디지털산업정보학회 논문지 제11권 제2호-2015년 6월
http://dx.doi.org/10.17662/ksdim.2015.11.2.001

단위테스트 중 매개변수 경계오류제거를 위한 코드 자동생성 시스템 설계와 구현*

박 영 조** · 방 혜 자***

Design and Implementation of code generator to remove the parameter boundary failure in the unit test

Park Youngjo · Bang Hyeja

〈Abstract〉

As programs get more complicated and they are developed by various hands, the possibility that there are program bugs in the code has been increasing. And developers usually run unit tests to find these problems in the code. Besides, the developers are at the pain of getting stability of the code when they have to modify a code very often for clients requirements. In the methodlogy of TDD(Test Driven Development), developers write a unit test code first, and then write a program code for passing the unit test. The unit test must include the boundary condition test the reason why the possibility of occurring the bugs is very high. When failed to pass the test because of the value of a function is incorrect, not existed, out of the range or not matched etc, the program code will return the error code or occur the exception. In the document, the system is designed and implemented in order to insert the generated code automatically or suggest it to the developer, when the boundary condition test is failed. In conclusion, it is possible that the developer will get the code stability by searching the code and checking the code to be omitted automatically through this system.

Key Words : Test Driven Development, Code Generation

Ⅰ. 서론

사용자들은 컴퓨터와 네트워크의 발달과 스마트폰의 폭넓은 이용을 통해 소프트웨어의 중요성을 알게 되었다. 소프트웨어는 사용자들의 요구사항들이 점점 다양해지고 복잡해지면서 코드의 복잡성이 증가하고 있다. 소프트웨어가 점차 복잡해짐에 따라 소규모를 제외한 개인이 개발하는 프로젝트는 점차 사라지고 팀 또는 회사가 주체가 되어 개발 및 유지하는 프로

* 이 연구는 서울과학기술대학교 교내 학술연구비 지원으로 수행되었습니다.
** 현대통신(주) 책임연구원
*** 서울과학기술대학교 컴퓨터공학과 교수 (교신저자)

젝트가 대부분을 차지하게 되었다. 또한, 소프트웨어에 대한 사용자의 요구사항은 수시로 변경되지만, 소프트웨어의 복잡성은 변경된 요구사항을 반영한 수정된 코드의 안정성을 위협하고 있다. 이에 소프트웨어 대한 안정성, 유지보수성, 코드 수정 용이성을 유지하기 위해 TDD(Test Driven Development) 방법이 개발 및 활용되고 있다. TDD 방법론은 테스트 코드를 지속적으로 개발과 테스트를 진행함으로써 코드의 안정성을 확보하고 코드의 수정 후 테스트의 성공적인 결과를 항상 유지함으로써 코드 수정의 위험성을 줄이고 있다. TDD는 협업의 과정에 발생하는 공용으로 작성된 코드를 수정하는 경우 기존의 테스트 과정을 통과함으로써 협업 코드에 대한 안정성도 확보할 수 있다. TDD의 테스트 부분 중 경계조건에 대한 테스트는 반드시 포함되어야 한다. 왜냐하면 협업하는 과정에서 해당 함수들이 잘못 사용하거나 또는 잘못된 경로로 접근한 매개변수, 범위를 벗어난 매개변수 값 등은 실행중인 소프트웨어에 대해 치명적인 문제를 발생하기 때문이다. 경계조건은 사용자의 요구에 의거 수시로 변경될 수 있다. 예를 들어, 회계 프로그램의 경우 소득세율이 달라지면 해당 경계 조건에 대한 테스트도 같이 달라져야 한다. 기존의 TDD는 이러한 경계 조건이 달라지면, 해당 테스트 코드를 변경하고, 다시 실제 프로그램 코드도 변경하여야 한다. 본 논문에서는 경계 조건에 대한 테스트 코드를 변경함으로써 실제 프로그램의 코드도 자동으로 변경하거나 후보 코드를 개발자에게 제시하는 시스템을 설계 및 구현하고자 한다. 이렇게 함으로써 개발자는 테스트 코드 변경에 집중함으로써 테스트 코드를 안정화 하고, 실제 프로그램 코드에 대한 경계 조건의 변경이 누락되지 않기 때문에 실제 프로그램에 대한 경계조건과 관련된 오동작을 없앨 수 있다. 따라서, 본 논문에서 2장은 TDD의 기본적인 내용을 기술하며, 3장에서는 본 논문에서 제안한 시스템에 대한 논하고, 4장에서 실제 구현 부분을 예제를 들어 설명하고, 5장에서는 결론 및 향후에 연구내용에 대해 기술하고 끝을 맺는다.

