テストファースト開発におけるユニットテストに関する研究

2000MT031 伊藤宏平　　2000MT058 永田英人

指導教員 青山幹雄

1. はじめに

本研究では，JUnit[4]を用いたテストファースト開発における，GUI(Graphical User Interface)のユニットテストの効率化の方法を提案し，例題を用いて検証した．JUnit は，エクストリーム・プログラミング[3]をはじめとするテストファースト開発において，テスト支援ツールとして用いられている．しかし，JUnit によるユニットテストにおいて，GUI のテストが難しいという問題があった．そこで我々は，プログラムを MVC に分離し，Model 部→Control 部→View 部の順に実装する開発手順を提案した．そして，テストの柔軟性，テスト工数，実装コード量，実装手順の妥当性の観点から評価を行った．

2. GUI テストの効率化

2.1. GUI テストの問題点

JUnit をはじめとするテスティングフレームワークを用いたテストにより，ユニットテストは大幅な効率化が図られた．しかしその一方で，このようなテスティングフレームワークではテストが難しい項目が存在する[1]．その一つが，GUI のテストである．基本的に GUI のテストは，テストコードを使ってキー入力などのイベントを発生させ，結果がどのように表示されるかをテストする，という手順で行われる．しかし，多くの現場ではこれを手作業に頼っているのが実情である．無数に存在するユーザの GUI 操作手順をすべてテストすることは不可能に近い．近年では，WinRunner や VisualTest 等のテストツールや，Abbot をはじめとするテスト用スクリプト作成のフレームワークにより，テストの効率化が図られているが，そのようなツールやフレームワークを用いた場合でも，手作業で行っていた操作をスクリプトに記述し自動実行させたに過ぎず，すべてのユーザ操作を網羅するテスト用のスクリプトを作成する方法は，スクリプト編集には多くの時間を要すため，実用的でない．

2.2. MVC モデルを応用した開発方法の提案

2.2.1. プログラム設計の構造

GUI のテストにおける問題について，一つの解決方法として考えられるのが，プログラム中の計算など内部処理を行うロジック部と，視覚的に入出力などを操作するインタフェース部分を分けて設計する方法である．この考え方をさらに進め，我々は MVC(Model View Controller)モデルの考え方を取り入れたプログラミングを行うことにより，ユニットテストの効率化を図ることを考えた．

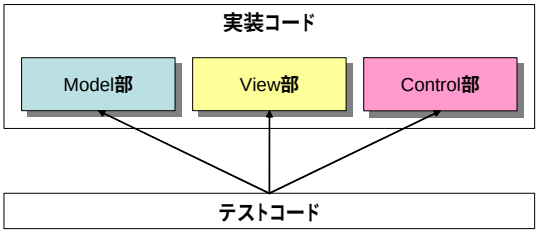

図 1　MVC 分離によるプログラム設計の構造

図 1 に示すように，プログラム全体を「Model 部」，「View 部」，「Control 部」に分けて設計し，それぞれに対してユニットテストを行うものとする．

2.2.2. MVC の定義

Model 部, View 部, Control 部の定義を表 1 に示す．

表 1　MVC の定義

名称	働き
Model 部	データと，それを計算する計算処理を行う部分.
View 部	GUI の表示やレイアウトを行う部分.
Control 部	イベント処理や，それによる Model 部，View 部の呼び出し，連携など.

この定義に従った，GUI アプリケーションにおける MVC 分離の例を図 2 に示す．

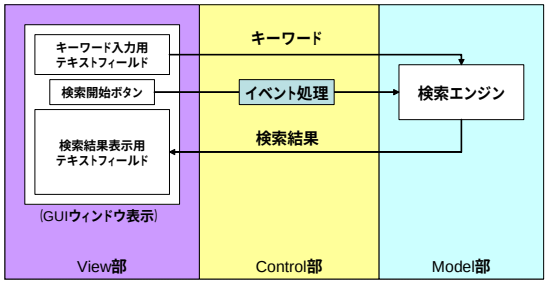

図 2　MVC 分離の例

アプリケーションは，キーワードを入力し，検索開始のボタンを押すことで検索エンジンにより処理を行い，その結果を表示するものとする．この場合 Model 部は，実際の検索処理を行う検索エンジンの部分であり，View 部は GUI の表示や，レイアウトを指定する部分，Control 部は，ボタンが押されたことによるイベント処理や，GUI から入力されたキーワードを読み取り，検索エンジンに受け渡す処理，また，検索エンジンが返した結果を，検索結果を表示させ GUI に受け渡す処理を行う部分である．

