회귀 테스팅의 테스트 케이스 우선 순위화 기법의 실험적 연구

소 선 섭[†] · 채 의 근[††]

요 약

테스트 케이스 우선 순위화는 회귀 테스팅이 시간 제약 하에서 주어진 모든 테스트 케이스를 수행할 수 없을 때 테스트 케이스의 실행 순서를 스캐줄링하는 것이다. 본 논문에서는 장기적인 회귀 테스팅 환경에서 과거의 테스트 실행 및 오류 검출 정보를 활용한 HED우선 순위화 방법을 제안하고, 이를 기존의 Random 및 LRU 방법과 비교하였다.

본 실험을 통해 몇 가지 중요한 통찰을 얻을 수 있었다. 첫째, 우선 순위화 방법들이 프로그램의 특성에 따라 성능 면에서 상호 보완적이라는 점이다. 오류를 찾는 테스트 케이스들을 많이 갖고 있는 프로그램의 경우에는 Random이 효과적이고, 상대적으로 오류를 찾는 테스트 케이스의 비율이 작은 경우에는 제안된 HED방법이 좋은 성능을 보였으며, 중간 정도인 경우에는 LRU 방법이 효과적이었다. 둘째, 전체적인 성능이 테스트 스위트의 크기에 영향을 많이 받는다는 점이다. 테스트 스위트의 크기를 달리하여 실험한 결과 오류의 수명 값과 그 성능 순위에 차이를 보였다. 마지막으로 전체 테스트 케이스의 20%만을 실행하여도 전체 테스트 케이스 모두를 실행하는 것과 성능 면에서 유사한 결과를 얻을 수 있다는 점 등이다.

Empirical Study on Test Case Prioritization Techniques of Regression Testing

Sun Sup So[†] · Yigeun Chae[††]

ABSTRACT

Test case prioritization methods schedule test cases for execution when we can not practically run all test cases for regression testing. We proposed a new prioritization method that is based on historical execution and error detection data. And we conducted an experiment to compare the proposed method with existing Random and LRU methods using the fault age under the long run environment as criterion.

The experiment shows several interesting results. First, our results show that they are complementary. Random method shows good performance for programs that have many error-detectable test cases and HED is more effective for the programs that can be detected by very small amount of test cases. But LRU is more effective for the programs that have relatively medium amount of error detectable test cases. Next, the performance of prioritization method is affected by the size of test suites. Two experiments that have different size of test suites show considerably different fault ages and performance order. And lastly, the 20% of test cases shows considerably good performance compared to the execution result of the full test suite.

키워드 : 회귀 테스팅(Regression Testing), 테스트 케이스(Test Case), 테스트 스위트(Test Suite), 우선 순위화(Test Case Prioritization), 오류 수명(Fault Age)

1. 서 론

소프트웨어 시스템은 개발이 완료되어 운영되는 과정에도 요구의 변경이라든가 성능 향상, 그리고 많은 개발 과정 중의 테스팅에 의해서도 찾지 못한 미묘한 오류 등, 다양한 이유로 일련의 변경 과정을 거친다. 프로그램의 변경은 변경된 부분이 다른 부분에 영향을 미칠 수 있다는 점에서 매우 신중하게 이루어진다. 변경된 부분에 의해 다른 부분이 영향을 받아 발생할 수 있는 오류가 회귀 오류(Regression error)이고 이 회귀 오류를 없애려는 노력이 회귀 테스팅(Regression testing)이다.

가장 간단한 회귀 테스팅 방법은 개발 과정에 작성된 전체 테스트 케이스를 모두 실행하는 것이다. 하지만 이 방법은 변경된 부분이 전체 프로그램에 비해 작을수록 비용면에서 매우 비효율적이다. 특히 회귀 테스팅은 이미 운영 중인 소프트웨어에 행해진다는 점을 고려할 때 빠른 처리는 필수적이다. 결과적으로 회귀 테스팅의 시간 비용을 효과적으로 줄이려는 많은 노력들이 이루어졌다. 먼저 모든 테스트케이스를 재실행하지 않고 일부 테스트 케이스만을 선정하여 오류를 찾고자 하는 회귀 테스트 선정 기법들이 제안되었다. 대표적인 테스트 선정 알고리즘으로는 Random, Minimization[1, 2], Data Flow[3], Safe[4, 5] 알고리즘을 들 수 있다.

