ソフトウェアテストシンポジウム 2003
JaSST'03: Japan Symposium on Software Testing 2003

Delphi Application への WinRunner の適用手法の検討

古江 智和†

†株式会社フォーラムエイト宮崎支社 TestGroup 〒889-2155 宮崎市学園木花台西 2-1-1
E-mail: †furue@forum8.co.jp

あらまし MERCURY INTERACTIVE 社の自動テストツール WinRunner を用いて Borland Delphi で作成した Application をテストする際のポイントについて解説する．WinRunner は Delphi Application 認識の手段として 2 種類の Add-in を提供している．本論文では，各 Add-in のもたらす効果を明らかにした上で，テストに費やされるコストパフォーマンス及び開発プロセスにおけるテストの位置付けを考慮した際，どの Add-in を適用するのが適切か弊社の事例に基づいて考察を行う．

キーワード Delphi，WinRunner，Add-in，自動テスト

Approach for Delphi Application Automated Testing with WinRunner

Tomokazu FURUE†

†FORUM8 INC. Miyazaki Branch TestGroup 2-1-1 Gakuenkibanadai-nishi, Miyazaki, 889-2155 Japan
E-mail: †furue@forum8.co.jp

Abstract The main point in the case of testing Delphi Application using WinRunner is explained. The WinRunner offers two kinds of Add-in, in order to recognize Delphi Application. In this paper, the effect of Add-in is clarified first. Next, the cost performance of testing and positioning of it in a development process are taken into consideration. And Add-in which should be applied is examined.

Keyword Delphi，WinRunner，Add-in，Automated Testing

1. まえがき

弊社では社内で開発・販売を行っている製品群に対し，製品の一層の品質向上を目的として，2002 年 5 月に TestGroup が発足した．当 TestGroup では数社から販売されている自動テストツールに対して試用・評価を行った結果，当時既に Delphi6 への対応済みであった MERCURY INTERACTIVE 社の WinRunner を採用するに至った．

WinRunner は Delphi オブジェクトを認識するための Add-in を複数用意しているが、実際にこれらの Add-in を適用するに当たっては開発プロセスにおけるテストの位置付けを考慮した上で決定する必要があった．これは Add-in 適用の方法によって、テストに費やされるべきコストが大きく変わって来るからである．

本論文では WinRunner の Delphi Add-in の内容と適用例を幾つか示した上で、各適用パターンにおけるメリットについて考察する。これにより本テストツールを使用するテスト担当者が、どのような手法を用いてテストを実施するか判断する参考となれば幸いである。

2.ではまず，WinRunner がオブジェクトを認識する仕組みについて簡単に説明する．**3.**では Delphi Application を構成する各オブジェクトを幾つかの種類に分類し、その対応方法について各々説明する。また WinRunner が提供する 2 種類の Add-in について，先に分類したオブジェクトとの関係を示しつつ解説する．**4.**では Add-in 適用パターンを 3 つの種類に分類し，各パターン毎のオブジェクト認識率を測定した結果を比較する．**5.**では各適用パターンのメリットあるいはデメリットをコスト及び業務フローの面から考察する．

2. WinRunner の概要

WinRunner がオブジェクトを認識する仕組みに絞って説明する．

WinRunner のテストスクリプト(TSL)内では一般的に各オブジェクトは"論理名"で表現される．"論理名"とはスクリプトの可読性を高めるために用いられる一種のエイリアス(別名)である．この"論理名"は GUI マップと呼ばれるファイルの中で"物理的記述"と結びついている。"物理的記述"には WinRunner がテスト対象アプリケーションにアクセスするためにオブジェクトを一意に識別するためのプロパティ情報が記述されている．GUI マップファイルは"論理名"と"物理的記述"を結びつける役割を果たしており、WinRunner におけるオブジェクトの認識率は GUI マップの記述内容(特に物理的記述)に左右されると言ってもよい．

GUI マップの作成方法には以下のパターンがある．

・テストレコーディング(自動記録)時に自動作成

・対象となるオブジェクトの GUI マップ作成を明示的に指示（学習機能）

ユーザは GUI マップファイル内の"論理名","物理的記述"を自由に編集することが可能である。オブジェクト認識の仕組みを図 1 に示す.

