ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

소프트웨어 시험노력 추정 모델의 설계

Design of Test-Effort Estimation Model

김 한 경1*
Hankyoung KIM

요 약

지금까지는 FP, UCP, COCOMO 모델에 의하여 시험노력을 추정하거나, 또는 개발한 수많은 프로젝트 데이터 측정을 통하여 각 단계별 노력 투입 비율에 의거 시험단계에 투입된 시험노력을 추정하였다. 본 연구에서는 소프트웨어 시험노력 추정을 소프트웨어 개발노력 추정과 독립적으로 이루어질 수 있도록 시험노력 추정 모델을 만들고 또 시험노력 추정절차를 제시한다. 모델은 시험노력이 테스트 케이스의 수와 복잡도에 비례하는 특성을 반영하고, 통합시험, 시스템시험, 인수시험 등 시험 태스크를 수행하는 시험 조직의 역량에 영향을 받는 점을 고려하였다. 제시한 시험노력 추정 모델과 절차에 의해 기존의 프로젝트 데이터에 시험에 관련된 추정 데이터를 이용하여 시험노력을 추정한 결과와, 개발계획 수립을 위하여 추정한 개발노력 상에서 배분된 시험노력과 비교하였을 때 4.7% 정도의 오차를 보였다. 시험 조직이 갖는 기술적인 경험, 구축된 시험환경의 정도, 프로젝트의 복잡성과 개발조직의 환경 등을 측정하여 주어진 모델의 조정 계수 값에 반영한다면, 보다 정교한 독자적인 시험노력 추정이 가능하다.

☞ 주제어 : 소프트웨어시험, 시험점수, 시험노력, 시험비용

ABSTRACT

Test effort estimated so far is as a by-product of the development effort estimation activity which is based on the FP, UCP, COCOMO model, or calculated data from the project knowledge base which is containing test effort information for the test phase on software development life cycle. In this paper, test effort estimation model and calculating procedures are suggested, which is independent from software development effort estimation model. Generally test efforts is depends on the number and the complexity of test cases, and also maturity of test organization that performs test activities, such as integration test, system test, acceptance test and so on. The estimated results with the suggested test effort estimation model has deviation of 4.7% compare to the corresponding test efforts generated by the development effort estimationprocedures. The suggesting model will be accurate more and more with refinements of coefficients which reflect the technical and environmental maturity level of test organization, and also including the software complexity level of projects.

☞ keyword : Software Test, Test Point, Test Effort, Test Cost

1. 서 론

소프트웨어 시험노력 추정은 시험 작업의 수행에서 비논리적인 요인이 있어 어렵다. 소프트웨어 엔지니어는 시험에 투입하는 노력을 소프트웨어 개발 일정계획에 맞추어 진행하는 경향이 있기 때문이다[1]. 동일한 시험작업이라도 현실적으로는 계획에 따라 투입되는 노력이 틀려질 수 있음을 의미한다. 이러한 상황에서 적정한 시험노력을 추정하고 시험 계획에 반영하는 것은 의미 있는 작업이다.

소프트웨어 시험에 관련된 연구는 시험방법론, 품질보증, 시험노력추정 등과 같은 분야에서 발전이 이루어졌다. 시험방법론은 많은 양의 테스트 케이스를 가능한 축소시켜 시험노력을 줄이면서 소프트웨어의 신뢰도를 보장할 수 있는 방법을 강구하고 있고, 품질보증 분야에서는 어떤 시험방법을 적용하여도 소프트웨어의 에러는 완벽하게 제거가 될 수 없다는 가정하에서 소프트웨어의 신뢰도를 예측하고 에러 제거능력의 범위와 함께 프로그램의 에러 포함 정도를 추적한다. 이러한 분야의 연구는 일단 구현이 된 소프트웨어에 대한 검증 방식의 시도인데 비하여, 시험노력 추정에 대한 연구는 소프트웨어 구현 이전 단계에서 시험활동의 종류와 각 활동 별 투입 노력을 예측하는 것이다. 따라서 시험노력 추정은 개발이

1 Dept. of Computer Engineering, Changwon National University, Changwon-si, Gyeongsangnam-do 641-773 KOREA
* Corresponding author (hkim@changwon.ac.kr)
[Received 16 August 2012, Reviewed 10 September 2012(R2 22 November), Accepted 04 January 2013]

되는 과정을 설정하고, 그에 따른 시험노력을 유추하여야 한다.

