소프트웨어의 결함 검출 효과에 관한 연구

김선일* · 최규식** · 조인준***

A study on the fault detection efficiency of software

Sun-il Kim* · Gyu-Shik Che** · In-June Jo***

요 약

소프트웨어의 신뢰도 모델링에서 테스트노력과 결함검출비를 동시에 고려하여 효과적인 파라미터 분석 기법을 이용하여 기존의 방법과 비교하고자 한다. 일반적으로, 소프트웨어 결함검출/제거 메카니즘은 이전의 검출/제거 결함과 테스트노력을 어떻게 활용하느냐에 달려 있다. 결함 제거 효율은 개발중인 소프트웨어의 신뢰도 성장이나 테스트 및 수정비용에 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 신뢰도와 비용면에서 불완전 디버깅의 영향을 연구하는 것은 매우 중요하다고 할 수 있으며, 이는 최적 인도 시각이나 운영 예산에도 영향을 줄 수 있다.

본 논문에서는 개발중인 소프트웨어를 대상으로 하여 디버깅이 완전하지 않으며, 따라서 결함검출비가 완벽하지 않다는 가정 하에 보편적으로 사용되는 신뢰도 모델을 대상으로 불완전 디버깅 범위로까지 소프트웨어의 신뢰도와 비용 문제를 확장하여 연구한다.

ABSTRACT

I compare my parameter estimation methodology with existing method, considering both of testing effort and fault detecting rate simultaneously in software reliability modeling. Generally speaking, fault detection/removal mechanism depends on how apply previous fault detection/removal and testing effort of S/W. The fault removal efficiency makes large influence to the reliability growth, testing and removal cost in developing stage S/W. This is very useful measure during all the developing stages and much helpful for the developer to estimate debugging efficiency, and furthermore, to anticipate additional working amount.

키워드

SRGM, mean value function, fault detection rate, NHPP, testing effort function, S/W

Ⅰ. 서 론

소프트웨어 신뢰도와 하드웨어 신뢰도는 모두 확률분포로 설명될 수 있으므로 둘이 유사하다. 그러나, 소프트웨어 결함은 눈에 보이는 하드웨어 결함에 비하여 실현, 검출, 교정이 쉽지 않다. 어떤 시스템이 되었든 그 신

* (주)핸디소프트
** 건양대학교
*** 배재대학교

접수일자 : 2008. 2. 29

뢰도는 시스템 설계의 정확성, 시스템의 설계를 구현하여 매핑시키는 정확성, 시스템 부품의 신뢰도에 의하여 결정된다. 소프트웨어는 적어도 물리적인 의미에서 열화되거나 낡아지지 않으므로 소프트웨어의 시스템이 하드웨어의 시스템과는 상당히 다르다. 그러므로, 소프트웨어와 하드웨어는 결함과 고장 패턴이 상이하며, 그 상이성은 중요하고도 근본적인 것이다. 소프트웨어 시스템은 일반적으로 고장율이 감소하는 경향을 가진다. 소프트웨어 신뢰도가 확률적이냐 결정론적이냐 하는 데에는 약간의 혼동이 있다. 그 대답은 우리가 소프트웨어의 신뢰도를 어떻게 보냐에 달려있다. 그러나, 소프트웨어는 그 주위 환경과 연관시켜 검토해야 한다. 소프트웨어 신뢰도를 측정하는 여러 가지 동일한 소프트웨어라도 상이한 테스트/운전 조건 하에서 다른 값을 나타내기 때문이다. ANSI의 정의에 의하면 "Software reliability is the probability of failure-free software operation for a specific period of time in a specified environment."이다. 따라서, 시스템 신뢰도를 결정함에 있어서 소프트웨어 신뢰도를 정확하게 모델링하고 그의 가능한 여러 가지 경향을 예측하는 것이 필수적이다. 소프트웨어 테스트에 의해서 발견되는 누적결함(또는 소프트웨어 고장간의 시간간격)과 소프트웨어 테스트 시간간격 사이의 관계를 짓는 모델들을 소프트웨어 신뢰도 성장모델(software reliability growth model : SRGM)[1]이라 한다. 소프트웨어 시스템을 신뢰성 높게 성취하려면 프로그램개발자나 테스트팀에 의해서 여러 소프트웨어 결함검출/제거 기법을 사용할 수 있어야 한다. 실제로 소프트웨어 결함 디버깅은 매우 복잡한 공정이다. 검출된 결함에 대하여 이를 해결하기 위해 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[2][3]. 한편, 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. 본 논문에서는 결함 제거 효율과 결함 개념을 소프트웨어 신뢰도 성장 모델에 도입시키는 방법을 제안한다. 이는 소프트웨어 개발시 디버깅이 완벽하다고 가정하여 소프트웨어의 신뢰도를 평가했던 그동안의 논문들과 다른 개념이다. 2장에서는 결함 제거 효율과 결함도입 확률을 고려한 신뢰도 모델에 대하여 목표 신뢰도 산출방법을 기술한다. 3장에서는 각 테스트 노력을 도입한 모델을 연구하여 소프트웨어의 수정 비용이 최저로 되는 시간을 구하는 알고리즘을 개발하고 그 비용을 산출하는 방법을 연구한다.

