제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

결함 원인 분석을 통한 코드검증 절차 도입 방안

노지호, 이인환
한양대학교 전자통신전파공학과
rjksuper@naver.com

Introducing Code Inspection Procedure Based on Defect Causal Analysis

Jiho Rho, Inhwan Lee
Electrical and Computer Engineering. Hanyang University

요 약

코드검증은 소프트웨어공학에서 제안한 개발 프로세스 상의 여러 절차 중 비용 대 효과 측면에서 가장 효과적이라고 알려져 있다. 그러나 코드검증 절차 도입 초기에는 적용에 따른 기대 수준이 낮으며, 추가 작업에 대한 개발자와 관리자의 부담이 높기 때문에 투입시간, 자원할당 등의 계획은 설득력 있는 근거를 바탕으로 체계적으로 수립되어야 한다. 본 논문에서는 기존 제품 개발 시 발생되었던 결함의 원인 분석과 결함 제거에 투입된 공수 산출을 바탕으로 코드검증 절차를 개발 프로세스 상에 도입, 계획할 수 있는 방안을 제시한다

1. 서 론

소프트웨어공학은 소프트웨어 개발 생산성과 품질을 향상시키기 위한 다양한 이론과 이에 대한 참조 모델들을 제공하고 있으며, 그 중 소프트웨어 검증절차(software inspection)는 설계 및 구현단계에서 결함을 발견 수정함으로써 결함의 전이 및 확산을 방지할 수 있어, 비용 대 효과 측면에서 가장 효율적인 기법으로 알려져 있다.

그러나 개발현장에서 소프트웨어 검증 절차를 실제 개발 단계에 적용하는데 있어 추가 비용 발생 및 개발 일정의 확대에 따른 관리자와 개발자의 심리적인 거부감이 가장 큰 장애 요소로 작용한다[1]. 이는 새로운 절차 도입 효과에 대한 확신을 가질 수 있는 근거가 부족하고, 외부의 성공 사례들도 NAH(Not Applied Here) 증후군에 의해 충분한 설득력을 가지지 못하기 때문이며, 이를 극복하기 위해서는 객관적인 데이터 도출 및 계획 수립이 매우 중요하다 [2].

본 논문은 코드검증(code inspection) 절차가 도입되지 않았던 과제를 대상으로 시스템 시험 및 인수 시험 단계에서 검출, 정정 조치된 결함을 취합하여 발생 원인별로 분류하고, 분류 정보를 바탕으로 코드검증 투입 최적 공수(effort)를 결정할 수 있는 정량적 모델을 중심으로 한 코드검증 절차 도입 방안을 제시한다.

2. DCA(Defect Causal Analysis)를 통한 SPI 절차

개발 단계에서 코드 검증 작업이 실시되지 않았고, 결함 방지 계획이 수립되지 않은 상황에서 결함 원인 분석 결과를 코드 검증 단계 도입을 통한 소프트웨어 개발 프로세스 개선에 활용하는 절차는 그림1)의 점선 아래에 도시 된 바와 같이 첫째, 개발 프로세스의 검사 및 유지보수 시 결함을 수집, 원인 분석하는 단계, 둘째 결함 중 코드검증 대상을 선정하는 필터링 단계, 셋째 코드검정 계획을 수립하는 단계로 나뉘어 진다.

본 논문에서는 코드검증 작업이 실시되지 않았던 동일한 분야(domain)의 3개 과제를 선정하여 구현

이후의 과정(시스템 시험 및 인수 시험 단계)에서 보고된 결함 중 구현 단계에서 발생된 것으로 파악되는 결함을 대상으로 결함 원인 분석, 검증 대상 정의, 코드 검증 공수 산정 방식을 적용해 보았다.

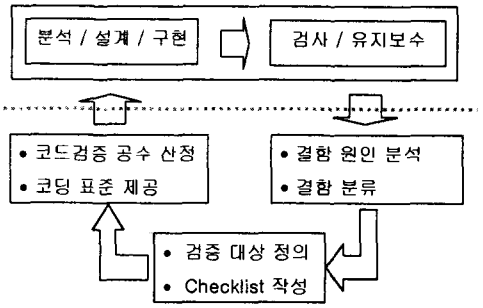

그림 1 DCA를 통한 코드검증 도입 절차

3. 결함 원인 분석 및 결함 분류

결함의 직접적인 원인은 시험에서 추출된 결함을 추적하여 정정하는 디버깅 작업 결과를 이용하여 파악할 수 있으며 결함 원인 파악을 통한 개선 작업이 이루어지기 위해서는 유사한 성격별로 분류되어 정리될 필요가 있다.

본 논문의 경우처럼 결함의 추출이 시스템 검사 및 인수 검사에서만 이루어 지고, 구현 결함 개선작업만을 목표로 할 경우에는 비교적 단순한 분류 기준을 적용하여 크게 모듈 구현 결함, 모듈간 인터페이스 결함으로 나누어 정리할 수 있다 . 아래 표1은 적용 예에 있어서 모듈 구현 결함(1~5), 모듈간 인터페이스 결함(6~9) 으로 나누어 정리된 것이다 [3][4].

