ソフトウェアテストシンポジウム 2003
JaSST'04: Japan Symposium on Software Testing 2003

テストパターンの有効性について
ーテストファーストとテストパターンによるソフトウェアテストー

小井土　亨†

†株式会社オーエスケイ　〒136-0071 東京都江東区亀戸 7-6-4

E-mail:　†koido@kk-osk.co.jp

あらまし　現在、効率的にソフトウェアを開発する手法のひとつである eXtreme Programming のプラクティスの中で、テストファースト（実装を行う前に、テストプログラムを作成する）の有効性が注目されている。テストファーストで、良いテストを作成するためには、テストに対する知識を必要とすることは当然であるが、ターゲットプログラム（製品となる部分のプログラム）の構造がテストを意識したものになっていることも重要である。しかし、テストを意識した良いプログラム構造を作成するためには知識や経験が必要となる。そこで、テストしやすいプログラム構造とテストの組合せをテストパターンとして紹介し、その有効性について解説する。

キーワード　アジャイル，eXtreme Programming，テストファースト，パターン

About the effectiveness of a test pattern
－Software Test by Test First and Test Pattern－

Toru Koido†

† OSK Co.,LTD. 7-6-4, Koutou-ku, Tokyo, 136-0071 Japan

E-mail:　†koido@kk-osk.co.jp

Abstract　In the practice of eXtreme Programming which is one of the techniques which develops software efficiently, the test first's effectiveness attracts attention. Although the knowledge over a test naturally needs in order to create a good test at test first, it is also important that the structure of a target program is what was conscious of the test. However, knowledge and experience are needed in order to create the good program structure which was conscious of the test. The combination of the program structure which is easy to test, and a test is introduced as a test pattern, and the effectiveness is explained.

Keyword　Agile，eXtreme Programming，Test First，Test Pattern

1. はじめに

近年、ソフトウェアはビジネスを成り立たせるために必要不可欠と位置づけられるものが数多く見られるようになった。例えば、Web 上のオークションサイトなどは、この典型的なものである。この様なソフトウェアは、動作停止がビジネスの停止を意味し、高い品質が求められる。また、この様なビジネスは、顧客のニーズをいち早く取り入れ、常に新しい機能を提供することがビジネスで勝ち残る必須条件である。したがって、常にシステムを変更させることが要求される。

この安定して稼動させることと、常に変更することは、相反する要件である。多くの Web アプリケーションなどは、この二つの側面を満足させるという、難しい課題を常に突きつけられることになる。

このような状況下で、システム開発を迅速に行うために、eXtreme Programming（以下 XP）などのアジャイル開発手法が注目されている。XP のプラクティスの中で、テストファーストや自動テストを利用したリファクタリング、自動テストを利用した常時結合などは、その有効性に注目が集っている。また、テストを開発の中心に置いたテスト駆動開発という方法論も注目を集めている。

このように、迅速でかつ継続的な変更が必要とされるソフトウェアの開発では、テストファーストとテストの重要性が高く認識されている。しかし、経験の浅い開発者が、良いテストを書くことは、決して簡単なことではない。そこで、良いテストとターゲットプログラムの組合せをテストパターンという形式で解説し、テストファーストとテスト作成に於けるヒントになることを希望し、本論文を記述する。

本論文では、2.でテストパターンについて明確にする。3.でテストパターンの基本方針について説明を行う。そして、4.で各テストパターンについて解説を行う。最後に、5.でテストパターンを応用した事例を紹

介する。

2. テストパターンとは

2.1. 良いテストの効果について

テストファーストで作成するテストの中で、良いテストと呼ばれるものは、以下の条件の多くを満たしている必要がある。

①テスト目的が明確である
　何をテストするか明確である。また、その目的も明確で、ひとつであること。

②テストの判定が正しい
　テストの成功、失敗が正しく判断されている。

③テストを独立して実行することができる
　テストが他のテストに依存することなく、独立している。

④繰り返し実行することができる
　何度でも繰り返して、テストを実行することができる。

⑤テストを実行しても、状態が変化しない
　テストを実行し、成功した場合でも失敗した場合でも、テストを実行する前と後で何も変わらない。

これらの条件を満たすテストによって、ターゲットプログラムは、必然的に以下の条件のいくつかを満足することになる。

①メソッドが一つの機能を実現している
　メソッドが明確な一つの機能だけを提供する。

②メソッドの結果を提供する
　外部に対して、メソッドの処理結果を判断できるような何らかの方法を提供する。

③クラスやメソッドの独立度が高い
　クラスやメソッドが、他のクラスやメソッドの依存が低い。

④特定の環境への依存度が低い
　特定のファイルやデータベース構造などに対する依存度が低い。

良いテストが与えるターゲットプログラムへの効果をまとめると、以下の２つとなる。

①シンプル
　テスト、クラス、メソッドなど、全てが明確でシンプルとなる。

②低い依存度
　テスト、クラス、メソッドなど、全てが他のテストやクラスやメソッドに対して、依存している部分が少なくなる。依存度が低いことで、高い独立性が実現される。

