ソフトウェア工学 54－4
（1987. 6. 24）

ソフトウェア信頼性モデルに関する一考察
―― 劣化特性を考慮した冗長的モデル ――

張 学健　　菅野 文友
東京理科大学　工学部　経営工学科

本報告では、ソフトウェア製品のライフサイクルにおける劣化特性について検討した。信頼性工学の基本的な考え方の一つとしてのバスタブ曲線が、ソフトウェア製品の場合にも適用できるという観点から、ソフトウェア製品の故障時間分布に対応して、ワイブル分布を導入した。ソフトウェア製品は、間欠的に使用される複数のコンポーネント（機能モジュール）から構成されているシステムとする。各コンポーネントの使用確率を検討し、その使用確率がソフトウェア製品の信頼度に対する影響を調べた。ソフトウェア製品の運用段階における事後保全（リビジョンアップ）および予防保全（バージョンアップ）は、ソフトウェア製品の稼動信頼性に対して、無視できない影響を与える。本報告では、こういった影響を吟味し、バージョンアップの効果を表わすバージョンアップ効果関数も導入して、若干の定式化を試みた。

A STUDY ON SOFTWARE RELIABILITY MODEL
— REDUNDANT MODEL WITH CONSIDERATION OF DETERIORATION CHARACTRISTICS —

Xue-jian ZHANG　and　Ayatomo KANNO
Dept. of Management Science, Faculty of Engineering, Science University of Tokyo
1-3, Kagurazaka, Shinjukuku, TOKYO, 162 JAPAN

In this paper, the characteristics of deterioration in software product life cycle are discussed. In reliability engineering, bath-tub curve is one of the most fundamental concept. From the point of view that bath-tub curve is also applicable in software product, Weibull distribution is introduced for our work. Software products are considered to be the systems composed by mutiple intermittently-used components (i.e. function modules).The oparetional probabilities of these components are investigated, and the effects of the oparetional probabilities on reliabilities of the software products are discussed.In application phase of software products, corrective maintenance(revision up) and preventive maintenance(version up) have significant effects on the reliabilities of software products. In this paper, for consideration of these factors, the version up function is introduced to represent the effect of version up, and some numerical examples are discussed.

１．はじめに

現在、ソフトウェアの製品としての工業化生産が、目覚ましく発展している。これと対応するように、ソフトウェア製品の品質管理／信頼性についての研究が盛んに行なわれている。ソフトウェア製品の信頼性を検討する際に、当該ソフトウェア製品の内部に存在するバグおよびソフトウェア製品のシステムについてとしての故障率が注目されていて、多数のソフトウェア信頼性モデル提案された。ここでは、ソフトウェア製品のライフサイクルに関しての観点から、ソフトウェア製品製品の劣化パターンを吟味し、それと対応するような定式化を図る。一方、ソフトウェア製品が、複数のコンポーネント（機能モジュール）から構成される一つのシステムとして考えられる。こういった機能モジュールは、間欠的に使用されるので、各機能モジュールそのものの性質だけではなく、各機能モジュールの使用状態の、ソフトウェア製品全体の信頼性に対する影響を考え、また、システムとしてのソフトウェア製品の構成に注目し、その冗長性を吟味すべきである。したがって、ソフトウェア製品の劣化特性を踏まえた上で、ソフトウェア製品を構成する各機能モジュールの使用確率を導入し、システムとしてのソフトウェア製品の信頼性モデルに関する検討を行なう。

２．ソフトウェア製品の劣化特性

２．１．ソフトウェア製品におけるバスタブ曲線（ bath-tub curve ）

ソフトウェア製品は、通常の製品と同じように、開発・設計、製造・試験、検査および運用・保全という生産から廃棄までのライフサイクルが見られる。したがって、信頼性工学の基本的な考え方としてのバスタブ曲線は、ソフトウェア製品のライフサイクルを検討する際にも、有効と思われる。

