유닛 테스트 기법:
LG전자 상업용 디스플레이 실제 사례 연구

김윤호 [O1], 홍기창 [2], 김문주 [1]

[1] 한국과학기술원, [2]LG 전자

kimyunho@kaist.ac.kr, kichang.hong@lge.com, moonzoo@cs.kaist.ac.kr

Unit Testing Techniques:
A Real-world Case Study of LG Commercial Display

Yunho Kim[O1], Kichang Hong[2], Moonzoo Kim[1]

[1]School of Computing, KAIST [2]LG Electronics

요 약

유닛 테스팅은 소프트웨어가 올바르게 작성되었는지 점검하기 위해 널리 사용되고 있다. 하지만 기존 유닛 테스팅 기법은 개발자가 직접 유닛 테스트를 작성해야 했기 때문에 개발자의 시간과 노력이 많이 들고 개발자가 생각하지 못한 시나리오에서 발생하는 오류를 찾기 어려웠다. 이런 문제를 해결하기 위해 자동으로 유닛 테스트를 생성하는 기법이 제안되었으나 특히 C 프로그램의 경우 아직 산업체에서 쉽게 적용할 수 있을 만큼 기법 및 도구가 성숙하지 않아 많은 산업체에서 자동 유닛 테스트 생성 기법을 도입하지 못하고 있다.

본 논문에서는 LG전자 상업용 디스플레이를 구성하는 유닛에 유닛 테스팅을 적용하였다. 개발자가 테스트 드라이버 및 입력 값을 직접 작성하는 매뉴얼 테스팅, 테스트 드라이버는 직접 작성하지만 입력 값은 자동 생성되는 Concolic 테스팅, 테스트 드라이버 및 입력 값을 자동 생성하는 자동화된 Concolic 유닛 테스팅 세 기법을 적용하고 유닛 테스팅 수행에 필요한 준비 시간 및 달성한 커버리지를 비교하였다. 그 결과 다차원 배열 및 복잡한 구조체를 입력으로 받는 유닛의 테스트 드라이버를 직접 작성할 경우 자동화된 Concolic 유닛 테스팅 기법보다 테스트 작성 시간이 더 오래 소요되었으며 더 낮은 커버리지를 달성하였다.

1. 서 론

유닛 테스팅은 소프트웨어가 올바르게 작성되었는지 점검하기 위해 널리 사용되고 있다. 유닛 테스팅은 개발 초기 단계부터 적용 가능하기 때문에 소프트웨어 오류를 조기에 검출할 수 있으며 소프트웨어의 작은 단위를 테스트하기 때문에 오류의 원인을 추적하여 수정하기 쉬운 장점이 있다. 하지만 기존 유닛 테스팅은 개발자가 직접 유닛 테스트 드라이버와 입력 값을 작성하고 테스트 스텁(stub)을 작성해야 했기 때문에 개발자의 시간과 노력이 많이 들고 개발자가 생각하지 못한 시나리오에서 발생하는 오류를 찾기 어려웠다.

이런 문제를 해결하기 위해 자동으로 유닛 테스트를 생성하는 기법[1]이 제안되었다. 하지만 기법 및 도구 성숙도가 산업체에서 바로 도입하여 사용할 수 있을 정도로 높지 않아 대부분의 산업체에서는 아직 개발자가 직접 유닛 테스트 드라이버/입력 값을 작성하고 있다. 특히 C 프로그램의 경우 자동화된 유닛 테스팅을 지원하는 도구가 미흡하여 많은 산업체에서 자동 유닛 테스트 생성 기법을 도입하지 못하고 있다.

본 연구는 LG전자 CTO부문 SW공학연구소 PIT팀, 한국연구재단 한-아프리카 협력기반조성사업(NRF-2014K1A3A1A09063167), 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업(IITP-2016-H8501161012000100 2), 초소형, 고신뢰(99.999%) OS와 고성능 멀티코어 OS를 동시 실행하는 듀얼 운영체제 원천 기술 개발(R0101-15-0081)의 지원으로 수행되었음

본 논문에서는 LG전자 상업용 디스플레이를 구성하는 유닛에 유닛 테스팅을 적용하고 자동화된 유닛 테스트가 효과적으로 테스트를 생성할 수 있는지 확인하는 사례 연구 결과를 보여준다. 개발자가 테스트 드라이버 및 입력 값을 직접 작성하는 매뉴얼 테스팅, 테스트 드라이버는 직접 작성하지만 입력 값은 자동 생성되는 Concolic 테스팅, 테스트 드라이버 및 입력 값을 자동 생성하는 자동화된 Concolic 유닛 테스팅 세 기법을 적용하고 유닛 테스팅 수행에 필요한 준비 시간 및 달성한 커버리지를 비교하였다. 그 결과 다차원 배열 및 복잡한 구조체를 입력으로 받는 유닛의 테스트 드라이버를 직접 작성할 경우 자동화된 Concolic 유닛 테스팅 기법보다 테스트 작성 시간이 더 오래 소요되었으며 더 낮은 커버리지를 달성하였다.

