▨응용논문

대규모 통신 소프트웨어의 결함 수 예측에 관한 사례 연구

박영식 · 윤병남
한국전자통신연구원
임재학
대전산업대학교 회계학과

An Empirical Study on Faults Prediction for Large Scale Telecommunication Software

Young-Sik Park · Byeong-Nam Yoon
Electronics & Telecommunication Research Institute
Jae-Hak Lim
Dept. of Accounting, Taejon Nat'l Univ. of Technology

Abstract

In this paper, we consider the change request data collected from the system test of a large-scale telecommunication software and analyze the types and causes of failures. And we develop statistical models that incorporate a functional relation between the faults and some software metrics. To this end, we consider three possible regression models including a stepwise regression model and two nonlinear models. Three developed models are evaluated with respect to the predictive quality. We also discuss the advantage of proposed models and the application of our model to a new project.

1. 서론

대규모 교환 시스템이나 전송 시스템과 같은 통신 시스템의 개발에는 많은 인력과 예산 그리고 몇 년에 걸친 개발 기간이 요구된다. 이러한 대규모 시스템을 성공적으로 개발하기 위해서 다양한 개발 방법과 개발도구들이 시스템의 특성에 알맞게 적용되고 있다. 몇 년 전까지 만해도 대부분의 통신시스템은 그 기능을 수행하는데 하드웨어에 의존하였으나 시간이 지남에 따라 소프트웨어에 대한 의존도가 점차 높아지고 있다. Healy, Jain과 Bennett(1996)에 의하면 미국의 NRC(Network Reliability Council)에서는 교환 시스템의 고장 중 80%가 소프트웨어에 원인이 있다고 결론을 내리고 있다.

소프트웨어 수명 주기는 크게 네 단계로 구성된다. 즉, 소프트웨어 시스템을 그 개발방법에 따라 서브시스템, 블록들로 세분화시키는 설계 단계, 설계단계의 최종 단위인 블록들을 실현하는 구현 단계, 구현이 완료된 불럭들을 모아 기능을 확인하는 시험 단계, 그리고 이를 현장에 적용하여 운영하는 운영 및 유지보수 단계로 나눌 수 있다. Johansson과 Nord(1995)는 소프트웨어의 수명주기 중 운용 및 유지보수에 소요되는 비용이 전체 비용의 약 50%를 차지하고 있으며, Basili와 Hutchens(1983)와 Porter와 Selby(1990)는 임의의 소프트웨어에서 발생하는 결함의 대부분이 소수의 소프트웨어 블록에서 발생한다고 지적하고 있다. 또한 Johansson과 Nord(1995)는 통신 시스템의 경우 기능 검사에서 보고된 문제의 55%가 전체 코드의 20%이내에서 발생했음을 밝히고 있다. 따라서 소프트웨어의 신뢰성을 향상시키기 위하여 개발단계에서 결함이 유입되는 것을 방지하거나 가능한 조기에 결함을 발견하여 제거하는 것이 중요하다. 결함을 많이 포함할 가능성이 높은 블록 (이하, 다결함 블록)을 소프트웨어 개발 초기에 찾아내어 이들 블록에 대한 집중적인 관리를 통해 소프트웨어 시스템의 신뢰성을 향상시키고자 하는 방법들이 연구되어 왔다.

Karunanithi와 Malaiya(1995)는 소프트웨어 메트릭을 이용하여 잠재적 결함을 예측하는 분야에 대한 연구를 추정적 접근(estimative approach)과 분류적 접근(classification approach)으로 구분하고 있다. 추정적 접근에서는 시험과정을 통해 나타나게 될 잠재적 결함 수를 예측하는데 주로 회귀모형을 사용하고 있다. (Craw et al.(1985), Khoshgoftaar와 Munson(1990), Lyu et al.(1995).) 그리고 분류적 접근에서는 소프트웨어 블록들이 두 세 개의 그룹으로 나누고 판별분석과 같은 분류 방법을 이용하여 다결함 블록을 식별하는 모형을 개발하고 있다. (Rodriguez와 Tsai(1987)과 Munson과 Khoshgoftaar(1992).)

그러나 대부분의 연구들이 소스 코드의 라인 수가 적은 소규모 소프트웨어 시스템을 대상으로 하고 있다. 그리고 이런 연구들은 분석에 필요한 소프트웨어 메트릭으로 McCabe(1976)의 순환 복잡도(cyclomatic complexity)나 Henry와 Kafura(1981)의 정보 흐름 복잡도(infomation flow complexity)등과 같은 복잡도(complexity) 메트릭이나 Halstead의 메트릭 그리고 연산자와 피연산자의 수 등과 같이 소프트웨어의 구조

를 세밀하게 분석해야 얻을 수 있는 자료를 요구하고 있다.

