로지스틱 테스트 노력함수를 이용한 소프트웨어의 최적인도시기 결정에 관한 연구

최 규 식* · 김 용 경**

A Study on the Optimal Release Time Decision of a Developed Software by using Logistic Testing Effort Function

Gyu Shik Che* · Yong Kyung Kim**

Abstract

This paper proposes a software-reliability growth model incoporating the amount of testing effort expended during the software testing phase after developing it. The time-dependent behavior of testing effort expenditures is described by a Logistic curve. Assuming that the error detection rate to the amount of testing effort spent during the testing phase is proportional to the current error content, a software-reliability growth model is formulated by a nonhomogeneous Poisson process. Using this model the method of data analysis for software reliability measurement is developed. After defining a software reliability, This paper discusses the relations between testing time and reliability and between duration following failure fixing and reliability are studied. SRGM in several literatures has used the exponential curve, Railleigh curve or Weibull curve as an amount of testing effort during software testing phase. However, it might not be appropriate to represent the consumption curve for testing effort by one of already proposed curves in some software development environments. Therefore, this paper shows that a logistic testing-effort function can be adequately expressed as a software development/testing effort curve and that it gives a good predictive capability based on real failure data.

Keywords : Mean-Value-Function, NHPP, SRGM, Logistic Testing Efforts Function, Target Reliability, Optimum Release Time

논문접수일 : 2004년 1월 14일 논문게재확정일 : 2005년 3월 10일

※ 이 논문은 2005학년도 건양대학교 학술연구비 지원에 의하여 이루어진 것임.

* 교신저자, 건양대학교 정보전자통신공학부, (320-711)충남 논산시 내동 26 건양대학교,
Tel : 041-730-5283, e-mail : che@konyang.ac.kr

** 건양대학교 경영정보학과

1. 서 론

소프트웨어 개발 과정에서 품질 좋고 신뢰성 있는 소프트웨어를 효과적이고 효용성 있게 개발하기 위하여 소프트웨어의 신뢰도 측정 및 관리가 중요하다. 제품의 신뢰도를 특성화 하기 위해서는 정량적인 측정과 관리가 필수적이다. 특히, 소프트웨어 결함데이타 분석에 근거하여 소프트웨어 테스트 단계에서 소프트웨어의 신뢰도를 추정하는 것이 매우 중요하다. 그동안 테스트 단계동안 소프트웨어 결함 검출현상을 설명하기 위한 여러 가지 소프트웨어 신뢰도 모델이 개발되었다. 소프트웨어 테스트에 의해서 발견되는 누적결함(또는 소프트웨어 고장간의 시간간격)과 소프트웨어 테스트 시간간격 사이의 관계를 짓는 모델들을 소프트웨어 신뢰도 성장모델(SRGM ; software reliability growth model)[1]이라 한다. 이러한 모델들을 이용하여 평균 초기결함의 수, 평균 고장간 시간 간격, 임의의 테스트 시간에 소프트웨어 내의 평균 잔여결함수, 소프트웨어 신뢰도 함수와 같은 소프트웨어 신뢰도 척도를 추정할 수 있다.

소프트웨어 개발에는 많은 개발자원들이 소요된다. 소프트웨어 테스트 단계 기간 동안에는 소프트웨어의 신뢰도가 내재 결함을 검출 및 수정하는데 소요되는 개발자원의 양에 크게 의존한다. Musa 등[2]은 기존 소프트웨어 신뢰도 성장모델을 분류하는 알고리즘을 개발하였다. Yamada 등[3]은 역일 테스트 시간, 테스트 노력량, 테스트 노력에 의해서 검출되는 소프트웨어 결함의 수 사이의 관계를 명시적으로 설명할 수 있는 간단하고도 새로운 모델을 제시하였다. 테스트 노력은 테스트 단계에서 소요되는 인력, CPU 시간, 실행테스트 케이스 등등에 의해서 측정된다.

소프트웨어 신뢰도는 고객의 입장에서 본 소프트웨어의 품질 관점이다. 이는 단순한 프로그램 설계라기보다는 실제적인 운전과 관련이 있다. 그러므로 정적이라기보다는 동적이다. 소프트웨어(신뢰도) 엔지니어의 목표와 목적은 설계된 프로그램이 고객이 의도한 대로 잘 동작하도록 동작 확률을 높이는 것이다. 그러므로 소프트웨어 시스템의 신뢰도를 측정하고 계산하는 것이 매우 중요하다. 이들은 개발기간 동안 모든 테스트 자원을 계획하고 제어하는데 쓰일 수 있으어서 우리에게 소프트웨어의 정확성을 확인시켜 주는 것이다.

소프트웨어 신뢰도를 측정하기 위한 공통적인 접근법은 소프트웨어 고장으로부터 구한 가용한 데이터로부터 산출된 파라미터를 가진 분석모델을 이용하는 것이다. 소프트웨어 신뢰도 공학에서의 연구 활동은 20~30여년간 역동적으로 아루어져 왔으며, 그간 많은 SRGM이 제안 되었다. SRGM은 소프트웨어 신뢰도를 산출하는데 성공적이며, 소프트웨어 시스템에 잔여하고 있는 잔여결함의 수를 산출하는데도 성공적이라는 것이 증명되어 왔다. 이들은 소프트웨어의 개발 상태와 소프트웨어 신뢰도 공학 기법을 정량적으로 산출하는데도 쓰일 수 있다.

테스트 단계는 매우 중요하며 소프트웨어 개발의 핵심 부분이다. 그동안 많은 연구가들의 연구에서는 테스트 단계의 테스트 자원의 소모율은 일정하다고 가정하거나 또는 그러한 테스트 노력을 고려하지도 않는 것으로 간주하고 연구를 하였다. 여러 참고문헌에서 소프트웨어 신뢰도 모델링에서 역일시간보다 그 노력지수(실행시간)가 현실적인 적용에서 더 적합하다는 것을 보여주고 있다. 관찰된 신뢰도 성장곡선의 형상이 테스트 노력의 시간분포에 강하게 의존하기 때문이다. 또 몇몇 참고문헌에서는 역일 테스트, 테스트 노력량, 테스트 노력에 의하여 검출되는 결함의 수 사이의 관계를 설명하는

SRGM을 제안하였다.

신뢰도와 비용을 고려한 연구로서 Okumoto와 Goel[5]은 전체 평균 소프트웨어 비용을 최소화시키는 비용-최적 SRP(software reliability process)를 발표하였다. Yamada와 Osaki[6]는 전체 평균 비용을 최소화시키고 소프트웨어 신뢰도를 만족시키는 전체평균비용-신뢰도-최적 SRP를 도입하였다. 이러한 연구결과를 참조하여 Hou, Kuo, Chang[7]은 지수 곡선과 로지스틱 곡선에 적용하는 연구를 수행하였다.