소프트웨어 개발 프로젝트에서 시험에 투입되는 비용은 개발, 유지보수 등의 관점에 따라 다를 수도 있지만 전체 개발비용의 30% ~ 90% 비용이 소요된다고 한다[2]. 범위를 시험 조직에 의한 정규 시험 활동으로 제한할 경우에도 소요되는 노력이 개발비용에 대략 10% ~ 25% 정도 된다고 한다[2]. 이러한 인식은 개발비용의 최적화에서 시험의 중요성이 더 이상 간과될 수 없는 이유이며, 이 분야에 대한 연구가 지속적으로 이루어지는 이유이기도 하다.

소프트웨어 개발 노력을 추정하는 모델은 많이 제시되었으나, 시험을 예측하는 모델은 독자적으로 존재하지 못하고 개발 모델과 더불어 제시되었다. 개발비용 산정을 위한 개발 모델은 Kilo Line of Code(KLOC). Cost Constructive Model(COCOMO), Function Point(FP), Use Case Point(UCP) 등이 있는데, 이를 통하여 전체 개발 비용을 산출하고, 그 개발 프레임에서 시험이 차지하는 비중을 계산하여 시험노력을 추정하는 방법이다. 따라서 시험노력을 추정하기 위해서는 개발 노력을 먼저 추정하여야 하고, 그 개발 노력에서 시험에 투입될 노력을 추정하는 것이 대부분의 방법이다.

2. 관련 연구

2.1 시험노력 최적화 연구

McCabe와Halstead에 의해 소프트웨어 비용에 영향을 주는 요인에 대한 연구가 있었다[3-5]. McCabe는 프로그램의 구조에 따라 Cyclomatic Number를 제시하였는데, 소프트웨어의 복잡도의 기준으로 삼았다[3]. 복잡도가 높은 프로그램은 유지보수 비용이 증가하고, 시험 비용도 증가한다.

Halstead는 software science 개념을 제시하여 프로그램의 길이(N), 부피(V), 알고리즘의 최소부피(L), 프로그램 개발노력(E)을 프로그램에서 사용된 operator, operand를 이용하여 계산할 수 있도록 수식을 제시하였다[4,5].

또 다른 형태의 소프트웨어 시험노력의 최적화 연구는 참고자료 [6]에서의 테스트 케이스에 관한 연구이다. 즉, 시험에 투입될 노력을 최소화하기 위하여는 테스트 케이스 선정을 최적화하여야 한다. 소프트웨어 시험은 동작시험(operational test)과 디버깅시험(debugging test)으로구별한다. 동작시험을 위해서는 operational profile에 의해 테스트 케이스를 선정하여야 하는데, operational profile이란 프로그램 동작 중에 발생 가능한 입력 데이터 도메인의 요소 d가 사용될 확률분포 p(USE(d))를 의미하며, 입력 도메인 D에 속한 테스트 케이스 t가 시험에 사용될 확률이 임의의 값 μ보다 큰 경우의 모든 테스트 케이스의 집합 To 에 의한 시험을 동작시험이라고 한다. 즉, To = {t | t$\in$ D $\wedge$ p(USE(t)) $\geq$ μ} 으로 표현되는데, μ는 상수이며, 값은 To에 속한 모든 요소에 대한 p(USE(t)) 값이 미리 주어진 값 γ과 같거나 보다 큰 값을 갖도록 선택한다. To에 의한 시험이 양호하다고 판정되면, 그 프로그램은 γ ×100% 신뢰할 수 있다라고 한다.

디버깅시험은 수행 빈도가 낮은 테스트 케이스(very low p(USE(t)))보다 에러가 발생할 가능성이 높은 테스트 케이스(very high p(¬OK(t)))를 선정하여 시험하는 방법이다. 디버깅시험을 통하여 장애를 발견하고 또 제거하여도, 테스트 케이스 선정을 오류가 발생할 가능성이 높은 것 위주로 선정하였기 때문에 프로그램의 신뢰도에 미치는 영향은 상대적으로 적을 수 있다.