バスタブ曲線を、次の図１に示す。

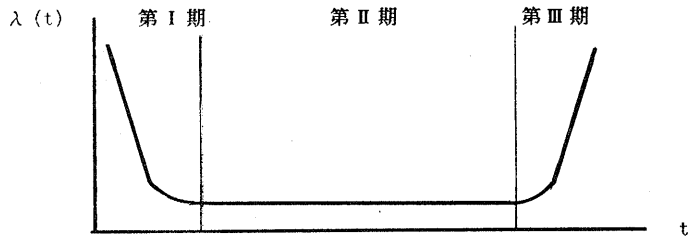

図２．１　バスタブ曲線

ここで、一般的には、縦軸のλ(t)を故障率とすると、ソフトウェア製品の場合には、不具合率、クレーム率として考えても差し支えない。一方、横軸のｔについては、時間（カレンダ時間、CPU 時間など）としての解釈と、時間以外の測度（使用回数など）としての解釈が考えられる。

バスタブ曲線の第Ⅰ期は、ソフトウェア製品のデバッグ段階であり、λ(t)が、低下していく期間である。バスタブ曲線の第Ⅱ期は、ソフトウェア製品の運用段階であり、未検出のバグによる不具合が出現し、事後保全的な修正（リビジョンアップ）を行なう期間である。

特に問題になるのは、第Ⅲ期である。ソフトウェア製品の場合には、物理的な摩耗がない。但し、ユーザの立場からみると、ソフトウェア製品の機能／性能について、ハードウェアの進歩や新しいアルゴリズムの出現あるいは競合製品の利用などによる相対的な劣化が生じる。したがって、第Ⅲ期に入ると、こういった陳腐化によって、ソフトウェア製品の故障率（あるいは不具合率、クレーム率）が、急速に増大してきて、最終的には、ソフトウェア製品の廃棄に至るまで進んでいく。

２．２．ソフトウェア製品のライフサイクルに関する定式化

以上述べたように、ソフトウェア製品のライフサイクルは、ハードウェアのそれと同じように、バスタブ曲線で表わすことができる。次に、この考え方を基礎として、ソフトウェア製品のライフサイクルに関する定式化を検討する。

今までのソフトウェア信頼性モデルは、ソフトウェア製品生産過程のテスト段階までに注目するものが多い。例えば、故障間隔モデル（ JM モデル、SW モデル、Goel and Okumoto Imperfect Debugging モデルなど）、故障数モデル（ Goel-Okumoto NHPP モデルなど）がある。テスト段階が終わって、ソフトウェア製品の運用段階に入ると、ソフトウェア製品の事後保全に相当する修正版発行（リビジョンアップ，revision up ）および予防保全に相当する改版（バージョンアップ，　version up ）が行なわれる。以上挙げた各モデルは、こういった保全を行なってから、当該ソフトウェア製品が、元のものではなく、修正された新しいソフトウェ

ア製品になってしまったと考える。但し、ソフトウェア製品のライフサイクルから見ると、リビジョンアップしたり、バージョンアップしたりするとしても、最終廃棄されるまでのソフトウェア製品は、元のものが変更・改変されたに過ぎないと考えられる。その一方において、リビジョンアップやバージョンアップによって、在来のバグを修正したと同時に、局部的修正および新機能追加による不整合の生起も考えられる。これは、ディグレード（degrade)現象の招来でもある。特に、大規模なソフトウェア製品では、ディグレードの出現は数多くなりがちであるともいえよう。

