快速 脱!休日出勤 デバッグ技法

.NETアプリケーション障害解析の極意

第7回

[番外編] テスト項目から漏れやすい例外と例外シミュレートツール

株式会社NTTデータ
飯山 教史
IIYAMA, Takashi

テスト項目と例外

最近はテスト関連の書籍や記事も多く、テストの実施方法に関しても自動化ツールが充実してきている。それでもテスト項目を作成するときにどこまで考慮した項目を作ればよいのかがあいまいで、不安を感じながらテスト作業をしている開発者は多い。この記事は開発者向けなので、皆さんが関心があるのは主に単体テスト（UT）のテスト項目だ思われるが、書籍などでもC0やC1などテスト項目の立て方に関する情報は多く、それらを理解しておけば大丈夫だろう。

それでも私の経験上、あえて説明しておきたいのは「例外」である。それはなぜか。テスト項目から漏れやすい例外があるからだ。今回はまずこれらの例外についてテスト項目をどのように作るべきかを説明する。そして次にこれらの例外をコードの任意の場所で発生させることができるツールを紹介し、その利用方法を簡単に説明する。

今回の記事が皆さんのコードの安定に少しでも役立てば幸いである。

レベル >>> Level

1 2 3 4 5

ツール >>> Tool

- Visual Studio 2005 Professional
- DevPartner Fault Simulator

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。

考慮から漏れやすい例外

皆さんはUTのテスト項目を作るとき、どのような項目を作るだろうか。一番多いパターンとしては、

- 入力値チェック
- コードの網羅性
- 呼び出したメソッドの戻り値チェック

などではないだろうか。私はこれまで何度かテスト項目をレビューしたことがあるが、ほとんどのテスト項目がこれらの観点で作成されていた。

ところが、実際のシステムでは以下のようなシナリオも考えられる。

「購買システムで販売履歴をログファイルに保存しているが、ログの書き込みによりディスクの容量が不足し、書き込み時にエラーが発生した」

どのようなテスト項目を立てれば、このようなケースを予防できたか想像できるだろうか。おそらく無理だろう。このようなタイミングに依存する例外がテスト項目に挙げられていることはほとんどない。それはこのような問題が発生すること自体少なく、事前にこのようなタイプの例外があることを想定できないからだ。しかし発生する可

能性が0ではない以上、このような例外もテストしておかなければならない。そのためどこかのタイミングでテストを実行し、どのような結果になるのかを確認する必要がある。

このようにテスト項目で漏れやすいのは以下の例外である。

・実環境で発生することもあるが簡単には発生しない例外
・発生させることが困難な例外

このタイプの例外には以下のようなものがある。

・ある検索が失敗する
例：DLLが見つからない。ネットワーク越しにマシンが見つからない
・容量不足
例：メモリ利用量が足りない。ディスク容量が足りない
・破壊されている
例：レジストリが破壊されている

もちろんこの他にもレアケースの例外はいくつもあるが、これらの例外の発生も考慮したテストを行なわないと皆さんのコードの堅牢性は損なわれてしまうのである。

どんな例外をテスト項目にするか

しかしこれらレアケースの例外を考慮しても、すべての例外をテストしようとすると時間がいくらあっても足りない。そこでこれらレアケースの例外のいくつかをピックアップしてテストを実行することになる。ではどのような例外をテスト対象とすればよいのだろうか。たとえばリスト1のコードを見てみよう。

リスト1：入力結果をテキストファイルへ出力するコード

```
Private Sub btnWrte_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnWrte.Click
  Dim Writer As New StreamWriter("C:¥VBTest.txt")

  Try
    Writer.WriteLine(txtInput.Text)

  Catch ObjectDisposedException As Exception

    ' 何か記述

  Catch IOException As Exception

    ' 何か記述

  Finally
    Writer.Close()
  End Try
End Sub
```

このような場合、「ObjectDisposedException」と「IOException」はCatchブロックで処理されるので、“十分予想されている”と考えてよい。予想されているということはその発生はテスト項目で記述される可能性が高い。

ではCatchブロックで検出されない例外はどうだろうか。Catchブロックで処理されずにFinally句のコードが実行され、このコードの呼び出し側へと例外が通知される。このとき、呼び出し側でどう処理されるかは不明なので、この例外が発生したときの影響度は“予想できない”ことがわかる。もしかしたら上位側で適切な後処理がなされているかもしれないが、そんなことはめったにない。つまり、“深刻な影響を与える可能性が高い”ということだ。

このことから、テスト対象としてケアすべきレアケースの例外は以下のように選べばよいことがわかる。

①自分の書いたCatchブロックで補足されていない
②Catchブロックで補足されている例外を継承している例外ではない[注1]
③Tryブロック中のコードがアクセスするリソースに関連する例外の中で、レアケースに分類できるもの

最後の③は主観になってしまうが、常識で考えればネットワークやハードディスクなど、ハードウェア系の例外を想定すれば問題ないだろう。そのときは必ずTryブロック内で利用するメソッドがアクセスするハードウェアに関連する例外にしなくてはならない。

注1）例外はExceptionを基本クラスとする階層を持っている。基本クラスをCatchブロックに記述すると、それを継承する例外クラスすべてを補足する。例外クラスの継承関係についての詳細は以下のURLを参照。
http://msdn2.microsoft.com/ja-jp/library/z4c5tckx.aspx