死滅過程モデルのソフトウェア信頼性評価への応用

YASUNOBU, Nayuko / 安信, 那有子

(発行年 / Year)
2012-03-24
(学位授与年月日 / Date of Granted)
2012-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2011 年度　修士論文

死滅過程モデルの
ソフトウェア信頼性評価への応用

Non-homogeneous Death Process Modeling
for Software Reliability Analysis

法政大学大学院工学研究科
システム工学専攻修士課程

安信　那有子
Nayuko　YASUNOBU

指導教員　木村光宏　教授

概要

本研究では，ソフトウェア開発におけるテスト工程の累積フォールト発見数の推移を死滅過程モデルに適用する進捗度評価手法について示す．本手法は，ソフトウェア内に潜在する総フォールト数の期待値を新たに構築したモデルによって導出した上で死滅過程モデルに適用し，全ての潜在するフォールトを発見・修正する時刻を推定することで構成される．その数値例として，本研究で採用している 2 チームによる 2 段階ソフトウェアテスト方法で得た実際の観測データに対して本評価手法を適用した結果を示す．適用結果から，本手法がテスト工程の定量的な進捗度管理とソフトウェア信頼度評価に対して有効であることを示す．

Abstract

This study describes a methodology for estimating degree of testing progress in a software development process by applying software fault-detection data as a time-series which were obtained from a testing activity to a death process model. The methodology is constructed by estimating eradication time of residual faults after applying the number of initial fault content in the software to the death process model. As a trial, the methodology was applied to the result of data analysis from the actual software development which employs somewhat uncommon testing procedure. The procedure can be called as a two-stage testing by two teams. The result demonstrates that the proposed methodology is effective for both of quantitative estimation of the degree of testing progress and software reliability evaluation.

目 次

概要 **1**

Abstract **2**

1 序論 **1**
1.1 研究背景 1
1.2 研究目的 4
1.3 本論文の構成 4

2 ソフトウェア開発のテスト工程における問題点とその解決のためのアプローチ **5**

3 2チームによる2段階ソフトウェアテスト **7**

4 静的信頼性評価モデル **9**
4.1 数値計算によるソフトウェア内に潜在する総フォールト数の推定方法 . . . 11
4.2 適用例 12

5 動的信頼性評価モデル **14**
5.1 死滅過程モデル 14
5.2 ソフトウェア信頼性評価への適用方法 16
5.2.1 平均フォールト発見数 16
5.2.2 平均全フォールト発見時刻 18
5.3 最小二乗法による未知パラメータの推定方法 18

6 ソフトウェア信頼性評価への適用例 **19**
6.1 ソフトウェア内に潜在する総フォールト数の推定結果 21
6.2 未知パラメータ a, b, c, d および平均全フォールト発見時刻の推定結果 . . . 22
6.3 平均フォールト発見数の期待値の推移 23

7 考察 **25**
7.1 テスト経過に伴う期待残存フォールト数の推移 25
7.2 全フォールト発見時刻の期待値と実際の出荷日との比較 26
7.3 全フォールト発見時刻の 90 %信頼区間の推移 27
7.4 テスト経過に伴う残り必要テスト単位時間の推移 27

8 結論 **30**

謝辞 **31**

参考文献 **32**

付録 A　平均全フォールト発見時刻の期待値の推移 **付録 A-1**

表 目 次

1 変数によって表すことができる関数式. 11
2 データセット. 13
3 全てのデータセット. 20
4 回帰テスト二段階目の各テスト時間における N の推定値. 21
5 未知パラメータ a, b, c および c の推定値. 22
6 未知パラメータの推定値（テスト時間 $t_{29} = 29$ のとき）. 23
7 ソフトウェア内に潜在する総フォールト数の期待値と累積フォールト発見数. 26
8 平均全フォールト発見時刻までの必要テスト単位時間. 29

図 目 次

1 2チームによる2段階ソフトウェアテストの流れ. 7
2 2チームによる2段階ソフトウェアテストの詳細. 9
3 状態遷移図. 15
4 データセットを打点したグラフ. 19
5 平均フォールト発見数の期待値の推移（テスト時間 $t_{29} = 29$ のとき). . . . 23
6 フォールト発見数の密度関数の概形（テスト時間 $t_{29} = 29$ のとき）. . . . 24
7 フォールト発見数の分布関数の概形とその中央値（テスト時間 $t_{29} = 29$ のとき）. 24
8 ソフトウェア内に潜在する総フォールト数の期待値と累積フォールト発見数の推移. 25
9 テスト時間毎の平均全フォールト発見時刻とその中央値の推移. 26
10 テスト時間毎の全フォールト発見時刻の中央値とその90 %信頼区間の推移. 28
11 テスト時間毎の平均全フォールト発見時刻までの必要テスト単位時間の推移. 29
A.1 平均フォールト発見数の期待値の推移（テスト時間 $t_{21} = 21$ のとき）. 付録 A-1
A.2 平均フォールト発見数の期待値の推移（テスト時間 $t_{22} = 22$ のとき）. 付録 A-1
A.3 平均フォールト発見数の期待値の推移（テスト時間 $t_{23} = 23$ のとき）. 付録 A-2
A.4 平均フォールト発見数の期待値の推移（テスト時間 $t_{24} = 24$ のとき）. 付録 A-2
A.5 平均フォールト発見数の期待値の推移（テスト時間 $t_{25} = 25$ のとき）. 付録 A-3
A.6 平均フォールト発見数の期待値の推移（テスト時間 $t_{26} = 26$ のとき）. 付録 A-3
A.7 平均フォールト発見数の期待値の推移（テスト時間 $t_{27} = 27$ のとき）. 付録 A-4
A.8 平均フォールト発見数の期待値の推移（テスト時間 $t_{28} = 28$ のとき）. 付録 A-4

1　序論

1.1　研究背景

現在のような高度情報化社会のさまざまな分野において，IT（情報技術）への依存度が高まるとともに，個人的な生活や職場，さらにはビジネスの現場にもITは不可欠な存在になっており，その果たす役割はますます大きくなっている．また，現在の厳しい経済競争環境により，ユーザー市場からの情報機器製品などに対する高品質（Quality）・低コスト（Cost）・開発期間の短縮（Delivery）への要求が強まっている上に，実現される製品は大規模で複雑性の高いものが増えている．一方で，これらの製品の不具合やトラブルで多くの顧客・ユーザーに迷惑をかけるような問題が多発している．最近でも，鉄道会社や航空会社の運行管理システムや運賃精算システム，あるいは銀行・証券会社などの金融機関の現金自動預払機（ATM）および振込処理システム，そして通信会社のネットワークシステムのシステム障害などの社会的責任のある大規模なコンピュータシステムのトラブルは記憶に新しい．それだけではなく，個人の日常生活や生命にも影響を与えうる家電製品・携帯電話や自動車に搭載されたデジタル製品の組み込みソフトのバグによる不具合も，何件もマスコミ機関により公表されていることも事実である．さらに，これらのシステムダウンが長期化することになれば，企業や個人の生活への影響が生じることになり，社会生活を困難にさせることも考えられる．これらの克服，あるいは未然の防止のために，ITを構成する大きな要素のひとつであるソフトウェアの開発形態には，大規模化や複雑化，あるいは多様化への対応が求められている．

現在，一般的に採用されているソフトウェアの開発形態として，開発過程をいくつかの工程に分け，各工程の終了時にはその工程の成果物を文書（document）の形式で作成し，次の工程に移るというウォーターフォール（waterfall）型と呼ばれる開発形態がある．このモデルの概要は以下のとおりである[1].

1）要件定義（specification）

ソフトウェアの使用目的や外部から与えられる条件を明確にし，ユーザーがコンピュータに対して要求している課題や仕様を理解する．その結果，検討された要求事項を要件定義として設定し，それを厳密にかつ完全に記述した要件定義書を作成する．

2）設計（design）

ソフトウェアの要件定義を満たす機能・性能を持つ詳細な処理内容を，使用目的や作成条件に基づいて設計する．設計は，基本設計と詳細設計に大別される．

基本設計では，要件定義に基づいてソフトウェアを機能分割によりできるだけ独立したモジュールに展開し，これらの間の接続方法を決定したうえで，モジュール間の相互関係を確実に実現する制御構造を設計する．

詳細設計では，基本設計で展開されたモジュールの機能をどのようにアルゴリズムで実現するかを定義し，入出力，データ操作など全ての要素を設計する．

その結果，基本設計および詳細設計の設計内容を記述した文書である基本設計書および詳細設計書を作成する．

3）コーディング（coding）

設計結果を特定のプログラミング言語で記述し，コンピュータ上で実行可能なソフトウェアに仕上げる．記述された結果はソースコード（source code）であり，これをコンパイルしてオブジェクトコード（object code）が得られる．

4）テスト（test）

コーディングで作成されたプログラムが要件定義あるいは設計内容を満足するかどうかを検査・確認する．このとき，コーディング終了時点までに開発プロセスでソフトウェア内に潜入した多くの欠陥や誤りを発見し修正する．一般に，単体テスト，統合テストおよび総合テストに分かれている．

単体テストでは，各モジュールに対して設計された機能および性能が満たされているかどうかを確認するもので，プログラム論理を中心としたテストである．

統合テストでは，単体テストで所定の機能を有すると確認されたモジュールを結合して，サブシステムあるいはシステムとしてテストするものなので，モジュール間のインターフェースや理論のつながりなどが確認される．

総合テストでは，ユーザーの使用環境をシミュレートして，要件定義を満たしているかどうかを，機能面および性能の面から全体的にテストを行う．

以上のような工程からソフトウェア開発は構成される．

上に述べた開発形態からもわかる通り，元来ソフトウェアは人間の作った理論を表現した知的生産物である．そのため，その開発の上流工程である設計やコーディングにおいて十分に品質管理を実現したとしても，開発作業の結果として作成されたソフトウェアの中には多くの人為的誤りや欠陥，いわゆるソフトウェアフォールト（以下，単にフォールトと称する）が作り込まれることは不可避である．そして，そのフォールトがもたらすソフトウェアの障害や故障の発生は，のちに不確定現象となって表面化する．したがって，これらの欠陥やフォールトを発見・修正・除去するために実施されるテスト工程は，ソフトウェアという商品を市場に出荷する前にその品質・信頼性を高めるために最終的な品質評価を行うことから，ソフトウェアの開発工程において重要な最終工程である．

また，ソフトウェア開発のプロジェクトマネジメントの立場からすれば，テスト活動を行う最終工程において，品質・信頼性だけでなく開発計画の安定性や開発終了までの進捗度を評価したいという要求があるのも当然である．これに加えて，前に述べたような大規模プロジェクトにおいては，開発管理者個人の能力によって管理することにも限界があり，適切な管理は不可能となってきている．このような状況の中で，適切な工程管理をするた

めに必要となってくるのが，正確で定量的な見積もりを行うことである．正確な見積もり行うことで，ソフトウェア開発においてリソース，開発期間などの適切な配分を可能とすることが期待できる．このように，コスト・納期を考慮したうえでソフトウェアの性能や信頼性を高めることを目的に開発プロセスの進捗度を定量的に管理していこうというソフトウェア工学の考え方への関心が高まって来ている．近年では，より高い品質のソフトウェア製品を効率よく作ることへの重要性は深く認識されている．そして，現在ではソフトウェア生産における管理技術の一つとしてソフトウェアの信頼性評価技法が注目されている．

ソフトウェア信頼性評価技法とは，プロジェクト管理や品質管理を科学的に実践するためにソフトウェア信頼性(注釈:1) の評価を定量的に行う技法である．これに関する研究として，ソフトウェアのテスト工程や運用段階におけるフォールト検出過程やソフトウェア故障(注釈:2) の発生現象をモデル化して，信頼性の達成度合いや推移状況を把握するソフトウェア信頼度成長モデル（software reliability growth model, 以下，SRGM と略す）について，以前より数多く議論されている．SRGM とは，テスト時間の経過とともにソフトウェア内に潜在するフォールト数は発見・修正・除去されて減少していくことによって，ソフトウェア故障の発生する確率が減少してソフトウェア信頼度が増加したり，ソフトウェア故障の発生時間間隔が長くなったりするソフトウェアの実行過程を記述するものである．これまでに研究されてきた SRGM の中でも，非同次ポアソン過程（non-homogeneous Poisson process; NHPP と略される）に基づく SRGM は事実上極めて有効であり現在では広く応用されている [6]〜[8].

この NHPP モデルとは，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を観測して，これらの個数を数え上げる確率変数 $X(t)$ が一定の性質を満たす非同次・非定常なポアソン過程に従うときの SRGM である．具体的に SRGM を用いてソフトウェアの信頼性評価を行うためには，強度関数としてテスト時間 t における瞬間フォールト発見率 $\phi(t)$ を特定化しなければならない．以前より，種々のテスト環境要因を考慮して，フォールト発見事象あるいはソフトウェア故障発生現象を記述する NHPP モデルが，現実的な仮定の下で数多く提案されている．その代表的な NHPP モデルの例として指数形 SRGM モデルや遅延 S 字形 SRGM などが挙げられる [9].

