불완전 디버깅을 고려한 개발 소프트웨어의 최적 인도 시기 결정 방법에 관한 연구

論 文
54D-6-7

A Study on the Optimum Release Time Determination of Developing Software Considering Imperfect Debugging

崔 圭 植†
(Che Gyu Shik)

Abstract - The software reliability growth model(SRGM) has been developed in order to evaluate such measures as remaining fault number, fault rate, and reliability for the developing stage software. Most of the study literatures assumed that this detecting efficiency was perfect. However the actual fault detecting is generally imperfect, and widely known to many persons. It is not easy to develop and remove the fault existing in the software because the fault finding is difficult, and the exact solving method also not easy, and new fault may be introduced depending on the tester's capability. There, the fault removing efficiency influences the software reliability growth or developing cost of software. It is a very useful measure throughout the developing stage, much helpful for the developer to evauate the debugging efficiency, and evaluate additional workload. Hence, the study for the imperfect debugging is important in point of software reliability and cost.

This paper proposes that the fault debugging is imperfect and new fault may be introduced for the developing software during the developing stage.

Key Words : SRGM, 평균치 함수, 결함 검출비, 결함 도입 확률, 목표 신뢰도, 최적 인도 시간

1. 서 론

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.

많은 연구가와 참여자들에 의해서 보편적으로 널리 연구되고 사용되는 SRGM은 NHPP(nonhomogeneous Poisson process) 모델이며, 이러한 모델들은 테스트 기간 중에 검출되는 결함들이 완벽하게 제거 및 수정되는 것을 가정한 바탕 위에서 수립되었다.

그러나, 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[3]. 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[4]는 그들의 마코프모델에서 유사한 고찰을 하고 있다. 그들은 고장 후에 잔여 결함이 동일하게 남아 있다는 것을 고려하였으며, 현재의 값보다 낮아진다는 것을 가정하였다. 크레머[5]는 불완전 결함 제거 확률과 결함도입 확률 두 개를 모두 고려하여 소프트웨어 신뢰도 모델링에 적용하였다. 그동안의 논문들에서는 테스트 중에 결함도입 가능성만을 고려했지만 최근의 논문들에 의하면 불완전디버깅[2], 결함제거효율[3] 등을 고려한 연구논문들이 발표되고 있다.

테스트 기간 동안에는 소프트웨어의 신뢰도가 내재 결함을 검출 및 수정하는데 소요되는 개발자원의 양에 크게 의존한다. Musa 등[6]은 기존 소프트웨어 신뢰도 성장모델을 분류하는 안을 개발하였다. Yamada 등[7]은 달력 경과 테스트 시간, 테스트 노력량, 테스트 노력에 의해서 검출되는 소프트웨어 결함의 수 사이의 관계를 명시적으로 설명할 수 있는 간단하고도 새로운 모델을 제시하였다. 소프트웨어 신뢰도 모델링 분야에서는 소프트웨어의 개발 노력이 자주 전통적인 지수함수, 레일레이함수, 또는 웨이불 곡선으로 표현된다. 또, 실제 테스트/디버그 데이터 집합에 근거하여 실험을 수행하여 다양한 모델간의 예측 능력을 비교해본 결과, 초기 결함의 수를 산출하는 데는 로지스틱 테스트 노력 함수를 가진 SRGM이 이전의 접근법에 비하여 더 적합하다는 것을 보여주고 있다.

본 논문에서는 결함 제거 효율과 결함 개념을 소프트웨어

† 교신저자, 正 會 員 : 건양대학교 의공학과 교수
E-mail : che@konyang.ac.kr
接受日字 : 2004年 11月 15日
最終完了 : 2005年 5月 12日

신뢰도 성장 모델에 도입시키는 방법을 제안한다. 이는 소프트웨어 개발시 디버깅이 완벽하다고 가정하여 소프트웨어의 신뢰도를 평가했던 그동안의 논문들과 다른 개념이다. 2항에서는 결함 제거 효율과 결함도입 확률을 고려한 신뢰도 모델에 대하여 목표 신뢰도 산출방법을 기술한다. 3항에서는 각 테스트 노력을 도입한 모델을 연구하여 소프트웨어의 수정 비용이 최저로 되는 시간을 구하는 알고리즘을 개발하고 그 비용을 산출하는 방법을 연구한다. 4항에서는 테스트 노력이 일정한 경우에 대해서 기존의 몇 개 실례에서 수집된 데이터에 의해서 각각의 경우에 대한 목표 신뢰도에 이르는 시간, 최저 비용에 이르는 시간을 구하고 그 때의 신뢰도 및 비용을 구한다. 5항에서는 이와 같이 검토한 내용에 대해서 결론을 맺는다.