Ⅱ. 본 론

2.1. 소프트웨어 신뢰도 모델

2.1.1 평균치 함수 및 신뢰도

소프트웨어 신뢰도는 규정된 환경 하에서 주어진 시간에 소프트웨어를 결함 없이 운영할 수 있는 확률인 것으로 정의하며, 다음과 같이 조건확률로 표현할 수 있다. 테스트공정이 비동차포아송공정(Nonhomogeneous Poisson Process : NHPP)를 따르고 테스트 노력이 일정한 경우, NHPP의 표준 이론으로부터 평균치 함수를

$m(t) = a(1 - e^{-bt})$ (2.1) 로 정의할 때[4] 임의의 $t \geq 0$과 $x > 0$에서 $\Pr\{N(t+x) - N(t) = k\} =$

$$\frac{[m(t+x) - m(t)]^k}{k!} \exp\{-[m(t+x) - m(t)]\} \quad (2.2)$$

이므로, 소프트웨어의 신뢰도는 다음과 같이 표현할 수 있다.

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

$R(x|t)$는 시각 t에서 최종적으로 결함을 발견하여 수정한 후 x 유니트의 경과시간 동안 새로운 결함이 발견되지 않을 확률이다. 소프트웨어를 개발하여 결함테스트를 하면 할수록 결함을 발견하여 수정하는 빈도가 작아지므로 신뢰도가 성장되며, 결함 수정 후 소프트웨어의 운영시간(경과시간)이 길어지면 질수록 결함 발견 확

률이 높아지기 때문에 소프트웨어의 신뢰도는 낮아진다. 한편, 테스트 단계에서는 얼마나 오랜 시간동안 결함이 발견되지 않느냐가 중요한 것이 아니라, 현 단계에서 소프트웨어 내에서 발견되지 않고 잔존하는 결함의 수가 얼마나 되는가가 더 중요하다. 주어진 기간에 맞추어 소프트웨어 시스템을 개발할 때 시간, 자금, 인력과 같은 자원들이 소모된다. 특히, 소프트웨어 테스트에까지 이르는 자원들이 소프트웨어 신뢰도에 상당한 영향을 미친다. 소프트웨어 개발 자원 전체의 약 40-50%가 테스트 단계에서 소요된다[5]. [6]에서는 테스트 기간중의 테스트 노력과 소프트웨어 개발 노력 모두가 레일레이 곡선으로 설명될 수 있다는 것을 가정하여 소프트웨어 테스트에 쓰이는 테스트 노력의 양을 고려한 소프트웨어 신뢰도 성장 모델을 제시하였다. 그들은 레일레이 곡선의 대안으로서 지수곡선도 제안하였다. 그러나, 많은 경우의 소프트웨어 테스트에 있어서 테스트노력을 지수곡선이나 레일레이 곡선으로만 설명하는 것이 무리가 따른다. 이는 실제 테스트 노력 데이타가 다양한 형상을 나타내기 때문이다. 상기에서 언급한 곡선들은 일반적으로 웨이불곡선의 특수한 경우에 해당하는 것으로 하여 테스트 노력을 고찰하면 다음과 같다[5].

