フトウェア開発プロセス　並行実行に対する拡張一般化確率ネットに基づく評価

水野 修　楠本 真二　菊野 亨

大阪大学 大学院基礎工学研究科 情報数理系専攻
〒 560 大阪府豊中市待兼山町 1-3
Phone: 06-850-6567　　Fax: 06-850-6569
e-mail: o-mizuno@ics.es.osaka-u.ac.jp

実際のソフトウェア開発では　前の工程の作業が完全に終了しないうちに　次の工程の作業が見切り発車的に並列実行されることが多い．こうした並列実行をした場合のソフトウェア品質への影響についてはほとんど議論がなされていない．我々は拡張一般化確率ペトリネットを用いてソフトウェア開発プロセスを形式的にモデル化し，シミュレーションによって評価する方法を提案している．本稿では，このモデルを開発プロセスの並列実行に適用し，開発期間，残存フォールト数への影響を定量的に評価する．

キーワード: 並列実行，ソフトウェア品質，拡張一般化確率ペトリネット，シミュレーション

Application of Extended Generalized Stochastic Petri-net Model to Parallel Execution of Design and Coding Activities

Osamu Mizuno, Shinji Kusumoto, Tohru Kikuno

Department of Informatics and Mathematical Sciences
Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560, Japan
Phone: 06-850-6567　　Fax: 06-850-6569
e-mail : o-mizuno@ics.es.osaka-u.ac.jp

In actual development, parallel executing of design and coding activities are usual. Since the coding activity is forced to start its execution with the incomplete design specification, some serious effects occur to the overall development process. we have already presented quantitative evaluations of the software development process described an extended Generalized Stochastic Petri-net model. In this paper we apply the model to the parallel executions of design and coding activities and evaluate the variations of the cumulative efforts and the number of residual faults.

Keywards: parallel execution, software quality, Generalized Stochastic Petri-net, simulation

1 えがき

大規模化, 複雑化が進むソフトウェアの開発において品質不良, 納期遅延, 予算超過といった問題の解決が求められている. これまでにドキュメントやソースコードなどのプロダクトやその生成過程である開発プロセスを評価し, その結果に基づいてプロダクトや開発プロセスの改善を行う研究が活発に行われている [11][12].

ところで実際のソフトウェア開発の現場では, 前の工程の作業 (例えば詳細設計) が完全に終了しないうちに, 次の工程の作業 (コーディング) が見切り発車的に開始され, これら 2 つの作業が並列実行されることが多い. こうした並列実行をした場合の品質, 納期, 予算への影響についてはほとんど議論がなされていない.

一方, ソフトウェア開発の全過程を通して品質, コスト, 開発期間の各項目を統合的に管理する, いわゆるソフトウェアプロジェクト管理の考え方が注目されている. しかし, プロダクトの特性 (開発対象となるソフトウェアのアプリケーション分野, 規模等) やプロセスの特性 (開発方法, 作業者, 期間, コスト等) はプロジェクト毎に大きく異なるため, プロジェクト管理者の経験や勘に基づいて試行錯誤的にプロジェクト管理が行われているのが現状である [5]

我々はこれまでに, 客観的かつ定量的なプロジェクト管理手法を確立することを目指して, 一般化確率ペトリネットを用いてソフトウェア開発プロセスを形式的にモデル化し, 評価する方法を提案してきている [7][10]. 提案モデルでは作業量という新しい概念を導入することによって, 開発の進行に応じて変動する開発期間の大きさやプロダクトの成長を表現することが可能にしている.

本稿ではこの一般化確率ペトリネットを用いてソフトウェア開発の並列実行をモデル化し, シミュレーションによって品質, 納期への影響について定量的に評価する.

2 ペトリネットモデル

2.1 一般化確率ペトリネット (GSPN)

ペトリネットの構成要素にはトークン, プレース, トランジション, そしてプレースとトランジションを結ぶアークがある. これまでにペトリネットに対する種々の拡張が報告されている.