2. 비교 대상 유닛 테스트 기법

본 장에서는 사례 연구에서 수행한 세 종류의 유닛 테스트 생성 기법을 소개한다.

2.1 매뉴얼 유닛 테스트 기법

매뉴얼 유닛 테스트 적용 기법은 테스트 드라이버와 테스트 입력 값을 개발자가 직접 작성하는 유닛 테스트 기법이다. 매뉴얼 유닛 테스트는 크게 테스트 드라이버 및 입력 값

작성과 테스트 스텁 작성으로 나뉘어져 있다.

테스트 드라이버는 테스트 대상 유닛의 선행 조건(예. 2진 탐색을 수행하는 binsearch()함수의 입력 배열은 항상 정렬되어 있어야 한다 등)을 만족하도록 전역 변수 및 파라메터 변수 값들을 설정하고 테스트를 수행하고자 하는 테스트 대상 유닛을 실행하는 함수이다. 각각의 테스트 대상 유닛마다 1개 이상의 테스트 드라이버를 작성하게 되며 테스트 대상 유닛에서 개발자가 테스트 하고자 하는 코드 범위 및 기능에 따라 테스트 드라이버 및 입력 값 설정이 달라지게 된다.

테스트 스텁은 테스트 대상 유닛을 격리하여 실행할 수 있도록 테스트 대상 유닛이 호출하는 실제 함수를 대체하는 함수를 나타낸다. 유닛 테스트는 테스트 범위를 최소화 하는 것이 목적이고 다른 구성 요소가 완성되기 전에도 수행될 수 있기 때문에 테스트 대상 유닛 함수가 호출하는 실제 함수를 스텁 함수로 대체하여 수행하게 된다. 스텁 함수는 실제 기능을 수행하는 함수를 단순화하여 테스트에 필요한 결과값만 반환하게 하거나 미리 정해진 간단한 입력/출력에 대해서만 동작하는 등으로 구현된다.

2.2 Concolic 테스트 기법

Concolic 테스트 기법[2]은 동적 분석과 정적 분석 기법을 결합하여 자동으로 프로그램의 모든 실행 가능한 경로를 테스트하는 테스트 케이스를 생성하는 기법이다. 2.1절의 매뉴얼 유닛 테스트 기법과 비교하여 테스트 드라이버 및 스텁 함수를 개발자가 직접 작성하는 것은 동일하지만 테스트 입력 값을 Concolic 테스트 기법을 적용하여 자동으로 생성하는 점이 다르다. 개발자가 테스트 드라이버를 작성할 때 테스트 대상 유닛의 입력 변수 중 자동으로 입력 값을 생성하길 원하는 변수를 심볼릭 입력 변수로 지정하면 Concolic 테스트 도구가 동적 분석을 수행하며 테스트 대상 유닛의 모든 분기를 실행하도록 테스트 입력 값을 자동으로 생성한다.

2.3 자동화된 Concolic 유닛 테스트 기법

자동화된 Concolic 유닛 테스팅 기법은 대상 프로그램의 유닛 테스트 드라이버/스텁 함수를 자동으로 생성하고 해당 유닛 테스트의 입력 값을 동적 기호 실행 기법을 사용하여 자동으로 생성하는 기법이다. 대상 함수 f()의 모든 파라메터와 f()가 사용하는 모든 전역 변수를 동적 기호 실행의 심볼릭 입력 값으로 설정하고 f()가 호출하는 모든 함수를 심볼릭 값을 리턴하는 심볼릭 스텁 함수로 대체한 후 생성된 테스트 드라이버/스텁 함수에 동적 기호 실행 기법을 적용해 테스트 입력 값을 생성한다. 대상 함수 f()의 파라메터와 전역 변수의 자료형에 따라 표 1과 같이 심볼릭 입력 값을 설정한다. 기본 자료형의 경우 해당 자료형에 맞는 심볼릭 입력 값으로 설정하고 배열의 경우 원소의 자료형에 맞는 심볼릭 입력 값을 모든 원소에 설정한다. 구조체의 경우 모든 구조체 필드에 대해 해당 필드의 자료형에 맞는 심볼릭 입력 값을 설정한다. 포인터의 경우 해당 포인터가 가리키는 자료형 크기만큼 메모리를 할당하고 가리키는 자료형에 따라 심볼릭 입력을 설정한다. 단 포인터의 경우 구조체 포인터 등으로 인해 무한히 심볼릭 입력을 설정할 수 있기 때문에 최대 2번만 포인터 참조를 수행하여 입력 값을 생성한다.