Ⅱ. 관련연구

2.1. TDD

테스트 주도 개발은 점진적으로 소프트웨어를 개발하는 기법이다. 실패하는 테스트를 먼저 작성하지 않으면 제품 코드를 작성하지 않는 방법이다. 테스트 코드는 작다. 테스트는 자동화 된다. 테스트 주도의 개발은 논리적으로 나타난다. 개발시 제품 코드부터 시작하여 테스트를 미루는 대신, TDD 실천가들은 코드가 어떤 식으로 동작하리라는 기대를 테스트의 형태로 먼저 표현한다. 테스트 실패를 확인한 다음 비로소 그들은 코드를 작성하여 작성해둔 테스트가 통과하게 만든다[1].

2.1.1. TDD의 이득

(1) 버그 감소

나중에 심각한 문제를 야기할 수 있는 크고 작은 논리적 오류들이 TDD를 진행하는 중에 금방 발견된다. 결함을 예방할 수 있다.

(2) 디버깅 시간 단축

버그가 줄어든다는 것은 그만큼 디버깅 시간도 줄어든다는 것을 의미한다.

(3) 부대효과(side effect) 결함 감소

테스트는 개발 중의 가정이나 제약 조건들을 수행하고 대표적인 쓰임새를 보여준다. 새로 작성하는 코드가 기존의 제약 조건이나 가정에 위배되면 테스트는 실패한다.

(4) 거짓말하지 않는 문서들

구조가 잘 잡힌 테스트는 그 자체로 실행가능하고 모호함이 없는 분명한 문서역할을 한다.

(5) 마음의 평온

철저히 테스트한 코드를 가지고 있다는 것과 포괄적인 회귀테스트 세트를 가지고 있다는 것은 자신감을 준다.

(6) 더 나은 설계

좋은 설계는 테스트하기에도 쉬운 설계다. 긴 함수, 서로 얽혀있는 코드들, 복잡한 조건식들은 모두 복잡하고 테스트하기 어려운 코드다. 코드를 수정하기 위해 테스트를 작성하려는데 쉽게 작성할 수 없다면 설계적 문제가 조기에 드러난 것이다.

(7) 진척 모니터

어디까지 동작하고 얼마나 완료했는지를 테스트가 정확히 추적한다. 일정 산정의 근거이자 훌륭한 '완료' 정의의 기준이 될 수 있다.

(8) 재미와 보상

TDD는 개발자들에게 즉각적인 만족감을 안겨준다. 코딩할 때마다 여러분은 기능을 완료하는 것과 더불어 그것이 제대로 동작한다는 것도 알게 된다[1].

2.2. 경계조건

경계 조건 (boundary condition)을 알아내는 것은 단위 테스트에서 가장 중요한 부분 중 하나이다. 대부분의 버그가 보통 '경계'에 서식하기 때문이다. 다음과 같은 조건들을 생각해 봐야 할 것이다.