$N(t)$: 총 테스트노력
a : 초기부터 소프트웨어 내에 존재하고 있는 결함의 갯수
$m(t)$: $E[N(t)]$, 평균치 함수
$R(x|t)$: 소프트웨어의 신뢰도, 시각 $t(x\geq 0)$에서 결함이 검출된 후 $(t, t+x)$에서 고장이 일어나지 않을 확률
R_o : 목표신뢰도, $0<R_o<1$
C_1 : 테스트기간중의 결함수정비용
C_2 : 운전중의 결함수정비용
C_3 : 단위시간당 테스트 비용
$C(T)$: 전체 평균 소프트웨어 비용
T_{LC} : 소프트웨어의 수명주기
T : 전체 테스트 시간
T^* : 최적 소프트웨어 테스트 시간
T_1 : $dC(T)/dT=0$을 만족시키는 유일 해 T
T_2 : $R(x|T)=R_o$를 만족시키는 유일 해 T
α : 테스트노력 소모율
λ(t) : dm/dt 고장강도
w(t) : 시각 t에서의 테스트노력 소모
W(t) : w(t)의 적분치
r : 매 테스트노력당 결함검출율
A : 상수

본 논문에서는 로지스틱 곡선으로서 테스트노력의 시간종속 거동을 설명한다. 소프트웨어 테스트에서 결함 검출비가 현재의 내재 결함수에 비례하고, 임의의 테스트 시간에서 그 비례가 현재의 테스트 노력 여하에 달려 있다는 것을 가정하여 비동차 포아송 과정(Non-Homogeneous Poisson Process ; NHPP)[4]에 바탕을 둔 신뢰도 성장모델을 개발한다.

2항에서는 문헌에 있는 기존의 테스트노력함수를 간략하게 기술하고 제안된 로지스틱 테스트노력함수를 고찰하였다. 3항에서는 로지스틱형 SRGM을 연구하였다. SRGM의 파라미터는 LSE와 MLE를 이용하여 산출한다. 4항에서는 로지스틱테스트노력함수를 이용하여 비용면에서 본 최적 인도시간 T_1을 구하는 방법과, 목표신뢰도를 맞추는 최적인도시간 T_2를 구하는 방법을 연구한다. 그리고 이 두 가지 조건을 동시에 만족시킬 수 있는 방정식을 제시한다. 참고문헌에서 제시한 산출파라미터를 실제에 적용하는 예를 들었다. 5항에서는 이 모델을 비용-신뢰도 기준에 근거하여 최적인도수법을 적용한다.

2. 테스트 노력 함수에 대한 이론적 배경

테스트 노력은 CPU 시간의 양, 실행된 테스트 케이스의 수 등으로 나타낼 수 있다. 때때로 테스트 시간은 실행시간 대신 테스트의 수로 나타낼 수도 있다. 소프트웨어 신뢰도 모델링 분야에서는 소프트웨어의 개발 노력을 자주 전통적인 지수함수, 레일레이함수, 또는 웨이불 곡선으로 표현한다. 그러나, 많은 소프트웨어 테스트 환경에서 이러한 3개의 노력함수 곡선만으로 소프트웨어 테스트 노력함수를 기술하는 것은 어려운 일이다. 본 논문에서는 로지스틱 테스트노력함수가 소프트웨어 개발/테스트 노력 곡선으로 표현될 수 있다는 것을 보여주고자 한다. 실제 테스트/디버그 데이터 집합에 근거하여 실험을 수행해 왔다. 다양한 모델간의 예측능력을 비교해본 결과, 로지스틱 테스트노력함수를 가진 SRGM이 이전의 접근법에 비하여 초기결함의 수를 산출하는데에 더 적합하다는

것을 보여준다.

주어진 기간에 맞추어 소프트웨어 시스템을 개발할 때 시간, 자금, 인력과 같은 자원들이 소요된다. 일반적으로 이러한 소요자원의 시간종속 동태를 예측할 때 레일레이 곡선을 사용해왔다. 특히, 소프트웨어 테스트에까지 이르는 자원들이 소프트웨어 신뢰도에 상당한 영향을 미친다. 소프트웨어 개발 자원 전체의 약 40~50%가 테스트단계에서 소요된다. Yamada 등[3]은 테스트 기간중의 테스트 노력과 소프트웨어 개발 노력 모두가 레일레이 곡선이나 웨이불 곡선으로 설명될 수 있다는 것을 가정하여 소프트웨어 테스트에 쓰이는 테스트노력의 양을 고려한 소프트웨어 신뢰도 성장 모델을 제시하였다. 그들은 레일레이 곡선의 대안으로서 지수곡선도 제안하였다.

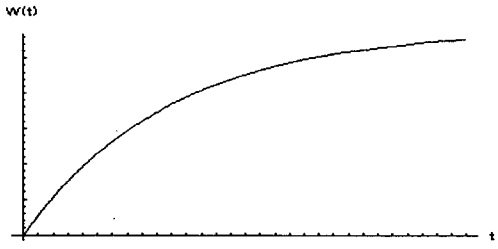
2.1 웨이불형 테스트노력 함수

Yamada등[3]이 제안한 웨이불형 테스트 노력 함수에 의하면 소프트웨어의 테스트 노력이 테스트 단계 전체에 걸쳐서 일정하다고 가정해서는 안된다고 한다. 순간적인 테스트 노력이 결국은 테스트 수명주기 동안 감소한다. 그 이유는 누적 테스트 노력이 유한치에 접근하기 때문이다. 그 어떤 소프트웨어 개발회사도 소프트웨어 개발에 무한정으로 자원을 투입하지 않기 때문에 이러한 가정이 합리적이라 할 수 있다. 여러 관련문헌에서는 테스트 노력이 웨이불형 분포로 설명될 수 있고, 아래와 같은 3개의 케이스를 가진다는 것을 보여주고 있다.

1) 지수함수곡선 : (0, t]에서 소모되는 누적 테스트 노력은

$$W(t)=N[1-\exp(-\beta t)] \qquad (2.1)$$

로서 웨이불함수의 m=1인 경우에 해당된다.

