예측모델을 이용한 소프트웨어 컴포넌트 결함 분석과 신뢰도 평가

Evoluation of the Reliability Prediction and Classification of Fault-Prone Software Components Using the Prediction Model

이재기(J.K. Lee) 네트워크시험팀 책임연구원
이경호(K.H. Lee) 네트워크시험팀 책임연구원, 팀장
박권철(K.C. Park) 네트워크전략연구부 책임연구원, 부장

소프트웨어 품질은 시스템의 소프트웨어에 많은 영향을 준다. 특히 대형 소프트웨어 시스템에 고장이 발생하게 되면 소프트웨어에 심각한 문제가 발생하게 되며, 커다란 손실과 함께 이를 유지보수하는 데에도 많은 비용을 지출해야 한다. 이러한 문제들을 조기에 발견, 조치함으로써 안정성과 높은 신뢰도를 갖는 고품질의 소프트웨어를 확보할 수 있게 된다. 본 논문에서는 이런 관점에서 개발중인 소프트웨어나 기 개발된 소프트웨어 컴포넌트에 대해 다각도로 분석하고 컴포넌트에 대한 fault-prone 판단을 수행하고 예측 모델을 이용하여 소프트웨어 신뢰도를 측정한다.

I. 서 론

오늘날 사용자의 다양한 서비스 요구와 고품질의 서비스를 제공 받기를 원하는 욕구를 만족시키기 위해서는 다양한 응용 프로그램의 수용과 이를 실시간(real time)으로 처리하는 고도의 기술(high-technique)이 필요하게 되었다.

이러한 요구사항에 따라 시스템 소프트웨어의 규모가 커지고 그에 따른 많은 결함도 발생하게 되었다. 특히 대형시스템의 소프트웨어에 오류가 발생하게 되면 커다란 경제적인 손실은 물론 시스템 전체에 치명적인 에러를 유발하게 된다. 이러한 문제점을 해결하기 위해서는 고품질의 소프트웨어 개발과 함께 고 신뢰성을 갖는 개발방법이 필요하다.

이러한 관점에서 대형시스템의 개발시 구현되고 있는 소프트웨어 컴포넌트의 결함을 다각도로 분석하여 고 신뢰도를 갖는 소프트웨어 개발 방안을 모색한다.

본 논문의 구성은 II장에서 소프트웨어 신뢰도 예측방법을 위해 시도된 여러 가지 방안과 소프트웨어 컴포넌트에 대한 결함검출 예측모델을 이용하여 분석한 결과를 살펴보고 III장에서 예측된 데이터를 이용해 소프트웨어 신뢰도를 측정하고, IV장에서 향후 연구방향 및 결론을 제시한다.

II. Fault-Prone 예측방법 및 수행 결과

신뢰도 예측을 위해 시도한 방법은 시험에서 검출된 결함데이터를 분석하고 각 컴포넌트에 대한 결함발생률을 측정하여 이를 예측모델에 적용하여 추정하는 방법을 취했다. 그리고 다각도로 컴포넌트의

결함을 분석하고 신뢰도성장모델과 비교, 분석하는 방법을 택했다.

단계별 고장이 확인된 컴포넌트 수는 <표 1>과 같다.

<표 1> 단계별 Fault-Prone 컴포넌트 수

	개발	시험	배포
Basic defect cohesion	30	30	12
Multi-file defect cohesion	63	43	5
Both combine	12	6	4

고장의 결합 상태별로 살펴보면 개발 및 시험단계에서는 multi-file 형태의 고장을 포함하는 컴포넌트가 단일고장(basic defect)을 포함하는 컴포넌트보다 월등히 많다. 그 이유는 개발 단계별로 새로운 기능이 추가되면서 인접 기능과의 연동이 많아지기 때문이다.

