불완전디버깅이 주문형 개발소프트웨어의 인도시기에 미치는 영향 연구

최 규 식*

A Study on the Imperfect Debugging Effect on Release Time of Dedicated Develping Software

Gyu Shik Che*

Abstract

The software reliability growth model(SRGM) has been developed in order to estimate such reliability measures as remaining fault number, failure rate and reliability for the developing stage software. Almost of them assumed that the faults detected during testing were evetually removed. Namely, they have studied SRGM based on the assumption that the faults detected during testing were perfectly removed. The fault removing efficiency, however, is imperfect and it is widely known as so in general. It is very difficult to remove detected fault perfectly because the fault detecting is not easy and new error may be introduced during debugging and correcting. Therefore, the fault detecting efficiency may influence the SRGM or cost of developing software. It is a very useful measure for the developing software, much helpful for the developer to evaluate the debugging efficiency, and, moreover, help to additional workloads necessary. Therefore, it is very important to evaluate the effect of imperfect dubugging in point of SRGM and cost, and may influence the optimal release time and operational budget.

I extent and study the generally used reliability and cost models to the imperfect debugging range in this paper.

Keywords : SRGM, Mean Value Function, Fault Detecting Rate, Fault Introduction Possibility, Target Reliability, Optimal Release Time

논문접수일 : 2004년 7월 8일 논문게재확정일 : 2004년 10월 20일

* 건양대학교 정보통신학과 교수

1. 서 론

소프트웨어의 신뢰도를 특성화하기 위해서는 정량적인 측정과 관리가 필수적이며, 특히, 결함데이타 분석에 근거하여 테스트 단계에서 소프트웨어의 신뢰도를 산출하는 것이 매우 중요하다. 그동안 테스트 기간중의 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)[Ramamoorthy & Batani, 1982]이라 한다. 이러한 모델들을 이용하여 평균 초기결함의 수, 평균 고장간 시간 간격, 임의의 테스트 시간에 소프트웨어 내의 평균 잔여 결함수, 소프트웨어 신뢰도 함수와 같은 소프트웨어 신뢰도 척도를 산출할 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[1987]은 기존 소프트웨어 신뢰도 성장모델을 분류하는 하나의 안을 개발하였다. Yamada 등[1986]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, CPU시간, 실행테스트 케이스 등등에 의해서 측정된다.

그동안 많은 연구가와 참여자들에 의해서 보편적으로 널리 연구되고 사용되는 SRGM(software reliability growth model)의 부류는 NHPP (nonhomogeneous Poisson process) 모델이며, 이러한 부류의 모델은 실제로 많은 장점을 가지고 있어서 그간 많은 관심을 끌어왔다. 이러한 모델들은 테스트 기간 중에 검출되는 결함들이 완벽하게 제거 및 수정되는 것을 가정한 바탕 위에서 수립되었다.

그러나, 실제로 소프트웨어 결함 디버깅은 매우 복잡한 공정이다. 보통은 테스트자가 소프트웨어 요건과 벗어난 것을 발견했을 때 수정요구를 한다. 검출보고를 받은 해당 분과위원들은 이 요구를 해당 개발자에게 할당한다. 개발자는 소프트웨어 결함을 연구한 후에 그것을 교정하기 위한 코드변경을 제출한다. 변경된 코드는 보고된 문제를 해결하기 위해 다시 여러 가지 테스트(유니트 테스트, 집적 테스트, 시스템 테스트) 과정을 거쳐야 한다. 어떤 경우에는 이러한 테스트를 거치지 않을 수도 있고 때로는 이러한 테스트를 거쳤다 하더라도 그 테스트 환경이 고객의 환경과 동일하지 않아서 결함이 완벽하게 제거되지 않을 수도 있다[Xie & Yang, 2000 ; Zhang & Teng, 2003]. 그러한 과정중에 새로운 결함이 도입될 수도 있다[Zhang & Teng, 2003]. 소프트웨어 개발팀이 문제의 결함이 최종적으로 제거되기 전에 여러 번 보고된 소프트웨어 결함이라는 것을 발견하는 것도 흔한 일이다. 어떤 결함들은 고객이 현장에서 사용할 때만 나타나는 것도 있다. 그러므로, 결함 제거 효율은 소프트웨어 신뢰도 계산에서 중요한 인자이고 소프트웨어 사업관리에도 중요하다.