- 완전 엉터리거나 일관성이 없는 입력값
  예를들면 "!*WSDAskdljlzjljfklds;a@"라는 파일 이름
- 잘못 형식화된 데이터, 예를 들면 최상위 도메인 이름이 없는 이메일 주소 (fred@foobar.)
- 아예 없거나 빠뜨린 값, 예를 들면, 0, 0.0, "", null
- 합리적인 예상치에서 한참 떨어진 값. 예를 들면, 10,000 살이라는 사람 나이.
- 중복된 값이 있으면 안되는 목록에서 중복된 값
- 순서가 매겨져야 하는 목록인데 순서대로 되어 있지 않은 경우, 또는 그 반대. 예를 들어 정렬 알고리즘에 이미 정렬된 목록을 넘겨주거나, 거꾸로 정렬된 목록을 넘겨줘 보라.
- 잘못된 순서로 생기거나 기대한 순서와 다르게 일어나는 일, 예를 들면 로그인 하기 전에 문서를 출력하려고 시도하는 것.

가능한 경계조건 항목은 테스트하려는 메서드에 아래 <표 1>과 같은 비슷한 조건이 있는지와 이 조건을 위반할 때 어떤 일이 일어날지 고려해야 한다[2].

<표 1> 경계 조건

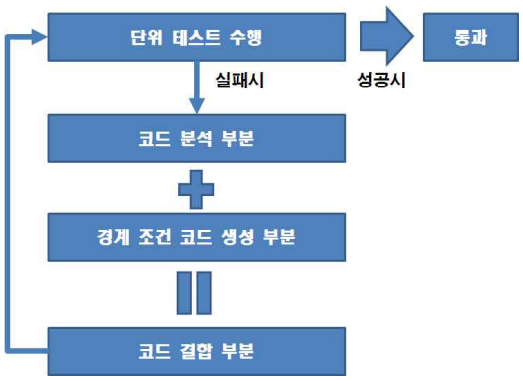
(1) 형식 일치 (Conformance) - 값의 형식이 기대한 형식과 일치하는가?
(2) 순서 (Ordering) - 적절히 순서대로 되어 있거나 그렇지 않은 값인가?
(3) 범위 (Range) - 적당한 최소값과 최대값 사이에 있는 값인가?
(4) 참조 (Reference) - 코드가 자기가 직접 제어하지 않는 외부 코드를 참조하는가?
(5) 존재성 (Existence) - 값이 존재하는가? (예 : null이 아님, 0이 아님, 집합안에 존재함 등)
(6) 개체 수 (Cardinality) - 확실히 충분한 값이 존재하는가?
(7) 시간 (Time - 모든 것이 순서대로 일어나는가? 제시간에? 때맞추어?

Ⅲ. 설 계

3.1 전체 시스템 구조

전체 시스템은 크게 3부분으로 구성되며, Unit Test 실패 이벤트를 받아 기존의 코드를 분석하는 부

분, 테스트한 조건과 관련해서 추가할 코드를 생성하는 부분, 기존의 코드와 생성된 코드를 합치는 부분으로 구성된다.

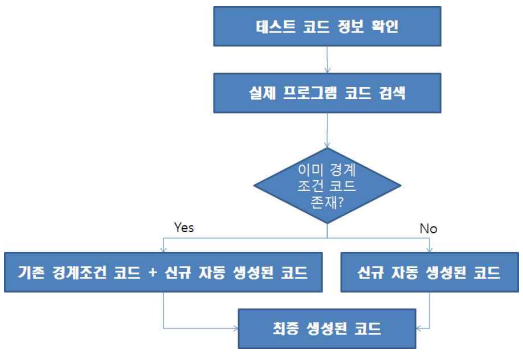


<그림 1> 전체 시스템 구조

3.1.1. 코드 분석 부분

코드 분석부분은 실제로 사용하는 코드를 분석하여 향후 단위테스트가 실패할 경우 생성된 코드가 삽입될 위치를 결정하기 위함이다.