표 1 심볼릭 입력 설정

입력 값 자료형	심볼릭 입력 설정 방법
기본 자료형	해당 자료형에 맞는 기호 실행 입력 값 설정
배열	모든 배열 원소에 대해 배열 원소의 자료형에 맞는 기호 실행 입력 값 설정
구조체	구조체의 모든 필드에 대해 해당 자료형에 맞는 기호 실행 입력 값 설정
포인터	포인터 자료형 T가 가리키는 *T의 메모리 크기만큼 메모리를 할당하고 *T의 동적 기호 실행 입력을 설정 하여 포인터 입력 값으로 할당. 만약 *T가 구조체여서 중첩 구조체를 형성하는 경우 참조 깊이가 2(예: a->b->c)이하인 경우만 심볼릭 입력 값을 생성하고 2 초과인 경우 NULL 할당

3 유닛 테스트 적용 설정

유닛 테스트 적용 대상은 LG전자에서 개발 중인 상업용 디스플레이에 탑재되는 디스플레이 제어 소프트웨어의 한 유닛 함수이다. 해당 함수는 C 언어로 작성되었으며 173줄의 코드로 구성되어 있고 124개의 분기들(1개의 if문이 2개의 분기로 구성)로 구성되어 있다.

유닛 테스트는 LG전자의 경력 5년차 개발자가 수행하였다. 3.2절의 Concolic 테스트 기법을 적용하기 위해서는 Concolic 테스트 기법에 대한 이론적 배경과 도구 사용 방법에 대한 이해가 필요하기 때문에 Concolic 테스트 강연 및 실습을 이수한 개발자가 Concolic 테스트 기법을 사용한 유닛 테스트 및 자동화된 Concolic 유닛 테스트를 수행하였다. 매뉴얼 유닛 테스트를 위해 C/C++ 유닛 테스트 프레임워크인 CppUTest[3]를 사용하였다. Concolic 테스트 기법과 자동화된 Concolic 유닛 테스트 기법에서 Concolic 테스팅 도구로는 CREST-BV[4]를 사용하였다. 테스트 드라이버/스텁 자동 생성 도구는 Clang/LLVM[5]을 사용하여 개발하였다.

4 유닛 테스트 적용 결과

4.1 유닛 테스트 환경 설정 및 테스트 작성 시간 비교

표 2의 2번째 열과 3번째 열은 각각 테스트 환경 설정 소요 시간과 테스트 코드 작성 소요 시간을 나타낸다. 매뉴얼 유닛 테스트의 경우 CppUTest 프레임워크를 설치하고 테스트 대상 유닛을 격리 하여 CppUTest 환경에서 빌드하도록 빌드 환경을 수정하는데 6시간의 시간이 소요되었다. Concolic 테스트의 경우 Concolic 테스트 도구인 CREST-BV를 설치하는데 4시간의 시간이 소요되었다. 개발 환경이 격리되어 있어 외부 라이브러리 설치가 제한적이어서

표 2 테스트 수행 결과

비교 항목	매뉴얼 유닛 테스트	Concolic 테스트	자동화된 Concolic 유닛 테스트
환경 설정 소요 시간	6시간	4시간	8시간
개발자의 테스트 코드 작성 시간	12시간	8시간	0시간
개발자의 테스트 코드 줄 수	1607	18753	0
달성 분기 커버리지	68%	83%	79%

의존성이 있는 라이브러리 설치에 대부분의 시간이 소요되었다. 자동화된 Concolic 유닛 테스트 역시 CREST-BV의 설치와 테스트 생성시 사용한 Clang/LLVM을 설치하는데 8시간의 시간이 소요되었다. Concolic 테스트가 4시간으로 가장 짧고 자동화된 Concolic 유닛 테스트가 8시간으로 가장 긴 환경 설정 시간이 필요했지만 유닛 테스트 환경 설정의 경우 초기 셋업이 완성되고 난 후에는 기존 테스트 환경을 재활용해서 사용할 수 있기 때문에 실제 유닛 테스트 수행시 세 유닛 테스트 적용 방법 사이의 차이가 미미하다.

테스트 작성 시간의 경우 매뉴얼 유닛 테스트가 12시간이 소요되어 가장 긴 시간이 걸렸으며 자동화된 Concolic 유닛 테스트의 경우 테스트 드라이버/스텁 함수를 자동으로 생성하기 때문에 0시간이 소요되었다. 매뉴얼 유닛 테스트의 경우 총 1607줄의 테스트 코드를 작성하였으며 Concolic 테스트의 경우 18753 줄의 테스트 코드를 작성하였다.