효과적인 테스트 선정 방법이 결정되더라도 현실적으로

※ 이 논문은 한국과학재단의 해외 Post-doc. 연수지원에 의하여 연구되었음.
† 정 회 원 : 공주대학교 컴퓨터공학부 부교수
†† 정 회 원 : 공주대학교 컴퓨터공학부 조교수
논문접수 : 2004년 12월 4일, 심사완료 : 2005년 3월 4일

시간과 자원이 제한적이어서 선정된 테스트 케이스를 모두 실행하지 못하는 경우가 많다. 이러한 경우 테스팅 담당자는 원하는 특성을 잘 만족하는 순으로 테스트 케이스에 순서를 정하여 실행하고자 할 것이다. 이와 같은 시간 제약의 문제를 해결하고자 최근에는 테스트 케이스의 우선 순위화에 관한 연구가 활발히 진행되고 있다.

W.Wong 등[6]은 커버리지 당 드는 비용에 따라 테스트 케이스 선정 우선순위를 정하는 방법을 제안하였다. 이 방법은 테스팅 과정 중 커버리지 정보를 기반으로 가능하면 일찍 오류를 찾으려는 것으로 실험을 통해 제안 방법이 비용 효과적임을 보였다. 하지만 이들은 적용할 수 있는 회귀 테스팅 방법과 우선순위화의 전반적인 기법에 대한 고찰을 기술하지는 못했다. G.Rothermel 등[7]은 Wong의 연구를 발전시켜 우선순위화 전반에 대한 문제를 기술하고 여러 가지 기법을 제안하고 오류 검출 비율을 향상시키는 효과에 대한 실험을 실시하여 상호 방법간에 장단점이 있음을 보였다.

우선 순위화에 대한 또 다른 방향은 회귀 테스트 선정 기법과 우선순위 알고리즘을 적용할 때 과거 테스팅 세션의 기록을 활용하려는 것이다. 연속적으로 회귀 테스팅이 행해지면 테스트 로그에는 많은 유용한 정보들이 포함되어 있다. 예를 들어, 특정 테스트 케이스의 실행 횟수나 실행된 시기, 오류를 찾은 횟수 등을 들 수 있다. 회귀 테스팅을 연속적인 활동으로 보고 과거 테스팅 로그를 활용하는 연구로는 J. Kim과 A. Porter의 연구[8]를 들 수 있는데 테스트 케이스 선정 우선 순위를 부여할 때 과거의 테스팅 정보를 반영하는 것이다. 즉, 과거 실행 자료로부터 최근에 사용이 안된 순으로 테스트 케이스에 우선 순위를 정하는 LRU(Least Recently Used) 방법을 제안하고 실험을 통해 제안 방법과 Random, Minimization 방법을 비교하였다. 그들은 실험을 통해 제안 방법이 Random과 Minimization 방법에 비해 오류 검출력이 좋음을 보였다.

본 논문에서는 LRU 연구[8]의 연속선 상에서 과거 테스팅 기록으로부터 이전 테스트 케이스 선정뿐만 아니라 과거 오류 검출 정보를 테스트 케이스 선정 우선 순위화 규칙에 사용하는 새로운 HED(Highly Error Detectable) 선정 방법을 제안한다. 제안 방법은 소프트웨어의 많은 현상들이 Pareto 규칙을 따르는 것처럼, 테스트 케이스들도 이러한 규칙을 따를 수 있다는 가설로부터 비롯된다. 즉 전체 테스트 케이스 중 20%가 전체 오류의 80% 이상의 오류를 검출할 수 있다는 것이다. 실험을 통해 제안 휴리스틱의 성능을 이전의 Random과 LRU 방법의 성능과 비교하였다.

2. 테스트 케이스 우선 순위화

테스트 케이스 우선 순위화는 시간과 자원이 제한적인 상황에서 테스트 케이스에 우선순위를 두어 실행 순서를 정하려는 것이다. 한번의 회귀 테스팅을 위해 주어진 모든 테스트케이스를 실행할 경우 많게는 수 주에서 수 개월에 이르기도 한다. 이런 경우 현실적으로 충분한 테스팅이 어렵기 때문에 테스팅 담당자는 테스트 케이스를 선별하여 실행하고자 할 것이다. 많은 오류를 찾는 테스트케이스를 먼저 실행하면 정해진 시간 내에서 비용대비 효과를 높일 수 있다.