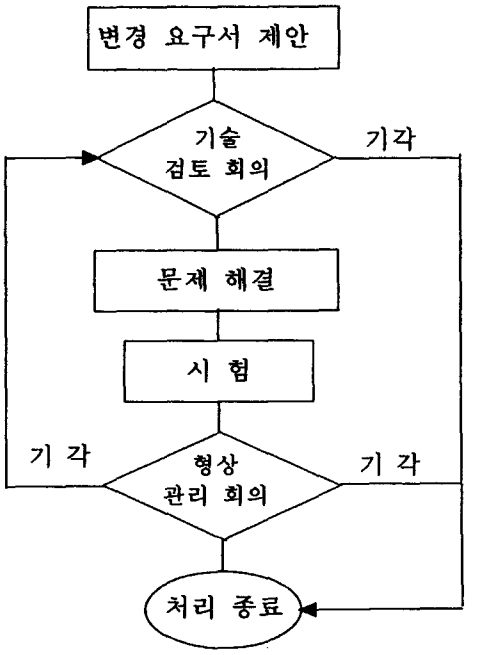
그러나 본 고에서 고려하는 소프트웨어는 교환 시스템의 소프트웨어로서 2년 이상의 개발기간이 소요되며 소스 코드의 라인 수가 백만 라인이상 되는 대규모 소프트웨어이다. 이러한 대규모 소프트웨어 시스템은 방대한 프로그램 규모와 소프트웨어 관련 문서의 부족 및 장기간의 개발기간에 따른 프로그래머들의 교체 등으로 인해 앞에서 언급한 상세한 데이터를 수집하는 것은 어려운 일이며 때로는 불가능하게 된다. 본 고에서는 이러한 문제점을 해결하기 위하여 소프트웨어의 현장 시험과정에서 수집 가능한 자료를 이용하여 소프트웨어의 신뢰도를 예측하는 방법을 제시하고자 한다. 제2장에서는 TDX-10 ISDN교환기와 시스템 변경 요구서를 이용한 소프트웨어 개발 관리 체계를 소개하고 제3장에서는 수집된 자료를 바탕으로 교환기의 고장 유형 및 원인에 대해서 분석하고자 한다. 그리고 제4장에서는 수집된 자료를 바탕으로 소프트웨어 블록이 갖고 있는 잠재적 결함 수를 예측할 수 있는 통계적 모형들을 개발하고 이 모형들의 유용성을 비교 하고자 한다.

2. TDX-10 ISDN 교환시스템의 변경 요구 데이터

본 고에서 고려되는 TDX-10 ISDN 교환시스템은 한국전자통신연구원에서 개발되었으며 이 교환 시스템의 소프트웨어는 139개의 소프트웨어 블록과 130만 라인의 소스로 구성된 대규모 소프트웨어이다. 시스템 변경 요구 데이터는 TDX-10 ISDN 교환 시스템의 개발 확인 시험이 끝난 1993년 6월부터 상용 서비스를 위해 현장에 적용되는 1995년 3월 사이의 약 93주 동안 실시된 시스템 현장 시험을 통해 수집되었다. 소프트웨어 시스템의 모든 오류나 기능의 추가 사항 및 보완 사항은 시스템 변경 요구서를 통해서만 반영되도록 관리를 일원화하였으며, 이러한 개발 관리는 시스템 개발의 효율성 및 개발에 대한 기록을 위하여 효과적인 방법이다. 시스템 변경 요구서는 시스템에서 발생했던 각종 오류나 기능의 추가에 따른 보완사항이 기술된 문서로서 형상관리 절차에 따라 처리되었다.

<그림 1>은 시스템 변경 요구서가 어떤 절차를 통해 수집되는 지 보여주고 있다. 시험 도중 고장이 발생하면 시험자는 고장과 관련된 제반사항을 기술한 시스템 변경 요구서를 작성하여 기술 검토 위원회에 제출한다. 기술 검토 위원회에서는 변경 요구서에서 명시된 문제점을 검토하여 해결 가치가 있는 요구서를 분류하고, 중복된 변경 요구서나 사용자의 오작동에 기인한 변경 요구서를 기각하여 처리를 종료한다. 문제 해결 단계에서는 선별된 변경 요구서에 따라 고장과 연관된 소프트웨어 블록을 수정하고 이를 변경요구서에 기록한다. 시험단계에서는 수정된 내용을 시험하여 더 이상의 오류가 발생하지 않으면 시험결과를 변경 요구서에 기록하여 형상 관리 위원회에 제출한다. 그리고 형상관리 위원회에서는 문제점이 완벽하게 보완되지 않았거나 재시험이 충분하지 않다고 판단되는 경우(왼쪽 기각) 이전 단계를 다시 반복하게 한다. 그

리고 기존의 블록을 수정 보완된 블록으로 교체할 것인지를 결정한다. 기존의 블록을 수정된 블록으로 교체하지 않는 이유(오른쪽 기각)는 사업 차원에서 수정된 블록의 적용으로 얻어지는 이익이 전체 시스템에 가져올 위험 파급효과보다 적다고 판단되기 때문이다.