上記のように考えた場合のソフトウェア製品のライフサイクルにおける故障／不良の出現様相を、図２．２に示す。

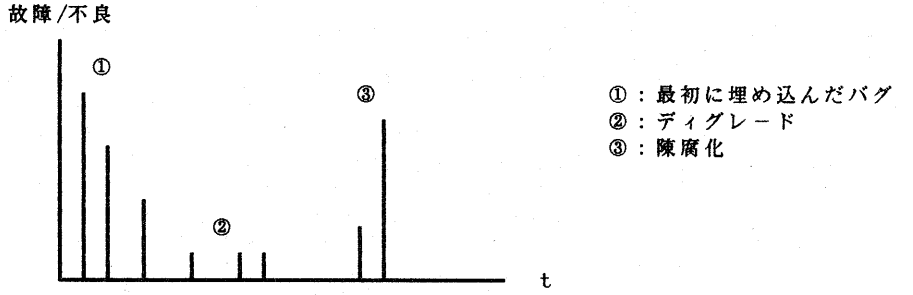

図２．２　ソフトウェア製品のライフサイクルにおける故障／不良

こういったソフトウェア製品のライフサイクルの各段階における故障／不良のパターンを明確に表現するのに、ワイブル分布の適用が有効であると考えられる。

各コンポーネントの故障時間分布が、２母数ワイブル分布に従うとすれば、その信頼度は、

$$R(t)=1-\exp(-t/\eta)^{m} \qquad (1)$$

になる。

但し：m：形状パラメータ

η：尺度パラメータ

ワイブル分布の形状パラメータmの変化によって、バスタブ曲線の第Ⅰ期（m＜０）、第Ⅱ期（m＝０）、第Ⅲ期（m＞０）を表わすことができる。

３．　モデルの設定

３．１．　モデルの設定に関する仮定条件

前述のソフトウェア製品の劣化特性およびシステムとしてのソフトウェア製品構成を考慮して、以下のようにモデルの設定に関する条件を仮定する。

①　ソフトウェア製品は、複数のコンポーネント（機能モジュール）からなるシステムである。

②　各コンポーネントは、間欠的に使用される。

③　各コンポーネントでは、使用されないと、不良が検出されない。

④　各コンポーネントのライフサイクルにおける故障時間の分布は、ワイブル分布に従う。

⑤　発見されたバグは、直ちに修正される。

３．２．間欠的に使用されるコンポーネント

ソフトウェア製品は、一般に、複数のコンポーネント（機能モジュール）から構成していると考えられる。これらのコンポーネントは、要求された機能を満たすように、間欠的に使用される。各コンポーネントの使用確率は、時間ｔの関数とする。

以下においては、ソフトウェア製品が、複数個のコンポーネントから構成しているとし、ｉ番目のコンポーネントが時間ｔまで使用される確率を、$P_i(t)$ $(i=1,2,..,n)$ と表わす。

$P_i(t)$は、ソフトウェア製品のコンポーネント構成や運用の仕組みなどによって、経験的に決められる。一番簡単な場合としては、それぞれのコンポーネントが、一定の使用率で使用される。そのときには、$P_i(t)$は、次のように指数分布に従う。

$$P_i(t)=\exp(-p_{oi}t) \qquad (2)$$

但し：p_{oi}：ｉ番目のコンポーネントの使用率

３．３．モデルの設定

（１）間欠使用によるコンポーネントの信頼度

まず、間欠的に使用される各コンポーネントを考える。あるコンポーネントの故障は、そ

のコンポーネントの使用中にある時点で故障するものと、使用されない時点で不具合が発生し、その後に使用されたために故障となるものとの和として考えられる。

すなわち、間欠使用を考慮する場合のｉ番目のコンポーネントの信頼度$R_i^*(t)$は、

$$R_i^*(t) = 1 - [P_i(t)F_i(t) + \int_0^t \{1 - P_i(\tau)\} * \{P_i(t) - P_i(\tau)\} dF(\tau)] \quad (3)$$

になる。

但し：$F_i(t)$：ｉ番目のコンポーネントの累積故障分布関数

$P_i(t)$：時点ｔまで、ｉ番目のコンポーネントが使用される確率

式（３）の右辺では、第一項が、当該コンポーネントの使用中に故障するものを表わし、第二項が、そのコンポーネントが使用されないときに不具合が発生し、その後に使用されたために故障となるものを表わす。

（２）次に、ソフトウェア製品運用中のバージョンアップの効果を考える。ソフトウェア製品ライフサイクルにおいて、バージョンアップは、何回も行なわれるのが、普通である。但し、バージョンアップによるオーバヘッドの増大、ＯＳやハードウェアなどとの不整合の生起もありうる。したがって、バージョンアップ実施回数の増加と同時に、バージョンアップの効果は、低下していくと考えられる。バージョンアップの実施は、前回のバージョンアップ実施から、今回のバージョンアップ実施直前までの間に、ソフトウェア製品の信頼度が低下した分を補償しようとすることである。なお、ソフトウェアライフサイクルのある時点を越えたら、バージョンアップを実施しても、その効果が期待できなくなると考えられる。

こういったバージョンアップの補償効果を表わすために、バージョンアップ効果関数Ｖ（ｔ）を導入する。

$$V(t) = \begin{cases} a\{1 - N(t)\}^{-b} & t < t_0 \\ 0 & t > t_0 \end{cases} \quad (4)$$