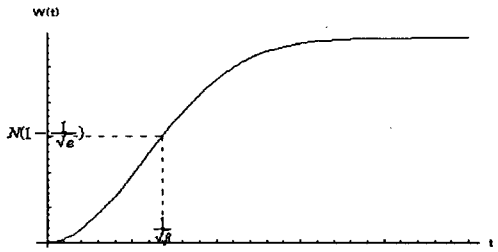


〈그림 2.1〉 지수함수형 누적테스트노력

2) 레일레이 곡선 : 소모되는 누적 테스트 노력은

$$W(t)=N[1-\exp(-\frac{\beta}{2}\cdot t^2)] \qquad (2.2)$$

로서 웨이불함수의 m=2인 경우에 해당된다.

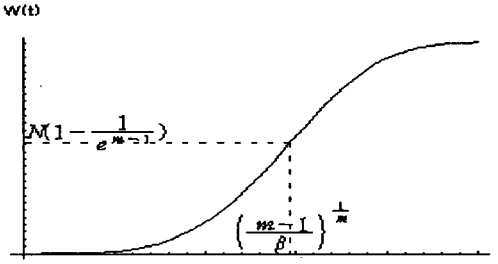


〈그림 2.2〉 레일레이형 누적 테스트노력 곡선

3) 웨이불 곡선 : 소모되는 누적 테스트 노력은

$$W(t)=N[1-\exp(-\beta t^m)] \qquad (2.3)$$

로서 웨이불 함수의 일반적인 경우, 즉 m=3, 4, …인 경우에 해당된다.

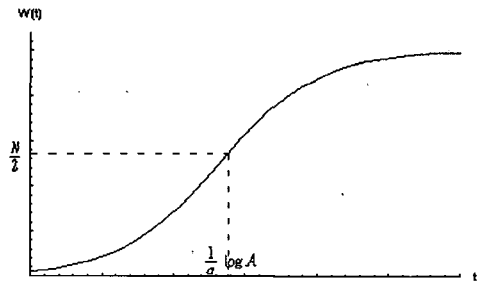


〈그림 2.3〉 웨이불형 누적테스트노력 곡선

웨이불형 곡선(2.3)에 대해서 m=1 또는 m=2일 때 그 결과는 각각 지수함수 곡선이거나 레일레이 곡선이다. 따라서, 이들은 웨이불 함수의 특수한 경우에 해당된다. m=3, 4, 5일 때는 소프트웨어 개발 공정기간 동안 공칭 피크 현상(원활하지 못하게 증가하며, 소모 곡선을 열화시킴)을 나타낸다. 즉, 피크 작업량이 나타난다. 이러한 현상은 실제 소프트웨어 개발/테스트 공정을 설명하는데 공통적으로 쓰이지 않으므로 현실적이지 못한 것으로 보인다. 그래서 웨이불 곡선은 형상 파라미터를 조정하여 많은 분포를 적합하게 맞출 수 있고 융통성 있게 테스트노력을 나타낼 수 있음에도 불구하고 테스트 노력 소모량을 모델링하는 데는 부적합한 것으로 사료된다.

2.2 로지스틱형 테스트 노력 함수

실제 테스트 노력 데이타가 여러 가지 소모 패턴을 나타내므로 때때로 테스트 노력 비용을 지수함수나 레일레이 곡선만으로 설명하기는 어렵다. 웨이불형 곡선은 일반적인 소프트웨어 개발 환경 하에서 데이터에 잘 맞지만, 그리고 소프트웨어 신뢰도 모델링에 널리 쓰이지만 m>3일 때 공칭 피크현상을 가진다. 그 대안으로 제시된 것이 로지스틱형 테스트노력 함수이다. 이 함수는 실제 프로젝트 탐사에 의해서 보고된 바와 같이 매우 정확하다. (0, t]에서의 누적 테스트 노력 소모는



〈그림 2.4〉 로지스틱형 누적테스트노력 곡선

$$W(t)) = \frac{N}{1 + A \cdot \exp(-\alpha t)} \tag{2.4}$$

이다.

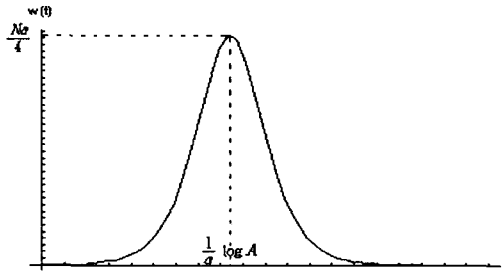
그림의 초기점에서 웨이불형 테스트 노력 함수와 비교하여 로지스틱 테스트노력 함수인 경우 $W(0) \neq 0$이다. 공정이 때때로 확인하기 힘든 소프트웨어 개발의 초기단계에 웨이불형 곡선과 W(t)와의 차이점이 존재한다. 그리고 적용된 테스트 노력의 양을 기록하기 위한 공식적인 계수 공정절차가 수립되지 못한 곳에서도 차이가 난다.

이러한 두 개의 모델을 실제 고장 데이타와 잘 맞는 통계적인 테스트를 이용하여 이러한 모델 사이를 판정하는 것이 가능하다.

현재의 테스트 노력 소모량은 W(t)의 미분치로서

$$w(t) = \frac{dW(t)}{dt} = \frac{NA\alpha \cdot \exp(-\alpha t)}{[1 + A \cdot \exp(-\alpha t)]^2} = \frac{NA\alpha}{[\exp(\alpha \frac{t}{2}) + A \cdot \exp(-\frac{\alpha t}{2})]^2} \tag{2.5}$$

와 같이 표현된다.



〈그림 2.5〉 로지스틱 테스트노력 소모량

그림에서 보는 바와 같이 이 양은 $t = \frac{1}{\alpha} \log A$일 때 최대치 $\frac{N\alpha}{4}$를 중심으로 좌우 대칭형이다.

3. 로지스틱 테스트노력 SRGM

기본 SRGM은 아래와 같은 가정 하에 적용한다.[8]

1) 결함검출 공정이 NHPP를 따른다.
2) 소프트웨어 시스템은 시스템에 잔존하는 결함에 의하여 발생되는 무작위 고장에 의존한다.
3) w(t)에 의하여 (t, t+Δt]동안에 검출되는 결함의 평균수는 시스템에 잔여하는 결함의 평균치에 비례한다.
4) 비례상수는 시간에 따라 변하지 않는다.
5) 테스트 노력 소모는 로지스틱 함수로 모델링한다.
6) 고장이 발생할 때마다 그것을 발생시키는 결함은 새로운 결함 도입 없이 즉시 그리고 완벽하게 제거된다.
7) 결함 제거 시간은 무시할 정도이며, 검출된 오류는 완벽하게 제거된다.

