GUI 기반 기능 테스트 자동화 스크립트의 재사용성 향상 기법

김성빈*, 김희천*
*한국방송통신대학교 대학원 정보과학과
e-mail:hckim@knou.ac.kr

A Technique to increase reusability of GUI-based Functional Test Automation Scripts

Sung-Vin Kim*, Hee-Chern Kim*
*Dept. of Computer Science, Graduate School, Korea National Open University

요 약

기능 테스트 자동화를 위해서는 스크립트 구현에 많은 노력이 소요된다. 또 구현된 스크립트의 재사용성이 떨어져 소프트웨어 UI가 변경되면 수정이 필요하고 여러 어려움이 따른다. 본 연구의 목적은 GUI 기반 시스템 테스트 자동화의 특성과 문제점을 분석하여 테스트 스크립트의 재사용성을 향상시키는 방법을 제시하고 적용 사례를 보임으로써 테스트 자동화 작업의 생산성을 높이고 안정적이고 신뢰성이 있는 자동 테스트 결과를 확보하는데 있다.

1. 서론

소프트웨어 테스트는 소프트웨어 개발 생명주기에서 중요한 부분을 차지하고 있으며 그 역할과 중요성이 커지고 있다. 그러나 테스트 작업은 최근에도 사람에 의해 수동으로 진행되는 경우가 많다. 대다수의 개발 프로젝트가 인력, 시간 및 자원 등의 제약사항을 가지고 있으며 이로 인해 소프트웨어 테스트 작업도 제한된 일정 안에서 효율적 활동을 통해 일정 수준 이상의 품질 확보를 추구해야 한다[1].

대부분의 개발 조직들은 테스트를 통한 품질 향상을 위해 테스트 기간을 늘리거나 테스트 전문 인력을 충원하는 방식을 택한다. 그러나 이러한 대응만으로 일정 수준 이상의 테스트 품질을 확보하는 것은 어렵다. 그 이유는 기본적으로 테스트의 품질이 테스트를 수행하는 사람의 경험과 자질에 좌우되기 때문이다. 결국 기존 방식인 사람에 의한 테스트 수행 방식은 일정 수준 이상의 테스트 품질을 지속적으로 확보하는 것을 매우 어렵게 한다[2].

현업에서는 단기적인 비용절감 문제가 아니라 장기적인 관점에서 테스트 품질을 향상시켜서 궁극적으로 품질 활동의 총 비용을 감소시키기 위해 접근하다. 그래서 테스트 업무에 대한 자동화 필요성을 인식하였고 이러한 테스트 자동화의 필요성은 자동화 구현을 위한 상용 도구들의 기능을 지속적으로 향상시키는 요인이 되어 왔다[3]. 테스트 자동화의 대상을 고려할 때 개발 단계에서 소스코드 수준의 함수나 메서드 등을 테스트하는 경우에 테스트 자동화의 성공률과 재사용률이 높다. 그러나 실제 테스트 작업에서는 완성된 소프트웨어를 대상으로 GUI 환경에서 테스터가 대상 응용 프로그램을 사용함으로써 테스트를 수행하는 경우가 매우 흔하다[4]. 이러한 GUI 기반의 기능 테스트 자동화는 그 필요성에도 불구하고 테스트 자동화의 성공률은 매우 낮은 편이다. 가장 큰 원인은 사용자 인터페이스의 변경에 따라 기존 구현된 자동화 스크립트를 재사용할 수 없어서 자동 테스트를 수행할 때마다 계속 기존 스크립트를 수정해야 하는 문제 때문이다. 이로 인해 전체적으로 높은 유지보수 비용이 발생하고 결국 테스트 자동화를 중지하게 되며, 이러한 사례는 테스트 자동화에 대한 불신과 회의로 이어져 심지어 테스트 자동화를 시도조차 하지 않게 하는 원인이 되기도 한다[5]. 그럼에도 불구하고 현업에서는 테스트 자동화를 통한 테스트 효율성 제고와 테스트 품질 확보에 대한 필요성이 지속적으로 요구되고 있다.

테스트 자동화 스크립트의 재사용성을 향상시킨다면 효율적인 자동 테스트 수행이 가능하고 이를 통하여 테스트 품질 확보가 가능할 것이다. 본 연구에서는 GUI 기반의 테스트 자동화에서 발생하는 문제점 중 사용자 인터페이스 변경에 따른 기능 위치 변경으로 인해 스크립트 재사용성이 떨어지는 문제점에 대한 해결방안을 제시함으로써 테스트 자동화의 성공률을 높이는데 기여하고자 한다.