Both combine 성격의 결함은 시스템 공통으로 사용되는 컴포넌트들과 함께 변경되어야 하는 결함(defect)으로 헤더(hdr) 정보나 미들웨어(middle-ware), 입·출력 메시지(i/o msg), 시그널(sig), 릴레이션(rel), 모드(mode), 도메인(domain), 시스템 라이브러리(sylib) 및 망관리(TMN) 관련 컴포넌트 등이 이에 속한다.

시스템 개발 단계별 적용된 소프트웨어 컴포넌트는 <표 2>와 같으며, TMN 및 AAL-1/2, Gateway call control 기능 등 12개 블록이 별도의 시험과정을 거쳐 배포되었다(시험환경 구축 및 개발일정 필요).

<표 2> 개발 단계별 컴포넌트 적용 현황

	개발(D)	시험(T)	배포(R)
컴포넌트 수	94	99	111

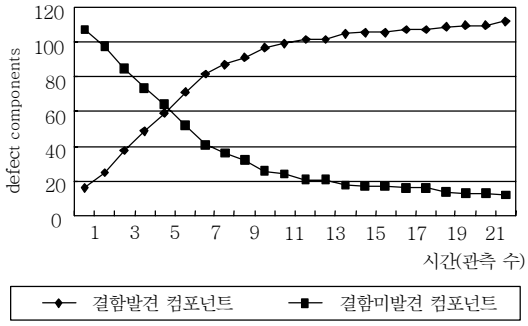
발생된 결함을 수정(fixed)하기 위해 변경된 소프트웨어 컴포넌트 현황은 <표 3>과 같다. 일부 결함을 해결하기 위해서는 변경할 컴포넌트와 시스템 공통으로 사용하는 컴포넌트를 함께 변경하는데 이는 미들웨어 사용과 밀접한 관계가 있기 때문이다.

이에 대한 데이터는 both combine 형태의 컴포넌트 변경 수와 밀접한 관계가 있다.

<표 3> 컴포넌트 배포 횟수 – Fault-Prone

	Number of fault-prone	무변경	1회	2회	3회
Number of components	Basic defect cohesion	45	48	13	0
	Multi-file defect cohesion	37	43	28	3
	Both combine	113	3	3	4

결함 번호별 분류에 의한 방법을 적용시 관계되는 소프트웨어 컴포넌트(여기서는 하나의 functional block(fb)을 의미) 추이는 (그림 1)과 같다. 이 방법은 결함 검출을 시간상으로 분류한 방법으로서 시험 시나리오와 순서, 우선순위가 고려된 시험방법 및 개발 스케줄 등에 관계가 있다.



(그림 1) 컴포넌트 결함발견/미발견 추세

시스템 특성상 초기에는 시스템의 근간(根幹)이 되는 OS와 DBMS 등이 집중 수행되고 그 다음이 호제어, 운용, 보전기능 순서에 대해 수행됨으로써 문제가 발생되는 컴포넌트도 초기에는 주로 OS와 DBMS, 시스템 형상관리, 호제어 기능 등에 많은 결함이 발견, 수정되고 있다.

1. 소프트웨어 시스템별 예측 모델 구축

<표 4>는 시스템 개발 및 시험기간동안 결함이 발견된 컴포넌트와 미발견된 컴포넌트 등 2개의 범주 분류(two-category classification)에 의한 방법으로 결함이 검출된 컴포넌트에 대해 fault가 발견된 것과 발견되지 않은 컴포넌트로 분류한 결과이다.

<표 4> Two-Category Classification에 의한 매트릭스

		Prediction results	
		Fault-free	Fault-prone
Actual results	Fault-free	C_{11}	C_{12}
	Fault-prone	C_{21}	C_{22}

이때 적용되는 정확성 및 정밀성(도), 완전성에 대한 산술식은 (1)과 같다.

$$\begin{aligned} Accuracy(A) &: \left[\frac{C_{11}+C_{22}}{C_{11}+C_{12}+C_{21}+C_{22}}\right]\times 100(\%) \\ Correctness(C) &: \left[\frac{C_{22}}{C_{12}+C_{22}}\right]\times 100(\%) \\ Completeness(Comp) &: \left[\frac{C_{22}}{C_{21}+C_{22}}\right]\times 100(\%) \end{aligned} \quad (1)$$

2. 예측 실행결과 평가

예측 평가 방법은 <표 4>에 언급한 것과 같이 4개의 범주로 분류하고 예측된 결과에 대한 정확성, 정밀도, 완전성을 고려한다.

