xUnit Test Patterns
Chapter 22
Fixture Teardown Patterns

2010/7/24

Garbage-Collected Teardown

Garbage-Collected Teardown

- 反復可能で、堅牢なテストはfixtureがテストごとにtear downされて次のテストで作成される ことによって実現される。→Fresh Fixture

- Garbage Collection(GC)を提供する言語では、リソースにローカル変数やインスタンス変数経由でアクセスすることによって実現される。

How it Works

- テストの過程ごとに、オブジェクトが生成されるが、大半はtransientなオブジェクトであり、生成したプログラムの(ライフサイクル)でのみ保持される。
- GCの仕組みを備える、モダンな言語では「ごみ」を検出するために、さまざまなアルゴリズムが使われている

- 重要なのは、なにが「ごみ」ではないかを判別する方法
- 他の生存しているオブジェクトやglobal(static)から参照する方法はGCによって回収されない。
- テストを動かす際に、 Test Automation Framework はTestcase Class 内のTest Methods ごとにTestcase Object を生成し、 Test suite Object 内に追加する。

- あたらしいテストが走るごとに、フレームワークが既存のsuiteを放り投げ、新しいsuiteを生成する。古いテストが破棄されるたびに、インスタンス変数としてのみ参照されるオブジェクトはガベージコレクションにまわされる。

When to Use it

- Garbage-Collected Teardown をつかえば、様々な労力から解放される。
- GCをサポートしない環境または、GCの対象でないリソース(ファイル、ソケット、DBのレコード)がある場合には、リソースを明示的に破棄または解放する必要がある。

- Shared Fixtureを使用する場合には、テストが走り終わって、fixtureが参照から外れるまでは、何らかの方法でfixtureへの参照が残っている限りはGarbage-Collected Teardownは使用できないことになる。

- 属性がリリースされることを保証するために、In-line Teardown (page 509), Implicit Teardown (page 516), またはAutomated Teardown (page 503) toを使用することができる。

Implementation Notes

- xUnit familyやIDEによっては、test sutieが走るたびにクラスが置き換わったりするものさえある。
- こういうふるまいは、「Reload Classes」オプションと呼ばれるか、強制的に実行される。

- fixture がクラス変数を持つときに、Garbage-Collected Teardownを行うために有効だが、IDEを変えたり、コマンドラインからテストを走らせたときにテストが通らなくなることに留意する必要があります。

Motivating Example

- 例にあげたテストでは、fixtureがメモリ上のオブジェクトして生成され、In-line Teardownによって明示的に破棄される。
- これらのオブジェクトが永続的でない場合は、コード上でproposedFlightを削除することは不要であり、 テストの理解を難しくする。

- Garbage-Collected Teardown に変更するには、不要なクリーンアップのコードを削除する必要がある。
- もしオブジェクトへの参照を保持するためにクラス変数を使用している場合は、インスタンス変数かローカル変数を使用するように変更することによって、Shared Fixture からFresh Fixture に変更する。