まず, 確率ペトリネットは, トランジションの発火が可能になってから実際に発火するまでに遅延を許し, その遅延を表す指数分布確率変数を各トランジションに関連づけたペトリネットである.

次に, 一般化確率ペトリネット (Generalized Stochastic Petri-net:以降 GSPN と呼ぶ) は通常のペトリネットと確率ペトリネットの概念を融合して一般化したペトリネットであり, 発火に時間がかからない瞬間 (即時) トランジションと発火に時間がかかる時間トランジションの 2 種類のトランジションがある [9]. 時間トランジションの発火時間は指数分布に従うが, 各時間トランジションにそれぞれ固有の発火レートを割り当てることで発火に要する平均時間を変化させることができる.

この GSPN をソフトウェア開発プロセスのモデル化に用いることにより, 開発に要する時間やコストを評価することができる. しかし, ソフトウェア開発におけるもう 1 つの重要な管理項目である品質を評価することはできない.

2.2 拡張 GSPN [3]

拡張 GSPN では, 開発の進行にともなって変化するプロダクト中のフォールト数を評価するために GSPN に対して次の 2 点の拡張を行なっている.

(1) トークンに対する属性の導入.
トークンは, 開発作業の実行中に変化するプロダクトサイズやフォールト数, 作業の進捗度の値を属性として保持する.

(2) トランジションに対する発火実行関数の導入.
各トランジションには, その発火毎に評価される発火実行関数を与えることができる. この関数により, プロダクトサイズやフォールト数といったトークンの属性値の変化を記述することができる.

3 階層的モデル

3.1 作業量

従来, 作業の大きさを表すために工数が用いられている. 工数は, 作業に従事した作業者の数と要した時間の積で表される. しかし, この値は作業が終わった時点で分かる値では容易に求まるが, これから行なおうとする作業の場合には状況予測がつきにくいので, その値をあらかじめ予測して求めるのは難しい.

そこで, 標準的な作業者が 1 人で作業を行なったときに必要な総時間数をその作業の作業量と定義し, この状況における作業効率を 1 と定める. 作業効率は作業を行なう人数や各作業者の能力, 使用するツールの性能, といった作業が行なわれる条件により変化するものとする. このように定めると, 作業時間は作業量を作業効率で割った値として求めることができる.

作業量はある作業の大きさを作業条件に依らず一意に定めることができ, 同一作業に対する様々な条件のもとでの作業時間を計算することができる.

提案するモデルでは, 作業への入力となるプロダクトに依存して作業量が決定されるものとする. 例えば,

コーディング作業であれば入力となる設計仕様書のサイズが大きいほど作業量は大きくなる．デバッグ作業であれば入力プロダクトに含まれるフォールト数が多いほど作業量は大きくなる．また，作業の進行は達成した作業量の増加という形で表される．達成した作業量の増加にしたがって出力プロダクトのサイズやフォールト数を変化させることでプロダクトの成長をモデル化できる．

3.2 モデルの概略

提案するモデルは，プロジェクトモデルとプロセスモデルの 2 つからなる階層的モデルである．図 1にその概略を示す．

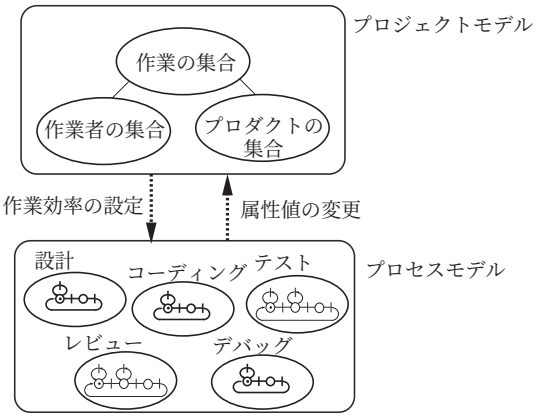


図 1: モデルの概略

上位のプロジェクトモデルでは，ソフトウェア開発の諸要素を作業，作業者，プロダクトの 3 つの概念に集約し，それぞれに対して様々な属性を付加している．

下位のプロセスモデルは，設計，コーディング，レビュー，テスト (単体テスト，結合テスト，受け入れテスト等)，デバッグ，等個々の作業モデルの集合であり，それぞれの作業モデルは拡張 GSPN を用いて記述される．