これまでの数多くの SRGM に関する研究がされてきたが，本研究で扱う死滅過程モデルを用いた SRGM については考えられて来なかった．その主な理由として，死滅過程モデルへの適用には，NHPP モデルのような他モデルと異なり，ソフトウェア内に潜在するフォールト数をモデルに含まれる初期状態パラメータとして推定することが困難であることと，NHPP モデルに比べて数式表現が複雑であることなどが挙げられる．

(注釈:1) ソフトウェアが，規定の環境の下で，意図する期間中に，ソフトウェア故障を引き起こすことなく動作することができる性質や度合い [2]〜[4].

(注釈:2) ソフトウェア内に潜在するフォールトにより期待通りにソフトウェアが動作せずに正しく機能しないこと [5].

1.2 研究目的

先行研究 [10] では実際のテスト活動で得たテスト項目消化数とフォールト発見数の観測データの死滅過程モデルへの適用として，テスト工程の進捗度とソフトウェア信頼度の評価を試みている．ここで，テスト項目消化数の推移に状態依存を考慮した死滅過程を適用しているのに対し，フォールト発見数のそれは遅延 S 字形 SRGM へ適用している．そして，事前に用意されたテスト項目数が全て消化される時刻を示す平均テスト完了時刻とそのときのソフトウェア信頼度を評価している．

この研究で挙げられた問題として，死滅過程モデルをテスト項目の消化過程に適用するのではなく，ソフトウェアフォールトの減少過程に直接適用しようとした場合に，平均テスト完了時刻の推定においては，死滅過程モデルに含まれる初期状態パラメータにはソフトウェアテストに先立って事前に用意した既知のテスト項目の総数と置き換えができることに対して，フォールト発見数のモデルへの適用に対してはそれができず，結果として死滅過程モデルをソフトウェアの信頼性評価モデルとしては構築できなかったことが挙げられる．

しかしながら，その後の研究の成果によって，ある特定のソフトウェアテスト方法（3 章を参照）を実施することで，ソフトウェア内に潜在する総フォールト数を連立方程式によって導出できることがわかった．これによって，初期状態パラメータを決定し，フォールト発見数の死滅過程モデルへの適用が可能となる．

最終的に，本手法がソフトウェア開発におけるテスト工程の定量的進捗度管理とソフトウェア信頼性評価の精度向上に役立てることができると考えている．

1.3 本論文の構成

まず，2 章にて現在のソフトウェア開発におけるテスト工程が抱える問題を提起する．この問題解決へのアプローチとして，今回新しく採用した 2 チームによる 2 段階ソフトウェアテスト方法の手順を 3 章にて述べる．そして，このモデルを 4 章で記述するとともに数値計算によるソフトウェア内に潜在する総フォールト数の導出方法を示す．この数値の導出によって可能となる動的信頼性評価モデルとして扱う死滅過程モデルへの累積フォールト発見数の適用方法を 5 章で述べる．その適用例として，6 章にて実際のテスト活動で得たフォールト発見数の観測データを用いて死滅過程モデルへの適用を試みる．この適用により，ソフトウェア内に潜在する総フォールト数とこれを全て発見・修正する将来の時刻を示す平均全フォールト発見時刻の推定値が得られる．これを実際に任意のテスト時間においてテスト進捗度評価を実施すると想定し，それぞれのテスト時間までの観測データのみを用いて逐次推定する．そして，それぞれの推定結果の値を表とグラフを用いて示す．最後に 7 章で，推定結果と実際の数値との比較を行い，本評価手法がどの程度有効であるかについて考察し，8 章にて本研究の結論，および今後の課題について検討する．

2　ソフトウェア開発のテスト工程における問題点とその解決のためのアプローチ

一般に，ソフトウェア開発者は，ソフトウェア開発におけるテスト工程や，製品が出荷された後の運用状況を模擬したソフトウェアテストにおいて，開発されたソフトウェアが，ユーザーの要件定義書通りに作動し，設定された性能水準をクリアしているか否か，あるいは，機能の程度に何らかの逸脱はないかなどの妥当性を確認するために，大量のテストケースを用意してテスト活動を行うことが通例である．これらのテスト活動は，膨大な作業量となることが多く，このような場合，全てのソフトウェアプログラムの動作を検証することは，前章で述べたようにソフトウェア開発の期間およびコストの制約を考慮すれば，明らかに困難である．その一方で，この困難に対処するためにユーザーのシステムやサービスに対する要求事項または変更事項が漏れなく各文書に反映され、各種文書間で整合性が取られた文書を維持することを目的とするトレーサビリティマトリックス[11]を用いる方法なども知られており，ソフトウェア開発の現場において，テスト活動を効率化するための様々な取り組みが積極的になされている[12][13].

こうした中，ソフトウェア開発者がしばしば経験する事象として，以下を挙げることができる．

- テストを十分に実施したにも関わらず，ソフトウェアの出荷直前に実施するユーザーの環境下での受け入れテストにおいて，いくつかのフォールトが検出される．
- 市場に出荷した組込系ソフトウェアを有する機器などが，不特定多数のユーザー環境下において実用された際，不具合に関するクレームを受ける．

これらの原因は，開発現場やその他において，依然として指摘されているように，ソフトウェアの動作確認が不十分であったこと，つまりテストの網羅性に属する問題であると考えられており，現在では様々な網羅性を確認する方法やツールが知られている．これについては近年，テスト項目数の効率的な削減を目的とした，吉澤らによる直交表を用いたテストケース数の論理的および効率的な削減手法[14]などが知られるようになった．この手法は，テストケース全体の質をなるべく落とさないまま，テストケース数を削減するという意味において，テストの網羅性を実効的に高められる可能性がある．

一方，テストの網羅性および十分性がどの程度かを判断する際，ソフトウェア内に潜在する総フォールト数と，あるテスト時間における発見・修正されたフォールト数の実績値とを差し引きした値を用い，これがゼロに近づくことをもって，テストの網羅性を逐次的に推定できる可能性も考えることができる．もちろん，それだけを頼みにしてテストの進捗度や網羅性を評価することは危険であるが，静的モデルに分類されるソフトウェアの残存フォールト数の推定方法と，テストの経過時間に沿って更新される，いくつかのテスト時間における累積フォールト発見数の時系列データとしての振る舞いを捉える動的モデル

とを併用することによって，テストの網羅性の程度（すなわちテスト網羅度）の時間経過に伴う向上の様子を，定量的に推定することができる可能性が考えられる．

以上に述べた静的モデルのうち，古典的なものとしては例えば，捕獲・再捕獲法（capture-recapture method）[15],[16] や２段階エディット法 [17] などがそれに該当する．一方，動的モデルとしては前章で述べたような成長モデルに確率的な実行過程の意味付けを与えた SRGM が挙げられる．これのうち，本研究で述べるモデルは静的モデルに該当する．

さて，本研究の協力者の一人は，ある PC 向けソフトウェア製品の開発を行った際，それが Windows 7(注釈:3) の 32 ビット版 OS および 64 ビット版 OS において動作することを確認する機会を得た．以下にその概要を述べる．

まず，そのソフトウェア開発においては，コーディング工程までを終えたソフトウェアに対して，初めに 32 ビット版 OS 上での動作確認（テストおよびデバック）を実施した．その後，回帰テストの意味も付加して， 64 ビット版 OS 上にそのソフトウェアを導入した上で，再度同じテストを行った．この２度のテストにおいては，同一のテストケースを用いた．さらにこのとき， 32 ビット版 OS 上でのテストおよびデバックの際，デバックによる他のプログラムによる影響がない，つまり不完全デバックになっていないかを検証する作業（回帰テスト）も行なっていたことを付記しておく．

このようなテストを行った場合，原理的には，同一のテストケースを再び用いた２度目（64 ビット版）のテストにおいては，新たなフォールトが発見されることは無いと予想されるが，実際にはいくつかのフォールトが検出されることとなった．その原因を調査したところ，いくつかの事例が挙げられ，これが前述したユーザー環境下での受け入れテストや，出荷後において発生する不具合の原因となっているのではないかと考えられる．それらは以下の事柄である．

- テスト担当者が，テスト対象に対する操作方法について，本質的な部分での理解が若干不足しており，全てのテストパスがテストされず，フォールトを見逃した．
- テスト対象ソフトウェアが期待通りの動作あるいは出力結果でないにも関わらず，テスト担当者がその詳細確認を怠ったため，フォールトを見逃した．

これらに対して，上記のうちの第１原因によるフォールトの見逃しについては，例えば，テスト仕様の策定や設計要求の整理と仕様化におけるチェックリストやレビューの充実化によって，ある程度はその頻度が抑えられるものと考えられる．また，第２段階の原因として挙げたものについてはテスト支援ツールやいわゆるテストの自動化ツールを効果的に用いることによって，避けることができると考えられる．

以上を踏まえた上でテスト作業の構成を発展させ，本研究では，後述する２段階のテスト作業を２つのチームによって行うことを発案した．これによって，以下に述べるテスト方法は，担当したチームのフォールト発見率（潜在フォールトを見逃さない能力）も評価できることがわかった．

(注釈:3) ：Windows 7 は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標．

3　2チームによる2段階ソフトウェアテスト

前章にて述べたように，Windows 7 OS の 32 ビット版および 64 ビット版のどちらでも動作するソフトウェアについてのテスト作業を以下のように行った．また，その概要を図 1 に示す.

1. ソフトウェア開発のためのプロジェクトメンバーを２つのチームに分けた．これらのチームをチーム A およびチーム B と呼ぶ.

2. ソフトウェア全体に対するユーザーからの要求仕様と照らし合わせて，コーディング後にそれらの独立テストを行なっても差し支えない２つの機能群に分割し，整理した．それらを機能群 A および機能群 B と呼ぶ.

3. ソフトウェア開発の上流である要件定義から基本設計の工程においては，両チームが合同で開発作業を担当した.

4. コーディングからソフトウェアテスト１段階目においては，チーム A が機能群 A を，チーム B が機能群 B を担当した．ここで，テストケースは各チームがコーディング後に独自に作成したが，その後一旦それらを持ち寄り，全員でレビューを行ってテストケースの問題点を抽出し，それらを修正し，合意が得られたテストケースを作成した.

5. 上記 4. で作成されたテストケースを用いて，各チームがそれぞれコーディングを担当した機能群を独立かつ並行して 32 ビット版 OS 上での機能確認およびデバックを行った．これをテスト 1 段階目と呼ぶ.

6. テスト１段階目終了後，互いのチームのテスト成果物を入れ替え，機能群 B をチーム A が，機能群 A をチーム B がテストを実施した．このとき実施したテストケースは，上記 4. で各チームが追加・修正したものと同様のものを用いた．これを回帰テスト２段階目と呼ぶ.

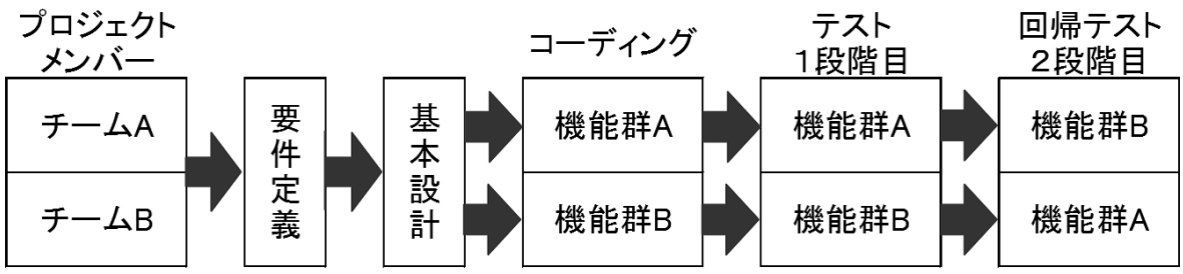

図 1: 2 チームによる 2 段階ソフトウェアテストの流れ.

