論文

ペトリネットによるプログラム開発演習のモデル化とそのモデルによるプログラマ作業効率の定量的評価

正員 楠本 真二[†]　正員 松本 健一[††]
正員 菊野 亨[†]　正員 鳥居 宏次[††]

Using a Petri-Net Model for Quantitative Analysis of Debugging Processes in Academic Environment

Shinji KUSUMOTO[†], Ken-ichi MATSUMOTO[††], Tohru KIKUNO[†] *and* Koji TORII[††], *Members*

あらまし 本論文では，プログラム開発演習を通じてプログラマの作業効率がどのように向上したかを定量的に評価する試みについて述べる．まず，フォールト除去（文法フォールトと論理フォールトの除去）プロセスでの作業内容を定量的に評価するため，フォールト除去プロセスをペトリネットモデルで記述する．提案するモデル記述ではペトリネットのトランジションにフォールト除去の基本作業を，プレースに状態をそれぞれ対応させる．各基本作業をプログラマが使用するコマンドの系列として認識し，データを収集する点にモデル化の特徴がある．次に，このモデルに基づいて四つの尺度：文法フォールト除去作業の繰返し回数（*Count SF*）と総時間（*Time SF*），論理フォールト除去作業の繰返し回数（*Count LF*）と総時間（*Time LF*）を導入する．これらの尺度はトランジションの発火回数と発火時間，プレースにトークンが置かれていた時間，等として計測される．更に，大学の情報工学演習（実験Ⅰと実験Ⅱ）を対象に適用実験を行った．その結果，文法フォールトと論理フォールト除去の作業効率に向上が見られること，および，それぞれの向上の中身に違いがあることが定量的に確認できた．

キーワード フォールト除去プロセス，定量的評価，ペトリネットモデル，プログラム開発演習

1. まえがき

大学，企業等でのソフトウェア教育では，演習が重要な位置を占めている．演習では，講義で習ったソフトウェア開発技法（例えば，アルゴリズムやプログラミングの設計手法，テスト，デバッグ技術）を具体的な課題に適用することによって，開発技法への理解を深め，応用力を高めることを目指す[9]．中でも，テスト，デバッグの技術は演習を通じて修得せざるを得ないのが現状である．従って，教官側は演習結果を評価することによって，受講者がこれらの技術を十分に理解しているか否かの判断をすることが必要になる．

† 大阪大学基礎工学部情報工学科，豊中市
Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan
†† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Advanced Institute of Science and Technology, Nara, Ikoma-shi, 630-01 Japan

筆者らの属している情報工学科においては時間的に連続して実施される2種類のプログラミング演習を通じて，受講者である学生の，テスト，デバッグ能力を効率良く向上させることを考えている．そのためには，(1)学生のプログラミング作業の進ちょく状況を把握し，(2)望ましい状況（目標）に対する現状のずれ（例えば，連続した演習を通じてのデバッグ能力の向上度）を評価し，(3)その評価結果に基づいて学生を指導するという一連の作業が必要になる[5]．ここでは研究の第1歩として，(1)の現状の把握と(2)の現状の評価に限って，そのための手法の提案と実験的評価について述べる．

これまでにもプログラミング作業の進ちょく状況を把握し，その現状を正確に評価しようという試みは報告されてきている．しかし，従来の報告の大半はプロダクト（例えば，受講生が作成したレポートやプログラム）の評価が中心になっており，開発プロセス[2]（すなわち，学生が実際に行った設計，コーディング，テス

トなどの具体的な作業内容）を評価するには至っていない．従って，プロダクトの評価結果が良い場合には問題はないが，評価結果が悪い場合に開発プロセスのどこに問題があるのかを特定することができず，学生への指導も困難である[9]．

このような問題を解決するためのアプローチとして最近注目を集めているものに，定量的なモデルに基づく開発プロセスの評価がある[1],[2]．定量的なモデルを用いることで，開発プロセスの明確な把握，および，客観的な評価が可能となり，開発プロセスに対する的確な改善指示が可能になる．そのモデルの例としてはテストプロセスを定量的に評価するソフトウェア信頼度成長モデル[10]（Software Reliability Growth Model, 以下では SRGM と略す）がある．SRGM ではテストプロセスにおけるソフトウェアプロダクト中の残存フォールト数を予測することによって，テストプロセスの進ちょく状況を間接的に把握，評価でき，テストを継続すべきかどうかの指示ができる．

先にも述べたように，本演習ではテスト，デバッグ能力の向上を最終的な目標とするので，SRGM に基づいた評価，指示だけでは必ずしも十分でない．そこで，本論文ではテスト作業の現状把握のための定量的モデルを新しく定義し，そのモデルに基づいた尺度の提案を行う．具体的には，学生のプログラム開発プロセスをペトリネットモデルで表現し，文法フォールトと論理フォールトの除去プロセスを評価するための尺度をそのモデル表現を利用して定義する．またモデルの適用のために必要となるデータは，我々が既に提案し，開発を進めている GINGER システム[6] を用いて自動収集する．