提案する階層的モデルでは，プロジェクトモデルと作業モデルが相互に情報を交換していくことでプロジェクトの進行をモデル化することができる．具体的には，ある一定の時間間隔で次の 3 つの手順を実行することによりプロジェクトモデルの属性値の更新が行なわれる．

Step 1. プロジェクトモデルの作業，プロダクト，作業者の各属性値に基づいて，実行する作業の作業効率を求める．

Step 2. 求まった作業効率の情報を対応する作業モデルに渡した後，作業モデルに従って一連の作業を実行する．

Step 3. 作業の実行結果をプロジェクトモデルに返す．プロジェクトモデルはその情報に基づいて作業とプロダクトの属性値を更新する．

3.3 プロジェクト モデル

プロジェクトモデルは作業，プロダクト，作業者の 3 つに着目し，プロジェクトをこれらの要素の集合としてモデル化したものである (表 1)．各要素にはそれぞれの概念を特徴づける属性を付加する．

作業には 6 種類の属性 (種類，開始/終了条件，入力/出力プロダクト，作業者，予定終了期日，作業量) を与える．

(1) 種類 ($type$) はその作業が設計，コーディング，レビュー，テスト，デバッグのうちのどれかを示す．

(2) 開始条件 ($entry\ c.$) と終了条件 ($exit\ c.$) はその作業の開始と終了のための条件をそれぞれ表す．

(3) 入力プロダクト ($input$) はその作業の入力となるプロダクト名とそのプロダクトが作業量の値の決定にどの程度影響を与えるのかを示す重みの 2 項組で表される．同一作業に対して 2 項組は複数指定してもよいものとする．一方，出力プロダクト ($output$) は生成，更新されるプロダクトとそのプロダクトに対する重みを 2 項組として与える．なお 2 項組が複数個存在する場合は，作業で得られるプロダクトのサイズやフォールト数の変化量を，この重みに従ってそれぞれのプロダクトに割り振る．従って，各プロダクトに対する重みの和は 1 である．

(4) 作業者 ($workforce$) の項では，どの作業者がどの程度の割合でその作業に携わるかを表す．すなわち，作業に携わる作業者とその作業者の稼働率 (その作業に携わることができる時間の割合) の 2 項組の集合として表す．

(5) 予定終了期日 ($deadline$) は開発計画で定められた作業の終了期日を表す．

(6) 作業量 ($workload$) では，その作業に割り当てられた作業量とそれまでに達成された作業量 (達成作業量) の 2 項組で表す．

プロダクトには 3 種類の属性: (1) プロダクトのサイズ ($size$)，具体的にはドキュメントのページ数やソースコードの行数を表す，(2) プロダクトに含まれている残存フォールト数 ($fault$)，(3) そのプロダクトに必要な記述事項の完全さを割合で表した完成度 ($c.rate$)，を与える．ここでプロダクトの完成度は記述事項の正しさには関係なく，必要とされる記述事項の漏れのなさを表すものとする．

作業者には経験レベル ($level$) を属性値として与える．経験レベルはソフトウェア開発の経験年数に応じて作業者を初級，標準，上級の 3 つに分類し，それぞれを 1,2,3 に数値化したものである．

このプロジェクトモデルを利用した記述例を表 2に

表 1: プロジェクトテンプレート

作業 A_iの属性	
種類	$type$
開始条件	$entry\ c.$
終了条件	$exit\ c.$
入力プロダクト	$input$
出力プロダクト	$output$
作業者	$workforce$
予定終了期日	$deadline$
作業量	$workload$
プロダクト P_iの属性	
サイズ	$size$
フォールト数	$fault$
完成度	$c.\ rate$
作業者 M_iの属性	
経験レベル	$level$

