第4章 基礎知識から実践的なテスト手法まで

レッスン④…リファクタリングを支えるユニット・テスト

中林 智之

ここからは具体的な開発スキルを説明します．

リファクタリングと切っても切れないユニット・テストを知るところから始めます．

ここでは第1部 第1章で使ったリファクタリング後のサンプル・コードを題材にユニット・テストを書いていきます．ユニット・テスト・フレームワークはGoogleTestを使います．

4-1 ユニット・テストとは

技1 ユニット・テストでコードをテストする

ユニット・テストとはコードの小さな部品（関数やメソッド）に対して，動作を検証することです．

テスト対象のコードが意図した通りに動作するかどうかを確認するためには，ユニット・テストのためのコードを記述します．

ユニット・テストはリファクタリングを進める上で不可欠な存在です．リファクタリングは外部から見た振る舞いを変えずに，内部構造を改善する活動ですが，変更時に意図せず動作が壊れるリスクがあります．どんなに慎重に小さなリファクタリングをしても，思わぬバグが紛れ込むことがあります．

しかし，ユニット・テストを事前に書いておけば，振る舞いに変化がないことを自動で確認できます．人間が手動で毎回テストするのは非効率です（そして何より退屈です）が，ユニット・テストを活用すれば繰り返し・高速に・自動でテストを実行できます．

ユニット・テストはリファクタリングの安全な足場となり，より大胆なコード改善を可能にします．

リスト1 シンプルな関数のユニット・テストの例

```
// ①テスト対象のコード
int32_t add(int32_t a, int32_t b)
{
    return a + b;
}

// ②add()関数をテストするユニット・テスト
TEST(SampleTest, Add)
{
    EXPECT_EQ(3, add(1, 2));
}
```

リスト2 リスト1のユニット・テストの実行結果

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from SampleTest
[ RUN      ] SampleTest.Add
[       OK ] SampleTest.Add (0 ms)
[----------] 1 test from SampleTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (
                                          0 ms total)
[  PASSED  ] 1 test.  ← テストを通った
```

● シンプルなユニット・テストを見てみる

リスト1の①のadd()関数がテスト対象のコードです．add()関数をテストするユニット・テストのコードは**リスト1**の②のようになります．

テスト対象のコードとユニット・テストのコードとをビルドして実行すると**リスト2**のようにテスト結果が出力されます．

このようにコードに期待する振る舞いを，テスト・コードとして書いたものを本稿ではユニット・テスト

表1 本稿で使うテスト用語

用　語	意　味
コード	テストの対象となるコード．テスト対象コードや，プロダクト・コードとも呼ばれる
テスト・コード	コードに期待する振る舞いをコードとして記述したもの
ユニット・テスト	テスト・コードそのもの，または，テスト・コードでコードをテストする行為