本論文の構成について述べる．まず，**2.** ではフォールト除去プロセス記述のためのペトリネットモデルの提案を行う．更に，本モデルに基づいて測定，導出可能な基本データ，評価尺度の検討を行う．**3.** では，フォールト除去プロセスの向上度，および，設計方法論を導入した場合の向上度への影響を評価するための適用実験について説明する．**4.** では，実験データを用いた分析を試み，最後に，**5.** ではまとめと今後の課題について述べる．

2. フォールト除去プロセス評価モデル

2.1 フォールト除去プロセス

フォールト除去プロセスで人間が計算機を使用して行う作業は，主に，「プログラムの編集」，「プログラムのコンパイル」，「実行プログラムを用いたテスト」の三つの作業である．「プログラム編集」作業ではプログラムテキストの作成，コンパイル時に検出された文法フォールト†（いわゆるシンタックスエラー）の除去，および，プログラムの実行時に検出された故障の原因となる論理フォールトの除去が行われる．次に「プログラムのコンパイル」作業では，コンパイラを使ってソースプログラムを実行プログラムに変換する．また，「実行プログラムを用いたテスト」作業では，実行プログラムにテストデータを与えて，プログラムが仕様どおりに作成されているかどうかを確認する．

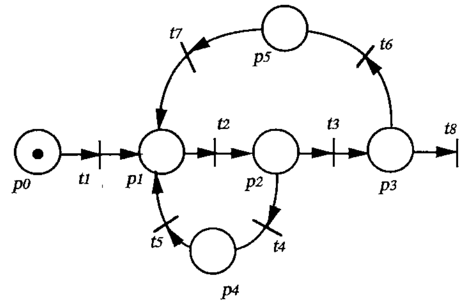


図1　ペトリネットを用いたモデル
Fig. 1 Debugging process model represented by Petri Net.

更に，「プログラムのコンパイル」作業中に文法フォールトが検出された場合は，そのフォールトがプログラムテキストのどこに存在するかを特定するための作業が必要となる．通常，コンパイラが出力するエラーメッセージを調べることで特定ができる．同様に，「実行プログラムを用いたテスト」作業中に故障が検出された場合は，その故障の原因となった論理フォールトの特定のための作業が必要となる．なお，この作業は故障が発生しなくなるまで繰り返される．

2.2 モ デ ル

2.1 で述べた作業の流れをペトリネット[7] で記述したものを図1に示す．同図においてプレースの集合 P は $P=\{p_0, p_1, p_2, p_3, p_4, p_5\}$ で，トランジションの集合 T は $T=\{t_1, t_2, \cdots, t_8\}$ である．なお，初期マーキングとしてプレース p_0 にだけトークンが1個割り当てられるものと仮定する．トランジションの発火，発火系列などの定義は通常のものに従うとする（例えば，文献(7)を参

† IEEE 標準の定義によると，与えられた問題を解くために必要な情報を理解する過程や，手法やツールを使用するための思考の過程でプログラマが犯す誤りをエラーと呼ぶ．一方，エラーがソフトウェア中に具体化したものがフォールトである．

照されたい)．トークンは直観的には一つのプロダクト(設計ドキュメント，プログラムコード等)に対応する．図1より明らかなようにこのペトリネットの発火によって各プレースにたかだか1個のトークンしか存在しない．

図1のモデル表現と実際のフォールト除去プロセスとの対応を以下に示す．直観的には，トランジションが作業に対応し，連続する二つの基本作業間に生じる待ち状態あるいはアイドル状態にプレースが対応する．

t_1：プログラム入力作業のためのエディタの起動

t_2：コンパイラの起動

t_3：プログラムの実行

t_4：文法フォールト特定のための作業(例えば，コンパイルリストの確認)

t_5：文法フォールト除去のためのエディタの起動

t_6：論理フォールト特定のための作業(例えば，デバッガの起動)

t_7：論理フォールト除去のためのエディタの起動

t_8：何も行われない

トランジション t_i の発火を t_i に対応する基本作業の実行と解釈する．従って，t_i の発火にある有限時間 Δ がかかるものと考える．同様に，プレース p_i にトークンが置かれれば p_i に対応する状態になったと解釈する．従って，一般に p_i に置かれたトークンは次の発火が起きるまでのある有限時間 Δ' の間，そのままの状態にあると考える．

p_0：設計作業が終了して，コーディング作業に移るまでのアイドル状態

p_1：プログラムの編集作業が終了して，コンパイルを行うまでのアイドル状態

p_2：コンパイルが終了して，プログラムを実行するまでのアイドル状態

p_3：プログラムの実行が終了した状態

p_4：コンパイルで検出された文法フォールトの特定作業が終了した状態

p_5：実行時に検出された故障の原因となる論理フォールトの特定作業が終了した状態

2.3 モデルのカスタマイズ

図1に示すモデルを初心者プログラマが行う大学の演習に適用する場合，トランジション t_6，t_7 に対応する基本作業を明確に区別して認識するのは困難である．その主な理由としては，初心者プログラマが論理フォールトを特定する場合，デバッガ等を使用することが少ないからである．通常，フォールト特定のためにエディタを起動し，特定後はすぐにフォールトの除去を行う．従って，トランジション t_6 と t_7 を識別するのは難しい．

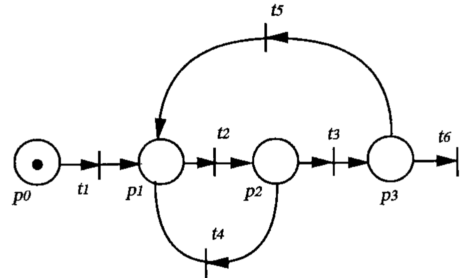


図2　簡単化したモデル
Fig. 2 Simplified Petri Net representation.