< 그림 1 > 변경 요구서 처리 흐름도

3. 변경 요구서의 기술 통계적 분석

시스템 변경 요구서는 관찰 기간동안 1503개가 발생하였으며 이 중 중복이나 기타 이유로 196개의 변경요구서가 기각되었으며 나머지 1307개의 변경 요구서를 바탕으로 소프트웨어를 수정 보완하였다. 시스템 변경 요구서를 보완 내역별로 분류해 보면 <표 1>과 같다.

<표 1>에서 관련 변경 요구서란 해당 분류항목을 포함하는 변경요구서이며, 순수 변경요구서는 오직 해당 분류항목만의 이유로 변경된 변경요구서이다. 즉, 임의의 변경 요구서가 DB와 SRC를 변경하였다면 그 변경요구서는 DB의 관련 변경요구서와 SRC의 관련 변경 요구서에 각각 추가되며 아울러 순수변경 요구서 개수에는 중복 난에 추가된다. 관련 변경 요구서의 비율을 살펴보면 프로그램을 보완한 것이 전체의 78%(1174/1503)를 차지하고 있으며 등록된 변경 요구서에서 차지하는 비율은 약

90%(1174/(1503-196))에 육박하고 있음을 알 수 있다. 또한 다른 분류항목과 관계없이 순수하게 프로그램만을 보완하여 문제를 해결한 경우가 57%(856/1503)에 해당한다. 따라서 프로그램 작성상의 오류가 고장을 발생시키는 주요 원인이라고 할 수 있다. 그리고 운용자 입력 및 출력과 블록간의 주고받는 메시지의 오류도 고장 발생의 주요 원인으로 나타나고 있다.

< 표 1 > 변경 요구서의 보완 내역별 분류

분류항목	관련 변경 요구서 개수 (비율(%))	순수 변경 요구서 개수 (비율(%))	연관도	보완 내역
ABO	196 (13.04)	196(13.04)	0.00	중복이나 기타이유로 기각
COM	123 (8.18)	3 (0.20)	97.56	시스템 공통 파일 보완
DB	54 (3.59)	4 (0.27)	92.59	데이터의 구조 보완
DG	106 (7.05)	46 (3.06)	56.60	운용에 필요한 데이터 값 보완
FW	22 (1.46)	14 (0.93)	36.36	firmware 보완
HW	9 (0.60)	5 (0.33)	44.44	hardware 보완
IMD	92 (6.12)	14 (0.93)	84.78	운용자의 입력 명령 보완
NULL	13 (0.86)	13 (0.86)	0.00	보완없이 재시험 실시
OMD	181 (12.04)	11 (0.73)	93.92	운용자 출력 메세지 보완
OS	13 (0.86)	9 (0.60)	30.77	운영체제 보완
SIG	115 (7.65)	3 (0.20)	97.39	블록간 메시지 보완
SRC	1174 (78.11)	856(56.95)	27.09	블록 내 프로그램 보완
중복	-	329(21.89)	-	둘 이상을 동시에 보완
합계	2098	1503		

<표 1>에서 연관도란 해당 분류 항목과 다른 분류 항목과의 관련 정도를 나타내는 것으로 다음과 같이 계산되었다.

(관련 변경 요구서 개수-순수 변경 요구서 개수) / 관련 변경 요구서 개수.

연관도 분석 결과 여러 소프트웨어 블록들과 관련이 있는 공통 파일의 성격을 명확하게 보여주고 있다. 즉, 분류항목들 중 COM, DB, IMD, OMD, SIG는 대부분 90%이상 타 항목과 관련되어 해결해야 하는 문제들이다.

위의 보완 내역별 분류에서는 항목별로 중복되는 변경 요구서가 있으므로 접수된 변경 요구서를 보완 유형별로 분류하였다. 어떤 변경요구서가 SIG와 SRC를 동시에 변경한 경우 상위 Type(T2)으로 분류하였고 그 결과는 <표 2>에 정리된 바와 같다. 분류된 결과를 살펴보면 시스템의 설계에 관련된 변경 요구서와 운용 데이터와 관련된 변경 요구서는 각 각 전체의 약 3%를 차지하여 매우 안정적인 반면 소프트웨어와

관련된 변경 요구서의 비율은 전체의 80%를 차지하고 있음을 알 수 있다. 이와 같은 결과는 시스템 전체의 신뢰성은 소프트웨어의 신뢰성에 의존하며 소프트웨어의 신뢰성에 대한 중요성을 입증하고 있다.