2. 관련 연구

2.1 테스트 자동화

소프트웨어 테스트의 자동화 분야에 대한 연구는 크게 단위 테스트 자동화와 GUI 기반의 기능 테스트 자동화로 나눌 수 있다[6]. 기존 연구를 살펴보면 GUI 기반의 기능

테스트 자동화 보다는 주로 단위 테스트 자동화 연구에 치중 되어 있다. 단위 테스트란 개발 단계에서 개발자가 구현한 함수나 메서드 등을 대상으로 테스트하는 것이다. 단위 테스트 자동화의 장점은 일반적으로 소프트웨어의 UI에 영향을 받지 않아서 한번 구현된 자동화 코드의 수정 요구가 적고 재사용성이 높으며, 또 안정적이고 지속적인 자동 테스트 수행이 가능하다는 점이다. 단위 테스트 자동화의 주요 연구 내용으로는 제한적이기는 하나 테스트데이터와 기대 결과 등을 자동생성 해주는 것 즉, 테스트케이스를 자동 생성하는 것에 관한 연구가 주류를 이룬다. 반면 GUI 기반의 기능 테스트는 완성된 소프트웨어를 대상으로 테스트하며, 일반적으로 사람이 소프트웨어의 기능을 UI를 통해 의도된 대로 조작하고 그 결과 값을 기대결과와 비교하여 테스트를 수행하는 방식이다. 즉 GUI 기반의 기능 테스트 자동화란 사람이 소프트웨어를 사용하는 행위(마우스와 키보드 입력)와 사람이 소프트웨어의 실제 결과와 기대 결과를 비교하는 과정을 그대로 자동화하는 것을 의미한다. 이러한 GUI 기반의 기능 테스트 자동화 연구는 구체적인 기법의 연구보다는 주로 거시적이고 전략적인 차원에서 연구가 진행되었다.

2.2 기능 분해 전략

현재까지 연구된 GUI 기반의 기능 테스트 자동화 전략은 기능 분해 전략과 키워드 주도 전략이 있다. 기능 분해 전략은 테스트 수행의 단위를 기본적인 업무 단위로 나누고 테스트 자동화 스크립트를 구현함으로써 업무 단위로 스크립트를 관리하고 이를 재조합 또는 재사용함으로써 전반적인 테스트 자동화 스크립트의 재사용성을 높이는 전략이다[7]. 보통 구현된 업무 단위의 스크립트를 액션 스크립트라 하고 이를 다시 테스트 스텝 코드와 테스트 체크로직, 그리고 테스트데이터로 분리한다. 특히 테스트 데이터는 외부 파일 형태로 존재한다. 테스트 스텝 코드란 테스트를 수행하기 위해서 해당 기능 위치까지 이동하거나 특정 입력용 UI에 값을 설정하는 과정 등이 기록된 스크립트를 의미한다. 테스트 체크로직은 사전에 확보된 기대 결과와 자동 테스트를 통해 나타난 응용프로그램의 수행 결과를 비교하는 코드이다. 테스트데이터는 별도의 외부 파일로 관리되는데, 테스트 액션 스크립트 내의 테스트 스텝 코드를 실행하는 시점에서 이 파일의 테스트데이터를 인자로 넘겨줌으로써 결과적으로 다양한 테스트데이터에 대해 테스트 액션 스크립트가 수행될 수 있게 된다. 즉 1개의 업무 스크립트가 여러 테스트데이터에 대해 수행됨으로써 다양한 테스트 작업을 수행할 수 있다. 이러한 과정을 통해 스크립트 재사용성과 테스트 품질을 향상 시킬 수 있다.