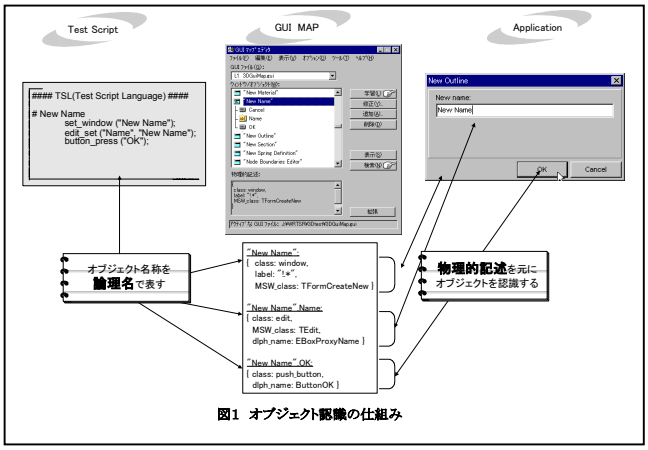

図1 オブジェクト認識の仕組み

3. Delphi 対応の概要

3.1 概要

WinRuuner における Delphi Application への対応とは，基本的に Delphi の VCL ライブラリが提供するオブジェクト群を WinRunner に登録することである．そしてテスト対象 Application を構成する各オブジェクトに対して WinRunner が最適な学習を行い，Appliccation の操作に対してはテスト･スクリプト言語(TSR)が最適な関数を使用して記録・再生出来るようにすることが目的である．

実際に弊社の Application を使用して WinRunner の自動記録・再生を行ったところ，オブジェクトを以下の３種類に分類することが出来た．

(1)標準 Delphi オブジェクト

Delphi Add-in を実装することにより、ほぼ問題なく自動記録・再生が可能なオブジェクト

(2)カスタム(コンポーネント)オブジェクト

スクリプトに手を入れることにより、記録再生が可能となるもの(類似クラスへのマッピング)

(3)認識不能オブジェクト

全く認識することの出来ないオブジェクト

以下の節で各オブジェクトへの対応方法を説明する．

3.2 標準 Delphi オブジェクト

WinRunner が提供する"Delphi Add-in Support"を組み込むことにより、Delphi に付属している多くのオブジェクト(コンポーネント)が WinRunner の認識対象となる．"Delphi Add-in Support"は次の 2 つの Type がある．(それぞれ仮に TypeA，TypeB と呼ぶことにする．)

(TypeA)WinRunner 本体に Load する Add-in

(TypeB)テスト対象 Application に組み込む Add-in

図 2 に両 Type の概要を示す.

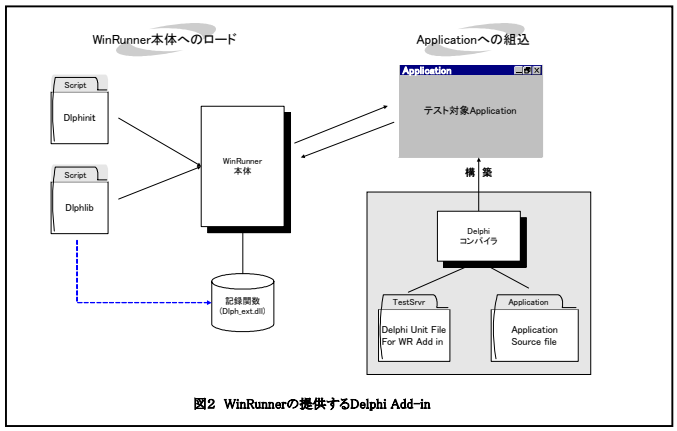

図2 WinRunnerの提供するDelphi Add-in

これらの Add-in でサポートされているオブジェクトを"標準 Delphi オブジェクト"と称する．各 Support の内容を以下に説明する．

3.2.1 TypeA

Delphi Add-in を Load すると"dlphinit"及び"dlphlib"スクリプトが WinRunner に読み込まれ，数十個の Delphi オブジェクトが WinRunner 本体に登録される．

(起動時もしくはテストに先立って読み込むスクリプトのことを初期化スクリプトまたは初期化テストと呼ぶことがある．)

"dlphinit"の一部を図 3 に示す.

```
##################################################################
# Map all standard Delphi classes
##################################################################
add_dlph_obj(“Tform”,”window”,”classMSW_classdlph_name",
                "label","Count_objects", DLPH_OBJ);
add_dlph_obj("TRadioGroup","object","classMSW_class dlph_name",
                "dlph_parent","Enabled", DLPH_OBJ);
add_dlph_obj("TEdit","edit","class MSW_class dlph_name",
                "dlph_parent","Compare", DLPH_OBJ);
```

