임베디드 소프트웨어를 위한 과거 이력 기반 테스트 케이스 순위화 기법

백창현° 태상원 김영상 신승훈 박승규
아주대학교 정보통신전문대학원
{labmedia°, bicumuri, guri92, shinsh, sparky}@ajou.ac.kr

A Test Case Prioritization Technique for Embedded Software using Fault History

Changhyun Baek°, Sangwon Tae, Youngsang Kim, Seunghoon Shin, Seungkyu Park
Graduate School of Information and Communication, Ajou University

요 약

소프트웨어의 구조가 점차 복잡해짐에 따라 소프트웨어 테스트 과정에서 테스트가 일정 수준 이상의 테스트 커버리지를 갖게하기 위해서는 많은 수의 테스트 케이스 실행이 불가피하며, 이로 인해 테스트 수행의 시간 비용이 증가되고 있다. 하지만 테스트 프로세스 안에서 어느 시점에 소프트웨어 결함을 발견하느냐에 따라 소프트웨어 배포 시점에서의 오류 수정 비용이 달라진다. 이를 위해 각각의 테스트 케이스에 우선순위를 부여하여, 보다 빠른 시간 내에 결함을 찾고자 하는 테스트 케이스 순서화 기법에 대한 연구가 활발히 진행되고 있다. 본 논문에서는 임베디드 소프트웨어의 시스템 테스트 결과를 활용한 과거 이력 기반 테스트 케이스 순서화 기법을 제안한다.

1. 서 론

정보통신 분야의 기술 발전과 함께 기술 융합 서비스에 대한 사회적 요구로 인해 상용 소프트웨어는 점차 복잡해지고 다양화되었다. 이와 같은 소프트웨어의 복잡화는 소프트웨어 테스팅 분야에서도 동일한 복잡성과 어려움을 야기하고 있다. 소프트웨어가 복잡도가 증가함에 따라 일정 수준의 테스트 커버리지를 만족하는 테스트 결과를 산출해내기 위해서는 더욱 많은 수의 테스트 케이스를 생성하여 테스트를 수행해야 한다. 그러나 이는 테스트 과정 상의 문제뿐만 아니라, 전체 개발 일정에 직접적인 영향을 주는 요소이므로 시간 비용 대 오류 검출의 관점에서 보다 높은 효율성을 제공하는 방식으로 테스트를 수행해야 한다.

따라서 정해진 테스트 수행 시간 내에 검출 가능한 오류 검출 비율을 최대화 하기 위한 테스트 케이스 순서화 기법에 대한 연구가 활발하게 진행되고 있다. 고전적인 개념의 소프트웨어 복잡도인 LOC(Line Of Code) 뿐만 아니라, 각 컴포넌트간의 결합도 또는 요구사항의 복잡함과 같은 개념을 바탕으로한 다양한 테스트 케이스 순서화 기법이 제시되고 있다. 이와 같은 방식을 통한 연구는 하나의 소프트웨어 내부에서 좀더 복잡한 부분일수록 소프트웨어 오류가 존재할 가능성이 높다는 것에서 출발한다.

본 논문에서는 각 테스트 케이스에 우선순위를 부여하기 위한 방법으로, 과거에 수행된 테스트 결과를 이용한 우선순위 계산법을 제안한다. 본 논문의 구성은 다음과 같이 구성되어 있다. 먼저 2절에서는 테스트 케이스 순서화 기법이 필요한 이유와 기존에 제시된 테스트 케이스 순서화 기법에 대해 설명하였다. 3절에서는 소프트웨어 분석과 과거 이력기반의 가중치 계산 방식을 제시하였다. 마지막으로 4절에서는 결론과 향후 과제를 제시하고 있다.

2. 테스트 케이스 순위화

소프트웨어 품질향상을 위해 각 기업은 많은 시간 및 인적 비용을 투자하여 테스트를 수행하고 있다. 아래 그림은 국내 기업의 테스트 자동화 도구[1]를 이용한 단위 임베디드 소프트웨어에 대한 시스템 테스트를 수행하는데 요구된 시간을 통계로 나타내고 있다.

