数理解析研究所講究録
第1636巻 2009年 267-274

運用期間中の不確実性を考慮したソフトウェア最適リリース問題に関する研究

鳥取環境大学・環境情報学部　豊田　寿行（Toshiyuki Toyota）*
鳥取大学大学院・工学研究科　山田　茂（Shigeru Yamada）**
*Faculty of Environmental and Information Studies,
Tottori University of Environmental Studies
**Graduate School of Engineering, Tottori University

1 はじめに

今日，コンピュータ・システムは私たちの日々の生活に欠かすことのできないものとなっているのは周知の事実である．つまり，コンピュータ・システムの必要性および重要性は常に増してきており，その応用分野は多岐に渡っている．それに伴って，コンピュータ・システムを構成するソフトウェアに対する大規模化，多様化および複雑化の要求は強まる一方である．そのため，高信頼性を具備したソフトウェアの効率的に開発することが開発者に対しての課題となっている．つまり，ソフトウェアの開発において，その開発を管理する立場となるプロジェクト・マネージャおよびプロジェクト・リーダーはソフトウェアの開発工程における定量的な管理技術の必要としているのは言うまでもない．プロジェクト・マネージャおよびプロジェクト・リーダーは開発するソフトウェアの品質，その開発コストおよびいつ出荷できるかという観点でその開発工程を日々管理して，そのプロジェクト自体の管理を行うのである．

ソフトウェアのライフサイクルを図1に示す．ソフトウェアの開発はユーザの要求を文書化する要求仕様定義をはじめに行い，具備すべき機能を明確にして設計を行う．そして，実際にコーディングして，テスト工程に移行して要求仕様を満たしているかどうかを確認する．テスト工程終了後，ユーザに対してソフトウェアを出荷して，運用・保守を行い，その後廃棄される．この一連の流れをソフトウェアのライフサイクルと呼ぶ．

ソフトウェアの開発工程において，テスト工程におけるテストコストは全開発コストの半分以上を占めるといわれている[1]．テスト期間が長期化すれば，ソフトウェアの品質はソフトウェア内に潜在するフォールトをより多く発見および修正できるために向上する．しかし，テストコストの増大および出荷時間の遅延が生じる可能性がある．逆に，テスト期間が短ければ，十分にフォールトを発見および修正されること無く，低い品質のままユーザに出荷され，出荷後の運用工程における保守コストの増大が考えられる．したがって，ソフトウェア開発プロジェクトにおいて，プロジェクト・マネージャおよびプロジェクト・リーダーはいつテスト工程を終了して出荷するかという意思決定は，いわゆるKKD（勘，経験，度胸）に依存してきたのは言うまでもない．そこで，ソフトウェアの品質および開発コストの観点から定量的にソフトウェアのテスト工程の打ち切り時刻を見積もり，ユーザへの出荷時刻を算出することはプロジェクト・マネージャおよびプロジェクト・リーダーにとって有益である．

このような意思決定問題をソフトウェアの最適リリース問題といい，テスト工程の打ち切り時刻を最適リリース時刻と呼ぶ．近年，ソフトウェアの最適リリース問題はたくさんの研究者によって議論が行われている[2]-[7]．

本研究では，運用期間中の不確実性を考慮した最適リリース問題を議論する．従来の最適リリース問題においては出荷されたソフトウェアは運用工程において単一ユーザによる使用を仮定している．しかしながら，実社会でのソフトウェアの使用状況を考えた場合，同じソフトウェアを複数のユーザで同時に使用する状況が大多数であるといって過言ではない．したがって，運用期間における複数ユーザの使用を考慮してモデル化して議論を行う点が本研究の新規性であるといえる．具体的には，運用工程において各ユーザのフォールト発見率は異なると仮定する．例えば，テスト工程におけるフォールト発見環境に比べて，

複数のユーザで使用する運用工程におけるフォールト発見環境は厳しいと考える．したがって，運用工程におけるユーザのフォールト発見環境に対して，厳しさ係数という概念を導入して議論する．

第2章では，ソフトウェアフォールトの発見事象を記述するために非同次ポアソン過程(nonhomogeneous Poisson process) に基づくソフトウェア信頼度成長モデルのひとつである指数形ソフトウェア信頼度成長モデルについて述べる．次に，ソフトウェア信頼度成長モデルを用いて，ソフトウェア保守コストモデルを定式化する．さらに，定式化したソフトウェア保守コストモデルに基づき保守コストを最小化するいくつかの最適リリース方策について議論する．そして，最適リリース方策に対する数値例を示して本研究で提案する最適リリース問題についての議論を深める．