2.2.3. MVC 分離による開発手順

次に，MVC 分離による実装方法の手順を示す．はじめに Model 部から実装を行う．Model 部の実装が完了したら，次に Control 部の実装を行う．Control 部では，実際に GUI を生成し，それにイベント処理等を記述することで，Model 部での処理を関連付ける．ここでは GUI を生成するが，実際に GUI を画面上に配置する処理は，View 部で行う．最後に Main メソッドを記述することで，これらの処理を統合し，アプリケーションを完成させる．これを XP によるテストファースト開発の手順に従って進めるものとする．

2.2.4. 利点

MVC に分離した GUI アプリケーション開発において考えられる利点を以下に挙げる．

GUI のテストが難しいことは，GUI 表示と内部処理が結びついているために起こるものと考えられる．しかし，プログラムを MVC に分離することにより，ユニットテストが難しいとされる GUI における View 部分を分けて考えることができる．前節に示した開発手順により，プログラム内部が正しく動作するかどうかは，Model 部と Control 部を実装し終えた段階ですでに確認が可能である．View 部分における GUI のレイアウトや配置はその後に行えばよい．また，MVC に分離し，それぞれに対してユニットテストを行うことにより，どこでエラーが発生しているのかが明確になる．

これにより，GUI アプリケーション開発におけるユニットテストの適応性が増すとともに，エラーの明確化により，開発期間の短縮やエラーの減少による品質の向上，結果としてソフトウェア開発の効率化が図ることができると考えられる．

2.2.5. 問題点

一つのアプリケーションにおいて，優れたユーザインタフェースを設計しようとすると，GUI 部分のコードはかなりの比重を占める．

図 3 は，ある GUI を持つプログラム A，B の中の，Model 部，View 部，Control 部の割合を示したものである．A，B を比較した場合，プログラム A に MVC 分離を適用した場合，View 部の割合は比較的少ない．従って，テストの難しい View 部分の分離は，効率の面から有効であるといえる．しかし，プログラム B の場合，View 部分の比率が高く，ユーザインタフェースをロジックと分離できたとしても，残ったテストの分量は少ないとはいえない．従って，MVC の分離が必ずしも効率的になるとはいえない可能性がある．

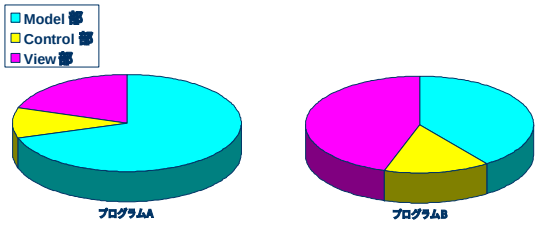

図 3　プログラムにおける GUI の分量

3. 評価

3.1. テストの柔軟性

MVC の分離により，プログラムの特定の箇所に対してテストを行うことが容易になった．

例として，我々が実装したプログラムにおける Model 部に対してのテストを挙げる．MVC に分離しない場合，Model 部，つまり入力された値を計算する部分に対してテストを行うと想定すると，計算部分だけをテストすることはできない．まず，イベントを発生させ，その結果返された値に対してテストする必要がある．一方，MVC に分離しない場合，Model 部をテストするためには，図 4 に示すように，Control 部のイベント処理を介す必要がある．

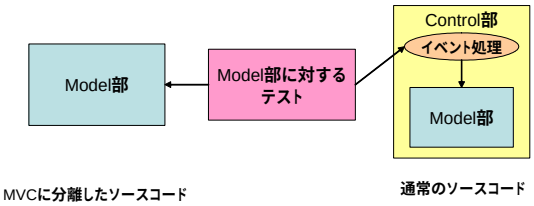

図 4　Model 部に対するテストの比較

その結果，エラーが発見されたとしても，それが計算部分で起こっているものなのか，イベント処理部分で起こっているものなのかが不明確になってしまう．それに対し MVC を分離した設計，実装では，それぞれが分離されているため，個々の部分に対してのテストに柔軟に対応することができ，エラー箇所の特定も明確になる．

