ソフトウェア開発プロセスの進捗予測システムの開発

水野修，平山裕司[0]，楠本真二，菊野 亨
大阪大学 基礎工学部 情報工学科
〒560 大阪府豊中市待兼山町 1-3
Tel: 06-850-6568 Fax: 06-850-6569
E-mail: {o-mizuno,hirayama,kusumoto,kikuno}@ics.es.osaka-u.ac.jp

あらまし

大規模化，複雑化が進むソフトウェアの開発においてプロジェクト管理の重要性が高まっている．効果的なプロジェクト管理を行うためには，開発プロセスの定量的かつ客観的な評価結果に基づいた管理手法を確立する必要がある．本稿では，まず，客観的かつ定量的なプロジェクト管理手法を確立するために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，評価する手法を提案する．次に，提案した一般化確率ペトリネットの拡張モデルに基づき，試作したソフトウェア開発プロセスの進捗予測システムの概要と基本機能，評価例について述べる．

1 まえがき

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 [13][17][18]．しかし,個々のプロダクトあるいは開発プロセスだけを独立に分析するだけでは,様々な要因の相互作用の結果として生じるこれらの問題を十分に解決することは困難である [1]．

こうした背景の下で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 [8][15]． これまでにもソフトウェア開発プロセスのモデル化とその評価を行う手法が数多く提案されてきている [9][10][13][18][19]．PERT/CPM も代表的なものの一つである．しかし，品質，コスト，開発期間の 3 つ観点からプロセスを定量的に評価するようなもの，実際の開発プロセスに適用して有効性の評価を行ったものはほとんどない．

本稿では，先ず，客観的かつ定量的なプロジェクト管理手法を確立するために，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する．提案する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なっている．

次に，提案した一般化確率ペトリネットの拡張モデルに基づくソフトウェア開発プロセスの進捗予測システムを設計，試作した．その概要と基本機能，さらに，提案システムの評価実験として，ある企業における開発実績データに基づいた例題プロセスの定量的評価を行なった結果についても報告する．

2 ペトリネットモデル

2.1 一般化確率ペトリネット (GSPN)

ペトリネットはソフトウェア開発プロセスのような並行システムの解析やシミュレーションでのモデル化に適した記述能力の高い数学的モデルである．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 [11]．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

[0]現在オムロン株式会社

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 [5][7]

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

提案する拡張GSPNに基づく開発プロセス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り，開発期間とプロダクトサイズの変動を考慮したソフトウェア開発の進捗予測を実現する．

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．例えば，工数で表した場合に1人の作業者が10日間で終えた作業も，10人の作業者が1日で終えた作業も同じ10人日となるが，一般に人数が多くなれば作業者間のコミュニケーションに要する時間が増えるため，それぞれの作業の大きさは必ずしも等しくない．工数には本来，作業の遂行に必要な作業の大きさだけでなく，コミュニケーションに必要な手間の大きさも含めている．従って，10人の作業者が1日で終えた作業(10人日の工数の作業)を1人の作業者が行なった場合，10日より少ない日数で終える可能性がある．

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．

例えば，作業量20時間の作業の実行について次の2つの場合を考える．

場合1：2人の標準的な能力の作業者が、次の作業条件でこの作業を行なうとする．2人の間のコミュニケーション・オーバーヘッドとして作業に携わる時間の10%は作業の進行に寄与しない．このとき，2人が10時間作業に携わったとしても，達成された作業量は18(= 10時間 × 2 × 0.9)時間となる．

場合2：標準的能力の作業者4人が次の条件のもとで5時間作業に従事するとする．4人の間のコミュニケーションにより作業に携わる時間の20%がオーバーヘッドとしてかかる．このときに達成される作業量は16(= 5 × 4 × 0.8)時間となる．

