운영중인 소프트웨어의 신뢰도에 관한 연구

최규식*

A Study on the Software Reliability of Operational Stage S/W

Gyu-Shik Che*

요 약

소프트웨어를 개발하여 발행하기 전에 품질을 향상시키는 방법으로서 신뢰도를 향상시켜야 하는데, 이의 직접적인 방법으로서는 테스트에 의해 결함을 검출하고 수정해가는 것이다. 신뢰도가 목표치에 도달하여 발행했을 때, 운영중의 신뢰도 문제가 발생하게 된다. 개발 발행된 소프트웨어가 패키지소프트웨어라 불리는 범용소프트웨어냐, 아니면 특수목적을 가진 전용소프트웨어냐에 따라 운영소프트웨어의 신뢰도가 달라진다. 본 논문에서는 범용소프트웨어와 전용소프트웨어의 운영중의 신뢰도를 산출하는 방법을 제안하고자 한다.

ABSTRACT

One method to improve quality before releasing of S/W after development is to enhance the reliability, whose direct methodology is to detect and revise fault through testing. Once the S/W is released because it meets the target reliability, the operational reliability problem arises. It is obvious the operational reliability different from that of testing stage depending on the condition whether it is universal(package) S/W or dedicated S/W. I propose the methodology to calculate opertional software reliability of universal and dedicated S/W in this paper.

키워드

SRGM, mean value function, testing reliability, operational reliability, universal S/W, dedicated S/W, fault strengh

Ⅰ. 서 론

소프트웨어가 주어진 시간 간격 동안 고장이 발생하지 않을 확률 즉, 신뢰도는 소프트웨어의 테스트 과정을 계속해서 반복 및 수정하면 더욱 더 향상된다. 그러한 결함 검출 현상을 설명해주는 소프트웨어 신뢰도 모델을 소프트웨어 신뢰도성장 모델(SRGM ; software reliability growth model)[1]이라 한다.

한편, 소프트웨어 개발에는 많은 개발자원들이 소요된다. Musa[2]는 기존 소프트웨어 신뢰도 성장모델을 분류하는 하나의 안을 개발하였다. Yamada[3]는 역일 테스트 시간, 테스트 노력량, 테스트 노력에 의해서 검출되는 소프트웨어 결함의 수 사이의 관계를 명시적으로 설명할 수 있는 간단하고도 새로운 모델을 제시하였다.

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 신뢰도성장모델을 이용하는 것

* 건양대학교

접수일자 2008. 08. 28

이다.

신뢰도와 비용을 고려한 연구로서 Okumoto와 Goel[5]은 전체평균 소프트웨어 비용을 최소화시키는 비용-최적 SRP를 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.

그런데 소프트웨어를 발행하기 전에 결함을 발견하여 수정하는 테스트단계와는 달리 운영단계에서는 현실적으로 디버깅이 어렵기 때문에 분명히 테스트단계와는 신뢰도에 차이가 있다.

따라서 본 논문에서는 운영단계의 소프트웨어 신뢰도를 연구함에 있어서 그것이 전용소프트웨어인가, 또는 범용소프트웨어인가에 따른 소프트웨어의 신뢰도 성장 차이를 연구하고자 한다.

Ⅱ. SRGM

2.1 모델설명

소프트웨어의 평가에 필수적인 평균치 함수를 다음과 같이 정의한다.

$$m(t) = a(1 - \exp[-rW(t)]) \tag{2.1}$$

여기서, a는 소프트웨어의 개발 초기부터 소프트웨어 내에 존재하고 있었을 것으로 추정되는 결함의 총수이고, W(t)는 웨이불형 테스트노력함수이며, r은 상수이다.