그림1에서 보여주는 결과와 같이 대부분의 기업에서 진행하는 테스트 프로세스 내에서 테스트 케이스를 수행하는데 소요되는 시간은 전체 테스트 프로세스 중 35% 이상을 차지하고 있다. 또한 "테스트 수행"과 직접적으로 관련이 있는 "오류 리뷰 및 수정" 시간의 경우 테스트 수행 도중 오류가 발생한 시점부터 시작되므로, 만약 테스트 수행의 마지막 단계에서 소프트웨어 오류가 발생하게 된다면 테스트 프로세스는 보다 많은 시간이 요구될 수 밖에 없다. 이에 따라 특별한 경우에는 발견된 오류를 수정하지 못하고, 제품을 상용화하는 경우도 발생한다. 따라서 테스트 수행의 효율성 증대를 위해 보다 빠른 시간 내에 효율적으로 소프트웨어 오류를 발견할 수 있도록 하는 테스트 케이스 순서화 기법에 대한 연구가 불가피하며, 이에 따라 테스트 프로세스 전반에 걸쳐 영향을 미치는 테스트 케이스 간의 우선 순위에 대한 평가를 제시하는 테스트 케이스 순서화

기법에 대한 많은 연구가 진행되고 있다.

Hema Srikanth 등은 고객-할당 우선순위와 요구사항 복잡도 등을 기반으로 가중치를 계산하는 방식을 제시하였다[2]. 그러나 이와 같은 방식은 고객과 개발자에 의해 제시되는 가중치를 테스트에 이용하므로 테스트에 이용되는 데이터가 데이터 생성자 중심이라는 한계를 가지기 때문에, 순서화 과정이 특정 전문가 의존적이 되는 단점을 가진다. 따라서 이 방식은 비전문가 집단에 의해 우선 순위가 부여될 경우 오히려 순서화를 적용하지 않은 경우보다 비효율적인 결과를 초래할 수 있다.

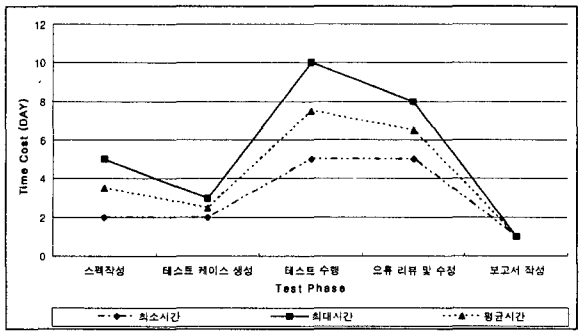

그림 1 테스트 단계별 소요시간

Sebastian Elbaum 등은 함수 호출 개수에 따라 테스트 케이스를 순서화하고, 또한 코드의 변화를 이용하여 회귀 테스트(Regression Testing)에서 변화된 부분에 가중치를 부여하는 방식을 제안하였다[3]. 그러나 이 방식은 화이트 박스 테스트에만 국한된 방식으로 블랙박스 기반의 시스템 테스트에는 적합하지 않다.

Bogdan Korel 등은 코드 기반으로 테스트 우선 순위를 정하는 방법을 대신해 state-based 모델을 사용하여 Regression Testing시 우선 순위를 정하는 기법을 제시하였다[4]. 기존의 모델과 변경된 모델의 차이점을 기반으로 우선 순위를 정하지만 위 방법을 사용하기 위해서는 시스템이 모델링 언어를 사용해 기술되어 있어야 하는 제약이 있다.

3. 임베디드 소프트웨어의 이력 분석

과거에 발생된 테스트 이력은 동일한 도메인 내에서 해당 로직이나 함수가 오류를 발생시킬 위험도로 간주될 수 있으며, 기존 연구에서의 복잡도(SW Complexity)와 같이 테스트 케이스의 우선 순위화 기준으로 활용될 수 있다. 즉, 과거에 발생한 오류 중 대부분이 특정한 로직에 연관된 것이라면, 해당 로직과 테스트 케이스 간의 연관성에 대한 가중치를 높여주어 오류 발생 확률이 높은 테스트 케이스의 우선 순위를 높여주어야 한다. 본 논문에서는 상용 임베디드 시스템에 탑재된 소프트웨어에 대한 과거 1년간의 시스템 테스트 결과를 바탕으로 한 임베디드 소프트웨어 테스팅에 과거 이력을 적용하는 방법을 제시한다.

3.1 테스팅 결과 분석

임베디드 소프트웨어는 기존 PC 상에서 작동하는 소프트웨어와는 다르게 소프트웨어의 하드웨어 연관성 등이 고려되어져야 하므로 오류를 일반화하는 과정에 제약이 존재한다. 따라서 단위 도메인에서 발생되는 오류의 특성을 파악하는 것이 매우 중요하다. 본 논문에서 제시하는 테스팅 기법의 대상이 되는 임베디드 시스템은 여러 개의 스위치와 모터, 액추에이터, 센서, 인디케이터, 디스플레이 등으로 구성된 제품으로 정의한다.