また，大学の演習で作成する程度の小規模プログラムに関しては文法フォールトの特定作業はコンパイラが出力するエラーメッセージを見るだけで十分であり短時間で終了する．従って，トランジション t_4 を識別するのは難しい．

以上の考察に基づいて以降では，図1のモデルを大学の演習用に簡単化した図2に示すペトリネットモデルを使用する．図2のモデルにおける，新しいトランジション t_4 と t_5 の意味を次に示す．それ以外のトランジションとプレースは図1中のものと同じである．

t_4：編集作業(具体的には，文法フォールトの特定と除去のための作業)

t_5：編集作業(具体的には，論理フォールトの特定と除去のための作業)

例えば，図2のモデルでトランジションの発火系列 $\delta_1 = t_1t_2t_4t_2t_3t_5t_2t_3t_6$ は次に示す一連の基本作業の系列に対応する．

プログラム編集，
コンパイル(ここで文法フォールトが検出される．)，
文法フォールトの除去，
(再び)コンパイル，
実行(ここで論理フォールトが検出される．)，
論理フォールトの除去，
コンパイル，
実行，
終了．

2.4 発火系列の例

ここでは，例としてモデル上のトランジションの発火系列 δ_2，δ_3，δ_4 を示し，それらがプログラマの典型的な振舞いに対応していることを述べる．

（1） 理想的な振舞い

$\delta_2 = t_1 t_2 t_3 t_6$

この系列 δ_2 で表される振舞いは文法フォールト，論理フォールトのいずれも発生しない場合の開発作業に対応する．熟練したプログラマによる小規模プログラムの開発がこの系列に相当すると考えられる．

（2） 初心者プログラマの振舞い

$\delta_3 = t_1 t_2 t_4 t_2 t_4 t_2 t_4 t_2 t_3 t_5 t_2 t_3 t_5 t_2 t_3 t_5 t_2 t_4 t_2 t_4 t_2 t_3 t_5 t_2 t_3 t_6$

この発火系列 δ_3 と δ_2 との違いは系列 $t_4 t_2$（$=\alpha$ とする）と $t_5 t_2 t_3$（$=\beta$ とする）が t_1 と t_6 の間に繰り返して挿入されている点にある．部分系列 α，β はそれぞれ文法フォールト，論理フォールトの除去に対応している．従って，発火系列 δ_3 は文法フォールトと論理フォールトを非常に多く作り込み，それらの除去作業を繰り返し行う初心者プログラマによる開発を表していると考えられる．

（3） ある程度の経験を積んだプログラマの振舞い

$\delta_4 = t_1 t_2 t_4 t_2 t_4 t_2 t_3 t_5 t_2 t_3 t_5 t_2 t_3 t_5 t_2 t_3 t_6$

この発火系列 δ_4 も δ_3 と同様，系列 α と β が t_1 と t_6 の間に繰り返して挿入されている．但し，δ_3 と違って，系列 α の出現回数が系列 β のそれに比べ少なくなっている．従って，発火系列 δ_4 は文法フォールトの除去作業はあまり行っていないが，論理フォールトの除去作業を繰り返し行った場合に対応する．ある程度慣れてきたプログラマの開発がこのような系列になると考えられる．

2.5 評価尺度

2.4 で述べたように，トランジションの発火系列 δ_3，δ_4 に含まれる系列 $\alpha = t_4 t_2$ と $\beta = t_5 t_2 t_3$ は，それぞれ，プログラマが行う文法フォールトと論理フォールトの除去作業に対応している．従って，プログラマの振舞いを表す発火系列 δ 中に現れる系列 α と β の回数，系列 α，β の実行に要する時間などを計測することで，プログラマのフォールト除去プロセスを評価することが可能である．

まず，プレース p_i とトランジション t_i に関して回数，時間を表す変数 C と T を次のように定義する．系列 δ は発火可能であると仮定する．

$C(p_i \mid \delta)$：発火系列 δ のもとで（すなわち，系列 δ に含まれるトランジションを先頭から順次発火していったとき），プレース p_i にトークンが置かれる回数

