論　文

モジュール交換手法によるマルチバージョンソフトウェアの信頼性向上

島　和之[†]　　松本　健一[†]　　鳥居　宏次[†]

Reliability Improvement of Multiversion Software by Exchanging Modules

Kazuyuki SHIMA[†], Ken'ichi MATSUMOTO[†], and Koji TORII[†]

あらまし　本論文では，開発するバージョンの数を増やさずに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．文献[11]で提案されたコミュニティエラーリカバリでは，バージョン中に複数のチェックポイントを設定し，チェックポイントにおいてバージョンの誤りを検出し修復する．これにより，バージョンの故障を防ぎ，バージョン数を増やすことなくソフトウェアの信頼性を向上させる．しかし，誤りの検出と修復のための処理も故障し得ると仮定すると，マルチバージョンソフトウェアの信頼性が低下することが指摘されている．本論文で提案する手法では，共通のモジュール仕様に従って開発された複数のバージョンとそれらのバージョンを制御するドライバを用意する．ドライバはバージョンの故障を検出すると，欠陥を含むモジュールを特定し，それを他のモジュールに置き換える．これにより，欠陥を含まないバージョンを生成し，マルチバージョンソフトウェアの信頼性を向上させる．バージョンの故障率を 0.000698，バージョン数を 5，モジュール分割数を 20 とした場合，本手法によりマルチバージョンソフトウェアの故障率は約 100 分の 1 になると推定される．

キーワード　ソフトウェア信頼性，耐故障システム，コミュニティエラーリカバリ，チェックポイント故障

1. まえがき

コンピュータシステムの応用分野が広がり，その役割が重要になればなるほど，その故障が社会に及ぼす影響が大きくなってくる．例えば，鉄道・原子炉・航空機・医療機器などの制御システムに故障が起きると，人命に危害の及ぶ可能性がある．また，航空会社の座席予約システム，証券会社の契約システム，製造業の生産ライン制御システムといった企業の業務に関わるコンピュータシステムがダウンすれば，その企業は多大な損害を被る．このようなシステムでは，システムの耐故障性（システムの一部に欠陥があったとしても，システムが故障しにくい性質）が重要になってくる[10]．

ソフトウェアの耐故障性を実現する方法の一つとして，マルチバージョンプログラミングがある[5],[7]．マルチバージョンプログラミングでは，複数のバージョン（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）とドライバ（バージョンを制御するプログラム）を用意する[1]．ドライバはシステム外部からの入力を各バージョンに与え，それらの出力の多数決を行い，最大多数となった出力をシステム外部に出力する．このため，少数のバージョンが故障して誤った出力を行っても，故障していないバージョンの出力がシステム外部に出力されるので，外部からはシステムが故障していないように見える．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をマルチバージョンソフトウェアと呼ぶ．

バージョンの数を増やすことによってマルチバージョンソフトウェアの信頼性は向上するが，開発コストも増大する．バージョンの数を増やさずに，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コミュニティエラーリカバリ（Community Error Recovery，CER）が提案されている[12]．コミュニティエラーリカバリ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対応する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，修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

† 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, Nara Institute of Science and Technology, Ikoma-shi, 630-01 Japan

トの数の増加に伴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，チェックポイントにおいて誤りが起こり得ると仮定すると，コミュニティエラーリカバリ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るという指摘がなされている[6]．

本論文では，バージョンの数を増やさずに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増殖法（Software Breeding）を提案する．この方法では，共通のモジュール仕様に従って独立に開発されたバージョン間で対応するモジュールを交換することにより，新たなバージョンを生成し，故障の原因となっているモジュールの検出と除去に用いる．バージョン中にチェックポイントを挿入しないため，チェックポイントによる故障を考慮する必要がない．2. では，コミュニティエラーリカバリについて述べる．3. では，ソフトウェア増殖法のアルゴリズムについて簡単に述べる．4. では，ソフトウェア増殖法を適用した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．5. では，ソフトウェア増殖法を適用したソフトウェアの信頼性に関する評価を行う．

2. コミュニティエラーリカバリ

コミュニテイエラーリカバリでは，チェックポイントとリカバリポイントの2種のポイントで誤りの修復を行う．チェックポイントは故障したバージョンの誤った結果を故障していないバージョンの結果に置き換えて部分的に誤りを修復する．リカバリポイントは，いくつかのチェックポイントを含むプログラムモジュールの間に挿入され，故障したバージョンの誤った状態を修復する．