表 2: プロジェクトの記述例

A_1	
$type$	FD
$entry\ c.$	(A_1,non-executed)
$exit\ c.$	(A_1,consumed)
$input$	$(P_0, 7.0)$
$output$	$(P_1, 0.3), (P_2, 0.5), (P_3, 0.2)$
$workforce$	$(M_1, 1.0)$
$deadline$	20
A_2	
$type$	PG
$entry\ c.$	(A_1,done)
$exit\ c.$	(A_2,consumed)
$input$	$(P_1, 1.2)$
$output$	$(P_4, 1.0)$
$workforce$	$(M_2, 1.0), (M_3, 1.0)$
$deadline$	35
A_3	
$type$	PG
$entry\ c.$	(A_1,done)
$exit\ c.$	(A_3,consumed)
$input$	$(P_2, 1.1)$
$output$	$(P_5, 1.0)$
$workforce$	$(M_1, 1.0)$
$deadline$	35
A_4	
$type$	FT
$entry\ c.$	(A_2,done), (A_3,done)
$exit\ c.$	(A_4,consumed)
$input$	$(P_3, 2.0), (P_4, 0.2), (P_5, 0.25)$
$output$	$(P_6, 1.0)$
$workforce$	$(M_2, 1.0)$
$deadline$	60
A_5	
$type$	FTD
$entry\ c.$	(A_4,running)
$exit\ c.$	(A_4,done),(A_5,consumed)
$input$	$(P_6, 2.4)$
$output$	$(P_4, 0.45), (P_5, 0.55)$
$workforce$	$(M_1, 1.0), (M_3, 1.0)$
$deadline$	65
P_0	
$size$	8
$fault$	0
$c.\ rate$	1.0
M_1	
$level$	3
M_2	
$level$	2
M_3	
$level$	1

示す．表 2の $A_1, \cdots, A_5$は作業の記述部分である．ここでは作業 A_1の記述を例に説明する．$type$ より A_1は設計 (FD) 作業であることがわかる．$entry\ c.$には作業 A_1が未実行であれば実行を開始することが記されている．一方 $exit\ c.$には A_1はその作業量が全て達成されれば終了することが書かれている．

次に $input$ にはプロダクト P_0が A_1の入力として与えられることと，A_1の作業量が P_0のサイズの 7 倍の値に設定されることが記されている．$output$ には A_1の出力プロダクトが P_1，P_2，P_3 の 3 つであり，作業の進行に伴って定まるプロダクトのサイズやフォールト数の変化量がプロダクト P_1，P_2，P_3に対して 3:5:2 の比率で割り振られることが記されている．$workforce$ は作業 A_1を作業者 M_1が担当することを表している．最後に $deadline$ より A_1の予定終了期日はプロジェクト開始から 20 日目に設定されていることが分かる．

P_0はプロダクトの記述である．プロダクト P_0のサイズが 8，フォールト数が 0，完成度が 100% であることが示されている．更に，作業者 M_1, M_2, M_3の属性より，経験レベルがそれぞれ 3，2，1 であることが分かる．

なお，各作業 $(A_1, \cdots, A_5)$ の作業量，および，初期入力プロダクト (この例では P_0) 以外のプロダクト (P_1,$\cdots$,P_6) の属性値はモデルの実行時に設定される．

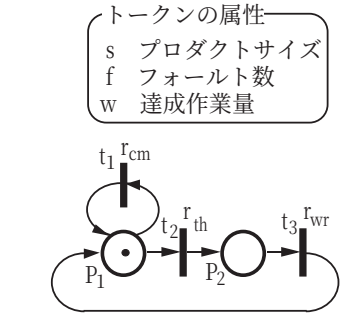
3.4 作業モデル

ここではプロセスモデルの構成要素である作業モデルについて説明する．作業モデルは作業の種類毎に定義し，その記述には拡張 GSPN を用いる．作業の種類として設計，コーディング，レビュー，テスト，デバッグの 5 つを考える．