但し：Ｎ（ｔ）：時刻ｔまでのバージョンアップの実施回数

t_0：ユーザ・ニーズ、市場情勢、あるいわシステムとの不整合などに起因して、バージョンアップが効かなくなる時点

$a < 1$，$b > 0$

バージョンアップ効果関数Ｖ（ｔ）の様相を、次の図３．１に示す。

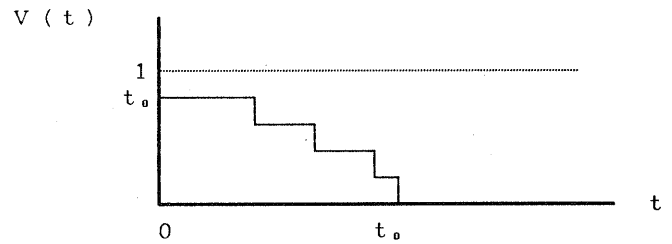

図３．１　バージョンアップ効果関数

なお、バージョンアップ効果を考慮した信頼度は、

$$R(t) = R^*(t) + V(t)\{R(t_{N-1}) - R(t_N)\} \quad (5)$$

とする。

但し：t_{N-1}：前回バージョンアップ実施直後の時間

t_N：今回バージョンアップ実施直前の時間

$R^*(t)$：バージョンアップ効果を考慮しないときの信頼度

４．システムとしてのソフトウェア製品の信頼度

４．１．システムとしてのソフトウェア製品の冗長性

既述のように、ソフトウェア製品は、複数個のコンポーネント（機能モジュール）からなるシステムと考えられる。次に、システムとしてのソフトウェア製品の冗長性について、検討を加える。

一般に、冗長性を持つシステムは、図４．１に示すように分類することができる。

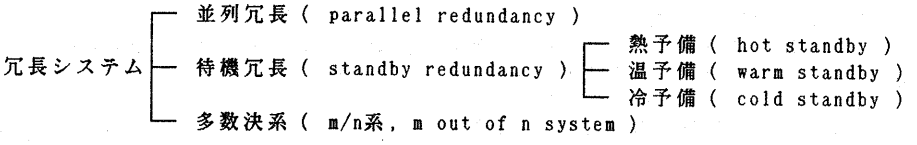

図４．１　冗長システムの分類

（１）並列冗長への対応

判定条件によって、別個の分岐先への実行を行なうことは、" time delay "を含めてのその判定条件の設定のしかたによっては、ある種の並列冗長と考えられる。また、ソフトウェアシステムは、内部記憶を持つ順序論理機械である点に注目すると、ソフトウェア製品の運用中に、ある障害が発生しても、まず内部記憶にエラー状態を起こすことにより、条件判断の結果によって、別の同種経路を辿れば、直ちにシステムの誤動作になることは限らない。これも、一つの並列冗長である。

（２）待機冗長への対応

ソフトウェア製品の使用中において、異常処理などのために、予備状態になっているモジュールを持ってきて、システムの正常運用を維持することが、待機冗長と考えられる。なお、ここでの待機冗長は、一般的に、常に実行できる状態になっている熱予備的なものである。また、分散処理システムの他の部分に待機中の同種モジュールを利用する温予備的なものおよび共通データベースから、新しいモジュールを呼び出して、チェックの上で使用する冷予備的なものも存在しうる。

（３）ｍ／ｎ系への対応

複数個（ｎ個）の条件、あるいは実行結果の中のいくつか（ｍ個）の結果の一致によって、次の処理を行なうことが、ｍ／ｎ系と考えられる。順序論理機械としてのソフトウェアは、ある時点での入力だけではなく、その時点での内部記憶の状態と合わせて、次段階の処理を行なう。そして、次段階の処理と同時に、内部記憶も更新される。それによって、新しいｍ／ｎ系が設定されることになる。