標準的なマルチバージョンプログラミングの場合，各バージョンはチェックポイントにおいてチェックポイント識別子，フォーマット文字列，および状態変数へのポインタ集合をドライバへ送る．フォーマット文字列は比較される状態変数の型と数を示す．ドライバは状態変数へのポインタによって計算結果を得る．

コミュニテイエラーリカバリでは，標準的なチェックポイントを拡張して，故障を遮へいする機能だけでなく故障したバージョンの誤り検出と部分的誤り修復の機能をもつチェックポイントを用いる．チェックポイントにおいて各バージョンは計算結果をドライバに提出する．ドライバはそれらの結果について多数決を行い，決定結果を欠陥バージョンに送り返す．このとき検出された誤りの記録はリカバリポイントにおける修復のために蓄積される．もし結果がプログラムの出力であれば，ドライバは多数決による決定結果を出力するので，少数派バージョンの故障は遮へいされる．

リカバリポイントには，リカバリポイント識別子，状態入力例外ハンドラ，および状態出力例外ハンドラが記述される．リカバリポイントにおいてはすべてのバージョンの状態変数が等しくなることが要求される．リカバリポイントにおいて各バージョンはドライバにリカバリポイント識別子を送る．リカバリポイント識別子が送られない場合，または，送られたリカバリポイント識別子が他のバージョンが送ったリカバリポイント識別子と異なる場合，または，前のリカバリポイント以降に通過したチェックポイントにおいて誤りが検出された場合，バージョンの故障が検出される．例外ハンドラは故障したバージョンが検出されたときにドライバによって呼び出される．ドライバは正しいバージョンの状態出力例外ハンドラにバージョンの状態を出力させ，多数決によって決定状態を生成し，故障したバージョンの状態入力例外ハンドラに決定状態を入力させる．リカバリポイント識別子が送られない場合，または，送られたリカバリポイント識別子が正しくない場合，制御フローに欠陥のあるバージョンが検出される．制御フローの欠陥が検出されたバージョンは，それを検出したリカバリポイントから再起動される．

あるチェックポイントまたはリカバリポイントから次のチェックポイントまたはリカバリポイントまでの処理が少ないほど，その間に各バージョンが故障する確率は小さくなる．各バージョンが故障する確率が小さいほど，それらの多数決において正しいバージョンが決定される確率が大きくなる．よって，各バージョンの大きさを一定とすると，チェックポイントやリカバリポイントが多いほど，誤りを修復できる可能性が高くなる．

しかし，チェックポイントやリカバリポイントもソフトウェアの一部であり，故障する可能性がある．文献[6]では，チェックポイントにおいて正しいバージョンの状態変数が間違った値に変更されるというチェックポイント故障を想定し，その生起確率を 10^{-7} とした場合のソフトウェアの信頼性を推定している．その結果，チェックポイントの数が増えるとチェックポイント故障の影響が大きくなり，マルチバージョンソフ

トウェアの信頼性を悪化させてしまうことが示された．

3. ソフトウェア増殖法

ソフトウェア増殖法は，マルチバージョンソフトウェアの信頼性を向上させる方法の一つである．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．ソフトウェア増殖法では，複数のチームが共通のモジュール仕様 $S_i(i=1,\cdots,m)$ 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 $P_j(j=1,\cdots,N)$ を初期バージョン（initial version）と定義する（図1）．共通のモジュール仕様 $S_i(i=1,\cdots,m)$ に従って作成されたモジュール ${}_iM_j(j=1,\cdots,N)$ は，共通のインタフェースをも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，バージョンは同じ動作を行うと仮定する．

図2は，ソフトウェア増殖法で用いるドライバの処理の流れを示している．ドライバは入力を与えられると，その入力を初期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換によって初期バージョンから欠陥を除去した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，信頼性の高いバージョンを生成する．生成されたバージョンの出力をドライバの出力とすることによって，ソフトウェアの信頼性を向上させる．