本研究では，以上のテストを２チームによる２段階ソフトウェアテストと呼ぶ．以下に，各チームによるコーディング終了時点以降のテスト活動の詳細について述べる．

1. 64 ビット版 OS の上でソースコードをコンパイルする．このとき，オブジェクトコードは 32 ビット版， 64 ビット版のいずれの OS においても動作するように作成している．

2. 予め与えられたユーザー要求を満たしているか否かを効率的に判定できるテストケースを，担当する各チームごとに設計する．この後，両チーム合同でレビューを実施し，設計したテストケースに修正を施し，両チームの合意が得られるまでレビューを継続する．

3. 各チームがテストケースに基づいて，十分にテストおよびデバックを行い，デバックの際の回帰テストも行う．この作業は 32 ビット版 OS 上での作業である（テスト 1 段階目でのテスト作業）

4. 機能群 A およぶ B に対してチームを入れ替え， 64 ビット版 OS 上でも期待通りに動作するか否かについて，それぞれの機能群の動作確認のために 2. で作成されたテストケースを用いて，テストを行う（回帰テスト２段階目でのテスト作業）．これは全体を通じた回帰テストの意味合いであり，予定ではこの作業で指摘されるフォールトはゼロであることが期待されていた．

これらの一連の作業において，項目 4. に挙げた， 64 ビット版 OS 上での繰り返しのテストは，ユーザー側での受け入れテストに変わるものとして位置付けることもできる．つまり，このテスト活動の項目 3. および 4. で発見されたフォールト数の情報を用いて，次の章で述べるソフトウェアの信頼性やテストを実施したチームの能力を表す指標を推定するモデルを作成することができる．

また，各テスト段階で発見されたフォールトは全て， OS のバージョンの差異などにより起因するものではなかったこと，また，回帰テスト２段階目で発見された全てのフォールトはテスト１段階目でのフォールトの発見と修正の記録に照らして，全く別の，新規のフォールトであることが確認されたことに注意すべきである．

4　静的信頼性評価モデル

以下に，3 章で説明したテスト方法のモデルを記述するための変数を定義する．

N_A : 機能群 A 内に潜在する総フォールト数
N_B : 機能群 B 内に潜在する総フォールト数
m_{1A} : テスト 1 段階目でのチーム A のフォールト発見率，あるいは能力
m_{1B} : テスト 1 段階目でのチーム B のフォールト発見率，あるいは能力
m_{2A} : 回帰テスト 2 段階目でのチーム A のフォールト発見率，あるいは能力
m_{2B} : 回帰テスト 2 段階目でのチーム B のフォールト発見率，あるいは能力
X_{1A} : テスト 1 段階目後の機能群 A に残存するフォールト数（確率変数）
X_{1B} : テスト 1 段階目後の機能群 B に残存するフォールト数（確率変数）
X_{2A} : テスト 2 段階目後の機能群 A に残存するフォールト数（確率変数）
X_{2B} : テスト 2 段階目後の機能群 B に残存するフォールト数（確率変数）

図 2 に，本研究で設定する 2 チームによる 2 段階ソフトウェアテストの概要を示す．この

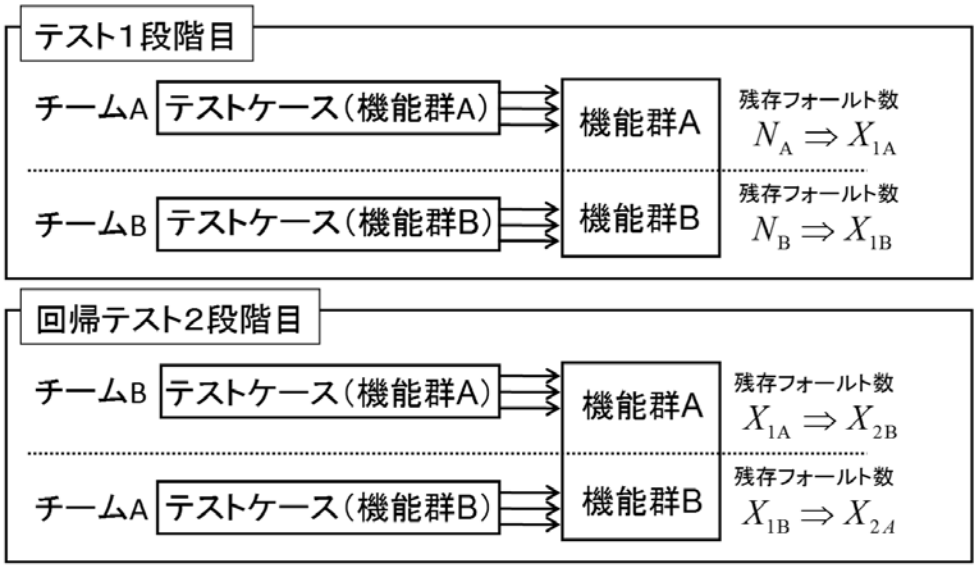

図 2: 2 チームによる 2 段階ソフトウェアテストの詳細.

テスト方式のモデル化について，単純に 2 項分布に基づくものを与えることにすれば，テスト 1 段階目において以下の定式化を行うことができる．

$$\Pr[X_{1A} = x_{1A}] \quad = \quad \binom{N_A}{x_{1A}} (1 - m_{1A})^{x_{1A}} {m_{1A}}^{N_A - x_{1A}}, \tag{4.1}$$

$$\Pr[X_{1B}=x_{1B}] = \binom{N_B}{x_{1B}}(1-m_{1B})^{x_{1B}}{m_{1B}}^{N_B-x_{1B}}. \tag{4.2}$$

ここで，x_{1A} および x_{1B} は，それぞれ X_{1A} および X_{1B} の表現値である．また，$\Pr[A]$ は事象 A が起こる確率を表す．

このことから，テスト 2 段階目終了後に残存するフォールト数である X_{2A} および X_{2B} の確率変数は以下のように与えられる．

$$\begin{aligned}
\Pr[X_{2A}=x_{2A}] &= \sum_{k=x_{2A}}^{N_B}\binom{X_{1B}}{x_{2A}}(1-m_{2A})^{x_{2A}}{m_{2A}}^{X_{1B}-x_{2A}}\Pr[X_{1B}=k] \\
&= \sum_{k=x_{2A}}^{N_B}\Big[\binom{k}{x_{2A}}(1-m_{2A})^{x_{2A}}{m_{2A}}^{k-x_{2A}}\binom{N_B}{k}(1-m_{1B})^{k}{m_{1B}}^{N_B-k}\Big] \\
&= \binom{N_B}{x_{2A}}\{(1-m_{2A})(1-m_{1B})\}^{x_{2A}} \\
&\quad \times\{1-(1-m_{2A})(1-m_{1B})\}^{N_B-x_{2A}},
\end{aligned} \tag{4.3}$$

$$\begin{aligned}
\Pr[X_{2B}=x_{2B}] &= \sum_{k=x_{2B}}^{N_A}\binom{X_{1A}}{x_{2B}}(1-m_{2B})^{x_{2B}}{m_{2B}}^{X_{1A}-x_{2B}} \\
&\quad \times\Pr[X_{1A}=k] \\
&\quad \sum_{k=x_{2B}}^{N_A}\Big[\binom{k}{x_{2B}} \\
&\quad \times(1-m_{2B})^{x_{2B}}{m_{2B}}^{k-x_{2B}}\binom{N_A}{k}(1-m_{1A})^{k}{m_{1A}}^{N_A-k}\Big] \\
&= \binom{N_A}{X_{2B}}\{(1-m_{2B})(1-m_{1A})\}^{x_{2B}} \\
&\quad \times\{1-(1-m_{2B})(1-m_{1A})\}^{N_A-X_{2B}}.
\end{aligned} \tag{4.4}$$

したがって，最終的に残存している機能群 A および B のフォールト数 X_{2A} および X_{2B} の期待値と分散は，それぞれ

$$\mathrm{E}[X_{2A}] = N_B\{(1-m_{2A})(1-m_{1B})\}, \tag{4.5}$$

$$\mathrm{E}[X_{2B}] = N_A\{(1-m_{2B})(1-m_{1A})\}, \tag{4.6}$$

$$\mathrm{Var}[X_{2A}] = N_B\{(1-m_{2A})(1-m_{1B})\}\{1-(1-m_{2A})(1-m_{1B})\}, \tag{4.7}$$

$$\mathrm{Var}[X_{2B}] = N_A\{(1-m_{2B})(1-m_{1A})\}\{1-(1-m_{2B})(1-m_{1A})\}, \tag{4.8}$$

となる(注釈:4)．

このことから，テスト 2 段階目終了後に残存するフォールト数の確率分布を評価することができ，次節に述べる観測データに基づくパラメータ推定を行うことによって，ソフトウェ

(注釈:4) 機能群 B を最終的に担当したのはチーム A であることに注意されたい．また，図 2 を参照のこと．

アの信頼性評価の一助とすることが可能である．また，各チームのテストにおけるフォールト発見率，あるいはフォールト発見能力である m_{1A}, m_{1B}, m_{2A}, および m_{2B} についても同様にそれらを評価することができる．

4.1 数値計算によるソフトウェア内に潜在する総フォールト数の推定方法

観測データが与えられたとき，上述したテストモデルに含まれる未知パラメータ N_A, N_B, $m_{1A}, m_{1B}, m_{2A},$ および m_{2B} の推定量は以下のように与えられる．チーム A のテスト 1 段階目における発見フォールト数の実現値を d_{1A}，またチーム B のそれを d_{1B} とすれば，期待値から，

$$N_A m_{1A} = d_{1A}(>0), \tag{4.9}$$

$$N_B m_{1B} = d_{1B}(>0), \tag{4.10}$$

となる．回帰テスト 2 段階目においても同様に，各チームの発見フォールト数の実現値 d_{2A} および d_{2B} とすれば，同様に

$$N_B(1-m_{1B})m_{2A} = d_{2A}(>0), \tag{4.11}$$

$$N_A(1-m_{1A})m_{2B} = d_{2B}(>0), \tag{4.12}$$

の関係式を得ることができる．このように，変数を用いた関係式によって表すことができる各テスト段階や機能群毎の発見・修正されたフォールト数と各テスト段階終了後に残存するフォールト数を表にしたものを表 1 に示す．ここで，各テスト段階において各チーム

表 1: 変数によって表すことができる関数式.

	機能群	発見・修正されたフォールト数	テスト終了後，各機能群に残存するフォールト数
テスト 1 段階目	A	$N_A m_{1A} = d_{1A}$	$N_A(1-m_{1A})$
	B	$N_B m_{1B} = d_{1B}$	$N_B(1-m_{1B})$
テスト 2 段階目	A	$N_A(1-m_{1A})m_{2B} = d_{2A}$	$N_A(1-m_{1A})\,(1-m_{2B})$
	B	$N_B(1-m_{1B})m_{2A} = d_{2B}$	$N_B(1-m_{1B})\,(1-m_{2A})$

の発見能力が等しいと仮定すれば $m_{1A} = m_{2A}$ および $m_{1B} = m_{2B}$ とすることができる．そして，上記の式（4.9）〜式（4.12）を連立させて解くことで，以下の推定量を得ることができる．

$$\widehat{N_A} = \frac{(d_{1A}+d_{2B})d_{1A}d_{1B}}{d_{1A}d_{1B}-d_{2A}d_{2B}}, \tag{4.13}$$

$$\widehat{N_B} = \frac{(d_{1B} + d_{2A})d_{1A}d_{1B}}{d_{1A}d_{1B} - d_{2A}d_{2B}}, \tag{4.14}$$

$$\widehat{m_{1A}} = \widehat{m_{2A}} = \frac{d_{1A}d_{1B} - d_{2A}d_{2B}}{(d_{1A} + d_{2B})d_{1B}}, \tag{4.15}$$

$$\widehat{m_{1B}} = \widehat{m_{2B}} = \frac{d_{1A}d_{1B} - d_{2A}d_{2B}}{(d_{1B} + d_{2A})d_{1A}}. \tag{4.16}$$

このとき，回帰テスト 2 段階目終了後，ソフトウェア内に残存するフォールト数の期待値は機能群 A および B のそれの和で表せるので，表 1 より，

$$N_A(1 - m_{1A})\,(1 - m_{2B}) + N_B(1 - m_{1B})\,(1 - m_{2A}), \tag{4.17}$$

によって導出できる．また，ソフトウェア内に潜在する総フォールト数は，

$$N_0 = N_A + N_B, \tag{4.18}$$

で求めることができる．また，後の 5 章で述べる死滅過程モデルへの適用のために，N_0 の値を四捨五入し，正整数化したものを N とする．

4.2 適用例

本節では，前に述べたパラメータ推定法を実際の観測データに適用する．ここに，表 2 のデータセットが与えられたとする．このとき，各機能群に潜在する総フォールト数は式（4.13）および式（4.14）より，

$$\widehat{N_A} = 20.36, \tag{4.19}$$

$$\widehat{N_B} = 20.36, \tag{4.20}$$

となる．これらの推定値を用いてソフトウェア内に潜在する総フォールト数は式（4.18）より，

$$N_0 \;=\; N_A + N_B = 40.72, \tag{4.21}$$

となる．これを四捨五入すると N は以下のようになる．

$$N = 41. \tag{4.22}$$

また，各チームのフォールト発見率，あるいは能力は，式（4.15）および式（4.16）より以下のように得られる．

$$\widehat{m_{1A}} = \widehat{m_{2A}} = 0.688, \tag{4.23}$$

$$\widehat{m_{1B}} = \widehat{m_{2B}} = 0.786. \tag{4.24}$$

表 2 をみれば，機能群 A でのテスト 2 段階目を通じての総フォールト発見数と，機能群 B でのそれは，共に 19 個となっており，このことから式（4.19）および式（4.20）の値は等しくなるが，一方でチーム A はテスト 1 段階目で取り残したフォールト数がチーム B より多く，またテスト 2 段階目においてもチーム A はチーム B よりも少ないフォールトの検出数を記録したため，式（4.23）および式（4.24）に示すように，フォールトの発見率ではチーム B が A を上回ることとなった．

さらに，式（4.3）および式（4.4）から，機能群 A および B を合わせたプログラム全体に残存するフォールト数の期待値および標準偏差の評価結果はそれぞれ

$$\hat{\mathrm{E}}[X_{2A} + X_{2B}] \simeq 2.7, \tag{4.25}$$

$$\sqrt{\widehat{\mathrm{Var}}[X_{2A} + X_{2B}]} \simeq 1.6, \tag{4.26}$$

となる．これらの結果は，$m_{1A} = m_{2A}$ および $m_{1B} = m_{2B}$ を仮定したことにより，2 項分布の再生性に基づいて得ることができる．

