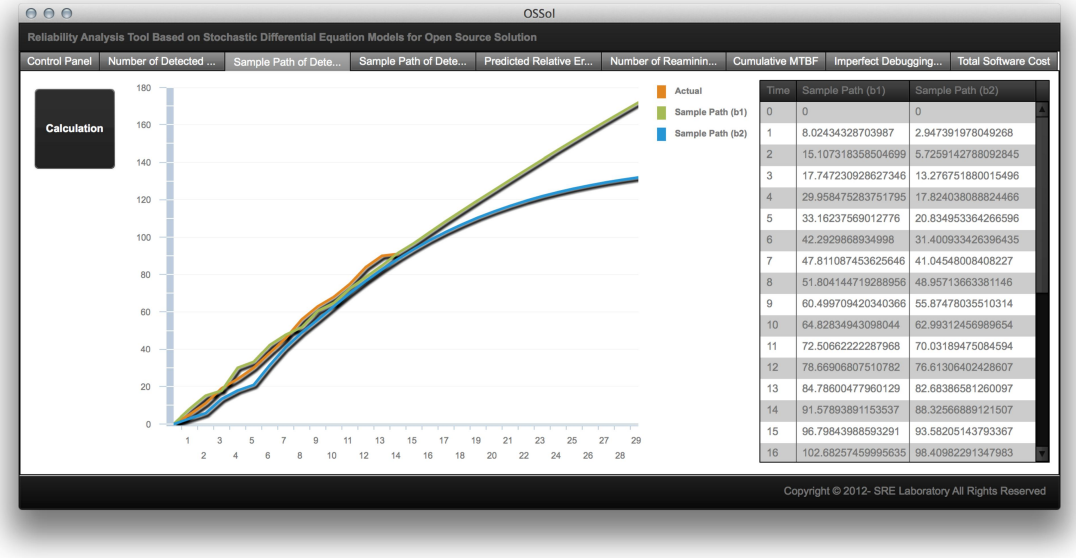


図 3： 推定された提案モデルに対するフォールト発見数のサンプルパス．

## 6　おわりに

本論文では，大規模オープンソースソリューションに対する確率微分方程式モデルについて議論するとともに，提案モデルに基づく信頼性評価ツールを開発した．また，本ツールの実行例として，フォールトデータに対する信

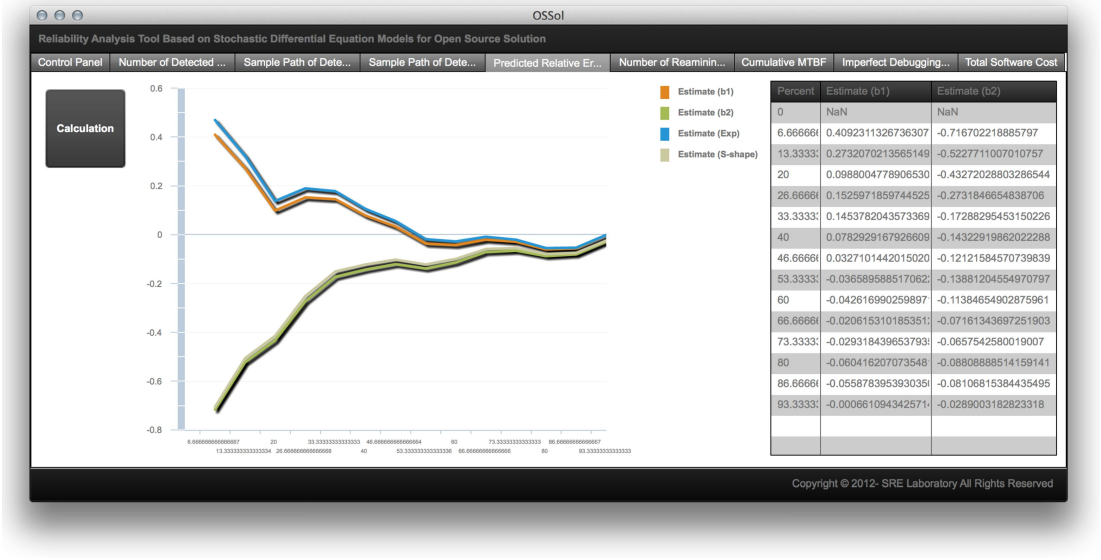


図 4： 推定された予測相対誤差.