図 2は拡張 GSPN で記述した設計作業である．この記述ではトークンの属性としてプロダクトサイズ s，フォールト数 f，達成作業量 wの 3 つを持たせている．各属性の意味は次の通りである．(1) プロダクトサイズ s は作業により生成されたプロダクトのサイズを，(2) フォールト数 fは作業中に作り込まれたり，取り除かれたフォールトの個数を，(3) 達成作業量 wはその時点までに達成された作業量を，それぞれ表す．

図 2中のトランジションはすべて発火に時間を要する時間トランジション (図中の太線) である．例えば，トランジション t_1の発火レートは r_{cm}で与えられており，これはトランジション t_1の発火時間が平均 $1/r_{cm}$ の指数分布に従うことを意味する．これらのトランジションには思考，記述，会話といった作業者の振る舞い，あるいは，テスト中での故障の発生といった作業中の事象を対応させる．一方，プレースは振る舞いや事象が起きるまでの待ち状態を表している．

図 2に示した設計作業の場合，トランジション t_1, t_2, t_3はそれぞれ作業者間のコミュニケーション，



トランジション	t_1	t_2	t_3
発火実行関数	-	w=w+1	s=s+1 f=f+1 (p_{in})

図 2: 設計作業

問題の解決に必要な思考，ドキュメントとして記録するための記述に対応しており，発火レートとしてそれぞれ r_{cm}, r_{th}, r_{wr} を与えている．これらの発火レートは一般に次のような関数として表す．

コミュニケーションレート
$r_{cm} = f_{cm}$(作業者数，各作業者の経験レベル，入力プロダクトの完成度)

思考レート
$r_{th} = f_{th}$(作業者数，各作業者の経験レベル)

記述レート
$r_{wr} = f_{wr}$(作業者数，各作業者の経験レベル)

これらの関数の利用により作業者数や経験レベル，あるいは，入力プロダクトの完成度に応じて，コミュニケーションの発生頻度，思考や記述の難易度を動的に設定することができる[1]．

また，思考や記述のトランジションが発火する毎に達成作業量 wやプロダクトサイズ s が増加するといったトークンの属性値の変化は図 2の記述にもある様に，各トランジションの発火実行関数として記述される．なお，設計作業におけるフォールトの混入 (すなわち，フォールト数 fの変化) はフォールト混入率 p_{in}による確率的な事象として取り扱う．このフォールト混入率 p_{in}は一般に次のような関数として表す．

$p_{in} = f_{in}$(作業者数，各作業者の経験レベル，予定終了期日, 入力プロダクトの完成度)

この関数の利用により，デッドラインや作業者の経験がフォールトの混入率に与える影響も考慮することができる[1]．例えば，図 2の設計作業ではトークンが P_1 にあるとき，２つのトランジション t_1(コミュニケーションを表す) と t_2(思考を表す) が発火可能である．t_1の発火によって時間は経過するがプロダクトや作業量の達成には何の影響もなく，トークンは P_1に戻る．次に，t_2が発火すると 発火実行関数によって達成作業量 wが 1 増加するとともにトークンは P_2に移る．トークンが P_2にあるときは t_3だけが発火可能である．この t_3(記述を表す) が発火するとプロダクトサイズ s が増加し，フォールト混入率 p_{in}の値によってフォールト数 fの増加が確率的に起こる．このときトークンは P_1に移る．

[1] $f_{cm}, f_{th}, f_{wr}, f_{in}$の具体的な式は適用するプロジェクトに応じて決めるものとする．

3.5 シミュレ ションシステム

提案したモデルに基づくシミュレーションシステムをすでに開発している [7][10].