W.Wong 등은 우선순위를 기반으로 두 가지 회귀 테스트 선정 방법을 제안하였다[6]: (1) 변경을 기초로 테스트 케이스를 선정한 후 블록-커버리지를 유지하면서 최소화하는 방법; (2) 변경을 기초로 테스트 케이스를 선정한 후 커버리지 당 추가 비용 오름차순으로 우선 순위화. 그들은 사례 연구를 통해 제안한 두 우선순위 기법이 제한된 환경에서 비용 효율적임을 보였다. 하지만 그들은 우선 순위화 전반에 대한 메커니즘과 방법에 대한 고찰을 기술하지는 못했다.

G.Rothermel 등은 Wong의 연구를 발전시켜 우선 순위화에 대한 개념을 정립하고 일련의 우선 순위 기법을 제안하였다[7]. 우선 순위화는 원하는 특성을 반영하여 정의될 수 있는데, 다음과 같은 것들이 소개되었다.

- 오류 검출 비율을 높이기 위한 방향: 오류를 빨리 찾을 가능성이 높은 테스트 케이스 순으로 순위화
- 심각한 문제를 야기하는 오류의 검출 비율을 높이는 방향: 심각한 문제를 야기할 오류를 찾을 가능성이 높은 순으로 순위화
- 특정 회귀 오류 검출 가능성을 높이는 방향: 이전 단계에서 변경된 특정 코드와 연결된 회귀 오류를 찾을 가능성이 높은 순으로 순위화
- 최소의 테스트 케이스로 커버리지를 높이는 방향: 빨리 커버리지를 모두 만족할 수 있도록 순위화
- 시스템 신뢰도에 대한 확신을 높이는 방향 등이다.

저자들은 오류 검출을 높이기 위한 우선 순위화를 위해 주로 프로그램의 구조 정보를 사용하는 9가지의 기법을 제안하고 이를 여러 개의 서로 다른 프로그램과 테스트 스위트를 대상으로 그 성능을 측정하였다. 실험 결과 제안 우선순위화 방법들이 오류 검출 능력을 향상시킬 수 있음을 보였다.

이전의 두 연구는 전적으로 프로그램의 구조 정보에 기반을 두고 있다. 하지만 회귀 테스팅이 프로그램의 수명 주기 동안 계속해서 반복 실시되는 점을 고려하여 테스트 선정에 과거 테스팅 정보를 활용하려는 또 다른 접근이 제시되었다. 과거 실행 정보에는 특정 테스트 케이스의 실행 기록과 실행 후 오류 정보 등 많은 유용한 정보들이 포함되어 있다. J.Kim과 A.Porter 등은 이전 회귀 테스팅 방법들이 프로그램 내부 정보만을 활용할 뿐 연속적으로 얻어지는 테스팅 로그 정보를 활용하지 못하고 있는 점을 지적하고 새로운 테스트 케이스를 선정방법을 제안하였다[8]. 제안 방법은 테스트 케이스의 이전 실행 기록을 보고 최근에 선정된 적이 없는 테스트 케이스를 우선 선정하는 LRU(Least Recently Used) 방법이다.

저자들은 제한된 시간 내에서 제안 방법을 Random 방법과 문장-커버리지 기준을 적용한 *minimization* 방법을 비교

〈표 1〉 프로그램 프로파일

프로그램 명	함수의 수	줄 수	버전의 수	테스트 풀 크기	평균 테스트 스위트 크기
space	136	6218	30	5179	80
totinfo	7	346	18	1052	92
tcas	9	138	18	1600	84
replace	21	516	15	5542	226
printtokens	18	402	5	4130	128
Printtokens2	19	483	8	4113	65
schedule	18	299	8	2650	107
Schedule2	16	297	8	2710	113

하였다. 실험을 위해 5개에서 30개까지의 서로 다른 개수의 버전을 갖는 8개의 프로그램이 사용되었고, 테스트 풀로부터 Edge-coverage-adequate을 만족하도록 선정된 1000개의 테스트 스위트를 구성하고 이중 100개를 사용하였다. 사용된 프로그램에 대한 정보는 <표 1>과 같다.

각 테스트 스위트는 모든 오류를 찾을 수 있도록 구성되었고 각 대상 프로그램에 대해 50 세션의 회귀 테스팅을 실시하였다. 성능 평가의 척도로는 평균 오류 수명을 사용하였고 실험 결과는 <표 2>와 같다. 그들은 실험을 통해 제안 LRU 방법이 Random에 비해서 오류 검출력과 비용면에서 모두 좋은 성능을 보였지만, Minimization 방법에 비해서는 오류 검출력은 좋으나 비용면에서는 성능이 떨어짐을 보였다.

