소프트웨어 결함 발견 및 제거 노력 기반 신뢰성 추정 모델

(Software Fault Detection and Removal Effort-based Reliability Estimation Model)

강 명 묵 † 구 태 완 †† 백 종 문 †††
(Myungmuk Kang) (Taewan Gu) (Jongmoon Baik)

요 약 최근 소프트웨어는 고성능의 많은 기능을 제공해야 하기 때문에 그 중요성이 증가할 뿐만 아니라 복잡도 또한 증가하고 있다. 그러므로 신뢰할 만한 소프트웨어를 개발하는 것이 중요한 이슈가 되고 있다. 신뢰할 만한 소프트웨어를 개발하기 위해서는 신뢰성을 초기단계에서부터 관리해야 할 필요성이 있지만 대부분 신뢰성 추정 모델의 경우 시스템 또는 운영 테스트 단계에서 주로 사용되고 있다. 신뢰성 높은 소프트웨어를 개발하기 위해서는 초기 테스트 단계에서부터 개별 유닛의 신뢰성을 관리할 필요성이 있기에 이 단계에서의 특징을 반영해야 한다. 그러나 초기 테스트 단계에서는 개발자와 테스터가 분리되는 것이 아니라 개발자가 테스트뿐만 아니라 디버그까지 함께 수행을 하게 된다. 그렇기 때문에 테스트 시간과 디버그 시간을 모두 고려하는 신뢰성 추정 모델이 필요하다. 본 논문에서는 초기 테스트 단계에서부터 개별 유닛의 신뢰성 관리를 지원하고자 새로운 신뢰성 모델을 제안하였다. 그리고 실제 산업에서 수집된 데이터를 이용하여 제안한 모델이 실제 데이터와 얼마나 일치하는지 그리고 기존 모델과 어떤 차이를 보이는지를 확인하기 위한 실험을 수행하였다.

키워드 : 소프트웨어 신뢰성, 소프트웨어 신뢰성 모델, 지수형 모델, 소프트웨어 신뢰성 도구(SRTpro)

Abstract Relative importance and complexity of recent software is getting increased because the software is needed to provide considerable amount of functions and high performance. Therefore, developing reliable software is importantly issued. In order to develop reliable software, it is necessary to manage software reliability at the early phases, but most reliability estimation models are used at system or operational test phases. In order to develop highly reliable software, it is necessary to manage software reliability at the early test phases based on characteristic of the phases that is developers and testers are not separated and developers perform test and debug activities together. Therefore, a new reliability estimation model considering test and debug time together is necessarily needed. In this paper, we propose a new reliability estimation model to manage reliability of individual units from the early test phases and in order to show how to fit the model to actual data and usefulness, we collected industrial data and used it for the experiment.

Key words : Software Reliability, Software Reliability Model, Exponential Model, Software Reliability Tool (SRTpro)

· 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2008-313-D009321)

† 학생회원 : KAIST 전산학과
mmkang@kaist.ac.kr
†† 정 회 원 : KAIST 정보전자연구소
gutaewan@kaist.ac.kr
††† 종신회원 : KAIST 전산학과 교수
jbaik@kaist.ac.kr

논문접수 : 2010년 4월 30일
심사완료 : 2010년 5월 26일

정보과학회논문지: 소프트웨어 및 응용 제37권 제7호(2010.7)

1. 서 론

현대의 시스템은 사용자의 수많은 요구사항을 반영하여 제품화되고 있을 뿐만 아니라 수많은 사용자의 요구사항으로 보다 많은 기능과 높은 성능까지 제공할 필요가 발생하였으며, 이에 따라 기능과 성능 제공을 위한 소프트웨어의 비중이 증가하고 복잡도도 함께 증가되고 있는 실정이다. 예를 들어, 통신기기의 경우 2002년도에 39.3%에서 2006년도에 54.3%로 증가하였으며 전투기의 경우에도 50%이상을 소프트웨어가 차지하고 있다. 특히, F-22의 경우에는 80%이상을 소프트웨어가 차지할 정도로 소프트웨어의 비중이 급격하게 증가되어 왔다[1]. 특히, 무기체계 시스템의 경우는 소프트웨어가 올바르게 작동하지 않을 경우 사람의 목숨 해칠 수 있는 위험한 상황에 처하게 될 정도로 그 중요성이 강조되고 있다. 이처럼 비중이 증가하고 중요한 임무 수행을 하는 소프트웨어는 충분히 신뢰할 수 있도록 개발되어야 할 필요성이 있으며, 지속적으로 신뢰성이 관리되고 개선되어야 한다. 이를 지원하기 위한 기술이 소프트웨어 신뢰성 공학(Software Reliability Engineering, SRE)이다.

소프트웨어 신뢰성은 소프트웨어 품질을 보증하기 위한 중요한 요소 중 하나로 그 정의는 내리는 사람 또는 기관에 따라 약간씩 다르다. IEEE 1633 표준의 정의에 따르면 소프트웨어 신뢰성은 "명시된 시간 동안 명시된 조건하에서 요구된 기능이 고장 없이 수행될 능력"을 의미한다[2]. 따라서 소프트웨어 신뢰성 측정 결과에 따라 소프트웨어의 배포시기를 결정할 수 있다. 이때 소프트웨어 신뢰성 측정을 위한 척도에는 소프트웨어에 내재된 고장수, 또는 고장 사이의 시간 등이 있으며, 이 결과 값에 따라 소프트웨어를 계속적으로 테스트할지 또는 배포를 할지 결정할 수 있게 된다.

또한 소프트웨어 신뢰성 공학은 이러한 소프트웨어 신뢰성을 정량적으로 평가, 개발, 그리고 유지 관리하기 위한 공학 기술에 초점을 두고 있다. 즉, 소프트웨어 신뢰성 공학은 "사용자의 요구사항에 따라 소프트웨어의 행위를 정량적으로 나타내기 위한 공학 기술"이다[3]. Michael Lyu[3]가 정의한 소프트웨어 신뢰성 공학 프로세스를 살펴보면 다음 네 가지 (1) 신뢰성 목표(Reliability Objective), (2) 운영 파일(Operational Profile), (3) 신뢰성 모델(Reliability Modeling), 그리고 (4) 신뢰성 검증(Reliability Validation) 이 중요하게 다루어지고 있다. 특히, 소프트웨어 신뢰성 공학 프로세스 중 신뢰성 모델은 신뢰성을 측정 및 평가하기 위한 것으로 본 논문에서도 새로운 모델을 통한 신뢰성 측정을 살펴볼 것이다.

소프트웨어 신뢰성을 측정하기 위해 1970년대 이후 현재 약 200개 이상의 모델들이 개발되었으며[4], 의료, 국방, 조선, 그리고 우주선 개발 등의 다양한 도메인에 적용되고 있다. 소프트웨어 신뢰성 모델은 초기 결함 수, 고장 간 시간, 남아 있는 고장 수 등의 소프트웨어 신뢰도 측도들을 추정하기 위한 수리적 모델로써 이러한 측도들을 추정하기 위해서 테스트 동안에 발생하는 고장, 결함 및 시간 등이 이용되고 있다. 소프트웨어 신뢰성 모델들은 소프트웨어 개발 생명 주기의 여러 단계에 따라서 분류될 수 있다[4-6]. 특히, 소프트웨어 신뢰성 추정 모델은 크게 지수형(Exponential) 모델과 S-Shaped 모델[7]로 분류되며 각 모델은 고장의 발견과정을 추정하는 모델과 제거과정을 추정하는 모델의 두 가지 형태의 모델을 가진다[8].