- Accuracy(A) - 정확성

이 비율은 소프트웨어 컴포넌트의 fault-prone 예측의 정확성을 의미한다.

- Correctness(C) - 정밀성

이 비율은 컴포넌트에 적어도 하나 이상의 결함을 가지고 있는 fault-prone 모듈을 의미한다. 이 값이 낮은 경우 더 많은 fault-prone 모듈을 예측할 수 있다.

- Completeness(CP) - 완전성

하나 이상의 결함을 가지고 있는 fault-prone 모듈로 이 값이 적은 경우는 실제 fault-free로 예측된 모듈에 대해 잘못 예측될 가능성이 많음을 의미한다.

3. Constructed Model에 의한 Fault-Prone 예측

서로 다른 시스템의 소프트웨어 모듈에 대한 예측 효과를 측정하기 위해 <표 4>와 같이 구축된 예측모델을 적용하여 소프트웨어 컴포넌트(또는 모듈; 기능블록)에 대한 fault-free와 fault-prone을 조사하였다.

소프트웨어 컴포넌트에 대한 예측 실험에 적용된 2개 시스템에 대한 분석 결과는 <표 5>와 같다.

<표 5> 시스템별 소프트웨어 컴포넌트 Fault-Free, Fault-Prone 예측 결과

		Prediction results(system 1)		Prediction results(system 2)	
		Fault-free	Fault-prone	Fault-free	Fault-prone
Actual results	Fault-free	3	24	45	61
	Fault-prone	12	47	37	74

4. 결함 데이터 수집과 모듈 정보 수집

결함데이터 수집은 고장보고에 의해 작성된 문제점을 검토하여 결함으로 판단되는 데이터를 자동화된 툴로 관리하며 defect data tracking system(ddts)의 메트릭스 정보를 이용, 결함이 검출된 컴포넌트에 대한 결과는 <표 5>와 같이 분석되었고 이 정보를 이용해 식 (1)에 의거 산출된 결과는 <표 6>과 같다. <표 6>과 같이 추출하여 신뢰도 모델을 이용하여 시스템에 실장되는 소프트웨어 컴포넌트의 신뢰도를 측정하는 기초 데이터로 활용한다.

<표 6> 2개의 프로젝트에 대한 결과 분석

	System 1	System 2
Accuracy(A)	58.14%	54.84%
Correctness(C)	66.20%	54.81%
Completeness(CP)	79.66%	66.67%

이러한 메트릭스 방법을 이용하여 신뢰도를 측정하는 방법이 많이 연구되어 왔는데[1],[2] 이러한 종류의 예측 방법 중 대표적인 것은 다각도로 소프트웨어 모듈의 규모(size)나 software science metrics를 이용한 복잡도(complexity)를 측정하는 방법[3]과 McCabe의 심리적 복잡도(psychological complexity; cyclomatic number measurement) 메트릭스 분석[4], 모듈의 cohesion 메트릭스 정보를 이용하는 방법[5], 컴포넌트의 결함 수를 이용하

여 측정하는 방법[1],[2],[6],[7], 프로젝트 수행에 관련된 제어 구조나 program volume, nesting depth 등 다양한 복합 메트릭스를 이용하는 방법[8],[9] 등이 있다.

예측 정확성(A)의 값은 시스템에 실장된 컴포넌트에 대해 결함 발견 및 미발견 비율을 의미하며, 예측된 값의 정밀도(C)의 값은 1개 이상의 fault를 감지한 fault-prone 컴포넌트 비율을 말한다. 이 값이 적으면 실제 에러가 없는 fault-prone 컴포넌트 발생 확률이 더 많이 존재하고 있음을 의미한다. 또 결함 예측 완전성(CP)은 정밀도와 같은 상황이나 그 의미는 fault-free로서 더 많은 fault-prone 컴포넌트가 잘못 예측될 가능성이 많다는 것을 의미한다.

