

マルチバージョンソフトウェアのモジュール交換による高信頼化手法の提案

島 和之[†] 松本 健一[†] 鳥居 宏次^{†‡}

[†]奈良先端科学技術大学院大学情報科学科

[‡]大阪大学基礎工学部情報工学科

本稿では、マルチバージョンソフトウェアの信頼性を向上させる手法を提案する。この手法では、バージョン間でモジュールを交換することによって新たなバージョンを生成する。新たなバージョンは欠陥を含むモジュールの特定と除去に用いられる。また、本稿では提案する手法でのソフトウェア信頼性を推定する式を示す。典型的な値として、バージョンの故障率を 0.000698、バージョン数を 3、モジュール分割数を 10 とすると、提案する手法でのソフトウェアの故障率は N バージョンプログラミングでの故障率の約 10 分の 1 になると推定される。

A New Approach to Improve The Reliability of Multiversion Software by Exchanging Modules

K. Shima[†], K. Matsumoto[†] and K. Torii^{†‡}

[†]Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma
Nara 630-01, Japan

[‡]Department of Information and
Computer Sciences, Osaka University
1-3 Machikaneyama, Toyonaka
Osaka 560, Japan

This paper presents a new method to improve reliability of multiversion software. In the method the program versions of multiversion software are crossed to provide new program versions. The new program versions are used for searching and deleting faulty modules.

In this paper, a formula to estimate the reliability of the software in the method is presented. The failure probability in the method is estimated to be about a tenth of the failure probability in N-version programming where the failure probability of each version is 0.000698, the number of versions is 3 and the number of modules is 10.

1 はじめに

ソフトウェアシステムの信頼性を向上させる方法としてマルチバージョンプログラミングが考案され、実際のシステム開発で用いられている [7]。マルチバージョンプログラミングでは、同じ要求仕様に従って複数のチームが独立に開発したプログラムをバージョンと呼ぶ。各バージョンに混入された欠陥の多くは相互に独立であると考えられるので、同時に複数のバージョンが故障を発生する確率は一つのバージョンが故障を発生する確率よりも低くなる。よって、システムに対する入力を複数 (3 つ以上) のバージョンに与え、それらの出力の多数決によってシステムの出力を決定することで、システムの信頼性を向上させることができる。

マルチバージョンプログラミングによって開発されたソフトウェア (マルチバージョンソフトウェア) の信頼性を向上させる方法として Community Error Recovery (CER) が提案されている [5][8]。CER ではマルチバージョンプログラミングによって開発された複数のバージョンに共通のチェックポイントをおく。チェックポイントにおいてバージョンの状態を比較して誤りを検出し、回復することによってバージョンの故障を防ぐ。チェックポイントの数の増加にともなってソフトウェアの信頼性は向上すると考えられる。

しかし、チェックポイントもソフトウェアの一部であり、故障の原因となる可能性があるという指摘がなされている [6]。チェックポイントの故障を仮定すると、システムの信頼性を最も向上させる最適なチェックポイント数が存在し、チェックポイントの数がその値を越えて増加していくとシステムの信頼性は低下していく。さらにチェックポイントの数を増加させた場合、CER はマルチバージョンソフトウェアの信頼性を逆に悪化させることもありえる。また、異なるプログラムが常に対応するチェックポイントを同じ順番で通過する必要があるため、チェックポイントを挿入できる状況はかなり限定されている。

そこで、マルチバージョンソフトウェアの信頼性を向上させる方法としてソフトウェア品種改良

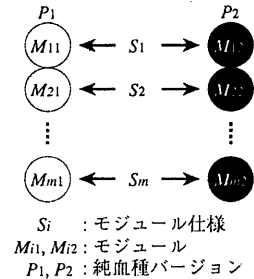


図 1: 純血種 (purebred) バージョン

法を提案する。ソフトウェア品種改良法では、バージョンを生物の個体とみなして交配を行い、より優れた個体を生成することによって、システムの品質を向上させる。ここで、各バージョンは複数のモジュールから構成され、対応するモジュールをバージョン間で交換することが可能であるように、共通のモジュール外部仕様書に従って独立に作成されていることを前提条件としている。ただし、モジュールが実行される順序は共通である必要がないため、CER よりも実現が容易であると考えられる。