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．

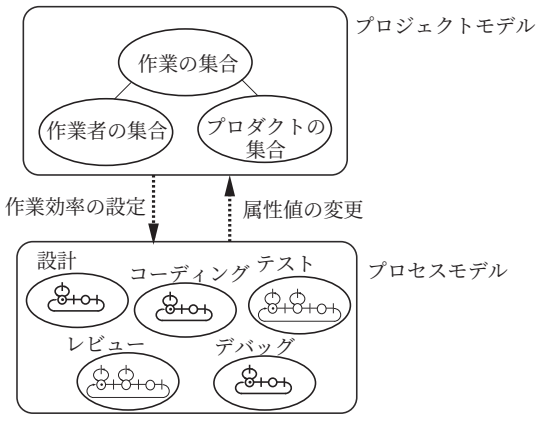


図 1: モデルの概略

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．

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．
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．
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

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

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．

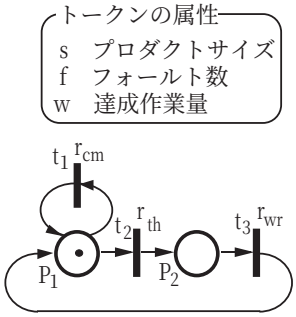
なお，各作業 ($A_1, \cdots, A_5$) の作業量，および，初期入力プロダクト (この例では P_0) 以外のプロダクト (P_1,$\cdots$,P_6) の属性値はモデルの実行時に設定される．

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．



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

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，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的には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} は一般的には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)

[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．

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．

4 進捗予測システム

本章では，提案したモデルに基づく進捗予測システムとそれを用いたシミュレーションについて述べる．

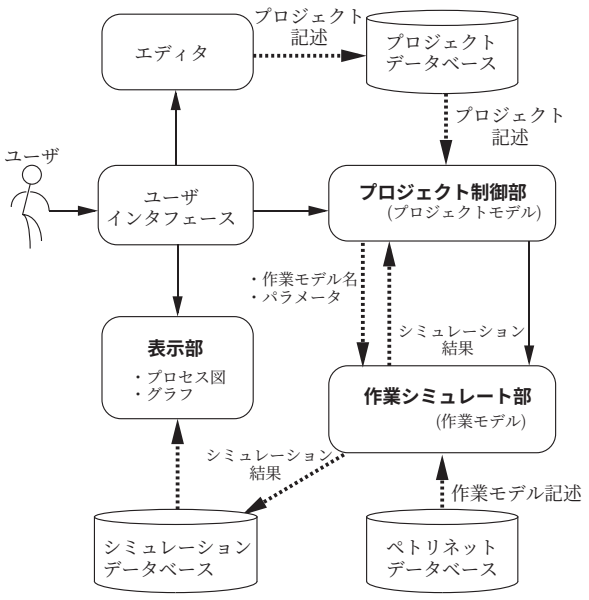


図 3: システム構成

4.1 システム構成

システムの全体構成を図3に示す．システムはプロジェクト制御部，作業シミュレート部，ユーザインタフェース部，表示部，エディタから構成される．実線が制御の流れを，点線がデータの流れを表す．次に各部について説明する．

(1) プロジェクト制御部: プロジェクト制御部は，プロジェクトデータベースに保存されているプロジェクト記述に基づいて，作業シミュレート部でシミュレートすべき作業モデルとそのパラメータを設定する．具体的には，作業，プロダクト，作業者の関係の定義や各作業，プロダクト，作業者についての属性値を設定する．最後に，設定した作業モデル，パラメータを作業シミュレート部へ受け渡す．

(2) 作業シミュレート部: 作業シミュレート部では，プロジェクト制御部から与えられた情報(作業モデル名とパラメータ)に基づいて，シミュレーションを行う．具体的には，先ず，与えられた作業モデル名に対応する拡張GSPN作業モデル記述をペトリネットデータベースより読み出す．次に，与えられたパラメータを用いてモデルを動作する．シミュレーション結果は一定時間間隔でプロジェクト制御部へ渡され，同時に，シミュレーションデータベースにも保存される．

(3) ユーザインタフェース部: ユーザインタフェース部はユーザとシステム(エディタ，プロジェクト制御部，表示部)との間のデータとコマンドのやりとりを管理する．

(4) 表示部: シミュレート部において得られたデータを画面に表示する．また，シミュレーションデータベースに保存されている過去のプロジェクトのシミュレーション結果を表示することも可能である．データの評価を支援するためのグラフ表示機能や統計処理機能を持つ．