以上のモデル化において，もし，チーム A のテスト 2 段階目におけるフォールトの発見数 d_{2A} がゼロであった場合は，チーム B のフォールト発見率 $m_{1b}(= m_{2B})$ は 1 となる．また，同様にしてチーム B のテスト 2 段階目におけるフォールトの発見数 d_{2B} がゼロであった場合は，チーム A のフォールト発見率 $m_{1A}(= m_{2A})$ が 1 となることは明らかである．これらはどちらも，テスト 1 段階目を担当したチームが，相手チームが後に発見し得ないという意味での，すべてのフォールトを発見したという意味である．

表 2: データセット．

テスト段階-チーム	1-A	1-B	2-A	2-B
フォールト発見数	$d_{1A} = 14$	$d_{1B} = 16$	$d_{2A} = 3$	$d_{2B} = 5$
テストされた機能群	A	B	B	A

5　動的信頼性評価モデル

本章では，本研究で動的信頼性評価モデルとして使用する死滅過程モデルについて述べ，これをソフトウェア開発におけるテスト工程の進捗状況を把握するための適用方法を示す．これによって，いくつかのテスト時間における累積フォールト発見数の時系列データとしての振る舞いを動的に捉えることが可能となる．

5.1　死滅過程モデル

まず一般的な死滅過程モデルについて述べる．

今，既知の人口 K（$K \geq 1$）を初期状態とし，ある時刻 t において残存する人口を確率変数 $X(t)$ と表すことにする．残存する人口は，時間の経過と共に次第に減少していくことになる．そこで，計数過程 $\{X(t), t \geq 0\}$ は，以下に列挙する仮定（a）から（d）に基づいて推移することを考える [19]．

(a)　$\Pr[X(t+\Delta t) = x-1 | X(t) = x] = \phi(t)\Delta t + o(\Delta t)$
すなわち，任意の微小区間 Δt で人口が 1 が減少する確率は人口の減少確率（強度関数）$\phi(t)$ に比例する．

(b)　$\Pr[X(t) - X(t+\Delta t) \geq 2] = o(\Delta t)$
すなわち，任意の微小区間 Δt で人口が 2 以上減少する確率は無視できるほど小さい．

(c)　$\Pr[X(0) = K] = 1$　　（K は正整数）
すなわち，時刻 $t = 0$ では人口は確率 1 で K である．

(d)　$\{X(t), t \geq 0\}$ は独立増分をもつ
相異なる計測時間区間で死滅した人口は統計的に独立である．

ここで，$o(\Delta t)$ は高位の無限小であり，$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$, が成立する．図 3 に，これらの仮定に基づいて状態が遷移する図を示す．

上記の（a）から（d）の仮定に基づいて人口が経過する時間によって減少していく過程は，以下の連立方程式で表される．

$$
\begin{aligned}
\Pr[X(t+\Delta t) = K] &= 1 - \phi(t)\Delta t + o(\Delta t)\Pr[X(t) = K], \\
\Pr[X(t+\Delta t) = n] &= \phi(t)\Delta t \Pr[X(t) = n+1] \\
&\quad +1 - \phi(t)\Delta t + o(\Delta t)\Pr[X(t) = n] \quad (n = 1, 2, ..., K-1), \\
\Pr[X(t+\Delta t) = 0] &= \phi(t)\Delta t + o(\Delta t)\Pr[X(t) = 1]\Pr[X(t) = 0] + o(\Delta t), \qquad (5.1)
\end{aligned}
$$

これらを t についてそれぞれ微分すると，以下の連立微分方程式

$$\begin{aligned}
\frac{d\Pr[X(t)=K]}{dt} &= -\phi(t)\Pr[X(t)=K], \\
\frac{d\Pr[X(t)=n]}{dt} &= \phi(t)\Pr[X(t)=n+1]-\phi(t)\Pr[X(t)=n] \\
&\quad (n=1,2,...,K-1), \\
\frac{d\Pr[X(t)=0]}{dt} &= \phi(t)\Pr[X(t)=1], \qquad (5.2)
\end{aligned}$$

を得る．この連立微分方程式の初期条件は仮定（c）より $\Pr[X(0)=K]=1$ と $\Pr[X(0)=n]=0$ $(n=1,2,...,K)$ であり，これらを解くと，時刻 t での残存する人口が n 人である確率は，

$$\Pr[X(t)=n] = \frac{\{G(t)\}^{K-n}}{(K-n)!}e^{-G(t)} \qquad (n=1,2,...,K), \qquad (5.3)$$

$$\Pr[X(t)=0] = \sum_{n=K}^{\infty}\frac{\{G(t)\}^{n}}{n!}e^{-G(t)}, \qquad (5.4)$$

$$G(t) = \int_0^t \phi(t)dt \qquad (t\geq 0), \qquad (5.5)$$

となる．実際の解析においては式（5.5）に含まれる強度関数 $\phi(t)$ を適切に与える必要がある．本研究では，死滅過程の推移率を与える強度関数として以下を与えることとした．

$$\phi(t) = \frac{abe^{-bt}(bt)^{c-1}}{\Gamma(c)}+d \qquad (a,b,c,d>0). \qquad (5.6)$$

また，これを式（5.5）に代入すると以下のようにして表される．

$$G(t) = a\cdot\frac{1-\Gamma(c,bt)}{\Gamma(c)}+dt. \qquad (5.7)$$

なお，ここで用いられている完全ガンマ関数および不完全ガンマ関数を以下に示す．

$$\Gamma(a) = \int_0^{\infty} e^{-x}x^{a-1}dx, \qquad (5.8)$$

$$\Gamma(a,b) = \int_b^{\infty} e^{-x}x^{a-1}dx. \qquad (5.9)$$

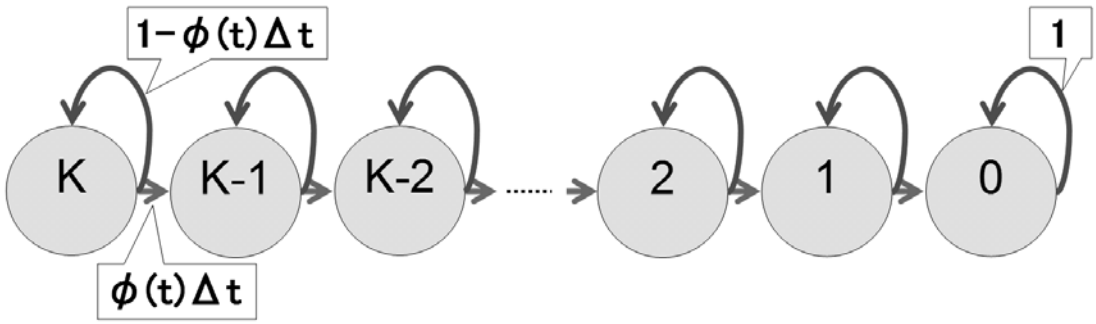

図 3: 状態遷移図．

5.2 ソフトウェア信頼性評価への適用方法

本研究では，前章で述べた死滅過程モデルをソフトウェア開発におけるテスト工程の進捗状況と任意のテスト時間でのソフトウェア内に含まれる残存フォールト（バグ）数を把握するために用いる．本節では，そのための有用な評価尺度として，平均フォールト発見数，および平均全フォールト発見時刻の導出方法を示す．ここで，死滅過程モデルに用いる初期状態 K（$K \geq 1$）を 4.1 節で算出方法を述べたソフトウェア内に潜在する総フォールト数 N，死滅過程 $\{X(t), t \geq 0\}$ をテスト時間 t のときにソフトウェア内に残存するフォールト数，$\phi(t)$ を強度関数としてテスト時間 t における瞬間フォールト発見率と置き換えて考える．

5.2.1 平均フォールト発見数

平均フォールト発見数とは，テスト時間 t において，それまでに発見・修正されたフォールト数の期待値を表す．まず，確率変数 $X(t)$ はテスト時間 t のときにソフトウェア内に残存するフォールト数であることから，その期待値 $\mathrm{E}[X(t)]$ は式（5.3）より，

$$\begin{aligned}
\mathrm{E}[X(t)] &= \sum_{n=0}^{N} n \Pr[X(t)=n] \\
&= \sum_{n=1}^{N} \frac{G(t)^{N-n}}{(N-n)!} e^{-G(t)} \\
&= e^{-G(t)} \sum_{n=1}^{N} n \frac{G(t)^{N-n}}{(N-n)!}, \qquad (5.10)
\end{aligned}$$

となる．ここで，式（5.10）の最後の部分を展開すると，

$$\begin{aligned}
\sum_{n=1}^{N} n \frac{G(t)^{N-n}}{(N-n)!} &= 1\frac{G(t)^{N-1}}{(N-1)!} + 2\frac{G(t)^{N-2}}{(N-2)!} + \\
&\quad + (N-1)\frac{G(t)^{N-(N-1)}}{(N-(N-1))!} + N\frac{G(t)^{N-N}}{(N-N)!} \\
&= N\frac{G(t)^0}{0!} + (N-1)\frac{G(t)^1}{1!} + \\
&\quad + N - (N-1)\frac{G(t)^{N-2}}{(N-2)!} + N - (N-1)\frac{G(t)^{N-1}}{(N-1)!} \\
&= N \cdot \left[1 + \frac{G(t)}{1!} + \frac{G(t)^2}{2!} + \ + \frac{G(t)^{N-1}}{(N-1)!}\right] \\
&\quad - G(t)\left[1 + \frac{G(t)}{1!} + \frac{G(t)^2}{2!} + \ + \frac{G(t)^{N-2}}{(N-2)!}\right] \\
&= N \cdot \sum_{n=0}^{N-1} \frac{G(t)^n}{n!} - G(t) \sum_{n=0}^{N-2} \frac{G(t)^n}{n!}, \qquad (5.11)
\end{aligned}$$

となる．ところで，

$$\sum_{n=0}^{n} \frac{x^r}{r!} = e^x \left[1 - \frac{1}{n!} \int_0^x e^{-z} z^n dz\right], \tag{5.12}$$

であるので，

$$\begin{aligned} \sum_{n-1}^{N} n\frac{G(t)^{N-n}}{(N-n)!} &= N \cdot e^{G(t)} \left[1 - \frac{1}{(N-1)!} \int_0^{G(t)} e^{-z} z^{N-1} dz\right] \\ &\quad -G(t)e^{G(t)} \left[1 - \frac{1}{(N-2)!} \int_0^{G(t)} e^{-z} z^{z-2} dz\right], \end{aligned} \tag{5.13}$$

と書き換えられる．ここで，

$$\int_0^a f\,(x)dx = \int_0^\infty f\,(x)dx - \int_a^\infty f\,(x)dx, \tag{5.14}$$

を用い，式（5.13）を変形すると，

$$\sum_{n=1}^{N} n\frac{G(t)^{N-n}}{(N-n)!} = N \cdot e^{G(t)} \left[1 - \frac{1}{(N-2)!}\left(\int_0^\infty e^{-z} z^{k-2} dz - \int_{G(t)}^\infty e^{-z} z^{N-2} dz\right)\right], \tag{5.15}$$

となる．ここで，式（5.8）および式（5.9）によって表される不完全ガンマ関数比

$$A\,(N, G(t)) = \frac{\Gamma\,(N, G(t))}{\Gamma\,(N)}, \tag{5.16}$$

を用いると，(5.15）は，

$$\begin{aligned} \sum_{n=1}^{N} n\frac{G(t)^{N-n}}{(N-n)!} &= N \cdot e^{G(t)} \left[1 - \frac{1}{\Gamma\,(N)} \left[\Gamma\,(N) - \Gamma\,(N, G(t))\right]\right] \\ &\quad -G(t)e^{G(t)} \left[1 - \frac{1}{\Gamma\,(N-1)} \left[\Gamma\,(N-1) - \Gamma\,(N-1, G(t))\right]\right] \\ &= N \cdot e^{G(t)} \frac{\Gamma\,(N, G(t))}{\Gamma\,(N)} - G(t)e^{G(t)} \frac{\Gamma\,(N, G(t))}{\Gamma\,(N-1)} \\ &= N \cdot e^{G(t)} A\,(N, G(t)) - G(t)e^{G(t)} A\,(N-1, G(t)), \end{aligned} \tag{5.17}$$

となる．よって，テスト時間 t のときにソフトウェア内に残存するフォールト数の期待値は，

$$\mathrm{E}[X(t)] = e^{-G(t)} \left[N \cdot e^{G(t)} A\,(N, G(t)) - G(t)e^{G(t)} A\,(N-1, G(t))\right], \tag{5.18}$$

これらを整理して，

$$\mathrm{E}[X(t)] = N \cdot A\,(N, G(t))] - G(t)A\,(N-1, G(t)), \tag{5.19}$$

となる．ここで，確率変数 $X(t)$ はテスト時間 t のときにソフトウェア内に残存するフォールト数であることから，それまでに発見・修正されたフォールト数はソフトウェア内に潜在する総フォールト数 N（$N \geq 1$）と $X(t)$ との差分で求めることができる．つまり，テスト時間 t における平均フォールト発見数の期待値 $\mathrm{E}[N - X(t)]$ は以下のように表せる．

$$\begin{aligned} \mathrm{E}[N - X(t)] &= N - \mathrm{E}[X(t)] \\ &= N - [NA\,(N, G(t))] - G(t)A\,(N-1, G(t))] \\ &= N[1 - A\,(N, G(t))] + G(t)A\,(N-1, G(t)). \end{aligned} \tag{5.20}$$

5.2.2 平均全フォールト発見時刻

平均全フォールト発見時刻とは，ソフトウェア内に潜在する総フォールト数 N が全て発見・修正されるときの将来時刻 T の期待値 $\mathrm{E}[T]$ を表す．まず，ソフトウェア内に潜在する総フォールト数 N がテスト時間 t までにすべて発見される確率の分布関数を $B(t)$ とすると，

$$\begin{aligned} B(t) &= e^{-G(t)} \sum_{n=N}^{\infty} \frac{G(t)^n}{n!} \\ &= \int_0^t \frac{\phi(x) G(t)^{N-1}}{(N-1)!} e^{G(x)} dx, \end{aligned} \tag{5.21}$$

であることから，T の期待値は以下のように表せる．

$$\begin{aligned} \mathrm{E}[T] &= \int_0^{\infty} t dB(t) \\ &= \int_0^{\infty} t \frac{\phi(t) G(t)^{N-1}}{(N-1)!} e^{-G(t)} dt. \end{aligned} \tag{5.22}$$

5.3 最小二乗法による未知パラメータの推定方法

本研究では，前節に述べた 2 つの信頼性評価尺度に含まれる未知パラメータのうち，テスト開始前にソフトウェア内に潜在していた総フォールト数を表すパラメータ N 以外のものについての推定に最小二乗法を用いる．最小二乗法とは，観測したデータを推定したモデルの関数を用いて近似するとき，推定したモデルの関数と観測データの距離が最小となるような未知パラメータを決定する方法である．この方法を用いて，5.2.1 節および 5.2.2 節で述べた平均フォールト発見数と平均全フォールト発見時刻を導出する式（5.20）および式（5.22）における，関数 $G(t)$ に含まれる未知パラメータ a, b, c, および d の値を推定する [20].