2. 소프트웨어 신뢰도 모델

2.1 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 NHPP를 따르고 테스트 노력이 일정한 경우, 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.

$$R(x|t)\equiv\Pr\{N(t+x)-N(t)=0\}=\exp[-m(x)e^{-bt}] \tag{2.3}$$

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

소프트웨어를 개발하여 결함테스트를 하면 할수록 결함을 발견하여 수정하는 빈도가 작아지므로 신뢰도가 성장되며, 결함 수정 후 경과시간이 길어지면 길수록 결함 발견 확률이 높아지기 때문에 소프트웨어의 신뢰도는 낮아진다.

한편, 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. Yamada 등[7]은 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[9].

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

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. $m>1$이면 척도모수는

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

$t_{w\ max}$: 테스트노력 $w(t)$의 양이 최대로 되는 시각

(2.4)의 적분형태

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

는 시각 (0,t]에서의 누적 테스트 노력량을 나타낸다.

가 된다.

Yamada[9]가 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

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

로서 웨이불 함수의 m=1인 경우에 해당되며, 신뢰도는

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

로 표현된다.

2) 레일레이 곡선 : 소요되는 누적 테스트 노력은

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

로서 웨이불함수의 m=2인 경우에 해당된다. 신뢰도는

$$R(x|t)=\exp[-ae^{-rN(1-e^{-\frac{\beta}{2}t^2})}\ (1-e^{-rN(1-e^{-\beta x^2/2})})] \tag{2.10}$$

로 표현된다.

3) 웨이불 곡선 : 소요되는 누적 테스트 노력은

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

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

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

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, 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 소요는

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

이고, 신뢰도는

$$R(x|t)=\exp[-ae^{-rN(\frac{1}{1+Ae^{-at}}-\frac{1}{1+A})}$$

$$(1-e^{-rN(\frac{1}{1+Ae^{-ax}}-\frac{1}{1+A})})] \quad (2.14)$$

로 표현된다.

초기점에서 웨이불형 테스트 노력 함수와 비교하여 로지스틱 테스트노력 함수인 경우 $W(0)\neq 0$ 이다. 이와 같이 디버깅 중의 테스트 노력까지를 고려할 경우 소프트웨어를 시각 t에서 발행할 때 발견되는 누적 분포는 평균치 함수

$$m(t)=a(1-e^{-rW(t)}) \quad (2.15)$$

이므로

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

$$=\exp[-a\cdot e^{-rW(t)}(1-e^{-rW(x)})] \quad (2.16)$$

이 된다.

발행시각을 T로 하고 목표 신뢰도를 $R(x|T)=R_0$ 라 하면

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

여기서, $rW(t)=B(t)$ 라 놓으면

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

이고, 이를 이용하여 $T^*=T$를 구한다.

본 논문에서는 결함 제거 효율과 결함 도입비를 NHPP의 평균치함수에 포함시키는 안을 제시한다. 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 참고문헌[3]에서 제시한 아래와 같은 시스템의 미분방정식을 풀어서 구한다.

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

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

여기서, $a(t)$는 소프트웨어 초기결함 기대치의 누계 및 시각 t에서 도입되는 결함의 총기대치 합이며, p는 결함 제거 효율, $\beta(t)$는 시각 t에서 새로운 결함이 도입될 확률이다. 일반적으로 $p\gg\beta$이며, 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 100%인 것으로 가정하였다.

고장율은 아래와 같이 표현된다.

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

그러므로, 평균치 함수는 아래와 같다.

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

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

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

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

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

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

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

테스트 노력이 일정한 경우, 이 때는 곧 결함 검출비와 결함 도입비가 일정하다는 것을 의미하므로

$b(t)=b$, $\beta(t)=\beta$로 제안한다.