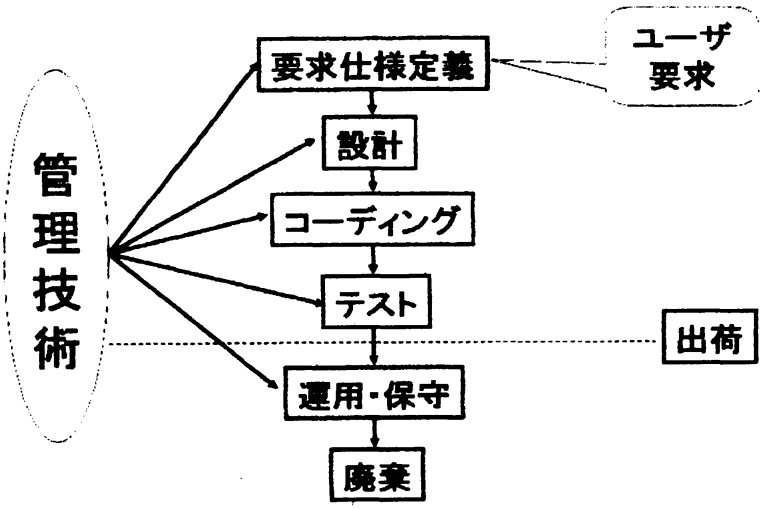

図1．ソフトウェアのライフサイクル

2．ソフトウェア信頼度成長モデル

ソフトウェアの開発においてテスト工程および運用工程では，ソフトウェア故障発生事象およびソフトウェアフォールト発見事象に対して，確率・統計論を適用して議論することは自然なことであると考えられる．これらの事象は様々な数理モデルによって定式化されており，ソフトウェア信頼性モデルと呼ばれている．特に，ソフトウェアの開発工程の特性から考えて，テスト工程に実施される様々なテストによってソフトウェア内のフォールトは発見および修正される．ここで，テスト工程において新しいフォールトはソフトウェア内に発生しないと仮定した場合，ソフトウェアテストを行うことによってフォールトは除去され，ソフトウェア内に潜在する総フォールト数は減少する．つまり，ソフトウェア故障の発生確率は減少する，すなわち，ソフトウェアの信頼度は成長するといえる (図2参照)．

本研究では，ソフトウェアフォールト発見事象に対して，非同次ポアソン過程 (a nonhomogeneous Poisson process)に基づく，ソフトウェア信頼度成長モデルである指数形ソフトウェア信頼度成長モデルを適用する．指数形信頼度成長モデルは次の式(1)および(2)で表される．

$$m(t) = a(1-e^{-bt}) \quad (a>0, b>0), \tag{1}$$

$$h_m(t) = abe^{-bt}, \tag{2}$$

ここで，$m(t)$ は平均値関数と呼ばれ，テスト時間 $(0,t]$ における期待累積フォールト発見数を表す．また，$h_m(t)$ は強度関数と呼ばれ，フォールト発見率を表す．そして，パラメータ a および b はそれぞれテスト開始前にソフトウェア内に潜在する総期待フォールト数およびフォールト1個当たりのフォールト

発見率を表す.

指数形ソフトウェア信頼度成長モデルはA. L. Goel and K. Okumoto[8]により提案された有名なソフトウェア信頼度成長モデルのひとつである. この指数形ソフトウェア信頼度成長モデルは大規模ソフトウェアシステムに対するソフトウェア信頼度を評価する場合において, 特にテスト時間をCPU時間で測定する場合のソフトウェア故障発生事象にしばしば適用される[9]. そして, 様々な研究者がたくさんのソフトウェア信頼度成長モデルを提案している. 例えば, S字形ソフトウェア信頼度成長モデル, 修正指数形ソフトウェア信頼度成長モデルおよび遅延S字形ソフトウェア信頼度成長モデルなどが挙げられる[10].