〈표 2〉 LRU 실험에서의 평균 오류 수명

테스트 케이스 선정비율	Random	LRU	Minimization
5%	8.7	7.4	11.0
10%	5.8	5.2	
20%	3.7	3.5	

저자들은 테스팅 로그를 활용하는 방안으로 실행 정보 이외에 테스트 케이스의 오류 검출 능력에 대한 정보도 활용할 수 있다는 가능성을 제시하였다. 본 논문에서는 LRU 연구의 연속선상에서 새로운 우선 순위화 기법을 제시하고자 한다. 먼저 제안 기법을 소개하고, 이를 LRU 실험에서 사용한 프로그램들을 대상으로 LRU 및 Random 방법과 성능을 비교하였다.

3. 실험 설계

3.1 연구 주제

본 연구를 통해 우리는 다음과 같은 의문을 해결하고자 한다.

- 질문 1: 테스팅 기록 중 실행 및 오류 검출 정보를 활용한 제안 방법이 장기적으로 진행되는 회귀 테스팅의 오류 검출 효과를 높일 수 있나?
- 질문 2: 테스트 스위트의 크기가 성능에 영향을 미치나?
- 질문 3: 어느 정도의 테스트 케이스를 선정할 때 전체 테스트 케이스를 모두 실행한 것과 유사한 성능을 얻을 수 있는가?

3.2 측정 방법

연구 주제에서 언급한 오류 검출 효과를 측정하고 이를 평가할 수 있는 측정 방법이 있어야 한다. 본 논문에서는 평균 오류 수명을 측정 척도로 사용하였다. 오류 수명은 오류가 발생하여 검출될 때까지의 테스팅 세션의 수를 의미한다.

3.3 프로그램

본 실험을 위해 LRU 실험[8]에서 사용한 8개의 프로그램 중 4개의 프로그램을 선택하였다. <표 1>은 대상 프로그램의 함수 개수, 줄 수, 버전의 개수, 테스트 풀 크기를 보여준다. Space 프로그램은 ADL(Array Definition Language)의 인터프리터이고 totinfo는 입력 자료의 통계치를 계산하는 프로그램, Tcas는 비행기 충돌 회피 시스템, replace는 패턴 처리 프로그램으로 개발되었다. 이들 프로그램들은 회귀 테스팅 분야에서 실험 대상으로 주로 사용되는 것으로 space 프로그램은 Vokolos와 Frankl[9]에 의해 그리고 남은 세 프로그램은 Hutchins 등[10]에 의해 처음 사용되었다.

각 프로그램의 버전들은 하나의 오류 만을 포함하고 있고, 오류간에는 상호 간섭이 없이 독립적인 것으로 구성되었다. 다른 4개의 프로그램은 8개 미만의 버전 밖에 가지고 있지 않아 장기간 진행을 가정한 본 실험 환경에 적합하지 않다고 판단되어 제외하였다.

3.4 테스트 스위트

실험 [8]에서 각 프로그램의 테스트 풀로부터 각각 1000개의 작은 테스트 스위트를 만들었다. 각 테스트 스위트는 모든 오류를 다 찾을 수 있도록 하였으며 테스트 스위트 간에는 테스트 케이스의 중복을 허용하였다. 테스트 케이스가 중복된 여러 테스트 스위트들로부터 일정 비율의 테스트 케이스를 선정한다고 할 때, 이는 전체 테스트 케이스들로부터 일정 비율을 선정하는 것과 큰 차이가 있다. 즉, 각 테스트 케이스의 선정 확률이 서로 다르게 된다.

일반적으로 회귀 테스팅은 개발 과정에서 만들어진 모든 테스트 케이스가 실행 대상이므로, 우선 선정을 할 경우에도 모든 테스트 케이스에 동일한 기회가 주어져야 한다고

본다. 따라서 본 실험에서는 전체 테스트 풀을 하나의 단일 테스트 스위트로 설정하여 실험하였다. 이와 같이 전체 풀에서 테스트 케이스를 선정함으로써 여러 개의 작은 테스트 스위트들을 구성하여실험한 실험[8]과 테스트 스위트의 크기에 따른 변화를 비교할 수 있게 되었다.