いま，観測データ（t_i, x_i）（$i = 1, 2, ..., n$）が与えられているとき，最小二乗和を求める式は

$$\min_{a,b,c,d} \sum_{i=1}^{I} (\mathrm{E}[N - X(t)] - x_i)^2, \tag{5.23}$$

となる．この最小値を求めることにより，未知パラメータ a, b, c, および d の推定値を求めることができる．ここで，I はテスト 2 段階目における任意のテスト時間であり，そのとりうる値は $I = 13, 14, ..., 29$ である（表 3 を参照）.

6　ソフトウェア信頼性評価への適用例

本章では，実際のテストで観測されたデータを死滅過程モデルへ適用し，ソフトウェアの信頼性評価を行う．なお，未知パラメータ N の推定には 4 章にて示した静的信頼性評価モデルを用いて推定する．これに対して，未知パラメータ a, b, c, および d の推定には 5.3 節にて述べた最小二乗法を用いる，

ここに，3 章で述べた 2 チームによる 2 段階ソフトウェアテスト方法に従って実施した 29 組の観測データ（t_i, x_i）（$i = 1, 2, ..., 29$）のデータセットを用意した（表 3）．これを打点し，グラフにしたものが図 4 である．ここで，t_i はテスト時間（実働日数），x_i は時刻 t_i のときの累積フォールト発見数である．またテスト時間 t_{29} は事前に用意したテストケースを全て消化したテスト時間であると同時に，実際の出荷日でもある．

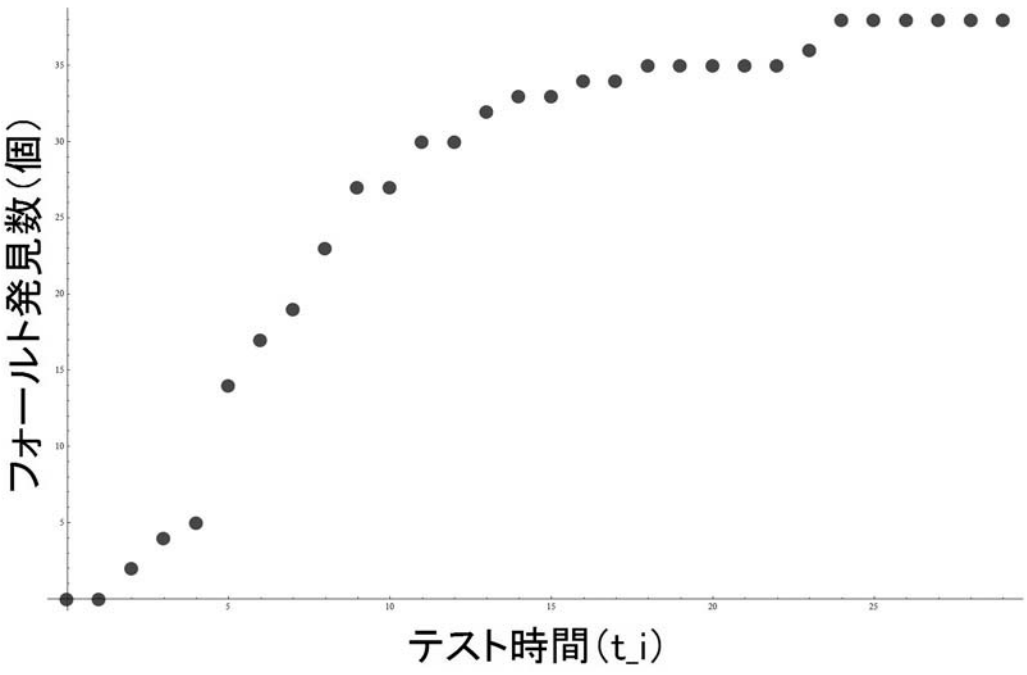

図 4: データセットを打点したグラフ.

表 3: 全てのデータセット.

i	テスト時間 (t_i; 実働日数)	合計フォールト発見数 (A チーム, B チーム)	累積フォールト発見数 (x_i)
0	0	0 (0, 0)	0
1	1	0 (0, 0)	0
2	2	2 (1, 1)	2
3	3	2 (2, 0)	4
4	4	1 (0, 1)	5
5	5	9 (9, 0)	14
6	6	3 (0, 3)	17
7	7	2 (2, 0)	19
8	8	4 (0, 4)	23
9	9	4 (0, 4)	27
10	10	0 (0, 0)	27
11	11	3 (0, 3)	30
12	12	0 (0, 0)	30
	↑ テスト一段階目終了	$d_{1A} = 14$, $d_{1B} = 16$	
13	13	2 (1, 1)	32
14	14	1 (0, 1)	33
15	15	0 (0, 0)	33
16	16	1 (1, 0)	34
17	17	0 (0, 0)	34
18	18	1 (1, 0)	35
19	19	0 (0, 0)	35
20	20	0 (0, 0)	35
21	21	0 (0, 0)	35
22	22	0 (0, 0)	35
23	23	1 (0, 1)	36
24	24	2 (0, 2)	38
25	25	0 (0, 0)	38
26	26	0 (0, 0)	38
27	27	0 (0, 0)	38
28	28	0 (0, 0)	38
29	29	0 (0, 0)	38
	↑回帰テスト 二段階目終了	$d_{2A} = 3$, $d_{2B} = 5$	

6.1 ソフトウェア内に潜在する総フォールト数の推定結果

4 章で述べたソフトウェア内に潜在する総フォールト数 N を表 3 のデータセットを用いてパラメータ推定を行った．この結果を表 4 に示す．なお，N は N_0 を四捨五入したものである（4.1 節を参照）.

表 4: 回帰テスト二段階目の各テスト時間における N の推定値.

テスト時間 (評価時間)(t_i)	$\widehat{N_A}$	$\widehat{N_B}$	N_0	N	累積フォールト発見数 (x_i)
13	15.1	17.1	32.1	32	32
14	16.1	17.1	33.3	33	33
15	16.1	17.2	33.3	33	33
16	16.3	18.3	34.6	35	34
17	16.3	18.3	34.6	35	34
18	16.4	19.5	36.0	36	35
19	16.4	19.5	36.0	36	35
20	16.4	19.5	36.0	36	35
21	16.4	19.5	36.0	36	35
22	16.4	19.5	36.0	36	35
23	17.7	19.8	37.5	38	36
24	20.4	20.4	40.7	41	38
25	20.4	20.4	40.7	41	38
26	20.4	20.4	40.7	41	38
27	20.4	20.4	40.7	41	38
28	20.4	20.4	40.7	41	38
29	20.4	20.4	40.7	41	38

累積フォールト発見数 x_i は表 3 を参照.

6.2 未知パラメータ a, b, c, d および平均全フォールト発見時刻の推定結果

次に，5.2 節で述べた式（5.20）を用いて平均フォールト発見数を推定し，式（5.22）を用いて平均全フォールト発見時刻を推定する．ここで，未知パラメータ a, b, c および d の推定には式（5.23）を用いる．平均全フォールト発見時刻の中央値および 90 %信頼区間の推定値を表 5 に示す．また，死滅過程モデルに含まれる未知パラメータ初期状態 K の値には 4.1 節にてその導出方法を示したソフトウェア内に潜在する総フォールト数 N の推定値を使用する．ここで，中央値（Median) とは（5.21）の分布関数の値が 0.5，つまり $B(t) = 0.5$ を満たす t の値である．また，後に示す図 6 のグラフが横軸と囲む面積比において左右を 50 %ずつに分ける値ということもできる．

表 5: 未知パラメータ a, b, c および c の推定値.

テスト時間 (評価時間)(t_i)	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	90% 信頼区間 (下限値)	中央値 (Median)	90% 信頼区間 (上限値)	E[T]
13	34.15	0.51	32.1	0.16	8.19	12.18	56.24	18.83
14	39.18	0.41	3.14	*	8.52	13.06	44.40	17.90
15	*	*	*	*	*	*	*	*
16	25.51	0.69	4.65	0.68	8.84	14.28	28.93	16.07
17	23.81	0.75	4.99	0.80	8.84	14.18	26.88	15.60
18	20.41	0.99	6.26	0.97	9.21	15.75	26.74	16.55
19	19.87	1.03	6.50	1.01	9.21	15.69	26.28	16.42
20	19.94	1.03	6.47	1.00	9.21	15.69	26.34	16.44
21	20.34	0.99	6.29	0.98	9.21	15.73	26.66	16.52
22	20.97	0.95	6.02	0.94	9.21	15.76	27.16	16.64
23	26.25	0.71	4.65	0.54	10.11	21.24	41.68	22.87
24	26.49	0.74	4.77	0.47	12.14	29.85	53.87	30.93
25	25.60	0.79	5.02	0.53	12.22	28.63	50.29	29.55
26	25.26	0.81	5.13	0.55	12.25	28.24	49.14	29.11
27	25.22	0.81	5.14	0.55	12.25	28.19	49.01	29.06
28	25.36	0.80	5.10	0.54	12.24	28.34	49.44	29.22
29	25.60	0.79	5.02	0.53	12.21	28.60	50.23	29.52

$*$ は推定不可.

6.3　平均フォールト発見数の期待値の推移

図 5 に，テスト時間 $t_{29} = 29$ のときに式（5.20）を用いて推定した平均フォールト発見数の期待値 $\mathrm{E}[N - X(t)]$ の推移（実線）を示す．打点は実際の観測データのデータセット（表 3 を参照）である．また，このときに使用した未知パラメータ N, a, b, c, d, 中央値および平均フォールト発見時刻 $\mathrm{E}[T]$ の期待値を表 6 に示す．ここで，平均全フォールト発見時刻 $\mathrm{E}[T]_{t=29} = 29.5162$ はテスト時間 $t_{29} = 29$ 時点までの全ての観測データを用いて推定した値である．なお 付録 A として，テスト時間 $t_{21} = 21$ から $t_{28} = 28$ までのそれぞれのときに推定した平均フォールト発見数の期待値 $\mathrm{E}[N - X(t)]$ の推移（実線) と実際の観測データ（打点）を図 $A.1$ から図 $A.8$ に加えておく．これらの図より，本研究で使用した死滅過程モデルが全てのテスト時間での推定においても実際の観測データに対して高い適合率を保っていることがわかる．

また，平均全フォールト発見時刻はソフトウェア内に潜在する全てのフォールト数が発見・修正される将来の予想時刻であることも再度確認しておいていただきたい（5.2.2 節を参照）.

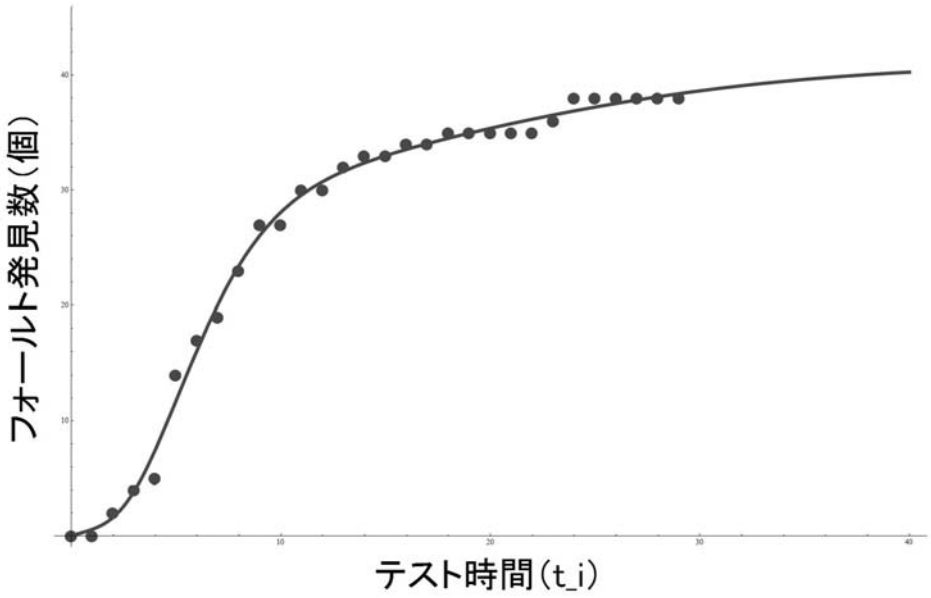

図 5: 平均フォールト発見数の期待値の推移（テスト時間 $t_{29} = 29$ のとき).

