サンプルで理解する単体テストのコツ

実践的な単体テストの実装方法

>>>中垣 健志 NAKAGAKI, Kenji 株式会社CSKシステムズ IT生産技術部

単体テストの実際

アプリケーションの開発者は、「短い開発期間」「不完全な仕様書」といった、とても不安定な環境の中で実装しなければならないことが多々あります。そのような環境の中で自分自身を守るためにも単体テストを用意しておくことが重要です。

しかし、まわりを見回しても実際に単体テストを実施しながら開発を進めている.NETのプロジェクトは聞いたことがない、という方も多いかもしれません。いったいなぜ単体テストは、なかなか開発現場に広まらないのでしょうか？

単体テストを実施することによる効果は、書籍や技術文書など、さまざまな場所で説明されています。そのため、自分のプロジェクトで単体テストを導入できないか検討したことのある方もいるでしょう。しかし、多くの開発者は、説明されているような効果的な単体テストを作成することができません。ひとつのクラスに数個のメソッドがあるようなクラスなら単体テストを書くのは簡単ですが、実際にIISやデータベースを操作するような複雑なアプリケーションに対しては、どのように単体テストを書けばよいかわからないからです。

実は単体テストを作成するには、コツがあります。それは、

単体テストがしやすいコードを書く

ということです。単体テストが作成できないとあきらめてしまうコードのほとんどは、「本当に」単体テストが作成できないコードになっています。このようなコードは、リファクタリングす

レベル >>> Level

1 2 3 4 5

言語 >>> Language

- Visual Basic

サンプル >>> Sample

この記事で取り上げたソースコードおよびサンプルプログラムは、
http://www.shoeisha.com/mag/windev/
からダウンロード可能です。

ツール >>> Tool

- Visual Studio 2005 Professional
- SQL Server 2005
- NCover v1.5.4 Beta
- TestDriven.NET-2.0.1761 RC1

※記事内ではVisual Basicでコードを紹介していますが、サンプルプログラムはVisual Basic版、C#版の2種類を用意しています。

ることで単体テストがしやすいコードへと変換することが可能です（リファクタリングとは、挙動を変えないままコードの見通しを良くする改良行為のこと）。

そこで本稿では、まず最初に単体テストがしやすいコードとはどのようなものなのかを説明してから、実際の開発現場でよく作成されるコードを、単体テストがしやすいコードへリファクタリングする方法について解説していきます。

単体テストがしやすいコード

先述のとおり、アプリケーションを開発するときに覚えておきたいのは、「単体テストがしやすいコードとしにくいコードがある」ということです。単体テストがしやすいコードは、意識しなければ書くことができません。

そこで、ここでは単体テストがしやすいコードを書くための3つのポイントを紹介します。

ポイント1 ひとつのメソッドに多くの処理を実装しない

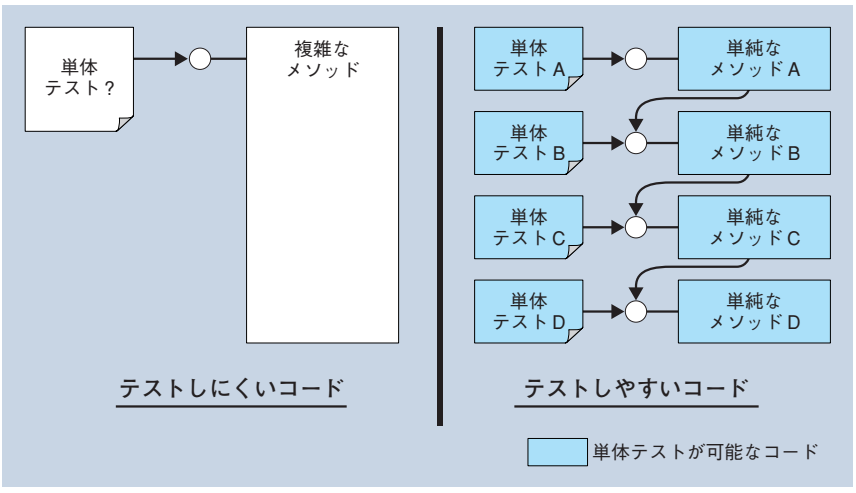
NUnitによる単体テストの対象は、メソッドです。あるメソッドを呼び出したときに行なわれる処理についてテストケースを作成し、その結果を検証することで単体テストを組み立てていきます。しかしメソッドが複雑になると、どのようなテストケースを用意すればすべての処理を検証したといえるのかが決めにくくなり、テストケースの作成が困難になります。

たとえば、「Mainメソッドしかないが、メソッドの中で複雑な条件分岐に従ってデータベース操作の処理が行なわれるアプリケーション」だったら？　Mainメソッドの返り値を確認しても、正しく動いているかどうかを判断することは難しいでしょう。

このような場合、複雑なメソッドを、複数の単純なメソッドに分割し、それぞれのメソッドに対して単体テストを作成するようにすれば、よりきめ細かにテストケースを用意することができます。

つまり複雑なメソッドをひとつ作成するより、単純なメソッドをいくつか作成したほうが単体テストが作成しやすいコードになるのです（図1）。

図1：メソッドの分割



ポイント2 メソッド同士の依存関係をなくす

先ほどの「ポイント1」に従って、図1のように複雑なメソッドを複数の単純なメソッドに分割できたとします。ここで図1の単体テストAと単体テストDについて、どのようなテストケースを作成すればよいか検討してみます。

単体テストDのテスト対象となるメソッドDは、メソッド内から別のメソッドを呼び出していません。したがって単体テストDでは、メソッドDに実装されているコードに対してのみテストケースを作成すればよいことになります。

同様に単体テストAでも、メソッドAに実装されているコードに対してのみテストケースを作成すればよいかというとそうはいきません。なぜならメソッドAは、内部でメソッドBを呼び出しているからです。さらにメソッドBはメソッドCを、メソッドCはメソッドDを呼び出しています。つまりメソッドAは、他のメソッドに強く依存してしまっているのです。このままでは単体テストAには、A～Dすべてのメソッドを考慮してテストケースを作成するという複雑なテストコードを書かなければなりません。

これを防ぐためには、メソッド間にある直接的な依存関係をなくす必要があります。幸いにもオブジェクト指向言語では、インターフェイスを使って2つのクラス間にある直接的な依存関係をなくすことができます。クラスBの