< 표 2 > 보완 유형별 분류

유형	내용	보완 내역	관련 변경 요구서 개수 (백분율%)
T1	시스템 설계 관련	HW, FW, OS	41(27.29)
T2	SW설계 관련	DB, SIG, COM, IMD, OMD	335(22.29)
T3	SW코딩 관련	SRC	872(58.02)
T4	운용 데이터 관련	DG	46 (3.06)
T5	무보완 및 기각	NULL, ABO	209(13.91)
계			1503(100.00)

< 표 3 > 보완 유형별 변경 요구서 처리기간 분석

(단위 : 일)

유형	변경 요구서 개수	평 균	표준편차	최 소	최 대
T1	40	94	71.08	7	294
T2	318	48	61.84	7	496
T3	836	34	43.66	1	476
T4	45	40	68.89	8	335
T5	205	30	56.82	1	324
전체*	1444	38	53.73	1	496

* 처리 종료일이 표시 안된 변경 요구서가 59개 임.

그리고 변경 요구서의 해결 기간을 보완 유형별로 분류하여 보면 <표 3>과 같다. 처리 기간은 변경 요구서 발생일부터 해결 종료일까지로 계산하였다. <표 3>에서 알 수 있듯이 시스템 설계에 관련된 변경 요구서의 해결기간이 평균 94일로 다른 유형에 비하여 상대적으로 많은 시간이 소요되었는데 그 이유는 해결 부서가 조직상 타부서에 소속되어 있기 때문인 것으로 판단된다. 따라서 조직간의 원활한 협조체계가 이루어질수록 문제해결 기간이 단축될 수 있을 것이다. 그리고 문제 해결기간이 평균적으로 1개월 이상 소요되며 특히 변경 요구서를 기각하는데도 1개월이 소요되는 것은 정형화된 형상관리 체계에 따라 변경 요구서가 처리되기 때문이다.

4. 결함수 예측을 위한 통계적 모형

소프트웨어의 결함과 관련된 소프트웨어 메트릭들 중 소프트웨어의 규모를 나타내는 소스코드의 라인 수는 결함과 밀접한 관계에 있다고 알려져 있으며 Basili와 Hutchens(1983), Gremillion(1984), Musa et al.(1987)는 소프트웨어의 규모와 잠재적 결함사이의 상관계수가 일반적으로 0.5와 0.6사이에 있다고 밝혔다. 그러나 소스 코드의 라인 수에만 의존하여 소프트웨어 블록의 고장 수를 예측하는데는 한계가 있기 때문에 소프트웨어의 구조를 분석하여 복잡도에 관련된 척도들을 측정하여 예측에 이용하고 있다. 그러나 이러한 복잡도에 관련된 척도들은 본 고에서 고려하는 대형 교환 시스템을 대상으로 그 값을 측정하기가 매우 어렵다.

본 고에서는 소스 코드의 라인 수 이외에 변경 요구서를 바탕으로 수집 가능한 소프트웨어 메트릭을 이용하여 예측 모형을 개발하였다. 고려된 소프트웨어 메트릭들과 그 약어를 정리하면 다음과 같다.

LOC 블록의 소스코드 라인 수
DC 블록을 담당하고 있는 개발자들의 평균 경력(년)
M 블록과 간련된 메시지의 수
N 블록의 도입 시기

소프트웨어 블록들간에 교환되는 메시지의 수는 소프트웨어의 복잡도를 반영한 것으로 많은 메시지를 포함한 블록은 좀 더 많은 결함을 내포하고 있을 것으로 판단되어 선택하였다. 또한 경력이 풍부한 개발자가 코딩한 블록은 경력이 짧은 개발자가 담당한 블록보다 적은 결함을 포함할 것이라는 통념에 의해 블록 담당자들의 평균 근무 년수를 또 다른 독립 변수로 선택하였다. 그리고 대형 시스템을 개발하는데는 많은 기간이 요구됨으로 설계단계에 고려되었던 블록과 후에 추가된 블록간의 결함 정도는 다를 것으로 판단되어 블록의 도입 시기를 예측 요인으로 추가하였다. 도입 시기의 값은 1='설계단계부터 존재', 2='중간에 추가', 3='최근에 새로 추가' 이다.

앞에서 열거한 소프트웨어 메트릭 및 결함 수에 대한 기술 통계값들은 <표 4>에 정리된 바와 같다.