故障検査では，各バージョンにドライバの入力を与え，各バージョンの出力を比較する．すべてのバージョンの出力が等価であれば，それをドライバの出力とする．異なる出力が存在する場合，バージョンの故障を検出する．図3は四つのモジュールで構成される三つの初期バージョン (P_1, P_2, P_3) の故障検査の例を示している．初期バージョン P_1, P_2, P_3 は，それぞれ 1, 2, 1 を出力している．P_2 の出力が P_1, P_3 の出力と異なるので，故障が検出される．

不一致モジュール探索では，不一致モジュール（故障検査で検出された二つのバージョンの出力の違いの原因となった二つのモジュール）を2分探索によって探す．図4は，P_1 と P_2 の出力が異なっている場合の不一致モジュール探索の例を示している．(a), (b) の順に，ドライバの入力を与えたときの各バージョンの出力を比較する．C_1 は1番目と2番目のモジュールが P_1 と同じであり，3番目と4番目のモジュールが P_2 と同じである．C_1 の出力は P_1 の出力と等しく，P_2 の出力とは異なる．C_1 と P_2 とでは1番目と2番目のモジュール以外のモジュールは同じであるので，1番

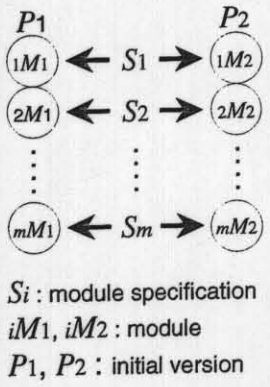


図1 初期バージョン
Fig. 1 Initial version.

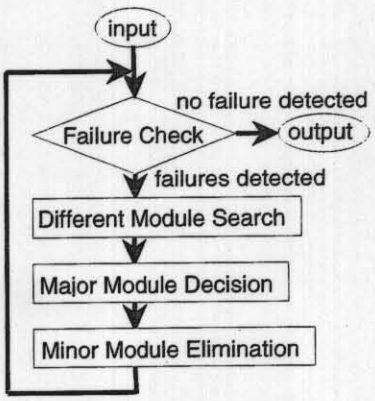


図2 ソフトウェア増殖法
Fig. 2 Software breeding.

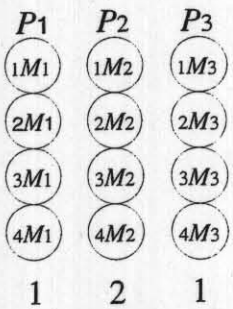


図3 故障検査
Fig. 3 Failure check.

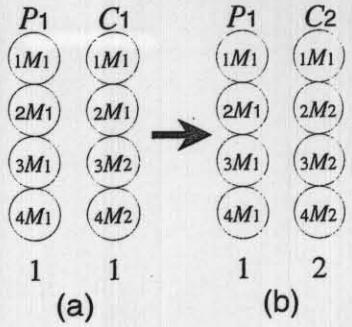


図4　不一致モジュール探索
Fig. 4　Inconsistent module search.

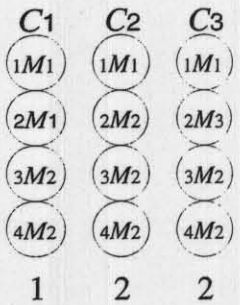


図5　多数派モジュール決定
Fig. 5　Major module decision.

目または2番目のモジュールが不一致モジュールであることがわかる．C_2 は1番目のモジュールが P_1 と同じであり，2〜4番目のモジュールが P_2 と同じである．C_2 の出力は P_1 の出力と異なっている．つまり，C_1 の出力と C_2 の出力は異なる．C_1 と C_2 とでは2番目のモジュール以外のモジュールは同じであるので，2番目のモジュール ${}_2M_1, {}_2M_2$ が不一致モジュールであることがわかる．

多数派モジュール決定では，i 番目のモジュール ${}_iM_j, (j = 1, \cdots, N)$ の中に不一致モジュールがある場合に，i 番目のモジュールのみ異なり，i 番目以外のモジュールが同じであるバージョン $C_j (j = 1, \cdots, N)$ の出力について多数決を行い，多数派の出力を生成したバージョンの i 番目のモジュールを多数派モジュールとする．図5は，不一致モジュール探索において C_1, C_2 の出力が異なり，モジュール ${}_2M_1, {}_2M_2$ が不一致モジュールである場合の多数派モジュール決定の例を示している．バージョン C_1, C_2, C_3 を構成するモジュールは ${}_2M_1, {}_2M_2, {}_2M_3$ のみが異なり，他のモジュールは同じである．この例では，C_2 と C_3 の出力が等しく，C_1 の出力とは異なっているので，多数派モジュールは ${}_2M_2$ と ${}_2M_3$ と決定される．

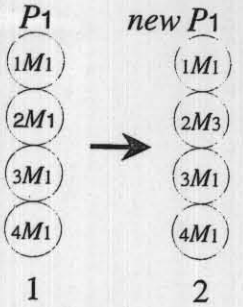


図6　少数派モジュール置換
Fig. 6　Minor module elimination.