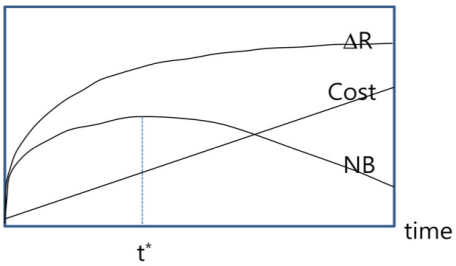
요약하면 소프트웨어의 시험노력이 프로그램의 테스트 케이스의 수에 비례한다는 것과, 시험노력을 단축하기 위한 방법이 동작시험과 디버깅시험으로 나누어, 동작시험에서는 통계적으로 그리고 디버깅시험에서는 논리적으로 존재하는 du-Path를 찾고 축소하는 것이 중요한 것으로 간주하였다.

2.2 적정한 시험종료 시점의 추정

A Sherer는 시험에 투입할 적정 비용, 즉 시험을 중단할 시점을 결정하기 위한 방안에 대하여 연구하였다[7]. 시험 비용이란 시험에 투입되는 장비의 사용료와 시험에 투입되는 인건비를 의미한다. Sherer에 의하면 프로그램의 디버깅시험 비용에 대한 손익추정이 가능한데, 만일 시험 비용과 그에 따라 얻을 이익을 비교하여 손익의 교차점 이전에 시험을 중단할 시점을 결정할 수 있다면, 시험에 투입할 노력의 적정량 산출이 가능하다. 이를 위해서는 (1) 프로그램에 잠재된 오류(faults)를 도출하고, (2) 각각의 오류에 의해 야기되는 경제적인 손실을 평가하고, (3) 예정된 프로그램 운용 기간에 고장(failure)이 발생할 확률에 의해 리스크 R을 계산하는 과정으로 진행된다.

에러를 방치하였을 때 에러가 야기하는 비용을 리스크 R이라고 하고, 에러를 제거할 경우 감소되는 리스크의 값을 ΔR 이라고 하면, 시험활동을 통해 획득하게 되는 실질적인 이득(NB, Net Benefit)은 NB = ΔR - C의 식으로 표현된다. 여기서 C 는 소프트웨어 시험 비용을 의미한다.

시험 비용(C)은 시험 시간에 비례하여 증가하는 특성을 가진다. 반면 리스크 비용의 경우에는, 일반적으로 프로그램의 시험 초기 단계에는 내재된 오류의 수가 많아서 오류 검출이 용이하므로 시험활동 초기에는 리스크 비용의 감소액(⊿R)은 지수급수로 증가하지만, 점차 잠재된 오류의 수가 줄어들게 되면 찾기가 어려워지므로, ⊿R의 크기는 변동폭이 줄어들게 되는데 증가 폭이 멈추는 시점, 즉 시험노력에 대한 이득의 변동이 거의 없는 시점(t^*)이 시험을 종료할 타이밍이 된다. risk, cost, benefit의 관계가 (그림 1)에 보여준다. 시점 t^* 이후에도 계속 시험을 하게 되면, 비용(C)의 증가분이 이득의 증가분을 추월하게 되어 실질이득(NB)은 점점 감소하다가 손실로 바뀌게 된다.



(그림 1) 디버깅시험에서 손익관계
(Figure 1) Net-benefit curve in debugging test

시험비용 지출 면에서는 NB 값이 최대가 되는 시점을 t^*로 선정하여야 하지만, 높은 신뢰성이 요구되는 프로그램의 경우에는 NB 값이 0이 되는 시점을 t^*로 잡아야 한다. 적정한 시험노력에 대한 연구는 소프트웨어 시험노력을 시간적인 함수로 보았다는 것에 주목한다.

2.3 소프트웨어 시험계획과 시험 카테고리

Srivastava[8]는 시험노력을 추정하는 기준으로 Component Rank(CR), Kilo-Line Of Code(KLOC), Cyclomatic Complexity (CC), Customer Priority(CP), average KLOC($KLOC_{avg}$)의 5가지를 제시하였으며, 각 기준의 가중치를 '매우 낮음'에서 '매우 높음'까지의 6단계(VL, L, M, H, VH)로 분류하고 이들 가중치 값을 0.0에서1.0사이의 값으로 평가할 수 있도록 하였다. 특히 그는 평가자에 의한 각 기준의 평가를 triangular membership value weight 에 의한 퍼지 방식으로 평가를 하고, 평가자들의 평가 결과를 행렬식으로 나타내고 각 행렬식의 항목별로 preference value를 반영한 다음 low, medium, high로 구별되는 영역에서 gravity center를 택해 *confidence value C*를 구한다. C가 구해지면 수정된 (KLOC') = KLOC x C를 구한다. 이KLOC'는 개발에 필요한 실질적인 프로그램의 크기이며, 이KLOC 값을 COCOMO 모델에 적용하여 개발노력을 산정하고, 총 개발노력의50%를 시험노력으로 제시하였다.