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

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

이며, 이를 이용하여 신뢰도를 구하면

$$m(t+x)-m(t)=$$

$$\frac{a}{p-\beta}[1-e^{-(p-\beta)b(t+x)}]-\frac{a}{p-\beta}[1-e^{-(p-\beta)bt}]$$

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

$$=m(x)e^{-(p-\beta)bt}$$

이므로,

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

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

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

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

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

참고로 기존의 문헌에서처럼 p=1, β=0인 경우에는

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

$$=\exp\{-a[1-e^{-bx}]e^{-bt}\} \quad (2.26b)$$

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

와 같이 표현된다.

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

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

또는

$$B^*(t)=\ln\frac{a(1-e^{-B^*(x)})}{(p-\beta)\ln\frac{1}{R_0}} \quad (2.33)$$

이와 같은 이론을 일반적인 경우까지 확장하여 고려해보면 다음과 같다.

가. 테스트 노력이 일정한 경우

$b(t)=b$, $B(t)=bt$

- 평균치 함수

$$m(t)=\frac{a}{p-\beta}\{1-e^{-(p-\beta)bt}\} \quad (2.34)$$

- 신뢰도

$$R(x|t)=\exp\left\{-\frac{a}{p-\beta}[1-e^{-(p-\beta)bx}]e^{-(p-\beta)bt}\right\} \quad (2.35)$$

- 발행시각

$$T_1=T^*=\frac{1}{(p-\beta)b}\ln\frac{a(1-e^{-(p-\beta)bx})}{(p-\beta)\ln\frac{1}{R_0}} \quad (2.36)$$

나. 지수함수인 경우

$$B(t)=rN(1-e^{-(p-\beta)bt})$$

- 평균치함수

$$m(t)=\frac{a}{p-\beta}\{1-e^{-(p-\beta)rN(1-e^{-(p-\beta)bt})}\} \quad (2.37)$$

- 신뢰도

$$R(x|t)=\exp[-\frac{a}{p-\beta}e^{-rN(1-e^{-(p-\beta)bt})}(1-e^{-rN(1-e^{-(p-\beta)bx})})] \quad (2.38)$$

- 발행시각

$$T_1=T^*=-\frac{1}{(p-\beta)b}\ln\left[1-\frac{1}{rN}\ln\frac{a(1-e^{rN(1-e^{-(p-\beta)bx})})}{\ln\frac{1}{R_0}}\right] \quad (2.39)$$

다. 레일레이함수인 경우

$$B(t)=rN(1-e^{-\frac{(p-\beta)b}{2}t^2})$$

- 평균치 함수

$$m(t)=\frac{a}{p-\beta}\left\{1-e^{-(p-\beta)rN(1-e^{-\frac{(p-\beta)b}{2}t^2})}\right\} \quad (2.40)$$

- 신뢰도

$$R(x|t)=\exp\left\{-\frac{a}{p-\beta}e^{-rN(1-e^{-\frac{(p-\beta)b}{2}t^2})}(1-e^{-rN(1-e^{-(p-\beta)\frac{b}{2}x^2})})\right\} \quad (2.41)$$

- 발행시각

$$T_1=T^*=\left\{-\frac{2}{(p-\beta)b}\ln[1-\frac{1}{rN}\ln\frac{a(1-e^{rN(1-e^{-bx})})}{\ln\frac{1}{R_0}}]\right\}^{1/2} \quad (2.42)$$

라. 로지스틱함수인 경우

$$B(t)=\frac{rN}{1+Ae^{-at}},\quad B(0)=\frac{rN}{1+A}$$

- 평균치함수

$$m(t)=\frac{a}{p-\beta}\left\{1-e^{-(p-\beta)rN(\frac{1}{1+Ae^{-at}}-\frac{1}{1+A})}\right\} \quad (2.43)$$

- 신뢰도

$$R(x|t)=\exp\left\{-\frac{a}{p-\beta}e^{-rN(\frac{1}{1+Ae^{-at}}-\frac{1}{1+A})}\cdot[1-e^{-rN(\frac{1}{1+Ae^{-ax}}-\frac{1}{1+A})}]\right\} \quad (2.44)$$