$C(t_i \mid \delta)$：発火系列 δ に含まれる t_i の総数

$T(p_i \mid \delta)$：発火系列 δ のもとで p_i にトークンが置かれていた時間の合計

$T(t_i \mid \delta)$：発火系列 δ のもとで t_i の発火に要した時間の合計

次に，文法フォールトと論理フォールトの除去の作業効率を評価する尺度として次の四つを定義する．

（1） 文法フォールト除去作業の時間 $Time\ SF(\delta)$

$Time\ SF(\delta)$ はトランジション t_4，t_2 に対応する作業の実行に要した時間 $T(t_4 \mid \delta)$，$T(t_2 \mid \delta)$ とプレース p_1，p_2 にトークンが置かれていた時間 $T(p_1 \mid \delta)$，$T(p_2 \mid \delta)$ を合計したものとなる．しかし，p_1 はアイドル状態であり，また，t_2 は計算機が行う作業であり，プログラマが働いていた時間ではないのでそれらに費やした時間は除く．そこで，

$$Time\ SF(\delta) = T(p_2 \mid \delta) + T(t_4 \mid \delta)$$

と定める．

（2） 文法フォールト除去作業の回数 $Count\ SF(\delta)$

系列 δ の中に含まれるトランジション t_4 の総数 $C(t_4 \mid \delta)$ が文法フォールト除去作業の実行回数に等しい．そこで，

$$Count\ SF(\delta) = C(t_4 \mid \delta)$$

と定める．

（3） 論理フォールト除去作業の時間 $Time\ LF(\delta)$

$Time\ LF(\delta)$ はトランジション t_5，t_2，t_3 に対応する作業に要した時間 $T(t_5 \mid \delta)$，$T(t_2 \mid \delta)$，$T(t_3 \mid \delta)$ とプレース p_1，p_2，p_3 にトークンが置かれていた時間 $T(p_1 \mid \delta)$，$T(p_2 \mid \delta)$，$T(p_3 \mid \delta)$ を合計したものとなる．しかし，$T(p_2 \mid \delta)$ は実行待ち状態なのでプログラマは働いていない．同様に，t_2，t_3 は計算機が行う作業であり，プログラマが働いていた時間ではないのでそれらに費やした時間は除く．そこで，

$$Time\ LF(\delta) = T(p_3 \mid \delta) + T(t_5 \mid \delta)$$

と定める．

（4） 論理フォールト除去作業の回数 $Count\ LF(\delta)$

系列 δ の中に含まれるトランジション t_5 の総数 $C(t_5 \mid \delta)$ が論理フォールト除去作業の実行回数に等しい．そこで，

$$Count\ LF(\delta) = C(t_5 \mid \delta)$$

と定める．

［注1］ フォールト除去作業の効率は与えられた問題の難しさにも依存する．すなわち，問題が難しいほどプログラムの開発作業は複雑になり，フォールトが発生しやすく，フォールト除去に必要な作業量も多くなる．従って，例えば文法フォールト除去作業の時間を，

$$Time\ SF(\delta) = (T(p_2 \mid \delta) + T(t_4 \mid \delta)) / f(q)$$

と定義する方がより妥当である．ここで，関数 f は問

題の難しさ q をパラメータとするある正規化関数とする．他の評価尺度についても同様である．但し，本論文で実施した二つの適用実験(酒屋問題と残金管理問題)の場合，問題の難しさはほぼ同じと判断できるので，$f(q)$ は特に考慮しないことにする．

3. 適用実験

3.1 実験の概要

提案したモデルの有効性を確認する目的で大阪大学基礎工学部情報工学科の学部2年生に対する二つのプログラム演習(実験I，実験II)を対象に適用実験を行った．学生は初心者プログラマとみなすことができる．但し，端末，キーボード，コマンド等の使用方法は学部1年生のときに行うプログラム演習で習得している．実験Iと実験IIは時間的に連続して実施され，これらの実験は同じ12名の学生に対して行われた．実験Iの終了後に，複合設計法[3],[4]の講義を行い，それを実験IIで利用した．

実験Iではいわゆる酒屋問題[11]に対するプログラムをC言語で作成する．但し，特に設計方法論は指定せず各学生に自由な形式で設計を行わせる．作成されるプログラムサイズは約300行である．

実験IIではスーパマーケットのレジの残金管理問題(文献(12)中の問題に若干の修正を加えたもの)に対するプログラムを作成する．複合設計法を用いてプログラムを設計し，作成した設計ドキュメントに基づいてプログラムをC言語で作成する．作成されるプログラムサイズは約300行である．

なお，実験は担当教官の指導のもとで行われ，計算機は限られた演習時間にしか基本的に使用できないため，演習時間に課題作成以外のことを長時間行うのは難しい環境になっている．更に，実験終了時に学生が作成したプログラムが与えられた仕様書を満足しているかどうかを確認するための受け入れテストを実施している．評価対象である12名の学生のプログラムはこの受け入れテストに合格している．

3.2 収集データ

データ収集には我々が提案し，開発を行ってきているGINGERシステム[6]を用いた．GINGERシステムでは計算機上でのコマンドの実行履歴，計算機使用時間の記録，などを自動収集できる．コマンド実行履歴としては実行された各コマンドごとに次の情報を含んでいる．

