社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ベイジアンネットに基づく ソフトウェア開発工程の最終品質予測モデルの提案

天嵜　聡介† 　水野　修†† 　菊野　亨††

† 大阪大学 大学院基礎工学研究科 情報数理系専攻
†† 大阪大学 大学院情報科学研究科 情報システム工学専攻
〒 560–8531 大阪府豊中市待兼山町 1–3
E-mail: †amasaki@ics.es.osaka-u.ac.jp, ††{o-mizuno,kikuno}@ist.osaka-u.ac.jp

あらまし　本研究では，ソフトウェア開発プロジェクトの各工程における作業とソフトウェアプロダクトの最終的な品質の関係をモデル化することによる最終品質予測モデルの構築を試みる．モデル化の手法としては，予測を行う時点において値が不確実なメトリクスをモデルに含むことが可能なベイジアンネットを採用する．モデル化の対象はソフトウェア開発工程における残存不具合数の推移である．具体的には，残存不具合数の推移を軸として，レビューやテスト作業によって発見された不具合数，それらの作業に要した工数の依存関係を利用してモデル化を行った．そしてモデルの性能評価を行い，最終品質予測モデルとして有用であることを示した．
キーワード　ベイジアンネット，ソフトウェア品質モデル，品質予測

Empirical Software Quality Prediction Model based on Bayesian Network

Sousuke AMASAKI†, Osamu MIZUNO††, and Tohru KIKUNO††

† Graduate School of Engineering Science, Osaka University
†† Graduate School of Information Science and Technology, Osaka University
1–3 Machikaneyama, Toyonaka-shi, Osaka 560–8531, Japan
E-mail: †amasaki@ics.es.osaka-u.ac.jp, ††{o-mizuno,kikuno}@ist.osaka-u.ac.jp

Abstract　In this paper, we propose a model to predict the final quality of software products by using the Bayesian network model. Generally speaking, software development includes various uncertain factors which cannot be determined a priori. We thus use Bayesian network model, which can handle uncertainties as probabilistic events, to represent causal relationships between metrics related to the software quality. We then constructed a quality model that focuses on the structure of software process explicitly. Concretely, the effect of review activity is considered. Based on the metrics data collected from the development projects in a certain company, we evaluated the constructed model. As the result of empirical evaluation, we confirmed that the constructed model can be applicable to the prediction of final software quality.
Key words　Bayesian network, Software quality model, Quality prediction

1. ま え が き

近年，ソフトウェアの社会的な位置づけがますます重要性を帯びてくると共に，その品質を保証することが重要な目標となっている [1]．ソフトウェアの品質は，通常そこに含まれる不具合の数によって評価される．つまり，ソフトウェア開発においてソフトウェアプロダクト中に作り込まれた不具合の数と，検出され修正された不具合の数によって決定される．従って，設計・コーディング工程において作り込まれる不具合を少なくし，テスト・デバッグ工程において検出・修正される不具合を多くすることが高品質なソフトウェアを作成することにつながる．

特に，最終的な品質を決定する最後の作業がテスト・デバッグ工程であることから，そこで発見された不具合の数が十分であることを保証するため (つまり，十分なテストが行われたことを保証するため) に，残存する不具合の数を予測する手法が必要となる．

以上のような背景から残存不具合数の予測手法は多く提案されている [11]．例えば，ソフトウェア信頼度成長モデル (SRGM) はテスト工程中に発見された不具合の数の推移から残存する不具合の数を予測する．ただし，この手法では単一のテスト工程のみが対象であり，設計工程やプロダクトの品質は考慮されていない．

一方，プロダクトから得られるメトリクスを用いた手法としては統計的分析を基にしたものが多く知られている．特に，回

帰分析を用いた手法は多く試みられており，[3] や [9] などが知られている．しかし，こうした回帰分析をベースとした手法には問題があることが指摘されている [5]．その最も特徴的な問題は，従属変数の独立性を保証しなければならないという点である．実際のソフトウェアメトリクスでは相関性の高いメトリクスが多く，そのため同時に使用できるメトリクスを用いた回帰モデルを構築することが困難であると述べられている．