본 논문에서 다루는 소프트웨어 신뢰성 모델은 소프트웨어 테스트 단계에서 사용되는 모델로써 가장 기본이 되는 지수형 모델[4,9]을 기반으로 하고 있다. 특히, 3장에서 소개하고 있는 기존 모델에서 고려하지 못한 요소를 반영하여 지수형 모델을 변형한 새로운 모델을 제안한다. 우선 기본 지수형 모델 중 하나인 Schneidewind 모델[2,10]은 테스트 단계에서 발생하는 고장 데이터 및 시간을 입력으로 하여 예상되는 고장 수, 남은 고장 수, 그리고 목표 달성을 위한 시간 등을 추정할 수 있게 개발되었다. 두 번째로 Schneidewind는 이 모델을 통해서 결함이 제거되는 과정을 모델링하기도 했다[11]. 이 모델에서는 지연 시간(Delay Time)을 소개하였는데, 이는 기존 모델에서의 가정 중 하나인 "발견되는 고장은 바로 제거된다"는 가정에 대해 다른 의견을 제시하는 것이다. 마지막으로, 결함 발견 및 수정 과정을 통합한 모델[12]에서는 발견되는 결함이 수정되는데 생기는 지연 시간뿐만 아니라 결함이 발견되는 과정까지도 함께 고려하고 있다. 기존 모델에서는 결함을 발견하고 제거하는 과정이 분리되어 수행되기 때문에 결함을 찾는 과정과 수정하는 과정을 따로 분리하여 모델을 개발하고 적용하여왔다. 그러나 테스트 초기 단계에서나 규모가 작은 조직에서는 테스터와 개발자가 분리되지 못하고 개발자가 테스트 및 디버그까지 동시에 수행해야 하는 경우가 빈번히 발생하기 때문에 기존 모델들 사용에는 한계가 있어 이를 지원하기 위해 새로운 모델을 제안한다. 단, 본 논문에서 제안한 모델이 모든 소프트웨어 신뢰성 측정 및 평가에 적용 할 수 있음을 의미하지는 않는다. 즉, 테스터와 개발자를 분리하여 테스트와 디버깅을 반복적으로 수행하는 것이 일반적인 방법론임에는 틀림없으나, 단위 테스트나 통합 테스트와 같은 초기 테스트 단계에서는 현실적 어려움으로 인해 개발자와 테스터를 따로 분리하지 않고 개발자가 테스트와 디버깅을 함께 수행하고 있는 현실을 반영한 모델

을 제안하고자 하는 것이다.

본 논문의 구성은 다음과 같다. 2절에서는 새로운 모델을 제안하기 위한 기존 모델들의 문제를 제기하며 3절에서는 관련연구로 기존의 신뢰성 추정 모델에 대해 설명한다. 4절에서는 새로 제안하는 모델에 대해 구체적으로 설명하며 5절에서는 새로운 모델을 통한 실험 및 결과를 제시한다. 실험을 하는데 있어 기존의 신뢰성 도구 CASRE(Computer Aided Software Reliability Estimation)와 SMERFS(Statistical Modeling and Estimation of Reliability Functions for Software)[13,14]로 제안하는 모델을 실험하는데 어려움이 있어 직접 개발한 신뢰성 측정 도구 SRTpro(Software Reliability Tool professional)[15,16]를 이용한다. 6절에서는 본 논문의 결론을 기술하도록 한다.

2. 문제 제기

NHPP(Non-Homogeneous Poisson Process) 기반의 기존 모델들은 "결함이 발견되면 즉시 제거된다."는 가정을 전제로, 테스트 시간동안 발견된 고장수를 통해 앞으로 발견될 고장수를 추정하거나 디버그 시간에 따라 제거된 고장을 통해 앞으로 제거될 고장을 추정하게 된다. 여기서 각 모델들은 테스트 시간과 디버그 시간을 분리하여 사용되고 있다. 그러나 실제 소프트웨어 개발에서 발생하는 결함은 발견 즉시 제거되기 어려우며, 이로 인한 지연시간이 발생하게 된다[11,12]. 즉, 일반적인 신뢰성 추정 모델에서는 결함의 제거시간은 고려되지 않고 실제 테스트 시간만을 고려하는 한계를 갖는다. 그러나 일부 연구에 따르면 "고장이 발견되면 해당 고장에 대한 결함이 일정 시간이 지연된 후에 수정된다."는 가정을 갖는 개선된 모델에 대한 연구가 수행되었다[11,12]. 이들 연구들에서의 지연 시간은 "결함이 발견된 시점에서부터 결함이 수정되는 시간을 지연시간"이라 정의하고 있으며 제거시간은 "순수하게 결함을 제거하는 데 소요한 시간"으로 정의되었다. 결국, 기존 모델에서의 지연 시간은 제거 시간을 포함한 의미로 사용되고 있으며, 이때 테스트 시간과는 따로 분리된 개념으로 사용되고 있다. 그림 1은 발견 모델, 제거 모델, 그리고 본 논문에서 제안하는 모델에 대한 테스트 시간, 지연 시간, 그리고 제거 시간을 설명하고 있다.

그림 1에 따르면 발견 모델의 경우에는 지연 시간 또는 제거 시간이 고려되지 않고 있으며 제거 모델의 경우 지연시간을 고려하고 있으나 여기에서 말하는 지연시간의 경우에는 고장이 발견되고 제거되는데 소요되는 시간으로 테스트 시간과는 분리되어 고려가 되고 있다. 논문에서 제안하는 모델의 경우에는 고장을 발견하는데 소요되는 시간뿐만 아니라 이를 제거하는데 시간이

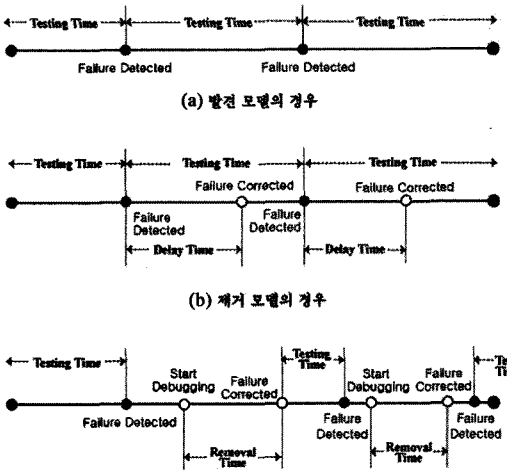

그림 1 테스트 시간, 지연 시간, 제거 시간

함께 고려되어야 한다는 것으로 고장이 발견되면 이를 고치는데 일정 시간(순수하게 제거하는데 소요되는 시간)이 소요되고 또다시 고장을 찾기 위한 테스트가 수행된다는 것을 의미한다. 즉, 기존 소프트웨어 신뢰성 추정 모델인 고장 발견 모델에서는 테스트 시간만을 고려하여 결함 제거 시간을 '0'으로 가정하거나[2,10], 또는 고장 제거 모델의 경우에도 고장의 발견을 확인하기 위한 테스트 시간, 그리고 제거 시간을 포함한 지연 시간을 고려하고 있다[11,12]. 이는 기존 모델 들이 시스템 테스트 또는 운영 테스트와 같이 개발자와 테스터가 분리되어 각각 독립적으로 테스트 또는 디버그 활동을 수행한다는 것을 가정하고 있기 때문이다.