シミュレーションの実行中の一例を図 3に示す．同図では画面に各作業に割り当てられた作業量と消費した作業量，各プロダクトのサイズと残存フォールト数が表示されている．これらの情報を見ることで，プロジェクトの進行状況を視覚的にとらえることができる．また，工数と残存フォールト数の移り変わりを示すグラフを表示することも可能である．

4 並行実行への適用

4.1 プロセスの定義

図 4に示す作業の流れで実行される開発プロセスを考える．ここでは次の 2 つの事例についてのシミュレーションを行なう．

- 事例 1: 各作業 $A_i(2 \leq i \leq 34)$ は前の作業が完全に終了してから開始する理想的な場合．
- 事例 2: コーディング作業 $A_{11}, A_{12}, A_{13}, A_{14}$の開始条件を “直前の詳細設計が 70 %終了した時点” とする並列実行を許す場合 (他の作業の開始条件については事例 1 と同じ)．

事例 2 では，設計仕様書が不完全な状態で次のコーディング作業の実行を開始する．つまり，詳細設計の終盤 30%の作業とコーディング作業が並行して行なわれることになる．納期やコストに厳しい制限が課せられる開発現場ではこのような状況が起こる可能性がある．

ここではこの 2 つの事例をそれぞれ提案モデルを用いて記述し，シミュレーションを行ない，開発期間，フォールト数の変化，開発工数を評価する．

4.2 仮定

コミュニケーションレート r_{cm}，思考レート r_{th}，記述レート r_{wr}，フォールト混入率 p_{in}に対して次の (H1)〜(H3) の仮定をおく．以下で，M は作業者数，