그 밖에 구현된 소프트웨어 컴포넌트의 각 기능 그룹별 예측자(predictor)의 실행 결과는 호처리가 comment line(C_line), 구조문(ST_), function point 수(F_), 운용·보전 기능은 소스라인 수(SLOC)와 comment line, function point 수(F_)가 가장 유효한 것으로 분석되었다.

<표 7> 기능 그룹별 소프트웨어 컴포넌트 예측자 상관관계

Call processing	
SLOC	0.619
C_line	0.882
F_	0.996
CONST_	0.538
ST_	0.842
Administration	
SLOC	0.738
C_line	0.927
F_	0.971
G_VAL	0.619
ST_	-
Operation & Maintenance	
SLOC	0.746
C_line	0.856
F_	0.896
G_VAL	0.318
ST_	-

대표적인 5개 컴포넌트는 <표 7>에 언급한 것과 같이 global 변수(G_VAL)가 포함되었다.

5. 2개 시스템 현황 및 소프트웨어 규모

예측모델에 의해 추정된 시스템은 2개의 시스템으로 서로 다른 환경의 소프트웨어로 사용된 언어 및 시스템 구조 등이 다른 시스템이다. 시스템 1이 하나의 시스템에서 모든 기능을 수용, 제어하는 반면에 시스템 2는 분산구조로 객체지향개념을 도입하고 미들웨어를 채택하여 다양한 개발환경에서 개발된 소프트웨어를 하나의 통일된 OS 환경에 맞게 동작하도록 시도하고 시스템과 워크스테이션간의 기능을 분산 수용하여 처리하도록 한 것이 주요 차이점이다(<표 8> 참조).

<표 8> 시스템 개발 현황 및 소프트웨어 소스 규모

	System 1	System 2
Software platform	Main frame	Main frame
Size	131만 SLOC (136 modules)	252만 SLOC (123 modules)
Language	CHILL	C/C++, java
Fault data	System integration & test	System integration & test

III. 결함데이터 수집 및 신뢰도 예측 결과

III장에서는 II장에서 수집된 결과를 토대로 결함이 발견된 소프트웨어 컴포넌트와 관련된 데이터를 활용해 소프트웨어 신뢰도를 예측해 본다.

우선 결함발견 컴포넌트 수정을 위해 제기된 결함데이터(혹은 고장데이터)를 대표적인 2개의 신뢰도 측정모델에 적용하여 결과를 분석하고 고려할 사항을 언급한다.

1. 모델 파라미터 추정 및 실행결과

<표 9>의 모델 파라미터 추정 결과는 <표 8>에

언급한 system 2 에 대해 시험에서 검출된 고장데이터를 토대로 소프트웨어 결함을 추정한 것이다. 시스템 시험에서 검출된 결함데이터를 대표적인 소프트웨어 신뢰도성장모델(Software Reliability Growth Model: SRGM)인 지수형 모델(exponential model)과 S-자형모델(S-shaped model)에 적용한 결과로 소프트웨어 내에 잔존하고있는 결함 수(a) 및 고장발생률(b), 순간에러발견율[h(t)]에 대한 추정치 및 실측치에 대한 비교는 <표 9>와 같다.