비록 그동안 약간의 소프트웨어 신뢰도 연구에서 불완전 디버깅 현상에 대해서 언급을 했지만 그들 대부분은 기존의 결함을 제거하는 중에 새로운 결함이 도입될 가능성에 대해서만 고려를 하였다. 그러나, 불완전 디버깅은 검출된 결함이 불완전 제거가 된 것을 의미한다. 고엘과 오꾸모또[1979]는 그들의 마코프모델에서 유사한 고찰을 하고 있다. 그들은 고장 후에 잔여결함이 확률 q로 동일하게 남아 있으며, 확률 p로서 현재의 값보다 낮아진다는 것을 가정하였다. 이는 결함제거가 항상 100%는 아니라는 것이

다. 크레머[1983]는 불완전 결함 제거 확률(사공정)과 결함도입(생공정) 두 개를 모두 고려하여 생사공정을 소프트웨어 신뢰도 모델링에 적용하였다. 그동안의 논문들에서는 테스트중에 결함도입 가능성만을 고려했지만 최근의 논문들에 의하면 불완전디버깅[Xie & Yang, 2000], 결함제거효율[Zhang & Teng, 2003] 등을 고려한 연구논문들이 발표되고 있다.

본 논문에서는 결함 제거 효율과 결함도입을 소프트웨어신뢰도 성장모델에 연합시키는 방법론을 제안한다. 2장에서는 결함 제거 효율과 결함도입 확률을 고려한 목표신뢰도 산출방법을 수립하고, 3장에서는 제안된 모델을 연구하여 소프트웨어의수정비용이 최저로 되는 시간을 구하는 알고리즘을 개발하고 그 비용을 산출하는 방법을 연구한다. 4장에서는 기존의 몇 개 실례에서 수집된 데이터에 의해서 각각의 경우에 대한 목표신뢰도에 이르는 시간, 최저비용에 이르는 시간을 구하고 그 때의 신뢰도 및 비용을 구하였다. 5장에서는 결론을 요약하였다.

2. 결함 제거 효율과 결함 도입을 고려한 소프트웨어 신뢰도 모델

소프트웨어 신뢰도는 규정된 환경 하에서 주어진 시간에 소프트웨어를 결함 없이 운영할 수 있는 확률인 것으로 정의하며, 다음과 같이 조건확률로 표현할 수 있다.

테스트공정이 NHPP를 따른다면 NHPP의 표준 이론으로부터 평균치 함수를

$$m(t) = a(1-e^{-bt}) \tag{2.1}$$

로 정의할 때[8] 임의의 $t \geq 0$과 $x > 0$에서

$$\Pr\{N(t+x)-N(t)=k\} = \frac{[m(t+x)-m(t)]^k}{k!}\exp\{-[m(t+x)-m(t)]\} \tag{2.2}$$

이므로, 소프트웨어의 신뢰도는 다음과 같이 표현할 수 있다.

$$\begin{aligned} R(x|t) &\equiv \Pr\{N(t+x)-N(t)=0\} \\ &= \exp\{-[m(t+x)-m(t)]\} \\ &= \exp[-a(1-e^{-bx})e^{-bt}] \\ &= \exp[-m(x)e^{-bt}] \end{aligned} \tag{2.3}$$

즉, 어느 시각 t부터 $(t+x)$ 시각까지 새로운 결함이 발견되지 않을 확률을 신뢰도로 정의하며, 이는 평균치 함수의 차이를 지수함수의 지수로 취한 형태를 하고 있다.