2 節では、ソフトウェア品種改良法のアルゴリズムについて簡単に述べる。3 節では、ソフトウェア品種改良法のソフトウェアをモデル化し、ソフトウェア信頼性を推定するための式を導く。4 節では、ソフトウェア品種改良法を適用したソフトウェアの信頼性に関する評価を行なう。

2 ソフトウェア品種改良法

ソフトウェア品種改良 (Software Breeding) 法は、マルチバージョンソフトウェア [1] の信頼性を向上させる方法として提案された。ソフトウェア開発の設計工程において、ソフトウェアはモジュールに分割され、各モジュールの仕様が設計される。ソフ

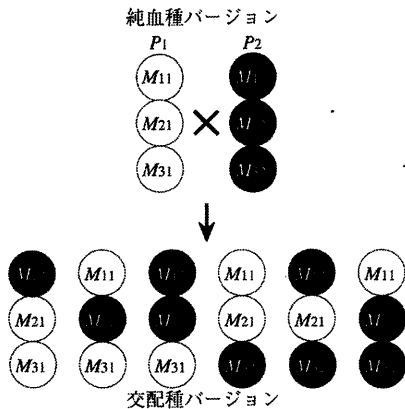


図 2: 交配

ソフトウェア品種改良法では、複数のチームが共通のモジュール仕様に従って独立にプログラム（バージョン）を作成するものとし、作成されたプログラムバージョンを純血種バージョン（purebred version）と定義する（図 1）。共通のモジュール仕様に従って作成されたモジュールは、共通のインターフェースを持つと考えられるので、モジュールに欠陥がない限り、互いに置き換えても同じ動作を行うと仮定する（モジュールの互換性の仮定）。

ソフトウェア品種改良法では、2つのバージョンのモジュールを交換することをバージョン間の交配（crossing）という。このとき、純血種バージョン間の交配によって生成されたバージョンを交配種バージョン（crossbred version）と定義する。また、純血種バージョンと交配種バージョンの交配によって生成されたバージョンも交配種バージョンと定義する（図 2）。このとき、ソフトウェア品種改良法は、交配を繰り返すことにより、純血種バージョンよりも優れた交配種バージョンを生成することと定義される。

図 3は、ソフトウェア品種改良法の流れを示している。システムは入力を与えられると、その入力を純血種バージョンに与えて故障検査を行う。故障検査で故障が検出された場合、不一致モジュール探索、多数派モジュール決定、少数派モジュール置

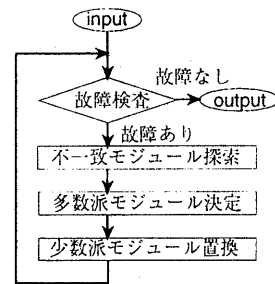


図 3: ソフトウェア品種改良法の流れ

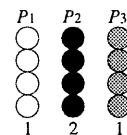


図 4: 故障検査

換によって純血種バージョンから欠陥を除去した交配種バージョンを生成し、再び故障検査を行う。これを、故障が検出されなくなるまで繰り返すことによって、信頼性の高い交配種バージョンを生成する。この交配種バージョンの出力をシステムの出力とすることによって、システムの信頼性を向上させる。

故障検査では、各バージョンにシステムの入力を与え、各バージョンの出力を比較する。このとき、異なる出力が存在する場合、故障を検出する。図 4は4つのモジュールで構成される3つの純血種バージョン（ P_1 , P_2 , P_3 ）の故障検査の例を示している。純血種バージョン P_1 , P_2 , P_3 は、それぞれ 1, 2, 1 を出力している。 P_1 と P_2 が出力が異なり、 P_2 と P_3 の出力が異なるので、故障が検出される。

不一致モジュール探索では、故障検査で検出された2つのバージョンの出力の違いの原因となった2つのモジュール（不一致モジュール）を二分探索によって探す。図 5は、 P_1 と P_2 の不一致モジュール探索の例を示している。矢印の順に、システム