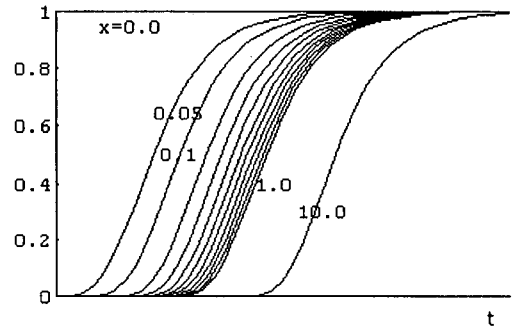
3.1 소프트웨어 신뢰도 척도

평균치 함수 m(t)를 가진 NHPP 모델에 의해서 소프트웨어 신뢰도 평가에 대한 두 가지 정량적 평가 척도를 얻을 수 있다. 시각 t에서의 신뢰도는 다음과 같다.

$$R(x \mid t) = \exp[-m(t+x)+m(t)] \quad (3.1)$$

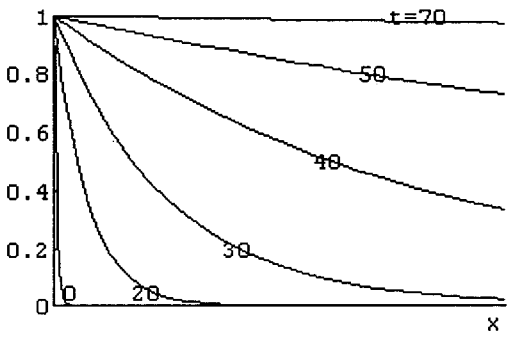
여기서, $R(x|t)$는 시각 t에서 최종적으로 결함을 발견하여 수정한 후 x 단위시간 동안 새로운 결함이 발견되지 않을 확률이다.

식 (3.1)로 표시된 신뢰도의 특성을 이해하기 위해 시험시간과 신뢰도의 관계 및 최종 검출 결함 수정 후 경과시간과 신뢰도의 관계를 그림으로 나타내면 <그림 3.1>, <그림 3.2>와 같다.



〈그림 3.1〉 인도시각과 신뢰도와의 관계

<그림 3.1>의 경우, 최종 결함 수정 후 경과되는 각각의 시간에 대해서 인도시각과 신뢰도 성장과의 관계를 보여주고 있다. 일정한 경과시간 x에 대해서 시험시간 및 인도시기를 늦추면 늦출수록 신뢰도가 성장함을 알 수 있다. 또한, 비록 신뢰도가 성장하여 목표신뢰도 이상이 될 수 있으나, 결함 수정 후 경과시간이 길어지면 길어질수록 결함에 의한 고장 확률이 높아 신뢰도가 저하된다는 것도 알 수 있다. 그림에서 x = 0.0일 때는 시험 시간에 관계없이 신뢰도가 1이나, x의 값이 커지면 커질수록 곡선이 횡축의 우측으로 이동하여 신뢰도가 저하됨을 알 수 있다.



〈그림 3.2〉 결함 수정후 경과시간과 신뢰도와의 관계

이와는 대조적으로 <그림 3.2>의 경우, 주어진 각각의 시험 시간 t에 대해서 최종 결함 수정 후 경과시간 x와 신뢰도 성장과의 관계를 보

여주고 있다. 일정한 시험시간 t에 대해서 경과시간 x가 작으면 작을수록 신뢰도가 높으며, 경과시간 x가 증가함에 따라 신뢰도가 급격히 감소된다. 이 그림에서 보듯 시험 시간이 길면 길수록 경과시간에 대한 신뢰도의 저하가 작아짐을 알 수 있다.

상기 식 (3.1)에서 정의한 시험 단계의 신뢰도 의미를 고찰해보기로 한다.

소프트웨어를 개발하여 결함시험을 하면 할수록 결함을 발견하여 수정하는 빈도가 작아지므로 신뢰도가 성장되며, 결함 수정 후 경과시간이 길어지면 길수록 결함 발견 확률이 높아지기 때문에 소프트웨어의 신뢰도는 낮아진다. 한편, 시험 단계에서는 얼마나 오랜 시간동안 결함이 발견되지 않느냐가 중요한 것이 아니라, 현 단계에서 소프트웨어 내에서 발견되지 않고 잔존하는 결함의 수가 얼마나 되는가가 더 중요하다.

3.2 로지스틱형 성장모델

통계적인 예측에 의하여 현재의 결함 내용은 어느 경우이든 유한하므로 m(t)는 t에 대한 증가함수이며 m(0)=0이다. 이러한 가정 하에

$$\frac{m(t+\Delta t)-m(t)}{w(t)} = r[a-m(t)]\cdot\Delta t\ ,$$

$$r = \lim_{\Delta t\to 0}\left(\frac{m(t+\Delta t)-m(t)}{[a-m(t)]w(t)\Delta t}\right) \quad (3.2)$$

이며, 현재의 테스트 노력 비용에 의해서 검출되는 결함의 수가 잔여결함의 수에 비례하면

$$\frac{dm(t)}{dt}\cdot\frac{1}{w(t)} = r[a-m(t)],$$

$$a>0,\ 0<r<1 \quad (3.3)$$

과 같이 방정식을 세울 수 있다.

즉 식(3.3)은 현재의 테스트 노력에 의해서 검출되는 결함의 수를 나타낸다.

경계조건 m(0)=0인 조건에서 (3.3)을 풀면

$$\frac{dm(t)}{dt} + rw(t)m(t) = raw(t),$$

$$m(t) = e^{-\int rw(t)dt}\left\{\int raw(t)e^{\int rw(t)dt}dt + k\right\}$$

$$= a + k\cdot e^{-r[W(t)-W(0)]}\text{에서}$$

$$m(0) = a+k=0$$

$$\therefore k=-a$$

$$\therefore m(t) = a(1-e^{-r[W(t)-W(0)]})$$

$$= a\{1-e^{-r(W(t)-W(0))}\}$$

$$= a\{1-\exp[-r\cdot W^*(t)\}$$

$$W^*(t) = \frac{N}{1+A\cdot\exp(-at)} - \frac{N}{1+A} \quad (3.4)$$

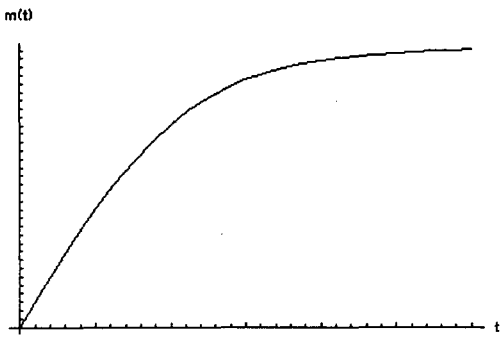
이고, 고장강도는

$$\lambda(t) = \frac{dm(t)}{dt} = arw(t)\cdot\exp[-rW^*(t)]$$

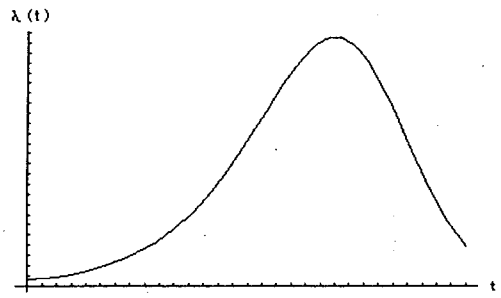
$$= arw(t)\cdot\exp\left(-\frac{N}{1+A\cdot\exp(-at)} + \frac{N}{1+A}\right)$$