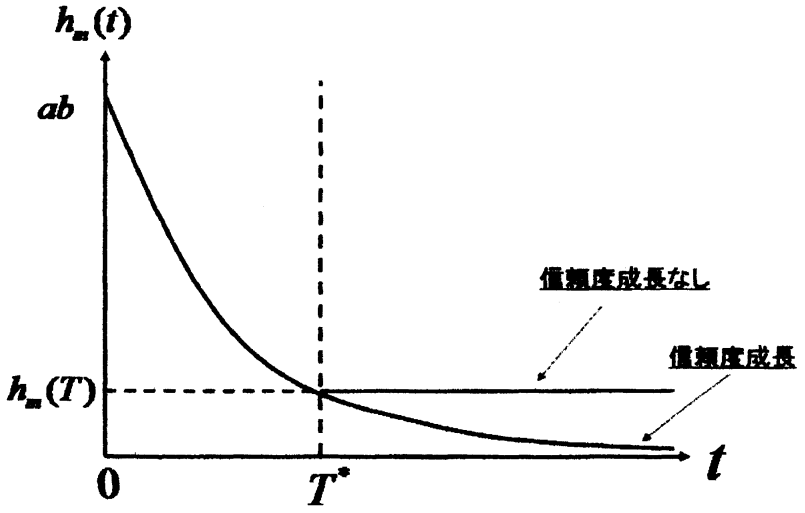

図2. ソフトウェアの信頼度成長

3. 最適リリース問題

従来のソフトウェアの最適リリース問題を議論する際に取り扱われてきたのはソフトウェアのライフサイクル（図1参照）において主にテスト工程についてである. 具体的には, テスト工程と運用工程との間のソフトウェアフォールトの発見環境の差異について焦点をあて, 最適リリース問題を定式化して議論したものは皆無といって過言ではない. 本研究では, テスト工程に加えて, 運用工程のソフトウェアフォールトの発見環境についても議論可能なフレームワークについてのひとつの提案を行う. 特に, 先行研究においてはテスト工程に注目するがあまり, 運用工程におけるソフトウェアのユーザをひとりと仮定して定式化されることがほとんどであった. そこで, 定式化にあたり, 運用工程において複数ユーザの使用を考慮して, より実環境に即した定式化を行う. その定式化にあたり, 運用工程における複数のユーザごとのフォールト発見事象は異なることを仮定する. この仮定によって, 複数ユーザの使用環境を表現することが可能であると考える. 当然のことながら, 複数のユーザを考慮する手段は他にも考えることは可能であるが, より単純でかつ素朴な定式化を念頭にこのような仮定を行うこととした.

前述のように, 運用工程において複数のユーザの使用環境を考慮する場合にユーザの間のフォールト発見事象の差異の存在を仮定した. その仮定により, 運用工程における不確実性は単一ユーザの場合と比べて増加すると考える. その不確実性による最適リリース時刻の挙動について議論することによって有益な知見を得ることを本研究におけるモデル化によって取り組む課題である.

そこで, 運用工程における複数のユーザの間のフォールト発見事象の差異を考慮するために厳しさ係数の概念を導入する. 例えば, 複数のユーザがソフトウェアを使用する運用工程におけるフォールト発見事象はテスト工程と比べて, より厳しいと考える. したがって, i番目のユーザの厳しさ係数を$\gamma_i(\gamma_i > 0)$とすると, $\gamma_i > 1$ ならば, i番目のユーザの運用工程におけるフォールト発見率はテスト工程と比べて, 厳しいことを意味する. このような厳しさ係数の概念を用いることによって, 前章で述べた, 式(1)および(2)の指数形ソフ

トウェア信頼度成長モデルは次のように定式化できる.

$$m_i(t) = a(1-e^{-\gamma_i bt}) \quad (a>0, b>0, \gamma_i>0), \tag{3}$$

$$h_{m_i}(t) = a\gamma_i b e^{-\gamma_i bt}, \tag{4}$$

ここで, パラメータaおよびbは前章と同様に, それぞれテスト開始前にソフトウェア内に潜在する総期待フォールト数およびフォールト 1 個当たりのフォールト発見率である. さらに, $m_i(t)$および$h_{m_i}(t)$はそれぞれi番目のユーザの期待累積フォールト発見数およびフォールト発見率を表す.

3.1 ソフトウェア保守コストモデル

最初に, ソフトウェア保守コストモデルの定式化に際して, 各パラメータは次のとおりである.

c_0: 最低限必要なテストコスト

c_t: 単位時間当たりのテストコスト

c_u: 運用工程中のフォールト 1 個の保守コスト

T: ソフトウェアのリリース時刻, すなわち全テスト時間

T^*: 最適ソフトウェアリリース時刻

T_{lc_i}: i番目のユーザのライフサイクルの長さ (テスト工程開始後から廃棄までを指し, 図 1 の一般的なソフトウェアのライフサイクルとは異なる)

N: ユーザ数

次に, ソフトウェア保守コストモデルを以下のように定義する.

$$UC(T) \equiv c_0 + c_t t + C_{op}(T), \tag{5}$$

ここで, $C_{op}(T)$は運用期間中の保守コストを表す. 本研究では, $C_{op}(T)$に関して, 次の 2 つのケースを議論する.