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6は，モジュール ${}_2M_1$ が少数派モジュールであり，モジュール ${}_2M_3$ が多数派モジュールであるときの少数派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バージョン $newP_1$ がもとのバージョン P_1 の代わりに用いられる．但し，新たな入力系列では初期バージョンが用いられる．これは，新たな入力系列に対しても置換後のバージョンを用いると，2個のモジュールに共通故障が生じる危険，すなわち，置換による故障の伝搬が懸念されるからである．

故障検査，不一致モジュール探索，多数派モジュール決定および少数派モジュール置換のバージョンの仕様に依存しない部分のアルゴリズムおよび不一致モジュール探索の停止性と正当性については[8]で示している．ソフトウェア増殖法において，バージョンを制御するドライバがバージョンの仕様に依存する部分は，バージョンに入力を与えるルーチン，バージョンの出力を受けとるルーチンおよびバージョンの出力を比較するルーチンのみである．これらのルーチンは，マルチバージョンソフトウェアでは必ず必要になる部分である．よって，ソフトウェア増殖法のドライバを作成するために必要となる追加の開発コストはほとんどない．

4.　信頼性の推定

以下の場合，ソフトウェア増殖法のソフトウェアは故障する．

- 全初期バージョンの故障

故障検査において，すべての初期バージョンが故障

し，それらの出力がすべて同じ場合，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．

- ドライバの故障

故障検査・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換を行うドライバに欠陥があり，故障する．

この章では，ソフトウェア増殖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によってソフトウェアの信頼性を示す．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．

コミュニティエラーリカバリ法では，チェックポイントにおいてモジュールバージョン間の多数決を行う．ソフトウェア増殖法では，多数派モジュール決定においてモジュールバージョン間の多数決を行う．どちらの手法においても，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である．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，

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

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]～[4]が，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て信頼性を推定することが可能である．

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，

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

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．ソフトウェア増殖法の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換が行われ，故障したモジュールが除去される．この故障モジュールの除去過程でドライバが故障する確率を f_{SBD} とおくと，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せず，かつ，故障モジュールの除去過程でドライバが故障しない確率は，

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

と示される．

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，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，

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

よって，

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

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，故障したプログラムバージョンの数が n 以下である確率は，

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

M 個以上のプログラムバージョンが故障しなければ，すなわち，故障したプログラムバージョンの数が $N - M$ 以下であれば，Nバージョンプログラミングのソフトウェアは故障しない．その確率は，

$$R_{NVP} = R(N - M)$$

で示される．

コミュニティエラーリカバリ法では，チェックポイントにおいて誤りを検出し，リカバリポイントにおいて誤りを修復することによって信頼性を向上させる．このため，全プログラムバージョンで故障が発生したとしても，それらの故障が異なるチェックポイントにおいて発生すれば，誤りを検出して修復することができる．但し，チェックポイントにおいて正しく誤りを検出するためには，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，チェックポイントにおいて故障が起きないことが必要である．よって，チェックポイントの故障率を f_{CP} とおくと，コミュニティエラーリカバリのソフトウェアが故障しない確率は，

$$R_{CER} = (1 - f_{CP})^m R(N)$$

で示される．

ソフトウェア増殖法では，過半数のプログラムバージョンが故障したとしても，故障の原因となっているモジュールを検出し，それらを除去したプログラムバージョンを生成することによって信頼性を向上させる．但し，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，ソフトウェア増殖法の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，ソフトウェア増殖法の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．また，ソフトウェア増殖法では，チェックポイントを用いないため，チェックポイント故障を考慮する必要はない．よって，モジュール交換法のソフトウェアが故障しない確率は