$$= \frac{arNA\alpha}{\left[e^{\frac{-at}{2}} + Ae^{-\frac{at}{2}}\right]^2}\exp\left[\frac{N}{1+Ae^{-at}} - \frac{N}{1+A}\right] \quad (3.5)$$

이다.



〈그림 3.3〉 평균치함수



〈그림 3.4〉 고장강도함수

그러므로, 테스트를 무한시간 계속한다면 검출되지 않는 평균 결함수는

$$a-m(\infty)=a\cdot\exp\left[-r\cdot\frac{NA}{1+A}\right]$$
$$\approx a\cdot\exp(-rN)(A\gg 1) \qquad (3.6)$$

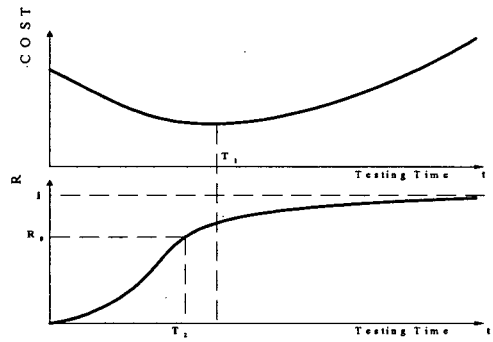
이다. 결국 테스트 단계 기간 동안에 쓰이는 노력이 N으로 제한되므로 유한 테스트 노력으로는 소프트웨어 시스템 내에 존재하는 모든 결함을 검출할 수는 없다는 것을 알 수 있다.

4. 비용-신뢰도를 고려한 인도 기법

소프트웨어 프로젝트 관리자에게 있어서는 소프트웨어 테스트를 중단하고 시스템을 사용자에게 인계하는 최적 인도 시각을 결정하는 것이 중요하다. 이것을 최적 소프트웨어 인도 문제라고 부른다. 이는 시스템의 신뢰도와 투입된 테스트자원 사이의 관계를 고려하여 공식화할 수 있다. 목표 신뢰도를 맞추면서 전체 소프트웨어 비용이 최소가 되도록 하는 것이 중요하다. 테스트 단계중의 테스트 노력 소요량 비용과 인도 전후의 결함 수정 비용을 소프트웨어 비용인자로 계산한다. <그림 4.1>에서는 테스트 기간을 횡축으로 하여 비용과 신뢰도의 관계를 동시에 나타낸 것이다. 최적 소프트웨어 인도문제를 다음과 같이 정의한다.

$c_2 > 0$, $c_1 > c_3 > 0$, $x \geq 0$, $0 < R_0 < 1$인 경우에 대해서

$R(x|T) \geq R_0$, $T \geq 0$인 조건하에 $C(T)$를 최소화 (4.1)



〈그림 4.1〉 비용-신뢰도 관계 곡선

이러한 방법으로 하여 비용-신뢰도최적 소프트웨어 인도시각에 대한 해를 구할 수 있다.

$$T^* = \max\{T_1,\ T_2\} \qquad (4.2)$$

여기서, T_1은 비용을 최저로 하는 인도시간, T_2는 목표신뢰도를 만족하는 인도시간이다. 비용을 최저로 하면서 목표신뢰도를 맞추기 위한 최적 인도 시각은 그 두 시간 중에 큰 값을 취해야 한다는 것을 의미한다.

4.1 비용 기준에 의한 인도 시각

본 항에서는 비용 기준에 근거하여 비용 모델을 고찰하고 인도기법을 고찰한다. 비용 기준에 의해서 산출된 소프트웨어 비용을 이용한 소프트웨어 테스트/개발 단계 기간 동안의 테스트 노력 비용과, 인도 전후의 오류 수정 비용은 다음과 같다.

$$C(T) = C_1 m(T) + C_2[m(T_{LC}) - m(T)]$$
$$+ C_3 \int_0^T w(x)dx$$

$$= -(C_2 - C_1)m(T) + C_2 m(T_{LC}) + C_3 W^*(T)$$

$$= -(C_2 - C_1)a\{1 - e^{-rW^*(T)}\} + C_2 a\{1 - e^{-rW^*(T_{LC})}\}$$

$$+ C_3\left\{\frac{N}{1+Ae^{-\alpha T}} - \frac{N}{1+A}\right\}$$

$$= (C_2 - C_1)ae^{-r\left[\frac{N}{1+Ae^{-\alpha T}} - \frac{N}{1+A}\right]} + \frac{C_3 N}{1+Ae^{-\alpha T}}$$

$$+ a\left\{C_1 - C_2 e^{-r\left[\frac{N}{1+Ae^{-\alpha T_{LC}}} - \frac{N}{1+A}\right]}\right\} - \frac{C_3 N}{1+A} \tag{4.3}$$

C_2는 보통 C_1의 크기보다 크기 때문에 $C_1 < C_2$ 이다.