그러나 소규모의 소프트웨어 개발 조직 또는 소프트웨어 개발 단계에서 구현 및 단위 테스트 단계의 경우, 주로 개발자가 직접 테스트와 디버깅을 병행하며 소프트웨어 개발이 이루어진다. 이 경우 테스트 시간과 디버그 시간을 분리하는 기존 소프트웨어 신뢰성 모델을 적용하는 것은 모델 적용상의 문제가 발생할 뿐만 아니라, 부정확한 소프트웨어 신뢰성 측정이 이루어질 수 있다. 그러므로 소프트웨어 테스트 초기 단계 또는 소규모 조직을 위한 새로운 모델이 필요하게 되었다. 또한 그림 2에서 보는 바와 같이 시스템 테스트 단계부터는 신뢰성 측정 및 평가에 있어 기존 신뢰성 모델을 사용할 수 있지만, 개발자에 의한 코딩, 테스트, 디버깅 활동을 병행하는 단위 테스트 또는 통합 테스트 단계에서는 기존 소프트웨어 신뢰성 모델의 직접적인 적용에 이 적합하지 않게 된다. 그러므로 소프트웨어 신뢰성 측정을 위해 단위 테스트와 같은 초기 테스트 단계 또는 소규모 조직을 위한 적합한 새로운 모델이 필요하게 되었다.

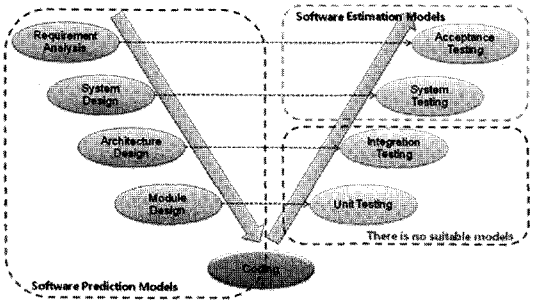

그림 2 V 모델에서의 소프트웨어 신뢰성 측정 및 관리

본 논문에서 정의한 신뢰성 모델은 소프트웨어 단계에서 단위 테스트와 통합 테스트를 통해 발견되는 고장의 수가 어느 정도이며, 고장 사이의 시간이 어느 정도인지를 확인하는 것을 목적으로 한다. 이는 전체 시스템에서의 신뢰성을 추정 한다기 보다, 개별 유닛의 신뢰성을 초기 테스트 단계에서부터 관리하고자 함이다. 즉, 개별 유닛의 신뢰성을 개발자가 수행하는 전체 테스트 및 디버깅 소요시간 동안 발견되는 고장의 수를 통해 남아 있는 잔존 결함을 추정해 보고, 이를 제거하는 데에는 어느 정도의 노력이 필요한지를 추정할 수 있다. 또한 개별 유닛에 대한 신뢰성 목표를 설정했다면, 목표에 도달시 현재 단계에서의 테스트를 중단하고, 다음 테스트 단계로 넘어갈 준비를 할 수 있도록 지원하는 것을 목적으로 한다.

3. 연구 배경

소프트웨어 신뢰성 성장 모델 중 가장 기본적인 모델이 바로 지수형 모델이다. 지수형 모델 중 가장 잘 알려진 모델이 Goel-Okumoto 모델[4]이며 본 논문에서 제안하는 모델 또한 Goel-Okumoto 모델을 변형한 것이다. Goel-Okumoto 모델에 대해서는 3장에서 자세히 다루도록 한다. 본 장에서는 지수형 모델 중 한 모델인 Schneidewind 모델을 시작으로 결함의 제거 과정을 보이는 모델들을 기술하며 이 모델들을 통해서 왜 기존 모델들에 변경이 필요한지에 대해 논의한다. 본 논문의 연구 배경으로 Schneidewind 모델을 설명한 이유는 고장 발견 모델의 대표적인 모델로 Schneidewind 모델과 Goel-Okumoto 모델을 들 수 있는데 두 모델의 경우 유사한 점이 많아 여기에서는 Schneidewind 모델을 설명하고 제거 모델에서 이를 수정한 Schneidewind 제거 모델을 설명하기 위함이다. 그리고 Goel-Okumoto 모델의 경우는 실제 제안하는 모델에서 이를 수정하여 개발한 모델이기 때문에 제안하는 모델을 설명할 때 함께 자세히 논의하도록 한다.

3.1 고장 발견 추정 모델

고장 발견 추정 모델은 고장의 발견 과정을 모델링한 것으로 잘 알려진 모델이 Schneidewind 발견 모델[2,10]이며 이 모델은 IEEE Std. 1633에서 추천하는 모델 중 하나로 미 항공 우주 연구 센터인 NASA의 우주선 소프트웨어의 고장 데이터를 기반으로 개발되고 검증된 모델이다. Schneidewind 고장 발견 추정 모델은 동일한 시간 간격 동안의 결함수를 기반으로 과거의 고장률을 이용하여 현재의 고장률을 구함으로써 앞으로의 고장발생을 보다 정확히 예측하고자 하는 모델이다. Schneidewind 모델은 (1) 테스트 기간 동안 발견된 모든 고장수를 이용, (2) 처음 테스트 시간부터 특정 시점까지의 고장수는 무시하고 선택된 시점부터의 고장수만을 이용, (3) 처음 테스트 시간부터 특정 시점 이전까지의 고장수를 하나의 데이터로 보고 그 이후 발생하는 고장수와 함께 사용하는 3가지 접근 방법을 제공하고 있는데, 특히, (1)의 경우, 기존 수집된 모든 데이터를 이용하여 신뢰성을 측정하고 평가하고 있다. 그러나 (2)와 (3)의 경우에는 모든 데이터를 동일하게 이용하는 것이 아니라 가장 최근의 데이터가 앞으로 고장 발견 패턴에 더 많은 영향을 미친다는 것을 기반으로 한 시점 이전의 데이터를 무시하거나 하나의 데이터로 간주하고 사용하게 된다. 또한 각 접근 방법에 따라 매개변수 α와 β가 MLE(Maximum Likelihood Estimation)[17] 기법에 의해 추정된다.

이 모델을 사용하기 위해서는 다음과 같은 프로세스를 따를 필요성이 있다.

- 데이터 수집을 위한 가정
  - √ 완벽한 디버그
  - √ 제거시간 무시
- 데이터 수집
  - √ 발견된 고장
  - √ 테스트 시간
- Mean Value Function 파라미터 설정
  - √ 전체 고장 수 파라미터
  - √ 고장 발생 율 파라미터
- 신뢰성 검증
  - √ 발견되지 않은 고장 수 추정
  - √ 테스트 또는 배포 결정

3.2 고장 제거 추정 모델

고장 제거 추정 모델은 고장의 제거 과정을 모델링한 것으로 Schneidewind 제거 모델과 JungHua's 제거 모델이 고장 제거 추정 모델에 속하는 모델들이다[11,12]. Schneidewind 제거 모델은 기존의 기본 Schneidewind 발견 모델을 수정한 모델로 기존의 가정인 결함의 제거 시간을 무시하는 것이 현실에 적합하지 않아 지연시간(Delay Time)을 소개하였다. 지연시간이란 고장이 발견

되고부터 제거되는 데까지의 시간을 의미한다.

JungHua's 제거 모델은 기존의 기본 Goel-Okumoto 모델을 수정한 모델로 지연시간뿐만 아니라 고장 정정을 또한 추가로 고려한 모델이다. 즉, 이 모델은 결함이 제거되는 과정을 보이기 위함이며 이를 위해서 결함이 제거되는 과정뿐만 아니라 결함이 발견되는 과정까지도 함께 고려한 모델로 개발되었다.

이 모델들은 다음과 같은 프로세스를 통해서 모델이 사용되어 진다.

- 데이터 수집을 위한 가정
  - √ 완벽한 디버그
  - √ 지연 시간 발생
- 데이터 수집
  - √ 발견된 고장 및 제거된 고장
  - √ 테스트 시간 및 지연 시간
- Mean Value Function 파라미터 설정
  - √ 전체 고장 수 파라미터
  - √ 고장 발생율 파라미터 및 고장 제거율 파라미터
- 신뢰성 검증
  - √ 제거된 고장 수 추정
  - √ 남은 발견된 고장수 및 전체 고장 제거 시간 추정

3.3 기존 모델들과의 비교

3장 연구배경에서 소개한 3개의 모델들은 모두 지수형 기반 모델로 고장 수 기반 모델이다. 기존 모델의 한계를 보완하고자 새로운 모델을 개발하여 제안하게 되었는데 새로 제안하는 모델과 기존의 모델의 비교 내용이 표 1에 정리되었다. 모델 간 사용단계, 가정, 제거시간 사용 유무, 그리고 발견 시간 및 제거 시간의 통합 여부 등을 중심으로 비교하여 정리하였다.