(1) コマンド名

```
1988 11 11 11 56  vi     1.38 secs
1988 11 11 11 58  cpp    0.19 secs
1988 11 11 11 58  hccom  2.47 secs
1988 11 11 11 58  cc     0.05 secs
1988 11 11 11 58  vi     1.95 secs
1988 11 11 12 01  cpp    0.17 secs
1988 11 11 12 01  hccom  2.42 secs
1988 11 11 12 01  cc     0.05 secs
1988 11 11 12 01  vi     4.61 secs
1988 11 11 12 07  cpp    0.20 secs
        •
        •
        •
```

図3 コマンド実行履歴データ
Fig. 3 Example of command execution data.

```
1988-10-29 10:53 12:34
1988-10-29 12:35 12:36
1988-10-29 12:37 12:37
1988-11-08 13:53 15:04
1988-11-11 11:48 13:17
1988-11-18 12:19 13:21
1988-11-19 10:17 12:05
        •
        •
        •
```

図4 計算機使用時間履歴データ
Fig. 4 Example of terminal access time data.

(2) コマンド実行が終了した時刻

(3) コマンド実行に要したCPU時間(1/100秒単位で計測した時間)

コマンド実行履歴の例を図3に示す．図3で例えば，

1988 11 11 11 56 vi 1.38 secs

は1988年11月11日11時56分にコマンドviの実行が終了し，実行に要したCPU時間が1.38秒であることを表す．

一方，計算機使用時間の記録としては各学生が計算機を使用するごとにその使用開始時刻と終了時刻を残している．計算機使用時間の記録の例を図4に示す．この図で例えば，

1988-10-29 10：53 12：34

は1988年10月29日10時53分から12時34分まで計算機が使用されたことを表す．

3.3 変数 C, T の測定方法

フォールト除去プロセスの効率を評価する上で必要となる4種類の変数 $T(t_i \mid \delta)$, $T(p_i \mid \delta)$, $C(t_i \mid \delta)$, $C(p_i \mid \delta)$ に関するデータを測定する方法について述べる．

本実験ではプログラム編集のためのエディタとして"vi"を用いている．一方，コンパイルにはコマンド"cc"を用い，プログラムの実行にはコマンド"a.out"を用いている．従って，トランジション t_1, t_4, t_5 の発火は

いずれもコマンド“vi”の実行に対応させることができる．同様に，トランジション t_2 の発火はコマンド“cc”の実行に，トランジション t_3 の発火はコマンド“a.out”の実行にそれぞれ対応させることができる[†]．

時間は計算機の累積使用時間を基準にして算出する．その理由は，計算機が各学生に1台ずつ割り当てられているため，作業のほとんどが計算機上で行われているからである．また，フォールト特定のための机上デバッグは計算機を使用中にしたままの状態で行われるため，その時間は計算機の累積使用時間に含まれている．

なお，コマンド実行履歴はコマンドの実行開始時刻を含んでいないので，実行開始時刻が必要な場合には，直前のコマンドの実行終了時刻を近似的に用いる．

以上より，各トランジション t_i の回数 $C(t_i \mid \delta)$ と時間 $T(t_i \mid \delta)$ を次に述べる方法で測定する．

$C(t_1 \mid \delta)$：値1を割り当てる

$C(t_2 \mid \delta)$：コマンド実行履歴中に含まれる cc の総数

$C(t_3 \mid \delta)$：コマンド実行履歴中に含まれる a.out の総数

$C(t_4 \mid \delta)$：コマンド実行履歴中に cc と vi がこの順番に連続して現れる回数

$C(t_5 \mid \delta)$：コマンド実行履歴中に a.out と vi がこの順番に連続して現れる回数

$C(t_6 \mid \delta)$：値1を割り当てる

$T(t_1 \mid \delta)$：コマンド実行履歴中に初めて現れた vi の実行時間

$T(t_2 \mid \delta)$：コマンド実行履歴中に含まれる cc の実行時間の合計

$T(t_3 \mid \delta)$：コマンド実行履歴中に含まれる a.out の実行時間の合計

$T(t_4 \mid \delta)$：コマンド実行履歴中に cc と vi が連続して現れている場合の各 vi の実行時間の合計

$T(t_5 \mid \delta)$：コマンド実行履歴中に a.out と vi が連続して現れている場合の各 vi の実行時間の合計

$T(t_6 \mid \delta)$：コマンド実行履歴中に最後に現れた a.out の終了時刻から作業終了時刻までの時間

次に，モデルの構成法より各プレース p_i の回数 $C(p_i \mid \delta)$ は次の関係式を利用して求める．

$C(p_0 \mid \delta)=1$

$C(p_1 \mid \delta)=C(t_1 \mid \delta)+C(t_4 \mid \delta)+C(t_5 \mid \delta)$

$C(p_2 \mid \delta)=C(t_2 \mid \delta)$

$C(p_3 \mid \delta)=C(t_3 \mid \delta)$

各プレース p_i の時間 $T(p_i \mid \delta)$ は次の方法で測定する．

$T(p_0 \mid \delta)$：作業開始時刻からコマンド実行履歴中に初めて vi が現れる時刻までの時間

$T(p_1 \mid \delta)$：コマンド実行履歴に連続して現れるコマンド vi，cc のすべてについての，vi の終了時刻から cc の開始時刻までの時間の合計