최적값(비용 최저값)을 구하기 위해 식 (4.3)을 T에 관하여 미분한다.

$$\frac{d}{dT}C(T) = C_1\frac{dm(T)}{dT} - C_2\frac{dm(T)}{dT} + C_3 w(T)$$

$$= w(T)\{-(C_2 - C_1)ar \cdot \exp[-rW^*(T)] + C_3\} \tag{4.4}$$

이 식에서 비용이 최소가 되는 시각을 구하면

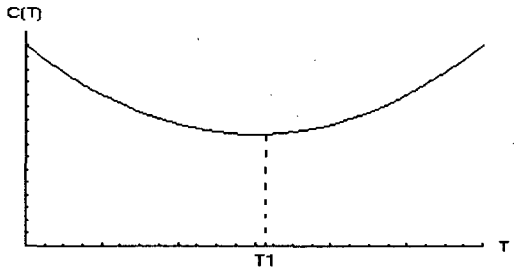
$$e^{-rW^*(T)} = \frac{C_3}{ar(C_2 - C_1)}$$

$$\frac{N}{1+Ae^{-\alpha T}} - \frac{N}{1+A} = -\frac{1}{r}\log\frac{C_3}{ar(C_2 - C_1)}$$

$$e^{-\alpha T} = \frac{1}{A}\,\frac{NA + \frac{1+A}{r}\log\frac{C_3}{ar(C_2 - C_1)}}{N - \frac{1+A}{r}\log\frac{C_3}{ar(C_2 - C_1)}}$$

$$T_1 = \frac{1}{\alpha}\log\frac{N - \frac{1+A}{r}\log\frac{C_3}{ar(C_2 - C_1)}}{N + \frac{1+A}{Ar}\log\frac{C_3}{ar(C_2 - C_1)}} \tag{4.5}$$

으로서 이 관계를 그림으로 나타내면 <그림 4.2>와 같다.



<그림 4.2> 인도시각 - 비용 관계 곡선

여기서 $0 < T_1 < T_{LC}$인 조건이 되어야 한다.

$$0 < \frac{1}{\alpha}\log\frac{N - \frac{1+A}{r}\log\frac{C_3}{ar(C_2 - C_1)}}{N + \frac{1+A}{Ar}\log\frac{C_3}{ar(C_2 - C_1)}} < T_{LC}$$

인 범위 즉

1) $\exp\left[-\frac{Ar(e^{\alpha T_{LC}} - 1)}{(1+A)(A+e^{\alpha T_{LC}})}\right] \le \frac{C_3}{ar(C_2 - C_1)} \le 1$

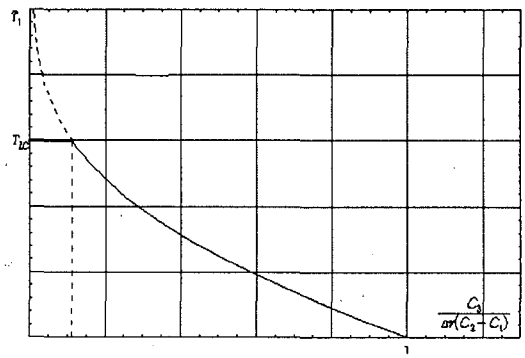
에서 유일하고도 유한한 양의 해 $T^* = T_1$이 존재하고 그 값은 (4.5)와 같다.

2) $\frac{C_3}{ar(C_2 - C_1)} > 1$이면 $T_1 < 0$이므로 $T^* = T_1 = 0$이다.

3) $\frac{C_3}{ar(C_2 - C_1)} < \exp\left[-\frac{Ar(e^{\alpha T_{LC}} - 1)}{(1+A)(A+e^{\alpha T_{LC}})}\right]$

이면 $T_1 > T_{LC}$이므로 $T^* = T_1 = T_{LC}$이다.

이와 같은 내용을 <그림 4.3>에 표시하였다.



<그림 4.3> 조건에 따른 인도시각

즉, $\frac{C_3}{ar(C_2-C_1)} > 1$이면 비용이 단조증가하는 경우로서 결함시험을 하면 할수록 비용이 증가되어 시험 없이 인도하는 것이 최적인 것을 의미한다.

$$\exp\left[-\frac{Ar(e^{aT_{LC}}-1)}{(1+A)(A+e^{aT_{LC}})}\right] \le \frac{C_3}{ar(C_2-C_1)} \le 1$$

이면 비용최저점이 결함시험과 소프트웨어의 전 수명기간 사이에 존재하는 경우이다. 이러한 경우는 본 논문이 추구하고자 하는 이상적인 경우로서 목표신뢰도를 만족시키는 인도시기와 총 비용을 최저로 하는 인도시기 중 큰 값을 취하는 것이 이상적이다.

$$\frac{C_3}{ar(C_2-C_1)} < \exp\left[-\frac{Ar(e^{aT_{LC}}-1)}{(1+A)(A+e^{aT_{LC}})}\right]$$

인 조건은 비용이 단조감소하는 경우이다. 이러한 경우는 결함시험을 하면 할수록 총 비용이 감소된다. 이러한 경우는 소프트웨어의 결함시험에 의해서 신뢰도가 높아지고, 따라서 드물게 결함이 검출되어도 그 수정비용이 미미한 경우에 해당된다.

주어진 조건에서 MLE와 LSE를 이용하여 관련 파라미터를 구하고 이 파라미터를 이용하여 식 (4.5)으로부터 비용 최적 인도시각 T_1을 결정한다.

4.2 신뢰도 기준에 의한 인도 시각

신뢰도 기준에 의한 인도수법을 고찰한다. 일반적으로 소프트웨어 인도시간문제는 소프트웨어 시스템의 신뢰도와 연동되어 있다. 소프트웨어 시스템의 신뢰도가 적정한 수준에 이르면 소프트웨어를 인도하는 시간이 된 것을 알 수 있다. 여러 참고문헌에서는 소프트웨어의 비용경제성 면에서 인도문제를 검토하였다. 시각 t에서 마지막 고장이 발생한 후 조건 확률함수는 아래와 같으며, 여기서 평균치함수 $m(t)$는 식 (3.4)와 같다.

$$R = R(x|t) = \exp\{-[(m(t+x)-m(t)]\}$$
$$= \exp\{-a[\exp(-rW^*(t)) - \exp(-rW^*(t+x))]\} \quad (4.6)$$

위의 식으로부터

$$\log R(x|0) = -m(x) = -a(e^{-rW^*(0)} - e^{-rW^*(x)})$$
$$= -a(1-e^{-rW^*(x)})$$
$$= -a\left\{1-e^{-r\left(\frac{N}{1+Ae^{-\alpha}}-\frac{N}{1+A}\right)}\right\}$$
$$R(x|0) = \exp\left[-a\left(1-e^{-r\left(\frac{N}{1+Ae^{-\alpha}}-\frac{N}{1+A}\right)}\right)\right] \quad (4.7)$$

과, 또 이 결과로부터

$$e^{-\alpha} = \frac{1}{A}\left\{\frac{1+A}{1-\frac{1+A}{rN}\log[1+\frac{1}{a}\log R(x|0)]}-1\right\} \quad (4.8)$$

을 얻는다.