표 1 결함 원인 및 정정 공수 분석 결과

ID	Defect Type	e0	Ae
1	Data design/usage	1.28	5.8
2	Range check	1.59	6.2
3	Language Pitfall	0.26	4.8
4	Functionality	1.79	5.9
5	System library usage	0.51	4.5
6	Resource allocation/usage	0.46	9.3
7	Module Interface	2.67	6.7
8	Hardware Interface	0.56	8.4
9	Change coordination	2.62	6.9
합계(평균)		11.74	6.5

표 1에서 e0는 kncsl(1000 non-comment source line)당 결함수, 즉 결함밀도이며, Ae는 결함 당 평균 수정 공수(man-day)로써 결함 발생 보고 후 결함추적, 결함 수정 및 Regression 검사 시간까지 포함된 수치이다.

4. 코드검증 대상 정의

코드검증 대상을 정의하는데 있어 고려해야 하는 점은 결함 형태에 따라 코드검증 작업의 효율이 다르다는 점이다 [5]. 특히 타이밍 관련 결함은 코드점검을 통해 추출하기 힘들기 때문에 점검 항목에서 제외하는 것이 효과적이다. 표1의 ID6, ID7 형태의 결함이 이에 해당한다.

또 하나의 고려 사항은 결함 밀도(E0)로써 결함밀도가 낮을 경우 코드검증 효율이 떨어진다는 점이며 [6], 코드검증 절차 도입의 전제조건(식-6)에서도 이유를 설명한다. 표1의 ID3, ID5 형태의 결함이 이에 해당한다.

5. 코드검증 계획 수립

본 절에서는 코드검증 계획 수립에서 우선적으로 고려해야 할 사항으로써 코드검증에 투입 최적 공수와 이에 따른 개선정도가 어떠한지를 정량적으로 제시한다.

5-1 코드검증 공수(Effort) 산정

본 논문에서는 정량적인 코드검증 공수 추정 모델을 유도하기 위해 D.A Christenson이 제안한 코드검증 모델 (식1)을 활용하여 이후 수식을 전개한다 [6].

$$C_t(h) = C_h h + e_0 C_h w f(h) + e_0 A[1 - f(h)] \qquad (1)$$

여기서 Ct는 kncsl당 결함 수정을 위한 총 비용이고, Ch는 시간당 인건비, h는 kncsl 당 코드검증에 투입되는 공수, w는 코드검증에 의해 검출된 결함 당 수정 평균 비용, e0는 (3절에서 정의된) 초기 결함밀도, A는 코드검증에 의해 검출되지 않은 결함 당 수정 평균 비용, f(h)는 코드검증에 의한 결함 검출비율에 해당한다. 이중에서 f(h)는 코드검증 단계에서 검출된 결함에 대한 수정 작업만을 진행한다고 가정할 때 아래와 같이 정의될 수 있다.

$$f(h) = 1 - e^{-h/\eta} \qquad (2)$$

여기서 h는 kncsl 당 코드검증에 투입된 공수, η는 코드검증의 효율과 관련된 공수 상수 (effort constant)이다. 식(1)의 양변을 Ch로 나누면, 비용 모델을 아래와 같은 공수 모델로 변경할 수 있다.

$$E_t(h) = h + e_0 w f(h) + e_0 A_e [1 - f(h)] \qquad (3)$$

여기서 Et는 kncsl 당 결함 수정을 위한 총공수이고, Ae는 (3절에서 정의된) 코드검증에 의해 검출되지 않은 결함 당 수정 평균 공수이다. Et를 최소화 할 수 있는 코드검증 투입공수를 구하기 위해 식(3)을 h에 대해 미분 하여 정리하면,

$$h_0 = -\eta \log(\frac{\eta}{e_0(A_e - w)}) = \eta \log(\frac{e_0(A_e - w)}{\eta}) \quad (4)$$

에서 최소값을 가지게 된다. 여기서 최적 투입 공수 h0가 의미를 가지려면 양실수(positive real value)가 되어야 한다는 점에서 다음의 조건들이 만족되어야 한다.

$$e_0(A_e - w) > 0, \quad \frac{\eta}{e_0(A_e - w)} < 1 \quad (5)$$

$$\text{i.e, } A_e > \frac{\eta}{e_0} + w \quad (6)$$

조건식 (6)은 공수 측면에서 코드검증 절차가 도입되어 의미를 가질 수 있는 전제조건으로 정의할 수 있으며, 조건이 만족되기 위해서는 Ae에 비해 우변의 식이 충분히 작은 값이 되어야 한다. 따라서 코드검증 작업 효율이 높아야 하며 (), 결함밀도가 높아야 하며(e0), 코드검증에서 검출된 결함 수정시간이 충분히 작아야 한다.