3.5 테스트 케이스 선정 방법

본 실험에서는 테스팅 기록 중 각 테스트 케이스의 실행 정보와 오류 검출 정보를 활용하는 새로운 휴리스틱을 제안한다. 일반적으로 오류들이 전체적으로 균등하게 분포되기보다는 특정 영역에 집중되는 Pareto 현상을 따르는 것은 알려진 사실이다. 따라서 오류가 특정 부분에 집중되는 만큼 오류를 찾는 테스트 케이스의 오류 검출 능력도 Pareto 규칙을 따를 수 있는 지를 살펴보고자 한다. 즉, 과거 실행 및 오류 검출 정보를 활용하여 가장 오류를 잘 찾을 것 같은 테스트 케이스들을 선정하는 것이다.

새로운 휴리스틱을 정의하기 전에 먼저 세션 t에서의 테스트 케이스 선정 확률 $P_{tc,t}(H_{tc},\alpha)$를 정의한다. 여기서 H_{tc}는 테스트 케이스 tc의 이전 실행으로부터 얻어진 시간 순으로 관측된 값 $\{h_1, h_2, \cdots, h_t\}$들의 모임이다. α는 관측된 개별 값에 대한 가중치 값으로 최근 관측 값에 대한 비중을 높이려면 큰 값을, 이전 관측 값에 대한 비중을 높이려면 작은 값을 사용한다. 확률 P의 수식은 다음과 같다:

$$P_0 = 1$$
$$P_k = \alpha h_k + (1 - \alpha)P_{k-1},\ 0 \le \alpha \le 1,\ k \ge 1$$

여기서 h_k는 다음과 같이 정의된다:
- 테스트 케이스가 선정되어 오류를 찾았으면 1
- 테스트 케이스가 선정되어 오류를 찾지 못했으면 0
- 테스트 케이스가 선정조차 되지 못했으면 β, $0 < \beta < 1$

LRU 실험에서는 h_k가 테스트 케이스의 선정 여부에 따라 0과 1중 어느 한 값이 지정된 것에 반해, HED에서는 테스트 케이스 선정 여부와 오류 검출 여부를 함께 고려하여 h_k가 0, β, 또는 1 중 어느 한 값으로 지정된다. 여기서β는 선정되지 않아 오류 검출 여부를 알 수 없는 테스트 케이스에 대해 주어지는 값으로서, 선정되지 못한 케이스들에 선정 가능성을 높이려면 큰 값을, 그렇지 않으면 작은 값을 사용한다. 따라서 선정 확률, P_k는 이전에 선정되어 오류를 찾은 경력이 많은 테스트 케이스가 선정이 되지 못했거나 선정되어 오류를 찾은 경력이 없는 테스트 케이스보다 더 잘 선정되도록 해준다. 즉 이전에 오류를 잘 찾은 테스트 케이스가 다음 번에도 찾을 가능성이 높다는 가설을 반영한다. 제안 방법은 오류를 잘 찾을 가능성이 높은 순으로 테스트케이스를 선정하자는 의미에서 HED(Highly Error Detectable) 방법으로 명명하였다.

본 논문에서는 HED 우선 순위에 기반한 회귀 테스트 선정 방법을 포함하여 다음과 같은 방법들을 대상으로 제한된 환경에서의 검출 효과를 비교 분석하였다:

- HED(n): 위에서 기술한 실행과 오류 검출 정보를 사용한다. n 값은 시간 제약에 따라 전체 테스트 케이스들을 모두 실행 시키는 것이 아니라 우선순위에 따라 이 중 n% 만이 선정됨을 의미한다.
- LRU(n): Kim 등이 제안한 방법을 구현한 것으로 순차적으로 실행이 안된 순으로 n%의 테스트 케이스를 선정하는 방법이다.
- Random(n): 원래의 테스트 스위트에서 n%의 테스트 케이스를 무작위로 선정하는 방법이다.

3.6 프로그램 진화 모델

본 연구에서는 Kim et al.의 연구와 마찬가지로 회귀 테스팅을 각 세션이 이전 세션의 결과에 의존적인 연속적인 과정으로 본다. 이를 반영하기 위해 프로그램의 제한된 버전을 가지고 진화하는 소프트웨어 시스템을 다음과 같이 모델링하였다: $P_0, P_1, \cdots, P_{n-1}, P_n$. 여기서 P_0는 베이스 버전이고 P_{i+1}는 P_i로부터 단 하나의 변경만을 포함한다. 본 실험에서는 각 프로그램의 오류 조합에 대해 500 세션까지 수행하였다.