このように良いテストは、ソフトウェアに求められているシンプルさと低い依存性という、特性を与える具体的なプラクティスとなる。

2.2. テストパターンの範囲

本論文で扱うテストパターンは、テストだけに関するパターンではなく、テスト対象となるターゲットプログラムと自動テストの組合せについてのパターンを対象としている。これは、2.1 で述べたように、テストファーストに於ける、テストとターゲットプログラムには、密接な関係があり、相乗効果があることになる。

2.3. テストパターンの必要性

ウォーターフォール型開発では、プログラム実装を行い次にテストを行うという、作業手順であるため、テストパターンは必要とされていなかった。その理由は、実装とテストの間に時間などのさまざまなキャップが存在するからである。

テストファーストを採用した開発プロセスでは、少しテストを作成し、少し実装するといった繰り返し開発を行うことになる。このような作業手順では、ターゲットプログラムとテストが並列に近い状態で作業が進められる。したがって、両方の側面を意識したパターンが有効に作用する。

3. テストパターンの基本方針

各テストパターンについて、目的、解説、利用場面、制限事項、構造を記述する。テストパターンは、他のパターンと同様に、必ずしも利点ばかりとは限らない。パターンを採用することで、システムに対して、副作用的な影響が発生することがある。本論文では、この様な用件に対して、制限事項として記述している。

4. テストパターンについて

4.1. インフォメーションセンターパターン

4.1.1. 目的

①環境に依存するプログラムをテストする
　プログラムの動作がファイルやデータベースなどの環境に依存する場合がある。このようなケースで、効率的にテストを行う。

4.1.2. 解説

環境に依存するプログラムをテストする場合、テストを実行する前にテストのための環境を構築する必要がある。テストを実行する前に、テスト用の環境を動的に構築する方法では、テストの作成工数やテストの実行時間が掛かってしまう。

この様なときに、事前にテスト用に環境を構築し、実行時に環境を切り替えてテストを可能とする。ターゲットプログラムの環境に依存する設定部分をインフォメーション･センタークラスで一元管理する。また、一元管理した、環境情報を設定変更可能とし、テスト時には、様々なテスト環境を切り替えてテストを行う。

4.1.3. 利用場面

以下のような場合に、利用することができる。

①データベースの状態に依存するテストの実行

データベースの状態に依存するテストを行う場合に、状態を再現したデータベースに環境を変更してテストを実行する。

②特定のファイルなどに依存するテストの実行

ファイルの内容に依存するテストを行う場合に、様々な内容のファイルを用意し、対象を変更してテストを実行する。

4.1.4. 制限事項

このテストパターンには、以下のような制約事項がある。

①必要とされない設定変更機能が残ってしまう

このパターンで導入したインフォメーション・センターのクラスは、要求に必要ない機能として実装される。

4.1.5. 構造

①Target

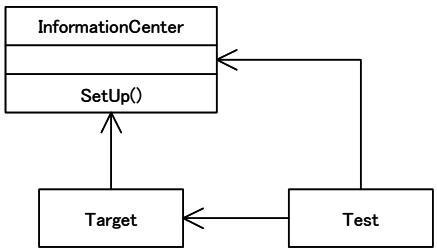
テスト対象となるクラス

②InformationCenter

設定管理クラス

③Test

テストクラス



4.2. ビューステートパターン

4.2.1. 目的

①ユーザーインターフェイスのロジックテスト

ユーザーインターフェイスのテストは、自動化することが難しい。ユーザーインターフェイスの状態やロジックを別クラスに分離して、テストする。

②ユーザーインターフェイスの状態変化テスト

ユーザーインターフェイスは、状態変化に応じて描画を変更する場合がある。この様な動作をテストすることは難しい。ユーザーインターフェイスの状態やロジック分離したクラスにイベントを追加することで、テストを行う。

4.2.2. 解説

ユーザーインターフェイスのテストは、一般に手作業で行うことが多い。これは、ユーザーインターフェイスの動作や変化を自動テストで確認することが難しいからである。

この様なときに、ユーザーインターフェイスの状態やロジックを別クラスに分離し、自動テストを行う。また、分離したクラスにイベントを追加することで、描画の更新タイミングなどのロジックに対してテストを可能とする。

4.2.3. 利用場面

以下のような場合に、利用することができる。

①ユーザーインターフェイスのテスト

ユーザーインターフェイスの状態やロジックを自動テストする。

②シナリオテスト

複数のユーザーインターフェイスに対応したビューステートクラスを作成し、様々なシナリオテストを行う。

4.2.4. 制限事項

このテストパターンには、以下のような制約事項がある。

①描画情報を二重持ちする場合がある

ユーザーインターフェイス自体ではなく、分離したクラスが状態を管理するために、情報の二重持ちが発生する場合がある。

②描画状態はテストできない

ユーザーインターフェイスで高度なコントロールを使用してグラフを表示する場合などでは、データなどはテストすることができるが、グラフ描画自体のテストを行うことはできない。