이상운[9], 박주석[10]의 시험노력 추정모델은 시험노력의 투입되는 양상으로 모델을 설정하고, 다양한 프로젝트 데이터를 이용하여 회귀분석 방법으로 계수를 조정하여 예측모델을 완성 시켜나간 방법인데 비하여, Srivastava의 방법은 프로젝트 전문가의 전문지식과 경험을 이용하여 5가지 평가 영역에 대해 각기 평가한 것의퍼지 평균값을 구하고, COCOMO 모델을 이용하여 개발노력과 개발기간을 추정하였다.

2.4 개발노력 추정에서의 시험노력

시험노력은 대부분 개발노력을 예측한 다음 그 중의 일부를 시험에 배분해주는 방식을 택하기 때문에 소프트웨어 개발노력을 추정하는 모델도 중요하다. 소프트웨어 개발 노력 추정 모델은 LOC[11,12], COCOMO [13], FP [14,15], UCP [1], PDB(Project Data Base)[1]에 의한 모델 등이 있다.

LOC에 의한 소프트웨어 개발 노력의 추정은 개발해야 할 소스 코드 라인 수를 추정하여 생산성 매트릭을 이용하여 개발 노력을 산정한다. 이에 비해 COCOMO 모델은 같은 LOC를 이용한 개발노력 추정 방식이지만 공식에 의해 노력과 개발기간을 추정한다.

FP에 의한 시험노력 추정 방식도 경험 데이터를 이용한다. LOC의경우 사용언어와 프로젝트의 복잡성, 난이도의 차이점을 반영할 수 있는 방법을 보정계수를 이용한 반면, FP 방식에서는 입출력 데이터, 입출력 파일, 인터페이스, 질의어의 수 등measuring parameters의 데이터를 분석하여 프로그램의 구현 복잡도를 반영하고, 프로젝트의 기술적인 난이도 14항목에 대하여 5단계 가중치를 반영하여 FP 점수를 산정한다. 산정된 점수에 점수 당 생산성 매트릭을 통하여 개발노력을 산정한다.

참고자료[8]에 제시된 Rational의 UCP 방식은 프로젝트에 구현할 use case를 그 use case에 속한 transaction 수에 의해 단순, 보통, 복잡(S/M/C) 상태로 분류하고, 각 상태의 use case 수 x (상태 별 가중치)에 의해 UUCP(Unadjusted Use Case Point)를 계산한다. UUCP에 기술적인 난이도와 환경적인 난이도를 반영하여 UCP를 산출한다. 즉,

UCP = UUCP * TCF * EF

TCF = 0.6 + (0.01 * TFactor)

$$\text{TFactor} = \sum_{i=1}^{13} (\text{rate}_i \text{ x weight}_i) \text{ for Technical Complexity Factors}$$

EF = 1.4 + (-0.03 * EFactor)

$$\text{EFactor} = \sum_{i=1}^{8} (\text{rate}_i \text{ x weight}_i) \text{ for Environmental Factors}$$

2.5 PDB에 의한 시험노력 추정

Infosys에서 다양한 프로젝트를 진행하면서 측정된 데이터를 프로젝트 데이터베이스(PDB)로 구축해놓고, 이를 이용하여 새로운 프로젝트의 추정에 활용한다[1]. 표1에 제시된 Infosys사의 ACIC 프로젝트의 개발단계 별 노력 배분은 유사 프로젝트의 경우를 비교하여 배정되었다. 개발 단계별로 배정된 review, inspection, debugging, test 등의 노력을 실제의 개발노력과 비교하였을 때 5% 정도의 오차율을 갖는 정밀성을 유지한다. 소프트웨어 프로젝트 개발노력 추적에 경험 데이터를 활용하는 것은 신뢰도 면에서 양호한 결과를 나타내지만, 이는 시험노력 추정이 개발계획 범위 안에서 이루어져야 한다.