図３　Dlphinit ファイル

add_dlph_obj の機能については割愛するが，本関数の実行により各々のオブジェクトが最適なクラスにマッピングされ，各クラスに用意された関数が使用可能になる．WinRunner7.5 に付属の Delphi Add-in では 66 個の標準的なオブジェクトが登録される他，Grid(TStringGrid，TDBGrid)等も登録される．例えば"TEdit"は"edit"クラスにマッピングされることにより edit_set, edit_replace 等のエディットボックス系の関数が使用可能となる。

Add-in の Load の有無によるスクリプトの相違を図４に示す.

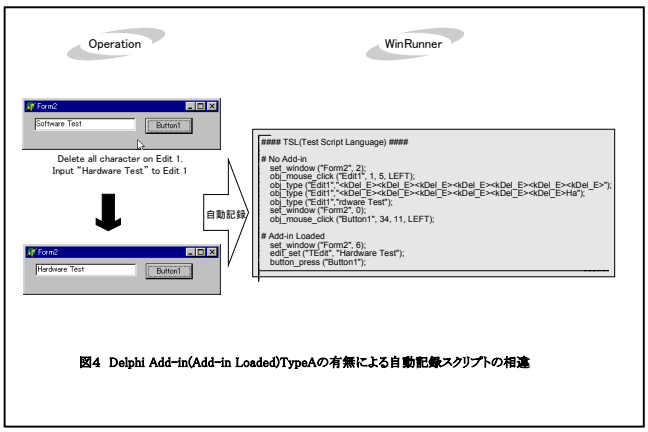

図4 Delphi Add-in(Add-in Loaded)TypeAの有無による自動記録スクリプトの相違

3.2.2 TypeB

テスト対象アプリケーションに WinRunner から提供されている Unit ファイル群(TestSrvr)を追加して再構築することにより，以下に示す効果が得られる.

- オブジェクトの Name プロパティが取得可能になり(dlph_parent)，認識率及びスクリプトの可読性向上が更に期待出来る.
- 一部のオブジェクトに関しては特別な関数が使用可能となる. (Grid に対する tbl_get_cell_data や TPanel に対する dlph_panel_button_press 等)

尚, TypeB の Add-in は TypeA と連携して初めて効力を発揮するものであり，TypeB のみを適用しても意味はない.

Add-in 適用パターンの種類によるスクリプトの相違を図５に示す.

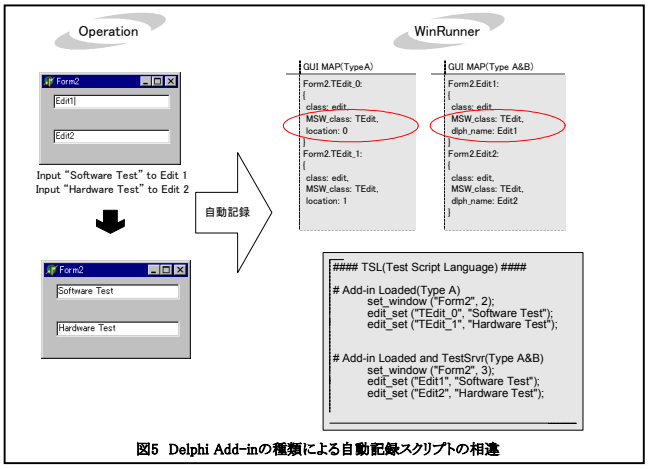

図5 Delphi Add-inの種類による自動記録スクリプトの相違

3.3 カスタムオブジェクト

カスタムオブジェクトに対しては類似クラスに"マッピング"することにより, 当該オブジェクトに対してより多くの TSR 関数が使用可能となる. あるいは学習されないオブジェクトを，学習するようにカスタマイズすることも可能である.

