Test-Driven Development :By Example

22장 - 실패 처리하기 &
23장 - 얼마나 달콤한지 &
24장 - xUnit 회고 &
25장 - 테스트 주도 개발 패턴

Date
2023.09.12

Presenter
황유란

이전까지 상황

- 실패하는 테스트 케이스를 만들어놓고 예외처리는 하기 전
- 실패하는 테스트를 좀 더 세밀한 단위의 테스트를 작성해서 올바른 결과를 출력하는 걸 확인하자

WasRun

```
    def testBrokenMethod(self):
        raise Exception
```

TestCaseTest

```
def testFailedResult(self):
    test = WasRun("testBrokenMethod")
    result = test.run()
    assert("1 run, 1 failed" == result.summary())
```

테스트 실패 문자열

- test 실패 시 문자열이 제대로 나오는지 확인하자
- 횟수는 맞다면 제대로 출력할 수 있지만 일단 신경 쓰지 않는다.

```
def testFailedResultFormatting(self):
    result = TestResult()
    result.testStarted()
    result.Failed()
    assert("1 run, 1 failed" == result.summary())
```

```
class TestResult:
    def __init__(self):
        self.runCount = 0
        self.failureCount = 0
    def testStarted(self):
        self.runCount = self.runCount + 1
    def testFailed(self):
        self.failureCount = self.failureCount + 1
    def summary(self):
        return "%d run, %d failed" % (self.runCount, self.failureCount)
```

testFailed() 호출

- **언제 호출해야 할까 -> 예외를 잡았을 때**
- self.setUp()에서 문제가 발생한 경우는 예외가 잡히지 않을 것
- 하지만 코드를 수정하기 위해서는 또 다른 테스트가 필요 -> 연습문제..

```
def run(self):
    result = TestResult()
    result.testStarted()
    self.setUp()
    try:
        method = getattr(self, self.name)
        method()
    except:
        result.testFailed()
    self.tearDown()
    return result
```

검토

- 작은 스케일의 테스트가 통과하게 만듦 (formatting 메서드)
- 큰 스케일의 테스트를 다시 도입 (testFailed)
- 작은 스케일의 테스트에서 보았던 메커니즘을 이용하여 큰 스케일의 테스트를 빠르게 통과 (횟수가 맞게 출력될 거라고 생각하고 넘어가기)
- 중요한 문제를 발견했는데 이를 바로 처리하기보다는 할일 목록에 적어놓음 (self.setUp에서 예외가 발생한다면?)

중복 코드

- 파일 마지막에 **테스트를 호출하는 코드가 중복**이 많다.
- 테스트를 한 번에 모아서 실행할 수 있는 기능을 원함
- TestSuite를 구현하자

```
print TestCaseTest("testTemplateMethod").run().summary()
print TestCaseTest("testResult").run().summary()
print TestCaseTest("testFailedResultFormatting").run().summary()
print TestCaseTest("testFailedResult").run().summary()
```

TestSuite

```
class TestSuite:
    def __init__(self):
        self.tests = []
    def add(self, test):
        self.tests.append(test)
    def run(self):
        result = TestResult()
        for test in self.tests:
            test.run(result)
        return testResult
```

```
class TestCaseTest(TestCase):
    def testSuite(self):
        suite = TestSuite()
        suite.add(WasRun("testMethod"))
        suite.add(WasRun("testBrokenMethod"))
        result = suite.run()
        assert("2 run, 1 failed" == result.summary())
```

run 인터페이스

- test.run()에서 TestCase.run() 메서드에 매개변수가 추가되면 여기도 매개 변수를 추가해줘야 한다.
- 파이썬의 기본 매개 변수 기능을 사용
  - 기본값은 컴파일타임에 평가되므로 TestResult 재사용 불가
- 메서드를 두 부분으로 나눈다.
  - TestResult를 할당하는 부분과 TestResult를 가지고 테스트를 수행하는 부분
  - 두 부분에 대한 좋은 이름이 떠오르지 않음 → 별로 좋은 전략이 아니다
- **호출하는 곳에서 TestResults를 할당한다.**
  - **매개 변수 수집 패턴이라 부른다.**

중복 제거하기

TestCaseTest

```
class TestCaseTest(TestCase):
    def testSuite(self):
        suite = TestSuite()
        suite.add(WasRun("testMethod"))
        suite.add(WasRun("testBrokenMethod"))
        result = TestResult()
        suite.run(result)
        assert("2 run, 1 failed" == result.summary())
```

suite.run에 만들어 놓은 TestResult를 전달한다.

TestSuite

```
    def run(self, result):
        for test in self.tests:
            test.run(result)
```

반환값이 없어짐

TestCase

```
def run(self):
    result.testStarted()
    self.setUp()
    try:
        method = getattr(self, self.name)
        method()
    except:
        result.testFailed()
    self.tearDown()
    return result
```

첫번째 줄에 result = TestResult() 부분을 지움

테스트 호출 코드 정리

```
test = TestSuite()
suite.add(TestCaseTest("testTemplateMethod"))
suite.add(TestCaseTest("testResult"))
suite.add(TestCaseTest("testFailedResultFormatting"))
suite.add(TestCaseTest("testFailedResult"))
suite.add(TestCaseTest("testSuite"))
result = TestResult()
suite.run(result)
print result.summary()
```