(ケースA): 運用工程において, ソフトウェアの信頼度は成長はしない. したがって, $C_{op}(T)$ は次のように表せる:

$$C_{op}(T) = c_u \sum_{i=1}^{N} h_{m_i}(T) T_{lc_i}, \tag{6}$$

(ケースB): 運用工程においてもソフトウェアの信頼度は成長する. したがって, $C_{op}(T)$ は次のように表せる:

$$C_{op}(T) = c_u \sum_{i=1}^{N} \{m_i(T_{lc_i}+T) - m_i(T)\}. \tag{7}$$

ここで, ソフトウェアのライフサイクルの長さはテスト工程開始から廃棄までを意味するものとする (図 1 の一般的なソフトウェアのライフサイクルとは異なる). 式(5)のソフトウェア保守コスト$UC(T)$を最小化することによって, 最適リリース時刻を導出する.

3.2 最適リリース方策

本節では，式(3)および(4)を用いて，前述のソフトウェア保守コストモデルを分析する．そして，(ケースA)および(ケースB)に対して，それぞれ最適リリース方策を導出する．

最適リリース方策の導出のために，式(3) および(4)を用いて，式(5)のソフトウェア保守コスト$UC(T)$を最小化する最適リリース時刻を求める．

最初に，(ケースA)の最適リリース方策の導出について議論する．式(6) を式(5)に代入すると，次のようになる．

$$UC(T) = c_0 + c_t + c_u \sum_{i=1}^{N} h_{m_i}(T) T_{lc_i}, \tag{8}$$

ここで，

$$UC(0) = c_u ab \sum_{i=0}^{N} \gamma_i T_{lc_i}, \qquad UC(\infty) = \infty.$$

式(8) をT に関して，微分して，イコールゼロとおくと，

$$\frac{dUC(T)}{dT} = c_t - c_u ab^2 \sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} = 0, \tag{9}$$

$$\sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} = \frac{c_t}{c_u ab^2}. \tag{10}$$

さらに，

$$\frac{d^2 UC(T)}{dT^2} = c_u ab^3 \sum_{i=1}^{N} \gamma_i^3 e^{-\gamma_i bT} T_{lc_i} > 0. \tag{11}$$

したがって，$UC(T)$ はT に関して，凸関数である．$\sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} > \frac{c_t}{c_u ab^2}$ ならば，

$dUC(T)/dT = 0$ は式(10)を満たす，唯一の有限解T_1 をもつ．一方，$\sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} \le \frac{c_t}{c_u ab^2}$ ならば，

式 (10) は正の解をもたない，すなわち$T^* = 0$ である．

以上のことから最適リリース方策は，次のように導出できる．

[最適リリース方策 1]

(a) $\sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} > \frac{c_t}{c_u ab^2}$ のとき，最適リリース時刻$T^* = T_1$.

(b) $\sum_{i=1}^{N} \gamma_i^2 e^{-\gamma_i bT} T_{lc_i} \le \frac{c_t}{c_u ab^2}$ のとき，最適リリース時刻$T^* = 0$.

次に，(ケースB)の最適リリース方策の導出について議論する．前述の**[最適リリース方策 1]**の導出の同様にして，次の**[最適リリース方策 2]**を得る．

[最適リリース方策2]

(a) $\sum_{i=1}^{N}\gamma_i m_i(T_{lc_i}) > \frac{c_t}{c_u b}$ のとき，最適リリース時刻$T^* = T_2$.

(b) $\sum_{i=1}^{N}\gamma_i m_i(T_{lc_i}) \le \frac{c_t}{c_u b}$ のとき，最適リリース時刻$T^* = 0$.

ここで，T_2は$\sum_{i=1}^{N}\gamma_i m_i(T_{lc_i}) > \frac{c_t}{c_u b}$のとき，$dUC(T)/dT = 0$を満たす唯一の有限解である.