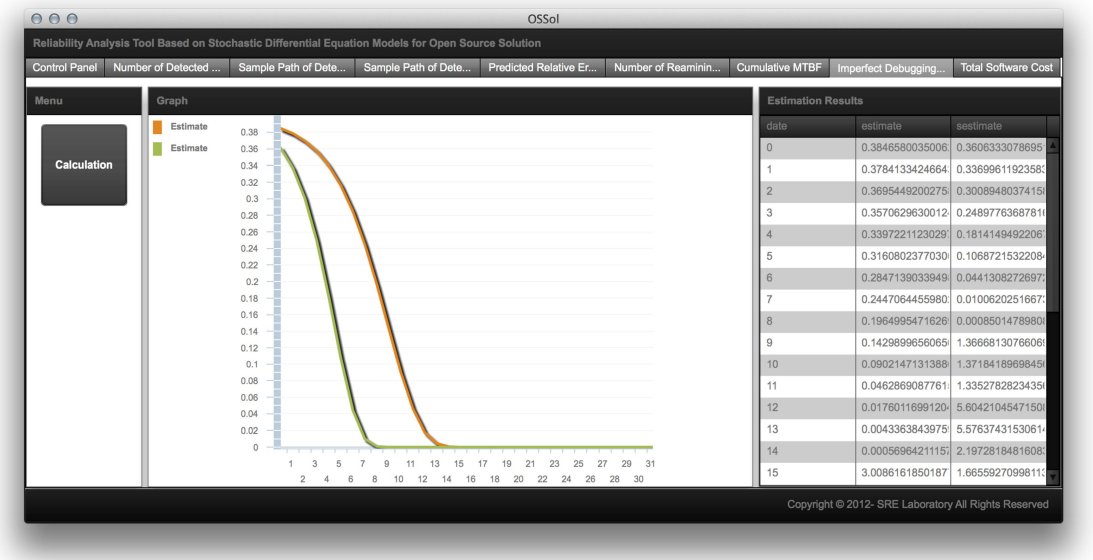


図 5： 推定された不完全デバッグ率.

頼性評価結果とともに最適メンテナンス時刻の推定例を示した．本ツールにより，大規模オープンソースソリューションに対して，品質上における何らかの指標を得ることができるものと考える．