< 표 4 > 변경 요구 데이터의 기술 통계량

	평균	표준편차	최소값	최대값
결함 수	15.78	17.22	0.00	80.00
LOC	8749.36	9310.52	514.00	74325.00
DC	8.64	2.66	3.00	15.00
M	91.96	94.87	0.00	786.00
N	1.58	0.66	1.00	3.00

<표 5>는 결함 수와 고려된 소프트웨어 메트릭 간의 선형적 관계를 나타내는 상관계수 행렬이다. <표 5>에서 알 수 있듯이 블록의 크기와 메시지의 수 및 블록의 도입시기는 결함과 유의적인 관계에 있으며 그 중에서 블록의 크기가 다른 요인들보다 더 밀접한 관계에 있다. 그리고 잠재적 결함과 개발자의 경력 사이에는 예상했던 바와 달리 유의적인 관계가 존재하지 않는 것으로 나타났다. 이에 대한 원인을 면밀하게 분석해본 결과 다음과 같은 두 가지 결론을 도출하였다. 아주 잘 계획된 소프트웨어 개발 체계는 개발자들의 경력이 소프트웨어의 신뢰도에 미치는 영향을 최소화할 수 있으며 또 다른 이유는 일정 기간 이상의 경력을 지닌 개발자들은 일정수준의 코딩기술을 갖게되어 경력과 신뢰도와는 관계가 적어지게 된다.

< 표 5 > 결함 수와 소프트웨어 메트릭들과의 상관계수

(괄호안의 숫자는 p-value)

	LOC	DC	M	N
Fault	0.5994 (.0001)	0.0730 (.3932)	0.3031 (.0003)	0.2106 (.0128)
LOC		0.0444 (.6038)	0.2118 (.0123)	0.0873 (.3068)
DC			0.0342 (.6893)	0.0378 (0.6585)
M				-0.0555 (.5166)

소프트웨어 고장 자료를 수집하는 또 다른 이유는 소프트웨어의 신뢰성을 향상시키는데 있다. 소프트웨어의 고장 개수는 여러 가지 소프트웨어 메트릭에 의존하게 된다. 본 고에서는 앞에서 고려한 소프트웨어 메트릭들을 이용하여 임의의 소프트웨어 블록이 갖고있는 잠재적 결함 수를 예측하기 위하여 다음과 같은 세 가지 모형들을 고려하였다. 첫 번째 모형은 네 가지 소프트웨어 메트릭들을 독립변수로 놓고 블록의 고장 수를 종속변수로 하여 단계별 회귀 방법을 적용하였다. 단계별 회귀 방법에서는 독립변수들 중에서 결정계수(R^2)를 가장 크게하는 변수를 선택하여 이 변수에 대한 부분 F-검정을 실시하고 유의한 경우 모형에 포함시킨다. (만일 유의하지 않으면 변수 선택 절차를 중단한다.) 그리고 이미 모형에 있는 변수들에 대해 부분 F검정을 각각 실시하여 유의하지 않을 경우 모형에서 제거한다. 그리고 결함과 소프트웨어 메트릭들간의 비선형 관계의 가능성을 고려하여 Gaffney(1984)가 제안한 모형을 이용하였다. Gaffney(1984)가 제안한 모형은 다음과 같다.

$$y = a + b * (LOC)^{4/3} \quad (1)$$

여기서 y는 결함 수를 나타내는 종속변수이다. 그러나 Gaffney(1984)의 모형은 소프트웨어의 크기가 아주 작은 경우에도 y-절편항 때문에 결함이 존재하는 것으로 나타나므로 본 고에서는 이 모형을 변형하고 일반화하여 식(2)와 같은 모형을 제안하고 이를 이용하여 결함 수를 예측하고자 한다.

$$y = a * (LOC)^b \tag{2}$$

미국의 Bellcore에서는 소프트웨어의 신뢰성 향상을 위한 방법으로 포아송 회귀 모형을 고장 수 자료에 적용하여 결함 수 예측모형을 제안하고 있다. 이 포아송 회귀모형에서는 임의의 소프트웨어 블록에 존재하는 결함 수는 평균이 μ_i인 포아송 분포를 따르며 식(3)과 같은 확률질량함수를 갖는다.

$$P(Y_i = y_i) = \exp(-\mu_i)\ \mu_i^{y_i} / y_i!, \quad i = 1, 2, \ldots, n \tag{3}$$

여기서 Y_i는 i-번째 블록의 결함 수를 나타내는 확률변수이다. 그리고 평균 μ_i는 소프트웨어 메트릭들과 다음과 같은 관계에 있다.

$$\log \mu_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \tag{4}$$

여기서 독립변수 $x_{i1}, x_{i2}, \ldots, x_{ip}$는 소프트웨어 메트릭들이며 $\beta_0, \beta_1, \ldots, \beta_p$는 추정되어야 할 모수들이다. 식(4)에서 i-번째 블록의 결함 수에 관한 다음과 같은 식을 얻는다.