表 6: 未知パラメータの推定値（テスト時間 $t_{29} = 29$ のとき）.

N	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	中央値	$\mathrm{E}[T]$
41	25.598	0.78838	5.0179	0.52698	28.5952	29.5162

全ての値は表 4 および表 5 を参照.

図 6 に，テスト時間 $t_{29} = 29$ における平均フォールト発見数 $N - X(t)$ の密度関数の概形を，図 7 にこれを積分したもの，つまり分布関数の概形を示す.

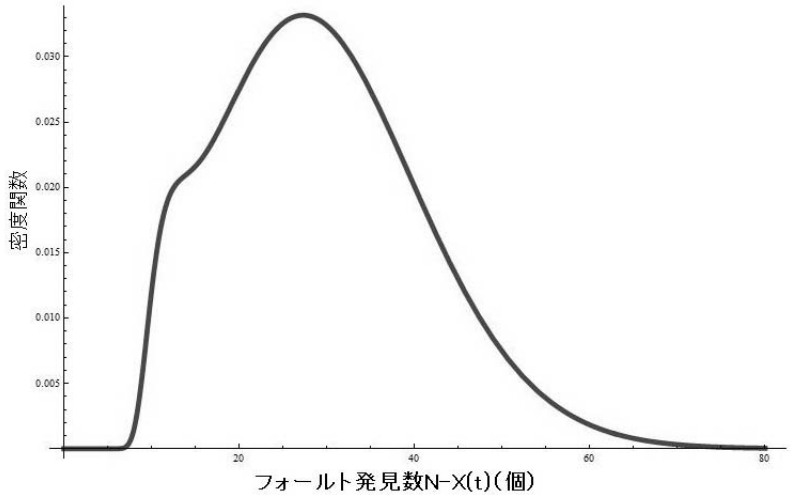

図 6: フォールト発見数の密度関数の概形（テスト時間 $t_{29} = 29$ のとき）.

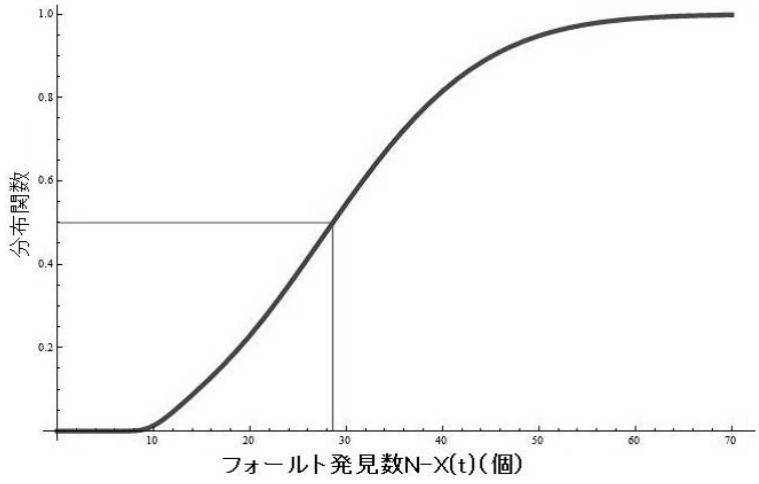

図 7: フォールト発見数の分布関数の概形とその中央値（テスト時間 $t_{29} = 29$ のとき）.

7　考察

本章では，前章にて実際のテスト活動から得た観測データの本手法への適用することによって得た推定結果を用いて，実際のテスト工程の定量的進捗管理とソフトウェアの信頼性評価のために，本手法がどの程度有効であるかについて述べる．

7.1　テスト経過に伴う期待残存フォールト数の推移

図 8 に，テスト時間 $t_{16} = 16$ から $t_{29} = 29$ のそれぞれのときにおけるソフトウェア内に潜在する総フォールト数の期待値および累積フォールト発見数の推移を示す．また，表 7 にそのときの推定値を示す．本節では，テスト時間 t_i のときにそれまでの観測データのみを用いて推定した総フォールト数の期待値 N_{t_i} とそのときの実際の累積フォールト発見数 x_i の数値の比較を行う．このことから，その差分よりあと何個のフォールト数を発見すればソフトウェア内に潜在する総フォールト数を発見できるか，つまり N までの必要フォールト発見数 $N - x_i$ を導出することが可能となる．

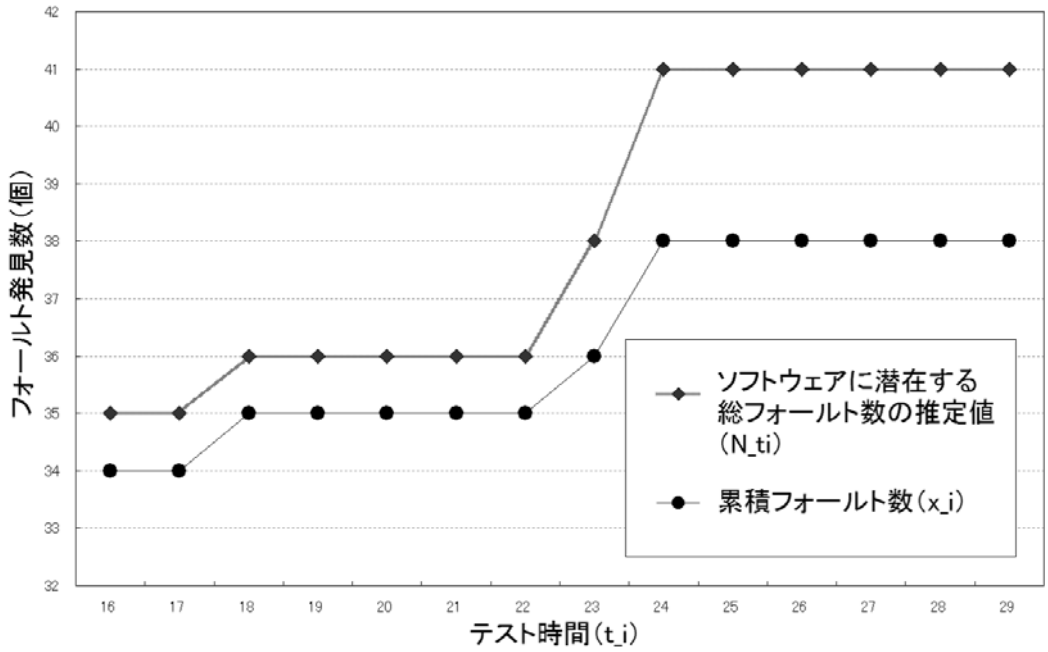

図 8: ソフトウェア内に潜在する総フォールト数の期待値と累積フォールト発見数の推移.

ここで，実際のソフトウェアの出荷日である $t_{29} = 29$ の後，ユーザー環境下において新たに 1 つのフォールトが発見された報告がある．つまり，本研究で扱っているソフトウェア内に潜在する総フォールト数の真値は 39 と考えることができる．このことから，テスト

表 7: ソフトウェア内に潜在する総フォールト数の期待値と累積フォールト発見数.

t_i	16	17	18	19	20	21	22	23	24	25	26	27	28	29
N	35	35	36	36	36	36	36	38	41	41	41	41	41	41
x_i	34	34	35	35	35	35	35	36	38	38	38	38	38	38
$N - x_i$	1	1	1	1	1	1	1	2	3	3	3	3	3	3

x_i は表 3，N は表 4 を参照.

の終了に近づく $t_{24} = 24$ から $t_{29} = 29$ において潜在する総フォールト数の推定値は真値より 2 個多く見積もっており，悲観的な（安全側の）評価を示していることがわかる.

7.2 全フォールト発見時刻の期待値と実際の出荷日との比較

図 9 に，テスト時間 $t_{16} = 16$ から $t_{29} = 29$ のそれぞれのときの平均全フォールト発見時刻 $\mathrm{E}[T]$ の期待値（実線）およびその中央値（点線）の推移を示す．推定値は表 5 を参照していただきたい．一例として，テスト時間 $t_{25} = 25$ におけるこの 2 つの推定値は，表 4 における $N_{25} = 41$ の推定値と表 3 の $x_1 = 1$ から $x_{25} = 25$ までの観測データのみを死滅過程モデルに適用して推定された平均全フォールト発見時刻 $\mathrm{E}[T]$ の期待値およびその中央値の値である.

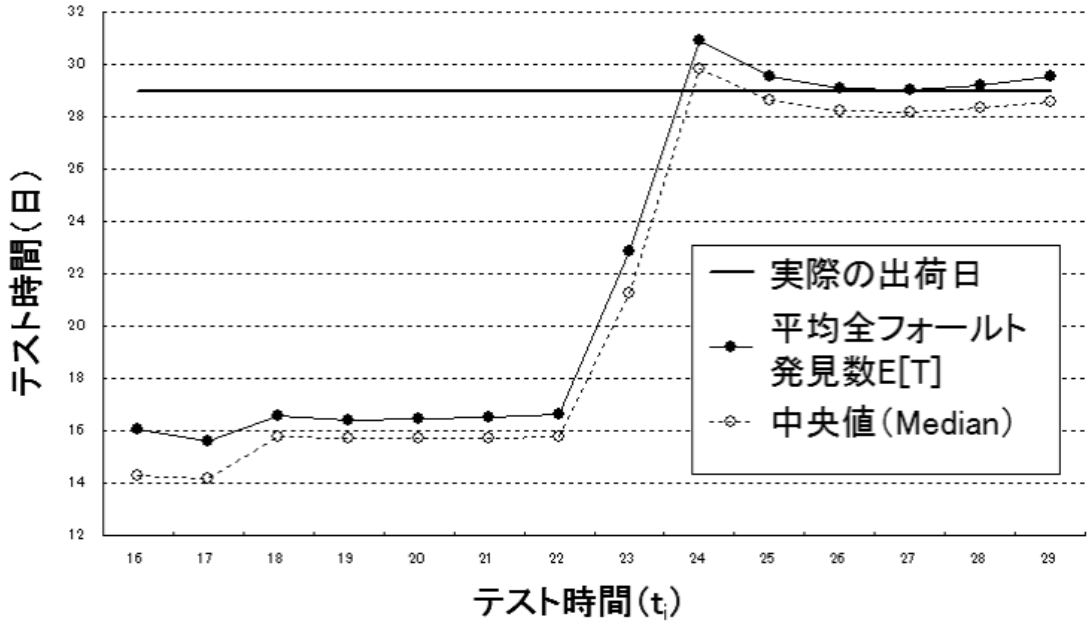

図 9: テスト時間毎の平均全フォールト発見時刻とその中央値の推移.