Fenton らは，[5] において回帰分析を用いた手法で発生する問題点を克服する為の手法としてベイジアンネットを用いたモデル化を提案している．ベイジアンネットは複雑な依存関係にある対象をモデル化する手法の一つであり，その特徴の一つにモデルに含まれる要素の値が不確実であっても予測が可能である点が挙げられる．この点は，実用的な予測モデルを構築するためには大きな利点であり，また回帰分析によって得られるモデルにはない特徴である．

本研究では，実際のプロジェクトで記録されたメトリクスのみを用いて，最終品質を予測するモデルの構築を試みる．具体的には，開発プロセスにおける残存不具合数の推移に着目したモデル化を行う．また，最終工程完了時の残存不具合数をプロダクトの品質とみなす．さらに，[5] ではモデルの評価についての手法について触れていないので，本研究では定量的にモデルの評価を行うことを試みる．

以降の本報告の構成は次の通りである．まず，2 節では本研究の目的，対象プロジェクト，使用するメトリクスについて述べる．3 節では，モデル化に使用するベイジアンネットの構築手順について説明する．そして，4 節ではその手法を用いて，ソフトウェア開発プロセスのモデル化を行う．続く 5 節では，4 節で構築されたモデルの評価を行う．最後に本報告のまとめを 6 節で述べる．

2. 目的と準備

2.1 研究の目的

本研究の目的は次の 2 点である．

- 開発プロセスの設計終了時の段階においてソフトウェアプロダクトの最終品質を予測するモデルの構築
- 構築された最終品質予測モデルの性能評価

最終品質予測モデルの構築にはベイジアンネットを用いる．また，評価に当たっては，テスト完了時にどれぐらい残存不具合数が存在するかという確率の変動に着目する．

2.2 対象プロジェクト

本研究で対象とするプロジェクトは，ある企業で実施されたプロジェクトのうち，既に終了した 225 個のプロジェクトである．対象プロジェクトは次の基準で選ばれている．

（1） 1995 年から 1998 年の間に実施されている

（2） テスト・デバッグ工程全体で検出された不具合数がある一定値以上となっている

対象プロジェクトの開発プロセスは，図 1 に示すような標準的なウォーターフォールモデルとなっている．設計・コーディング工程は，構想設計，機能設計，構造設計，モジュール設計，

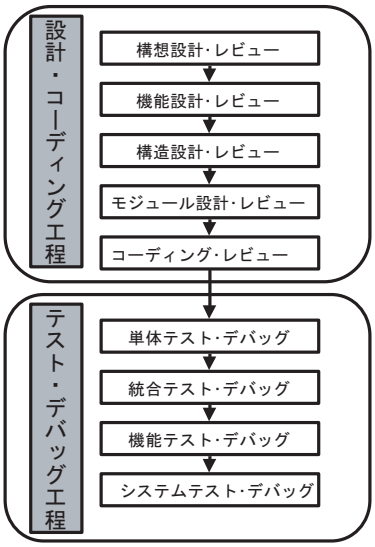

図 1　開発プロセス

コーディングの 5 つの作業から構成されており，各作業の終了時にはレビュー作業が行なわれる．レビュー作業とは，設計ドキュメントやソースコードなどのプロダクトを構成した直後に，そのプロダクトに作り込まれた不具合を静的解析により発見し，修正する作業である．レビュー作業を行なうことで，テスト・デバッグ工程におけるデバッグ作業を削減することができ，開発全体に費やす工数を削減できることが知られている．なお，コーディングでのレビュー作業のことを特にコードレビューと呼ぶ．

本来は設計作業とレビュー作業が交互に行われるが，本研究では設計作業とレビュー作業は共に単一の作業と考えてモデルを構築している．

テスト・デバッグ工程はモジュールテスト，統合テスト，機能テスト，システムテストの 4 つの作業からなり，各作業ごとにテスト作業とデバッグ作業が行なわれている．テスト作業によって検出され，デバッグ作業によって除去された不具合についてはその作業内容の詳細を必ず記録するように指示されている．

本研究ではこれらテスト作業を単一の作業と考えてモデルを構築している．

2.3 メトリクス

本研究では使用するメトリクスは開発プロジェクトの進行中に観測が可能なものと不可能なものとに分類できる．それぞれのメトリクスは次の通りである．

- 観測可能
  - 設計工数
    構想設計からコーディングに要した工数 (人日)
  - レビュー工数
    構想設計からコーディングの直後に行われるレビュー作業に要した工数 (人日)
  - テスト工数
    モジュールテスト～システムテストに要した工数 (人日)
  - 発見不具合数 (レビュー完了時，テスト完了時)
    各工程が完了した時点で発見された不具合の数