3.1.2. 경계조건 코드 생성부분

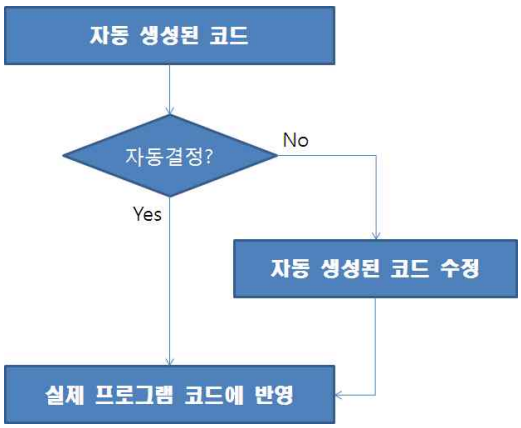


<그림 2> 경계 조건 코드 생성 부분

경계조건 코드 생성부분은 단위테스트가 실패할 경우 단위테스트 코드에 정의된 기본 정보를 이용해서 코드를 생성한다.

3.1.3. 코드 결합부분

코드 결합부분은 생성된 경계조건 코드와 기존 실제 사용 코드를 결합하는 부분이다. 생성된 코드는 자동으로 결합되거나 개발자에게 제시하여 직접 수정한 코드를 반영한다. 이렇게 결합된 코드는 단위테스트를 반드시 통과해야 한다.



<그림 3> 코드 결합부분

3.2. 동작 시나리오

본 논문에서 제안한 시스템의 동작은 아래와 같은 형태로 동작된다.

단위 테스트들 진행하는 과정에서 실패가 발생하면 테스트를 진행한 목적이 되는 코드를 파악한다. 또한 실패한 테스트에 대한 정보를 파악한다. 테스트를 진행한 목적이 되는 코드와 실패한 테스트 코드에 기입된 테스트 정보로부터 경계조건 코드를 생성한다. 생성된 코드는 자동으로 포함되거나 개발자에게 후보코드로 제안되며 최종 수정된 코드는 기존 코드

와 합쳐진다. 이렇게 합쳐진 코드를 이용해 동일한 테스트를 다시 시작한다.

<표 2> 동작 시나리오

```
단위 테스트를 수행
   만약 단위 테스트가 실패하면
      단위 테스트 되는 실제 코드를 검색
      단위 테스트 정보로 경계조건 코드를 생성
      실제 코드와 생성된 경계조건 코드를 자동으로 합치거나
      개발자에게 제시하여 수정한 코드 적용
      다시 단위 테스트 수행
      단위 테스트 결과에서 통과 여부를 확인
```

3.3. 경계조건 생성

경계조건코드는 테스트 코드에 포함된 정보를 기반으로 생성된 코드이다. 경계 조건 테스트는 상수의 경우, 최대값과 최소값에 대한 테스트, 참조 객체의 경우는 해당 값이 실제로 할당되었는지 여부를 확인하는 null값 테스트, 매개변수에서 넘어온 객체가 원하는 객체 타입인지 여부를 확인하는 타입테스트 등이 존재한다. 위와 같은 테스트에서 실패가 발생하는 경우 경계조건 코드는 생성된다. 경계조건 코드는 프로그램 동작에 예외를 발생하거나 에러코드를 되돌려 준다. 이를 통해 개발자는 매개변수의 잘못된 사용 위치를 파악하는데 용이하다. 만약 기존에 경계조건 코드를 포함하는 경우 해당 경계조건 코드에 새롭게 생성된 코드를 추가적으로 삽입한다.

Ⅳ. 구 현

4.1. 개발 환경

본 논문은 Java 언어, JUnit 4 를 이용하여 본 논문에서 제어한 시스템을 설계 및 구현하였다.

4.2. 테스트 코드

본 논문은 아래 코드를 이용하여 본 시스템 적용시 코드의 변화에 대해 예를 들어 설명한다. setData, getData 메소드는 정수형 변수에 대해 최대값과 최소값에 대한 경계조건 테스트에 사용되며, setString(), getString() 메소드는 null 값에 대한 테스트, setStringObject(), getStringObject()는 타입 경계 조건에 대한 테스트에서 사용한다.