$$w(t) = N \cdot b \cdot m \cdot t^{m-1} \cdot \exp[-bt^m] \quad (2.4)$$

N : 소프트웨어 테스트에서 필요로 하는 테스트노력 소요 총량 b, m : 척도모수, 형상모수 m=1, m=2일 경우 각각 지수 및 레일레이 곡선 테스트 노력 함수를 얻을 수 있다. m>1이면 척도모수는

$$b = 1/\left[\left(\frac{m}{m-1}\right) t_{w\ \max}^{m}\right] \quad (2.5)$$

tw max : 테스트노력 w(t)의 양이 최대로 되는 시각 (2.4)의 적분형태

$$W(t) = N(1 - \exp[-bt^m]) \quad (2.6)$$

는 시각 (0,t]에서의 누적 테스트 노력량을 나타낸다. [6]이 제안한 웨이불형 테스트 노력 함수에 의하면 소프트웨어의 테스트 노력이 테스트 단계 전체에 걸쳐서 일정하다고 가정하면 비현실적이다. 그리고, 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 관련 문헌에 의하면 테스트 노력이 웨이불형 분포로 설명될 수 있고, 아래와 같은 3개의 케이스를 가진다는 것을 보여주고 있다.

1) 지수함수곡선 : (0, t]에서 소요되는 누적 테스트 노력은

$$W^*(t) = N[1 - e^{-\beta t}] \quad (2.7)$$

로서 웨이불 함수의 m=1인 경우에 해당되며, 신뢰도는

$$R(x|t) = \exp[-a \cdot e^{-rN(1-e^{-\beta t})} \cdot (1 - e^{-rN(1-e^{-\beta x})})] \quad (2.8)$$

로 표현된다. 2) 레일레이 곡선: 소요되는 누적 테스트 노력은

$$W(t) = N[1 - \exp^{-\frac{\beta}{2} \cdot t^2})] \quad (2.9)$$

로서 웨이불함수의 m=2인 경우에 해당된다.
신뢰도는

$$R(x|t) = \exp[-a \cdot e^{-rN(1-e^{-\frac{\beta}{2}t^2})} \cdot (1 - e^{-rN(1-e^{-\frac{\beta}{2}x^2})})] \quad (2.10)$$

로 표현된다.

3) 웨이불 곡선 : 소요되는 누적 테스트 노력은

$$W(t) = N[1 - e^{-\beta t^m})] \quad (2.11)$$

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

$$R(x|t) = \exp[-a \cdot e^{-rN(1-e^{-\beta t^m})} \cdot (1 - e^{-rN(1-e^{-\beta x^m})})] \quad (2.12)$$

로 표현된다.

4) 로지스틱 곡선

실제 테스트 노력 데이타가 여러 가지 소요 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 제시

된 것이 로지스틱형 테스트노력 함수이다. 이 함수는 실제 프로젝트 탐사에 의해서 보고된 바와 같이 매우 정확하다. (0, t]에서의 누적 테스트 노력 소요는

$$W(t) = \frac{N}{1 + A \cdot e^{-\alpha t}} \tag{2.13}$$

이고, 신뢰도는

$$R(x|t) = \exp[-a \cdot e^{-rN(\frac{1}{1+Ae^{-\alpha t}} - \frac{1}{1+A})} \cdot (1 - e^{-rN(\frac{1}{1+Ae^{-\alpha x}} - \frac{1}{1+A})})] \tag{2.14}$$

로 표현된다. 이와 같이 디버깅 중의 테스트 노력까지를 고려할 경우 소프트웨어를 시각 t에서 발행할 때 발견되는 누적 분포는 평균치 함수

$$m(t) = a(1 - e^{-\gamma W^*(t)}) \tag{2.15}$$

이므로, 잔여 분포

$$\bar{a} = a - a(1 - e^{-\gamma W^*(t)}) = a \cdot e^{-\gamma W^*(t)}$$

이다. 이것이 소프트웨어를 발행해서 운전하는 초기 결함수이므로

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

로서, $R(x|t) = \exp[-m(t+x) + m(t)]$

$$= \exp[-a \cdot e^{-\gamma W^*(t)}(1 - e^{-\gamma W^*(x)})] \tag{2.16}$$

이 된다. 따라서, 시각 t=T에서 발행된 소프트웨어의 신뢰도는 경과시간 x에 대해서 지수 함수적으로 감소한다. 감소비를 줄이기 위해서는 발행시각을 늦추거나 결함 검출비를 높여야 한다. 발행시각을 T로 하고 목표 신뢰도를 $R(x|T) = R_0$ 라 하면

$$W^*(t) = \frac{1}{r} \ln \frac{a(1 - e^{-rW^*(x)})}{\ln \frac{1}{R_0}} \tag{2.17}$$