표 1 기존 모델들과 새로운 모델 간 비교

	Detection Model	Removal Model	Our Model
사용 단계	·시스템 테스트 ·운영 테스트		·유닛 테스트 ·통합 테스트
가정	·완벽한 디버그 ·지연시간 없음 ·제거시간 없음	·완벽한 디버그 ·결함 제거/지연 시간 발생	·완벽한 디버그 ·지연시간 없음 ·제거시간 발생
제거 시간	×	△ (지연시간)	○
통합 시간	×	△ (분리)	○
결과	·누적 발견된 고장 수 ·발견되지 않은 남은 고장 수	·누적 제거된 고장 수 ·제거되지 않은 남은 고장 수	·누적 발견된 고장 수 ·발견되지 않은 남은 고장 수
사용 목적	·테스트 중단 시점 결정	·디버그 완료 시점 확인	·개발 초기 단계에서의 유닛별 신뢰성 관리 ·다음 테스트 단계 시점 결정

4. 결함 발견 및 제거 노력 기반 소프트웨어 신뢰성 모델

연구 배경에서 소개하였듯이 기존 신뢰성 추정 모델들은 테스트와 디버그 활동이 분리되어 수행되지만 초기 테스트 단계 또는 소규모의 조직에서는 개발자와 테스터가 분리되지 못하고 개발자가 테스트를 수행하면서 발견되는 결함을 제거하게 된다. 이 경우 테스트 활동과 디버그 활동이 분리되지 못하고 함께 수행이 되기 때문에 이 두 활동에서의 노력을 고려한 새로운 소프트웨어 신뢰성 모델이 개발되었다. 본 장에서는 유닛 또는 통합 테스트 단계의 특징과 이 단계에서 개별적인 유닛의 신뢰성을 측정 관리하기 위한 모델을 기술하도록 한다.

4.1 개요

본 논문에서 제안하는 모델은 NHPP 모델로 잘 알려진 Goel-Okumoto 모델[9]을 기반으로 테스트 초기단계의 특성을 반영하였다. Goel-Okumoto 모델은 Mean Value Function (MVF)에 따라서 누적 고장수를 예측하게 된다.

$$MVF = \alpha \times [1 - \exp(-\beta t)],\ \alpha > 0, \beta > 0$$

이 MVF를 적용하기 위해서는 두 파라미터 α와 β의 값을 추정할 필요가 있는데 이는 수집된 데이터에 따라 Maximum Likelihood Estimation(MLE)을 통해 값들이 추정된다. α는 예상되는 전체 고장수를 의미하며 β는 고장 발생률을 의미한다.

Goel-Okumoto는 테스트 시간만을 고려하여 전체 고장수를 예측하게 된다. 그렇기 때문에 전체 고장수를 추정하는 데 있어서는 문제가 되지 않는다. 그래서 새로 개발된 모델 또한 Goel-Okumoto 모델에서 추정한 전체 고장수를 그대로 따르게 된다. 즉, 두 모델에서의 예측하는 전체 고장의 수는 동일해야 한다는 것을 의미한다. 하지만 Goel-Okumoto 모델의 경우 결함의 제거시간을 고려하지 않기 때문에 순수한 테스트 시간에 따른 예상되는 고장의 수만 예측하기에 테스트 초기 단계와 같이 개발자가 테스트 및 디버그를 모두 수행할 경우 정확한 시간을 예측할 수 없는 한계가 있다. 그래서 고장 발생률이 제거 시간에 따라 변경될 필요성이 있다. 즉, 고장의 발생이 제거시간에 따라 지연이 발생되어 고장 발생률이 변한다는 것이다. 이러한 2가지 특정을 정리하면 다음과 같다.

C1: 기본 모델에서 예측하는 전체 고장수와 새로 제안하는 모델에서의 전체 고장수는 동일하다.

C2: 고장의 발생률은 제거시간에 따라 지연이 발생되어 변하게 된다.

그림 3은 기본 지수형 모델의 결과와 제거시간이 포함되었을 경우의 지연발생으로 인한 그래프의 변화를

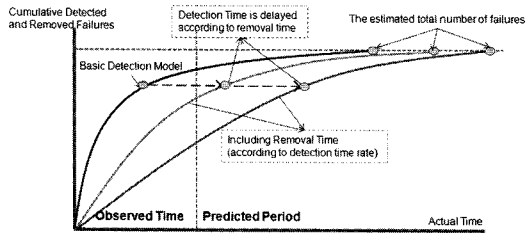

그림 3 기본 지수형 모델 결과와 제거시간에 따른 지연 발생 그래프

보여주고 있다. 여기서 예상되는 전체 고장의 수에는 변화가 발생하지 않음을 기억해야한다.

4.2 가정

본 논문에서 제안한 모델은 지수형 모델을 기반으로 한 모델이기 때문에 기본 지수형 모델의 가정을 그대로 따르고 있다. 그 중 대표적인 가정이 "모든 발견된 결함은 즉시 제거된다."는 것이다. 즉, 결함에 대한 제거 시간은 고려하지 않으며 테스트 시간만을 고려한다는 것이다. 그러나 실제에서는 결함을 제거하는데 일정 시간이 소요됨을 부인할 수 없다. 그렇기 때문에 이러한 가정의 한계를 극복하고자 하는 여러 모델들이 개발되어왔다[11,12].

새로 개발된 모델에서도 결함이 발견되고 그리고 제거되는 시간을 고려하고자 다음의 가정을 따르고 있다.

A1: 모든 발견된 결함은 즉시 제거된다. 그리고 결함이 제거되는 데는 일정한 시간이 소요된다.

A2: 고장 발생률은 시간에 따라 감소한다. 즉, 전체 소요 시간 대비 고장 발견 시간 율에 따라 고장 발생률은 감소한다.

가정 A1은 단위 또는 통합 테스트 단계에서는 개발자가 테스트를 수행하고, 테스트를 수행하면서 발견되는 결함을 제거하게 된다. 그러므로 테스트 수행 후 발견되는 결함을 개발자가 바로 제거하게 되며, 이때 결함을 제거하는데 소요되는 시간이 존재함을 가정함을 의미한다. 그리고 가정 A2의 경우 발 개발자가 테스트를 수행하며 발견된 고장의 원인을 분석하여 결함을 제거하게 된다. 그렇기 때문에 모든 발견된 결함은 즉시 제거되게 된다. 하지만 결함을 제거하는 데는 개발자가 일정시간을 소요하여만 하기 때문에 결함을 제거하는 시간이 반영된 것이다. 가정 A2는 기존 모델에서 발견 모델들에서는 테스트 시간만을 고려한 고장 발생률을 가지지만, 본 논문에서 제안하는 모델은 단위 또는 통합 테스트 단계에서 개발자가 테스트와 디버깅을 병행하기 때문에 테스트 시간뿐만 아니라, 디버그 시간을 함께 고려하고 있다. 그러므로 테스트 시간만을 고려한 고장 발생률은 전체 소요시간 대비 고장을 발견하는데 소요된 테스트 시간에 따라 고장 발생률이 감소되어야 한다는 것을 의미한다. 이처럼 새로 개발된 모델은 기본 지수형 모델의 가정 하에서 위 2가지 가정을 추가적으로 반영하였으며 이 두 가정이 다른 모델들과 크게 차별화어 유닛 또는 통합 테스트 단계의 특성을 반영한 것이라 할 수 있다.