$$\mu_i = \exp[\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}], \quad i = 1, 2, \ldots, n \tag{5}$$

그리고 포아송 회귀모형에서 만일 독립변수 LOC하나만을 이용한다면 이 모형은 Gaffney 모형의 변형된 형태인 식(2)로 축소됨을 알 수 있다. <표 7>의 결과에서 알 수 있듯이 단계별 회귀 방법을 이용하여 얻은 모형은 블록의 크기(LOC), 메시지의 수(M) 구리고 도입 시기(N)를 포함하고 있다. 포아송 회귀 모형은 단계별 회귀 모형과 비교하기 위하여 동일한 메트릭들을 독립변수로 이용하였다.

앞에서 설명한 세 종류의 예측모형을 요약 정리하면 다음과 같다.

모형 1　단계별 회귀 방법을 적용한 모형
모형 2　LOC를 독립변수로 한 Gaffney 모형의 일반화된 모형
모형 3　LOC, M, N을 독립변수로 하는 포아송 회귀 모형

모형에 대한 분산분석 결과는 <표 6>에 정리된 바와 같으며 세 모형 모두 유의함을 알 수 있다. 그리고 <표 7>은 각 모형의 모수들에 대한 추정치와 모형의 결정계수 및 수정 결정계수를 보여주고 있다. 결정계수를 바탕으로 세 모형들의 적합성을 비교해보면 모형 2와 3인 모형 1에 비해 우수한 것으로 나타났다. 즉, 소프트웨어의 결함 수와 고려된 소프트웨어 메트릭들 사이에는 비선형적인 관계가 더 적합하다고 할 수 있다. 그리고 비선형 모형 중에서는 포아송 회귀 모형이 조금 더 적합하게 나타났다. 또한 모형 3의 경우 독립변수들의 값이 0인 경우 결함수는 0에 아주 근사하므로 합리적인 모형임을 알 수 있다.

< 표 6 > 예측모형에 대한 분산분석표

모형	요 인	자유도	자승합	평균자승합	F-value	p-value
1	회 귀	3	17256.74	5752.25	32.82	< 0.0001
	잔 차	135	23662.79	175.28		
	계	138	40919.53			
2	회 귀	2	51375.03	25687.51	145.57	< 0.0001
	잔 차	137	24174.97	176.46		
	계	139	75550.00			
3	회 귀	4	54530.11	13632.53	86.37	< 0.0001
	잔 차	128	20201.89	157.83		
	계	132	74732.00			

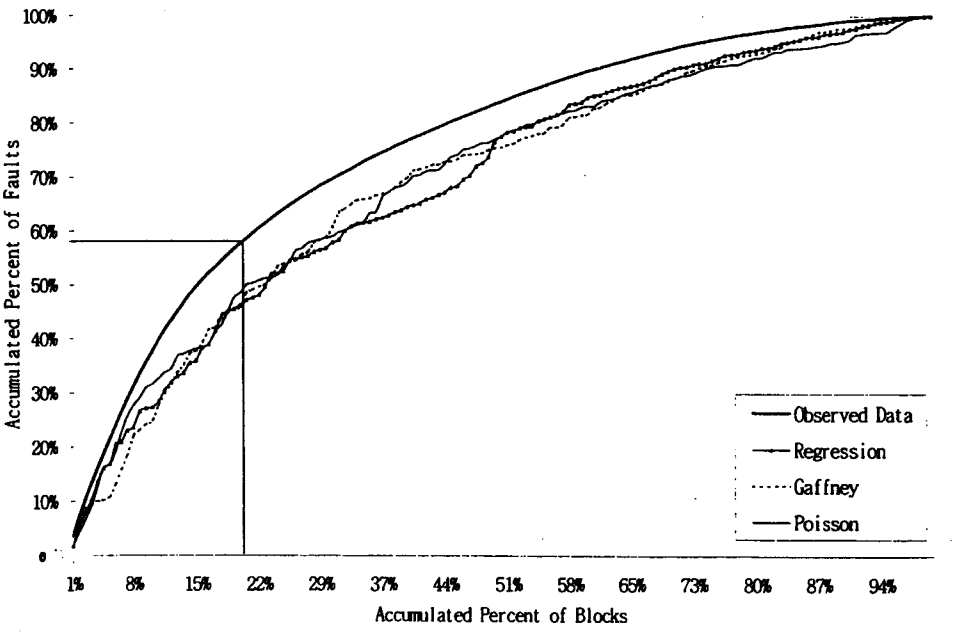
< 표 7 > 모수들의 추정치와 각 모형들의 적합도

모 형	모 수	추정치	표준오차	R^2	$adjR^2$
모형 1	y-절편	-3.5096	3.2366	0.4217	0.4089
	LOC	0.0010	0.0001		
	M	0.0359	0.0122		
	N	4.5586	1.7236		
모형 2	a	0.0413	0.0271	0.6800	0.6753
	b	0.6674	0.0672		
모형 3	β_0	-3.5340	0.6487	0.7297	0.7212
	β_1 (LOC)	0.5783	0.0665		
	β_2 (M)	0.2279	0.0811		
	β_3 (N)	0.4091	0.1563		