2.3 키워드 주도 전략

기능 분해 전략 외에 키워드 주도 전략이 있다. 기능 분해 전략에서 테스트 액션 스크립트를 실행한다는 것은 동일 업무 내에서 1개 또는 관련이 있는 여러 개의 테스트 케이스를 외부 파일로부터 읽어 들여 자동으로 테스트하는 것을 의미한다. 그러나 테스트 액션 스크립트 단위들을 조합하여 큰 규모의 테스트케이스나 테스트 시나리오를 만들려면 자동화 스크립트 내의 코드를 수정해야만 하는 문제에 직면한다. 이를 해결하기 위해 테스트 액션 스크립트 별로 특정한 키워드를 이용하여 이름을 정한다. 예를 들어 은행의 계좌이체를 테스트하는 스크립트에 대해서는 “계좌이체”라는 키워드로 이름을 정하고, 잔액조회를 테스트하는 스크립트에는 “잔액조회”로 이름을 정한다. 이후 테스트 자동화에 대한 사전 지식이 없는 테스트 인력이 엑셀 파일 내에서 “계좌이체”나 “잔액조회” 등의 키워드를 기술하면 테스트 자동화 도구는 엑셀 파일을 읽은 후 계좌이체 스크립트와 잔액조회 스크립트를 수행하는 것이다. 즉 키워드 주도 전략에서는 테스트 인력이 엑셀 파일에 사전에 정의된 키워드들을 다양하게 기술함으로써 테스트 케이스를 설계할 수 있고, 이를 통해 자동 테스트의 수행이 가능하다. 이러한 전략의 장점은 크게 두 가지가 있는데, 첫 번째는 액션 단위의 조합을 통한 테스트케이스 설계를 코딩 작업 없이 엑셀 파일 안에 키워드들의 기술만으로 가능하게 하여 전체적인 자동화 비용을 줄일 수 있다는 점이며, 두 번째는 테스트 자동화 엔지니어가 응용프로그램이 보여주는 행위 부분의 구현에만 집중하고 테스트케이스 설계 부분은 자동화 도구와 상관없이 수동 테스터에 의해서 설계가 가능하게 하다는 점이다. 이러한 전략적인 바탕위에서 테스트 자동화가 구축되면 테스트 자동화 소스는 업무 영역에 상관없이 빠른 커스터마이징이 가능해져 전체적인 테스트 자동화 비용이 축소된다.

3. GUI 기반 기능 테스트 자동화

3.1 GUI 기반 기능 테스트 자동화 방법

일반적인 GUI 기반의 기능 테스트 자동화 과정은 자동화 도구가 사용자의 마우스나 키보드 등의 입력 이벤트를 기록하고, 재생하는 방식이다. 또한 자동화 도구는 윈도우 내 컨트롤의 속성 정보 등을 인식하여 각각을 구별할 수 있어서 윈도우 내 특정 컨트롤의 위치가 변경되어도 스크립트를 수정할 필요 없이 정확한 재생이 가능하다.

(그림 1) 계산기 사용1

(그림 2) 계산기 사용2

(그림 3) 십진수 10

(그림 4) 이진수 1010

(그림 1)~(그림 4)의 과정은 윈도우 운영체제에 기본 설치되어 있는 계산기를 사용하여 10진수 "10"을 2진수 "1010"으로 변환하는 과정이다. 자동화 도구를 실행시킨 상태에서 위 과정을 그대로 따라하면 자동화 도구는 해당 액션을 스크립트 형태로 기록하게 된다.

```
Activate("계산기")
Window("계산기").MenuSelect("보기";"공학용").click()
Window("계산기").Button("1").click
Window("계산기").Button("0").click
Window("계산기").RadioButton("Bin").click

Gettextdata=window("계산기").editbox("결과값")
If Gettextdata="1010" then test result = "PASS"
Else test result = "FAIL"
```

(그림 5) 테스트 스크립트

(그림 5)는 생성된 자동화 스크립트를 보여준다. 계산기를 활성화 시키고, 메뉴를 따라 "공학용" 계산기 기능으로 이동한다. 그 다음 버튼 "1", "0"을 차례로 입력한 후 라디오 버튼으로 되어 있는 "Bin"을 선택함을 알 수 있다. (그림 5)의 아래쪽 3줄은 직접 코딩을 통해 작성된 부분으로 테스트 기대 결과와 실제 결과 값을 비교하는 체크 로직이다. (그림 5)에서 보듯이 테스트 데이터인 10진수 "10"의 입력과정인 "1" 과 "0"이 스크립트 내에 하드코딩이 되어있고 또한 역시 테스트 기대 결과인 "1010"이 하드코딩이 되어있다. 즉 테스트 스텝 코드와 테스트데이터 등이 함께 존재하는 상황이다. 이로 인해 본 스크립트의 재사용성은 10진수 "10"을 2진수 "1010"으로 변환하여 정상인지를 판정하는 경우에 한해서만 사용할 수 있다. 본 스크립트의 재사용성을 높이기 위해 재사용 가능한 부분을 구분하여 코드와 데이터를 분리하면 (그림 6)에 나타난 것처럼 함수나 서브루틴 형태로 변경된다.