PDB에 의하면 시험노력이30~90%에 이른다는 Boris Beizer의 언급과 비교할 때 시험노력의 범위를 통합시험, 시스템시험, 인수시험에 한정할 경우에는 15% 수준의 시험노력이 배분되는 것을 알 수 있다.

(표 1) ACIC 프로젝트 개발단계별 노력 배분율
(Table 1) Effort distribution rate for the ACIC project

Requirement	10
Design	12
Build	29
Integration testing	7
Regression testing	2
Acceptance testing	6
Project management	15
Configuration management	3
Training	10
Others	6
(total)	100%

3. 제안 모델

3.1 Test Point Estimation Process

시험노력은 테스트 케이스의 수와 테스트 케이스의 난이도에 비례하는 것을 알 수 있으며, 테스트 케이스의 수행은 시간에 비례하여 증가하고, 비용은 지수 함수로 증가한다는 것을 알 수 있었다. 동작시험과 디버깅시험에서 테스트 케이스의 도출 결과가 다를지라도 궁극적으로 시험노력의 최적화는 테스트 케이스의 최적화가 필요하다. 개발노력 추정과는 독립적인 방식의 시험노력 추정 모델을 만들고, 각각의 모델에 의해 도출된 결과를 서로 비교할 수 있도록 하는 것도 의미 있는 작업이다. 시험은 시험팀에 의해 Black Box test 형태로 진행되는 것이 보편적이므로, 시험 팀에 의해서도 시험노력을 예측하는 것이 필요하다.

TP의 계산을 다음의 절차에 따라계산하여 시험노력을 추정한다.

(step 1) test case 추정

요구사항을 참조하여 테스트 케이스의 수를 추정한다. 테스트 케이스의 추정은 Delphi 방식을 이용하여 추정하도록 권고한다. 전체적으로 총괄 추정이 어려운 경우에는 Customer Service 또는 Use Case 단위로 추정을 진행한다. 테스트 케이스는 black box test의 경우에 대하여 추출한다. 테스트 케이스 별로 난이도를 예측하여 S/M/C로 분류한다. 즉, S/M/C 등급별 테스트 케이스의 수를 추정한다.

(step 2) UTP 계산

Unadjusted Test Point(UTP)를 산정한다. 이때 가중치는 3가지 부류에 대한 상대적인 난이도를 반영한다.

$$UTP = \sum_{type=S,M,C} \left((\#\text{ of test cases})_{\text{type}} \times \text{weight}_{\text{type}} \right) \quad (1)$$

(step 3) TR, ER의 계산

UTP에 시험에 대한 복잡성과 환경적인 상황을 반영한 Technical Reference(TR)와Environmental Reference(ER)를 (표 2)와 (표 3)을 이용하여 구한다. 표에서 rate와weight는 프로젝트의 성격에 따라 조정된다.

$$TR = \sum_{n=1}^{13} (\text{rate}_i \times \text{weight}_i) \quad (2)$$

$$ER = \sum_{n=1}^{8} (\text{rate}_i \times \text{weight}_i) \quad (3)$$

(표 2) 복잡도 조정 항목
(Table 2) Adjustment of Technical References items

복잡도 조정항목 (Technical References)	rate	weight
1. 분산 시험환경 구축이 필요하다. 2. Response time, throughput 등 성능점검이 중요하다. 3. 온라인 사용자의 효율성 여부가 시험의 주요 목적이다. 4. 입출력, 파일, 질의가 다양하고 처리 알고리즘이 복잡하다. 5. 코드 재사용을 위한 요구사항도 함께 시험한다. 6. 마스터 파일들이 온라인으로 갱신된다. 7. 온라인 데이터 입력 및 다중 조작에 의해 트랜잭션이 구축된다. 8. 운영 platform환경이 다양하고 그 이식성도 시험하여야한다. 9. 온라인 중의 변경, 설치, 작업 편의성 시험이 필요하다. 10. 기능이 설치되는 노드가 많고, Concurrent 시험이 필요하다. 11. 백업과 복구기능과 함께 사용자의 보안 기능이 요구된다. 12. 제3자에 의한 접근 및 처리 기능이 포함된다. 13. 특별히 사용자 교육 프로그램이 필요하다.		