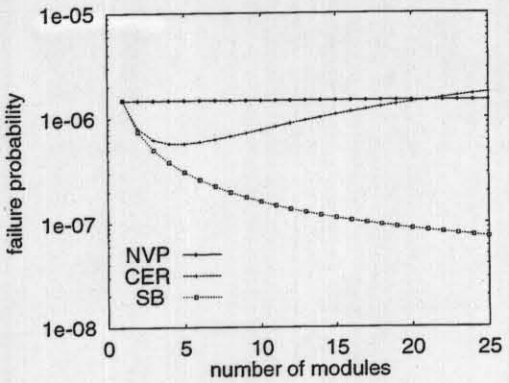
$$R_{SB} = R(N - 1)$$

で示される．

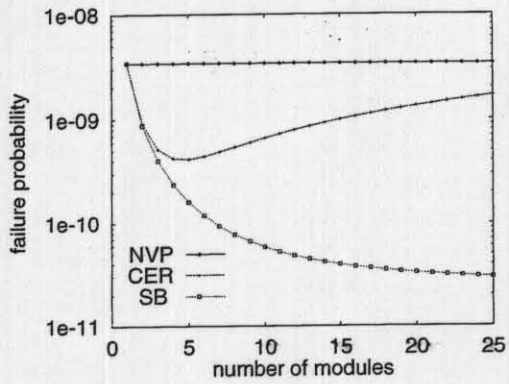
5. 信頼性の評価

この節では，Nバージョンプログラミングとコミュニティエラーリカバリとソフトウェア増殖法を適用したそれぞれのソフトウェアの信頼性を比較する．図7は，バージョンのモジュール分割数を変化させた場合のマルチバージョンソフトウェアの故障率の変化を示している．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．ソフトウェア増殖法における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換は Pascal プログラミング言語を用いて約 50 行で記述できる[9]．1行当りの故障率を一定の f と仮定すると，f が十分に小さいとき 50 行での故障率は $1 - (1 - f)^{50} \approx 50f$ である．一方，チェックポイント一つ当り少なくとも 1 行は必要であるから，ソフトウェア増殖法のドライバの故障率はチェックポイントの故障率の約 50 倍以下と見積もることができる．但し，プログラムの故障率はプログラムの規模からだけで決まるものではないので，ここでは大きめに 100 倍とする．

図7(a)はバージョン数を 3，コミュニティエラーリカバリのチェックポイントにおける故障率をプログラムバージョンの故障率の1万分の1とした場合のバージョンのモジュール分割数に対するマルチバージョンソフトウェアの故障率の関係を示している．モジュール分割数が1のときは各手法を適用したソフトウェアの故障率は等しく 1.461×10^{-6} である．モジュール分割数を増やした場合，コミュニティエラーリカバリとNバージョンプログラミングのソフトウェアの故障率は減少する．しかし，コミュニティエラーリカバリを適用したソフトウェアはモジュール分割数が 5 のとき，故障率が最小値 5.717×10^{-7} となり，モジュール分割数を 5 よりも大きくすると故障率が増加し，モジュール分割数が 21 のとき Nバージョンプログラミングよりも故障率が高くなる．一方，ソフトウェア増殖法のソフトウェアの故障率は単調に減少していく．モジュール分割数を 10 としたとき，ソフトウェア増殖法のソフトウェアの故障率は 1.611×10^{-7} となり，Nバージョンプログラミングのソフトウェアの故障率の約 9 分の 1 になる．モジュール数を 20 としたときは，ソフトウェア増殖法のソフトウェアの故障率は 8.804×10^{-8} となり，Nバージョンプログラミングのソフトウェアの故障率の約 17 分の 1 になる．