소프트웨어 결함검출 현상을 통계적으로 모델링할 때에 식(2.1)의 m(t)에 의한 NHPP에 근거하여 N(t)의 평균치함수를 정의하면 웨이블 테스트노력함수를 고려한 소프트웨어 신뢰도성장모델을 만들 수 있으며,

$$\Pr\{N(t) = n\} = poim(n; m(t)) \tag{2.2}$$

NHPP 고장강도함수는 평균치 함수의 미분 현태이다.

$$\lambda(t) = dm(t)/dt = a \cdot r \cdot w(t) \cdot \exp[-rW(t)] \tag{2.3}$$

식(2.1)로부터 N(t)의 제한적인 분포가 평균치 함수

$$m(\infty) = a(1 - \exp[-r^{\alpha}]) \tag{2.4}$$

을 가진 포아송분포라는 것을 보여주고 있다.

2.2 소프트웨어 신뢰도 척도

시각 t에서의 신뢰도는 다음과 같이 정의한다.

$$R(x \mid t) = \exp\{-a(\exp[-r \cdot W(t)] - \exp[-r \cdot W(t+x)])\} \tag{2.5}$$

식(2.5)로 표시된 신뢰도의 특성을 이해하기 위해 테스트시간과 신뢰도의 관계 및 최종 검출 결함 수정 후 경과시간과 신뢰도의 관계를 그림으로 나타내면 각각 그림 1, 그림 2와 같다.

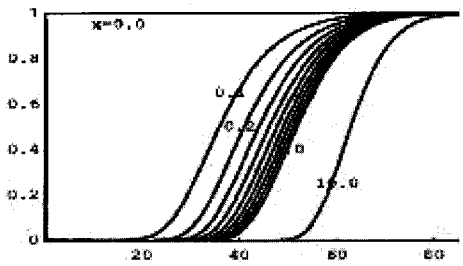

그림 1. 발행시각과 신뢰도와의 관계
Fig. 1 release time vs. reliability

그림 1의 경우, 주어진 경과시간 x에 대해서 테스트시간 및 발행시기를 늦추면 늦출수록 신뢰도가 성장함을 알 수 있다. 이와는 대조적으로 그림 2의 경우, 주어진 각각의 테스트 시간 t에 대해서 최종 결함 수정 후 경과시간 x와 신뢰도 성장과의 관계를 보여주고 있다. 일정한 테스트시간 t에 대해서 경과시간 x가 작으면 작을수록 신뢰도가 높으며, 경과시간 x가 증가함에 따라 신뢰도가 급격히 감소된다.

소프트웨어를 $t = T$에서 발행하는 경우,

$$m(T + x) = a(1 - e^{-\gamma W(T) - \gamma W(x)}) \tag{2.6}$$

이다. 그러므로,

$$R(x|T)=\exp\left\{-ae^{-\alpha\gamma(1-e^{-\beta T})}[1-e^{-\alpha\gamma(1-e^{-\beta x})}]\right\} \quad (2.7)$$

이다. 여기서, T는 특정소프트웨어의 발행시간, α와 β는 상수이다.

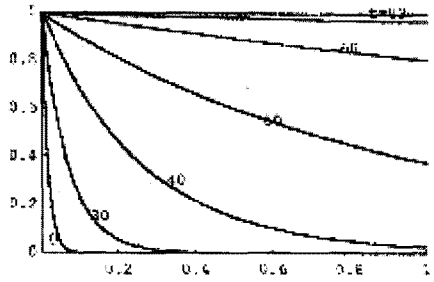

그림 2. 경과시간과 신뢰도와의 관계
Fig. 2 process time vs. reliability

Ⅲ. 테스트 노력 함수

Yamada[3]는 테스트 기간중의 테스트 노력과 소프트웨어 개발 노력 모두가 레일레이 곡선이나 웨이불곡선으로 설명될 수 있다는 것을 가정하여 소프트웨어 테스트에 쓰이는 테스트노력의 양을 고려한 소프트웨어 신뢰도성장 모델을 제시하였다. 그러나, 본 논문에서는 로지스틱 테스트 노력 함수도 소프트웨어 개발/테스트 노력 곡선으로 표현될 수 있다는 것을 보여주고자 하며, 운영단계의 신뢰도를 두 가지 경우로 나누어서 연구하고자 한다.