- 발행시각

$$T_1=T^*=-\frac{1}{a}\ln\frac{1}{A}\left[\frac{rN}{\frac{rN}{1+A}+\ln\frac{a(1-e^{-rN(\frac{1}{1+Ae^{-at}}-\frac{1}{1+A})})}{\ln\frac{1}{R_0}}}\right] \quad (2.45)$$

그림 2.1은 참고문헌[2][3]에 의해 제시된 a=142, b=0.1246, x=0.1, p=0.7, β=0.012, R_o=0.95, T_{LC} = 500인 경우에 대해서 테스트노력이 일정하다고 가정하여 결함 제거 확률과 결함 도입 확률을 고려한 경우와 그렇지 않은 경우의 신뢰도를 비교한 그래프이다. 실선은 결함 제거가 완전한 경우이고 점선은 불완전한 경우이다.

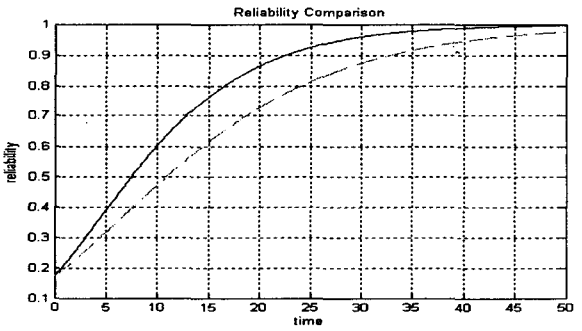

그림 2.1 신뢰도 함수 비교

Fig. 2.1 Reliability Comparison

이 그림에 의하면 결함제거 효율이 완전하여 결함발견 즉시 수정이 완벽하고 결함 수정중 새로운 결함이 도입될 가능성이 없는 경우에는 목표신뢰도 95%에 이르는 시간이 28.4이다. 한편, 결함제거 효율을 가정하여 그 값이 70%라 하고 따라서 결함 도입 확률을 1.2%로 가정한 경우에는 목표신뢰도에 이르는 시간이 41.3으로 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[6].

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

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

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

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.

참고문헌[2]에서는 이 식의 계수를 $c_3/(1-p)$라 제안하고 있으나 이는 문제가 있는 것으로 사료된다. 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

$$\frac{c_3}{p-\beta} \tag{3.3}$$

로 제안한다.

이는 디버깅 확률이 높으면 높을수록 테스트가 수월해지며, 따라서 테스트 기간도 단축시키고 테스트 비용도 줄어들 것이 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.

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

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

이다. 여기서, $m(t)$는 평균치함수이며, b_1, b_2는 각각 개발소프트웨어의 인도전후의 결함검출비이다. 최적 소프트웨어 발행시각은 전체 평균 소프트웨어 비용을 최소로 하는 테스트 시간이다. 각 테스트 노력에 대한 비용을 산출하는 공식과 최적 인도시각을 유도하면 아래와 같다.

가. 테스트 노력이 일정할 경우

- 비용함수

$$C(T,p,\beta) = -\frac{a(c_2-c_1)}{p-\beta}[1-e^{-(p-\beta)b_1 T}] + \frac{ac_2}{p-\beta}[1-e^{-(p-\beta)b_2 T_{LC}}] + \frac{c_3}{p-\beta}T \tag{3.5}$$

- 최적 인도 시각

$C(T, p, \beta)$의 최적값을 구하기 위해 식(3.5)를 T로 편미분하여 정리한다.

$$-a(c_2-c_1)b-1e^{-(p-\beta)b_1 T} + \frac{c_3}{p-\beta} = 0 \tag{3.6}$$

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

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

$$T_2 = \frac{1}{(p-\beta)b}\ln\frac{ab(c_2-c_1)(1-p+\beta)}{c_3} \tag{3.7a}$$

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

참고로 기존의 문헌에서처럼 p=1, β=0인 경우에는

$$T_2 = \frac{1}{b}\ln\frac{ab(c_2-c_1)}{c_3} \tag{3.7b}$$

와 같이 표현된다.