(5) エディタ: エディタはプロジェクト制御部の入力であるプロジェクト記述の作成を支援する．GUIによるプロジェクトの記述が可能である．提案モデルにおける全てのパラメータの設定が行なえる．エディタの出力はプロジェクト記述ファイルとなり，プロジェクトデータベースに保存される．

システムの実現に際して，高速な計算を求められる作業シミュレート部とプロジェクト進行制御部はC言語を用いて記述した．一方，高速性よりも操作性が重要である表示部とエディタはTcl/Tkで記述した．プログラムのサイズは，C 言語で記述した部分が約1000行，Tcl/Tkで記述した部分が約2500行 であった．

4.2 システムの動作

シミュレーションの実行は1日を単位として次のように進められる．作業の進行を1日進める時には，先ずプロジェクト進行制御部が前日までの進行状況，各作業の開始条件，終了条件に基づいて，その日に実行すべき作業を決定する．次に，作業シミュレート部に1日分の作業を行うように指示を出す．作業シミュレート部では，指示された拡張GSPNに従って作業を実行する．各作業では，その実行開始時に割り当てられた作業量を消費していき，これが0になった時に実行終了とみなす．

シミュレーションは任意に停止したり，再生することができる．また任意の時点の状態を保存可能であり，保

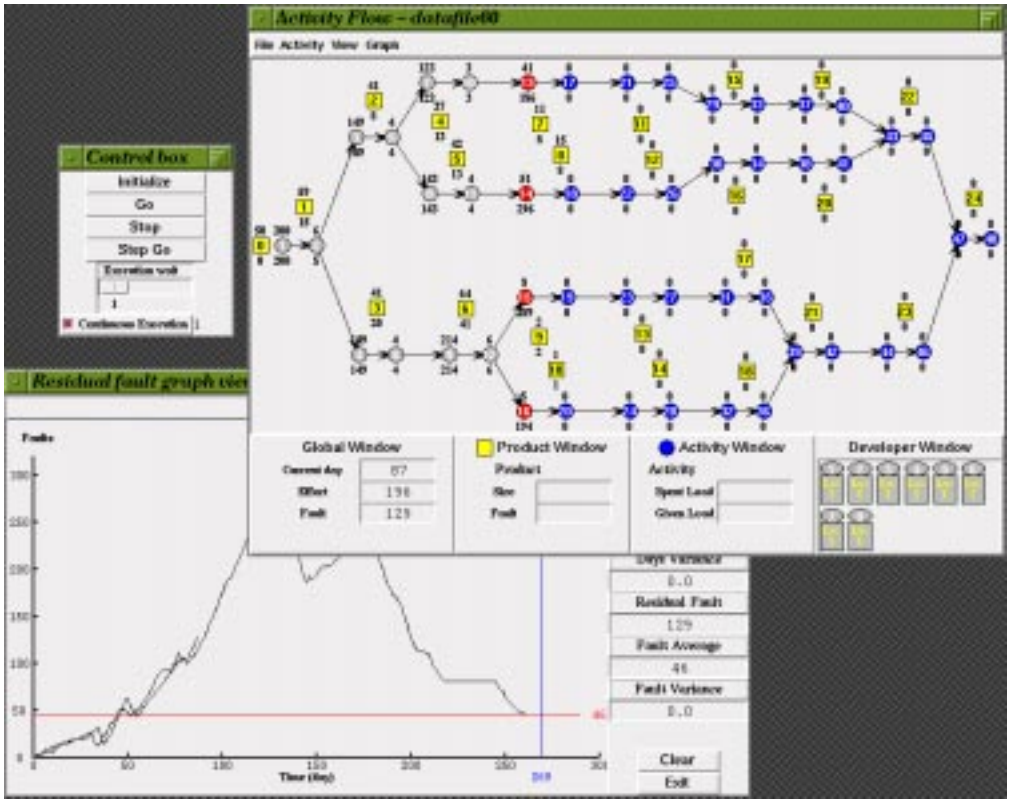


図 4: 実行画面

存した状態は前述のプロジェクト記述の形式をとっているので，これを用いてプロジェクトの状態を変更する (例えば，ある時点から作業者の数を増やす) ことも可能である．また，シミュレーション途中の各種パラメータの変更も行える．