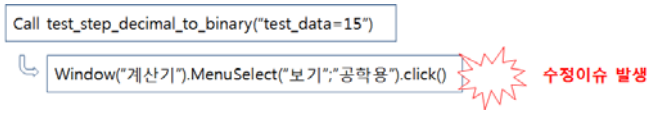
```
Call Run 계산기

Call test_step_decimal_to_binary("test_data=15")

Call test_check_decimal_to_binary(test_check_data=1111")
```

(그림 6) 테스트 스텝 코드와 체크 로직의 분리

본 논문에서는 지면 관계상 서브루틴을 호출하는 인자에 테스트데이터를 하드 코딩 했으나 실제 구현 형태는 테스트데이터를 엑셀 파일 등의 외부 파일로부터 읽어 들여 해당 함수의 호출시 인자로 전달하게 된다. 이러한 함수와 서브루틴들은 다시 라이브러리 형태로 구성되어 재사용 가능한 부품의 형태를 취하게 된다.

3.2 GUI 기반 기능 테스트 네비게이션 자동화의 문제점

테스트 네비게이션이란 테스트를 위해 해당 기능의 위치까지 이동하는 과정을 의미한다. 다른 관점에서 보면 테스트 스텝 코드 영역 중 테스트를 위해 특정 기능 위치로 이동하는 부분에만 한정된다. (그림 1)~(그림 4)의 계산기를 통한 테스트 자동화 예제에서 만일 "공학용" 계산기의 기능 위치가 변경되었다면, 즉 스크립트가 구현되는 시점에서는 「메뉴→보기→공학용」과 같았는데 UI 변경에 따라 기능 위치가 변경되어 「메뉴→공학용」과 같이 변경되었다면 스크립트 재수행 시점에서 실패가 발생할 것이다. 스크립트를 살펴보면 아래 (그림 7)과 같이 메뉴 선택 부분에 "보기";"공학용"과 같이 하드코딩이 되어있기 때문에 결국 UI 변경에 따라 해당 코드 영역, 즉 테스트 네비게이션 영역을 수정해주어야 하는 문제가 발생한다.



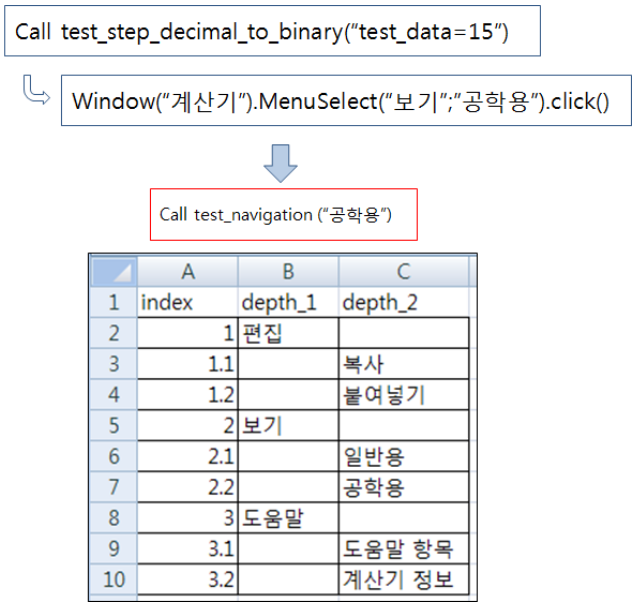
(그림 7) UI 변경으로 인한 문제 발생

이와 같이 기능 위치 변경에 해당하는 UI 변경은 풀다운 메뉴 뿐만이 아니라 버튼과 윈도우 그리고 대화상자를 통하여 계층구조를 갖는 모든 UI 형태에서 발생할 수 있으며, 특히 개발 단계에서 빈번하게 발생한다.

3.3 네비게이션 자동화 설계기법 제안

테스트 네비게이션 문제 중 첫 번째는 네비게이션 정보가 스크립트 내에 하드코딩이 되어있다는 것이고, 두 번째는 소프트웨어의 기능 위치가 수시로 변경된다는 것이다. 이것을 해결하기 위해서는 기능 위치 정보를 코드 내에서 데이터 파일로 분리시켜야 한다. 즉 테스트 네비게이션 함수를 만든 후 테스트 대상의 기능 이름을 인자로 전달함으로써 테스트 네비게이션 함수가 해당 기능 위치를 자동으로 찾아가도록 구현해야 한다. 이를 통해 한번 구현된 테스트 자동화 스크립트의 네비게이션 코드 영역은 수정이슈가 줄어들거나 없을 수도 있다. 아래 (그림 8)은 계산기 프로그램의 메뉴 구조를 엑셀 파일로 분리하였으며 test_navigation() 서브루틴은 인자 값 "공학용"과 엑셀 파일의 메뉴 구조 정보를 사용하여 런타임에 해당 기능을 자동으로 선택한다. 또한 기능 위치가 수정 되었을 때, 즉 계산기의 경우 풀다운 메뉴 구조가 수정되었을 때 테스트 자동화 스크립트를 수정하는 것이 아니라 메뉴 구조가 정의된 엑셀 파일만을 수정함으로써 스크립트 재사용성을

높일 수 있어 전체적으로 유지 보수비용이 감소된다.