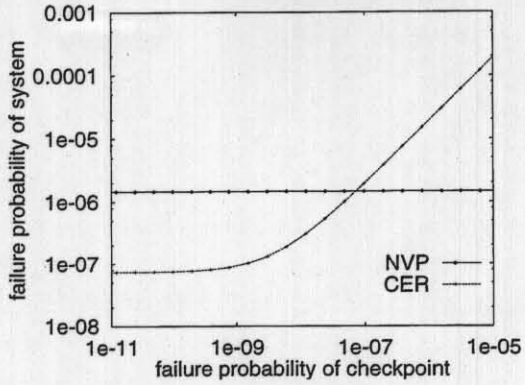


(a) $N = 3$

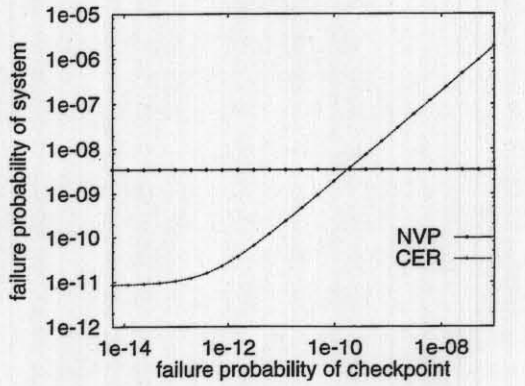


(b) $N = 5$

図7 モジュール数の影響
Fig. 7 Effect of the number of modules.



(a) $N = 3$



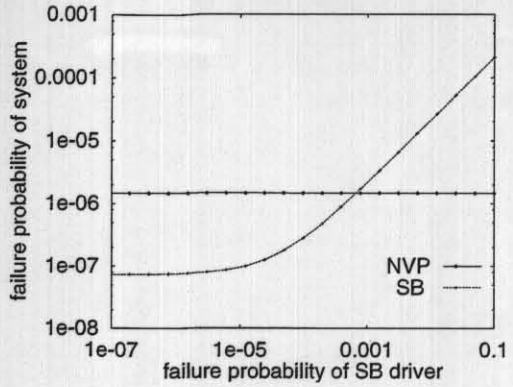
(b) $N = 5$

図8 チェックポイント故障の影響
Fig. 8 Effect of the checkpoint failure.

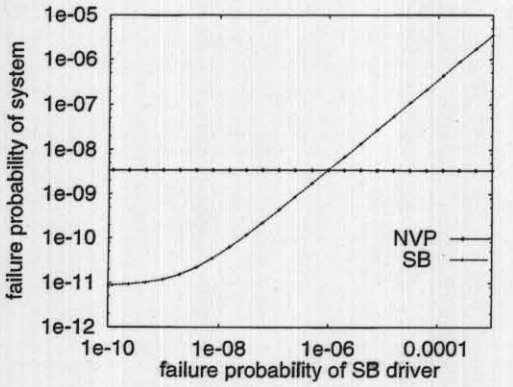
図7(b)はバージョン数を5，チェックポイントにおける故障率をプログラムバージョンの故障率の100万分の1とした場合のバージョンのモジュール分割数に対するマルチバージョンソフトウェアの故障率の関係を示している．Nバージョンプログラミングのソフトウェアの故障率はモジュール分割数にかかわらず 3.397×10^{-9} である．モジュール分割数が5のときコミュニティエラーリカバリのソフトウェアの故障率は最小値 4.153×10^{-10} となる，ソフトウェア増殖法のソフトウェアの故障率はモジュール分割数の増加に伴って単調減少し，モジュール分割数が10のとき故障率は 5.84×10^{-11} となり，モジュール数が20のとき故障率は 3.288×10^{-11} となる．それぞれ，Nバージョンプログラミングのソフトウェアの故障率の約58分の1と約103分の1である．

上記の結果より，バージョン数が多い場合とモジュール分割数が多い場合にソフトウェア増殖法による信頼性向上の度合は大きくなることがわかる．これは，プログラムバージョンの信頼性を固定しているので，モジュール分割数が増えると一つのモジュール当りの信頼性が高くなり，多数派モジュール決定において故障していないモジュールを正しく決定できる確率が高くなるからである．

図8はモジュール分割数を20としたときのコミュニティエラーリカバリのチェックポイントにおける故障率に対するシステムの故障率の関係を示している．この図より，バージョン数が3の場合はチェックポイントにおける故障率が約 1×10^{-7} のとき，バージョン数が5の場合はチェックポイントにおける故障率が約 2×10^{-10} のとき，それぞれの場合のNバージョンプログラミングによるソフトウェアとコミュニティエラーリカバリによるソフトウェアの故障率が等しくな



(a) $N = 3$



(b) $N = 5$

図9　ソフトウェア増殖法のドライバの故障の影響
Fig.9　Effect of the failure of SB driver.