(표 3) 환경 조정 항목
(Table 3) Adjustment of Environmental References items

환경조정항목(Environmental References)	rate	weight
1. 인터넷 및 시험환경이 잘 갖추어져 있다. 2. 도메인에 대한 시험 경험이 축적되어 있다. 3. 핵심 기술에 대한 경험이 축적되어 있다. 4. 경험이 있는 시스템 분석자가 개발에 참여한다. 5. 시험 참여자들의 동기부여가 잘 되어 있다. 6. 요구사항 변경 발생 빈도가 높다. 7. 전담 개발 참여자들의 비율이 높다. 8. 개발자의 프로그래밍 언어에 대한 숙련도가 높다.		

(step 4) TF, EF의 계산

계산된 TR과ER을 이용하여 TF와 EF를 산정한다. 시험을 수행할 시험조직의 역량을 경험 데이터를 통하여, α, β, γ, δ의 값으로 부여한다.

$$TF = \alpha + (\beta \times TR) \quad (4)$$

$$EF = \gamma + (\delta \times ER) \quad (5)$$

(step 5) TP의 계산

시험에 대한 기술적인 복잡성과 환경적인 요소 및 조직의 역량을 UTP에 반영하여 TP를 계산한다.

$$TP = UTP \times TF \times EF \quad (6)$$

(step 6) Test effort의 계산

생산성 매트릭을 이용하여 총 시험노력을 구한다.

$$Total\ Test\ Effort = \text{TP} \times (\text{TP 당 투입인력}) \quad (7)$$

TR, ER 계산을 위한 템플레이트에서 Rate는 프로젝트의 특성을 반영하여 0~5 사이의 값을 부여하고, Weight는 전체 항목에서 각 항목이 차지하는 상대적인 비중을 0~5 사이의 값으로 부여한다. 0은 무관, 5는 절대적인 경우를 나타낸다.

3.2 Case Study

추정 모델을 검증하기 위하여 참고자료 [1]에서 제시된 ACIC 프로젝트의 Project Knowledge Base(PKB)에 제시된 자료를 활용하였다. 자료에서는 UCP를 이용하여 반복적인 개발방법론을 적용한 사례를 제시하고 있으면서, Waterfall model에 의하여 개발하는 경우와 비교되어 있다.

(step 1) test case 추정

ACIC 프로젝트의 분석 결과 제시된 26개의 use case에 대하여 (표 4)과 같이S/M/C등급별 테스트 케이스의 수를 독자적으로 추정하였다. 이러한 추정은 실제 프로젝트가 진행되면서 더 정교한 값으로 반영이 될 수 있지만 초기에는 경험과 이전 프로젝트 자료에 의해 추정이 가능하다. 테스트 케이스 수를 추정하기 위하여 프로젝트 개발노력 추정에 사용한 use case의 수, use case 당 구현노력, use case당transaction의 수를 고려하였으며, 그 결과를 표 4의 5번째 칼럼에 제시되어 있다.

(표 4) 테스트 케이스 추정을 위한 use case 등급별 특성
(Table 4) Characteristics of use case categories to estimate test cases

UC의 복잡도	UC수	UC구현노력	UC당 트랜잭션수	Test case 수추정(S/M/C)
Simple	5	1 MD	3이하	2/1/0
medium	9	5 MD	4-7	20/10/5
complex	12	8 MD	7이상	40/30/10
합계				62/41/15

(step 2) UTP 계산

UTP의 계산은 S/M/C로 구별된 test case 각각에 대한 시험의 가중치를1:5:8로 설정하고, 수식 (1)에 따라 UTP = (62 x 1) + (41 x 5) + (15 x 8)에 의해 unadjusted test point 187이 구해진다.

(step 3) TR, ER의 계산

수식 (2)와 (3)에 의해 TR과ER을 계산하여야 하는데, (표 2)과 (표 3)에 제시되어 있는 항목에 대하여 rate와 weight를 각각 항목 순서대로 (2, 2, 1, 4, 2, 1, 2, 3, 4, 1, 5, 1, 1), (2, 1, 1, 1, 0.5, 1, 1, 2, 0.5, 1, 1, 0.5, 0.5), 그리고 (3, 2, 1, 1, 2, 4, 1, 3), (2, 1, 1, 1, 1, 2, -1, -1)와 같이 부여하고, 수식 (2)에 의해 TR = 2 x 2 + 2 x 1 + 1 x 1 + 4 x 1 + 2 x 0.5 + 1 x 1 + 2 x 1 + 3 x 2 + 4 x 0.5 + 1 x 1 + 5 x 1 + 1 x 0.5 + 1 x 0.5을 계산하면 30이 구해지고, 수식 (3)에 의해 ER =3 x 2 + 2 x 1 + 1 x 1 + 1 x 1 + 2 x 1 + 4 x 2 + 1 x (-1) + 3 x (-1)을 계산하면 14의 값을 얻을 수 있다.