3.1 평균치 함수 및 신뢰도

소프트웨어를 시각 t에서 발행하는 경우, 발견되는 누적 분포는 평균치 함수

$$m(t)=a(1-e^{-\gamma W^*(t)}) \quad (3.1)$$

로 정의되므로

$$m(t+x)=a(1-e^{-\gamma W^*(t)-\gamma W^*(x)}) \quad (3.2)$$

이며,

$$R(x|t)=\exp[-a\cdot e^{-\gamma W^*(t)}(1-e^{-\gamma W^*(x)})] \quad (3.3)$$

이 된다.

3.2 웨이불형 함수

여러 관련 문헌에서는 테스트 노력이 웨이불형 분포로 된다는 것을 주장하고 있으며, 대표적으로 아래와 같은 3개의 케이스를 가진다는 것을 보여주고 있다.

1) 지수함수곡선 : (0, t]에서 소요되는 누적 테스트 노력은

$$W(t)=N[1-\exp(-\beta t)] \quad (3.4)$$

로서 웨이불 함수의 m=1인 경우에 해당되며, 신뢰도는

$$R(x|t)=\exp[-a\cdot e^{-rN(1-e^{-\beta t})}\cdot(1-e^{-rN(1-e^{-\beta x})})] \quad (3.5)$$

로 표현된다.

2) 레일레이 곡선 : 누적 테스트 노력은

$$W(t)=N[1-\exp(-\frac{\beta}{2}\cdot t^2)] \quad (3.6)$$

로서 웨이불함수의 m=2인 경우에 해당된다. 이 때 신뢰도는

$$R(x|t)=\exp[-a\cdot e^{-rN(1-e^{-\frac{\beta}{2}t^2})}\cdot(1-e^{-rN(1-e^{-\frac{\beta}{2}x^2})})] \quad (3.7)$$

로 표현된다.

3) 웨이불 곡선 : 소요되는 누적 테스트 노력은

$$W(t)=N[1-\exp(-\beta t^m)] \quad (3.8)$$

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

$$R(x|t)=\exp[-a\cdot e^{-rN(1-e^{-\beta t^m})}\cdot(1-e^{-rN(1-e^{-\beta x^m})})] \quad (3.9)$$

로 표현된다.

3.3 로지스틱형 함수

웨이불형 곡선은 일반적인 소프트웨어 개발 환경 하에서 데이터에 잘 맞지만, m>3일 때 공칭 피크현상을 가진다. 그 대안으로 제시된 것이 로지스틱형 테스트노력 함수이다. (0, t]에서의 누적 테스트 노력 소요는

$$W(t))=\frac{N}{1+A\cdot\exp(-\alpha t)} \quad (3.10)$$

이며 신뢰도는

$$R(x|t)=\exp[-a\cdot e^{-rN(\frac{1}{1+Ae^{-\alpha t}}-\frac{1}{1+A})}\cdot(1-e^{-rN(\frac{1}{1+Ae^{-\alpha x}}-\frac{1}{1+A})})] \quad (3.11)$$

로 표현된다. 여기서, a는 소프트웨어의 발행시부터 내재하고 있을 것으로 생각되는 총 결함의 수이고, A는 상수, α는 테스트노력 소요비율이다. 현재의 테스트 노력 소요량은 W(t)의 미분치로서

$$w(t)=\frac{NA\alpha}{[\exp(\alpha\frac{t}{2})+A\cdot\exp(-\frac{\alpha t}{2})]^2} \quad (3.13)$$

와 같이 표현된다.