여기서는 결함 제거 효율과 결함 도입비를 NHPP의 평균치함수에 포함시킨다. 결함 제거 효율은 검토자가 검토, 검사, 테스트에 의해서 제거하는 결함의 비로 정의한다. 본 항에서는 제안된 모델의 미분방정식에 대한 명시적인 해법도 제공한다. 결함 제거 효율과 결함 도입 현상을 포함하는 평균치함수는 아래와 같은 시스템의 미분방정식을 풀어서 구한다.[Zhang & Teng, 2003]

$$\frac{dm(t)}{dt} = b(t)[a(t)-p\cdot m(t)] \tag{2.4}$$

$$\frac{da(t)}{dt} = \beta(t)\frac{dm(t)}{dt} \tag{2.5}$$

여기서, $a(t)$는 소프트웨어 초기결함 기대치의 누계 및 시각 t에서 도입되는 결함의 총기대치 합이며, p는 결함 제거 효율, β는 시각 t에서 새로운 결함이 도입될 확률이다. 개발공정을 통하여 결함의 p%가 완벽하게 제거될 수 있다는 것을 의미한다. 그러므로, (2.4)에서 $m(t)$는 시각 t에서 검출되는 결함의 기대치이며, 그래서 $pm(t)$는 성공적으로 제거하는 결함의 기대치를 명시적으로 표시한다. 기존의 모델들은 p 값이 보통 1(100%)인 것으로 가정하였다.

또한, 대부분의 기존 NHPP 모델은 결함 고

장비가 잔여 결함의 총수에 비례하는 것으로 가정하였다. 소프트웨어 시스템의 고장율은 어떠한 시간에 있어서도 잔여 결함의 수와 결함검출비(결함의 평균고장율로 나타냄)의 함수이다. 잔여 결함의 기대수는 아래와 같이 쓴다.

$$x(t) = a(t) - p \cdot m(t) \tag{2.6}$$

p=1일 때는 제안된 모델이 기존의 NHPP 모델로 단순화된다.

따라서, 결함제거 효율과 결함도입을 고려한 잔여결함의 기대치는

$$x(t) = a \cdot e^{-\int_0^t [p-\beta(\tau)]\, b(\tau)\, d\tau} \tag{2.7}$$

이고 고장율 함수는 아래와 같이 표현된다.

$$\lambda(t) = m'(t) = b(t)[a(t) - pm(t)] = b(t)x(t) \tag{2.8}$$

그러므로, 평균치함수를 아래와 같이 얻는다.

$$\begin{aligned} m(t) &= \int_0^t x(u)\, b(u)\, du \\ &= a\int_0^t b(u) e^{-\int_0^u (p-\beta(\tau))b(\tau)\, d\tau} du \end{aligned} \tag{2.9}$$

식 (2.8)의 결과를 이용하면 결함 내용 비함수에 대한 해를 얻을 수 있다. 결함 내용 비함수는 아래와 같다.

$$a(t) = a\left[1 + \int_0^t \beta(u)\, b(u) \cdot e^{-\int_0^u [p-\beta(\tau)]b(\tau)\, d\tau} du\right] \tag{2.10}$$

그러므로 NHPP에 근거한 신뢰도 함수는

$$R(x|t) = \exp\{-[m(t+x) - m(t)]\} \tag{2.11}$$

이며, 여기서, $m(t)$는 식 (2.9)에 표시한 바와 같다.

테스트노력이 일정하다고 가정할 때는

$b(t) = b$, $\beta(t) = \beta$라고 제안할 수 있으므로

$$\begin{aligned} m(t) &= a\int_0^t b(u) e^{-\int_0^u (p-\beta)b\, d\tau} du \\ &= \frac{a}{p-\beta}\left[1 - e^{-(p-\beta)bt}\right] \end{aligned} \tag{2.12}$$