マッピング用スクリプトの例を図６に示す。

```
#################################################################
# Customized Object
#################################################################
add_dlph_obj ("F8ToolBarClass", "toolbarwindow32", "class
                MSW_class dlph_name",
                "dlph_parent","Enabled", DLPH_OBJ);
set_class_map("TToolBar", "toolbarwindow32");
set_record_attr("TToolBar", "class MSW_class", "label",
                "location");
set_record_method("TToolBar", RM_RECORD);
```

図６　類似クラスにマッピングしたスクリプト例

図中の 1 行目はカスタムコンポーネント"F8ToolBarClass"に対してツールバー専用の TSR 関数が使用できるように記述したものであり，2 行目以降は TToolBar オブジェクトに対する同様の処理である。

3.4 認識不能オブジェクト

WinRunner では HWND(ウィンドウハンドル)を消費しないオブジェクトは認識出来ない. 例えば TSpeedButton に対する操作は一切記録されない。

このようなオブジェクトには"仮想オブジェクト"を定義して対応することになる.

"仮想オブジェクト"はウィンドウ内の任意の矩形領域を GUI オブジェクトとして WinRunner に認識させる機能である.

仮想オブジェクトの使用例を図７に示す.

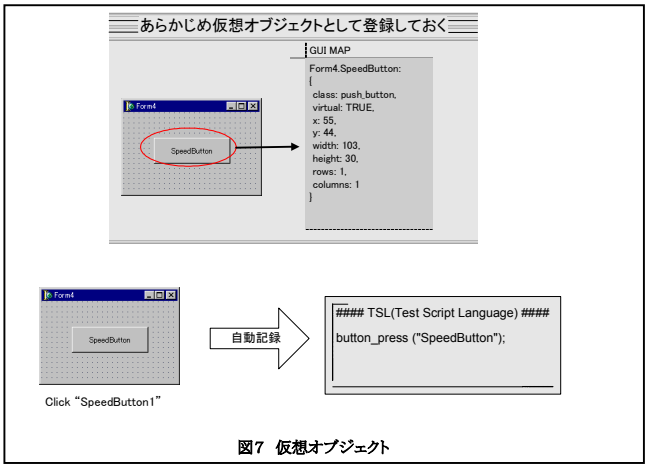

図7 仮想オブジェクト

4. 適用パターンと結果分析

4.1 認識率の比較

弊社製品の一つ UC-win/Frame(3D) を使用して Delphi Add-in の適用形態別にオブジェクトの認識率を測定した.

尚，ここで述べる"認識率"とは下記の条件を満たすものとする．

・シナリオに沿って Application の操作を行い，WinRunner で自動記録する．

・スクリプト及び GUI マップに対するカスタマイズは一切行わない．

・自動記録終了後，作成されたスクリプトを再生実行し、正しく再生された比率．

今回は 3 種類の環境でそれぞれ認識率の測定を行った。

・Pattern_A　Add-in 適用無し

・Pattern_B　Add-in を適用(Type A)

・Pattern_C　Add-in を適用(Type A + Type B)

測定結果を表 1 に示す．

表 1　Add-in パターン別認識率

	総 Step 数	再生動作		認識率
		OK	NG	
Pattern A	104	74	30	71%
Pattern B	104	81	23	78%
Pattern C	104	85	19	82%

各パターンによる認識率の相違は主に以下のような原因に起因するものであった．

Pattern_A と B の相違点

・同一画面に同種類(同じクラス)のオブジェクトが複数存在する場合，Add-in が適用されていないとオブジェクトによっては一意に識別することが出来ないことがある．

図８は label プロパティが取得出来ない為、コントロール ID が物理的記述に加えられた例である．

```
{
  class: object,
  label: 1,
  MSW_id: 2164510,
  MSW_class: TSpinEdit
}
```

図 8 Pattern_A における TSpinEdit の物理的記述

・類似クラスへのマッピングが行われていないため，正確に記録出来ない操作がある．

図 9 は ComboBox からあるエントリを選択した時の両適用パターンの記録例である．

```
#Pattern_A
  obj_mouse_click ("TComboBox", 174, 12, LEFT);
#Pattern_B
  list_select_item ("TComboBox_1", "Members");
```

図 9 TcomboBox の記録例

Pattern_B と C の相違点

・TPanel オブジェクトの認識

Pattern_C では関数"dlph_panel_button_press"が使用され，Panel オブジェクトに対する操作を記録出来ていたが Patten_B では全く記録出来ていなかった．

今回の測定では自動記録完了後，スクリプトを Step 走行させて認識率を正確に集計しているが，簡易的な方法としては一連の操作を記録した後，標準・カスタム各オブジェクトの出現率を集計するだけでおおよその認識率は求められることになる．