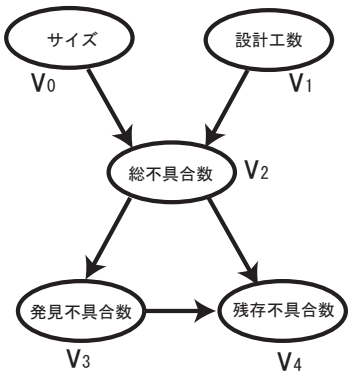

図 2　ベイジアンネットの例

  - 設計完了時開発対象容量
    設計完了時のプロダクトのサイズ (Kstep)

- 観測不可能
  - 残存不具合数 (レビュー完了時，テスト完了時)
    各工程が完了した時点でプロダクトに残存する不具合の数．本研究では前の工程の残存不具合数又は総不具合数と発見不具合数の差を残存不具合数と見なす．
  - 総不具合数
    プロダクトに混入された不具合の総数．本研究では各工程で発見された不具合数とテスト完了時の残存不具合数の和を総不具合数と見なす．

3. ベイジアンネット

3.1 ベイジアンネットの概要

ベイジアンネットとは確率をノードで表し，因果関係や依存関係といった依存する関係を持つ変数の間にリンクを張った非循環有向グラフ構造で表現される確率モデルの一種である [4]．近年，高速な計算法が現れたことで，広い分野で用いられるようになりつつある．その応用分野は医療におけるエキスパートシステムや音声認識，データマイニングなど多岐にわたる [2, 7, 8, 10].

その特徴としては以下の点が挙げられる．

- 予測結果が確率分布によって返される．
- モデル中の全てのノードの観測値が得られらない場合でも予測が可能となる．

ベイジアンネットでは，確率変数間の定性的な依存関係をグラフ構造によって表し，変数間の定量的な依存関係はその変数の間に定義される条件付き確率によって表すことで問題領域をモデル化する．

ここでは予測診断の視点からベイジアンネットの利用例を図 2 のモデルを用いて具体的に説明する．このモデルは今回作成した最終品質の予測モデルの一部分を抜き出したものである．簡単に図を説明すると，楕円が確率変数を表し，矢印が変数間の依存関係を表している．例えば，総不具合数は設計工数とサイズに依存，つまりこれら 2 つのメトリクスから決定されると考える．

ここでは例として，設計工程が完了した時点でプロダクトに残存する不具合数がどのぐらいになるのかを予測する．設計工程が完了しているので，設計工数とサイズは既知の情報である．そこで，これらの値を図 2 の対応するノードに入力として与える．すると，その情報をもとにして，図 2 の残存不具合数を表すノードに残存不具合数がどのくらいになるかという出力が確率分布の形で示される．

ベイジアンネットではこのように，総不具合数やその後の発見不具合数といったプロジェクトの実行中には分からない変数をモデルに入力として与えなくても残存不具合数を予測することが可能である．

また，ノードに観測値を与えて各ノードの確率分布の計算を行うと依存関係の逆方向にも影響を及ぼすので，サイズと発見不具合数と残存不具合数を入力として与えることで，設計工数がどのぐらいであったかということも推測可能である．

3.2 ベイジアンネットの構築手順

ベイジアンネットの構築手順は次の 3 段階で行われる．

（1） 変数の選択

（2） グラフ構造の設計

（3） 確率分布の割り当て

ここでは例として，前節で示した図 2 を用いて具体的なモデルの構築手順を述べる．

（1） 変数の選択

まず，モデル化したい確率変数を選択する．ここでは 5 つの確率変数 v_0, v_1, v_2, v_3, v_4 を用いる．これらはいずれも *High*,*Low* の二値を取る離散変数であると考える．