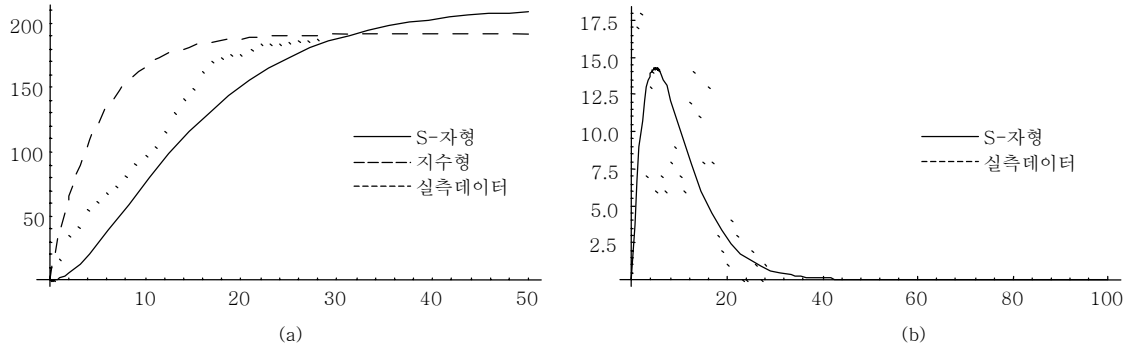
<표 9> 소프트웨어 결함데이터 추정치

적용 모델	a	b	h(t) (t=20주)	실측치
지수형	172.664	0.0848809	4.5	189
S-자형	187.635	0.203143	2.5	

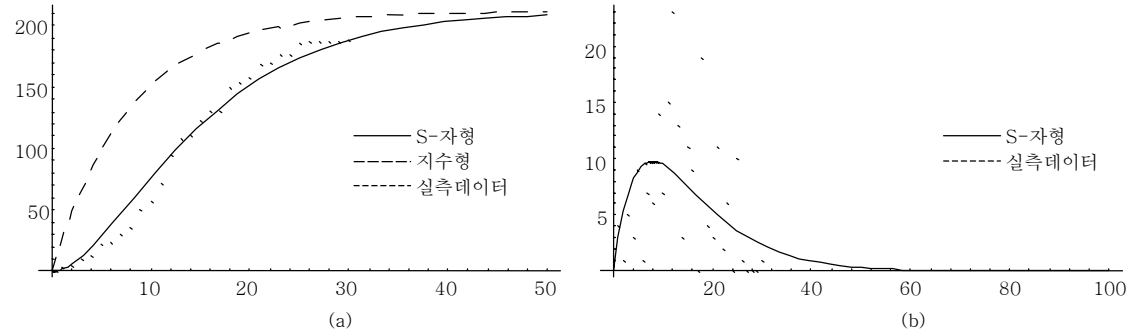
(그림 2), (그림 3)은 검출된 결함을 해결한 시점과 발견과 함께 결함이 수정된 것으로 가정한 결과에 대한 실측치, 2개 모델의 추정치를 나타낸 것이다. 좌측 그림의 실선은 S-자형, 대시 라인은 지수형, 점선은 실측데이터를 표시하며, 우측 그림의 점선은 실측 데이터를 의미한다.

(그림 2), (그림 3)의 우측 그림은 지연(delayed) S-자형 모델에 의한 평균치 함수 h(t)로 시간(t=week)에 따른 에러 발생율에 대한 추정치를 의미한다. 시험에서 검출되거나 검출된 결함을 해결한 시점에서 분석한 선형회귀방정식 및 분산분석(Analysis of variance: ANOVA), 모델 파라미터들에 대한 추정결과는 <표 10>과 같다.

추정결과에서 알 수 있는 것과 같이 결함 데이터를 적용한 경우는 가설 검정통계량(test statistic) 데이터 값(p-value)이 유의수준(α=0.05, 신뢰구간 95%)을 벗어나 가설검정(null hypothesis)이 성립되지 않는다. 그러므로 검출 및 해결된 결함 데이터는 선형(linear) 특성에 적합하지 않으며, 비선형(non-linear) 특성을 따른다. 이와 같이 비선형 특성을 갖는 결함 데이터를 문제점 해결시점을 기준으로



(그림 2) 주별 결함발견 기준 (a) 누적결함추정치 및 (b) 에러 발생률



(그림 3) 주별 결함해결 기준 (a) 누적결함추정치 및 (b) 에러 발생률

<표 10> 모델 파라미터 추정 결과 및 분산 분석

1) 결함해결 데이터 기준(linear regression function: $f_1(x) = 13.6667 - 0.481938x$)

