| Oral Session IX : Big Data, Smart Energy ICT, Smart Information

좌장 : 신춘성 (전남대)

요구사항 기반의 구조화된 테스트 설계 방법 제시

김기두, 김영철
한국정보통신기술협회, 홍익대학교
e-mail : kdkim@tta.or.kr, bob@hongik.ac.kr

Structured scope coverage for test design mechanism

Kim Kidu, R. Young Chul Kim
Telecommunication Technology Association, Hongik University

요 약

소프트웨어 테스팅은 개발 요구사항을 만족했는지 확인하기 위해 반드시 필요하다. 그리고 요구사항 만족 여부를 확인하기 위해서는 누락된 테스트케이스가 없도록 많은 테스트케이스를 개발해야 한다. 하지만, 요구사항만으로 테스트케이스를 모두 도출하는 것은 어렵다. 이를 해결하기 위해, 요구사항으로부터 유스케이스, 유스케이스 시나리오 등의 하위 레벨의 구조화된 방법을 통해 최대한 많은 수의 테스트케이스를 도출하는 방안을 제안하고자 한다.

1. 서 론

소프트웨어 개발 전 주기에서 테스트 단계는 소프트웨어 품질을 좌우하는 매우 중요한 단계이다. 또한, 개발 요구사항을 만족했는지 확인하는 마지막 단계이기도 하다. 따라서, 테스트 수행은 보다 철저히 수행해야 한다. 하지만, 입력과 사전 조건의 모든 조합을 고려하여 모든 가능성을 고려한 테스트를 수행하는 것은 일반적으로 불가능하다고 한다. 이를 위해, 기존에는 적은 테스트 케이스로 높은 테스트 커버리지를 갖는 방법들을 연구하였다. 하지만, 현재 개발되는 소프트웨어들은 복잡하고, 대형화된 소프트웨어들로 개발되고 있다. 테스트 수행을 위해 테스트 케이스를 줄이지 않고, 보다 많은 테스트 케이스를 도출할 수 있어야 한다. 하지만, 기존에 수해한 것과 같이 단순히 요구사항을 기반으로 테스트 케이스를 도출할 경우 실제 필요한 테스트케이스들이 누락될 수 있다. 이를 해결하기 위해, 요구사항에서부터 유스케이스, 유스케이스 시나리오 등의 구조화된 방법을 통해 많은 테스트 케이스를 도출하는 방안을 제안하고자 한다.

2. 관련 연구

2.1 기존 테스트 설계 기법

테스트 설계는 개발 산출물을 통해 테스트 케이스를 도출하고 정의하는 활동을 말한다. 테스트 설계는 테스트를 어떻게, 또는 얼마만큼 수행하는 지에 따라 다양한 방법들이 존재한다. 관련하여, 구문 커버리지(Statement Coverage), 결정 커버리지(Decision Coverage), 조건 커버리지(Condition Coverage), 변경 조건/결정 커버리지(MC/DC Coverage), 다중 조건 커버리지(Multiple Condition Coverage) 등의 다양한 방법들이 있다.

1) 구문 커버리지: 소스코드 전체 구문 중에서 실행되는 구문을 대상으로 테스트를 수행하는 방법
2) 결정 커버리지: 결정 커버리지는 소스코드 전체 분기 중에서 실행되는 분기를 대상을 테스트를 수행하는 방법
3) 조건 커버리지: 조건 커버리지는 모든 조건식에서 True 또는 False 중 실행된 조건을 대상으로 테스트를 수행하는 방법
4) 조건/결정 커버리지: 조건/결정 커버리지는 실행된 개별 조건식에서 True, False와 분기 모두를 테스트하는 방법
5) 변형 조건/결정 커버리지: 변형 조건/결정 커버리지는 다른 조건식과 무관하게 분기를 결정하는 개별 조건식의 True, False 중 테스트에 의해 실행된 것을 테스트하는 방법

앞서 언급한 구문 커버리지, 결정 커버리지, 조건 커버리지 조건/결정 커버리지, 변형 조건/결정 커버리지의 포함관계는 아래 그림 1과 같다.