실패하는 테스트 고치기

- test.run() 메서드에 result 매개변수가 없어 실패하는 메서드를 통과하게 수정
- result = TestResult()가 중복되고 있다 -> setUp()에서 생성하게 만들기

```
def testTemplateMethod(self):
    test = WasRun("testMethod")
    result = TestResult()
    test.run(result)
    assert("setUp testMethod tearDown " == test.log)
def testResult(self):
    test = WasRun("testMethod")
    result = TestResult()
    test.run(result)
    assert("1 run, 0 failed" == result.summary())
def testFailedResult(self):
    test = WasRun("testBrokenMethod")
    result = TestResult()
    test.run(result)
    assert("1 run, 1 failed" == result.summary())
```

```
class TestCaseTest(TestCase):
    def setUp(self):
        self.result = TestResult()
    def testTemplateMethod(self):
        test = WasRun("testMethod")
        test.run(self.result)
        assert("setUp testMethod tearDown " == test.log)
    def testResult(self):
        test = WasRun("testMethod")
        test.run(self.result)
        assert("1 run, 0 failed" == result.summary())
    def testFailedResult(self):
        test = WasRun("testBrokenMethod")
        test.run(self.result)
        assert("1 run, 1 failed" == result.summary())
    def testFailedResultFormatting(self):
        result = TestResult()
        result.testStarted()
        result.testFailed()
        assert("1 run, 1 failed" == result.summary())
    def testSuite(self):
        suite = TestSuite()
        suite.add(WasRun("testMethod"))
        suite.add(WasRun("testBrokenMethod"))
        suite.run(self.result)
        assert("2 run, 1 failed" == result.summary())
```

검토

- TestSuite를 위한 테스트 작성
- 테스트를 통과시키지 못한 채 일부분만 구현
- run 메서드의 인터페이스 변경
- 공통된 셋업 코드를 분리

xUnit를 직접 구현해볼 만한 두가지 이유

1. **숙달**
    - xUnit의 정신은 간결함에 있다.
    - 마틴 파울러
    - '소프트웨어 공학 역사에서 이토록 많은 사람이 이렇게 짧은 코드로 이토록 큰 은혜를 입은 적이 없었다.'
2. **탐험**
    - 테스트를 여덟 개에서 열 개 정도 통과하게 만들다보면 새로운 프로그래밍 언어로 프로그래밍하면 접하게 될 많은 기능을 한 번씩 경험해보게 된다.

어떻게 테스트할 것인가

- 테스트한다는 것은 무엇을 뜻하는가?
- 테스트를 언제 해야 하는가?
- 테스트할 로직을 어떻게 고를 것인가?
- 테스트할 데이터를 어떻게 고를 것인가?

테스트 (명사)

- 아무리 작은 변화라도 테스트하지 않고 릴리즈하지 않는다.
- 제랄드 와인버그의 Quality Software Management의 스타일의 영향도
  - 양성 피드백 고리
- 스트레스를 많이 받으면 테스트를 점점 더 뜸하게 한다. -> 테스트를 뜸하게 하면 에러는 점점 많아질 것이다.

  -> 에러가 많아지면 더 많은 스트레스를 받게 된다.
- 이 고리를 빠져나오는 법
  - **자동화된 테스트**
- 스트레스를 받기 시작할 때 테스트를 시작할 것
  - 테스트함으로써 좋은 느낌을 받고 에러가 낼 일이 줄어들며 스트레스도 적어진다.

격리된 테스트

- 테스트를 실행하는 것은 **서로 아무 영향이 없어야 한다**.
- 교훈
  - **테스트가 충분히 빨라**서 직접 자주, 실행할 수 있게끔 하자
  - 앞 부분에서 실행된 테스트가 실패한 후 그 영향으로 다음 테스트부터는 시스템이 예측 불가능한 경우가 허다하다.
  - 테스트는 **작은 스케일로** 하는 게 좋다.
  - 각각의 테스트는 **다른 테스트와 완전히 독립적**이어야 한다.

테스트 목록

- **시작하기 전에 작성해야 할 테스트 목록을 모두 적어둘 것**
  - 구현해야 할 것들에 대한 테스트를 작성
- 테스트를 전부 만들어 놓는 방법이 별로인 이유
  - 매개 변수의 순서를 바꿔야 하는 경우 리팩토링하지 않으려 할 수도
  - 초록 막대를 빨리 보는 방법은 실패하는 테스트들을 다 지워버리는 것
- **테스트를 통과하게 만드는 과정에서 새로운 테스트가 필요함**을 알게 된다.

테스트 우선

- 테스트는 테스트 **대상이 되는 코드를 작성하기 직전에** 작성하는 것이 좋다.
- 테스트를 먼저하면 스트레스가 줄고 테스트를 더 많이 하게된다.

단언 (assert) 우선

- **단언을 제일 먼저** 쓰고 시작하라
- **작업을 단순**하게 만드는 효과
- assert를 위해서 다른 작업들을 채워넣음

```
testCompleteTransaction(){
  Server writer = Server(defaultPort(), "abc");
  Socket reader = Socket("localhost", defaultPort());
  Buffer reply = reader.contents();
  assertTrue(reader.isClosed());
  assertEquals("abc", reply.contents());
}
```

테스트 데이터

- 테스트를 읽을 때 **쉽고 따라가기 좋을 만한 데이터를 사용**하라
  - 1과 2의 개념적 차이가 없다면 1을 사용하라
  - 세 항목만으로 동일한 설계와 구현을 이끌어낼 수 있다면 굳이 열개나 할 필요는 없다.
- **실제 데이터를 사용**하는 경우도 있다.

명백한 데이터

- 데이터의 의도는 어떻게 표현할 것인가
- 테스트 자체에 **예상되는 값과 실제 값을 포함하고 이 둘 사이의 관계를 드러내기 위해** 노력하라