[Parameter Table]

	Estimate	SE	TStat	PValue	RSquared	AdjustedRSquared	EstimatedVariance
1	8.88046	2.24232	3.96038	0.000467256	0.0598497	0.0262729	35.8555
x	-0.168632	0.126307	-1.33509	0.1926			

2) 결함발견 데이터 기준(linear regression function: $f_2(x) = 7.81766 - 0.0610501x$)

[Parameter Table]

	Estimate	SE	TStat	PValue	RSquared	AdjustedRSquared	EstimatedVariance
1	13.6667	1.34467	10.1636	0	0.576473	0.560184	11.9908
x	-0.481938	0.0810131	-5.94889	0			

3) 결함데이터에 대한 분산 분석(ANOVA) 결과

[ANOVA Table-1]

	DoF	SoS	MeanSS	FRatio	PValue
Model	1	63.9115	63.9115	1.78247	0.1926
Error	28	1003.96	35.8555		
Total	29	1067.87			

[ANOVA Table-2]

	DoF	SoS	MeanSS	FRatio	PValue
Model	1	424.346	424.346	35.3893	0
Error	29	311.761	11.9908		
Total	30	736.107			

주) ANOVA Table-1, 2는 결함해결 및 발견 데이터에 대한 분석 결과임

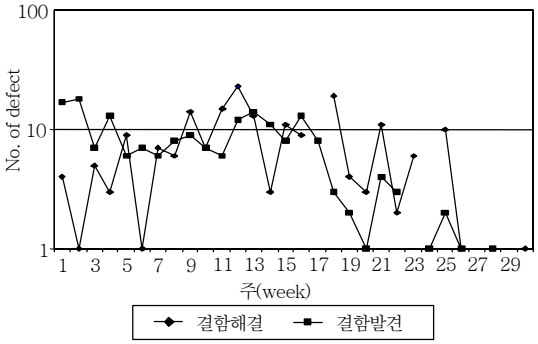
한 결함 데이터를 지수형 및 S-자형 모델에 적용하였을 때 신뢰도를 예측한 결과가 실제 프로젝트 수행 결과와 가장 근접함을 알 수 있다((그림 5)의 curve fitting 결과 참조).

이에 대한 예측 결과들은 (그림 5), (그림 6)에 표시하였다.

2. 결함 발견 및 해결 추이

(그림 4)는 매주(per week) 시스템 시험에서 검출된 소프트웨어 컴포넌트의 결함 발견 및 해결에 대한 시간 추이를 그린 것이다.

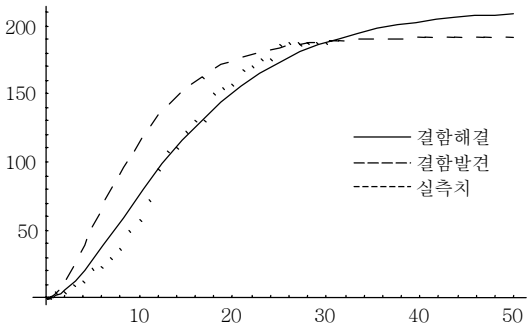
검출된 데이터를 살펴보면 시험시작 후 17주 및 24주 근처에서 소프트웨어 결함 발견과 해결이 일시적으로 이루어지지 않는 이유는 이 시기에 일시 시험을 중단하고 결과를 정리하여 공동개발 기업체로의 소프트웨어 배포(post-release)가 이루어졌기 때문이었다. 배포 이후 결함 발견은 급속히 줄어들고 결함해결은 많이 이루어졌는데 이 시기에 소프트웨어 배포시기에 맞추어 안정화(디버깅 작업)가 집중적으로 시도됨을 의미한다.