4.2.1. 최대값과 최소값에 대한 경계 조건 테스트

값을 설정하는 함수의 경우 설정하는 값에 대해 경계 조건을 갖는 경우가 있을 수 있다. 예를 들어, 시험성적의 경우는 0~100점을 가질 수 있으며, 0 보다 작거나 100 보다 큰 경우는 모두 입력 오류이다. 이러한 메소드의 경우 설정 가능한 매개변수의 최소값과 최대값을 확인하고 범위에 포함되지 않는 경우 오류를 나타냄으로써 잘못된 매개변수로 발생할수 있는 프로그램 오류를 방지한다.

<표 3> 최대값과 최소값 사용 코드

```
public class MinMaxBoundaryCode {
   private int data;
   public int setData( int data ) {
      this.data = data;
      return 0;
   }
   public int getData( ) {
      return this.data;
   }
}
```

setMinMax에서 최대값과 최소값을 미리 설정함으로써 테스트가 실패하는 경우 실패 이벤트를 받아 자동으로 해당 코드를 변경하게 된다.

<표 4> 최대값과 최소값 테스트 코드

```
public class TestMinMax {
  private BoundaryFailedData data;
  @Rule
  public MethodRule watchMan = new TestWatchman() {
    @Override
    public void failed(Throwable e, FrameworkMethod
method) {
      BoundaryClassHandler handler =
BoundaryClassHandler.getHander();
      handler.setFailed(data);
    }
  };

  @Before
  public void setUp( ) {
    data = new BoundaryFailedData( );
    data.setClassName( "test.testset.MinMaxBoundaryCode" );
  }

  @Test
  public void testBoundaryData_Min_Max() {
    data.setMethodName( "setData", "data" );
    data.setMinMax( 1, 10 );
    data.setReturnValue("int", "-1" );

    MinMaxBoundaryCode testCode = new
MinMaxBoundaryCode( );

    assertEquals("Min Max", testCode.setData( 12 ), -1 );
  }
}
```

<표 5> null 값에 대한 사용 코드

```
public class NullBoundaryCode {
  private String strData = null;
  public int setString( String strData ) {
    this.strData = strData;
    return 0;
  }

  public String getString( ) {
    return this.strData;
  }
}
```

4.2.2. null 값에 대한 경계조건 테스트

메소드에서 사용하는 매개변수 값이 존재한다고 가정하고 프로그램 코드를 개발하는 경우가 많다. 하지만, 내부적인 오류 또는 외부적인 요인에 의해 해당 매개변수의 값이 null 인 경우 프로그램의 오류가 발생할 가능성이 높다. 따라서, 이러한 null 값에 대한 경계조건 테스트는 반드시 수행하여야 한다.

<표 6> null값에 대한 테스트 코드

```
public class TestNull {
  private BoundaryFailedData data;
  @Rule
  public MethodRule watchMan = new TestWatchman() {
    @Override
    public void failed(Throwable e, FrameworkMethod
method) {
      BoundaryClassHandler handler =
BoundaryClassHandler.getHander();
      handler.setFailed(data);
    }
  };

  @Before
  public void setUp( )
  {
    data = new BoundaryFailedData( );
    data.setClassName( "test.testset.NullBoundaryCode" );
  }

  @Test
  public void testBoundaryData_Not_Null( ) {
    data.setMethodName( "setString", "strData" );
    data.setNotNull();
    data.setReturnValue("int", "-1" );
    NullBoundaryCode testCode = new NullBoundaryCode(
);
    assertEquals("Not Null", testCode.setString( null ), -1 );
  }
}
```

4.2.3. 타입에 대한 경계조건 테스트

메소드의 매개변수를 객체 형태로 사용하는 경우 인터페이스 형태 또는 상위 클래스의 이름을 사용할 수 있다. 메소드 내에서 타입 변환을 통해 하위 클래스 또는 인터페이스를 구현한 클래스에서 필요한 기