이며, 이를 이용하여 신뢰도를 구하면

$$\begin{aligned} m(t+x) - m(t) &= \frac{a}{p-\beta}[1 - e^{-(p-\beta)b(t+x)}] \\ &\quad - \frac{a}{p-\beta}[1 - e^{-(p-\beta)bt}] \\ &= \frac{a}{p-\beta} e^{-(p-\beta)bt}[1 - e^{-(p-\beta)bx}] \\ &= m(x) e^{-(p-\beta)bt} \text{이므로,} \end{aligned}$$

$$\begin{aligned} \therefore R(x|t) &= \exp[-m(x) e^{-(p-\beta)bt}] \\ &= \exp\left\{-\frac{a}{p-\beta}[1 - e^{-(p-\beta)bx}] e^{-(p-\beta)bt}\right\} \end{aligned} \tag{2.13a}$$

이고, 원하는 목표신뢰도를 R_o라 하면

$$R_o = \exp[-m(x) e^{-(p-\beta)bT}] \tag{2.14}$$

그런데, $R(x|0) = \exp[-m(x)]$

$= \exp\left\{-\frac{a}{p-\beta}[1 - e^{-(p-\beta)bx}]\right\}$로부터

$m(x) = \ln\frac{1}{R(x|0)}$를 대입하여 정리하면

$$T_1 = \frac{1}{(p-\beta)b} \ln \frac{\ln\frac{1}{R(x|0)}}{\ln\frac{1}{R_o}} \tag{2.15a}$$

가 된다. 여기서, $T^* = T_1$은 테스트 후 목표신뢰도를 만족하여 고객에게 인도해도 좋은 시간을 말한다.

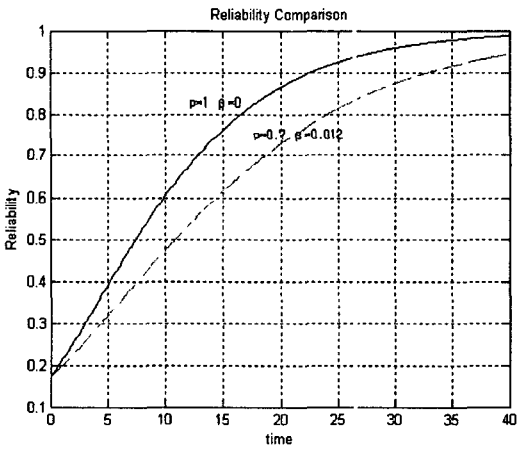
참고로 기존의 문헌에서처럼 p=1, $\beta = 0$인 경우에는

$$\begin{aligned} R(x|t) &= \exp[-m(x) e^{-bt}] \\ &= \exp\{-a[1 - e^{-bx}] e^{-bt}\} \end{aligned} \tag{2.13b}$$

$$T_1 = \frac{1}{b} \ln \frac{\ln \frac{1}{R(x|0)}}{\ln \frac{1}{R_o}} \qquad (2.15b)$$

와 같이 표현된다.

<그림 1>은 참고문헌[Xie & Yang, 2000 ; Zhang & Teng, 2003]에 의해 제시된 $a = 142$, $b = 0.1246$, $x = 0.1$, $p = 0.7$, $\beta = 0.012$, $R_o = 0.9$, $T_{LC} = 500$인 경우에 대해서 결함 제거 확률과 결함 도입 확률을 고려한 경우와 그렇지 않은 경우의 신뢰도를 비교한 그래프이다. 실선은 결함 제거가 완전한 경우이고 점선은 불완전한 경우이다.



<그림 1> 신뢰도 함수 비교

이 그림에 의하면 결함제거 효율이 완전하여 결함발견 즉시 수정이 완벽하고 결함 수정중 새로운 결함이 도입될 가능성이 없는 경우에는 목표신뢰도 90%에 이르는 시간이 22.6이다. 한편, 결함제거 효율을 가정하여 그 값이 70%라 하고 따라서 결함 도입 확률을 1.2%로 가정한 경우에는 목표신뢰도에 이르는 시간이 32.8로 크게 증가한다. 따라서, 결함제거를 완벽에 가깝도록 한다면 목표신뢰도에 이르는 시간 곧 인도시간이 크게 단축될 수 있다는 것을 알 수 있다.