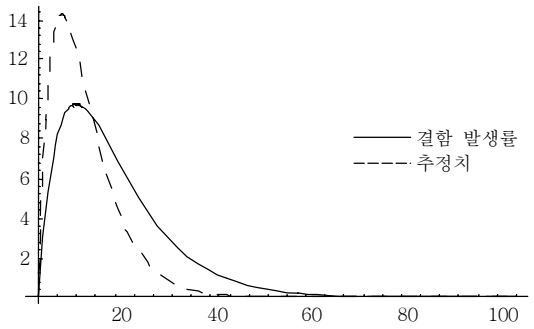
(그림 4) 주별 결함 발견/해결 추이

(그림 5)는 소프트웨어 결함 발견 및 해결을 S-자형 모델로 추정한 결과치를 비교한 것이다. (실선은 결함 해결, 점선은 실측치) 결과를 놓고 볼 때 결함을 발견한 시점을 기준으로 신뢰도를 추정한 것보다는 해결한 시점을 기준한 것이 실제 상황과 더 밀접하다. 이것은 결함이 검출되면 이에 대한 분석과정과 해결과정을 거쳐 최종 검증을 받은 후 결함이 종료되기 때문이다.

(그림 6)의 실선은 S-자형 모델에 대한 결함 해결 데이터를 적용하였을 때 추정한 결함 발생률이다.



(그림 5) 결함 발견 및 해결 추정치 비교



(그림 6) 결함 발생률(발견 대 해결) 비교

Dash line은 결함 발견 데이터를 적용하는 경우에 추정된 결과로서 시험에서 발견되는 결함발견율(b)에 따라 차이가 난다.

(그림 5)의 추정치와 비교해 볼 때 프로젝트에 적합한 모델 파라미터 추정치는 결함을 해결한 시점을 기준으로 삼아 시험시작 30주 전후에서 시스템의 신뢰도가 목표치의 95%에 이를 것으로 사료된다.

<표 11> 잔존 결함 수 및 결함 발생률 추정치

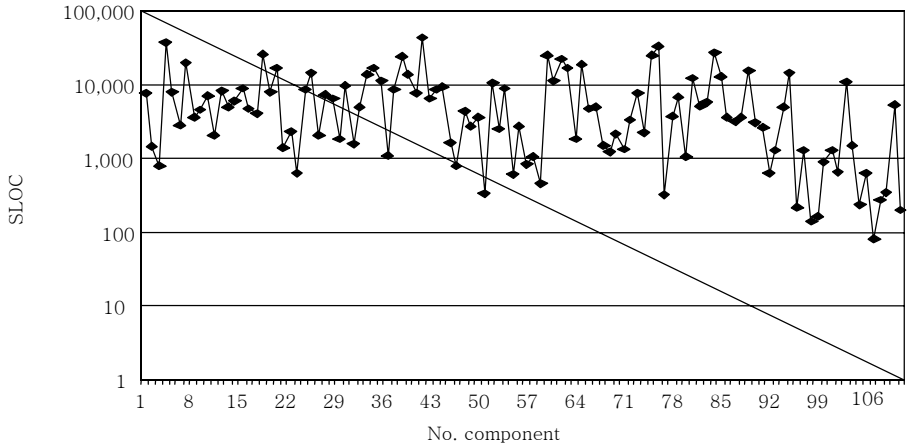
	a	b	h(t=20주)
결함발견	191.365	0.202842	7
결함해결	211.223	0.125672	3

<표 11>은 (그림 5), (그림 6)에 나타낸 추정치를 종합 정리한 것으로 주당 최대 결함 발견(발생)율은 14개(결함 발견 데이터 기준), 10개(해결 데이터 기준)로 나타났다. 또 소프트웨어 내에 잔존(殘存) 결함 수는 191개, 211개로 추정되고 시험으로 검출할 수 있는 결함 검출비는 0.2, 0.13으로 각각 추정되었다.