（2） グラフ構造の設計

次に，確率変数間の定性的な依存関係から非循環有向グラフ $G(V, E)$ を作成する．ここでは各確率変数の間に図 2 のような関係があると考えるので，グラフ $G(V, E)$ は次のようになる．

$$V = \{v_0, v_1, v_2, v_3, v_4\}$$

$$E = \{(v_0, v_2), (v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$$

（3） 確率分布の割り当て

最後に全ての確率変数に確率分布を割り当てる．まず，親ノードを持たない確率変数 v_0, v_1 には事前確率分布 $P(v_0)$, $P(v_1)$ を割り当てる．確率変数が離散値を取る場合は，表を用いて確率分布を表す．例えば，確率変数 v_0 の事前確率分布 $P(v_0)$ が $p(v_0 = High) = 0.6$, $p(v_0 = Low) = 0.4$ ならば，事前確率分布の表は表 1 のようになる．一方，親ノードを持つ確率変数 v_2, v_3 には条件付き確率分布 $P(v_2|v_0, v_1)$, $P(v_3|v_2)$, $p(v_4|v_2, v_3)$ を割り当てる．ここでも，確率変数が離散値を取る場合には，表を用いて確率分布を表す．例えば，v_2 の条件付き確率 $P(v_2|v_0, v_1)$ は，表 2 のような条件付き確率表 (CPT) を割り当てる．

以上の手順でベイジアンネットは構築される．

この例のように，開発プロセスをモデル化する際には確率変数とメトリクスを対応させてモデル化を行う．確率分布を実際にはどのように与えるかは 4 節で説明する．

表 1　事前確率分布 $P(v_0)$

v_0	
High	Low
0.6	0.4

表 2　条件付き確率分布 $P(v_2|v_0, v_1)$

v_0	v_1	v_2	
		High	Low
High	High	0.9	0.1
	Low	0.2	0.8
Low	High	0.3	0.7
	Low	0.6	0.4

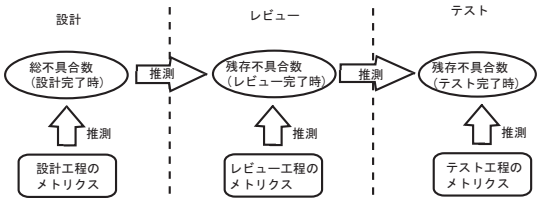

図 3　モデルの基本構造

4. 開発プロセスのモデル化

4.1 モデル化の指針

本研究におけるモデル化の指針は次の 2 点である．

（1） 実際の開発プロジェクトで記録されているメトリクスのみを使用

（2） 開発プロセスの各工程が完了した時点での残存不具合数に着目

(1) については，に実際に記録されている範囲でモデルを構築することで，過去に記録されたデータを活かしつつ，また実際の開発現場においての利便性も確保することが可能となる．

次に，(2) については，品質を実際に記録されているメトリクスから判定するとすれば，各工程における総不具合数や残存不具合数が妥当であると考えたからである．そこで，図 3 に示すように，モデル化にあたっては，図 1 の開発プロセスにおける各工程が完了した時点での総不具合数と残存不具合数をそれぞれの工程における出力とする．また，レビュー工程とテスト工程の残存不具合数はその工程で行われる作業と前の工程の総不具合数や残存不具合数から決定されると考える．例えば，図 3 では，設計工程完了時の総不具合数は，設計工程の作業の結果作り込まれた不具合の数と，設計工程で行われた作業の結果得られる工数などのメトリクスから決定されることを示している．

4.2 メトリクスの選択

このモデル化で用いる，実際のプロジェクトで記述されているメトリクスは次の通りである．

- 設計作業
  設計完了時開発対象容量，設計工数
- レビュー作業
  レビュー工数，発見不具合数
- テスト作業
  テスト工数，発見不具合数，フィールド不具合数

また，モデル化には以上のメトリクスの他に以下のメトリクスが必要となる．

- 総不具合数 (設計完了時)
- 残存不具合数 (レビュー完了時)
- 残存不具合数 (テスト完了時)

これらの真値は実際には不明であるが，テスト完了時の残存不具合数をプロダクトの出荷後 6ヶ月間に発見された不具合の数とし，レビュー完了時の残存不具合数はその数とテスト工程における発見不具合数の和として考えている．また，総不具合数はレビュー完了時の残存不具合数とレビュー工程での発見不具合数の和として考えている．以上のメトリクスの定義については，2.3 節で説明している．