４．２．複数個のコンポーネントからなるシステムについての検討

次に、複数個のコンポーネントから構成されているソフトウェアシステムを考える。そのソフトウェア製品のいくつかのコンポーネントの間には、機能的な関連性が存在しており、外部から見ると一つの機能群になる。機能群内部のコンポーネントは、類似した機能を持ち、冗長的な関係となっているものが少なくない。一方、機能群と機能群との間には、明らかに直列的な関係がある場合が多い。すなわち、システムとしてのソフトウェア製品は、要素並列（ series-parallel ）のように構成されている場合が少なくない。

こういったソフトウェア製品のシステム構成の一例を、図４．２に示す。

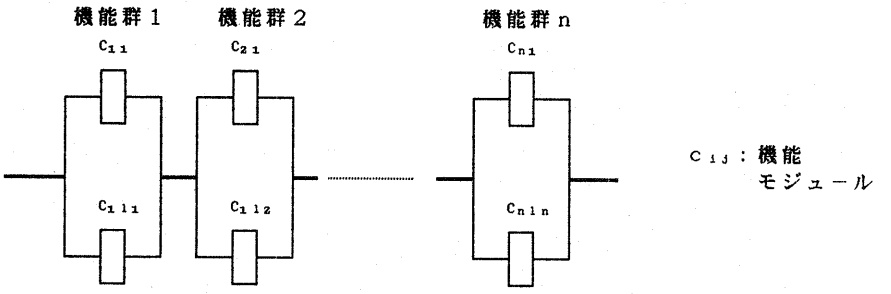

図４．２　機能モジュールからなるソフトウェア製品のシステム構成

図４．２に示したソフトウェアシステムの信頼度は、

$$Rs(t)=\prod_{i=1}^{n}\left[1-\prod_{j=1}^{l_i}\{1-R_{ij}^{*}(t)\}\right]+V(t)\{R(t_{N-1})-R(t_N)\} \quad (6)$$

になる。

但し：$R_{ij}^{*}(t)$：コンポーネントc_{ij}の信頼度

l_i：ｉ番目の機能群内のコンポーネントの数

ｎ：機能群の数

５．数値計算の結果および結果の検討

以上の検討に基づいて、ソフトウェア製品に関する劣化特性を考慮した冗長的モデルについて、数値計算を行ない、その結果を検討した。

５．１．ソフトウェア製品の劣化特性について

以下において、故障時間分布が、２母数ワイブルに従うソフトウェア製品の劣化特性について、数値計算の結果によって検討する。この場合について、各コンポーネントの使用率 $p_0=1$ という条件を設定した事例に対する数値計算を行ない、$y=t/\eta$ と基準化しておき、ソフトウェア製品のコンポーネントの信頼度に対する影響を調べる。

その結果を、図５．１に示す。

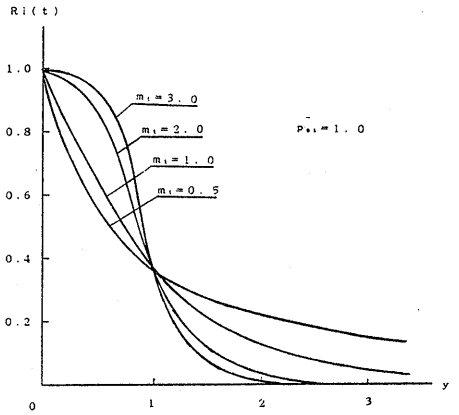

図５．１　形状パラメータｍの影響

形状パラメータｍが増大すると、時間の経過によるコンポーネント信頼度の低下が激しくなる。既述のように、ワイブル分布の場合に、ｍ＜１のとき、故障率減少の段階を表わし、ｍ＝１のとき、故障率一定の段階を表わし、ｍ＞１のとき、故障率増加の段階を表わす。特に、ｍが１よりはるかに大きな場合には、ソフトウェア製品内部の潜在バグによる不良よりも、周囲環境との不整合による不具合およびユーザのクレームの方が、圧倒的に多いと考えられる。図５．１の中で、ｍ＝３の場合には、信頼度が急速に下がっていく。これは、ソフトウェア製品における陳腐化が進んできて、当該コンポーネントが使えなくなる段階に来ていることを表明している。

実際のソフトウェア製品に関する不良データを解析し、ソフトウェア製品の劣化パターンを把握することによって、ｍ＜１の段階では、極力的にバグを検出し、$\lambda(t)$ を低く押える。ｍ＝１の段階では、タイムリーな事後保全によって、この正常運用段階を維持する。