<표 11>의 모델 파라미터 추정치와 실측치를 비교해 볼 때 소프트웨어 내에 잔존하고 있는 결함을 미 발견한 데이터가 존재하는데(2~12개) 이것은 운용상에 크게 지장이 없는 보이지 않는 결함, 즉 latent fault로 추정된다. 이러한 결함들은 반드시 제거되어야 시스템 운용에 지장이 없게 된다.

3. 소프트웨어 라인 당 결함 수 추정

소프트웨어 라인 당 평균 결함 발생은 1.28개/KLOC로 밝혀졌으며 각 컴포넌트들에 대한 라인 규모는 (그림 7)과 같다. 컴포넌트 당 평균라인 규모는 약 7,900라인 정도로 밝혀졌다. 최대는 PNNI 라우팅



(그림 7) 소프트웨어 컴포넌트 별 소스라인 규모

처리 관련 기능으로 44,300라인이고 최소는 장애처리 관련 라이브러리(library)로서 수 십 라인인 것으로 분석되었다.

IV. 결 론

시스템 개발에 있어서 소프트웨어 컴포넌트의 결함을 조기에 탐지하여 수정, 조치하면 시스템의 신뢰도는 물론 소프트웨어 품질이 향상되며, 개발 비용도 줄일 수 있다. 다시 말해서 초기에 문제가 집중 발생되는 모듈 또는 컴포넌트를 발견하고 또 이런 문제들을 조기에 발견할 수 있는 방안을 모색하는 것은 시스템 개발자나 관리자 모두에게 관심사항이다.

앞장에서 분석한 결과와 마찬가지로 예측 모델을 이용한 소프트웨어 신뢰도 예측 방법은 다양하다. 즉, 시스템에 구현된 각 기능을 수행시키는 소프트웨어 컴포넌트에 대한 결함발생률을 추정하여 이를 서브시스템 레벨로 추정하는 방법 및 시스템에 구현된 소프트웨어의 규모에 대해 발생되는 결함을 추적하여 추정하는 방법, 그밖에 소프트웨어 공학에 의거한 각종 메트릭스 정보를 가지고 추정하는 방법 등 다양하다. 이렇듯 다양한 추정방법에 대한 경험축적 및 새로운 추정 모델의 개발, 모델 파라미터 추출에 대한 검증방법 연구와 이를 실제 프로젝트에 적용하여 적합성을 검증하는 문제가 연구되어야 한다.

참 고 문 헌

[1] C. Stringfellow, A. Andrews, C. Wohlin, and H. Petersson, "Estimating the Number of Components with Defects Post-release that Showed No Defects in Testing," Software Testing, Verification and Reliability, Jan. 2002, pp. 93 - 122.

[2] Shin-ichi Sato, Akito Monden, and Ken-ichi Matsumoto, "Evaluating the Applicability of Reliability Prediction Models between Different Software," IWPSE 2002, pp. 97 - 101.

[3] M.H. Halstead, "Elements of Software Science," Elsevier Computer Science Library, 1997.

[4] T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, 1976, pp. 308 - 320.

[5] C. Stringfellow and A. Andrews, "Deriving a Fault Architecture to Guide Testing," *Software Quality Journal*, 2002, pp. 299 - 330.

[6] N. Ohelsson and H. Alberg, "Predicting Fault-prone Software Modules in Telephone Switches," *IEEE Transaction on Software Engineering*, Vol. 22, No. 12, 1996, pp. 886 - 894.

[7] Taghi M. Khoshgoftaar and Edward B. Allen, "A Comparative Study of Ordering and Classification of Fault-prone Software Modules," *Empirical Software Engineering*, Vol. 4. 1999, pp. 159 - 186.

[8] J.K. Lee et al., "Case Study of the Large Switching Software Metrics and Their Fault Analysis," *Journal of KICS*, Vol. 27, No. 10, 2002. 10., pp. 887 - 901.

[9] Yamada Shigeru, 高橋 宗雄, "ソフトウェア ヌネヅメント モデル 入門 : ソフトウェア 品質の可視化と評價法," (日本) 共立出版株式會社, 1993, pp. 113 - 155.