여기서, $rW(t) = B(t)$ 라 놓으면

$$B^*(t) = \ln \frac{a(1 - e^{-B^*(x)})}{\ln \frac{1}{R_0}} \tag{2.18}$$

이고, 이를 이용하여 T*=T를 구한다. 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.

$$\frac{dm(t)}{dt} = b(t)[a(t) - p \cdot m(t)] \tag{2.19}$$

$$\frac{da(t)}{dt} = \beta(t)\frac{dm(t)}{dt} \tag{2.20}$$

여기서, a(t)는 소프트웨어 초기결함 기대치의 누계 및 시각 t에서 도입되는 결함의 총기대치 합이며, p는 결함 제거 효율, β(t)는 시각 t에서 새로운 결함이 도입될 확률이다. 일반적으로 p≫β이며, 결함의 p%가 완벽하게 제거될 수 있다는 것을 의미한다. 그러므로, (2.14)에서 m(t)는 시각 t에서 검출되는 결함의 기대치이며, 따라서, pm(t)는 성공적으로 제거하는 결함의 기대치를 명시적으로 표시한다. 기존의 모델들은 p값이 보통 1(100%)인 것으로 가정하였다. 고장율은 아래와 같이 표현된다.

$$\lambda(t) = m'(t) = b(t)[a(t) - pm(t)] = b(t)x(t) \tag{2.21}$$

그러므로, 평균치 함수는 아래와 같다.

$$m(t) = \int_0^t x(u)b(u)du$$

$$= a\int_0^t b(u)e^{-\int_0^u (p-\beta(\tau))b(\tau)d\tau}du \tag{2.22}$$

(2.22)에서 결함 도입 확률이 일정하다고 가정하여 $\beta(t) = \beta$라 하면

$$m(t) = a\int_0^t b(u)e^{-(p-\beta)\int_0^u b(\tau)d\tau}du \tag{2.23}$$

으로서, $\int_0^u b(\tau)d\tau = B(u) - B(0) = B^*(t)$ 라 하면

$$m(t) = a\int_0^t B'(u)e^{-(p-\beta)B(u)}e^{(p-\beta)B(0)}du$$

$$= a\,e^{(p-\beta)B(0)}\left[-\frac{1}{p-\beta}e^{-(p-\beta)B(u)}\right]_0^t$$

$$= \frac{a}{p-\beta}\{1 - e^{-(p-\beta)B^*(t)}\} \tag{2.24}$$

테스트 노력이 일정한 경우, 이 때는 곧 결함 검출비와 결함 도입비가 일정하다는 것을 의미하므로 $b(t)=b$, β

$(t)=\beta$로 제안한다.

$$m(t) = a\int_0^t b \cdot e^{-\int_0^u (p-\beta) b d\tau} du$$

$$= \frac{a}{p-\beta}[1-e^{-(p-\beta)bt}] \quad (2.25)$$

이며,

$$R(x|t) = \exp[-m(x)e^{-(p-\beta)bt}]$$

$$= \exp\left\{-\frac{a}{p-\beta}[1-e^{-(p-\beta)bx}]e^{-(p-\beta)bt}\right\} \quad (2.26a)$$

이고, 원하는 목표신뢰도를 R_o라 하면

$$R_o = \exp[-m(x)e^{-(p-\beta)bT}] \quad (2.27)$$

로 된다. 그런데,

$$R(x|0) = \exp[-m(x)]$$

$$= \exp\left\{-\frac{a}{p-\beta}[1-e^{-(p-\beta)bx}]\right\} \quad (2.28)$$

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

$$T_1 = \frac{1}{(p-\beta)b}\ln\frac{\ln\frac{1}{R(x|0)}}{\ln\frac{1}{R_o}} \quad (2.29a)$$

가 된다. 여기서, $T^*=T_1$은 테스트 후 목표신뢰도를 만족하여 고객에게 인도해도 좋은 시간을 말한다. 본 항에서 디버깅 확률과 결함도입 확률을 도입하여 상기의 공식 (2.15)-(2.18)을 다시 정리하면 다음과 같다.