3. 소프트웨어의 수정 비용

소프트웨어를 주문 받아서 개발한 후, 발행 전 결함을 발견하기 위한 테스트를 수행하여, 신뢰도가 어느 목표치에 도달했을 때 발행하게 된다. 그리고, 발행 후 결함이 발견되면 이를 수정해야 하며, 그 각각의 과정에서 비용이 발생하게 된다. 이러한 비용을 소프트웨어의 수정비용이라 하며, 여기에는 다음과 같은 비용이 발생된다.

테스트 기간중의 결함 수정 비용은 검출된 결함 하나하나를 수정하는데 비용이 발생되므로, 테스트 기간중에 검출되는 총 결함의 수에 결함당 수정비용을 곱한 값이 된다[Musa et al., 1987].

$$c_1 m_1(T) = c_1 a(1 - e^{-b_1 T}) \qquad (3.1)$$

운영중에 검출되는 결함의 수정비용은 발행 후 수명이 끝나는 시점까지 발생되는 결함에 대해서 수정하는데 드는 비용이므로

$$c_2\{m_2(T_{LC}) - m_1(T)\} = c_2 a e^{-b_1 T}(1 - e^{-b_2(T_{LC} - T)}) \qquad (3.2)$$

와 같이 표현된다.

테스트 기간 중에 발생되는 비용은 단위시간당 테스트비용을 결함제거확률과 결함도입확률을 고려한 값으로 나누어 전 테스트기간의 시간을 곱한 값이 된다.

$$\frac{c_3}{1 - p + \beta} T \qquad (3.3)$$

상기와 같은 논리에 의해서 총 비용은 테스트 기간중의 결함 수정 비용, 운영기간중의 결함 수정 비용, 테스트기간중의 테스트 비용을 합한 값이 된다.

$$C(T,p,\beta)=c_1 m_1(T)+c_2\{m_2(T_{LC})-m_1(T)\}+\frac{c_3}{1-p+\beta}T$$
$$=-\frac{a(c_2-c_1)}{p-\beta}[1-e^{-(p-\beta)b_1T}]+\frac{ac_2}{p-\beta}[1-e^{-(p-\beta)b_2T_{LC}}]+\frac{c_3}{1-p+\beta}T \quad (3.4)$$

여기서, $m(t)$는 식 (2.12)에 제시한 바와 같으며, b_1, b_2는 각각 개발소프트웨어의 인도전후의 결함검출비이다. 최적 소프트웨어 발행시각은 전체 평균 소프트웨어 비용을 최소로 하는 테스트시간이다.

$C(T, p, \beta)$의 최적값을 구하기 위해 식 (3.4)를 T로 편미분하여 정리한다.

$$-a(c_2-c_1)b-1e^{-(p-\beta)b_1T}+\frac{c_3}{1-p+\beta}=0 \quad (3.5)$$

이 식을 만족시키는 $T>0$인 범위의 T값을 구하면 $C(T)$에 대한 최소값이 된다.

$$T_2=\frac{1}{(p-\beta)b_1}\ln\frac{ab_1(c_2-c_1)(1-p+\beta)}{c_3} \quad (3.6a)$$

여기서, $T^*=T_2$는 소프트웨어 인도 전후의 총 수정비용이 최소가 되는 시간이다.

참고로 기존의 문헌에서처럼 p=1, β=0인 경우에는

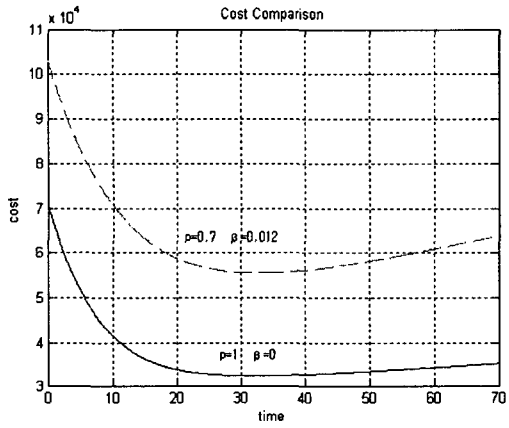
$$T_2=\frac{1}{b_1}\ln\frac{ab_1(c_2-c_1)}{c_3} \quad (3.6b)$$

와 같이 표현된다.