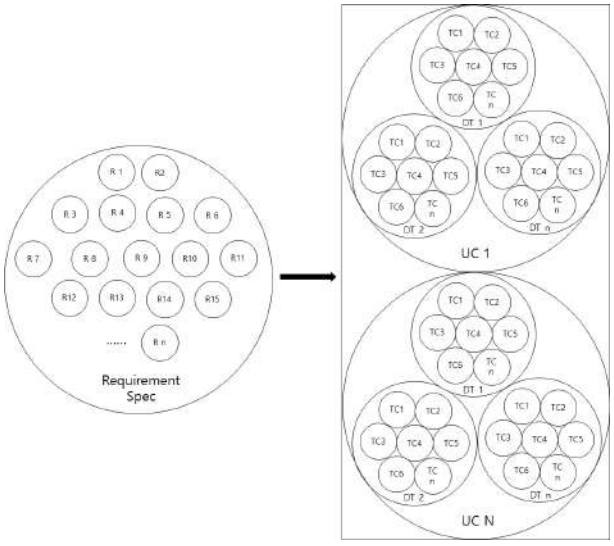
(그림 1) 커버리지 포함관계

3. 구조화된 테스트 설계 방법

관련 연구에서 언급한 기존의 테스트 설계기법들은 소스코드를 기반으로 수행된다. 이럴 경우, 코드 자체에 대한 테스트 수행이 되지만, 현재의 객체기반의 시스템에 적용하기에는 올바르지 않다.[1] 기존 유스케이스 기반으로 테스트 케이스를 도출하는 방법은 유스케이스 다이어그램부터 연속되는 과정을 통해 생성하는 방법이다.[2], [3] 하지만, 기존의 방법은 테스터가 확인해야할 테스트케이스를 줄이기 위해 사용한 방법이다. 하지만, 현재의 소프트웨어 제품은 대형화, 복잡화되어 보다 테스트케이스가 누락되지 않도록 상세한 수준으로 작성되여야 한다. 또한, 유스케이스 단계에서부터 테스트케이스를 도출하여, 요구사항이 고려되지 않는다. 이를 해결하기 위해서는 구조화된 테스트 설계 방법을 통해 테스트 케이스를 도출해야한다.

(그림 2) 구조화된 테스트케이스 설계 절차

그림 2는 요구사항으로부터 테스트케이스를 도출하는 절차를 보여준다. 요구사항 단계에서 유스케이스(유스케이스 시나리오)에 필요한 액터 및 시나리오를 식별한다. 이후, 유스케이스에 필요한 입력/출력/조건 값을 찾아 이후 단계인 시퀀스를 찾을 수 있다. 또한, 앞서 식별한 유스케이스와 시퀀스를 통해 상태 및 객체 정보를 찾을 수 있으며, 결국엔 테스트케이스까지 도출할 수 있다. 이를 통해, 개발되는 소프트웨어에 사용되는 모든 객체, 상태, 흐름을 고려하게되어, 그림 3과 같이 요구사항과 테스트케이스 생성에 관한 개념을 보여준다.



(그림 3) 구조화된 테스트 설계 기법을 통한 테스트케이스 생성

구조화된 테스트 설계 방법을 통해 최종적으로는 아래 표 1과 같이 n개의 요구사항에 대해 n개의 테스트케이스가 도출됨을 확인할 수 있다. 요구사항으로부터 식별되는 요소들과 최종적으로 테스트케이스 도출 형태를 표로 보여준다.

(표 1) 구조화된 테스트 설계 기법을 통한 테스트 설계

4. 결론

요구사항 기반의 테스트케이스 및 소스코드 기반으로 테스트케이스는 객체 기반의 소프트웨어 또는 시스템에서 요구하는 다양한 상태 및 흐름이 누락될 수 있다. 하지만, 본 논문에서 제안하는 구조화된 테스트 설계 방법은 소프트웨어 요구사항에서부터 유스케이스, 메시지 시퀀스 등의 절차를 통해 테스트케이스 필요 요소들을 식별한다. 구조화된 테스트 설계 방법은 누락된 테스트케이스가 발생되지 않고, 많은 테스트케이스를 도출할 수 있도록 지원한다.

참 고 문 헌

[1] 김동호, "이종 테스트 프로세스 메커니즘의 추상화 테스트케이스 커버리지 성숙도 모델", 2014, pp. 2

[2] Hyun Seung Son, Woo Yeol Kim, Jae Seung Kim, R. Young Chul Kim, "Automatic Test Case Generation using Multiple Condition Control Flow Graph", Information Technology and Computer Science 2012, ASTL 13, pp. 105-109.

[3] Dong Ho Kim, Hyun Seung Son, Woo Yeol Kim, R. Young Chul Kim, “Test Case Extraction for Intelligent Power Switch Heterogeneous Vehicles”, Information Technology and Computer Science 2012, ASTL 13,