ここで示したメトリクスは連続値を取るが，連続値を扱う場合，母集団の分布に対して仮定を置き，パラメトリックな分布で処理を行わなければならない．しかし，ソフトウェア開発で記録されるメトリクスでは母集団の分布が不明であることが多い．そこで，以下の手順でこれらのメトリクスを幾つかのクラスからなる離散変数に変換する．

- 残存不具合数 (テスト完了時)
  値に応じて $C_0, \ldots, C_4$ の 5 つのクラスに分類
- それ以外
  - メトリクスのデータを降順でソートし，上位約 10%の数のプロジェクトを 1 つのクラスと定める．
  - 残りのプロジェクトは 0 と最大値の間の範囲を三等分して 3 つのクラスを定める．

このように離散値を取る変数に変換されたメトリクスをノードとして利用し，以降では最終品質予測モデルの構築を行う．

4.3 グラフ構造の設計

グラフ構造は以下に示すような手順で設計した．基本的には，まず各工程の総不具合数や残存不具合数に直接影響を及ぼすメトリクスをモデルに追加し，その後に利用者がコントロールできるメトリクスを追加するという手順でモデル化している．

4.3.1 設 計 工 程

本研究では，設計工程は不具合が混入される作業と見なしている．設計によってプロダクトに作り込まれる不具合数は人員の質や問題の困難さなどにも左右されるが，これらのメトリクスは記録することが難しく現時点では収集されていない．

実際に記録されているメトリクスとしてはプロダクトのサイズ (設計完了時対象容量) と設計工数があり，これらのメトリクスはいずれも総不具合数に影響を及ぼしていると考えられる．

そこで図 4(a) に示すような形でこれらのメトリクスをモデルに追加した．

4.3.2 レビュー工程

レビュー工程は不具合を除去する作業と見なすことができる．そして，除去された不具合の数はの発見不具合数として記録されている．ここでレビュー完了時の残存不具合数と発見不具合

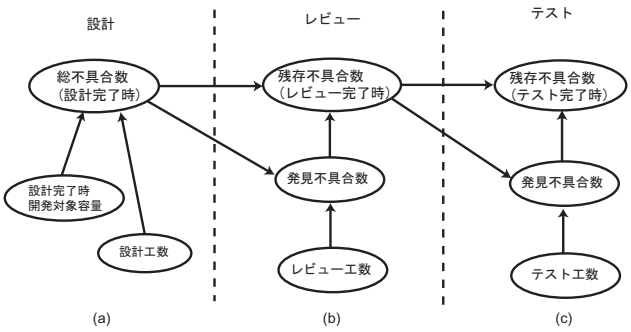

図 4　最終品質の予測モデル

数には以下のような関係がある．

(残存不具合数) = (総不具合数) – (発見不具合数)

そこで，図 4(b) ではレビュー完了時の残存不具合数は総不具合数と発見不具合数に依存するという形でモデル化した．レビュー作業完了時には発見不具合数は既知となるので，この時点でレビュー完了時の残存不具合数の予測は可能となる．

ここで更に実用性を考慮すると，設計工程完了時にレビュー完了時の残存不具合数を予測したい場合，利用者が着目するのはレビュー工程でどれぐらい不具合が取れたら良いかではなく，どのぐらいの工数をレビュー作業に費やせば良いのかという点である．発見不具合数はレビュー工数に依存していることは明らかであるので，ここでは図 4 のような形でレビュー工数をモデルに追加した．

4.3.3　テスト工程

テスト工程はレビュー工程と同様に不具合を除去する作業と見なすことができる．そこで，図 4(c) のようにレビュー工程と同様の依存関係でモデル化した．

最終品質をテスト完了時の残存不具合数と見なしているので，このようにモデル化することで設計工程からテスト工程までの依存関係が 1 つのグラフとして表現され，最終品質の予測を各工程の完了毎に行うことが可能となる．

4.4　確率分布の割り当て

各メトリクスに割り当てる確率分布は実際のプロジェクトデータから作成した．具体的には，実際のプロジェクトデータに記録されている各メトリクスの値を連続値から離散変数に変換する．次に，各メトリクスの離散変数が取りうる値のそれぞれにプロジェクトデータを分類する．そして，その結果得られる離散変数が取りうる値の度数分布をもとに確率分布を算出する．