모형의 예측 정확성를 평가하는 방법으로 Alberg, Johansson과 Ohlson(1993)이 제안한 Alberg 다이어그램을 이용하였다. Alberg 다이어그램은 예측 모형들의 정확성 및 유용성을 평가하고 비교하는데 이용되고 있는 방법이다. 이 방법은 자료들의 정규

성을 가정하지 않고도 사용될 수 있기 때문에 일반적으로 많이 치우쳐있고 이상치들이 많이 있는 소프트웨어의 고장 자료에 특히 적합하다. 이 방법은 관찰된 결함 수에 따라 정렬된 블록의 누적 백분율과 관찰된 결함 수 및 예측된 결함 수의 누적 백분율을 이용하여 쉽게 작성할 수 있다. Alberg다이어그램의 작성에 관한 자세한 내용은 Johansson과 Nord(1995)를 참고하기 바란다. Alberg 다이어그램에서 관찰된 결함 수를 바탕으로 작성된 곡선과 예측된 결함 수를 바탕으로 한 곡선이 가까울수록 모형의 예측 정확성이 높다고 할 수 있다.

<그림 2>는 완성된 Alberg 다이어그램을 보여주고 있으며 X축은 블록의 누적 백분율을 나타내고 Y축은 고장 수의 누적 백분율을 나타내고 있다. 가장 위에 있는 곡선은 관찰값을 바탕으로 작성된 것이며 아래의 세 곡선은 세 개의 모형을 바탕으로 작성된 것이다. <그림 2>에서 알 수 있듯이 세 개의 모형간에는 큰 차이가 없으나 모형 3 (포아송 회귀모형)이 근소하게 우위를 유지하고 있다. <그림 2>의 관찰값을 바탕으로 작성된 그래프에서 전체 블록의 약 20%에 해당하는 다결함 블록 28개를 선별하면 전체 고장의 약 58%가 선별된 블록에서 발생했음을 알 수 있다. 그리고 세 종류의 예측 모형을 바탕으로 상위 20%에 속하는 다결함 블록들을 뽑아 관찰값을 바탕으로 뽑은 상위 20% 다결함 블록과 비교를 한 결과가 모형 1과 모형 2는 64%, 모형 3은 68%가 일치하였다. 또한 모형 3의 경우 상위 20%에 속하는 블록이 전체 결함의 48%를 포함하고 있음을 알 수 있다. 그리고 상위 10% 및 30% 다결함 블록에 대해 분석한 결과는 <표 8>에 나타난 바와 같다.



< 그림 2 > Alberg 다이어그램

< 표 8 > 관찰된 다결함 블록과 예측된 다결함 블록간 비교

		모형 1	모형 2	모형 3	관찰값
상위 10% 다결함 블록	일치하는 블록 개수 (백분율)	8(57%)	6(42%)	9(64%)	14
	다결함 블록에 포함된 결함 수 (백분율)	507(23%)	396(18%)	553(25%)	828(38%)
상위 20% 다결함 블록	일치하는 블록 개수 (백분율)	18(64%)	18(64%)	19(68%)	28
	다결함 블록에 포함된 결함 수 (백분율)	1013(46%)	1005(46%)	1063(48%)	1264(58%)
상위 30% 다결함 블록	일치하는 블록 개수 (백분율)	26(62%)	25(60%)	28(67%)	42
	다결함 블록에 포함된 결함 수 (백분율)	1247(57%)	1291(59%)	1288(59%)	1514(69%)

5. 결론

본 연구에서는 소스 코드의 규모가 100만 라인 이상인 대규모 통신 소프트웨어의 신뢰성을 개발 초기에 평가하기 위하여 소프트웨어의 블록이 포함하고 있는 결함 수를 예측하는 방법에 관하여 연구하였다. 기존에 제안된 예측 방법들은 소프트웨어의 분석을 통해 복잡도 등과 같은 자료를 이용하였으나 대규모 소프트웨어의 경우 이러한 자료를 수집하는데 많은 시간과 자원을 투입해야 하므로 현실적으로 적용하기가 어렵다고 할 수 있다. 본 고에서 제안된 예측 방법은 필요한 자료의 수집이 용이할 뿐 만 아니라 모형의 예측 정확성도 일정한 수준이므로 현장에 적용하기에 적합할 것이다.