다음은 새로운 모델을 기술하기 이전에 반드시 알아야 하는 기본 표기 및 표기에 대한 설명을 기술하고 있다.

α : Goel-Okumoto 모델의 예상되는 전체 고장 수

α_p : 제안하는 모델에서의 예상되는 전체 고장 수 (α와 동일)

β : Goel-Okumoto 모델의 결함 당 고장 발생률

β_p : 제안하는 모델에서의 결함 당 고장 발생률

t : 고장 발견 시간

t_p : 전체 시간 (고장 발견 시간 + 제거 시간)

$MVF(t)$: Goel-Okumoto 모델에서의 시간 [0, t] 사이의 발견된 누적 고장 수 함수

$MVFp(t_p)$: 제안하는 모델에서의 시간 [0, t] 사이의 발견된 누적 고장 수 함수

4.3 제안된 모델

본 논문에서 제안하는 모델은 기존의 모델에서와 동일한 데이터 수집 절차를 따르고 있다. 테스트를 수행하며 발견된 고장에 대한 데이터를 분석하여 동일한 시간 간격으로 데이터를 정리한다. 뿐만 아니라 제안한 모델에서는 결함을 제거하는 시간 또한 고려하기 때문에 결함에 대한 데이터도 추가적으로 수집하여 정리한다. 제안하는 모델을 통해서 고장이 발견되는 과정을 개발자가 소요한 전체 시간을 반영하기 때문에 보다 정확하게 미래에 발생할 고장에 대한 시간을 예측할 수 있게 된다.

제안하는 모델은 기본 지수형 모델로 잘 알려진 Goel-Okumoto 모델을 기반으로 하였다. 그렇기 때문에 Goel-Okumoto 모델이 가지는 MVF를 기반으로 제안하는 모델에서의 두 가정을 반영하여 새로운 MVFp를 개발하게 되었다. 기본적으로 전체 발견되는 고장의 수에는 변화가 없어야 하기 때문에 테스트 시간만을 고려한 α의 값과 α_p의 값은 동일하야 함을 알 수 있다.

$$P1: \alpha_p = \alpha \quad (C1)$$

그리고 앞에서 가정한 A1과 A2에 따라서 테스트 시간만을 고려했을 때의 고장 발생 율 β는 전체 소요시간 대비 테스트 시간율에 따라서 그 값이 감소하게 된다.

$$P2: \beta_p = \beta \times [T_d \times (T_d + T_r)] \quad (C2)$$

T_d는 고장을 발견하는데 소요된 시간을 말하며 T_r은 결함을 제거하는데 소요된 시간을 말한다. 즉 T_d와 T_r의 합은 소요된 전체 시간을 의미한다. 위에서와 같이 P1과 P2에 따라서 기존의 MVF는 다음과 같이 표기되어 결함 제거 시간을 고려하여 누적 발견된 고장을 예측할 수 있게 된다.

$$MVF_p = \alpha_p \times [1 - \exp(-\beta_p t_p)]$$

새로 제안하는 모델의 프로세스는 다음과 같이 정리되며 이는 그림 4와 같다.

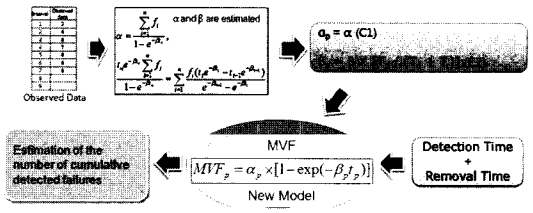

그림 4 새로 제안하는 모델의 프로세스

- 데이터 수집
  - √ 발견된 고장 및 제거된 고장
  - √ 테스트 시간 및 제거 시간
- 파라미터 추정
  - √ 전체 고장 수 파라미터
  - √ 고장 발생율 파라미터 및 고장 제거율 파라미터
- 파라미터 변경
  - √ 새로 제안하는 모델의 특성에 맞는 파라미터로 변경
  - √ α_p 와 β_p 값 계산
- 전체 시간 계산
  - √ 발견 시간 및 제거 시간 정리
  - √ 전체 소요 시간 t_p 계산
- Mean Value Function 재구성
  - √ α_p, β_p, t_p 적용 및 결과 추정
- 신뢰성 검증
  - √ 누적 발견된 고장 수 및 남은 고장 수 추정
  - √ 현재 신뢰성 상태 확인

5. 실 험

본 장에서는 새로 제안하는 모델을 실제 사용하는데 있어 문제없이 사용될 수 있는지를 확인하기 위해 실제 산업현장에서 수집된 유닛 및 통합 테스트 단계에서의 데이터를 이용하여 실험하였다. 실험을 위한 환경 및 수집된 데이터 그리고 실험 결과가 본 장에서 기술되어진다.

5.1 실험 환경

실제 산업데이터를 수집하고 이를 기존의 Goel-Okumoto 모델과 새로 제안하는 모델에 적용하는데 있어 기존의 도구들(CASRE, SMERFS)을 이용하는 데는 두 모델을 함께 비교하는데 어려움이 있어 직접 개발한 소프트웨어 신뢰성 도구 SRTpro를 이용하여 실험을 하였다. SRTpro는 기존의 도구들이 갖는 단점을 보완하고자 개발된 도구로 현재 무료로 사용가능한 도구이다.

5.2 수집 데이터

제안하는 모델을 실험하기 위해서 가상의 데이터가 아닌 실제 산업데이터를 이용하여 실험을 수행하였다. 데이터는 현재 개발 진행 중인 프로젝트로부터 수집되었으며 개발 조직은 현재 CMMI 레벨 5를 보유한 조직이다. 데이터 수집을 위해서 표 2와 표 3과 같은 두 수집 템플릿을 이용하였으며 개발자가 직접 데이터를 입력하여 전달하였다.

표 2는 발견된 고장에 대한 수집 템플릿으로 테스트를 수행하면서 수집된 데이터를 기반으로 작성되었으며, 표 3은 발견된 결함에 대한 수집 템플릿으로 결함의 제거 시간을 기록하게 되어있다. 이와 같이 두 수집 템플릿을 이용하여 실험 데이터가 수집되었으며 이를 다시 유닛별로 모델의 입력으로 적합하도록 재정리하였다.

수집 데이터는 전체 28개의 파일로 수집되었으며 그 중 실험 가능한 데이터를 분류하는 작업을 통해서 10개의 유닛에 대한 데이터가 현재 실험에 사용되었다. 현재 계속적으로 데이터를 분석하고 있는 단계이기 때문에 추후 추가적으로 사용 가능한 유닛은 증가할 것이다. 현재 이용가능하지 않은 데이터는 지수형 모델을 따르지 않는 경우와 유닛의 크기가 너무 작아 고장이 발견되지 않았거나 테스트 시간이 적어 실험 데이터로 이용하는 데는 적합하지 않았다.

5.3 실험 목표

표 2 고장 발견 템플릿

Milestone	CSCI	CSC	Detection Date	Test Script ID	Test Cases	People	Time	cpu time	Total Time	Failures	Faults
Integration testing	A	A-1	2009-12-08	1	10	1	215	105	215	2	3

표 3 결함 제거 템플릿

Milestone	CSCI	CSC	Fault ID	Test Script ID	i th test case	Removal date	Severity	People	Removal time
Integration testing	A	A-1	1	1	5	2009-12-08	Major	1	50
			2	1	7	2009-12-09	Minor	1	50

새로 제안하는 모델이 실제 수집된 데이터와 어떠한 관계를 보이는지를 확인하는 것이 최종 실험 목표이다. 우선 기존의 Goel-Okumoto 모델을 적용할 시에 결함의 제거시간을 무시하는 가정으로 인한 실제 데이터와의 차이를 보이는 실험을 통해 한계를 확인하고 이를 보완하고자 하는 새로 제안하는 모델을 확인하고 실제 데이터와의 차이를 비교한다. 실제 데이터와의 차이 비교를 위해서 Mean Relative Error(MRE)[12]와 Mean Square Error(MSE)[18] 방법을 사용하여 그 차이를 확인하고 Box-Plot을 통해서 값의 범위를 확인하는 것으로 기존 모델과 새로 제안하는 모델의 차이를 비교한다.