(step 4) TF, EF의 계산

수식 (4)의 TF를 계산하기 위해 먼저 α, β 값을 0.3, 0.01으로 부여하였다. 이것은 조직의 특성을 반영하는 값으로 측정에 의해 정제되어야 하는 값이지만, 원천 데이터의 특성을 고려하여 부여하였다. 동일한 방법으로 EF의 계산을 위해 γ, δ의 값을 0.7, -0.03으로 부여하였다.

주어진 조정계수를 반영하고 (step 3)에서 계산된 TR, ER 값을 적용하면, $TF=\alpha+(\beta\times TR)$=0.3+(0.01×30)=0.6 그리고, $EF=\gamma+(\delta\times ER)$=0.7+(-0.03×14)=0.28의 값을 구할 수 있다.

(step 5) TP의 계산

수식 (6)에 의해 TP=UTPxTFxEF=187x0.6x0.28=31.416으로 test point가 31.416이 된다.

(step 6) Test effort의 계산

생산성 데이터 20 MH/TP, 8 hours/day 를 적용한다면, 수식 (7)에 의해 Total Test Effort=TPx(TP당 투입인력) = 31.416x20=628.32 MH=78.54 MD의 test effort가 구해진다.

3.3 검토

78.54 MD의 test effort가 추정되는데, Infosys에서 추정한 개발노력에서 (표 1)을 통하여 배분된 시험단계(integration test, regression test, acceptance test)에 소요되는 시험노력 추정치 75 MD 와는3.54MD가 많은 4.7 % 정도의 오차를 보인다. 대체적으로 추정한 값이 실제 투입된 노력 값 보다 적은 경향을 보이는 것을 고려한다면4.7%의 오차는 오히려 바람직한 데이터로 간주할 수 있다.

TR, ER을 계산하기 위하여 (표 2)과 (표 3)에 제시한 rate와weight가 많은 프로젝트를 통하여 좀더 정교하게 조정이 된다면 시험노력 추정도 정교하게 될 것이다. TP 계산절차 4단계에서 제시된 조정계수 α, β, γ, δ는 조직의 특성에 맞추어 경험 데이터로 결정되어야 하는데, Infosys의 UCP 기반의 개발노력 계산에 적용한 값을 고려하여 α, β, γ, δ 값을 0.3, 0.01, 0.7, -0.03으로 부여하였다. 이 값은 조직의 경험과 노하우 및 시험기술력 그리고 시험환경이 개선되면 그에 맞추어 조정이 되어야 한다.

TP당 투입노력도 프로젝트에 따라 정제되어야 하는데, 1단계에서 S/M/C로 정리된 테스트 케이스의수에서 C 그룹의 수가 가장 많으면 30%의 범위 내에서 20MH보다 큰 값으로 조정할 수도 있다.

rate 값으로 0.5 및 -1 값의 지정은 원래 데이터 값을 인용하는 과정에서 0~5 사이의 값으로 부여하게 되면 발생하게 될 비교의 불일치 때문에 바로 잡지 못하였다. 이는 데이터 부여에 대한 규칙과 달리 운용하였다는 면에서 시정되어야 한다.

4. 결 론

시험을 언제 종료할 것인가 의사결정을 해야 하는 문제는 프로젝트 책임자로서 중요한 사항이다. 시험 비용을 최소화하는 방법, benefit을 극대화하는 방법, 신뢰도를 기준치 이상이 되도록 하는 방법 등 여러 선택사항 중에서 하나를 결정하여야 할 것이다.

본 논문에서는 프로젝트의 특성을 파악하여 use case를

도출한 다음, use case별 테스트 케이스의 수를 예측한다. 테스트 케이스 별로 시험 난이도가 다른 점을 고려하여 예측된 테스트 케이스를 simple/medium/complex 3등급으로 분류한다. 분류된 테스트 케이스 등급별 가중치를 반영하여 UTP를 구한다. 시험점수를 정제하기 위하여 시험수행 상의 고려사항 즉, 기술적인 복잡도와 시험수행 환경의 수준을 반영하기 위한 TR, ER 점수를 구한다. 구해진 TR, ER 점수는 시험조직의 역량을 반영하는 계수조정으로 시험점수 TP를 구하는 절차와 함께 연산모델을 제시하였다.