以上のモデル化の結果，[6] で挙げられているモデルでは考慮されていないレビュー作業がモデルに追加されており，より実際のプロセスに近いモデルになっているといえる．また現在使用しているプロジェクトデータでは記録されていない問題の複雑さといったメトリクスが排除されており，過去に記録されたプロジェクトデータを有効に利用してモデルの確率分布を与えることが可能となっている．

以上の手順によって構築された予測モデルを図 5 に示す．モデル化の手順で述べたように，各ノードは 4 つのクラスが存在する．また，ノード右側にはその確率分布が示されている．

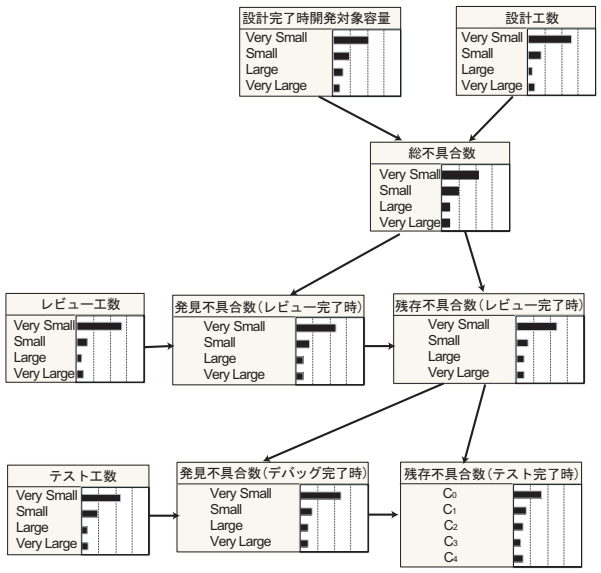

図 5　最終品質の予測モデル (確率分布付き)

5.　評　　価

5.1　評 価 方 法

モデルを構築する際に使用したデータをモデルに適用し，その結果を用いてモデルの評価を行う．予測結果が確率で返されることを考慮して，評価は次の手順で行う．

（1） 基準値を決める

モデルに観測値を代入していない状態におけるテスト完了時の残存不具合数の確率分布から，確率変数の取りうる値 C_0 から C_4 の確率をそれぞれ，$\theta_0, \theta_1, \theta_2, \theta_3, \theta_4$ と定める (図 6(a))．このとき，あるプロジェクト x のテスト完了時の残存不具合数の予測値が実際の観測値と一致する確率は $P_x = \theta_q$(q は実際の不具合数によって分類されるクラス) となる．

（2） 観測値の代入による予測

次に，評価に用いる実際のプロジェクト x で記録されているメトリクスをモデルに入力する．今回は実際に最終品質の予測を行う場合を考えて，レビュー完了時とテスト完了時のそれぞれの時点において最終品質の予測を行った．各時点で入力に使用するメトリクスは以下の通り．

- レビュー完了時
  設計完了時開発対象容量，設計工数，レビュー工数，発見不具合数 (レビュー完了時)
- テスト完了時
  設計完了時開発対象容量，設計工数，レビュー工数，発見不具合数 (レビュー完了時)，テスト工数，発見不具合数 (テスト完了時)

実際のプロジェクトの進行中には知ることの出来ない総不具合数と設計工程完了時，テスト完了時の残存不具合数は未知であるため入力はしない．

データを代入するとテスト完了時の残存不具合数の確率分布が変化するので，その時の確率変数がとる値が C_0 か

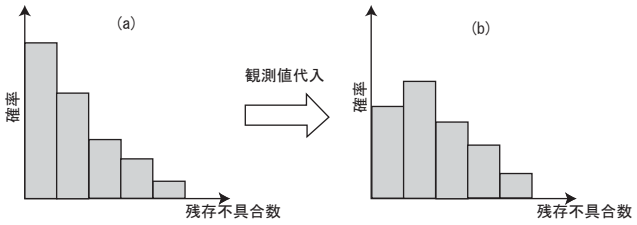

図 6　確率分布の変化

ら C_4 になる確率をそれぞれ，$\mu_0, \mu_1, \mu_2, \mu_3, \mu_4$ とする (図 6(b))．このとき，あるプロジェクト x の残存不具合数の予測値が実際の観測値と一致する確率は $P'_x = \mu_q$(q は実際の不具合数によって分類されるクラス) である．