図５．１から分かるように、ｙ＝１の点の左方、すなわち、ｙ＜１のときに、ｍ＞１の場合の信頼度が、ｍ＜１の場合のそれを上回っている。これに対して、ｙ＝１の点の右方、すなわち、ｙ＞１のときに、この関係が逆転して、ｍ＞１の場合の信頼度が下回ってくる。ｙ＝１の時点は、当該ソフトウェア製品の特性寿命に到達した時点であり、この図５．１の条件のもとでは、その時点での信頼度は、０．３７になる。したがって、ソフトウェア製品の特性寿命に到達したか否かについて、特に注目すべきである。ｍ＞１かつソフトウェア製品の特性寿命を越える場合には、ソフトウェア製品の信頼度低下が激しく（この場合は、０．３７以下）、陳腐化が急速に進んでくる。この段階に入ると、事後保全（リビジョンアップ）の単純な実施だけでは、こういう状況に対応できなくなる。したがって、ｍ＞１になる直前で、限界余裕点検（ marginal check ）などによる前駆現象の検知による予防保全を実施して、ｍ＝１の段階をさらに延長する努力を具現するというバージョンアップの実施が必要である。さらに、ｍ＞１の段階では、ソフトウェア製品の陳腐化に対処するリプレース（ replace ）などを実施することが、ソフトウェア製品のライフサイクルにおいて、ユーザの立場からみて、費用有効性（ cost effctiveness ）の立場からの信頼性向上に重要な役割を果たすものと考えられる。

５．２．コンポーネントの使用率について

使用率 p_0 が、コンポーネントの信頼度に対する影響を、図５．２に示す。

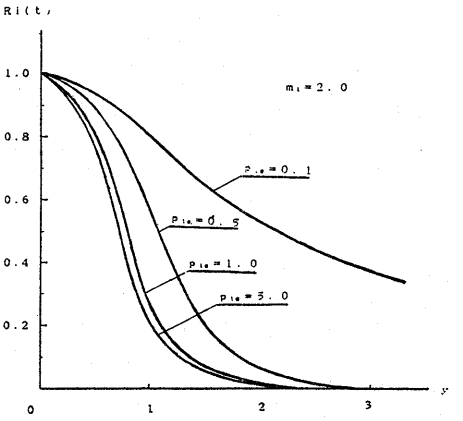

図５．２　使用率p_0の影響

図５．２から見ると、p_0の増大に従って、コンポーネントの信頼度は、急速に低下していく。図５．１の結果と比較すると、コンポーネントの使用率p_0が、形状パラメータｍと同じように、コンポーネントの信頼度に、強い影響を与える。

使用率の高いコンポーネントは、潜在するバグによる不具合も出やすい。それと同時に、バグに対する修正が多く行なわれると考えられる。こういった局部的修正によって、ディグレード現象の生起を招来することも、無視できない。

したがって、ソフトウェア製品内部の各コンポーネントの使用状況を把握して、その使用確率の信頼度に対する影響を吟味した上で、ソフトウェア製品の設計・製造、運用・保全の諸段階において、ソフトウェア製品の信頼性向上の諸対策を検討することが、今後の現実的な課題となっている。

５．３．バージョンアップ効果の検討

式（５）に示すように、バージョンアップ効果関数Ｖ（ｔ）については、時刻ｔまでのバージョンアップ実施回数Ｎ（ｔ）およびパラメータａ，ｂによって決められる。ここで、バージョンアップは、一定の時間間隔で実施されると想定し、Ｎ（ｔ）＝［ｔ］と決める。パラメータａが、バージョンアップ効果関数全体の影響の強さを測るものであるに対して、パラメータｂは、一回のバージョンアップ実施によるバージョンアップ効果への影響の強さを測るものである。したがって、パラメータｂが、バージョンアップ効果を検討するときに、もっと重要な意味を持つと考えられる。次の図５．３で、パラメータｂにいくつかの値を与えて、その影響を調べる。