Ⅳ. 단계별 신뢰도

4.1 테스트 신뢰도

시간간격 X_k가 소프트웨어의 테스트 단계이면 즉, 소프트웨어가 아직도 테스트중이고 테스트공정이 NHPP를 따르면 이의 표준 이론으로부터 평균치 함수를 정의할 때, 임의의 $t\geq 0$과 $x>0$에서

$$\Pr\{N(t+x)-N(t)=k\}=\frac{[m(t+x)-m(t)]^k}{k!}exp\{-[m(t+x)-m(t)]\} \quad (4.1)$$

이므로, 신뢰도는 다음과 같이 표현할 수 있다.

$$R_t(x|t)=\exp[-m(x)e^{-b_1 t}] \quad (4.2)$$

$R_t(x|t)$가 테스트 공정기간의 소프트웨어 신뢰도만을 측정할 수 있으며, 운영단계까지 연장되지 않으므로 식(4.2)에서 정의된 소프트웨어 신뢰도를 테스트 신뢰도로 부를 수 있다.

4.2 운영 신뢰도

운영 신뢰도는 운영 단계에서 결함이 발생되지 않을 확률이다. 그간 연구된 논문들, 예를 들면 [5], [6]과 같은 연구에 의하면 소프트웨어를 발행한 운영단계에서도 테스트단계와 마찬가지로 결함을 발견하여 수정할 수 있는 것으로 가정하여 신뢰도의 성장이 가능한 것으로 연구를 하였다. 그러나 실제로는 운영 기간 중에는 테스트단계와는 디버깅 환경이 다르므로 결함을 제거할 수 있는 경우도 있고, 그렇지 않은 경우도 있다. 보통 패키지 소프트웨어라 불리는 범용소프트웨어의 경우는 일단 발행되면 운영중에 소프트웨어의 품질에 문제가 있거나 다른 요건이 생겨도 이를 수정하기는 현실적으로 어렵다. 이와는 대조적으로 고객으로부터 주문을 받아서 특수한 용도로 개발하는 전용소프트웨어의 경우에는 운영중에 검출되는 결함에 대해서 고객의 요구에 의해 계속 수정을 하며, 품질을 유지 관리하기 때문에 고장률을 낮출 수 있으며, 따라서 신뢰도의 성장도 가능하다. 그러므로 운영단계에서는 발행된 소프트웨어가 범용소프트웨어인가, 전용소프트웨어인가에 따라 시간경과에 따른 신뢰도에 큰 차이가 있다.

4.2.1 범용소프트웨어

시간간격 X_k가 운영단계에 있다면 다음 고장시각은 파라미터 λ_r을 가진 지수분포를 따르며, 여기서 $\lambda_r=\lambda(t)$는 시각 t=T에서 계산된 본래 NHPP의 고장강도 함수이다. 범용소프트웨어의 신뢰도를 Ru(x|T)로 표현하면 그 함수는 아래와 같다.[8]

$$R_u(x|T)=\exp(-\int_0^x \lambda_r dt)=\exp[-\lambda(T)x] \quad (4.3)$$

4.2.2 전용소프트웨어

이 경우 평균치 함수는 다음과 같이 된다. 발행시각 T에서의 누적 결함수는

$$m(t)=ae^{-b_1 T}(1-e^{-b_2(t-T)})+a(1-e^{-b_1 T}) \quad (4.4)$$

따라서,

$$m_2(T_{LC})-m_1(T)=ae^{-b_1 T}(1-e^{-b_2(T_{LC}-T)}) \quad (4.5)$$

$$R_d(x|t)=\exp[-ae^{-b_1 T}(1-e^{-b_2 x})] \quad (4.6)$$

여기서, $m_1(t)$는 발행 전 평균치함수, $m_2(t)$는 발행 후 평균치 함수, T_{LC}는 소프트웨어의 수명기간이다.