본 실험에서는 결과의 신뢰도를 높이기 위해 실험 설계 요소들의 가능한 조합을 모두 고려하였다. 즉, 4개의 프로그램에 대해 500세션을 서로 다른 100개의 버전 시퀀스로 반복 실행하였다.

3.7 실험 구현

위의 설계된 실험을 위해 실험 [8]에서 사용한 프로그램 데이터와 테스트 풀, 테스트 스위트들을 그대로 사용하였으며, LRU와 Random 프로그램은 진화모델 등의 변경에 따른 약간의 데이터만을 조정하였다. HED 프로그램은 기존 LRU와 Random 프로그램의 틀에 맞추어 개발하였다.

4. 실험 결과 분석

각 방법의 성능을 분석하기에 앞서 대상 프로그램의 특성을 살펴보았다. 먼저, 각 프로그램 별로 테스트 케이스들이 얼마나 오류를 잘 찾는 지를 살펴보기 위해 평균 검출 오류 수를 조사하였다. 평균 검출 오류 수는 테스트 케이스가 오류를 찾을 가능성에 대한 것으로 선정된 테스트 케이스의 평균 검출 오류 수가 클수록 선정된 테스트 케이스가 오류를 찾을 가능성도 높다. 다음으로는 전체 테스트 케이스 중 얼마나 많은 테스트 케이스가 오류를 찾는지, 또 그들 중 중복으로 오류를 찾는 테스트 케이스는 얼마나 되는 지를 조사하였다. LRU와 HED 두 방법은 과거 실행 히스토리를 사용하는 방법으로서 LRU는 테스트 케이스를 순차적으로 모두 실행하여 오류 수명을 일정 시간 이내로 보장하려는 접근이고 HED는 오류를 많이 찾는 테스트 케이스에 좀 더 기회를 부여하고자 하는 접근이다. 따라서 이들 두 방법은 하나 이상의 오류를 찾는 테스트 케이스의 양과 연관이 있

〈표 3〉 프로그램과 테스트 케이스 분석 자료

프로그램 이름	줄 수	테스트 풀 크기	오류를 찾은 테스트 케이스(TC)의 수			TC당 평균 검출 오류 수	검출 오류의 표준편차
			0개	1개	2개 이상		
Space	6218	3235	1370(42%)	281	1584	2.01	2.26
Totinfo	346	1026	590(58%)	109	327	1.54	2.42
Tcas	138	1575	943(60%)	365	267	0.59	0.82
Replace	516	5435	4450(82%)	784	201	0.22	0.52

〈표 4〉 평균 오류 수명

선정비율	Space			Totinfo			Tcas			replace		
	Rand	HED	LRU	Rand	HED	LRU	Rand	HED	LRU	Rand	HED	LRU
5%	1.18	1.67	2.24	1.77	9.95	2.71	3.46	3.59	3.32	2.61	2.52	3.19
10%	1.06	1.11	1.45	1.33	3.24	1.21	2.08	2.06	1.97	1.66	1.6	1.87
20%	1.01	1.01	1.18	1.12	1.36	1.22	1.45	1.45	1.38	1.27	1.24	1.33
평균	1.08	1.26	1.62	1.41	4.85	1.71	2.33	2.37	2.22	1.85	1.79	2.13

기 때문에 이 자료가 결과를 이해하는데 중요한 요소가 될 것이다.

<표 3>은 대상 프로그램의 크기, 테스트 풀의 크기, 검출 오류 개수별 테스트 케이스의 개수, 그리고 테스트 케이스의 평균 검출 오류의 수를 보여준다. 테스트 풀의 크기는 <표 1>에서의 값과 차이가 있다. 이는 테스트 스위트의 크기에 따른 성능 변화를 비교 분석하기 위해, 이전 실험에서 사용한 100개의 테스트 스위트에 포함된 테스트 케이스들로만 구성하였기 때문이다. 프로그램별로 테스트케이스들의 평균 검출 오류 수를 살펴보면, space와 totinfo가 1.5보다 크고 tcas와 replace는 0.6 미만이다. 전체 테스트 케이스 중 하나 이상의 오류를 찾는 테스트 케이스의 비율을 살펴보면, space 프로그램의 경우 약 60% 정도에 해당하는 반면 totinfo와 tcas의 경우는 40% 정도이고, replace의 경우에는 18%로 아주 적은 양의 테스트 케이스만이 오류를 찾고 있다. 본 분석에서는 알려진 오류만을 대상으로 하므로 이와 같은 결과로 볼 때 Replace 프로그램은 매우 오류가 적은 신뢰성이 높은 프로그램이라 할 수 있고 상대적으로 Space 프로그램은 오류가 많아 신뢰도가 떨어지는 프로그램이라 할 수 있다.