5-2 코드검증에 의한 개발 공수 개선율 산정

코드검증을 도입한 이후, 도입 전에 비하여 공수 개선 비율을 아래와 같이 산정할 수 있다.

$$I = \frac{E_t(0) - E_t(h_0)}{E_t(0)} \quad (7)$$

5-3 적용 예

식(4)를 이용해 각 결함유형에 따른 코드검증에 투입 될 최적 공수를 결정하기 위해서, Ae와 e0는 표 1)을 이용할 수 있다. 문제는 w, (의 값인데, 이들은 코드검증 작업을 통해 산출될 수 있는 값이나, 본 논문에서처럼 기존에 코드검증이 행해진 적이 없는 과제에서는 값을 구할 수가 없다. 따라서 이들 값들은 코드검증 도입 전에 실시되는 교육을 통해서 추정하는 것이 합당하나, 본 논문에서는 문헌에 보고된 값들을 활용하여 정의하였다 [7].

먼저 초기 조건을 1회 인당 코드검증시간 6시간, 1회 코드검증 인원 4인, 1일 근무시간 8시간으로 전제한 상황에서 결함 검출 비율을 50%로 가정하고 식(2)를 이용하면 공수 상수 (=4.33을 얻을 수 있다. 그리고 코드검증에 의해 검출된 결함 수정 공수 w는 은 Ae에 비해 매우 작다고 보아 Ae/w = 5 로 가정하였다.

코드검증 대상은 4절에서 기술한 대로 결함 형태에 따라 결함 ID 6, 7을 제외하고, 식(6)에 의해 정의된 코드검증 도입 조건을 만족시키지 못하는 결함 ID 3, 5를 제외한 결함 ID 1,2,4,7,9로 정하였다. 표 1)에서 주어진 값과 가정한 w, (를 식(3)에 대입하여 kncsl당 결함 수정 총 공수 Et를 구해 도시하면 그림 2)와 같다.

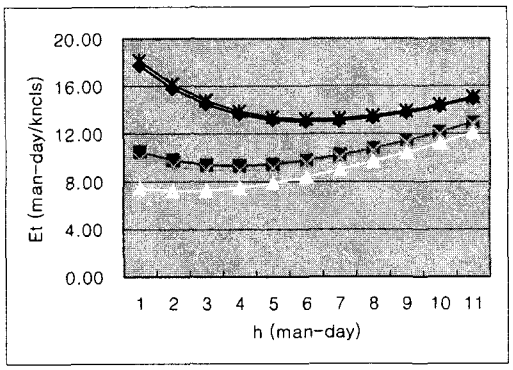

그림 2 코드검증 공수(h)에 따른 결함 수정 총공수 (Et)

그림 2)에서 빨간 마름모의 위치가 결함 수정 총 공수를 최저로 하는 최적 코드검증 공수 투입 시간에 해당하며, 식(4)를 이용해 각각의 값을 구할 수 있다

결론적으로 본 적용 예와 같은 유형의 결함이 발생된 경우 코드검증 대상으로 선정된 5가지 형태의 결함에 대해 kncsl당 17.25 man-day 정도의 코드검증 공수 투입을 계획할 수 있으며, 이때 구현 단계에서 발생하게 되는 결함의 전체 수정 공수의 37% 절약이 가능하다는 정량적인 수치에 근거한 비용 대 효과의 예측이 가능해 졌다.

6. 결론

본 논문에서는 소프트웨어 개발 시 코드검증 계획을 수립하는데 있어 과거 과제의 결함 원인 분석 결과를 이용할 수 있는 방안이 제시되었다.

특히 본 논문에서 제안된 방식은 기존 개발 프로세스에 코드검증 절차가 포함되지 않은 경우 첫째, 코드검증 대상 결함 형태를 식(6)에 의해 결정할 수 있도록 하였으며, 둘째, 계획에 필요한 최적 투입 공수를 식(4)에 의해 산출해 내고, 세째, 이를 바탕으로 코드검증 절차 도입에 따른 효과를 식(7)에 의해 산정할 수 있는 정량적인 방법을 제시함으로써 코드검

증 결차 도입 계획의 객관성과 설득력을 높이는데 유용하게 활용될 수 있을 것으로 보인다.

7. 참고 문헌

[1] Don O'neill, "Issues in Software Inspection", IEEE Software, pages 18-19, January 1997.

[2] Pankaj Jalote., "Overcoming the NAH syndrome for inspection development", IEEE Software Engineering, pages 371-378, 1998.

[3] Mark Sullivan, "Software Defects and their Impact on System Availability", IEEE Transactions on Software Engineering, pages 2-9, 1991.

[4] Marek Leszak, "A Case Study in Root Cause Defect Analysis", Proceedings of ICSE 2000, pages 428-437, 2000.

[5] Kathryn Bassin, "Evaluating Software Development Objectively". IEEE Software 15(6): pages 66-74, 1998.

[6] Dennis A. Christenson, "A Code Inspection Model for Software Quality Management and Prediction", Proceedings of GLOBECOM '88: IEEE Global Telecommunications Conference and Exhibition, pages 468-472, 1988.

[7] Forrest Shull, "What We Have Learned About Fighting Defects", Proceeding of IEEE Symposium on Software Metrics, pages 249-258, 2002.