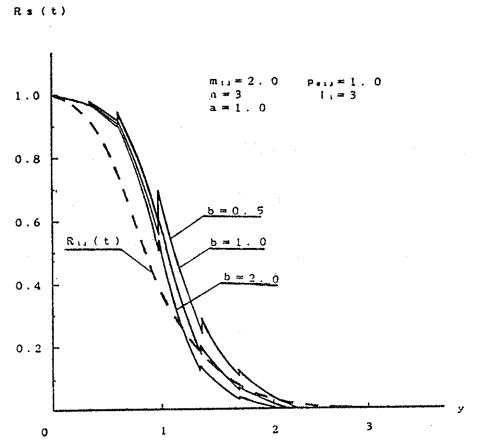

図５．３　バージョンアップ効果

図５．３から分かるように、ｂ＝０．５の場合には、新しいバージョンアップの実施によって、前回のバージョンアップ時点からの信頼度低下が、かなり補償される。ｂ＝２になると、バージョンアップの補償効果が、あまり見えなくなる。これらの結果から、ｂは、実施されるバージョンアップの「質」を説明するパラメータであると見られる。徹底的かつディグレードを生起させないバージョンアップほど、そのｂの値は、小さいはずである。

図５．３から、バージョンアップ実施回数の増加とともに、その効果が、ｂ値と無関係的に低下していく傾向が見られる。これは、完璧なバージョンアップを実施しても、それに伴うオーバヘッドの増大および周囲のハードウェアやソフトウェアとの適合性の低下が無視できないことを、意味していると考えられる。

６．まとめ

本報告では、ソフトウェア製品のライフサイクルにおける劣化特性を考慮し、当初に作り込まれたバグによる影響だけではなく、ソフトウェア製品の運用段階における保全（リビジョンアップおよびバージョンアップ）、さらにソフトウェア製品の陳腐化も検討した。ソフトウェア製品のライフサイクル全般にわたる劣化パターンを吟味し、ソフトウェア製品の故障時間分布に対応して、ワイブル分布を導入した定式化を図った。

この場合には、ソフトウェア製品は、本質的に、複数のコンポーネントから構成しているシステムとして考えられる点に注目する。各コンポーネントが、間欠的に使用され、それぞれの使用確率は、運用時間の関数とする。

数値計算の結果から明確に把握されるように、コンポーネントの使用率が、その信頼度に強い影響を与える。ソフトウェア製品作成の生産性および品質／信頼性向上のために、構造化は、重要な役割を果たしている。構造化の原則に従って、ソフトウェア製品の設計段階から、コンポーネントの使用率が、その信頼度に対する影響を考慮して、コンポーネントおよびシステムの構成を設計することにより、ソフトウェア製品の品質／信頼性の向上に繋がる。

一方、本研究では、使用率が一定の場合だけしか検討しなかった。なお、今後、いろいろな角度から、実際なデータを利用して、より具体的な使用率を検討し、多角的吟味を加えながら、層別的に研究を進めていくつもりである。

ソフトウェア製品の予防保全に相当するバージョンアップの実施によって、ソフトウェア製品に対するクレームの増大を補償する役割を果たす。適切なバージョンアップにより、より少ない工数／コストで新しい機能の追加、性能の向上を実現できる。一方、バージョンアップの実施回数の増加と同時に、システムとしての不整合を生じがちであり、バージョンアップによっても、ソフトウェア製品の寿命を制限なしに延長することは期待できない。

こういったバージョンアップの効果を測るために、バージョンアップ効果関数を導入して、ごくおおまかではあるが、バージョンアップ効果の定量的な検討を行なった。今後、ソフトウェア製品保全の仕組みを一層深く把握し、ソフトウェア製品のライフサイクルにおける保全に関する定式化を行なうことも、早急に課題としたい。

参考文献。

［１］菅野：「信頼性工学」，コロナ社，（1983）。

［２］菅野，前田：”ソフトウェアの信頼度に関する定式化の試み　”，情報処理学会第２１回全国大会講演論文集，（1980）。

［３］菅野，吉岡：”間欠的使用部分を含むシステムの信頼度　”，日科技連第９回信頼性保全性シンポジウム発表報文集，（1979）。

［４］A. L. GOEL: "Software Reliability Models ", IEEE Trans. software Eng., Vol.SE-11, No.12, (1985).

［５］N. F. Schneidewind: "The State of Software Maintenance", IEEE. Trans. software Eng., Vol. SE-13, No. 3, (1987).