(1) Mean Relative Error(MRE)

$$\frac{1}{n}\sum_{k=1}^{n}\left|\frac{m(t_k)-Z_k}{Z_k}\right|$$

Z_k는 k시간에서의 실제 누적 발생한 고장의 수로 소프트웨어를 테스트하면서 관측되어지며 $m(t_k)$는 모델의 MVF를 통해서 추정된 k시간에서의 추정된 누적 고장의 수를 의미한다. n은 전체 타임 유닛의 수를 말하며 평균을 계산한 결과 값을 통해서 실제와의 차이를 확인함과 동시에 작은 값을 가질수록 실제 데이터와 차이를 보이지 않는다고 해석될 수 있으며 미래의 고장수를 보다 정확할 수 있다고 할 수 있다.

(2) Mean Square Error(MSE)

$$\frac{1}{n}\sum_{k-1}^{1}[m(t_k)-Z_k]^2$$

여기서도 Z_k는 k시간에서의 실제 관측된 누적 고장수를 의미하며 $m(t_k)$는 모델의 MVF를 통해서 추정된 누적 고장수를 의미한다. 이렇게 실제 값과 모델에서의 값의 차이를 제곱한 평균을 구한 것이 MSE값이며 이 값을 통해서 모델이 실제 값과 얼마나 차이를 보이는지를 확인하는 것이다. 대부분 모델간의 우위를 비교하는 경우에도 자주 사용되는 값이며 이 값을 통해서 모델을 선택하기도 한다. MSE값이 작으면 작을수록 실제 값에 근접했다는 의미로 해석되며 미래의 고장수를 보다 정확하게 예측할 수 있다는 의미로 해석된다. 그런 다음 Goel-Okumoto 모델과 새로 제안하는 모델이 서로 차이가 있는지 확인하고 실제 값과 제안하는 모델에서의 결과가 통계적으로 차이를 보이는지를 확인하기 위해 Minitab[19] 도구를 이용하여 Paired T Test[20]를 통해 검증한다. 이러한 실험 결과 및 통계적 결과는 다음 장에서 기술한다.

본 논문에서는 제거 모델과의 비교 실험은 수행하지 않고, 대신 이를 향후 연구과제로 남겨두고자 한다. 이는 본 논문에서 제안한 모델의 경우, 고장을 발견하는데 소요된 시간과 제거하는데 소요된 시간을 모두 고려하여 고장이 발견되는 상태를 추정하기 때문이다. 즉, 제거되는 상태를 추정하는 것이 본 논문에서 제안하는 모델의 목적이 아니기 때문이다. 그러나 향후, 본 논문에서 제안한 모델을 수정하여 고장을 발견하는데 소요된 시간과 제거하는데 소요된 시간을 고려하여 고장의 발견 상태만을 추정할 수 있는 것이 아니라, 고장의 제거 상태 또한 추정이 가능하도록 수정 될 것이며, 이때 본 논문에서 제안하는 모델과 제거 모델의 비교를 수행하도록 한다.

5.4 실험 및 검증 결과

전체 이용 가능한 10개의 유닛 중 하나의 유닛의 고장 데이터를 이용하여 실험을 수행하였다. 우선 그림 5는 고장의 제거 시간을 무시하는 Goel-Okumoto 모델의 MVF결과 그래프이다.

그림 5는 오직 고장의 수와 테스트 시간만을 고려한 MVF를 이용하여 추정한 누적 고장 수 그래프이다. 이 그래프는 34시간 유닛에서 10개의 고장을 추정하며 38시간 유닛에서는 10.41의 고장수를 추정하고 있다.

그림 6은 고장의 제거시간을 반영하여 새로 제안하는 모델을 통한 결과 그래프와 기존 모델의 그래프를 함께 그려 비교를 할 수 있도록 하는 그래프이다.

그림 6은 기존 모델에서 고장 제거 시간을 반영하지 않은 결과를 보이고 있으며, 본 논문에서 제안하는 모델의 경우, 전체 시간에서 120분의 시간을 고장 제거에 사용해, 이를 반영하여 고장 발생률을 감소시킨 결과를 나타낸다. 즉, 실제 개발자가 고장을 찾는데 소요된 시간 이외에도 발견된 고장의 원인인 결함을 발견하여 제거하는데 120분이라는 시간을 추가적으로 소요하여 그만큼 지연이 발생하게 되어 이를 반영한 것이 제안한 모델이다. 그림 7은 이 두 그래프와 실제 개발자가 테스트와 디버그에 소요한 시간에 따른 발견한 고장수를 포함하고 있다. 그림 7에서와 같이 고장을 제거하는 데 소요된 시간을 고려한 제안하는 모델이 실제 데이터에 보다

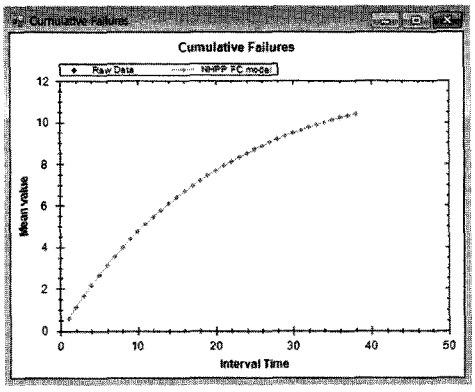

그림 5 Goel-Okumoto 모델의 결과

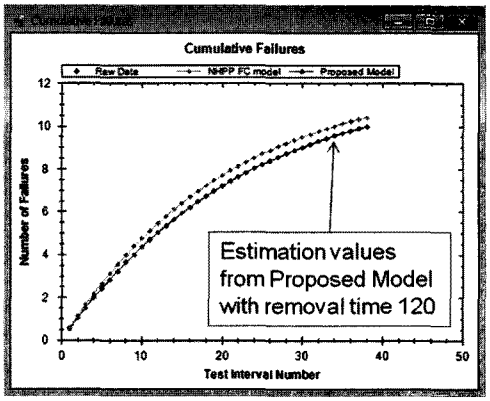

그림 6 Goel 모델과 제안하는 모델 비교

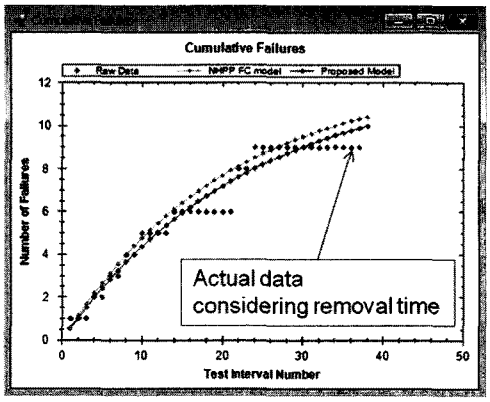

그림 7 실제 데이터와의 비교

표 4 실제 데이터와 Goel-Okumoto 모델과 제안하는 모델과의 비교 결과

	Goel-Okumoto Model	Proposed Model
Time	1140	1140
Detection time rate	-	0.8947
α	12.3437	12.3437
β	0.0489	0.0437
MRE	0.1276	0.0909
MSE	0.5964	0.2990
Total intervals	38	38
Failures	10.4161	10

정확하게 결과를 추정하고 있다고 할 수 있다.