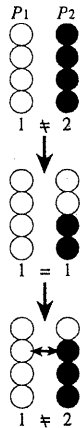


図 5: 不一致モジュール探索

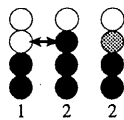


図 6: 多数派モジュール決定

の入力を与えた時の各バージョンの出力を比較する。この探索の結果、2 番目のモジュールが不一致モジュールであることが分る。

多数派モジュール決定では、不一致モジュール探索で見つかった 2 つのモジュールのうち、欠陥のないモジュール(多数派モジュール)を多数決によって決定する。図 6は、2 番目のモジュールが不一致モジュールである時の多数派モジュール決定を示している。この結果、各バージョンの出力の多数決によって、多数派モジュールは P_2 と P_3 のモジュールと決定される。

少数派モジュール置換では、欠陥のあるモジュール(少数派モジュール)を欠陥のないモジュール(多数派モジュール)に置き換える。図 7は、 P_1 の 2 番目のモジュールが少数派モジュールであり、 P_3 のモジュールが多数派モジュールである時の少数派

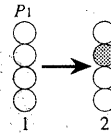


図 7: 少数派モジュール置換

モジュール置換の例を示している。次の故障検査では少数派モジュール置換によって生成された交配種バージョンが元の純血種バージョンの代わりに用いられる。

ソフトウェア品種改良法においてシステムが故障する状況として、以下の 2 通りが考えられる。

- 全純血種バージョンの故障

故障検査において、全ての純血種バージョンが故障し、それらの出力が全て同じ場合、故障の検出に失敗する。

- 多数派モジュールの欠陥

多数派モジュール決定において、欠陥のあるモジュールが多数派となった場合、欠陥のあるモジュールの除去に失敗する。

3 信頼性の推定

この節では、ソフトウェア品種改良法を適用したソフトウェアの信頼性を推定するための式を導出する。ここでは、ある入力についてソフトウェアが正しい出力を生成する確率をソフトウェアの信頼性と定義する。また、各プログラムバージョンは共通の仕様書に従って設計されているので、各入力に対するプログラムバージョンの出力はプログラムバージョンが故障しない限り一致すると仮定する。複数の異なる出力が正しいとみなせる場合もあり得るが、ここでは無視する。一方、故障した複数のプログラムバージョンの出力は一致すると仮定する。このようなことはほとんど起こらないと考えられるが、ここでは推定される信頼性が低くなる場合について考える。

ソフトウェア品種改良法ではモジュールレベルの多数決を行うので、全体が正しい出力を行うためには、共通のモジュール仕様に従って作成されたモジュールバージョンの最大多数が故障していないことが必要となる。共通のモジュール仕様 i に従っている N 個のモジュールバージョンのうちの特定の j 個が故障し、かつ、残りの $N - j$ 個が故障しない確率を $f_i(j)$ と定義する。モジュールバージョンが故障しない確率を χ_i とし、バージョンの故障が独立に発生すると仮定すると、

$$f_i(j) = (1 - \chi_i)^j \chi_i^{N-j}, \quad j = 0, 1, \dots, N.$$

しかし、バージョンの故障は必ずしも独立ではないということが指摘されている [2][3][4]。そのような場合には、 $f_i(j)$ として、実測値や推定値を用いる事が可能である。

特定の n 個のモジュールバージョン以外が故障せず、かつ、最大多数のモジュールバージョンが故障していない確率は、

$$r_i(n) = \sum_{j=0}^{\min(N-M, n)} \binom{n}{j} f_i(j), \quad n = 0, 1, \dots, N.$$

と示される。 M は最大多数とみなすことのできるバージョンの数である。ここでは、故障したプログラムバージョンの出力が一致すると仮定しているので、故障していないプログラムバージョンが最大多数を占めるためには過半数 $M = \lfloor \frac{N}{2} + 1 \rfloor$ 以上でなければならない。

全モジュール仕様についてモジュールバージョンの最大多数が故障せず、かつ、 N 個のプログラムバージョンのうちの特定の j 個が故障し、かつ、残りの $N - j$ 個が故障しない確率を $F(j)$ と定義する。ここで、 m はモジュール分割数である。全モジュール仕様についてモジュールバージョンの最大多数が故障せず、かつ、特定の n 個以外のプログラムバージョンが故障しない確率は、

$$\sum_{j=0}^n \binom{n}{j} F(j) = \sum_{i=1}^m r_i(n), \quad n = 0, 1, \dots, N.$$