<표 7> 타입에 대한 사용 코드

```
public class TypeBoundaryCode {
   private String objData;
   public int setStringObject( Object objData )
   {
      this.objData = ( String ) objData;
      return 0;
   }

   public Object getStringObject( ) {
      return ( Object )this.objData;
   }
}
```

<표 8> 타입에 대한 테스트 코드

```
public class TestType {
   private BoundaryFailedData data;
   @Rule
   public MethodRule watchMan = new TestWatchman() {
      @Override
      public void failed(Throwable e, FrameworkMethod
method) {
         BoundaryClassHandler handler =
BoundaryClassHandler.getHander();
         handler.setFailed(data);
      }
   };

   @Before
   public void setUp( )
   {
      data = new BoundaryFailedData( );
      data.setClassName( "test.testset.TypeBoundaryCode" );
   }

   @Test
   public void testBoundaryData_InstanceOf( ) {
      data.setMethodName( "setStringObject", "objData" );
      data.setTypeName( "String");
      data.setReturnValue("int", "-1" );
      TypeBoundaryCode testSet = new TypeBoundaryCode( );
      assertEquals( "Data Type", testSet.setStringObject( new
Integer( "1234" ) ), -1 );
   }
}
```

능을 사용할 때 형변환이 안되거나 잘못된 형변환으로 프로그램의 오류가 발생할 가능성이 있다. 따라서, 본 논문에서는 이러한 잘못된 타입의 객체를 사용할 경우 오류를 발생하고 올바른 타입의 객체의 경우만 허용하는 경계조건을 테스트 하였다.

4.3. 동작 설명

4.3.1. 최대값과 최소값 테스트 관련 동작 설명

1) JUnit을 이용하여 테스트 수행

2) 테스트 실패 이벤트 발생

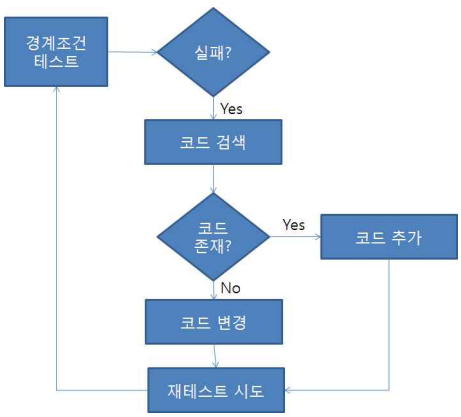
3) 실패 이벤트를 받으면 등록된 클래스와 메소드명, 최대값과 최소값에 대한 정보를 이용하여 코드 검색

4) 검색한 메소드 코드에 기존에 경계조건 코드가 존재하는 여부를 확인

5) 존재하는 경우 해당 코드에 추가로 최대값과 최소값에 대한 경계조건 코드를 삽입

6) 존재하지 않는 경우 조건문과 더불어 경계 조건 코드를 삽입

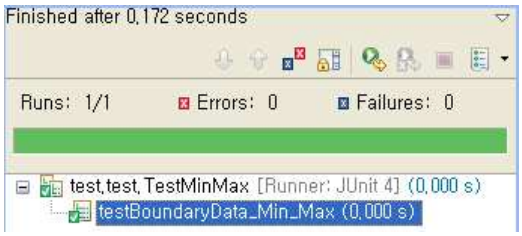
7) 삽입한 코드를 빌드한 후 1번 과정을 다시 수행하여 테스트 진행 및 통과



<그림 4> 최대값과 최소값 테스트 시나리오

<표 9> 최대값과 최소값에 대한 코드삽입 결과

```
public int setData( int data )
{
  if ( ( data >= 1 )  ||
    ( data <= 10 ) ) {
          return -1;
  }
  this.data = data;
  return 0;
}
```