3.2. テスト工数

次に，テスト工数の比較を行う．Model 部と Control 部に対するテストを例として図 5 に示す．

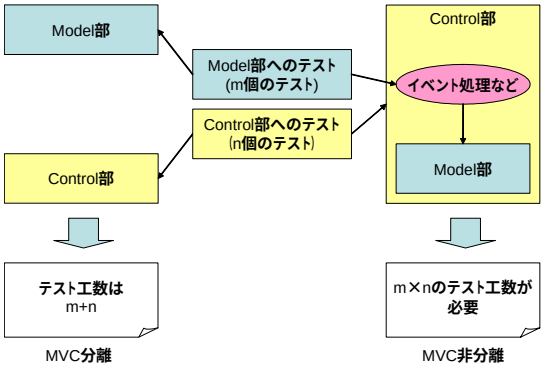

図5　テスト工数の比較

ここでは, Model 部に対して m 個のテスト, Control 部に対して n 個のテストが必要であると仮定する. このとき, MVC に分離した場合のテスト工数は, それぞれ Model 部に対して m 通り, Control 部に対して n 通りのテストを行えばよいため, テストの総数は m+n 個でよい.

それに対して, MVC 非分離の場合, Model 部に対するテストは, 前節にも示したように, Control 部を介す必要があるため, 全体では m×n 通りのテストが必要となってしまう. つまり, MVC を分離することにより, テスト工数を減少させることができる.

3.3. 実装コードの量

MVC に分離した場合の実装コードの行数と非分離の場合の実装コードの行数を比較した結果, 分離した場合は, 分離しなかった場合よりも 17 行増加した. しかし, 増加したコードの内容は, 処理を分けたことによる宣言や, メソッドの呼び出し, 値の受け渡しのための命令であり, プログラムを複雑化するものではない.

3.4. 実装手順について

MVC に分離したプログラムを Model 部→Control 部→View 部の順に実装していく手順により, GUI のレイアウトを意識することなく, 確実に動作するロジック部を実装できた. その後, GUI コンポーネントの配置やデザイン等の View 部の設計, 実装に専念することができた. すなわち, ロジック部の設計と, 見た目やレイアウトの設計である View 部の設計を分けて考えることができる.

ただ, このように Model 部→Control 部→View 部の順に実装していくことにより, 実装コードの構造は複雑化する傾向にある. これは 3.3 節でも述べたソースコードの増加にもその傾向を見ることができる. しかし, MVC に分けた場合のテストコードは, プログラムの流れを分かりやすく示しているため, プログラムの修正, 追加, 変更は容易である. したがって, 開発全体の効率は向上すると考えられる.

3.5. View 部のテストについて

当初, View 部分のテストはロジック部分を実装した後, 目で見ながら確認する方法を考えていたが, getLocationOnScreen()を用いて, 各 GUI の相対的な座標を比較することにより, テストが可能であることが分かった. これにより, 見た目のテストも JUnit を用いることにより, テスト可能であると言える. テストコードにテスト用の仮ウィンドウを生成するコードを記述することで, 実際の見た目も目で見て確認できる.

見た目のテストをどこまで行えば十分であるかという問題はあるが, 我々が行った実装においては, 各 GUI の相対的な座標を比較することと目で見て確認することで, ユニットテストの範疇は十分であると考えた.

3.6. 結論

XP によるテストファーストを用いた GUI アプリケーションの開発におけるユニットテストについて, プログラムを MVC に分離することで効率化を図ることを目標とした実装, 評価を行った結果, 以下のように結論付けることができた.

プログラムを MVC に分離し, 設計, 実装を行うことにより, より詳細なテストを行うことが可能となった. また, Model 部→Control 部→View 部の順に実装していく手順により, プログラムのロジック部を画面上での GUI コンポーネントの配置の変化に依存しない強固なものにできる. これらは, アプリケーション全体のエラーを減少させるという観点から大変有効である. さらに, MVC プログラムを分離することによってテスト工数を減少させることができた. また, View 部のテストも GUI コンポーネントの相対座標を比較することによりユニットテストが可能であることを示した. MVC の分離により, 全体のコード数, 実装コードの複雑さは増加傾向にはあったが, これはテストファースト開発の特徴でもあり, MVC を分離することにより発生した特有の問題ではないと考えた.

以上の結果から, 我々の提案する GUI アプリケーションの開発における MVC 分離の開発方法は, 従来ユニットテストが難しいとされていた GUI 開発においてもユニットテストが有効であるということを示すとともに, XP, テストファースト開発の利点を最大限に生かすことができる開発方法であるといえる.

4. 考察

4.1. XP の開発環境に関して

本研究では, XP の 14 のプラクティス[2]に準じ, 開発を進めていこうとしたのだが, 顧客がいないことやメンバが 2 人など環境や状況が違い検証不可能なプラクティスが多かった.