이들 프로그램을 대상으로 Random과 LRU, HED 방법을 회귀 테스팅에 적용한 후 오류 수명을 조사한 결과는 <표 4>와 같다. 이 표에서 보는 바와 같이 평균 오류 수명이 프로그램의 특성에 따라 다르게 나타나 상호 보완적임을 알 수 있다. 오류를 많이 포함하는Space와 Totinfo 프로그램에서Random 방법이 좋은 성능을 보였다. 이들 프로그램의 경우 테스트 케이스를 균등하게 실행하려는 LRU나 특정 테스트 케이스에 집중하려는 HED에 비해 Random이 좋은 결과를 보인 것으로 분석된다.

Tcas 프로그램의 경우 Totinfo 프로그램과 오류를 찾는 테스트 케이스의 비율면에서는 약 40% 정도로 유사하지만 평균 검출 오류 수는 0.6으로 상당히 작게 나타나 Random 방식의 오류 검출 테스트 케이스의 적중률이 상대적으로 떨어지고 테스트 케이스를 전체적으로 균등하게 실행하려는 LRU가 좋은 성능을 보였다.

오류의 개수도 적고 이들 오류를 찾는 테스트 케이스도 적은 Replace 프로그램에서는 HED 방법이 좋은 성능을 보였다. Replace 프로그램은 신뢰도가 가장 높은 프로그램으로 장기간 실행을 통해 오류를 찾을 수 있는 특정 테스트 케이스에 집중할 수 있었기 때문으로 보인다. 또한 HED 방법은 전반적으로 20% 선정 비율에서 좋은 성능을 보이고 5% 선정 비율에서는 좋지 않은 성능을 보였다. 선정 비율이 너무 작으면 초반부에 선정되어 오류를 찾은 테스트 케이스가 다음 세션에 계속 선정될 가능성이 많아져 선정된 적이 없는 테스트 케이스가 선정될 기회가 적어지기 때문으로 분석된다.

또 한가지 주목할 만한 사실은 이전 실험[8]과 테스트 스위트의 크기를 달리한 이번 실험의 결과를 비교할 때 <표 5>와 같이 오류 수명에서 큰 차이가 있음을 알 수 있다. 그리고 모든 프로그램에서 Random 방법이 전반적으로 상당히 좋은 결과를 보인다. 이전의 연구[8]에서 Random 방법이 오류 수명면에서 LRU에 비해 낮은 결과를 보인 것과 비교하면 큰 차이가 있다. 테스트 스위트의 크기 변화에 따른 결과의 변화로 볼 수 있다.

또 다른 중요한 결과는 모든 프로그램에서 전체 테스트

〈표 5〉 테스트 스위트 크기를 달리한 두 실험간 평균 오류 수명의 비교

선정비율	실험 [8]의 평균 오류 수명		HED 실험의 평균 오류 수명	
	Random	LRU	Random	LRU
5%	8.7	7.4	2.3	2.9
10%	5.8	5.2	1.5	1.6
20%	3.7	3.5	1.2	1.3

케이스의 20%를 실행하는 경우 평균 오류 수명이 1.2정도에 불과하다는 것이다. 이는 오류가 생성되어도 바로 다음 테스팅 세션이나 그 다음 번 세션에서 모두 검출될 수 있음을 의미한다. 테스트 케이스를 모두 실행해서 얻을 수 있는 오류 수명 1에 비해 크게 떨어지지 않는 값이다. 즉, 시간 제약이 많은 경우 선정 방법과 무관하게 전체 테스트 케이스의 20% 정도의 실행으로도 높은 오류 검출 효과를 볼 수 있다는 것이다.

5. 결 론

본 실험을 통해 여러 가지 흥미로운 사실을 발견할 수 있었다. 첫째는 우선 순위화 방법들이 프로그램의 특성에 따라 성능 면에서 상호 보완적인 특성을 보인다는 점이다. 오류를 찾는 테스트 케이스들을 많이 갖고 있는 프로그램의 경우에는 Random이 효과적이고, 상대적으로 오류를 찾는 테스트 케이스의 비율이 작은 경우에는 제안된 HED방법이 좋은 성능을 보였으며, 중간 정도인 경우에는 LRU 방법이 효과적이었다. 둘째는 전체적인 성능이 테스트 스위트의 크기에 영향을 많이 받는다는 점이다. 테스트 스위트의 크기를 달리하여 실험한 결과 오류의 수명 값과 그 성능 순위에 있어 큰 차이를 보였다. 마지막으로 20% 정도의 테스트 케이스의 만을 실행하여도 전체 테스트 케이스 모두를 실행한 것과 성능 면에서 유사한 결과를 얻을 수 있다는 점 등이다