よって、

$$F(n) = \begin{cases} \prod_{i=1}^m r_i(n), & n = 0 \\ \prod_{i=1}^m r_i(n) - \sum_{j=0}^{n-1} \binom{n}{j} F(j), & n = 1, 2, \dots, N. \end{cases}$$

このとき、全モジュール仕様についてモジュールバージョンの最大多数が故障せず、かつ、故障するプログラムバージョンの数が n 以下である確率は、

$$R(n) = \sum_{j=0}^n \binom{N}{j} F(j), \quad n = 0, 1, \dots, N.$$

故障したプログラムバージョンの出力が一致すると仮定したので、全バージョンが故障した場合は、故障検査で故障が検出されずにシステムは故障してしまうと考える。よって、ソフトウェア品種改良法でソフトウェアが故障しないためには、全モジュール仕様についてモジュールバージョンの最大多数が故障せず、かつ、故障するプログラムバージョンの数が $N - 1$ 以下であることが必要である。すなわち、その確率は、

$$R_{SB} = R(N - 1)$$

で示される。

4 信頼性の評価

この節では、それぞれ N バージョンプログラミングとソフトウェア品種改良法を適用したソフトウェアの信頼性を比較する。図 8 は、プログラムバージョンの信頼性を固定したときのモジュール分割数と故障率との関係を示している。ここでは、プログラムバージョンの故障率として [3] の実験から推定された値 0.000698 を用いている。モジュール分割数が 1 のときは両ソフトウェアの故障率は等しいが、モジュール分割数を増やしていくとソフトウェア品種改良法のソフトウェアの故障率は減少していく。

バージョン数を 3、モジュール数を 10 としたとき、 N バージョンプログラミングでの故障率は

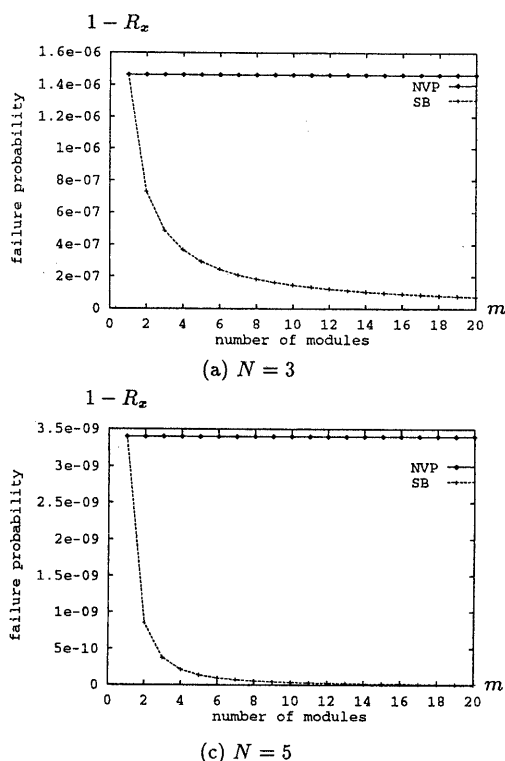


図 8: モジュール数と故障率

1.461×10^{-6} , ソフトウェア品種改良法での故障率は 1.465×10^{-7} となり, 故障率が約 10 分の 1 になっている。モジュール数を 20 としたときは, N バージョンプログラミングでの故障率は変わらず, ソフトウェア品種改良法での故障率は 7.342×10^{-8} となり, 故障率が約 20 分の 1 になっている。

バージョン数を 5, モジュール数を 10 としたとき, N バージョンプログラミングでの故障率は 3.397×10^{-9} , ソフトウェア品種改良法での故障率は 3.403×10^{-11} となり, 故障率は約 100 分の 1 になっている。モジュール数を 20 としたときは, N バージョンプログラミングでの故障率は変わらず, ソフトウェア品種改良法での故障率は 8.509×10^{-12} となり, 故障率が約 400 分の 1 になっている。

図 9 は, モジュール分割数を固定したときのプログラムバージョンの信頼性と信頼性向上の度合

いとの関係を示している。ここでは, N バージョンプログラミングによるマルチバージョンソフトウェアの故障率をソフトウェア品種改良法によるマルチバージョンソフトウェアの故障率で割った値で信頼性向上の度合いを示している。モジュール分割数は 10 としている。