$T(p_2 \mid \delta)$：コマンド実行履歴中に連続して現れる2種類のコマンド cc，a.out および，コマンド cc，vi のすべてについて，cc の終了時刻から a.out の開始時刻までの時間，および，cc の終了時刻から vi の開始時刻までの時間を求めて，それらの時間の合計をとったもの

表1　実験データ (実験 I)
(但し，$C(p_i \mid \delta)$，$T(p_i \mid \delta)$ を $C(p_i)$，$T(p_i)$ と略記)

学生	$C(p_0)$	$C(p_1)$	$C(p_2)$	$C(p_3)$	$C(t_1)$	$C(t_2)$	$C(t_3)$	$C(t_4)$	$C(t_5)$	$C(t_6)$
A	1	98	97	53	1	97	53	44	53	1
B	1	91	91	44	1	91	44	47	43	1
C	1	82	81	49	1	81	49	32	49	1
D	1	50	50	34	1	50	35	15	34	1
⋮										
学生	$T(p_0)$	$T(p_1)$	$T(p_2)$	$T(p_3)$	$T(t_1)$	$T(t_2)$	$T(t_3)$	$T(t_4)$	$T(t_5)$	$T(t_6)$
A	6	1770	104	27	29	605	67	49	446	4
B	1	562	32	11	2	392	12	31	40	0
C	3	1070	51	62	5	547	21	39	462	1
D	5	483	49	100	39	116	110	25	227	0
⋮										

(単位：C()…回数，T()…分，但し，$T(t_2)$と(t_3)は秒)

表2　実験データ (実験 II)
(但し，$C(p_i \mid \delta)$，$T(p_i \mid \delta)$ を $C(p_i)$，$T(p_i)$ と略記)

学生	$C(p_0)$	$C(p_1)$	$C(p_2)$	$C(p_3)$	$C(t_1)$	$C(t_2)$	$C(t_3)$	$C(t_4)$	$C(t_5)$	$C(t_6)$
A	1	46	46	19	1	46	19	27	18	1
B	1	15	14	4	1	14	4	10	4	1
C	1	31	30	8	1	30	8	22	8	1
D	1	51	50	22	1	50	22	28	22	1
⋮										
学生	$T(p_0)$	$T(p_1)$	$T(p_2)$	$T(p_3)$	$T(t_1)$	$T(t_2)$	$T(t_3)$	$T(t_4)$	$T(t_5)$	$T(t_6)$
A	3	415	30	14	210	156	9	64	91	0
B	0	170	18	6	0	87	2	41	37	7
C	1	628	59	5	52	201	6	207	38	46
D	2	416	49	38	3	401	28	65	102	14
⋮										

(単位：C()…回数，T()…分，但し，$T(t_2)$と(t_3)は秒)

† コンパイル時にコマンド“cc”のオプションとして実行ファイル名を指定できる．今回の実験の場合は，実験の指導書で，実行ファイル名(例えば，酒屋問題ならば sake)を定めているので，計測時にはコマンド“a.out”と“指定された実行ファイル名”をトランジション t_3 の発火に対応させている．

表3 評価結果

	CountSF	CountLF	TimeSF	TimeLF	TimeSF/CountSF	TimeLF/CountLF
実験I	39	44	111	326	3.2	10.8
実験II	20	12	131	80	7.8	6.0

$T(p_3\,|\,\delta)$：コマンド実行履歴中に連続して現れるコマンドa.out，viのすべてについてのa.outの終了時刻からviの開始時刻までの時間の合計

実験I，実験IIから上述の方法で計測した結果を表1，表2に示す．なお，表中で $T(p_i)\,(i=0,\cdots,3)$，$T(t_j)\,(j=1,4,5,6)$ の単位は分であり，$T(t_2)$，$T(t_3)$ の単位は秒である．

4. フォールト除去プロセスの分析

4.1 作業効率の向上

実験I，実験IIにおける各学生のプログラム開発 δ に対する $Time\ SF(\delta)$，$Count\ SF(\delta)$，$Time\ LF(\delta)$，$Count\ LF(\delta)$，および1回当りの作業時間 $Time\ SF(\delta)/Count\ SF(\delta)$，$Time\ LF(\delta)/Count\ LF(\delta)$ の平均値を表3に示す．

まず，フォールト除去作業の実行回数について分析する．文法フォールト除去作業の実行回数 *Count SF* は実験IIでの回数が実験Iでの回数の約半分に減少している．論理フォールト除去作業の実行回数 *Count LF* は実験IIでの回数が実験Iでの値の約4分の1に減少している．

次に，フォールト除去作業に要した総時間について分析する．文法フォールト除去作業に要した総時間 *Time SF* は実験IIでの時間と実験Iでの時間にほとんど差はなく，実験IIの方が若干時間が長くなっている．論理フォールト除去作業に要した総時間 *Time LF* は *Count LF* と同様に約4分の1に減少している．