4.2 認識率の比較

表 1 に示した通り，Add-in 適用による認識率の向上はそれほど大きなものではない。但し，記録されたスクリプトの再利用性の高さ，あるいはメンテナンスの容易さ等はこの認識率と一致するものでないことに留意する必要がある．

各 Add-in 適用パターン別の GUI マップ及び自動記録スクリプトの特徴を表 2 に示す。

テストの実運用においてはオブジェクト認識率 100%でないと意味を成さない．各 Add-in 適用パターン別に認識率が 100%となるよう修正を加えた．修正の特徴を表 3 に，修正用スクリプトを図１０に示す．

表2　Add-in適用パターン別　GUIマップ及び自動記録スクリプトの特徴

Add-in 適用パターン	GUIマップ 物理的記述	論理名	スクリプト
Pattern_A	・メニュー項目を除く全てのオブジェクトが objectクラスにマッピングされる(*1)。 ・HWNDやコントロールIDが物理的記述に採用された場合、 再生時認識不能。	"クラス名" or "labelプロパティ"(*2)	ほとんど全ての操作を "obj_mouse_click" or "obj_type" で記録。

Pattern_B	・dlph_init内で定義済みのオブジェクトは
	各々最適なクラスにマッピングされる。
	・カスタムオブジェクトは全てobjectクラスにマッピングされる。
	このようなオブジェクトはPattern_Aと同様認識不能となる場合有り。
		同上	マッピングされたクラス固有の関数で記録。
			button_press();
			eidt_set();
			tab_select_item(); 等

Pattern_C	同上	Delphiオブジェクトの Nameプロパティ	Pattern_Bの内容に加え、 PanelやGridの操作関数が記録される。 dlph_panel_button_press(); tbl_get_cell_data(); 等

(*1)#32770等のOSに用意されたダイアログを除く
(*2)labelプロパティは多くのDelphiオブジェクトには存在しないが、caption等がlabelとして採用されている

表3　Add-in適用パターン別　テスト修正方法の特徴

Add-in 適用パターン	修正方法	修正対象オブジェクト
Pattern_A	オブジェクト毎に類似クラスにマッピングする関数を 初期化スクリプトに追加する。	ほぼ全てのオブジェクト
Pattern_B	同上	カスタムオブジェクト及び dlph_initに含まれていないDelphiオブジェクト
Pattern_C	同上	同上

```
#### TSL(Test Script Language) ####

#Pattern_A修正用初期化スクリプト-----------------------------------------------------
##Customized object
set_class_map("F8ToolBarClass", "toolbarwindow32");
set_record_attr("F8ToolBarClass", "class MSW_class", "label", "location");
set_record_method("F8ToolBarClass", RM_RECORD);

set_class_map("TF8CheckListBoxClass", "listbox");
set_record_attr("TF8CheckListBoxClass", "class MSW_class", "label", "location");
set_record_method("TF8CheckListBoxClass", RM_RECORD);

set_class_map("TF8Panel", "object");
set_record_attr("TF8Panel", "class MSW_class", "label", "location");
set_record_method("TF8Panel", RM_RECORD);

set_class_map("TF8Edit", "edit");
set_record_attr("TF8Edit", "class MSW_class", "label", "location");
set_record_method("TF8Edit", RM_RECORD);

set_class_map("TF8ComboBox", "combobox");
set_record_attr("TF8ComboBox", "class MSW_class", "label", "location");
set_record_method("TF8ComboBox", RM_RECORD);

set_class_map("TF8ListView", "syslistview32");
set_record_attr("TF8ListView", "class MSW_class", "label", "location");
set_record_method("TListView", RM_RECORD);

##Delphi standard object
set_class_map("TToolBar", "toolbarwindow32");
set_record_attr("TToolBar", "class MSW_class", "label", "location");
set_record_method("TToolBar", RM_RECORD);

set_class_map("TSpinEdit", "object");
set_record_attr("TSpinEdit", "class MSW_class", "label", "location");
set_record_method("TSpinEdit", RM_RECORD);

set_class_map("TPanel", "object");
set_record_attr("TPanel", "class MSW_class", "label", "location");
set_record_method("TPanel", RM_RECORD);

set_class_map("TPageControl", "SysTabControl32");
set_record_attr("TPageControl", "class MSW_class", "label", "location");
set_record_method("TPageControl", RM_RECORD);

set_class_map("TComboBox", "combobox");
set_record_attr("TComboBox", "class MSW_class", "label", "location");
set_record_method("TComboBox", RM_RECORD);

set_class_map("TListBox", "listbox");
set_record_attr("TListBox", "class MSW_class", "label", "location");
set_record_method("TListBox", RM_RECORD);

set_class_map("TListView", "syslistview32");
set_record_attr("TListView", "class MSW_class", "label", "location");
set_record_method("TListView", RM_RECORD);

set_class_map("TTreeView", "systreeview32");
set_record_attr("TTreeView", "class MSW_class", "label", "location");
set_record_method("TTreeView", RM_RECORD);


#Pattern_B修正用初期化スクリプト-----------------------------------------------------
add_dlph_obj ("TF8CheckListBoxClass", "listbox", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);
add_dlph_obj ("F8ToolBarClass", "toolbarwindow32", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);
add_dlph_obj ("TF8Panel", "object", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);
add_dlph_obj ("TF8Edit", "edit", "class MSW_class dlph_name",  "dlph_parent","Compare", DLPH_OBJ);
add_dlph_obj ("TF8ComboBox", "combobox", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);

add_dlph_obj ("TListView","syslistview32", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);
add_dlph_obj ("TToolBar", "toolbarwindow32", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);

#dlpj_Panel_Button_Pressが記録をHookしてしまうのでここで再定義してやる必要有
add_dlph_obj ("TPanel", "object", "class MSW_class dlph_name",  "dlph_parent", "Enabled", DLPH_OBJ);


#Pattern_C修正用初期化スクリプト-----------------------------------------------------
add_dlph_obj ("TF8CheckListBoxClass", "listbox", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);
add_dlph_obj ("F8ToolBarClass", "toolbarwindow32", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);
add_dlph_obj ("TF8Panel", "object", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);
add_dlph_obj ("TF8Edit", "edit", "class MSW_class dlph_name",  "dlph_parent","Compare", DLPH_OBJ);
add_dlph_obj ("TF8ComboBox", "combobox", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);

add_dlph_obj ("TListView","syslistview32", "class MSW_class dlph_name",  "dlph_parent","Selection", DLPH_OBJ);
add_dlph_obj ("TToolBar", "toolbarwindow32", "class MSW_class dlph_name",  "dlph_parent","Enabled", DLPH_OBJ);
```