운영기간 중에는 대부분의 경우, 그것이 주문형 소프트웨어라 할지라도 테스트기간에 비하여 소프트웨어의 결함수정이 어렵기 때문에 신뢰도 성장이 어렵다. 따라서 발행 전 기간에 걸쳐서 다음과 같은 관계가 성립한다.

$$R_u(x|T) < R_d(x|T) \tag{4.7}$$

V. 신뢰도 개념의 그래프 해석

일반적으로 소프트웨어를 테스트하여 결함을 제거하게 되면 결함발견 빈도수가 줄어들기 때문에 고장강도함수가 지수함수적으로 감소하는 것으로 되어있다. 그러나, 개발 초기에는 소프트웨어에 익숙하지 못한 단계로서 불안정하여 그 결함들이 감소하기 전에 결함발생빈도가 오히려 증가한 후 어느 시점부터 다시 지수함수적으로 감소되는 경우도 있다. 식(4.3)을 그래프로 표현하기 위해 다음과 같이 다시 쓸 수 있다.

$$R_u(x|t) = \exp(-S_{ABC'D}) \tag{5.1}$$

여기서, $S_{ABC'D}$는 그림 3의 사각형 ABC'D의 면적과 같다. 이와 마찬가지로 전용소프트웨어에 대해서도 식(4.6)을 다음과 같이 나타낼 수 있다.

$$R_d(x|T) = \exp(-S_{ABCD}) \tag{5.2}$$

여기서, S_{ABCD}는 그림 3의 (편의상) 곡선 사다리꼴 ABCD의 넓이를 나타낸다.

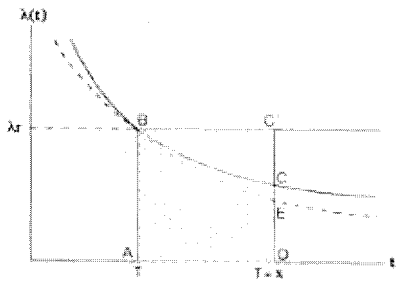

그림 3. $\lambda(t)$가 $t \geq 0$에서 단조감소하는 경우
Fig. 3 monotone decrease

이 그림에서 명백히 알 수 있는 바와 같이 $S_{ABC'D} > S_{ABCD}$ 이므로 $R_u < R_d$이다. 두 가지 신뢰도의 경우를 일반적으로 다음과 같이 나타낼 수 있다.

$$R(x|t) = \exp(-S) \tag{5.3}$$

여기서 S는 고장강도와 경과시간 사이의 면적이다. 따라서, 일반적으로 운영단계에서는 순간고장강도가 범용소프트웨어인 경우, 일정하여 운영신뢰도를 성장시키지 못하며, 시간 경과에 따라 신뢰도가 감소한다. 전용소프트웨어의 경우는 범용소프트웨어의 경에 비하여 신뢰도 저하가 사소한 편이다. 그러므로 소프트웨어의 운영 전 기간에 걸쳐서 전용소프트웨어의 신뢰도가 범용소프트웨어의 신뢰도보다 항상 높다.

VI. 실례

실제 계산 예로서 그림 3을 고려하여 T=800, x=100, a=33.99, b=0.006인 경우에 대해서 계산해보기로 한다.

$$\begin{aligned}
\lambda(t) &= 33.99 \times 0.006\exp[-0.006t] \\
&= 0.2039\exp[-0.006t], \\
S_{ABCD} &= \int_{800}^{900} 0.2039\exp[-0.006t]dt = 0.1262, \\
S_{ABC'D} &= 0.2039\exp[-0.006 \times 800] \times 100 \\
&= 0.1678, \\
\therefore\ R_d(100|800) &= \exp[-0.1262] = 0.8814, \\
R_u(100|800) &= \exp[-0.1678] = 0.8455
\end{aligned}$$

로서 $R_d(100|800) > R_u(100|800)$이다.

위의 예는 테스트 단계 신뢰도와 운영단계 신뢰도의 차이를 비교해보기 위한 단순한 예에 불과하지만 이는 어느 경우에나 유사한 결과를 얻게 된다.