以上の結果より，文法フォールト除去についてはフォールト除去作業に要する時間にほとんど変化がない．これは，二つの実験が短期間で実施されたため，学生の文法知識がそれほど向上せず実験Iと実験IIで作り込まれた文法フォールト数がほぼ同じであったためと考えられる．しかし，実験IIでの文法フォールト除去作業の繰返し回数は減少している．従って，実験IIにおいては1回の作業でより多くの文法フォールトの除去がなされており，その意味での作業効率の向上が見られる．

一方，論理フォールト除去作業については，実行回数，時間の両方ともに非常に向上している．この向上の原因としては二つ考えられる．一つは実験Iにおいて同種のプログラムを1度作成したことによる学習効果である．もう一つの理由として実験IIで導入した設計方法論の影響が考えられるが，それについては次節で詳しく調べる．なお，論理フォールト除去の1回当りの作業時間 *Time LF/Count LF* はほとんど変化がない．これは，実験が短期間に実施されたため論理フォールトの除去技術という点では著しい向上がなされなかったためであると考えられる．

4.2 設計方法論の影響

4.1では，実験IIにおいてすべての学生が複合設計法を一様に利用したと仮定して分析を行っている．しかし，複合設計法の講義が行われてから実験IIが行われるまでの期間は短い．また，学生にとって，実験IIは複合設計法を実際に利用する初めての演習である．従って，厳密に言えば，すべての学生が複合設計法を同程度に理解し，利用できたとは言えない．複合設計法の利用が論理フォールト除去作業に影響を与えたかどうかをより詳細に分析するためには，実験IIにおける複合設計法の利用状況を調べる必要がある．

ここでは，複合設計法の利用状況を，設計工程におけるプログラム機能の洗い出しの度合によって評価することにする．すなわち，プログラムで実現すべき機能がコーディング前にどれだけ多く列挙されていたか，その度合によって複合設計法が正しく利用されていたかどうかを評価する．そして，その評価には複合設計法で作成される多くの設計ドキュメントのうち，モジュール階層構造図[3],[4]を用いる．モジュール階層構造図とは，プログラムを構成するモジュール群とそのモジュール間の上下関係，モジュール間でのデータのやりとりを階層的な図として表したものである．

モジュール階層構造図上にモジュールとして記述された機能を設計機能と呼ぶ．一方，最終プログラムテキスト上に具体的なモジュールや関数として実現された機能を実現機能と呼ぶ．複合設計法においてはプログラムテキストをモジュール階層構造図に基づいて作成するので，設計機能と実現機能の間には1対1の対応関係が存在するはずである．しかし実際には，設計段階の作業が十分でなかったり，その作業内容に誤りを含む可能性があり，1対1の対応関係が崩れる．従って，この崩れ(すなわち設計機能と実現機能の差)が大きくなればなるほど，複合設計法は正しく利用されて

表4　追加機能数 (#*App*)

学生	追加機能数 #App
A	5
B	1
C	1
D	4
E	3
F	0
G	0
H	1
I	1
J	4
K	2
L	2

表5　相関係数

	#App
TimeLF	0.76
CountLF	0.82

おらず，機能の洗い出しに関して複合設計法を利用しなかった場合と同じと考えることができる．

ここでは，設計機能と実現機能の差を表す評価尺度として追加機能数 (#*App*) を次のように定義する．

追加機能数 (#*App*)：モジュール階層構造図上には存在しないが，プログラムテキスト上に存在する機能の数．

表4に各学生の追加機能数 (#*App*) を示す．更に，表5に #*App* と *Time LF*，および，*Count LF* との間の相関係数[8] を示す．表5より #*App* と *Time LF* との間の相関係数は 0.76 である．#*App* を説明変数，*Time LF* を基準変数と考えたときの #*App* の *Time LF* に対する決定係数 (関与率)[8] は 0.58 (＝0.76×0.76) となる．同様に，#*App* と *Count LF* との間の相関係数は 0.82 であり，#*App* を説明変数，*Count LF* を基準変数と考えたときの #*App* の *Count LF* に対する決定係数 (関与率) は 0.67 (＝0.82×0.82) となる．従って，複合設計法の利用が論理フォールト除去プロセスの作業効率の向上に大きく影響していることがわかる．関与率から見ても，複合設計法の利用が，作業効率向上の最も大きな原因と考えられる．但し，それが作業効率向上の唯一の原因とは言えない．他の原因としては，**4.1** で述べたように，学生の学習効果などが考えられる．

5. むすび

ソフトウェア開発におけるフォールト除去プロセスを定量的に評価するための数学的モデルの提案を行った．次に，GINGER システムで自動的に収集可能なコマンドの履歴データに注目して，本モデルに基づいた定量的評価法を検討した．更に，提案した方法を大学における学生実験に適用し，プログラマのフォールト除去プロセスの向上を定量的に評価した．

容易にわかるように，今回利用したペトリネットは有限状態機械と等価である．今後，複数のプログラマがチームでプログラムを開発する場合を考えると，プロセスの並行動作，同期等を考慮することが必要になる．そうした場合へのペトリネットモデルの拡張についての検討は今後の重要な課題の一つである．