표 4는 그림 7에서와 같이 Goel-Okumoto 모델을 이용하여 추정한 결과와 제안하는 모델에서의 추정한 결과를 실제 수집된 데이터와 비교하여 그 결과를 정리하였다. 즉, 초기 테스트 단계에서는 개발자가 테스트와 디버그를 모두 수행을 하지만 Goel-Okumoto 모델의 경우에는 이를 제대로 반영하지 못해 위 그림과 같이 실제와 오차를 발생하게 된다. 표 4는 전체 고장수를 의미하는 α 값과 고장 발생률을 의미하는 β가 어떻게 변경되었는지를 보이고 있으며, 본 논문에서 제안하는 모델에서는 실제 소요된 시간을 모두 반영하고 있기 때문에 Goel-Okumoto 모델에 비해 작은 MRE와 MSE를 값을 가짐을 알 수 있다. 즉, 기존 모델에서는 개발자가 테스트와 디버그 활동을 모두 수행하는 경우 개발자가 실제 소요한 시간 모두를 반영하지 못해 실제 시간과 차이를 보이고 있으나, 제안하는 모델은 이를 모두 반영하기 때문에 실제 값에 유사한 결과를 보이고 있음을 알 수 있다.

Goel-Okumoto의 경우 개발자가 테스트 및 디버그 활동을 수행하는 것을 함께 가정하지 못하기 때문에 실제 이 단계에서의 수집 데이터와 큰 차이를 보이게 된다. 즉, 제안하는 모델의 경우가 Goel-Okumoto 모델의 경우보다 MRE와 MSE의 값이 작은 것을 확인할 수 있다.

다음으로 각 모델에서의 추정 결과에 대한 측정치와 실제 값과의 차이를 이용하여 Box-plot[21]을 그린 것이 그림 8과 같다. 그림 8에서와 같이 제안하는 모델의 경우 중간 값이 0에 가까운 것을 알 수 있으나 Goel-Okumoto의 경우 결함 제거시간을 반영하지 못하기 때문에 실제 값과 큰 차이를 보이는 것을 Box-plot을 통해 쉽게 확인할 수 있다.

Box-plot에 대한 정확한 수치는 표 5에 정리되어있다. 표 5에서는 기존의 모델을 사용했을 경우 테스트 시간만을 고려하였기 때문에 실제 개발자가 소요한 시간의 일부분만을 반영하게 된다. 그렇기 때문에 실제 개발자가 소요한 시간(테스트 시간 + 디버그 시간)과는 차이를 보일 수밖에 없게 된다. 그러나 제안된 모델을 사용하게 된다면 실제 개발자가 소요한 모든 전체 시간이 반영되기 때문에 보다 정확하게 개발자가 어느 정도의 노력을 통해 고장을 발견했는지 그리고 해당 목표에 도달하기 위해서는 어느 정도의 노력이 더 필요한지 등을

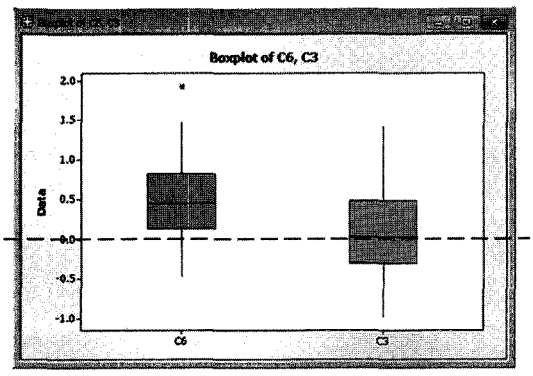

그림 8 Goel(C6)과 제안하는 모델(C3)의 결과

표 5 Box-plot 측정치

	Goel Model (C6)	Proposed Model (C3)
Max	0.8236	0.4844
Min	-0.1321	-0.3.29
Median	0.4463	0.0260
IQRange	0.6915	0.7873
N	38	38

표 6 실제 값과 Goel 모델과 제안한 모델에서의 값의 차이에 대한 Paired-T Test 결과

	N	Mean	StDev	SE Mean
Goel	38	0.501650	0.537837	0.087249
Proposed	38	0.113082	0.550028	0.089225
Difference	38	0.388568	0.186429	0.030243
T-Value	12.85		P-Value	0.000

표 7 실제 값과 제안하는 모델에서의 추정 결과에 대한 Paired-T Test 결과

	N	Mean	StDev	SE Mean
Actual Data	38	6.34211	2.76343	0.44829
Estimated Data from proposed model	38	6.645519	2.79828	0.45394
Difference	38	-0.113082	0.560018	0.089225
T-Value	-1.27		P-Value	0.213

보다 정확하게 추정할 수 있는 이익이 있다. 즉, 제안하는 모델에서의 추정 결과가 실제 데이터와의 차이가 그만큼 적다는 것을 의미한다.

지금까지는 Goel-Okumoto 모델과 제안하는 모델에서의 차이를 설명하기 위한 실험 및 그 결과를 기술하였다. 지금까지의 실험으로 두 모델에서의 차이가 확연히 나타났으며 개발자가 테스트 및 디버그를 모두 수행해야 하는 경우에는 제안하는 모델의 경우가 보다 실제 값과 일치한다는 것을 확인하였다.

표 6은 이 두 모델이 통계적으로 차이가 있는 모델인지 그리고 실제 데이터와 제안하는 모델에서의 측정치가 통계적으로 차이를 보이는지를 검증하기 위해 Paired T Test를 수행한 결과이다. 결과에서 확인할 수 있듯이 우선 귀무가설의 경우 두 모델 또는 두 데이터사이에 차이가 없다는 것이며 대립가설의 경우는 두 모델 또는 두 데이터사이에 차이가 있다는 것이다. 우선 두 모델의 경우 95%의 신뢰도 구간에서 P-Value의 값이 0으로 0.05보다 작은 값을 가진다. 그렇기 때문에 귀무가설을 기각하고 대립가설을 선택하게 된다. 즉, 두 모델은 통계적으로 차이를 보인다고 할 수 있다. 표 7은 실제 수집된 데이터와 제안하는 모델을 통해서 측정된 추정치와의 Paired T Test수행 결과이며 95%의 신뢰도 구간에서 P-Value의 값이 0.05보다 큰 값을 가지는 것을 확인할 수 있다. 즉, 귀무가설을 기각하고 대립가설을 선택할 수 없다는 것이다. 그러므로 두 데이터 사이에 차이가 존재한다고 통계적으로 증명할 수 없다는 것이다.

본 장에서는 새로 제안하는 모델을 사용하기에 앞서 기존 모델과 어떤 차이를 보이는지를 위한 실험을 수행하였다. 또한 두 모델 간 그리고 실제 데이터와 제안하는 모델에서의 추정치와의 차이 존재여부를 확인하기 위해 Paired T Test 검증 방법을 수행하였다. 본 실험을 통해서 제안하는 모델의 경우 개발자가 테스트와 디버그를 모두 수행할 경우 보다 정확한 추정결과를 보이는 것을 확인했으며 기존 모델과의 차이가 존재함을 통계적 검증방법을 통해서 확인하였다. 또한 실제 값과 제안하는 모델에서의 추정치 사이에서도 차이가 존재하는지 여부를 통계적으로 확인하였다.