그리고, 식 (4.6)에 대수를 취한다.

$$\log[R(x|t)] = -[m(t+x)-m(t)]$$

목표신뢰도를 R_0라하고 위의 식을 정리하면

$$\log R_0 = \log R(x|T) = -a(e^{-W^*(T)} - e^{-W^*(T+x)})$$
$$= -ae^{\frac{rN}{1+A}}\left\{e^{-\frac{rN}{1+Ae^{\alpha T}}} - e^{-\frac{rN}{1+Ae^{\alpha(T+x)}}}\right\}$$
$$\log\frac{1}{R_0} = ae^{\frac{rN}{1+A}}\left[e^{-\frac{rN}{1+Ae^{-\alpha T}}} - e^{-\frac{rN}{1+Ae^{-\alpha(T+x)}}}\right] \quad (4.9)$$

를 만족하는 시각 T_2를 구한다.

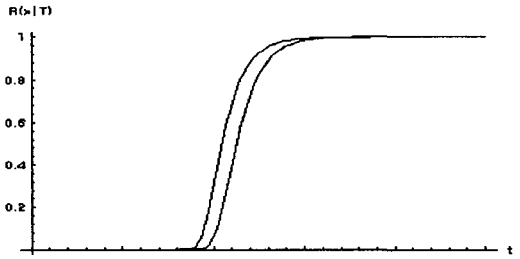
여기서, N, A, α, β는 최소자승법으로 구한다.

$$S(N,\ A,\ \alpha) = \sum_{k=1}^{n}[W_k - W(t_k)] \quad (4.10)$$

W_k=(0, t] 기간동안 실제로 소요되는 누적테

스트노력이고, $W(t_k)$=테스트노력함수에 의해서 산출된 누적 테스트 노력이다.

식 (4.9)를 풀어서 원하는 R_0에 이르는 시간을 결정한다. 식 (4.9)를 이용하여 목표신뢰도에 이르는 신뢰도를 얻는데 필요한 테스트시간을 얻거나 또는 주어진 시간간격 동안 신뢰도가 목표치에 도달하는지 아닌지를 결정할 수 있다.



〈그림 4.4〉 인도시각과 신뢰도와의 관계

식 (4.9)를 풀기가 쉽지 않다. 따라서, 수치해석적인 방법으로 식 (4.10)을 반복해서 풀어서 계수를 구한 후 해를 구하던가, 그림과 같은 그래프를 그려서 구하는 방법이 최선이라 할 수 있다.

주어진 조건에서 MLE와 LSE를 이용하여 관련 파라미터를 구하고 이 파라미터를 이용하여 식 (4.9)로부터 비용 최적 인도시각 T_2를 결정한다.

5. 적용 예

[8]로부터 다음과 같은 데이터를 인용한다.

N = 29.1095, A = 4624.89, α = 0.493515, a = 138.165, r = 0.145098, C_1 = 1, C_2 = 100, C_3 = 50, T_{LC} = 100, R_0 = 0.95, x = 1이다.

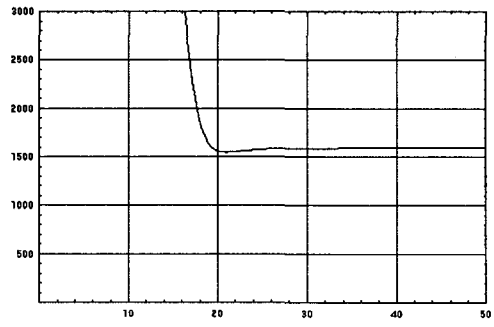
여기서, N은 테스트 기간중의 총 테스트 노력량, A는 로지스틱 테스트 노력 함수의 지수함수에 관계된 상수, a는 테스트 노력이 소요되는 소모율, a는 테스트 시작 전에 원래부터 소프트웨어 내에 존재하고 있는 결함의 총 수, r는 매 테스트 노력당 결함검출비, C_1은 테스트 기간중의 결함 수정 비용, C_2는 운영중에 발생되는 결함에 대한 수정 비용, C_3는 테스트 기간중의 단위 시간당 테스트 비용, T_{LC}는 소프트웨어의 사용 수명 기간, R_0는 사용자가 요구하는 목표 신뢰도 수준, x는 소프트웨어 인도 후 운영되는 경과 시간을 나타낸다.

5.1 비 용

4.1항의 조건에서

$$\frac{C_3}{ar(C_2-C_1)} = \frac{50}{138.165 \cdot 0.145098 \cdot 99}$$

= 0.025로서 조건1)에 해당하므로 식 (4.5)를 이용하여 그 시각을 구하면 T_1 = 20.04가 되고 식 (4.3)을 이용하여 최저비용을 구하면 C = 1,548.8이 된다. 이는 <그림 5.1>에서도 명백하게 드러난다. 식 (4.6)을 이용하여 이 때의 신뢰도를 구하면 0.5453으로서 고객이 요구하는 목표 신뢰도와 거리가 멀다. 즉, 비용을 최저로 하는 시점에서 소프트웨어의 테스트를 중지하여 고객에게 인도하게 되면 소프트웨어의 신뢰도가 54.53%밖에 되지 않아서 목표신뢰도 95%에 이르기 위해서는 좀더 경과시간을 두고 테스트를 계속하여 신뢰도를 향상시켜야 한다.

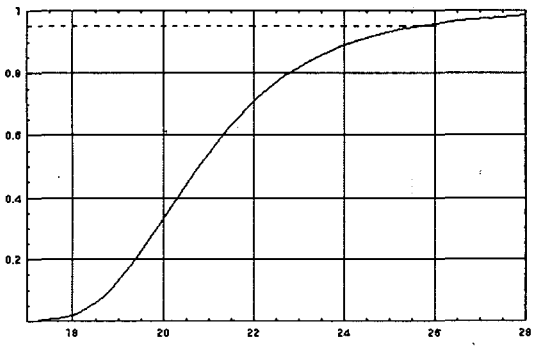


〈그림 5.1〉 적용 예의 비용곡선

5.2 신뢰도

한편, 식 (4.9)을 이용하여 목표신뢰도 95%를 만족하는 시각을 구하면 $T_2 = 26$이 된다. 이는 소프트웨어의 신뢰도를 목표치까지 상승시키기 위해서 그 시간이 26으로 될 때까지 소프트웨어의 테스트를 계속하여 신뢰도를 향상시켜야 함을 의미한다. 이 수치는 처음에 어떻게 정하냐에 따라 일수(日數)가 될 수도, 주수(週數)가 될 수도 있다.

따라서, (4.2)의 조건에 따라 $T^* = T_2 = 26$이 되어야 하며, 이 때의 총 비용은 1,584.4로서 최저비용보다 35.6만큼 증가된다.