그림 2는 평문 형태로 제공되는 사용자 매뉴얼을 바탕으로 소프트웨어 내부에 존재하는 로직을 정의하고, 이렇게 정의된 각 로직이 발생시키는 소프트웨어 오류 발생 빈도를 나타내고 있다. 이때 정의된 각각의 로직은 실제 소프트웨어 내부 구조와는 상이할 수 있으나, 블랙박스 테스트 관점에서 외부 입력과 내부 코드 사이의 관계를 정의하는 것은 불가능하므로 이와 같은 형태의 접근은 불가피하다.

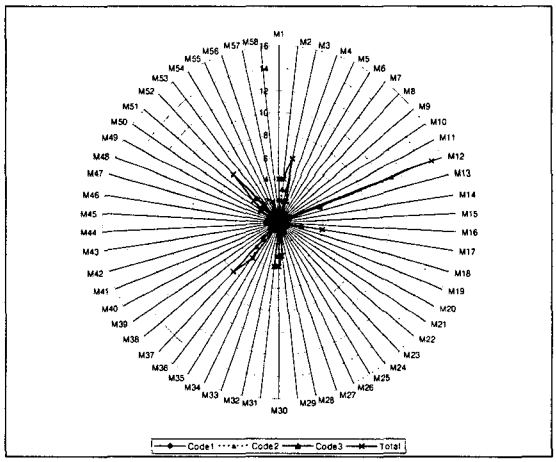

그림 2 모듈 별 소프트웨어 오류 발생

그림2에서 나타난 것과 같이 정의된 로직 별 오류 발생 수는 M3, M12, M37, M52 등에서 그 회수가 현저히 많음을 알 수 있다. 이러한 통계적 결과를 바탕으로 해당 소프트웨어 내부의 취약점을 찾을 수 있다. 또한, M3, M12, M37, M52는 특정 입력(특수한 스위치나 특정 센서의 값)에 따라 작동됨을 발견할 수 있다. 실제 M12와 M37은 유해 가스에 따른 에어컨 시스템의 작동을 수행하는 액추에이터와 디스플레이의 의사결정을 수행하는 로직이다. 이 경우 유해 가스 정도를 판단하는 센서나 사용자가 공기정화를 지시할 수 있는 스위치에 의해 각 모듈의 수행 결과가 결정된다. 즉, End Item의 오류 빈도는 그 만큼 해당 End Item을 제어하는 하나의 상의 로직의 취약성으로 판단할 수 있으며, 따라서 취약한 로직과 관련된 입력이 포함된 테스트 케이스는 보다 높은 우선 순위를 가져야 한다는 사실을 알 수 있다.

3.2 테스트 케이스와 과거 이력간의 관계 도출

임베디드 시스템에서 단위 End Item(하나의 디스플레이 또는 액추에이터 등과 같은 하드웨어 구성품)의 작동방식 결정에는 하나 이상의 소프트웨어 로직이 아래 그림 3과 같이 관여된다. 또한 하나의 내부 로직은 하나 이상의 입력값을 기반으로 End Item에 대한 의사결정을 한다.

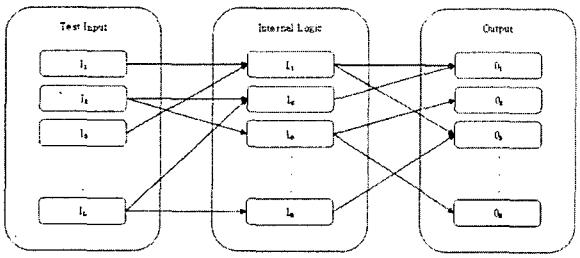

그림 3 테스트 입력과 내부로직 및 출력 관계

즉, 하나의 출력 결과를 결정하기 위해 사용되는 로직을 찾을 수 있으며, 본 논문에서는 그러한 로직을 Dominant Logic(DL)이라고 정의한다. 예를 들어 그림 3에서 O1에 대한 DL 리스트는 L_1과 L_2로 구성된다. 또한 하나의 사용자 입력에 대해 이를 이용하는 로직을 Control Logic(CL)이라고 정의할 수 있으며, 예를 들어 I2에 대한 CL 리스트는 L_2과 L_3으로 구성 된다.