4.2.5. 構造と構成要素

①View

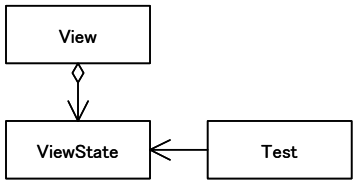
ユーザーインターフェイスのクラス

②ViewState

ユーザーインターフェイスの状態管理クラス

③Test

テストクラス



4.3. オーダーシートパターン

4.3.1. 目的

①クラス間の依存関係をなくしてテストを実行

クラス間に直接的な依存関係を無くし、クラスの独立性を高くして、単体テストを実行する。

②ユーザーインターフェイスと処理クラスの分離

ユーザーインターフェイス関連クラスと処理ク

ラスを分離し、個別に単体テストの実行を可能にする。

4.3.2. 解説

クラス間の依存関係が強い場合、単体テストを作成することが難しい。

この様なときに、処理実行に必要な情報をオーダーシートクラスに分離し、クラス間の関連を無くすことで、テストの作成を容易にする。また、複数のビューステートクラスに対して、オーダーシートクラスを渡して、処理実行情報を集めることで、処理を実行せずにシナリオテストを可能とする。また、ユーザーインターフェイスを通さずに、オーダーシートに様々な設定を行ったテストを可能にする。

4.3.3. 利用場面

以下のような場合に、利用することができる。

①処理のバリエーション実行テスト

オーダーシートクラスに様々な設定を行い、カバレッジの高いテストの実行を行う。

②シナリオテスト

複数のビューステートクラスにオーダーシートクラスを渡す方法で、シナリオ型のユーザーインターフェイステストを行う。

4.3.4. 制限事項

このテストパターンには、以下のような制約事項がある。

①ステートレス処理が対照となる

ユーザーインターフェイスと処理クラス間がステートレスであることが条件である。したがって、ユーザーインターフェイスと処理クラス間に密接な関係がある場合は適応できないことがある。

4.3.5. 構造と構成要素

①OrderSheet

オーダーシートのクラス

②IViewState

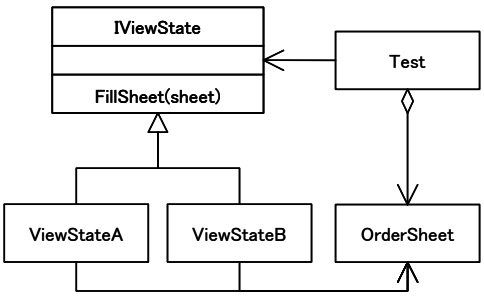
ユーザーインターフェイスのクラス

③ViewStateA , ViewStateB

ユーザーインターフェイスの状態管理クラス

④Test

テストクラス



5. テストパターンの適応事例

テストパターンを適応した、システム開発事例を紹介する。

5.1. 事例の内容と開発リスク

UNIX 系の OS で動作する FAX 機能を持つ複合機対応の組み込み業務システムの開発を行った。組み込みシステムであることによるリスクも含めて、以下のようなリスクを含んだ開発であった。

①ベータ版の開発環境

新規開発された開発環境であったため、開発当初は開発環境がベータ版であった。しがたって、仕様変更や品質の安定に関して高いリスクが存在した。

②テスト工数

開発対象が組み込みシステムであったため、実機でのテストは手動テストのみであった。したがって、テストに必要な工数が増大するリスクが存在した。

③仕様変更

開発関係者が全てにおいて初めて経験するタイプのシステムであった。したがって、何時どのような仕様変更が発生するか、予測することができないシステムであった。

以上のようなリスクを回避するために、システムのサブシステム間の依存度を低くすることと実機以外でテストできる部分の比率を高くすることを目的に、テストパターンを導入してシステムの開発を行った。また、仕様を明確にするために、順次動作するシステムを提供する形式の、繰り返し型の開発を行った。

5.2. テストパターン導入結果

事例のシステムは、品質の高いものとなった。具体的には、品質検査チームで２週間の最終製品検査を行ったところ、３点の問題しか発生しなかった。その内の２点は、操作方法についての改善要望であった。また、他の１点は、他の組み込みシステムとの相性に関する問題であった。このように、開発途中でも最終的な検査作業においても、簡単なミスによる不具合は非常に少なかった。しかも、問題点の発見コストと修正コストについても、低く抑えることができた。

結果としてこの事例において、テストパターンを導入し、テストファーストで良いテストを数多く作成することで、品質の高いシステムを開発することができた。今後、新たなテストパターンの発見と、テストパターン導入事例を増やしたいと考えている。

文　献

[1] ケント・ベッグ著，テスト駆動開発入門，ピュアソン・エデュケーション，東京，2003.