例えば，ペア・プログラミングについては，自宅で作業を進める事も多く，また作業を分担したほうが効率的であったと言える．週 40 時間労働にしても実質的には週 20 時間程度になってしまっていた．小さなリリースにしても実質的に 2〜3 週のサイクルになってしまった．この結果，我々学生には完全な 14 のプラクティスの適用は難しいとわかった．

XP におけるプラクティスは，適用する環境によって有効なものやそうでないものもある．そのため，XP を完全に適用することだけにとらわれず，適用する環境に応じて必要なプラクティスを取捨選択し，最適な開発環境を作り上げることが重要であると考えられる．その上で，XP の考え方やテストファースト，JUnit などのテスティングフレームワークは，開発期間の短縮，高品質を求める上で使う意義は十分にある．

4.2. 検証結果に対して

我々が提案，検証を行った GUI プログラムにおける MVC に分離した開発方法は，結論として，より詳細なテストができ，エラーの減少やテスト工数の減少から，開発の効率化が図れると述べた．これにより，従来テストが難しいとされてきた GUI のテストを効率よく行えるという点において一つの解決方法を見出せたといえるが，これは，MVC の分離方法に起因すると考えられる．

従来，GUI のテストが難しいとされてきたのは，Control 部を View 部に含めて考えられていたためである．そのため，View のテストは様々なパターンが生じ，複雑化してしまう．我々の MVC 分離方法では，イベント処理や値の受け渡し等を Control 部としたため，View 部のテストはプログラムの動作に影響しない，見た目のレイアウトのみとなる．Control 部のテスト数は増加するが，Control 部，すなわちロジック部分のテストは，テスト支援ツールを用いたテストで十分に効率よく対応できる．残る View 部のテストも，我々の検証においては GUI コンポーネントの相対位置をテストした上で，目で見て確認することで十分対応できた．

View 部において，さらに詳細なテストを行いたい場合は，具体的に GUI コンポーネントのサイズや位置の座標を指定し，テストすることも可能である．これにより，さらに規模が大きく，複雑なプログラムにもこの開発方法が適用できると考えられる．

しかし，テストコードにおいて座標を指定し，テストする方法は，実際の見た目を把握しにくい可能性がある．そこで，View 部のみ JBuilder 等に代表される RAD ツールを用いて設計し，後で Model 部，Control 部と統合する方法も，効率化の一つの手法として考えられる．RAD ツールでは，GUI コンポーネントの配置をマウス操作で行うことができ，レイアウトの調整も容易である．またそのような場合にも，MVC を分離した設計により，Model 部，Control 部との統合も容易であると考えられる．

5. まとめ

本研究では，XP によるテストファーストを用いた GUI アプリケーション開発において，ユニットテストの観点から効率化を図ることを目的とし，MVC を分離し Model→Control→View の順序でプログラムの実装を提案し，実装した．そして，我々が提案した手順や MVC に分離した場合でもテストが難しいとされていた View 部分，テストの柔軟性，テスト工数やコードの検証を行ってきた．その結果，従来難しいとされた GUI のユニットテストが可能であることを示すとともに，MVC に分けることがエラーの減少，開発効率の向上に有効であることを示すことができた．

今後の課題として行うべきことは，さらに大きなプログラムに適用した場合，また，プログラムに機能を追加した場合における MVC 分離の有効性の検証である．また，View 部の設計に RAD ツールを用いた手法の検証も必要である．本研究において，XP の開発プロセスへの適用が有効であるということが実証されたため，XP は拡張性の高い開発プロセスであるという観点から，上記の項目についても MVC 分離は有効であると考えられる．

謝辞

本研究を進めるにあたり，有益なアドバイスをいただいた研究室の皆さんや指導教員の青山幹雄先生に，この場を借りて深く感謝いたします．

参考文献

[1] 日本 XP ユーザグループ(著)，長瀬嘉秀(監修): eXtreme Programming テスト技法，翔泳社 (2001).

[2] P.McBreen(著)，村上雅章(訳):XP エクストリーム・プログラミング懐疑編，ピアソン・エデュケーション，(2002).

[3] 平鍋健児:eXtreme Programming の魅力を探る，JavaWorld, Jan.2001, pp. 87-95.

[4] 石井 勝:JUnit 活用の手引き，JavaWorld，Mar.2003, pp. 170-184.