나. 지수함수인 경우

- 비용함수

$$C(T,p,\beta) = -\frac{a(c_2-c_1)}{p-\beta}\{1-e^{-(p-\beta)rN(1-e^{-(p-\beta)bT})}\} + \frac{ac_2}{p-\beta}\{1-e^{-(p-\beta)rN(1-e^{-(p-\beta)bT_{LC}})}\} + \frac{c_3 rN}{p-\beta}(1-e^{-(p-\beta)bT}) \tag{3.8}$$

- 최적 인도 시각

$$T_2 = -\frac{1}{(p-\beta)b}\cdot \ln\left[1-\frac{1}{(p-\beta)rN}\ln\frac{ab(c_2-c_1)(p-\beta)}{c_3}\right] \tag{3.9}$$

다. 레일레이 함수인 경우

- 비용함수

$$C(T,p,\beta) = -\frac{a(c_2-c_1)}{p-\beta}\{1-e^{-(p-\beta)rN(1-e^{-(p-\beta)b/2T^2})}\} + \frac{ac_2}{p-\beta}\{1-e^{-(p-\beta)rN(1-e^{-(p-\beta)b/2T^2_{LC}})}\} + \frac{c_3 rN}{p-\beta}(1-e^{-(p-\beta)b/2T^2}) \tag{3.10}$$

- 최적 인도 시각

$$T_2 = \left\{-\frac{2}{(p-\beta)b}\cdot \ln\left[1-\frac{1}{(p-\beta)rN}\ln\frac{ab(c_2-c_1)(p-\beta)}{c_3}\right]\right\}^{1/2} \tag{3.11}$$

라. 로지스틱 함수인 경우

- 비용함수

$$C(T,p,\beta)=-\frac{a(c_2-c_1)}{p-\beta}\left\{1-e^{-(p-\beta)rN\left(\frac{1}{1+Ae^{-\alpha T}}-\frac{1}{1+A}\right)}\right\}$$
$$+\frac{ac_2}{p-\beta}\left\{1-e^{-(p-\beta)rN\left(\frac{1}{1+Ae^{-\alpha T_{LC}}}-\frac{1}{1+A}\right)}\right\}$$
$$+\frac{c_3N}{p-\beta}\left\{\frac{1}{1+Ae^{-\alpha T}}-\frac{1}{1+A}\right\} \quad (3.12)$$

- 최적인도시각

$$T_2=-\frac{1}{\alpha}\ln\frac{1}{A}\left[\frac{1}{\frac{1}{1+A}+\frac{1}{(p-\beta)rN}\ln\frac{a(c_2-c_1)(p-\beta)}{c_3}}-1\right] \quad (3.13)$$

그림 3.1은 상기와 마찬가지로 참고문헌[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. 그러나, 결함제거 효율을 가정하여 그 값이 70%라 하고 따라서 결함 도입 확률을 1.2%로 가정한 경우에는 비용을 최저로 하는 시간이 42.0이 되고 비용은 \$49,075로서 결함수정이 완벽한 경우에 비하여 크게 증가된다.