특히 Goel-Okumoto 모델의 경우 테스터가 수행한 테스트 시간과 고장 수 만을 고려하여 신뢰성을 추정하고 있다. 그러나 2장에서 언급한 바와 같이 단위 테스트와 통합 테스트와 같은 초기 테스트 단계에서는 개발자와 테스터가 분리되지 못하고 개발자가 테스트 및 디버그 활동을 함께 수행하게 된다. 이 경우 기존 Goel-Okumoto 모델을 적용했을 경우 실제 개발자가 소요한 전체 시간의 측면에서 상당한 차이를 보이게 된다. 그러므로 본 논문에서는 제안된 모델이 이러한 차이를 줄이고 개발자가 수행하는 모든 활동에서의 소요 시간을 반영한 결과 보다 정확한 소요 시간을 예측하고 있음을 알 수 있다. 그러므로 초기 테스트 단계에서 개발자가 테스트와 디버그 활동을 모두 수행하게 될 경우 기존의 모델을 사용하는 것이 아니라 본 논문에서 제안한 모델을 사용하게 된다면 개발자가 실제 소요한 시간에 다른 실제 발견 고장의 수를 보다 정확하게 추정할 수 있게 된다. 또한 향후 발생할 수 있는 잔여 고장, 그리고 목표 도달에 필요한 노력을 정확하게 추정할 수 있다는 점을 실험을 통해 살펴보았다.

6. 결 론

기존의 소프트웨어 신뢰성 추정 모델들은 나중 테스트 단계인 시스템 테스트와 운영 테스트 단계에서 주로 사용되고 있다. 그러한 이유로 기존의 모델들은 크게 고장 발견 추정 모델과 고장 제거 추정 모델로 나누어 질 수 있다. 고장 발견 추정 모델들은 테스트 시간만을 고려하여 시간 유닛당 발견되는 고장 수 데이터를 이용하여 앞으로 발생할 고장수를 추정하는 모델이다. 고장 제거 추정 모델들은 단위 시간당 제거되는 고장 수 데이

터를 이용하여 앞으로 제거될 고장수를 추정하는 모델이다. 이 모델들은 테스터와 개발자가 분리되기 때문으로 초기 테스트 단계에서 개발자가 테스트와 디버그를 함께 고려할 때를 반영하지 못하고 있다. 이를 보완하기 위해 테스트 시간뿐만 아니라 디버그 시간까지 함께 고려하는 새로운 모델을 제안했으며 이 모델을 통해서 초기 테스트 단계에서부터 개별 유닛의 신뢰성을 관리할 수 있기 때문에 나중 단계에서의 고장 제거 비용이 감소될 수 있다. 새로 제안하는 모델은 실제 산업데이터를 이용하여 기존 모델과 비교를 통해 그 특징을 살펴보았으며 Paired T Test를 통해서 기존 모델과의 차이와 실제 데이터 값과의 차이를 통계적인 방법으로 확인하였다. 이 모델을 통해서 테스트 측면에서는 보다 정확한 추정 결과를 제공할 수 있으며 개발 측면에서는 유닛별 신뢰성 관리를 통해 나중 단계에서의 결함 제거 노력이 감소될 수 있다. 관리 측면에서는 예측 모델, 초기 추정 모델, 나중 추정 모델의 3단계 프로세스를 제공하여 스케줄 관리 및 비용 감소 효과를 기대할 수 있으며 전체 프로젝트 측면에서는 신뢰할 수 있는 소프트웨어를 개발할 확률을 높일 수 있을 것으로 기대한다.

참 고 문 헌

[1] "소프트웨어 융합의 개요(上)", 한국소프트웨어진흥원(KIPA), April, 2009.

[2] "IEEE Recommend Practice on Software Reliability," *IEEE Reliability Society*, June, 2008.

[3] Michael R. Lyu, "Software Reliability Engineering: A Roadmap," FOSE, 2007.

[4] M. Xie, Software Reliability Modelling, World Scientific, 1991.

[5] Ch. Ali Asad Muhammad Irfan Ullah, Muhammad Jaffar-Ur Rehman, "An Approach for Software Reliability Model Selection," *Computer Software and Applications Conference*, 2004.

[6] Yinong Chen and Jean Arlat, "An Input Domain-Based Reliability Growth Model and Its Applications in Comparing Software Testing Strategies," *LAAS REPORT*, April, 1995.

[7] S. Yamada, M. Ohba, and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability*, vol. R-32, pp.5475-5478, 1983.

[8] Reliability Analysis Center, Introduction to Software Reliability: a State of the Art Review, Rome Laboratory, 1996.

[9] Michael R. Lyu, Handbook of Software Reliability Engineering, IEEE Computer Society Press, 1997.

[10] Norman F. Schneidewind, "Reliability Modeling for Safety-critical Software," *IEEE Transactions on Reliability*, March, 1997.

[11] Norman F. Schneidewind, "Modeling the Fault correction Process," 12th International Symposium on Software Reliability, November, 2001.

[12] Jung-Hua Lo, Chin-Yu Huang, "An Integration of Fault Detection and Correction processes in Software Reliability Analysis," The Tournal of Systems and Software, 2006.

[13] Allen Nikora, CASRE-A Computer-Aided Software Reliability Estimation Tool, http://www.openchannel foundation.org/projects/CASRE_3.0.

[14] William Farr, Oliver Smith, SMERFS-Statistical Modeling and Estimation of Reliability Functions for Systems, http://www.slingcode.com/smerfs/downloads/, 1996.

[15] Myungmuk Kang, Taewan Gu, Jongmoon Baik, "A User Friendly Software Reliability analysis Tool based on Development Process to Iteratively Manage Software Reliability," International Symposium on Software Reliability Engineering, 2009.

[16] Myungmuk Kang, Taewan Gu, Jongmoon Baik, Software Reliability Tool professional, http://spiral.kaist.ac.kr/SRTpro/SRTpro_Download.htm, SPIRAL in KAIST, 2009.

[17] Robert V. Hogg, Joseph W. McKean, Allen T. Craig, Introduction to Mathematical Statistics, Pearson, 2005.

[18] Wikipedia, Mean Squared Error, http://en.wikipedia.org/wiki/Mean_squared_error, 2010.

[19] Minitab, http://www.minitab.com/en-KR/default.aspx, 2010.

[20] Winks, Statistical Data Analysis, http://www.texasoft.com/winkpair.html, 2010.

[21] Wikipedia, Box plot, http://en.wikipedia.org/wiki/Box_plot, 2010.

강 명 묵

2006년 서경대학교 컴퓨터과학과 학사. 2010년 한국과학기술원 전산학과 석사. 관심분야는, 소프트웨어 신뢰성 분석 및 평가, 소프트웨어 프로세스 개선, 소프트웨어 테스팅, 소프트웨어 품질보증

구 태 완

2000년 한림대학교 컴퓨터공학과, 수학과 학사. 2002년 한림대학교 컴퓨터공학과 석사. 2007년 한림대학교 컴퓨터공학과 박사. 2007년 3월~2008년 5월 대한상공회의소 강원인력개발원 정보기술과 교사. 2008년 6월~2009년 5월 한국정보통신대학교 공학부 Post-Doc. 2009년 6월~현재 한국과학기술원 정보전자연구소 Post-Doc. 관심분야는, 임베디드 소프트웨어 신뢰성 모델링, 소프트웨어 신뢰성 분석 및 평가, 소프트웨어 프로세스 개선, 소프트웨어 테스팅, 소프트웨어 품질보증

백 종 문

1993년 조선대학교 컴퓨터과학 및 통계학과 학사. 1996년 미국 University of Southern California Computer Science 석사. 2000년 미국 University of Southern California Computer Science 박사 2001년~2005년 미국 모토로라 SSERL (Software and System Engineering Research Lab.) 수석 연구원. 2005년~2009년 2월 한국정보통신대학교 공학부 부교수. 2009년 3월~현재 한국과학기술원 전산학과 부교수 관심분야는 소프트웨어 신뢰성, 소프트웨어 매트릭스, 소프트웨어 비용추정, 소프트웨어 동적 모델링, 소프트웨어 프로세스 개선, 소프트웨어 품질보증, 소프트웨어 식스 시그마