<그림 6> 최대값과 최소값 테스트 결과

4.3.2. null 값 테스트

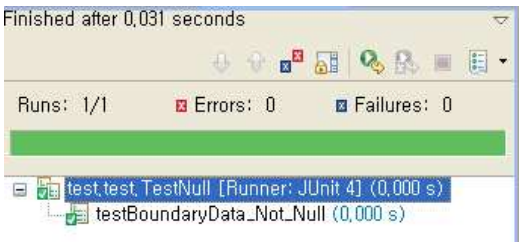
본 테스트에 대한 경계조건은 입력된 매개변수의 값이 null 값이 경우이다. 본 테스트를 통해 해당 객체가 초기화 또는 생성되지 않은 상태에서 접근함으로써 발생할 수 있는 문제를 미연에 방지한다. 따라서, 만약 null 인 경우 에러를 의미하는 값을 되돌려 준다.

4.3.3. 타입 변환 테스트

본 테스트는 매개변수의 객체의 형태가 올바른지를 검사를 진행한다. 본 테스트를 통해 다른 의미의 객체로 캐스팅함으로써 동작의 오류를 발생한다. 따라서, 캐스팅의 오류를 사전에 방지하기 위해 테스트를 진행하며 해당 타입인지 여부를 확인하는 코드를 삽입한다.

<표 10> null 테스트에 대한 코드 삽입 결과

```
public int setString( String data )
{
  if ( ( data == null ) ) {
    return -1;
  }
  this.strData = data;
  return 0;
}
```

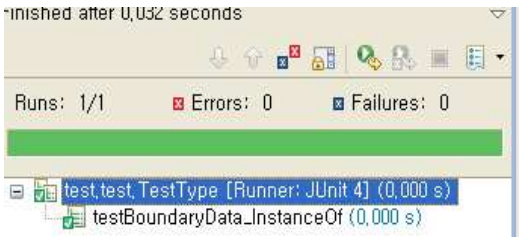


<그림 7> null값에 대한 테스트 결과

<표 11> 타입 변환 테스트에 대한 코드 삽입 결과

```
public int setStringObject( Object data )
{
  if ( ( !(data instanceof String)  ) ) {
    return -1;
  }

  this.objData = data;
  return 0;
}
```



<그림 8> 객체 타입에 대한 테스트 결과

4.4. 비 교

본 논문에서 제안한 시스템은 아래 <표 12> 와 같이 기존 개발 형태에 비해 분석 범위와 코드 수정에

대한 절차가 간소화한다. 테스트 코드가 미리 실제 코드 정보를 가지고 있으며, 테스트 실패시 실제로 연동되는 코드는 코드 분석 부분을 통해 자동으로 분석되기 때문에 개발자는 해당 테스트가 적용 되는 코드를 일일이 검색할 필요가 없이 테스트 실패가 발생하는 코드를 확인할 수 있다. 또한, 개발자는 테스트와 연관된 코드를 직접 수정하는 것이 아니라 자동 수정되거나 후보 코드를 제안 받아 수정함으로써 다른 코드들의 연관 관계 또는 코드 수정에 따른 부작용 등에 대해 고려하지 않아도 된다.

<표 12> 기존 개발 형태와 비교

항 목	기존 개발 형태	제안한 개발 형태
실제 코드 위치 탐색	개발자	시스템
코드수정주체	개발자	개발자 또는 시스템
코드수정절차	개발자가 실제 코드 확인 후 수정	시스템이 수정할 코드 제안 또는 자동 수정
테스트 오류시 분석 범위	테스트 분석 후 실제 코드 분석	실제 오류에 한정한 코드 분석