매뉴얼 유닛 테스트의 경우 개발자가 원하는 코드 부분을 실행하기 위한 입력 값을 개발자가 생각하는데 많은 시간이 소요되었다. 특히 테스트 대상 함수의 입력 값이 수 십 ~ 수 백 개의 필드가 존재하는 구조체들로 구성된 연결 리스트로 구성되어 있기 때문에 특정 코드를 실행하기 위해선 연결 리스트를 구성하는 복잡한 구조체들의 값을 설정해 줘야 한다.

Concolic 테스트의 경우 개발자가 수 천 개의 원소를 갖는 다차원 배열의 각 원소를 심볼릭 입력으로 설정하고 수 십 ~ 수 백 개의 필드가 존재하는 구조체들의 각각의 필드를 심볼릭 입력으로 설정하였기 때문에 18753 줄의 테스트 코드가 작성되었다. 반면 매뉴얼 유닛 테스트와 달리 특정 코드를 실행하기 위한 입력 값을 계산할 필요가 없이 Concolic 테스트 기법을 사용하여 자동으로 입력 값이 생성되기 때문에 입력 값 계산에 필요한 시간이 매뉴얼 유닛 테스트보다 적게 소요되었다.

자동화된 Concolic 유닛 테스트 기법은 테스트 코드를 자동으로 생성하기 때문에 테스트 코드 작성에 필요한 시간이 0이었다. 환경 설정 소요 시간과 달리 테스트 코드 작성 시간은 테스트 대상 유닛 함수가 증가할수록 비례하여 증가하기 때문에 자동화된 Concolic 유닛 테스트 기법이 매뉴얼 유닛 테스트 및 Concolic 테스트 기법보다 월등히 효율적이다.

4.2 유닛 테스트 커버리지 결과 비교

매뉴얼 유닛 테스트, Concolic 테스트 및 자동화된 Conclolic 유닛 테스트는 각각 68%, 83%, 79%의 분기 커버리지를 달성하였다. 매뉴얼 유닛 테스트의 경우 개발자가 제한된 시간 내에 모든 분기를 실행하는 테스트 입력 값을 생각하지 못했으며 제한된 개발 일정으로 인해 추가적인 테스트를 수행하지 못하였다. Concolic 테스트의 경우 모든 분기를 달성하는 것을 목표로 테스트를 수행하기 때문에 매뉴얼 유닛 테스트보다 높은 83%의 분기를 커버할 수 있었다. 17%의 분기문은 현재 CREST-BV에서 지원하지 않는 심볼릭 배열 인덱스 등의 제약사항으로 인해 커버하지 못하였다. 자동화된 Concolic 유닛 테스트의 경우 79%의 분기문을 커버하였다. Concolic 테스트보다 4%p 낮은 분기 커버리지를 달성하였다. 이는 2.3절의 포인터 입력 설정 방법에서 참조 깊이가 2가 넘는 포인터의 경우 NULL로 설정하여 무한 반복을 막아 참조 깊이가 2가 넘는 포인터 참조가 필요한 입력을 생성하지 못한 것이 원인이었다. Concolic 테스트와 자동화된 Concolic 유닛 테스트 모두 매뉴얼 유닛 테스트보다 높은 커버리지를 달성하여 자동 테스트 입력 생성 기법이 개발자가 작성한 입력 값보다 효과적인 테스트 입력 값을 생성했음을 확인하였다.

5. 결론 및 향후 연구

본 논문에서는 매뉴얼 유닛 테스트, Concolic 테스트를 사용한 유닛 테스트 및 자동화된 Concolic 유닛 테스트 기법을 LG전자 상업용 디스플레이 사례 연구를 통해 비교하였다. 유닛 테스트 수행 결과 매뉴얼 유닛 테스트가 가장 높은 테스트 코드 작성 시간이 필요했으며 반면 달성 분기 커버리지는 가장 낮았다. 자동화된 Concolic 유닛 테스트의 경우 Concolic 테스트를 사용한 유닛 테스트보다 약간 낮은 커버리지를 달성했으나 테스트 코드 작성 시간이 전혀 필요하지 않아 가장 비용대비 효과가 높았다.

향후 연구로는 자동화된 Concolic 유닛 테스트 기법을 다른 상업용 디스플레이 제어 코드의 더 다양한 함수에 적용하고 테스트 오라클을 작성하여 효과적으로 버그를 탐지하는 산학 연구를 수행할 예정이다.

참고문헌

[1] Kim et al., "Automated unit testing of large industrial embedded software using concolic testing", ASE, 2013

[2] Sen et al., “CUTE: A Concolic Unit Testing Engine for C”, FSE, 2005

[3] CppUTest: http://cpputest.github.io/

[4] 김윤호 외 2인, "CREST-BV: 임베디드 소프트웨어를 위한 Bitwise 연산을 지원하는 향상된 Concolic 테스팅 기법", 정보과학회 논문지, Vol. 40, No. 2, 2013

[5] Clang/LLVM: http://clang.llvm.org/