	A	B	C
1	index	depth_1	depth_2
2	1	편집	
3	1.1		복사
4	1.2		붙여넣기
5	2	보기	
6	2.1		일반용
7	2.2		공학용
8	3	도움말	
9	3.1		도움말 항목
10	3.2		계산기 정보

(그림 8) 기능 이름을 사용하여 코딩하며 UI 변경에 대비하여 메뉴 구조 정의 파일을 분리

이러한 테스트 네비게이션은 풀다운 메뉴뿐만 아니라 윈도우가 윈도우를 포함하고 있는 계층 구조에서도 응용할 수 있다. 제안된 기법이 테스트 수행 시 필요한 네비게이션을 일종의 데이터로 분리하여 관리하고 활용함으로써 테스트 자동화의 효율화를 꾀하고 있으나 만일 네비게이션 구조 자체가 빈번히 변화하고 규모가 커진다면 이러한 네비게이션 데이터를 유지 및 보수하는 것도 큰 노력과 비용이 따를 수 있다. 따라서 향후 연구가 필요한 부분은 근본적으로 자동화 도구가 대상 에플리케이션이나 서버스를 직접 탐험하여 네비게이션 구조를 만드는 방법이 강구되어야 할 것이다.

4. 결론

GUI 기반의 테스트 자동화는 필요성에도 불구하고 실제 현업에서 적용된 성공 사례를 찾기 힘들다. 이러한 이유는 자동화 도구의 한계에서 생기는 문제도 있지만 구체적이고 효과적인 테스트 자동화 기법을 적용하여 자동화를 구현하지 못하는데도 기인한다. 즉 테스트 자동화의 비효율적인 구현으로 인하여 높은 유지보수 비용이 발생하고 이로 인해 테스트 자동화가 실패하는 경우가 많다고 판단된다. 본 연구는 소프트웨어 기능 위치 변경에 따른 테스트 자동화 스크립트의 수정을 최소화시키는 기법을 제안하여 GUI 기반 테스트 자동화 스크립트의 재사용성과 유지보수성을 향상시키고자 하였다. 본 논문에서 제안된 기법을 실무에 적용해 보았을 때와 그렇지 않을 때를 비교해 보면 테스트 자동화의 효율성은 분명한 차이를 보였다.

한편 근본적으로 테스트 자동화가 한 단계 높은 차원으로 발전하려면 개발 과정의 초기부터 자동 테스트가 고려되어 소프트웨어가 개발되어야 한다. 예를 들어 현재 방식은 테스트데이터를 엑셀 파일 등에서 읽어 들이는 방식이지만 테스트 자동화가 고려된 소프트웨어는 자동화 도구가 객체를 인식하는 과정에서 객체의 속성 정보에 테스트 데이터가 사전에 준비되어 있고 이것을 바탕으로 테스트를 수행할 수 있다. 이렇게 된다면 한번 구현된 테스트 스크립트는 거의 수정되지 않고 바로 실무에 적용할 수 있을 것이다. 따라서 소프트웨어 개발 과정 초기부터 GUI 수준의 기능 테스트 자동화를 고려한 설계와 구현이 필요하다.

참고문헌

[1] Myers, Glenford J., Corey Sandler, and Tom Badgett, *The art of software testing*, John Wiley & Sons, 2011.

[2] Dustin, Elfriede, Thom Garrett, and Bernie Gauf, *Implementing automated software testing: How to save time and lower costs while raising quality*, Pearson Education, 2009.

[3] 이은학, "오픈마켓의 기능테스트 자동화에 관한 사례연구," 숭실대학교 정보과학대학원 석사학위 논문, 2012.

[4]Memon, Atif M, "GUI testing: Pitfalls and process," *Computer*, vol.35, no.8, 2002, pp.87-88.

[5] Memon, Atif M., Martha E. Pollack, and Mary Lou Soffa, "Hierarchical GUI test case generation using automated planning," *IEEE Transactions* on *Software Engineering*, vol.27, no.2, 2001, pp.144-155.

[6] Jorgensen, Paul C. *Software testing: a craftsman's approach*, CRC press, 2013.

[7] Li, Kanglin, and Mengqi Wu, *Effective GUI testing automation: Developing an automated GUI testing tool*, John Wiley & Sons, 2006.