이러한 예측 모형은 새로운 소프트웨어를 개발하는 경우 개발중인 소프트웨어의 신뢰성을 향상시키는 방법으로 다음과 같이 사용될 수 있다. 개발중인 소프트웨어로부터 블록의 결함 수 예측에 필요한 자료를 수집한 후, 이 자료를 바탕으로 예측 모형을 이용하면 각 블록들의 결함 수를 예측할 수 있다. 프로젝트 관리자는 예측된 블록의 결함 수를 바탕으로 다결함 블록을 선정하고 가장 경험이 많은 숙련자에게 선정된 블록을 담당하게 하거나 또는 소프트웨어의 시험에 엄격한 검사기준을 적용하게 함으로써 소프트웨어의 신뢰성을 향상시킬 수 있다.

본 연구에서 제안한 세 가지 모형들의 적합성과 예측 정확도를 비교해 본 결과 포아송 회귀 모형이 다결함 블록을 예측하는 정확도가 약 70%로서 가장 우수한 것으로 나타났으나 보다 정확한 모형의 개발을 위한 연구가 요구되고 있다. 이를 위하여 고려하고 있는 소프트웨어의 기능을 분류하고 각 기능에 관련된 소프트웨어 블록들을 분석대상으로 하여 예측 모형을 개발하는 경우 정확도가 향상될 것으로 판단되며 이는 향후 연구과제로 남겨둔다.

참고문헌

[1] 박성현(1984), 「회귀분석」, 대영사.

[2] Basili, V. R. and Hutchens, D. H.(1983), "An Empirical Study of a Syntactic Complexity Family," *IEEE Tr. on Software Engineering*, Vol. SE-9, pp. 664-672.

[3] Bellcore(1990), *The Analysis and Use of Software Reliability and Quality Data*, TR-TSY-0001547, pp.13-25.

[4] Craw, S., McIntosh, A. and Pregibon, D.(1985), "An Analysis of Static Metrics and Faults in C Software," *The Journal of Systems and Software*, Vol. 5, pp. 37-48.

[5] Gaffney, J. E.(1984), "Estimating the Number of Faults on Code," *IEEE Tr. on Software Engineering*, Vol. SE-10, pp. 459-464.

[6] Gremillion, L. L.(1984), "Determinants of Program Repair Maintenance Requirements," *Communications ACM*, Vol. 27, No. 8, pp. 826-832.

[7] Johansson, O. and Nord, C.(1995), "Using Predictions Improve Software Reliability," *Ericsson Review*, No. 1, pp. 30-35.

[8] Healy, J. D., Jain, A. K. and Bennett, J. M.(1996), "Reliability Pediction," 1996 *Annual Reliability and Maintainability Symposium*, pp. 1-16.

[9] Henry, S. and Kafura, D.(1981), "Software Structure Metrics Based On Information Flow," *IEEE Tr. on Software Engineering*, Vol. SE-7, pp. 510-518.

[10] Karunanithi, N and Malaiya, Y. K.(1995), "Neural Network for Software Reliabiity Engineering," *Handbook of Software Reliability Engineering*, Edited by M. R. Lyu, McGraw-Hill, New-York.

[11] Khoshgoftaar, T. M. and Munson, J. C.(1990), "Predicting Software Development Error Using Software Complexity Metrics," *IEEE Journal of Selected Area in Communication*, Vol. 8, No. 5, pp. 253-261.

[12] Lyu, M. R., Yu, J., Keramidas, E. and Dalal, S.(1995), "ARMOR: Analyzer for Reducing Module Operational Risk," *Proceedings of the 25th International Symposium on Fault Prone Tolerant Computing (FTCS-25)*, Pasadena, Calif., pp. 137-142.

[13] McCabe, T.(1976), "A Complexity Measure," *IEEE Tr. on Software Engineering*, Vol. SE-2, pp. 308-320.

[14] Munson, J. C. and Khoshgoftaar, T. M.(1992), "The Detection of Fault-Prone Programs," *IEEE Transaction on Software Engineering*, Vol. SE-18, pp. 423-433.

[15] Musa, J. D., Iannino, A. and Okumoto, K.(1987), *Software Reliability; Measurement, Prediction, Application*, McGraw-Hill, New York.

[16] Porter, A. A. and Selby, R. W.(1990), "Empirically Guided Software Development Using Metric-Based Classification Trees," *IEEE Software*, pp. 46-54.

[17] Rodriguez, V. and Tsai, W.(1987), "A Tool for Discriminant Analysis and Classification of Software Metrics," *Information and Software Technology*, Vol. 29, No. 3, pp. 137-149.