テスト時間 $t_{16} = 16$ から $t_{22} = 22$ において，推定値が 29 の下方で横ばいに推移しているのに対し，テスト時間 $t_{23} = 23$ と $t_{24} = 24$ の際には両方の推定値とも急激に増加していることがわかる．そして，テスト時間 $t_{24} = 24$ のときには，実際の出荷日である 29 日をテスト開始以来初めて上回っている．そして，テスト時間 $t_{25} = 25$ から $t_{29} = 29$ までにおいては，再び安定した推定値の推移を示している．これは，ある一定の値に収束していくと考えることができる．その値は，あるテスト時間 t_i において推定された全てのソフトウェア内に潜在するフォールト数を既に発見・修正しているとき，つまり $N_{t_i} - x_i = 0$ のときのテスト時間 t_i である．しかしながら，実際のソフトウェア開発においてテスト工程に費やせる時間やコストなどの資源は有限である．このことから，プロジェクトマネージャーはこれらの資源と許容できるソフトウェア信頼度を総合的に考慮した上でテストの終了を決定しなければならない．

また，図 9 において平均全フォールト発見時刻 $\mathrm{E}[T]$ の期待値とその中央値の推移を示す２つのグラフは全体を通して相対的に同じように推移していることがわかる．しかし，全てのテスト時間において中央値の方が平均全フォールト発見時刻 $\mathrm{E}[T]$ の値より楽観的な結果となっている．

7.3 全フォールト発見時刻の 90 %信頼区間の推移

次に図 10 に，図 9 で示した全フォールト発見時刻の中央値（Median）とその 90 %信頼区間の推移を示す．テスト時間 $t_{16} = 16$ から $t_{22} = 22$ までは上限値および下限値の差が平均 17.90 の幅で存在している．それに対してテスト時間 $t_{23} = 23$ 以降ではその差が平均 37.18 の幅に拡大している．全テスト時間において推定される全フォールト発見時刻の分散が大きいため 90 %信頼区間が広い傾向にあることがわかる（一例として 6.3 節の図 6 の分布関数（テスト時間 $t_{29} = 29$）の概形を参考）．

7.4 テスト経過に伴う残り必要テスト単位時間の推移

図 11 に，平均全フォールト発見時刻 $\mathrm{E}[T]_{t_i}$ の推定値とそのときのテスト時間 t_i を差分した値の推移を示す．また，表 8 にそのときの値を示す．この値は，それぞれのテスト時間のときの平均全フォールト発見時刻までの必要テスト単位時間である．

テスト時間 $t_{16} = 16$ から $t_{22} = 22$ において，その値がテストの経過につれて負の方向に増加している．これは，このテスト時間のときより過去の時間において，既に全ての推定されたソフトウェア内に潜在するフォールト数 N を発見されているということを示している．表 7 より，テスト時間 t_{16} から t_{22} におけるソフトウェア内に潜在する総フォールト数の期待値 N までの必要フォールト発見数 $N - x_i$ の値は全て 1 である．つまり，1 個のフォールトが未だソフトウェア内に残存していると推定しているのも関わらず平均全フォールト

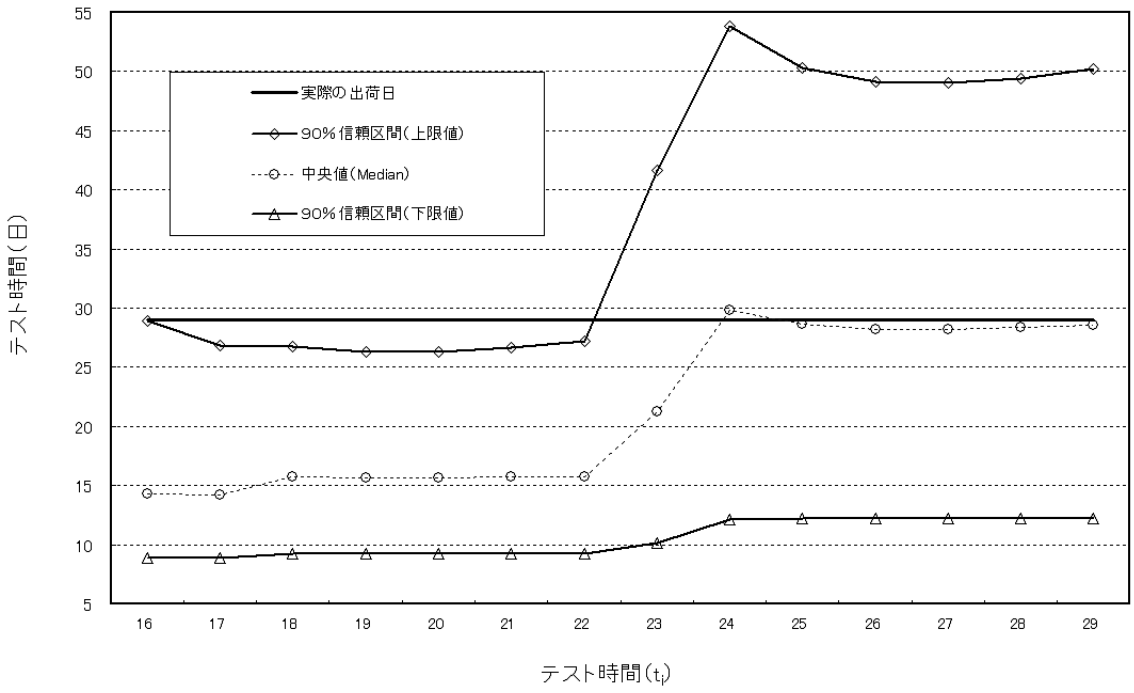

図 10: テスト時間毎の全フォールト発見時刻の中央値とその 90 %信頼区間の推移.

発見時刻は推定したテスト時間より過去の時刻（負の値）を示していることになる．これは，平均全フォールト発見時刻はソフトウェア内に潜在する総フォールト数が全て発見・修正されるときの時刻であるとう定義に矛盾している．この要因として，表 5 の 90 %信頼区間（下限値および上限値）の値，または図 10 からわかるように全フォールト発見時刻の推定値の分散が大きいことが挙げられる．しかしながらこの時点ではまだ事前に用意した実施するべきテストケースを全て消化していないので，この結果から，全てのソフトウェア内に潜在するフォールト数を発見・修正したとしてテストの終了を決定することはできない．したがって，テストを引き続き継続している．

そして，テスト時間 $t_{22} = 22$ から $t_{24} = 24$ にかけては，その値が正の値に急激に上昇している．これは，このテスト時間の間のフォールト発見数の値の変化が要因であると考えられる．表 3 より，テスト時間 $t_{18} = 18$ のとき以来フォールト発見数が観測されていないのに対して，$t_{23} = 23$ のときに 1 個，$t_{24} = 24$ のときに 2 個と短い時間に 3 個のフォールトが発見されていることがわかる．これが，テスト終了までの必要テスト単位時間を短期間で大きく伸ばしている．

その後テスト時間 $t_{25} = 25$ から $t_{29} = 29$ にかけては，± 0 の値に向けて収束していると予測することができる．これは，テスト時間 $t_{25} = 25$ 以降，表 3 からもわかるように，新しくフォールトが発見されていないため，テストの終了に近づいているためと考えることができる．

最後に，テスト時間 $t_{29} = 29$ は実際の出荷日であるが，このときの平均全フォールト発見時刻とテスト時間の差分は $\mathrm{E}[T]_{t_{29}} - t_{29} = 29.5162 - 29 = 0.516$（表 6 を参照）である．また，このときの累積フォールト発見数は $x_{29} = 38$（表 3 を参照），そしてソフトウェア内に潜在する総フォールト数の推定値は $N_{t_{29}} = 41$（表 4 を参照）である．このことからテスト時間 $t_{29} = 29$ において，それまでに発見・修正された 38 のフォールト発見数から，残存していると考えられる 3 個のフォールトを全て発見するためには，あと 0.516 テスト単位時間だけテストを継続する必要があるということがわかる．この数値は，ソフトウェア開発のプロジェクトマネージャーによってコストおよび納期を含めて総合的に考慮し，新たなテストケースを用意して引き続きテストの継続を行うか否かの判断指標の一助とすることができる．

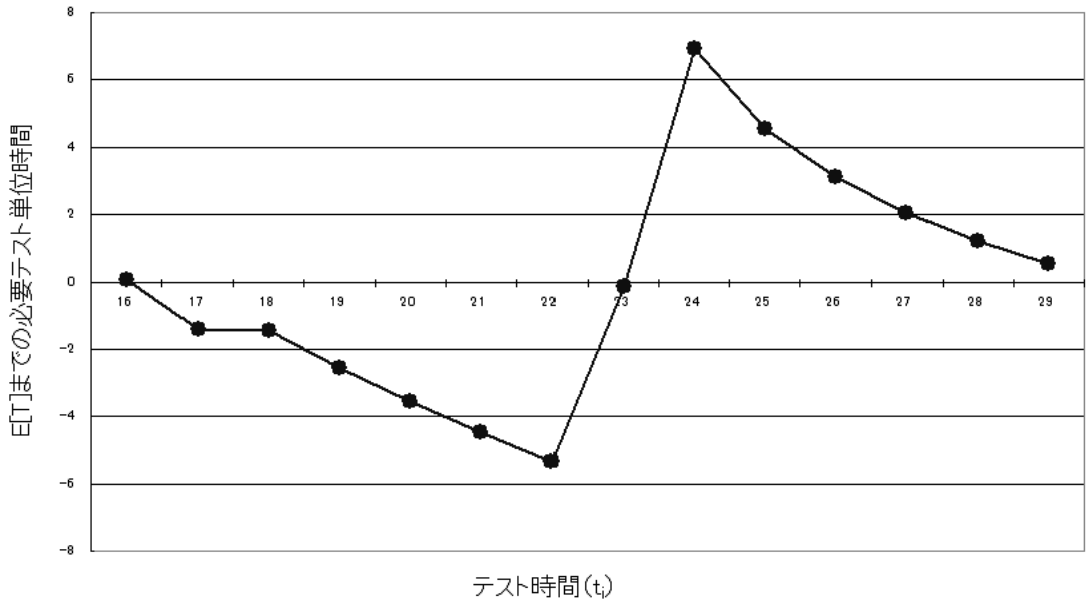

図 11: テスト時間毎の平均全フォールト発見時刻までの必要テスト単位時間の推移.

表 8: 平均全フォールト発見時刻までの必要テスト単位時間.

t_i	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$\mathrm{E}[T]_{t_i} - t_i$	0.7	-1.4	-1.5	-2.6	-3.6	-4.5	-5.4	-0.1	7.0	4.6	3.1	2.1	1.2	0.5

$\mathrm{E}[T]_{t_i}$ は表 5 を参照.

8　結論

本研究では，テストソフトウェア開発のテスト工程から得られた累積フォールト発見数の推移データを死滅過程モデルへ適用する手法を示し，本手法のテスト工程の定量的進捗度管理およびソフトウェア信頼性評価に対する有効性を実際の観測データを用いて示した．本手法で採用している死滅過程モデルをデータから推定するためには，初期状態パラメータとしてソフトウェア内に潜在する総フォールト数が必要である．このため，2 チームによる 2 段階ソフトウェアテストの方法を組み込んだ方法を提案した．実際の適用例として，任意のテスト時間でのソフトウェア内に残存するフォールト数とその全てのフォールト数の発見に必要な全フォールト発見時刻，およびその 90 %信頼区間の推定を行い，その推定結果を示した．推定結果より，ソフトウェア内に残存するフォールトを全て発見・修正するために必要なテスト時間を導出し，これがプロジェクトマネジメントにおけるテスト終了の意思決定の材料として有用な指標の一つとなり得ることがわかった．また，その推定値の精度はテスト工程が 86 %程度の進捗度（全テスト時間 $t_{29} = 29$ のうち $t_{25} = 25$ 時点）以降で高い結果を得られることがわかった．

また，本研究で考察したテストを担当したチームのフォールト発見率の数値も定量的な能力評価のための有用な指標の一つと考えられる．本研究で採用している 2 チームによる 2 段階ソフトウェアテスト方法によって，担当したチームのフォールト発見率，つまり潜在フォールトを逃さない能力が数値として導出できることがわかった．ソフトウェア開発が人的作業の積み重ねであることは周知の事実であり，この導出された能力値がプロジェクトマネジメントの観点より適切かつ定量的な人材配置の判断への一助となる可能性があることが言えよう．

しかしながら，今回のモデルでは，個々のフォールトがソフトウェアの品質に与える重要度，およびその発見から修正に必要とする時間を考慮していない．現実の問題として，これらを考慮した上でテスト工程の定量的な進捗度の管理およびソフトウェア信頼度の評価を行うことは必要である．このことから，個々のフォールトに対して修正に必要とする時間およびソフトウェアの品質への影響度の重み付けを考慮することが今後の課題として挙げることができる．これらの課題に取り組み本手法を展開していくことで，より安全で便利な我々の生活を支える高度情報化社会の発展に貢献できると考えている．

謝辞

本研究を進めるにあたり，終始懇切丁寧なるご指導をして下さいました，本学の工学研究科システム工学専攻，木村光宏教授に対し心より感謝致します．温かな御指導とともに数多くの貴重な御助言を頂いたのみならず，あらゆる面で７年に渡りお世話になり，本当に感謝しております．

最後に，多大なる協力をして頂いた研究室の先輩，ならびに研究室の同輩諸氏に感謝の意を表します．

参考文献

[1] 杉本淳一: 図解でわかる　ソフトウェア開発のすべて, Mint（経営情報研究会）(2000).

[2] 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（1994）.

[3] 山田茂, 高橋宗雄: ソフトウェアマネジメントモデル入門―ソフトウェア品質の可視化と評価法―, 共立出版（1993）.

[4] M. R. Lyu（ed.): Handbook of software Reliability Engineering, McGraw Hill, 1996.

[5] 山田茂: ソフトウェアの信頼性の基礎―モデリングアプローチ, 共立出版（2011）.

[6] Nelson, E. C. : "Estimaing software reliability from test data", Microelectronics and Reliabiity, Vol. 17, No. 1, pp. 67-74, 1978.

[7] J. G. Shanthikumar: "A general software reliability model for performance prediction", Microelectronics and Reliability, Vol. 21, No. 5, pp. 671-682（1981）.

[8] M. Xie : Software reliability modelling, World Scientific, Singapore, 1991.

[9] 山田茂: 「ソフトウェア信頼性モデルとその応用：ソフトウェア信頼性の定量的評価法と適用技術」, 日本応用数理学会誌（応用数理）, Vol. 5, No. 5, pp. 15-34, （1995）.

[10] 伊東晋弥，今井宏治, 安信那有子:「状態依存率を考慮した死滅過程によるソフトウェア進捗度評価モデルに関する考察」, 法政大学工学部経営工学科, 平成 20 年度卒業論文（2008）.

[11] 加藤淳: 「複雑システムの信頼性を向上させる開発手法－アーキテクチャ設計手法とモデル検査の融合－」, Synthesiology, Vol. 3, No. 3, pp. 197-212, （2010）.