〈그림 4.5〉 적용 예의 신뢰도 곡선

이 예에서는 $T_{LC} = 100$이라는 수명 기간 동안 경과시간 $x = 1$일 때의 목표 신뢰도 $R_0 = 0.95$를 유지하기 위한 시간을 구함에 있어서 비용을 최저로 하는 시간을 구하고 목표 신뢰도를 만족하는 시간을 구해봤을 때 $T_1 < T_2$인 경우로서 인도시간이 비용최저에 의해서 결정된 것이 아니고 목표 신뢰도를 맞추는 시점에서 결정되었다. 일반적으로 목표 신뢰도에 의해서 인도시기가 결정되는 경우가 많다. 이는 비용면에서 바람직하다고 보기 어려우므로 목표 신뢰도에 맞추어 발행할 수 있도록 상기 각각의 경우를 개선하는 것이 바람직하다.

6. 결 론

개발소프트웨어의 테스트노력함수로서 로지스틱함수를 채택하여 개발소프트웨어의 비용과 신뢰도를 동시에 만족시키는 기법을 연구하고자 하였다.

비용면에서는 개발, 테스트, 인도, 인도 후 A/S에 대한 총 비용을 구하는 방법을 연구하고 이를 최소비용으로 줄일 수 있는 기법을 제시하였다. 이는 조건에 따라서 그 시기가 달라진다.

즉, $\frac{C_3}{ar(C_2 - C_1)} > 1$이면 비용이 단조증가하는 경우로서 결함시험을 하면 할수록 비용이 증가되어 시험 없이 인도하는 것이 최적인 것을 의미한다.

$$\exp\left[-\frac{Ar(e^{\alpha T_{LC}} - 1)}{(1+A)(A+e^{\alpha T_{LC}})}\right] \le \frac{C_3}{ar(C_2 - C_1)} \le 1$$

이면 비용최저점이 결함시험과 소프트웨어의 전 수명기간 사이에 존재하는 경우이다. 이러한 경우는 본 논문이 추구하고자 하는 이상적인 경우로서 목표신뢰도를 만족시키는 인도시기와 총 비용을 최저로 하는 인도시기 중 큰 값을 취하는 것이 이상적이다.

$$\frac{C_3}{ar(C_2 - C_1)} < \exp\left[-\frac{Ar(e^{\alpha T_{LC}} - 1)}{(1+A)(A+e^{\alpha T_{LC}})}\right]$$

인 조건은 비용이 단조감소하는 경우이다. 이러한 경우는 결함시험을 하면 할수록 총 비용이 감소된다. 이러한 경우는 소프트웨어의 결함시험에 의해서 신뢰도가 높아지고, 따라서 드물게 결함이 검출되어도 그 수정비용이 미미한 경우에 해당된다.

또한, 신뢰도 기준에 의한 인도수법을 고찰한다. 일반적으로 소프트웨어 인도시간문제는 소프트웨어 시스템의 신뢰도와 연동되어 있다. 소프트웨어 시스템의 신뢰도가 적정한 수준에 이르면 소프트웨어를 인도한다. 마지막 고장이 발생한

후 조건 확률함수를 이용하여 목표신뢰도를 만족시키는 시간을 결정하는 방정식을 제시하였다.

이 방정식을 풀어서 원하는 R_0에 이르는 시간을 결정한다. 그 방정식을 이용하면 목표신뢰도에 이르는 신뢰도를 얻는데 필요한 테스트시간을 얻거나 또는 주어진 시간간격 동안 신뢰도가 목표치에 도달하는지 아닌지를 결정할 수 있다. 그러나, 불행하게도 신뢰도 방정식을 풀어서 적정시간을 구하는 것이 쉽지 않다. 이를 수치해석적인 방법으로 해를 구하던가, 그래프를 그려서 구하는 방법이 최선이라 할 수 있다.

참 고 문 헌

[1] Ramamoorthy, C.V. and Bastani, F.B., "Software reliability-Status and perspectives", IEEE Trans. on Software Eng., Vol. SE-8, Aug. 1982, pp. 354-371.

[2] Musa, J.D., Iannino, A. and Okumoto, K., "Software Reliability : Measurement, Prediction, Application", Mar. 1987, pp. 230-238,

[3] Yamada, S., Ohtera, H. and Narihisa, H., "Software reliability growth models with testing-efforts", IEEE Trans. Reliability, Vol. R-35, Apr. 1986, pp. 19-23.

[4] Ascher, H. and Feigold, H., "Repairable Systems Reliability : Modeling, Inference, Misconceptions, and Their Causes", 1984, Marcel Dekker.

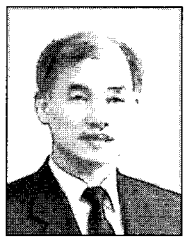
[5] Okumoto, K. and Goel, A.L., "Optimum release time for software systems based on reliability and cost criteria", J. System software, Vol. 1, 1980, pp. 315-318.

[6] Yamada, S. and Osaki, S., "Cost-reliability optimal release policies for software systems", IEEE Trans. on Reliability, Vol. R-34, Dec. 1985, pp. 422-424.

[7] Rong-Huei Hou, Sy-Yen Kuo, Yi-Ping Chang, "Optimal release policy for hyper-geometric distribution software-reliability growth model", IEEE Trans. on Reliability, Vol. 45, Dec. 1996, pp. 646-651.

[8] Chin-Yu Huang, Sy-Yen Kuo, "Analysis of Incorporating Logistic Testing-Effort Function into Software Reliability Modeling", IEEE Trans. on Reliability, Vol. 51, Sep. 2002, pp. 261-270.

□ 저자소개



최 규 식

저자는 서울대학교 공과대학 전기과를 졸업하고, 뉴욕공과대학에서 석사학위를 받았으며, 명지대학교 전기과에서 박사학위를 받았다. OPC 중앙연구소와 KOPEC 중앙연구소에서 근무하였다. 현재는 건양대학교 정보전자통신공학부 교수이며, 관심분야는 무선통신 및 소프트웨어 신뢰도이다.

김 용 경

저자는 고려대학교를 졸업하고, 숭실대학교에서 이학석사, 명지대학교에서 경영학 박사학위을 취득했다. 현재 건양대학교 경영정보학과 교수이며, 한국정보기술응용학회 회장이다. 관심분야는 소프트웨어공학, 소프트웨어 품질관리, 정보시스템 감리 및 감사 등이다.

◈ 이 논문은 2004년 1월 14일 접수하여 2차 수정을 거쳐 2005년 3월 10일 게재확정되었습니다.