図１０　Add-in適用パターンテスト別　修正用初期化スクリプト

5. 考察

5.1 コストの考察

テストスクリプト及びGUIマップの作成・維持コスト面に着目して考えると，当然のことではあるが全てのAdd-inを適用する方法(Pattern_C)が最もパフォーマンスが良い．特にスクリプト内で記述されるオブジェクトの論理名に判読が容易な名称が用いられることはスクリプトメンテナンスの見地からは無視出来ない重要な要素であり，その意味でオブジェクトのNameプロパティをデフォルトで論理名に採用してくれる適用方法には魅力がある．

次にPattern_AとBを比較すると，作成・維持コストの差はカスタムオブジェクトの使用個数及び使用割合に左右されるところが大きい．カスタムオブジェクトの割合が大きくなるにつれコスト差は小さくなる．仮にApplication内の全てのオブジェクトがカスタムオブジェクトであればPattern_AとBのコスト差はゼロに近いものになり，逆に全てが標準Delphiオブジェクトであればかなり大きな差が発生する．

次にテストに要するコストであるが，Pattern_Cではテスト対象アプリケーションに毎回テスト用ユニットを組み込む必要が生ずる．これはテストの実業務への導入においてデリケートな問題と成り得る要素である．この作業はテスト版Application更新の都度発生する単なるランニングコストの問題に止まらず，組込作業の担当や組込範囲の問題等，社内の業務フローや社としての製品に対するコンセンサスまで考慮すべき側面を含んでいるからである．本件については次節で詳しく考察する．ツール自体の導入・維持コストに関しては表4を参照されたい．

5.2 業務フローの考察

自動テストの作成・維持及び実施・報告はTest専門部署で行う前提に立つと，Add-inの適用パターン毎に想定される業務フローは以下の３種類である．

(1) Add-in Pattern_A or Bの場合（図１１参照）
(2) Add-in Pattern_C [TestGroupでAdd-in Unit組込]（図１２参照）
(3) Add-in Pattern_C [開発GroupでAdd-in Unit組込]（図１３参照）