- 평균치 함수

$$m(t) = \frac{a}{p-\beta}(1-e^{-B(t)}) \quad (2.30)$$

- 신뢰도

$$R(x|t) = \exp\left\{-\frac{a}{p-\beta}[1-e^{-rW(x)}]\cdot e^{-rW^*(t)}\right\}$$

$$= \exp\left\{-\frac{a}{p-\beta}[1-e^{-B(x)}]\cdot e^{-B^*(t)}\right\} \quad (2.31)$$

- 테스트 노력 함수

$$W^*(t) = \frac{1}{r}\ln\frac{a(1-e^{-rW^*(x)})}{(p-\beta)\ln\frac{1}{R_0}}$$

$$= \frac{1}{r}\ln\frac{a(1-e^{-B^*(x)})}{(p-\beta)\ln\frac{1}{R_0}} \quad (2.32)$$

Ⅲ. 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 수정 비용이라 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[6].

$c_1 m_1(T) = c_1 a(1-e^{-b_1 T})$ (3.1) 운영중에 검출되는 결함의 수정비용은 발행 후 수명이 끝나는 시점까지 발생되는 결함에 대해서 수정하는데 드는 비용이므로

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

와 같이 표현된다.

테스트 기간 중에 발생되는 비용은 단위시간당 테스트 비용을 결함 제거 확률과 결함 도입 확률을 고려한 값으로 나누어 전 테스트기간의 시간을 곱한 값이 된다. 즉 디버깅 확률이 클수록 테스트 기간중에 발생되는 테스트 비용이 증가되며, 디버깅 확률이 완벽하여 p=1인 경우에는 테스트 비용이 무한적으로 증가한다는 것이다. 이로 미루어 볼 때 이 식이 의미하는 것은 테스트 기간중에 훈련을 통하여 디버깅 확률을 목표 수준까지 높이려 노력할 때 발생되는 비용을 고려한 것으로 사료된다. 따라서, 본고에서는 결함 도입 확률 β까지를 고려하여 그 비용 계수를

$$\frac{c_3}{p-\beta} \quad (3.3)$$

로 제안한다. 이는 디버깅 확률이 높으면 높을수록 테스트가 수월해지며, 따라서 테스트 기간도 단축시키고 테스트 비용도 줄어들 것어 테스트 기간중의 테스트 비용이 감소한다는 개념이 타당성을 갖는다고 본 것이다. 상기와 같은 논리에 의해서 총 비용은 테스트 기간중의 결함 수정 비용, 운영 기간중의 결함 수정 비용, 테스트 기간중의 테스트 비용을 합한 값이 된다.

$$C(T, p, \beta) = c_1 m_1(T) + c_2\{m_2(T_{LC}) - m_1(T)\} + \frac{c_3}{p-\beta}\int_0^T w(\tau)d\tau$$

$$= -(c_2 - c_1)m_1(T) + c_2 m_2(T_{LC}) + \frac{c_3}{p-\beta}W(T) \quad (3.4)$$

이다. 여기서, $m(t)$는 평균치함수이며, b_1, b_2는 각각 개발 소프트웨어의 인도전후의 결함검출비이다. 최적 소프트웨어 발행시각은 전체 평균 소프트웨어 비용을 최소로 하는 테스트 시간이다.

그림 3.1은 상기와 마찬가지로 [2], [3]에 의해 제시된 a=142, b=0.1246, x=0.1, p=0.7, β=0.012, c_1=\$200, c_2=\$500, c_3=\$100, 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.