특별히 소프트웨어 발행 전후의 결함검출비율이 같을 경우에는 $b_1=b_2=b$가 되어 수정비용에 대한 해석이 단순해진다.

<그림 2>은 상기와 마찬가지로 참고문헌[4Xie & Yang, 2000 ; Zhang & Teng, 2003]에 의해 제시된 a=142, b = 0.1246, x = 0.1, p = 0.7, β = 0.012, c_1 = \$20, c_2= \$50, c_3 = \$10, T_{LC} = 500인 경우에 대해서 결함 제거 확률과 결함 도입 확률을 고려한 경우와 그렇지 않은 경우의 비용을 비교한 그래프이다. 실선은 결함 제거가 완전한 경우이고 점선은 불완전한 경우이다.

마찬가지로 이 그림에 의하면 결함제거 효율이 완전하여 결함발견 즉시 수정이 완벽하고 결함 수정중 새로운 결함이 도입될 가능성이 없는 경우에는 비용이 최저로 되는 인도시간은 31.9이고 이 때의 비용은 \$32,390이다. 그러나, 결함제거 효율을 가정하여 그 값이 70%라 하고 따라서 결함 도입 확률을 1.2%로 가정한 경우에는 비용을 최저로 하는 시간이 32.9로서 그 시간이 미미하게 증가하나, 비용은 \$55,514로서 결함수정이 완벽한 경우에 비하여 크게 증가한다.



〈그림 2〉 수정비용 비교

4. 적용 실례

참고문헌[Xie & Yang, 2000 ; Zhang & Teng, 2003]에서 인용한 데이터는 a = 142, b_1=b_2=b= 0.1246, R_o = 0.9, β = 0.012, C_1 = \$200, C_2 =\$500,

C_3 = \$100, T_{LC} = 500이며, 이를 근거로 하여 각 결함 제거 확률에 대한 결과를 <표 1>에 나타내었다. 결함도입확률이 0.012로서 일정하다고 가정하고 결함제거효율이 변화함에 따라 최적 발행 시각과 신뢰도, 수정비용이 어떻게 변하는지를 고찰해본다. 결함제거효율이 낮아짐에 따라 목표신뢰도에 이르는 시간은 증가비가 선형적이나 최저비용에 이르는 시간은 반비례의 관계를 유지하여 목표신뢰도 시간에 비하여 크게 증가하고 이는 검출확률이 낮아짐에 따라 이러한 현상은 급격해진다.

〈표 1〉 각각의 결함 제거 확률에 대한 영향

p	0.9	0.8	0.7	0.5
T_1	25.4	28.7	32.8	46.3
T_2	16.1	24.7	32.7	54.3
T^*	T_1 = 25.4	T_1 = 28.7	$T_1 = T_2$ = 32.8	T_2 = 54.4
$R(x\|T)$	0.90	0.90	0.90	0.94
$C(\$)$	57,548	52,808	55,514	71,296

참고로 동일 데이터에 의해서 p =1, β = 0인 경우 즉, 결함 검출시 결함제거가 완벽하고 새로운 결함이 도입되지 않는 경우에 대해서 기존의 방식대로 계산해보면 T_1= 22.6, T_2= 31.9로서 T^*= 31.9이고, $R(x|T)$ = 0.968, $C(T, p, \beta)$ = \$3,239로서 결함제거효율이 불완전한 것을 감안하고 또한 디버깅중에 결함도입가능성을 고려한 상기 고찰결과에 비하여 낙관적인 결과가 나온 것을 알 수있다.