따라서, 일반적으로 운영단계에서는 순간고장강도가 일정하여 운영신뢰도를 성장시키지 못하며, 시간 경과에 따라 신뢰도가 감소하므로 테스트 단계의 신뢰도보다 저하된다.

Ⅶ. 결론

소프트웨어의 테스트 단계에서는 테스트 공정이 NHPP를 따르는 것으로 연구되어 왔으므로 이 공정과 평균치 함수를 이용하여 어느 기간 동안 결함이 발견되지 않을 확률로 신뢰도를 정의한다. 이는 평균치 함수의 시간적 차이를 지수함수의 지수로 취한 형태를 하고 있다. 이 단계에서는 결함을 발견하기 위해 테스트를 하고 결함 발견 즉시 수정하여 동일한 결함이 다시 발생되지 않으므로 신뢰도가 크게 성장된다.

반면, 운영단계에서는 일반적으로 불특정 다수의 사용자가 사용하다가 발견되는 결함에 의해 신뢰도가 영향을 받으므로, 경우에 따라서 사용자가 사용중인 소프트웨어의 결함을 발견하여 이를 개발자에게 알려서 수정하도록 하고 수정된 것을 일반인에게 다시 반영하는 것이 현실적으로 어렵다. 따라서, 운영단계에서는 그것이 범용소프트웨어일 경우 신뢰도의 성장이 가능하지 않으므로 테스트 단계의 신뢰도와 다르며, 그 신뢰도가 훨씬 저하된다. 운영단계의 신뢰도는 신뢰도 함수의 지수부분이 평균치 함수의 시간적 차이가 아니라 일정한 고장강도에 경과시간을 곱한 형태를 취한다. 그러나 발행된 소프트웨어가 특정소비자의 요건에 맞도록 제작된 전용소프트웨어라면 신뢰도에 있어서 범용소프트웨어의 경우와 차이가 있으며, 범용소프트웨어의 경우보다 신뢰도가 높다는 것이 밝혀졌다. 결국 운영단계의 소프트웨어에 있어서 범용소프트웨어보다는 전용소프트웨어의 신뢰도가 항상 높다.

테스트 단계의 신뢰도와 운영 단계의 신뢰도를 실제 예를 가지고 분석해본 결과 테스트 단계의 신뢰도가 운영 단계의 신뢰도보다 높은 것으로 분석되었다. 이는 어느 시점에서도 동일하게 적용된다.

참고문헌

[1] C. V. Ramamoorthy, F. B. Bastani, "Software reliability - Status and perspectives", IEEE Trans. on Software Eng., vol. SE-8, pp354-371, 1982 Aug.

[2] J. D. Musa, A. Iannino, K. Okumoto, "Software Reliability : Measurement, Prediction, Application", pp230-238, 1987 Mar.

[3] S. Yamada, H. Ohtera, H. Narihisa, "Software reliability growth models with testing- efforts", IEEE Trans. Reliability, vol. R-35, pp19-23, 1986 Apr.

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

[5] K. Okumoto, A. L. Goel, "Optimum release time for software systems based on reliability and cost criteria", J. System software, vol. 1, pp315-318, 1980.

[6] S. Yamada, S. Osaki, "Cost-reliability optimal release policies for software systems", IEEE Trans. on Reliability, vol. R-34, pp422-424, 1985 Dec.

[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, pp646-651, 1996 Dec.

저자소개

최규식(Gyu-Shik Che)

1973년 서울대학교 공과대학 전기공학과(공학사)
1983년 뉴욕공과대학 전기공학과(공학석사)
1993년 명지대학교 전기공학과(공학박사)
1978년~1993년 한국전력기술 중앙연구소 책임연구원
1993년 ~ 현재 건양대학교 의공학과 교수
※관심분야 : 데이터통신, 무선랜 분야