본 실험을 통해 우리는 프로그램의 특성에 따라 이들 세 방법이 서로 다른 성능을 보여 상호 보완적이라는 결과를 얻었다. 이런 현상으로부터 우리는 오류가 많은 회귀 테스팅의 초기 세션에서는 Random 방법을, 어느 정도 신뢰도가 높아지면 LRU 방법을, 많은 회귀 테스팅 세션으로 신뢰도가 높다고 판단되는 경우엔 HED 방법을 적용하는 것도 가능하리라 본다. 하지만 실제 개발 환경에 적용하기 위해서는 이와 같은 가설을 포함하여 다양한 특성의 프로그램을 대상으로 한 객관적인 더 많은 실험 자료가 필요하다. 특히 프로그램의 진화 과정을 잘 보여줄 수 있는 프로그램을 대상으로 한 많은 실험들이 이루어져야 한다.

참 고 문 헌

[1] M.J. Harrold and M.L. Soffa, "An Incremental approach to unit testing during maintenance," In Proc. of the Conf. on Software Maintenance, pp.362-367, Oct., 1988.

[2] J. Hartmann and D. Robson, "Techniques for selective revalidation," IEEE Software, Vol.16, No.1, pp.31-38, 1990.

[3] T. Ostrand and E. Weyuker, "Using Dataflow Analysis for Regression Tesing," In Six Annual Pacific Northwest Software Quality Conference, pp.233-247, Sep., 1988.

[4] G. Rothermel and M.J. Harrold, "A safe, efficient regression test selection technique," ACM TOSEM, Vol.6, No.2, pp. 173-210, Apr., 1997.

[5] G. Rothermel and M.J. Harrold, "Empirical studies of a safe regression test selection technique," IEE TSE, Vol. 24, No. 6, pp.401-419, Jun., 1998.

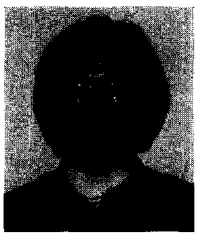
[6] W.E. Wong, J.R. Horgan, S. London, and H. Agrawal. A Study of Effective Regression Testing in Practice, In proc. of the 8th Int'l. Symp. on Software Reliability Engineering, pp.230-238, Nov., 1997.

[7] G. Rothermel, R. Untch, C. Chu, and M.J. Harrold, Test case prioritization: an empirical study. In proc. of Int'l. Conf. on Software Maintenance, 1999.

[8] J.-M. Kim and A. Porter, A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environment, In proc. of the 24th Int'l. Conf. on Software Engineering, 2002.

[9] F.I. Vokolos and P.G. Frankl, Empirical evaluation of the textual differencing regression testing technique, In Proc. of the Int'l Conf. on Software Maintenance, 1998, pp.44-53.

[10] M. Nutchins, H.Foster, T.Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow-and controlflow-based test adequacy criteria," In Proc. of 16th Int'l Conf. on Software Engineering, pp.126-135, Jan., 2000.



소 선 섭

e-mail : triples@kongju.ac.kr

1986년 이화여자대학교 전산학과(학사)

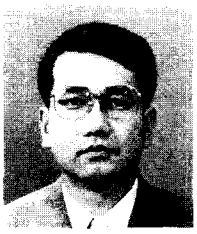
1988년 한국과학기술원 전산학과(석사)

2001년 한국과학기술원 전산학과(박사)

1988년~1995년 국방과학연구소 연구원

1995년 현재 공주대학교 컴퓨터공학부 부교수

관심분야 : 소프트웨어 테스팅, 회귀 테스팅, 임베디드 소프트웨어 등



채 의 근

e-mail : ygchae@kongju.ac.kr

1986년 경북대학교 통계학과(이학사)

1988년 한국과학기술원 전산학과(공학석사)

1988년~1996년 한국전자통신연구원 선임연구원

1996년~1997년 ㈜데이콤 종합연구소 선임연구원

1997년~현재 공주대학교 컴퓨터공학부 조교수

관심분야 : 독립성분분석, 패턴인식, 화자인식, 통계적자료분석