図より, バージョンの信頼性が上がる (故障率が下がる) に従って, 信頼性向上の度合いが上がり, ある上限に近付くことがわかる。バージョン数が 3 のときは故障率が約 10 分の 1 に, バージョン数が 5 のときは故障率が約 100 分の 1 に近づいている。

上記の結果より, 以下のそれぞれの場合にソフトウェア品種改良法による信頼性向上の度合いは大きくなることが分かる。

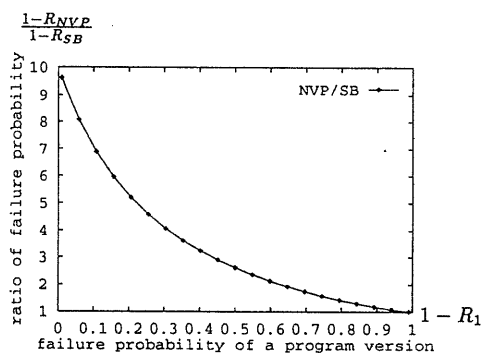
1. バージョン数が多い場合
2. モジュール分割数が多い場合
3. バージョンの信頼性が高い場合

5 おわりに

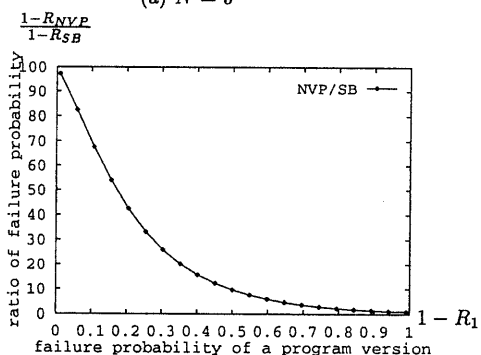
ソフトウェア品種改良法は, マルチバージョンソフトウェアの信頼性を向上させるための一つの手法である。この方法では, バージョン間のモジュール交換によって新しいバージョンを生成することによって故障を防ぐ。このため, バージョンの内部にチェックポイントのようなルーチンを追加する必要がない。

本稿では, ソフトウェア品種改良法を用いたソフトウェアをモデル化し, その信頼性を推定する式を導いた。この式を用いる事で, ソフトウェアの目標とする信頼性やモジュール分割数やバージョンの数から, 各バージョンに必要な信頼性を決定できる。こうして求めた信頼性を達成するように各バージョンは開発されテストされる。

ソフトウェア品種改良法では, 全バージョンが共通のモジュール外部仕様に従って作成される。このため, このモジュール外部仕様に欠陥が含まれていた場合には, 故障を防ぐ事ができない。ソフ



(a) $N = 3$



(c) $N = 5$

図 9: バージョンの故障率と信頼性向上の度合い

トウェア品種改良法では、モジュール外部仕様の欠陥はレビューによって検出し、除去しておかなければならない。マルチバージョンプログラミングではモジュール外部仕様も独立に設計されるので、その欠陥による故障を防ぐ事ができるかもしれない。このような場合を想定した評価は、今後の課題である。

参考文献

- [1] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during program execution," Proc. COMPSAC '77, Chicago, pp. 149-155, Nov. 1977.
- [2] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," IEEE Trans. Software Eng., vol. SE-11, no. 12, pp. 1511-1517, Dec. 1985.
- [3] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming," IEEE Trans. Software Eng., vol. SE-12, no. 1, pp. 96-109, Jan. 1986.
- [4] B. Littlewood and D. R. Miller, "A conceptual model of multi-version software," Proc. FTCS-17, Pittsburgh PA, pp. 150-155, July 1987.
- [5] J. D. Musa, A. Iannino and K. Okumoto, Software Reliability: Measurement, Prediction, Application, pp. 13-18, McGraw-Hill, 1987.
- [6] V. F. Nicola and A. Goyal, "Modeling of correlated failures and community error recovery in multiversion software," IEEE Trans. Software Eng., vol. 16, no. 3, pp. 350-359, Mar. 1990.
- [7] P. Rook, Software Reliability Handbook, pp. 96-110, Elsevier Science Publishing, 1990.
- [8] K. S. Tso and A. Avizienis, "Community error recovery in N-version software: A design study with experimentation," Proc. FTCS-17, Pittsburgh PA, pp.127-133, July 1987.