Ⅳ. 결 론

인터넷의 발달과 스마트폰의 폭넓은 보급은 많은 소프트웨어를 양산하는 계기를 마련하였다. 사용자들의 다양한 요구를 만족하기 위해 소프트웨어는 다양한 형태로 만들어 지고, 점차 복잡한 형태로 발전해 나가고 있다. 복잡한 소프트웨어 개발이 늘어남에 따라 소프트웨어의 안정성이 중요해졌다. 사용자들의 다양한 욕구에 대해 소프트웨어의 즉각적인 반영과 더불어 개발된 소프트웨어의 안정성을 동시에 만족하기 위해 TDD 방법론을 활용하는 사례가 늘고 있다. 소프트웨어 개발 중 상수값이 변경되거나 함수 형태가 변경되어 기존의 경계조건 테스트 수정과 더불어 코드 수정이 필요하게 되었다. 테스트 코드와 프로그램 코드 수정과정은 코드 누락, 테스트 코드와 프로그램 코드의 불일치 등의 문제가 발생한다. 따라서, 본 논문에서 이러한 문제점을 극복하기 위해 경계 조건 테스트 코드에서 경계 조건을 변경하면 프로그램 코드를 자동으로 변경하거나 후보 코드를 제시함으로써 아래 와 같은 장점을 갖는 시스템을 설계 및 구현 하였다.

- 테스트 코드와 프로그램 코드 간의 경계 조건 일치
- 경계조건에 대한 프로그램 코드의 자동 수정으로 인해 빠른 테스트 진행
- 경계 조건 변동으로 인한 코드 수정 과정에 발생할 수 있는 프로그램 오류 최소화
- 수정 코드에 대해 삽입할 코드 직접 수정을 통한 프로그램 코드 확장성 제공
- 코드 수정에 대한 후보를 제시하고 필요시 직접 수정할 수 있도록 함으로써 프로그램 개발 속도 향상

향후에는 머신러닝을 통한 프로그램 코드의 예상 코드 적용의 자동화에 대해 연구할 예정이다.

참고문헌

[1] 제임스 W 그레닝, 임베디드 C를 위한 TDD, 인사이트, 2012, pp. 5~7.

[2] 데이비드 토머스・앤드류 헌트, 실용주의 프로그래머를 위한 단위 테스트 with Junit, 인사이트, 2004, pp. 63~69.

[3] 박영조, 방혜자, “HTML을 위한 데이터베이스기반 스크립트 언어의 코드라이브러리 설계와 구현,” 디지털산업정보학회, 디지털산업정보학회논문지, 제9권, 제4호, 2013.

[4] 최종명, 박 경우, 오수열, “사용자 인터랙션을 지원하는 HTML5 기반 e-book 뷰어 시스템의 요구사항 분석 및 설계,” 디지털산업정보학회, 디지털산업정보학회논문지, 제9권, 제2호, 2013.
[5] 최종명, 최재영, 유재우, “프로그래머를 위한 Java2,” 홍릉과학출판사, 2004.
[6] 유석운 외 8명, “NHN은 이렇게 한다! 소프트웨어 품질관리,” 위키북스, 2010.
[7] 위드 커닝햄, 릭 머그리지 “Fit, 통합테스트 프레임워크,” 인사이트, 2010.

■ 저자소개 ■

박 영 조
Park Youngjo

2010년 5월~현재 현대통신(주) 책임연구원
2005년 2월 숭실대학교 컴퓨터학과(공학석사)
2001년 8월 서울과학기술대학교 컴퓨터공학과 (공학사)

관심분야 : 컴파일러, 프로그래밍 언어, embedded system, OS
E-mail : hanjava@nate.com

방 혜 자
Bang Hyeja

1985년~현재 서울과학기술대학교 컴퓨터공학과 교수
1993년 숭실대학교 컴퓨터공학과 (공학박사)
1983년 5월 Univ. of North Texas 전자계산학과 (이학 석사)
1977년 2월 숭실대학교 컴퓨터공학과 (공학사)

관심분야 : 컴파일러, 프로그래밍언어, 형식언어론, XML
E-mail : hjbang@seoultech.ac.kr

논문접수일: 2015년 5월 7일
수 정 일: 2015년 5월 20일
게재확정일: 2015년 5월 27일