（3） 基準値と予測値の比較

実際のプロジェクト x のデータを用いて，(2) の方法で P'_x を求め，(1) で得た P_x と比較する．ここで

$$P'_x > P_x$$

ならば，観測値をモデルに代入した結果，プロジェクト x のテスト完了時の残存不具合数の予測値が実際の観測値と一致する確率が増加しているので，モデルは正しい予測を行っていると考えることができる．

このように基準値と予測値の差の正負によって判定を行う理由は，使用しているプロジェクトデータのテスト完了時の残存不具合数に大きく偏りがあるためである．その結果，元から確率の低いクラスでは，確率の値が大きく増えたとしても，元から確率の高いクラスよりも確率が高くなるとは言えない．そこで，実際の残存不具合数と合致する確率が増えたかどうかで判定する方がより適切であると考えられるため，このような手法を採用している．

5.2 評価結果

評価に際しては，以下の 2 つの時点での予測結果を評価した．

（1） テスト完了時

まず，モデルの妥当性の確認のため，テスト完了時での評価を行った．その結果は表 3 の通りである．全体のうち，73.3%のプロジェクトでは，$P'_x > P_x$ が成り立っている．また，$P'_x > P_x$ であるプロジェクトでは，$P'_x - P_x$ の平均値は 10.2%であり，確率の増加の幅は十分に大きいと考えられる．

このことから，テスト完了時までに得られた観測値によってテスト完了時の残存不具合数がどのくらいになるのかを正しく示すモデルになっていると考えられる．

表 3　基準値とテスト完了時の確率の差

差の正負	プロジェクト数 (%)
正	165 (73.3%)
負	60 (26.4%)

（2） 設計完了時

次に，開発プロセスの早期段階における予測がどの程度有効であるかを評価するために，設計完了時までのプロジェクトデータを用いて評価を行った．その結果は表 4 のとおりである．この結果より，設計完了時の段階でも 68.8%のプロジェクトで $P'_x > P_x$ が成り立っている．また，$P'_x > P_x$ であるプロジェクトの $P'_x - P_x$ の平均値は 5.3%であり，テスト工程ほどではないが，確率の増加の幅は大きいと言える．

表 4　基準値と設計完了時の確率の差

差の正負	プロジェクト数 (%)
正	154 (68.8%)
負	71 (31.2%)

以上のことから，早期段階においてもある程度の最終品質予測が可能であると考えられる．

6. まとめと今後の課題

本研究では，実際の開発現場で収集されているメトリクスのみを用いた最終品質予測のモデルをベイジアンネットによって構築した．また，モデルの予測能力を確率の増減という側面から評価した．その結果，テスト工程終了時，設計工程終了時のいずれの時点においても観測値を代入することで，予測能力が向上することを確認できた．

今後の課題としては，次のようなものが挙げられる．まず，設計工程やテスト工程が実際には複数のサブ工程から構成されていることを考慮したモデルへの拡張である．次に，新規データによるモデルの評価である．モデルの評価手順ももう少し考える必要がある．また，現在は記録されているプロセスのメトリクスのみで予測を行っているが，より正確な予測を行うためにはやはり人的な要因を考慮する必要がある．

文　献

[1] V. R. Basili and J. D. Musa. The future engineering of software: A management perspective. *IEEE Computer*, 14(9):91–96, 1991.

[2] E. Charniak and R. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993.

[3] T. Compton and C. Withrow. Prediction and control of ada software defects. *Journal of Systems and Software*, 12:199–207, 1990.

[4] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[5] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Trans. on Software Engineering*, 25(5):675–689, 1999.

[6] N. E. Fenton and M. Neil. Software metrics and risk. In *Proc. 2nd European Software Measurement Conference(FESMA'99)*, pages 39–55, 1999.

[7] J. Forbes, T. Huang, K. Kanazawa, and S. Russel. The batmobile: Towards a Bayesian automated taxi. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1878–1885, 1993.

[8] P. Haddawy. An overview of some recent developments in Bayesian problem solving techniques. *AI Magazine*, 20(2):11–20, 1999.

[9] M. H. Halstead. *Elements of Software Science*. Elsevier, 1977.

[10] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, 1998.

[11] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, 1987.