4．数値例

本章では，前節で導出した最適リリース方策の有効性について議論するために数値例を示す．ここで，運用工程における，ユーザ数をN=1, 2, 3 とする．さらに，式(3) および(4) におけるパラメータをα=1000, b=0.05 とする．また，$c_0 = 1000$ および$T_{lc_1} = T_{lc_2} = T_{lc_3} = 100$ とおく．

最初に，テスト工程における単位時間当たりの保守コストc_tおよび運用工程におけるフォールト1個当たりの保守コストc_uについての振る舞いを議論する．つまり，**[最適リリース方策1]** および**[最適リリース方策2]**を**(ケースA)** および**(ケースB)**に対して，それぞれ数値計算を行った．この場合，c_uが大きくなるにつれて，最適リリース時刻は大きくなる．しかし，c_tが大きくなるにつれて，最適リリース時刻は小さくなる．一方，**(ケースA)** および**(ケースB)**に関してもそれぞれ同じ傾向の結果を得た．さらに，**(ケースA)** と**(ケースB)**を比較した場合，**(ケースA)** の最適リリース時刻は**(ケースB)**の最適リリース時刻は常に大きい．

次に，**[最適リリース方策1]**において，厳しさ係数γ_iおよびc_tについての数値計算の結果が表1である．

ここで，c_u=5.0 とする．さらに，**(ケースA)**および**(ケースB)**に対する，ユーザ数Nの場合の最適リリース時刻を$T^*_{k(\gamma_1,\gamma_2,\cdots\gamma_N)}(k = A, B)$とする．表1より，$T^*_{A(2.0)} < T^*_{A(1.0)} < T^*_{A(0.5)}$である．したがって，厳しさ係数が小さくなるにつれて，すなわち，運用工程におけるフォールト発見率が減少するにつれて，最適リリース時刻$T^*_{A(\gamma_1)}$は大きくなる．また，(c_t / c_u)が大きいとき，すなわち，運用工程における保守コストに対してテスト工程におけるが大きいとき，最適リリース時刻は$T^*_{A(2.0)} < T^*_{A(0.5)} < T^*_{A(1.0)}$である．しかし，一般的に考えて，運用工程おける保守コストとテスト工程における保守コストは運用工程のほうが大きいと考えるのが自然である．つまり，(c_t / c_u)が小さいときは，最適リリース時間は$T^*_{A(2.0)} < T^*_{A(1.0)} < T^*_{A(0.5)}$となることがわかる．さらに，ユーザ数のみを比較した場合，次のような最適リリース時刻の関係が得られた．$T^*_{A(0.5)} < T^*_{A(0.5,0.5)} < T^*_{A(0.5,0.5,0.5)}$，$T^*_{A(1.0)} < T^*_{A(1.0,1.0)} < T^*_{A(1.0,1.0,1.0)}$，そして$T^*_{A(2.0)} < T^*_{A(2.0,2.0)} < T^*_{A(2.0,2.0,2.0)}$．このことから,ユーザ数が増加すると，最適リリース時刻は大きくなる

ことがわかる．さらに，$T^*_{A(0.5,0.5)}$ と $T^*_{A(1.0,1.0)}$ を比較すると，$T^*_{A(0.5,0.5)}$ は $T^*_{A(1.0,1.0)}$ の 2 倍の値とはなっていない．

表1．[最適リリース方策1]に対する最適リリース時刻

$(a=1000.0, b=0.05, T_{lc_i}=100, c_0=1000, c_u=5.0)$

		c_t				
		1.0	5.0	10.0	50.0	100.0
γ_1	0.5	229.8	165.4	137.7	73.3	45.6
	1.0	142.6	110.4	96.6	64.4	50.5
	2.0	85.2	69.1	62.1	46.1	39.1
(γ_1,γ_2)	(0.5, 0.5)	257.5	193.1	165.4	101.0	73.3
	(1.0, 1.0)	156.5	124.3	110.4	78.2	64.4
	(2.0, 2.0)	92.1	76.0	69.1	53.0	46.1
	(0.5, 1.0)	230.3	167.8	142.0	88.0	67.7
	(0.5, 2.0)	229.8	165.4	137.7	75.5	54.9
	(1.0, 2.0)	142.7	110.7	97.2	67.0	55.1
$(\gamma_1,\gamma_2,\gamma_3)$	(0.5, 0.5, 0.5)	273.7	209.4	181.6	117.2	89.5
	(1.0, 1.0, 1.0)	164.6	132.4	118.5	86.3	72.5
	(2.0, 2.0, 2.0)	96.2	80.1	73.1	57.0	50.1
	(0.5, 1.0, 1.0)	230.8	169.8	145.4	95.4	76.7
	(0.5, 2.0, 2.0)	229.8	165.4	137.7	77.1	58.8
	(0.5, 0.5, 1.0)	257.6	193.8	165.6	106.3	82.4
	(0.5, 0.5, 2.0)	257.5	193.1	165.4	101.2	74.5
	(1.0, 1.0, 2.0)	156.5	124.4	110.6	79.0	65.8
	(1.0, 2.0, 2.0)	142.7	110.0	97.7	68.9	57.9
	(0.5, 1.0, 2.0)	230.3	167.8	142.0	88.4	69.1