5. 결 론

소프트웨어의 개발 단계에서 테스트 및 수정 단계를 거칠 때 실제로 결함 제거 효율은 통상 불완전하며, 이러한 사실은 그동안 널리 알려져 있다. 소프트웨어 결함은 그것을 찾아내는 것도 힘들지만 수정중에 새로운 결함이 도입될 수도 있기 때문에 검출된 결함이 완벽하게 제거되기는 어렵다. 따라서, 결함 제거 효율은 개발중인 소프트웨어의 신뢰도 성장이나 테스트 및 수정 비용에 영향을 크게 미친다. 이는 소프트웨어 개발의 모든 과정에서 매우 유용한 척도로서 개발자가 디버깅 효율을 평가하는데 크게 도움이 될 뿐더러, 추가로 소요되는 작업량을 예측할 수 있게 해준다. 그러므로 개발 소프트웨어의 SRGM과 비용면에서 불완전 디버깅의 영향을 연구하는 것은 매우 중요하다고 할 수 있으며, 이는 최적 인도 시각이나 운영 예산에도 영향을 줄 수 있다.

본 논문에서는 소프트웨어의 디버깅이 완전하지 않으며, 이 때문에 디버깅중 새로운 결함이 도입될 수도 있다는 제안 하에 보편적으로 사용되는 신뢰도 및 비용모델을 불완전 디버깅 범위로 확장하여 연구하였다. 그간의 기존 논문들을 참고하여 결함 제거 확률과 수정중 결함도입 확률을 고려한 이론을 전개 및 수립하였다. 이러한 알고리즘을 확인 및 실증하기 위해 그간 몇 개의 참고문헌에서 수집된 실제 데이타에 의해서 소프트웨어의 결함 제거 확률을 변화시켰을 때의 각각에 대해서 목표신뢰도에 이르는 시간, 최저 수정 비용에 이르는 시간을 계산하였다. 그리고, 최적시간을 산출하고 이때의 신뢰도 및 총비용을 계산하였다.

본 논문에서는 소프트웨어의 결함 제거가 불완전하다고 가정하고 수정중 결함 도입 확률까지를 고려하여 목표신뢰도와 최저 비용 시간을 구했으나, 여기서 제시된 알고리즘으로서 기존의 결함 제거 확률이 완벽한 경우의 비용을 산출할 수 없어서 이 때는 기존의 방법을 사용할 수밖에 없다는 단점이 있다. 이러한 문제는 추후 연구를 통하여 해결되어야 할 것으로 사료된다.

참 고 문 헌

[1] Goel, A.L., and Okumoto, Kazu, "A Markovian model for reliability and other performance measures of software systems", Proc. AFIPS Conf., June 1979, pp. 770-774.

[2] Goel, Amrit L., and Okumoto, Kazu, "Time-dependent error detection rate model for software reliability and other performance measure", IEEE Trans. on Reliability, Vol. R-28, No. 3, August 1979, pp. 206-211.

[3] Kremer, W., "Birth-death and bug counting", IEEE Trans. on Reliability, Vol. R-32, No. 1, 1983, pp. 37-47.

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

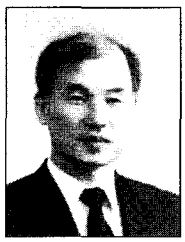
[5] Ramamoorthy, C.V., and Bastani, F.B., "Software reliability - Status and perspectives", IEEE Trans. on Software Eng., Vol. SE-8, August 1982, pp. 354-371.

[6] Xie, M., and Yang, B., "A study of the effect of imperfect debugging on software development cost", IEEE Trans. on Software Eng., Vol. 29, No. 5, May 2000, pp. 471-473.

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

[8] Zhang, X., and Teng, X., "considering fault removal efficiency in software reliability assessment", IEEE Trans. on Systems, man, and cybernetics, Vol. 33, No. 1, January 2003, pp. 114-120.

저자소개



최 규 식

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

◈ 이 논문은 2004년 7월 8일 접수하여 1차 수정을 거쳐 2004년 10월 20일 게재확정되었습니다.