본 모델을 이용하여 프로젝트 시험노력을 예측한 결과 실제 투입된 시험노력과 4.7%의 오차를 보였다. 시험 추정 데이터로서 양호한 것으로 판단된다.

앞으로는 어플리케이션의 종류에 따라 구별하여 적용할 수 있는 모델로 발전시키는 것이 필요하다. 자료처리, 온라인 게임, 통신 어플리케이션 등 소프트웨어 종류에 따라 시험의 난이도와 투입노력은 가중치에서 구별이 되어야 한다. 이를 위해 모델을 좀더 정밀하게 작동하도록 α, β, γ, δ 계수 조정 등의 노력이 필요하다.

참고문헌(Reference)

[1] Pankaj Jalote, "Software Project Management in practice," Addison-Wesley (ISBN 0-201-73721-3), 2002.

[2] Boris Beizer, "Software Testing Techniques," 2nd. Edition, Van Nostrand Reinhold, New York (ISBN 0-442-20672-0), 1990.

[3] T. McCabe, "A Software Complexity Measure," IEEE Transactions on Software Engineering, Vol.2, No.4, 1976, pp.308-320.

[4] M. H. Halstead, "Elements of Software Science," New York: Elsevier North-Holland, 1977.

[5] H. Halstead, "Toward a Theoretical Basis for Estimating Programming Effort," Writings of the Revolution selected reading on Software Engineering edited by Edward Yourdon, YOUTDON inc., 1982

[6] J. C. Huang, "Software Error Detection through Testing and Analysis," A John Wiley & Sons, Inc., Publication (ISBN 978-0-470-40444-7), 2009.

[7] S.A. Sherer, "A Cost-Effective Approach to Testing," IEEE Software, vol.8, no2, Mar. 1991, pp.34-40.

[8] P. R. Srivastava, "Optimal Software Release Using Time and Cost benefits via Fuzzy Multi-Criteria and Fault Tolerance," Journal of Information Processing Systems, Vol. 8, No. 1, March 2012, pp.21-54 .

[9] Sang-Un Lee, "Sigmoid Curve Model for Software Test-Effort Estimation," Journal D of Korea Information Processing Society, Vol. 11D, No. 4, Aug., 2004, pp.885-892.

[10] Ju-Seok Park, "An Estimating Method for Software Testing Manpower," Journal D of Korea Information Processing Society, Vol. 11D, No. 7, Dec., 2004, pp.1491-1498.

[11] I. Sommerville, "Software Engineering," 8th Ed. Addison-Wesley (ISBN 978-0-321-31379-9), 2007.

[12] E. Choi, "Software Engineering," 5th ed., Jeongiksa, 2011.

[13] B. W. Boehm, "Software Engineering Economics," Prentice Hall, 1981

[14] A. J. Albrecht et. al., "Software Function, Source Line of Code and Development Effort Prediction : A Software Science Validation," IEEE Transaction on Software Engineering, Vol.SE-9, No.6, pp.639-648, 1983

[15] J. E. Matson, et. al., "Software Development Cost Estimation Using Function Points,"IEEE Transaction on Software Engineering, Vol.20, No.4, pp.275-287

[16] Qu Yi Zhou Bo, et. al., "Early Estimate the Size of Test Suites from Use Cases," 15th Asia-Pacific Software Engineering, IEEE Computer Society, 2008

[17] Daniel Guerreiro e Silva, et. al., "A Simple Approach For Estimation of Execution of Function Test Case," IEEE-International Conference on Software Testing Verification and Validation, 2009

[18] Priya Chaudhary, C.S. Yadav, "An Approach for Calculating the Effort Needed on Testing Projects," International Journal of Advanced Research in Computer & Technology, Vol.1 Issue 1, March 2012

◐ 저 자 소 개 ◑

김 한 경

1973년 서울대학교 원자력공학과 졸업(학사)
1987년 충북대학교 대학원 전산통계학과 졸업(석사)
1996년 충북대학교 대학원 전자계산학과 졸업(박사)
1997년~현재 창원대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 프로젝트관리론
E-mail : hkim@changwon.ac.kr