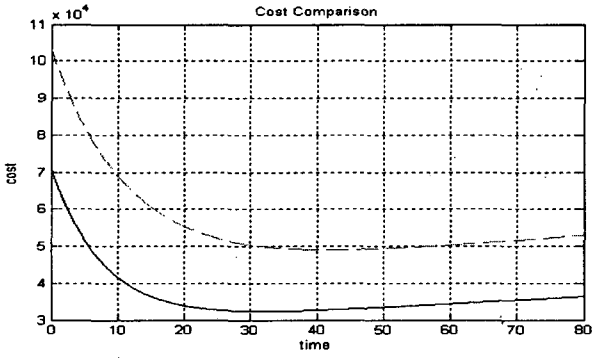

그림 3.1 수정비용 비교
Fig. 3.1 Fixing Cost Comparison

4. 적용 실례

참고문헌[2][3]의 현장 경험에 의해 분석하고 인용한 데이터는 a=142, b_1=b_2=b=0.1246, R_o=0.95, β=0.012, C_1=\$200, C_2=\$500, C_3=\$100, T_{LC} = 500이며, 여기서 a는 개발 시점부터 소프트웨어 내에 존재할 것으로 판단되는 초기 결함의 수, b_1, b_2, b는 소프트웨어 인도 전후 및 일반적인 결함검출비, R_o는 본 소프트웨어가 추구하는 목표신뢰도, β는 척도모수, C_1은 테스트 기간중에 검출되는 결함을 수정하는 단위비용, C_2는 운전 기간중에 검출되는 결함을 수정하는 비용, C_3는 단위시간당 테스트비용, T_{LC}는 고객에게 인도된 소프트웨어의 수명이다. 이를 근거로 하여 상기 3항에서 고찰한 알고리즘에 의하여 각 결함 제거 확률별 결과를 표 3.1에 나타내었다. 결함 도입 확률이 0.012로서 일정하다고 가정하고 결함 제거 효율이 변화함에 따라 최적 발행 시각과 신뢰도, 수정비용이 어떻게 변하는지를 고찰해본다. 결함 제거 효율이 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 4.1 각각의 결함 제거 확률에 대한 영향
Table 4.1 Effect for each Fault Removing Possibility

p	0.9	0.8	0.7	0.6	0.5
T_1	32.0	36.0	41.3	48.3	58.2
T_2	34.8	38.0	42.0	47.0	53.5
T^*	T_2 = 34.8	T_2 = 38.0	T_2 =42.0	T_1 = 48.3	T_1 = 58.2
$R(x\|T)$	0.963	0.960	0.953	0.950	0.950
$C(\$)$	36,921	42,159	49,075	58,618	70,123

또한, 이 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.

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=28.4, T_2=31.9로서 T^*=31.9이고, $R(x|T)$=0.968, $C(T, p, \beta)$=\$32,390으로서 결함제거효율이 불완전한 것을 감안하고 또한 디버깅중에 결함도입가능성을 고려한 상기 고찰결과에 비하여 낙관적인 결과가 나온 것을 알 수있다.

또한, 상기 표에 보듯이 결함제거 확률이 낮아질수록 인도 전후를 포함한 소프트웨어의 테스트 및 유지보수 비용이 크게 증가한다. 결함 제거 확률이 완벽에 가까운 경우에 비하여 그 확률이 0.5 정도로 크게 낮아지면 소프트웨어의 수정비용 및 유지보수 비용이 두 배 가까이 증가하는 것을 알 수 있다. 그러나, 이러한 비용의 증가가 개발업체의 개발전략에 어떠한 영향을 미칠 것인가에 대해서는 본 논문의 취지가 아니므로 여기서는 이에 대한 언급을 하기가 어렵다.

5. 결 론

소프트웨어의 개발 단계에서 테스트 및 수정단계를 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 수정비용에 영향을 크게 미친다. 이는 소프트웨어 개발의 모든 과정에서 매우 유용한 척도로서 개발자가 디버깅 효율을 평가하는데 크게 도움이 될 뿐더러, 추가

로 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 제안하에 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 확률이 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.

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.

감사의 글

이 논문은 2005학년도 건양대학교 학술연구비 지원에 의하여 이루어진 것임

참 고 문 헌

[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, "considering fault removal efficiency in software reliability assessment", IEEE Trans. on Systems, man, and cybernetics, vol.33, no.1, pp114-120, 2003.1

[4] A.L. Goel, K. Okumoto, "A Markovian model for reliability and other performance measures of software systems", Proc. AFIPS Conf., pp770-774, 1979.6

[5] W. Kremer, "Birth-death and bug counting", IEEE Trans. on Reliability, vol.R-32, no.1, pp37-47, 1983

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

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

[8] 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

[9] S. Yamada, J. Hishitani, S. Osaki, "Software-Reliability Growth with a Weibull Test-Effort : A Model & Application", IEEE Trans. on Reliability, vol.42, no.1, pp100-105, 1993.3

저 자 소 개

최 규 식 (崔 圭 植)

1976년 서울대학교 공과대학 전기과 졸업(학사)

1981년 뉴욕공과대학 전기과 졸업(제어공학 석사)

1993년 명지대학교 전기과 졸업(제어공학 박사)

1975년-1993년 OPC 중앙연구소, KOPEC 연구소 근무

1993- 현재 건양대학교 의공학과 교수

관심분야 : PLC, 소프트웨어 신뢰도