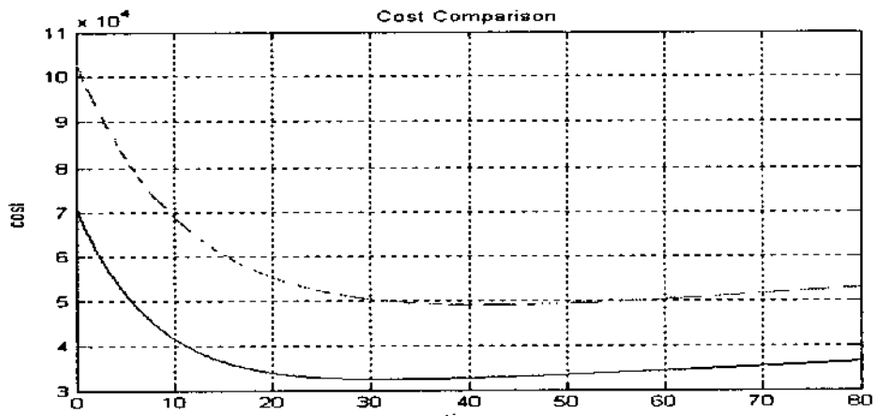


그림 3.1 수정비용 비교
Fig. 3.1 Repair cost comparison

Ⅳ. 결 론

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 이 때의 신뢰도 및 총비용을 계산하였다. 결과에 의하면 결함 제거 확률이 높을수록 최적 인도시기는 비용 최저점에 의하여 결정되나, 결함제거 확률이 낮아짐에 따라 목표신뢰도를 맞추는 시점으로 옮겨감을 알 수 있다. 이는 목표신뢰도를 예를 들어 0.95로 맞추면서 비용을 최저로 하기 위한 개발업체의 목적에 배치되는 경우로서, 개발업체는 테스트자의 품질을 높이기 위한 노력을 기울여야 함을 알 수 있다. 또한, 상기 표에 보듯이 결함제거 확률이 낮아질수록 인도 전후를 포함한 소프트웨어의 테스트 및 유지보수 비용이 크게 증가한다. 결함 제거 획률이 완벽에 가까운 경우에 비하여 그 확률이 0.5 정도로 크게 낮아지면 소프트웨어의 수정 비용 및 유지보수 비용이 두 배 가까이 증가하는 것을 알 수 있다. 현실적으로 결함을 완벽하게 제거하는 것은 불가능에 가까우며, 따라서, 운영경험에 의하여 결함검출 확률을 산출하여 이를 적용하는 것이 타당하다고 생각된다.

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.

참고문헌

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

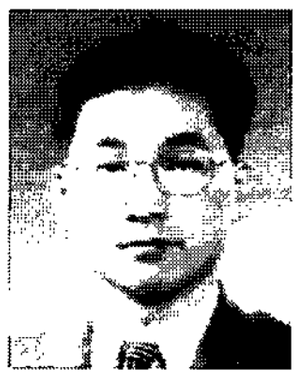
[2] Min Xie, Bo Yang, " A study of the effect of imperfect debugging on software development cost", IEEE Trans. on Software Eng., vol.29, no.5, pp471-473, 2003.5

[3] X. Zhang, X. Teng, H. Pham, " considering fault removal efficiency in software reliability assessment", IEEE Trans. on Systems, man, and cybernetics, vol.33, no.1, pp114-120, 2003.1

[4] Amrit L. Goel, Kazu Okumoto, " Time-dependent error detection rate model for software reliability and other performance measure", IEEE Trans. on Reliability, vol. R-28, no.3, pp206-211, 1979.8

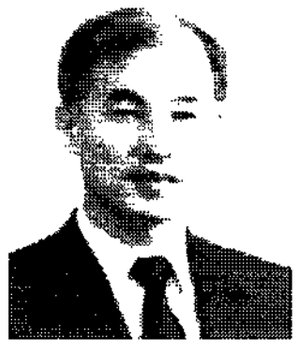
[5] S. Yamada, H. Ohtera, H. Narihisa, "Software reliability growth models with testing- efforts", IEEE Trans. on Reliability, vol. R-35, pp19-23, 1986 April

저자소개



김 선 일(Sun-Il Kim)

1995년 우석대학교 일어일문학과 (문학사)
2000년 중앙대학교 경영정보학과 (경영학석사)
2001년 ~ 현재 배재대학교 컴퓨터공학과 (박사수료)
1996년 ~ 현재 핸디소프트 이사대우
※관심분야 : BPM, 소프트웨어 품질



최 규 식(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

조 인 준(In-June Jo)

1982년 전남대학교 계산통계학과 (공학사)
1985년 전남대학교 전자계산학과 (공학석사)
1999년 아주대학교 컴퓨터공학과 (공학박사)
1983년 ~ 1994년 한국전자통신연구원 선임연구원
1994년 ~ 현재 배재대학교 컴퓨터공학과 교수
※관심분야 : 정보보호, 컴퓨터 네트워크, 전산조직응용