更に残されている重要な課題としては，文法フォールト，および論理フォールトの除去プロセスを改善するために，分析結果を開発者である学生にいかにフィードバックするかについて検討することがある．一つの方法として，4種類の尺度による評価結果と測定された向上の度合あるいは発火系列のパターンに基づいて，適切な情報を選択しそれを学生に指示することが考えられる．例えば，今回の実験IIにおいて *Time SF*(δ)，*Count SF*(δ)，*Count LF*(δ) については向上しているが，*Time LF*(δ) についてはあまり向上が見られない学生に対して，デバッガ使用法の習得を指導すること，等が考えられる．今後は分析結果に基づいて適切な指導内容をフィードバックするための体系的な方法の確立を目指す予定である[1],[2],[6]．

謝辞　本研究は一部，平成4年度文部省科学研究費補助金重点領域研究 (課題番号 02249106) の補助を受けている．

文　献

(1) Basili V. R. and Musa J. D.: “The future engineering of software: A management perspective”, IEEE Computer, **24**, 9, pp. 90-96 (1991).

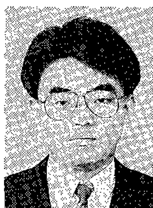
(2) Basili V. R. and Rombach H. D.: “The TAME project: Towards improvement-oriented software environment”, IEEE Trans. Software Eng., **14**, 6, pp. 758-773 (1988).

(3) Myers G. J.: “Composite/Structured Design”, VAN NOSTRAND REINHOLD (1978). “ソフトウェアの複合/構造化設計”, 國友義久, 伊藤武夫訳, 近代科学社 (1979).

(4) Myers G. J.: “Reliable Software Through Composite Design”, Mason/Charter Publishers, Inc. (1975). “高信

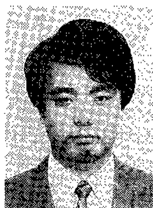
頼性ソフトウェアー複合設計", 國友義久, 久保未沙訳, 近代科学社 (1976).
(5) 楠本真二, 松本健一, 菊野 亨, 鳥居宏次: "開発者の能力向上を支援するプロジェクト管理の試み", ソフトウェア・シンポジウム '91 予稿集, pp. D2-D9 (1991).
(6) Kusumoto S., Matsumoto K., Kikuno T. and Torii K.: "On a measurement environment for controlling software development activities", Trans. on IEICE, **E74**, 5, pp. 1051-1054 (1991).
(7) Peterson J. L.: "Petri Net Theory and The Modeling of Systems", Prentice-Hall (1981). 市川惇信, 小林重信訳: "ペトリネット入門—情報システムのモデル化—", 共立出版 (1987).
(8) 芝 祐順, 渡部 洋, 石塚智一編: "統計用語辞典", 新曜社 (1984).
(9) 都倉信樹: "情報処理教育における実験・演習", 情報処理学会誌, **32**, 10, pp. 1101-1108 (1991).
(10) 山田 茂: "ソフトウェアの品質評価に関する考え方と動向—ソフトウェア信頼度成長モデルに基づく定量的品質評価法", 情報処理学会誌, **32**, 11, pp. 1189-1202 (1991).
(11) 山崎利治: "共通問題によるプログラム設計技法解説", 情報処理学会誌, **25**, 9, p. 934 (1984).
(12) 情報処理セミナー編集部: "第1種情報処理技術者試験問題全集", pp. 90-93, 新紀元社 (1986).

(平成4年9月29日受付, 5年4月16日再受付)



楠本 真二

昭63阪大・基礎工・情報卒. 平3同大大学院博士課程中退. 同年同大・基礎工・情報・助手. ソフトウェアの生産性や品質の定量的評価, プロジェクト管理に関する研究に従事. 情報処理学会, IEEE各会員.



松本 健一

昭60阪大・基礎工・情報卒. 平1同大大学院博士課程中退. 同年同大・基礎工・情報・助手. 平5奈良先端科学技術大学院大学・助教授. 工博. ソフトウェア開発における計測環境, ソフトウェア品質保証の枠組みに関する研究に従事. 情報処理学会, IEEE各会員.

菊野 亨

昭45阪大・基礎工・制御卒. 昭50同大大学院博士課程了. 同年広島大・工・講師. 同大助教授を経て, 昭62阪大・基礎工・情報・助教授. 平2阪大・基礎工・情報・教授. 工博. 主に分散処理システム, VLSI向きアルゴリズム, 組合せ最適化問題の解法に関する研究に従事. 情報処理学会, ACM, IEEE各会員.

鳥居 宏次

昭37阪大・工・通信卒. 昭42同大大学院博士課程了. 同年電気試験所(現電子技術総合研究所)入所. 昭50ソフトウェア部言語処理研究室室長. 昭59阪大・基礎工・情報・教授. 平4奈良先端科学技術大学院大学・教授. 工博. ソフトウェア工学の研究に従事. 情報処理学会, 日本ソフトウェア学会, 人工知能学会, ACM, IEEE各会員.