[12] Myers, G. J. : The Art of Software Testing, John Wiley & Sons, New York, 1979.（松尾正信（訳）, ソフトウェア・テストの技法, 近代科学社, 1980）.

[13] 小野間彰, 山浦恒央訳: ソフトウェアテスト技法～自動化、品質保証、そしてバグの未然防止のために～, 日経 BP 社（1994）.

[14] 木村光宏, 藤原隆次: ソフトウェアの信頼性, 日科技連出版社, 東京, （2011）.

[15] J.W. Duran, J. J. Wiorkowski: "Capture-Recapture Sampling for Estimating Software Error Content", IEEE Transactions on Software Engineering, Vol. SE-7, No. 1, pp. 147-148，1981.

[16] H. D. Mills: “On the Statistical Validation of Computer Programs”, Technical Report FSC-72-6015, IBM Federal Systems Division, 1972. Technical Report, Science Applications Inc., 1973.

[17] S.L. Basin: “Estimation of Software Error Rates via Capture-Recapture Sampling”, Technical Report, Science Applications Inc. 1973.

[18] M. Kimura and T. Fujiwara: “A Note on Two-Stage Software Testing by Two Teams”, Proceedings of the International Conference on Advanced Software Engineering and Its Applications (ASEA2011), included in CCIS 257, Springer-Verlag Berlin Heidelberg, pp.330-337 (2011).

[19] M. Kimura and S. Yamada: “A software testing-progress evaluation model based on a digestion process of test-cases”, International Journal of Reliability, Quality and Safety Engineering, Vol. 4, No. 3, pp. 229-239 （1997）.

[20] 田口玄一編: 確率・統計, 日本規格協会（1981）.

付録 A　　平均全フォールト発見時刻の期待値の推移

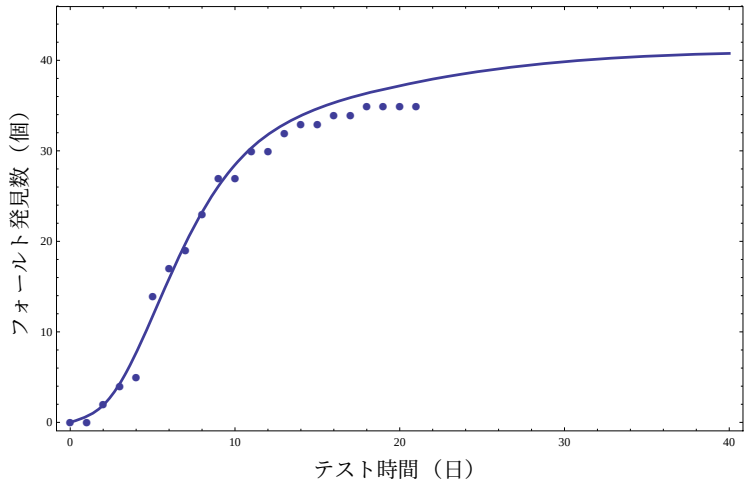

図 A.1：平均フォールト発見数の期待値の推移（テスト時間 $t_{21} = 21$ のとき）.
$[N = 36, \hat{a} = 20.34, \hat{b} = 0.99, \hat{c} = 6.29, \hat{d} = 0.98]$.

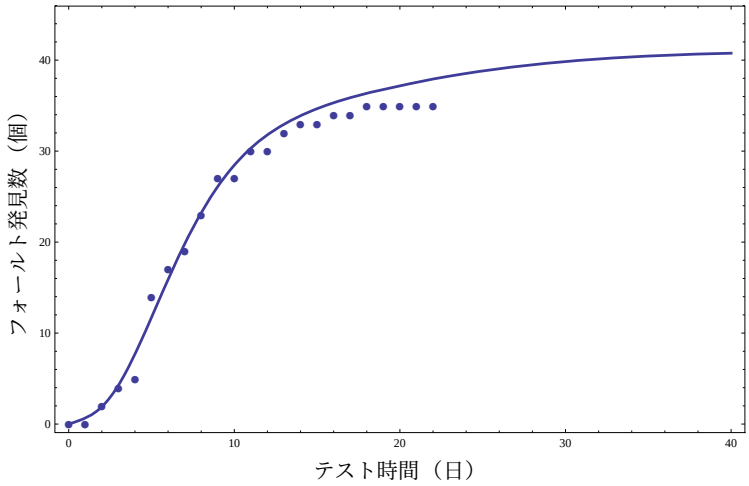

図 A.2：平均フォールト発見数の期待値の推移（テスト時間 $t_{22} = 22$ のとき）.
$[N = 36, \hat{a} = 20.97, \hat{b} = 0.95, \hat{c} = 6.02, \hat{d} = 0.94]$.

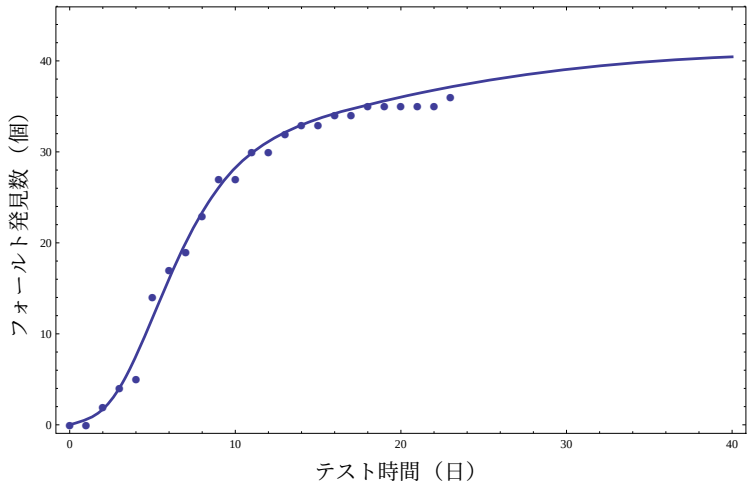

図 A.3：平均フォールト発見数の期待値の推移（テスト時間 $t_{23} = 23$ のとき）.
$[N = 38, \hat{a} = 26.25, \hat{b} = 0.71, \hat{c} = 4.65, \hat{d} = 0.54]$.

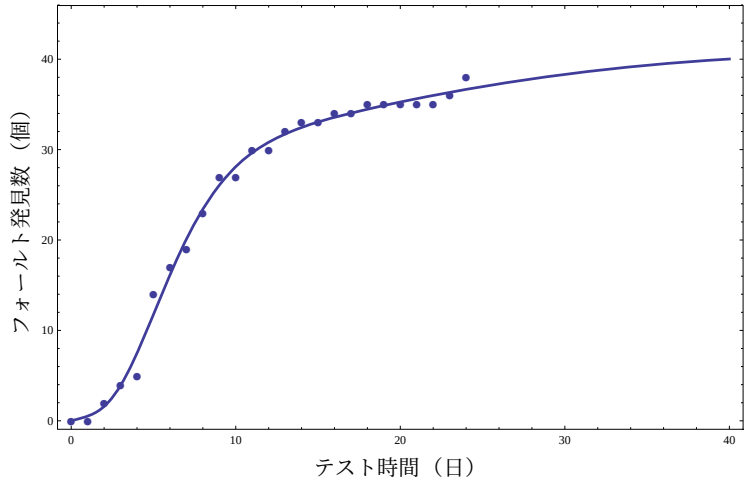

図 A.4：平均フォールト発見数の期待値の推移（テスト時間 $t_{24} = 24$ のとき）.
$[N = 41, \hat{a} = 26.49, \hat{b} = 0.74, \hat{c} = 4.77, \hat{d} = 0.47]$.

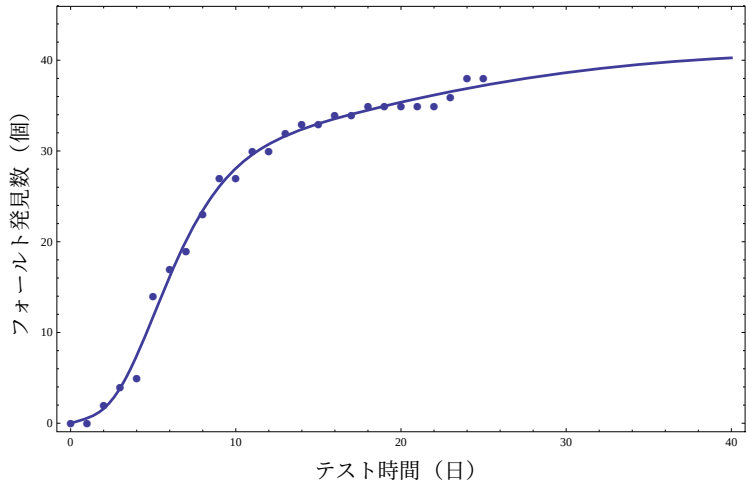

図 A.5：平均フォールト発見数の期待値の推移（テスト時間 $t_{25} = 25$ のとき）.
$[N = 41, \hat{a} = 25.60, \hat{b} = 0.79, \hat{c} = 5.02, \hat{d} = 0.53]$.

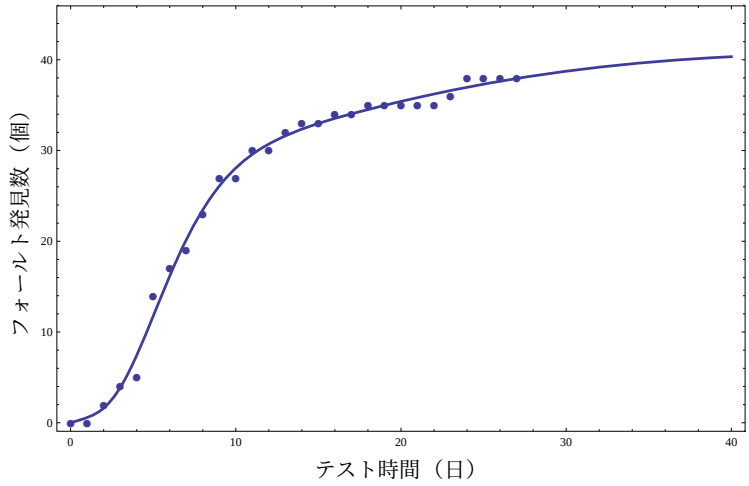

図 A.6：平均フォールト発見数の期待値の推移（テスト時間 $t_{26} = 26$ のとき）.
$[N = 41, \hat{a} = 25.26, \hat{b} = 0.81, \hat{c} = 5.13, \hat{d} = 0.55]$.

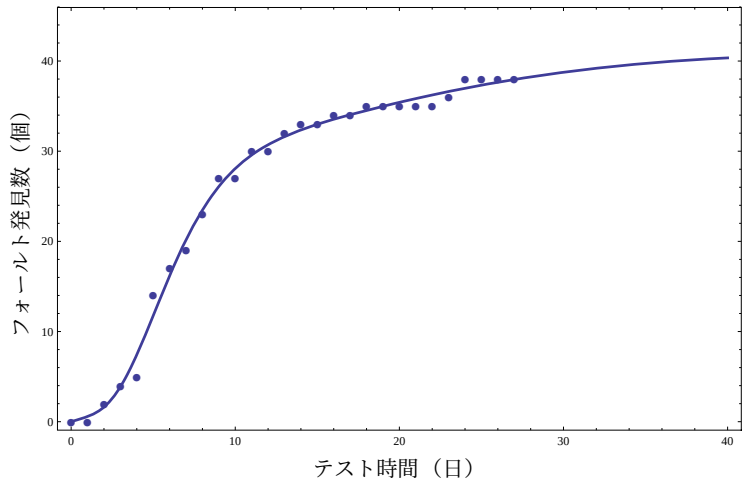

図 A.7：平均フォールト発見数の期待値の推移（テスト時間 $t_{27} = 27$ のとき）.
$[N = 41, \hat{a} = 25.22, \hat{b} = 0.81, \hat{c} = 5.14, \hat{d} = 0.55]$.

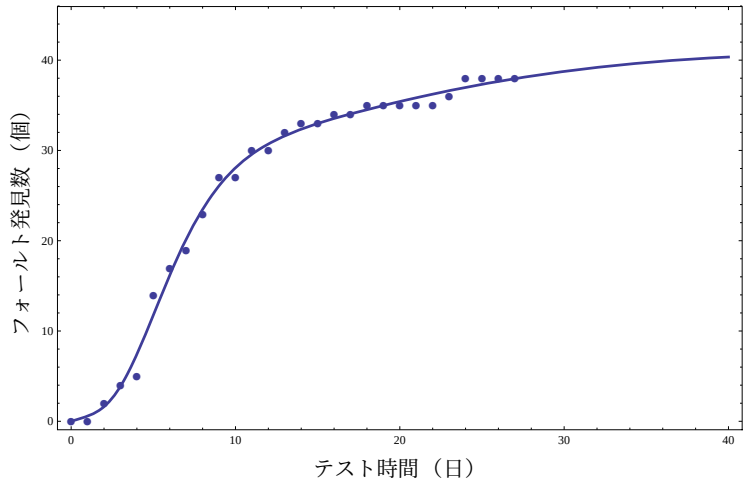

図 A.8：平均フォールト発見数の期待値の推移（テスト時間 $t_{28} = 28$ のとき）.
$[N = 41, \hat{a} = 25.36, \hat{b} = 0.80, \hat{c} = 5.10, \hat{d} = 0.54]$.