이와 같이 DL과 CL를 해당 소프트웨어에 대해 구성 완료한 후 각각의 Test Input의 상대적 중요도를 아래 와 같이 평가할 수 있다. 먼저 Oi의 오류 발생 이력 횟수를 이용하여 $FREQ_{Oi}$를 계산하고, DL_i의 개수는 NUM_{DLi}라 정의할 때 모든 DL_i에 대해 L_i가 갖는 중요도(Impact Factor, IF)는 아래 수식과 같이 계산될 수 있다. 단, 여기서 $SEVERITY_{Oi}$는 해당 End Item의 오류 발생 시 시스템에 미치는 심각도를 반영하기 위한 상수이다.

$$IF_{Li} = \frac{1}{NUM_{DLi}} \sum_{i=1}^{N} FREQ_{Oi} * SEVERITY_{Oi}$$

수식 1 단위 로직의 IF

하나의 테스트 케이스는 일련의 입력의 연속으로 고려되어질 수 있기 때문에, 하나의 테스트 케이스를 구성하는 모든 입력에 대한 IF의 합으로 우선 순위를 산출할 수 있다. 즉, 아래 수식과 같이 단위 입력 Oj에 대한 Pj는 아래 수식2와 같이 Oj에 관련된 CL리스트의 IF합을 CL 리스트에 포함된 로직의 개수로 나눈 결과와 같다.

$$P_j = \frac{1}{NUM_{CLj(Li \in CLj)}} \sum_{i=1}^{M} IF_{Li}$$

수식 2 단위 입력의 P

위의 수식2에 의해 계산된 Pj는 하나의 입력에 대한 우선 순위 척도이다. 하나의 테스트 케이스는 여러개의 순차적인 입력값으로 구성되므로, 단위 테스트 케이스의 최종 우선 순위는 각 입력에 대한 P의 합을 입력의 개수로 나눈 것과 같다.

이를 통해 계산된 각각의 테스트 케이스에 대한 우선 순위 점수를 순차적으로 정렬한 후, 정렬된 순서에 맞게 테스트를 수행 하면 과거 이력이 반영된 테스트를 진행할 수 있게 된다.

4. 결론 및 향후 과제

본 논문에서는 과거 이력을 이용한 테스트 케이스 우선순위를 생성하여, 소프트웨어 내부의 취약성을 찾고 이를 통해 테스트 프로세스 내에서 조기에 오류를 발견할 가능성을 높이는 방법을 제시하였다. 그림 4는 제시된 방법을 바탕으로 우선순위에 따라 실행된 테스트 케이스의 오류 발견 시간을 제시하고 있다. 그림에서 보여주는 결과는 과거 1년간의 데이터 중 6번의 테스트 결과 데이터를 과거 이력으로 활용하고 7번째 테스트 수행 결과와 우선 순위에 따른 발견 시간을 나타낸 것이다.

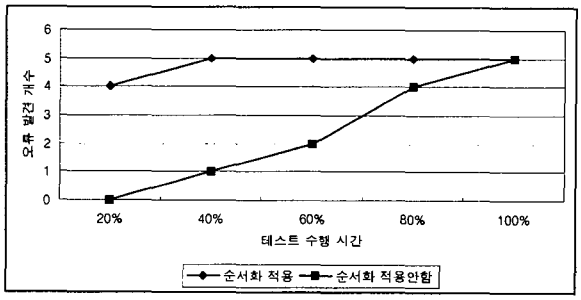

그림 4 테스트 수행 시간과 오류 발견

테스트 케이스의 정렬을 통한 소프트웨어에 내재하는 오류의 조기 발견은 보다 많은 오류 수정 시간을 제공하여 보다 높은 시스템 안정성을 추구할 수 있도록 해주며, 또한 시스템 개발 프로세스의 지연을 줄이는 데 기여를 할 수 있다.

그러나, 본 논문에서 제시된 방법은 하나의 테스트 케이스 내부에 존재하는 각각의 입력들이 서로 독립적이라는 전제를 바탕으로 한다. 이들 각각에 대한 의존성 고려 방안 등은 향후 연구 과제이다.

참고문헌

[1] Changhyun Baek, Seungkyu Park, and Kyunghee Choi, "TEST: An Effective Automation Tool for Testing Embedded Software", WSEAS Trans. On Information Science & Applications, Issue8, Volume2, August 2005.

[2] Hema Srikanth and Laurie Williams, Requirements-Based Test Case Priotization.

[3] Sebastian Elbaum, Gregg Rothreml, and Satya Kanduri, Selecting a Cost-Effective Test Case Prioritization Technique, Software Quality Journal, 12, 185-210, 2004.

[4] Korel B., Tahat L.H., Harman, and M., "Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on26-29 Sept. 2005.