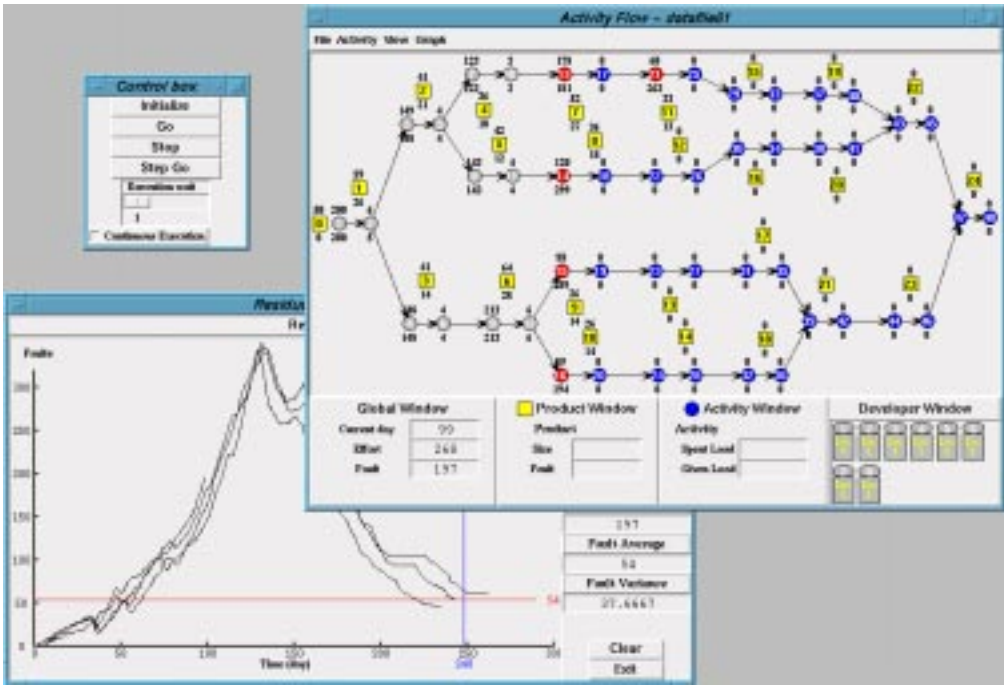


図 3: 実行画面

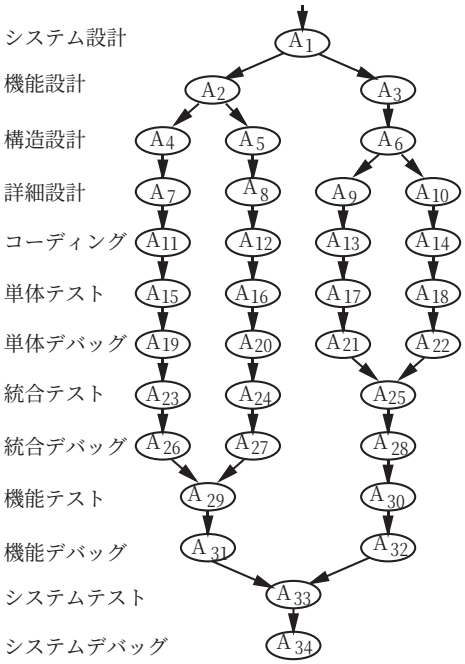


図 4: 開発プロセスの流れ

L は作業者の経験レベルの和，Rは入力プロダクトの完成度，Dは予定終了期日までの日数である．また，$K_{cm}, K_{th}, K_{wr}, K_{in}$は作業毎に与えられるパラメータであり，それぞれコミュニケーション，思考，記述，フォールト混入率に対応する．

(H1) コミュニケーションレートは作業者数の 2 乗に比例し，作業者の経験レベルと入力プロダクトの完成度に反比例する．つまり，

$$r_{cm} = K_{cm} \times \frac{M^2}{LR}$$

とおく．なお，今回の実験では設計，コーディング，テスト，デバグの K_{cm} を全て $K_{cm} = 0.10$ とした．

(H2) 思考レートと記述レートは作業者 1 人当たりの経験レベルと作業者数に比例する．したがって

$$r_{th} = K_{th} \times \frac{L}{M} \times M = K_{th} L$$

$$r_{wr} = K_{wr} \times \frac{L}{M} \times M = K_{wr} L$$

とする．さらに，設計，コーディング，テストの K_{wr} をすべて $K_{wr} = 0.20$ とおく．

(H3) フォールト混入率は作業者数に比例し，作業者 1 人当たりの経験レベルと入力プロダクトの完成度，予定終了期日までの日数に反比例する．したがって

$$p_{in} = K_{in} \times \frac{M}{LRD} \times M$$

とする．さらに，設計では $K_{in} = 15.0$，コーディングでは $K_{in} = 17.0$ とおく．

ただし，現時点ではモデルの記述に必要な開発データが完全には得られていないため，パラメータの一部に架空の値を設定してシミュレーションを行なっている (紙面の都合上，プロジェクトモデルの記述や設計作業以外で用いられるレートを表す関数の詳細は省略する).

4.3 実験結果と分析

モデルを用いてシミュレーションを行なった結果を表 3と表 4に示す．まず，表 3は設計作業とコーディングの並行実行を開始する直前の時点でのシミュレーション結果である．この時点では，事例 1 と事例 2 の間に差異は見られない．次に，表 4は開発プロセス終了時点でのシミュレーション結果である．事例 2 においては，平均開発期間が短かったものの平均開発工数は事例 1 に比べて 20 人日増加していることが分かる．事例 1 と事例 2 の開発プロセスにおける相違点は設計作業からコーディングに移る時に並行実行をしたか否かだけであるので，事例 1 と事例 2 の開発工数の差がこの並行実行にあると推測される．

表 4における事例 1 と事例 2 の差 (事例 2 の方が事例 1 に比べて開発期間は短くなっているが，工数は多くかかっていること) の原因について考える．事例 2 では詳細設計と並行してコーディングが行われていた期間中，そのコーディング作業の入力プロダクトは不完全なままであった．従って，(1) 混入されたフォールト数が増え，その後のテストとデバッグの作業量が増加した，(2) コミュニケーションのオーバーヘッドが増加した，という原因が考えられる．