る．すなわち，コミュニティエラーリカバリによって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には，バージョン数が3の場合はチェックポイントにおける故障率を 1×10^{-7} 以下に，バージョン数が5の場合はチェックポイントにおける故障率を 2×10^{-10} 以下にしなければならないことがわかる．

図9はモジュール分割数を20としたときのソフトウェア増殖法におけるドライバの故障率に対するシステムの故障率の関係を示している．この図より，バージョン数が3の場合はソフトウェア増殖法におけるドライバの故障率が約0.001のとき，バージョン数が5の場合はソフトウェア増殖法におけるドライバの故障率が約 1×10^{-6} のとき，それぞれの場合のNバージョンプログラミングを用いたソフトウェアとソフトウェア増殖法を用いたソフトウェアの故障率が等しくなる．すなわち，ソフトウェア増殖法によって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には，バージョン数が3の場合はソフトウェア増殖法におけるドライバの故障率を0.001以下に，バージョン数が5の場合はソフトウェア増殖法におけるドライバの故障率を 1×10^{-6} 以下にしなければならないことがわかる．

すなわち，バージョン数が同じ場合で比較すると，コミュニティエラーリカバリのチェックポイントはソフトウェア増殖法におけるドライバの故障率の5,000～10,000分の1以下の故障率を達成しなければならない．ソフトウェア増殖法のドライバはコミュニティエラーリカバリのチェックポイントに比べて複雑な動作を行うが，個々のシステムに依存する部分は各バージョンに入力を与えて，各バージョンの出力を比較するルーチンだけであり，それよりも上位のルーチンは再利用可能であるため十分に信頼性を高めることができると考えられる．

6. むすび

本論文では，バージョンの数を増やさずに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について述べた．提案する手法では，バージョンのモジュールを他のバージョンのモジュールと交換することによって新しいバージョンを生成し，バージョンの故障の原因となっているモジュールを検出し除去する．提案する手法においてバージョンを制御するドライバの大部分は再利用可能であり，提案する手法をマルチバージョンソフトウェアに適用するためにかかる追加の開発コストは小さい．

また本論文では，提案する手法を適用したソフトウェアをモデル化し，その信頼性を推定するための式を示した．バージョンの信頼性，バージョンの数，モジュール分割数が同じマルチバージョンソフトウェアに，従来法であるコミュニティエラーリカバリと提案する手法のそれぞれを適用したソフトウェアの信頼性を推定し，比較した結果より，提案する手法が従来法よりも高い信頼性を達成できることを示した．

ソフトウェア増殖法では，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，故障を防ぐことができない．ソフトウェア増殖法では，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こと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．

謝辞 関連文献の調査に際して御世話になった大阪大学菊野亨教授ならびに井上克郎教授に感謝致します．また，本論文に対し，貴重な御意見を下さいました査読者の方々に感謝致します．

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, pp.149–155, 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, 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, 1986.

[4] B. Littlewood and D.R. Miller, "A conceptual model of multi-version software," Proc. FTCS-17, pp.150–155, 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, 1990.

[7] P. Rook, "Software Reliability Handbook," pp.96–110, Elsevier Science Publishing, New York, 1990.

[8] K. Shima, K. Matsumoto, and K. Torii, "A mathematical comparison of software breeding and community error recovery in multiversion software," Proc. International Symposium on Software Reliability Engineering, pp.192–201, 1993.

[9] K. Shima, K. Matsumoto, and K. Torii, " A new method for increasing the reliability of multiversion software systems using software breeding," Proc. International Symposium on Software Reliability Engineering, pp.202–208, 1995.

[10] 当麻喜弘，南谷　崇，藤原秀雄，"フォールトトレラントシステムの構成と設計，" 槇書店，1991.

[11] K.S. Tso, A. Avizienis, and J.P.J. Kelly, "Error recovery in multi-version software," Proc. IFAC Workshop SAFECOMP'86, pp.35–41, 1986.

[12] 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, 1987.

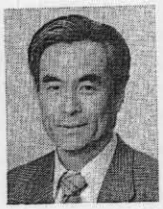
(平成7年6月26日受付，11月17日再受付)

島　和之　(正員)

平3阪大・基礎工・情報卒．平6同大大学院博士課程中退．同年奈良先端科学技術大学院大学・助手．工修．高信頼性ソフトウェアの研究に従事．

松本　健一　(正員)

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．



鳥居　宏次　(正員)

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