各々想定される問題点を以下に示す．

(1) テストの作成・維持コストが(2)(3)より比較的大きい．
(2) ・TestGroup側に開発Groupと同様のApplication構築環境が必要になる．
担当する製品の増加に比例して，環境の維持が困難になって行くと予想される．
・開発Group側のテスト用Build作成の都度，TestGroup側でもTestSrvr Unitを組み込んだBuildを作成する必要が発生する．
テストPhase中に短いスパンでBuildが作成されるような場合，TestGroup側の作業負担が大きくなる．
(3) 下記に示すような問題は，社内で充分なコンセンサスを取らない限り実施困難．
・開発側で作成するBuildにTestUnit(TestSrver)を組み込んでもらう必要がある．物理的負担は少ないが，心理的抵抗が大きい．
・TestUnitを出荷製品にまで組込むか否か．
組込まなければ実際はテストしていないモジュールが製品として出荷されることになる．

5.3 社内における適用手法の検討

平成14年秋より弊社TestGroupは自社開発製品への自動テスト導入を開始した．2つの製品(UC-win/Road, UC-win/Frame(3D) 図１４，１５参照)への導入が決まっていたが当時まだどのAdd-in適用パターンが最も有効か模索していた段階であったため，Roadについては Pattern_Bで，Frame(3D)については Pattern_C(Applicationへの Unit組込は TestGroupが担当)を適用して各々の有効性を検証することにした．

実製品に対する自動テスト作成作業を通して得た所感は以下の通りである．

・製品におけるカスタムオブジェクトの使用割合が高い．特に製品のオペレーションを通して数多く使用されるオブジェクトや細かい設定・操作等を行うオブジェクト程社内で開発したカスタムオブジェクトが使用される傾向にある．このため，Delphi Add-inを２種類とも適用しているPattern_Cのテストでさえも記録したテストに再生不良 Stepが数多く含まれることになり，スクリプト及びGUIマップ修正にかなりのコストを要することになった．
・TestGroup側でのBuild作成工数を無視出来ない場合がある．製品のリリース期日近くになると１日2,3回のBuildが作成されるようなケースもあり，その都度 TestGroup側で最新ソースファイルを取り込んだ上でTestUnitを組み込んだBuildを作成する作業は，工期が差し迫っている時期には大きな負担となった．

また自動テストの導入は現在の作業を通して社内での実績を作っている段階であり，現状では極力開発Group側に負担をかけない考慮も必要である．

上記の様な理由を検討した結果，TestGroup現時点の方針としては，Add-in 適用パターンの Pattern_Bが最も適しているとの結論に至っている．

表4　Add-in適用パターン別　コスト比較

Add-in 適用パターン	テスト作成・維持コスト	テスト実行時コスト	ツール自体のコスト
Pattern_A	×(*1)	ー	○(*5)
Pattern_B	○(*2)	ー	×(*6)
Pattern_C	◎(*3)	×(*4)	×(*6)

(*1) ほぼ全てのオブジェクトが自動ではobjectクラスとしてしか認識されない。
(*2) 標準Delphiオブジェクトに関しては最適なクラスにマッピングされる。
(*3) ・GridやPanelオブジェクトに対して、個別の便利な関数が使用可能となる。
　　・Nameプロパティを取得出来る。
(*4) ApplicationにTestUnitを組込んで、再構築する必要有り。詳細は”5．2 業務フロー”参照。
(*5) WinRunner本体以外のコストは発生しない。
(*6) 組込み用TestUnit(TestSrvr)は別売り。購入及び維持費用が別途発生。

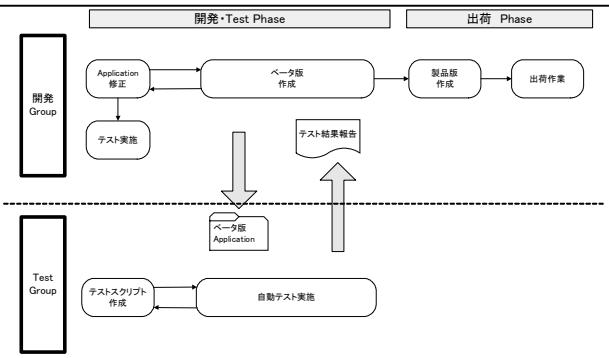

図11　業務フロー①(Add-in 適用Pattern_A or Bの場合)