シミュレーションの実行中の一例を，図4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．

4.3 ケーススタディ

図5に示す作業の流れで実行される開発プロセス [16] 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” とする例外的な場合 (他の作業の開始条件については事例 1 と同じ)．

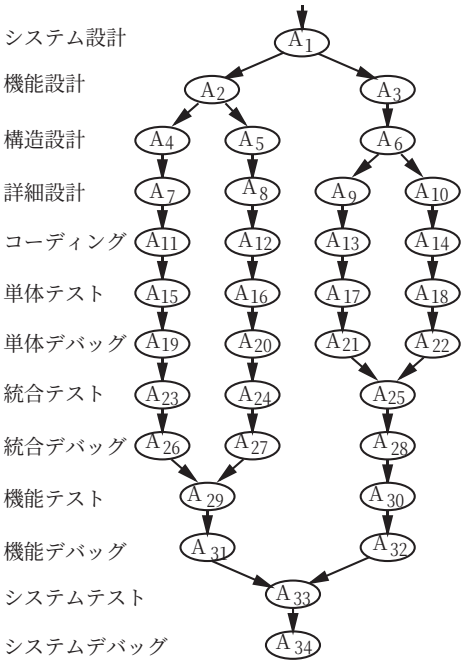


図 5: 開発プロセスの流れ

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，開発工数を評価してみる．なお，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，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群であり，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．
(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．
(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．

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)．

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，平均開発期間は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わかる．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，この開発工数の差が並行実行にあることがわかる．

表4における事例1と事例2の差(事例2の方が事例1に比べて開発期間が短くなっているが，工数が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，ということが事例1と事例2の差の原因と考えられる．

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反映したシミュレーションが可能であることが示せたものと考えている．

表 3: 並行実行開始前までのシミュレーション結果(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

表 4: 開発プロセス終了時でのシミュレーション結果(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

5 まとめ

本稿では，ソフトウェア開発プロセスのペトリネットモデルに基づく進捗予測システムを提案した．また，提案したシステムを用いてソフトウェア開発のシミュレーションを行ない，その結果に基づいて品質，コスト，開発期間を定量的に評価した．なお，システム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した事例のシミュレーションを行なうとともに，その結果に基づいて事例の分析を行なった．

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うことはできなかった．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] Abdel-Hamid T. K.: “The dynamics of software project staffing: A system dynamics based simulation approach”, IEEE Trans. of Software Eng., Vol.15, No.2, pp.109-119 (1989).

[2] 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).

[3] Brooks F. P., Jr.: “The Mythical Man-Month”, Addison-Wesley (1975).

[4] 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).

[5] 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).

[6] 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).

[7] 平山, 水野, 楠本, 菊野: “ソフトウェアプロジェクトの定量的管理のための階層的モデルの提案”, 電子情報通信学会技術研究報告 FTS95-74，pp.89-96 (1995).

[8] 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).

[9] 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).

[10] 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).

[11] Marsan A. et al.:“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).

[12] 大場 充: “ソフトウェア・プロジェクトの実績データ収集・分析技法”, ソフト・リサーチ・センター (1993).

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

[14] Sackman H., Erickson W. J. and Grant E. E.: “Exploratory experimental studies comparing online and offline programming performance”, Communications of the ACM, Vol.11, No.1, pp.3-11 (1968).

[15] 菅野文友, 吉澤 正 監修:“21 世紀へのソフトウェア品質保証技術”, 日科技連出版社 (1994).

[16] Takagi Y., Tanaka T., Niihara N., Sakamoto K., Kusumoto S. and Kikuno T.: “Analysis of review’s effectiveness based on software metrics”, Proc. of the 6th International Symposium on Software Reliability Engineering , pp.34-39 (1995).

[17] Tanaka T., Sakamoto K., Kusumoto S., Matsumoto K. and Kikuno T.: “Improvement of software process by process description and benefit estimation”, Proc. of the 17th International Conference on Software Engineering, pp.123-132 (1995).

[18] 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).

[19] 山田 茂, 高橋宗雄:“ソフトウェアマネジメントモデル入門”, 共立出版社 (1993).