オープンソースソリューションでは，各オープンソースコンポーネントの相互運用性確保が重要となる．また，

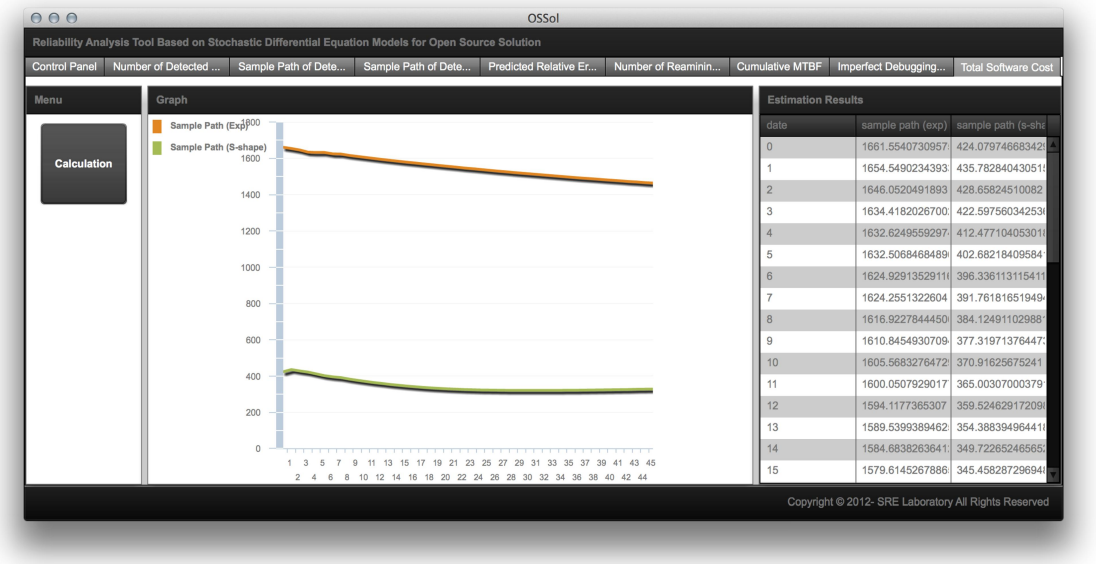


図 6： 推定された最適メンテナンス時刻.

表 1： AIC による適合性比較結果.

| Compared models | AIC |
|---|---|
| Our exponential SDE model | 79.57425 |
| Our S-shaped SDE model | 80.7033 |
| Conventional exponential SDE model | 79.95469 |
| Conventional S-shaped SDE model | 82.25969 |

大規模オープンソースソリューションが急速に普及し始めている現在，信頼性に関する指標を提示することが重要であると考える．本論文で提案した信頼性評価手法を適用することにより，より高品質な大規模オープンソースソリューションの開発に結びつくものと考える．近年，コスト削減，短納期といった観点から複数の OSS を組み合わせて，1 つの大規模オープンソースソリューションを開発する事例が増加している．特に，オープンソースソリューションは，複数の OSS を組み合わせているために品質の面で問題となることが多い．本論文では，こうした問題を解決するためにオープンソースプロジェクトの下で開発された複数の OSS を利用した大規模オープンソースソリューションに対する信頼性評価ツールを開発し，その実行例を示した．さらに，開発されたツールの性能評価および信頼性評価例を示した．本論文の数値例で取り上げた Apache HTTP Server，Apache Tomcat，MySQL は，大規模オープンソースソリューションにおいて構成される主要オープンソースコンポーネントとして近年注目されている．今後，オープンソースコンポーネントを利用した開発およびサービス形態は急速に発展するものと考えられることから，本論文において開発されたソフトウェアツールは，こうした大規模オープンソースソリューションに対する信頼性評価法として利用できるものと考える．

## 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 24500066 および 25350445）の援助を受けたことを付記する．

## 参考文献

[1] The Apache HTTP Server Project, The Apache Software Foundation, http://httpd.apache.org/

[2] Apache Tomcat, The Apache Software Foundation, http://tomcat.apache.org/

[3] PostgreSQL, PostgreSQL Global Development Group, http://www.postgresql.org/

[4] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Springer–Verlag, Tokyo/Berlin, 2013.

[5] L. Arnold, *Stochastic Differential Equations–Theory and Applications*, John Wiley & Sons, New York, 1974.

[6] E. Wong, *Stochastic Processes in Information and Systems*, McGraw–Hill, New York, 1971.

[7] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, "Software reliability measurement and assessment with stochastic differential equations," *IEICE Transactions on Fundamentals*, vol. E77-A, no. 1, pp. 109-116, Jan. 1994.

[8] S. Yamada and S. Osaki, "Cost-reliability optimal software release policies for software systems," *IEEE Transactions on Reliability*, vol. R-34, no. 5, pp. 422–424, 1985.

[9] S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements," *European Journal of Operational Research*, vol. 31, no. 1, pp. 46–51, 1987.

[10] Flex.org–Adobe Flex Developer Resource, Adobe Systems Incorporated, http://flex.org/