5．おわりに

本研究では，ソフトウェアの開発工程に着目して，従来の研究ではテスト工程を中心に議論されてきたソフトウェアの最適リリース問題について，特に，運用工程における複数のユーザがひとつのソフトウェアを使用するというより実社会に即した定式化による議論に取り組んだ．その定式化に際して，ユーザ間の使用環境の差異をフォールト発見事象の差異と仮定して，厳しさ係数の概念を導入して議論した．そして，ソフトウェア保守コストモデルの定式化して，ソフトウェアの特性である，運用工程中の信頼度の成長の有無により2つのケースを提案した．さらに，それぞれのケースに対して，最適リリース方策を導出して，数値例を示した．

最後に，本研究で得られた最適リリース時刻 T^* の特性についてまとめる．

(a) 運用工程におけるフォールト1個当たりの保守コストが大きくなるにつれて，T^* は大きくなる

(b) 単位時間当たりのテストコストが大きくなるにつれて，T^* は小さくなる

(c) 厳しさ係数が大きくなるにつれて，T^* は小さくなる

(d) 厳しさ係数が同じときユーザ数が大きくなるにつれて，T^* は大きくなる

本研究で議論した最適リリース問題はソフトウェア開発者にとって，従来，定量的な尺度が必要であったにもかかわらず，十分に確立されていない分野である．したがって，この結果はソフトウェアのテスト工程をいつ終了するかを決定するための定量的な尺度となりうる知見を得ることができた．しかしながら，実際のソフトウェアの開発工程は様々な複雑な要因がたくさんあり，全てを網羅したモデルを作ることは不可能であるかもしれない．けれども，特徴的な要因に対してモデル化を行うことにより分析が可能となり，効率的なソフトウェア開発に寄与できると確信している．したがって，今後の課題としてソフトウェアの開発に大きく影響を及ぼす要因を選択して，モデル化を試みることによって定量的な尺度および基準等を導出することが挙げられる．

参考文献

[1] Xie, M. and Hong, G. Y., "On Optimum Software Release Time Based on Unbound NHPP Models", Proc.3rd. ISSAT Int. Conf. Reliability and Quality in Design, pp. 102-105, Anaheim, U. S. A. 1997.

[2] Cho, B.C. and Park, K. S., "An optimal time for software testing under the user's requirement of failure-free demonstration before release", IEICE Trans., Fundamentals, Vol. E77-A, No. 3, pp. 563-570, 1994.

[3] Foreman, E. H. and Singpurwalla, N. D., "Optimal time intervals for testing-hypotheses on computer software errors", IEEE Trans., Reliability, Vol. R-28, No. 3, pp. 250-253, 1979.

[4] Koch, H. S. and Kubat, P., "Optimal release time for computer software", IEEE Trans. , Software Engineering, Vol. SE-9, No. 3, pp. 323-327, 1983.

[5] Kimura, M.. and Yamada, S., " Optimal software policies with random life-cycle and delivery delay", Proc. 2nd ISSAT Int. Conf., Reliability and Quality in Design, pp. 215-219, Orland, U. S. A., 1995.

[6] Toyota, T., Kimura, M., Yamada, S., "Optimal software release problems with warranty period and reliability requirement", Proc. 3rd ISSAT Int. Conf., Reliability and Quality in Design, pp. 215-219, Anaheim, U. S. A., 1997.

[7] Kimura, M., Toyota, T., Yamada, S., "Economic analysis of software release problems with warranty cost and reliability requirement", Reliability Engineering & System Safety, Vol. 66, No. 1, pp. 49-55, 1999.

[8] Goel, A. L. and Okumoto, K., "Time-dependent error-detection rate model for software reliability and other performance measures", IEEE Trans., Reliability, Vol. R28, No. 3, pp. 206-211, 1979.

[9] Musa, J. D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, prediction, Application*, McGraw-Hill, New York, 1987.

[10] 山田茂，大寺浩志，ソフトウェアの信頼性～理論と実践的応用～，ソフト・リサーチ・センター，東京，1990.