実際の開発データを全てのパラメータに対して用いた適用例ではないため必ずしも正確な評価ではないが，提案モデルを用いれば開発現場の実状をある程度反映したシミュレーションが可能になることを示せたものと考えている．

表 3: 並行実行開始直前の結果 (100 回平均)

並行実行開始直前時点	事例 1	事例 2
開発期間の平均	84	84
開発期間の標準偏差	3.41	2.85
開発工数の平均	202	201
開発工数の標準偏差	7.69	7.53
残存フォールト数の平均	89	91
残存フォールト数の標準偏差	8.65	10.01

5 まとめ

本稿では，ソフトウェア開発プロセスの並行実行を拡張一般化ペトリネットモデルでモデル化し，シミュレーションを行ない品質，コスト，開発期間を定量的に評価した．なお，モデルの適用に当たっては開発プロジェクトの実測データを用いて開発現場の実状を反映させることを試みるとともに，そのシミュレーション結果に基づいて事例の分析を行なった．

表 4: 開発プロセス終了時での結果 (100 回平均)

開発プロセス終了時点	事例 1	事例 2
開発期間の平均	212	201
開発期間の標準偏差	9.4	6.5
開発工数の平均	751	771
開発工数の標準偏差	22.00	24.75
残存フォールト数の平均	49	48
残存フォールト数の標準偏差	7.38	5.67

今回の適用実験ではモデルのパラメータの設定に当たって一部に架空のデータを用いたため，十分に正当な評価は行なえなかった．したがって，実測データに基づいたシミュレーションを行ない，提案システムの有効性を検証することが今後の課題である．

謝辞

本研究に進めるに当たって数多くの貴重なデータを提供して頂き，多大なご協力を賜った オムロン (株) の坂本啓司氏，高木徳生氏に感謝致します．

参考文献

[1] Basili V. R. and Rombach H. D.: “The TAME project: Towards improvement-oriented software environment”, IEEE Trans. Software Engineering, Vol.14, No.6, pp.758-773 (1988).

[2] Curtis B., Krasner H. and Iscoe N.: “A field study of the software design process for large systems ”, Communications of the ACM, Vol.31, No.11, pp.1268-1287 (1988).

[3] 古澤, 平山, 楠本, 菊野, 田中: “一般化確率ペトリネットに基づくソフトウェア開発プロセスのモデル化と定量的評価”, 第 15 回ソフトウェア信頼性シンポジウム論文集，pp.99-104 (1994).

[4] Furuyama T., Arai Y. and Iio K.: “Fault generataion model and mental stress effect analysis”, The Journal of Systems and Software, Vol.26, pp.31-42 (1994).

[5] Humphrey W. S.: “Managing the Software Process”, Software Engineering Institute, Addison-Wesley (1989). “ソフトウェアプロセス成熟度の改善”，藤野喜一監訳，日科技連 (1991).

[6] Kellner M. I.: "Software process modeling support for management planning and control", Proc. of the 1st International Conference on Software Process , pp.8-28 (1993).

[7] Kusumoto S., Mizuno O., Kikuno T., Takagi Y. and Sakamoto K.: "A New Software Project Simulator Based on Generalized Stochastic Petri-net", Proc. of the ICSE97,(to appear)

[8] Lee G. and Murata T.: "A β-distributed stochastic Petri net model for software project time/cost management", The Journal of Systems and Software, Vol.26, No.2, pp.149-165 (1994).

[9] Marsan A., Conte G. and Balbo G.: "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", ACM Trans. on Computer System Vol.2, pp.93-122 (1984).

[10] 水野, 平山, 楠本, 菊野 : ソフトウェア開発プロセスの進捗予測システムの開発, ソフトウェアシンポジウム'96 予稿集, (1996)

[11] Raffo D. M.: "Evaluating the impact of process improvements quantitatively using process modeling", Proc. of CASCON93, Vol.1, pp.290-313 (1993).

[12] Tvedt J. D. and Collofello J. S.: "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling", Proc. of COMPSAC95, pp.318-325 (1995).