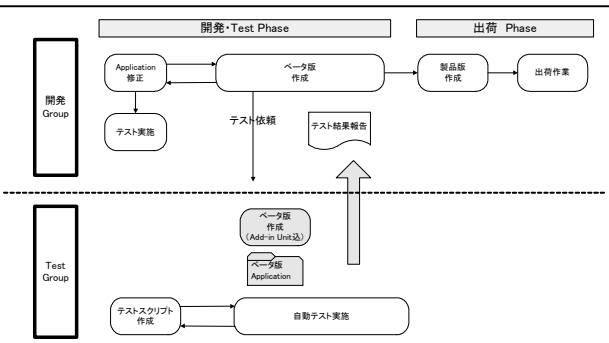

図12　業務フロー②(Add-in 適用Pattern_C [TestGroupでAdd-in Unit組込])

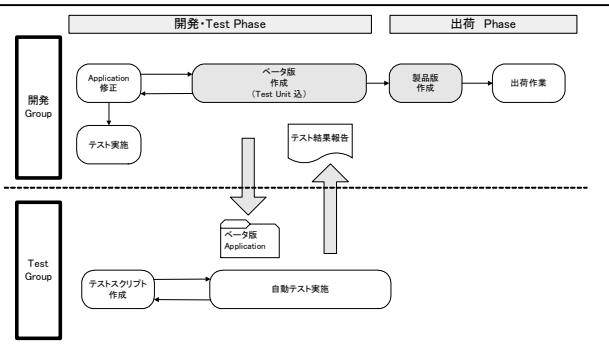

図13　業務フロー③(Add-in 適用Pattern_C [開発GroupでAdd-in Unit組込])

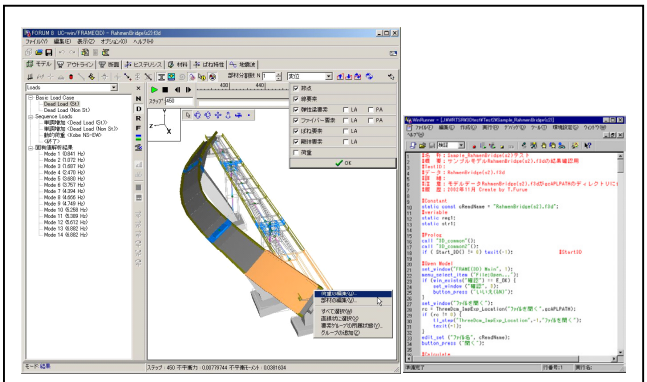

図14　テスト対象Application　UC-win/Frame(3D)

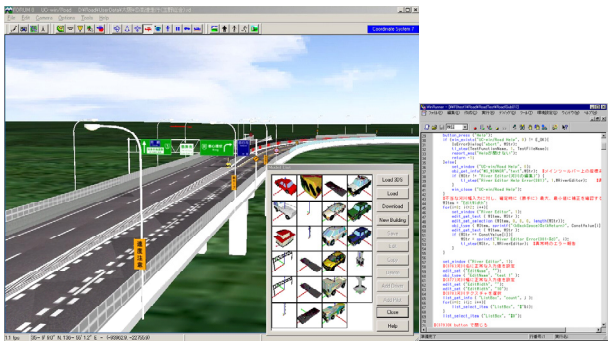

図15　テスト対象Application　UC-win/Road

6. むすび

本論文では弊社 TestGroup の業務経験に基づき，WinRunner の Delphi Add-in パターンを調査し，各 Add-in の有効性をコスト及び業務フローに鑑みて分析した．

現在弊社 TestGroup では初期の 2 製品に更に 2 製品を加え，4 製品に対して自動テストを実施しているが今後の課題として自動テスト作成自体のコストパフォーマンス改善があげられる．自動テストの作成工数は 1 回のマニュアルテストとは比較できないほど大きいため，必然的に再利用性の高さやメンテナンスの容易さが要求される．現在でも Free ライブラリ(注1) の利用

や，関数の共通化等でパフォーマンスの向上を図っているが，今後特に取り組むべきものとしてカスタムオブジェクトに対する"ユーザ定義記録関数"の開発をあげたい．これは別途作成したDLLと連係することにより，ユーザが意図(DLLにて設計)した通りのステートメントがスクリプトに記録される仕組みである．複数の製品間で使用頻度の高いカスタムオブジェクトに適用することにより，テストの作成・維持コストに大きく寄与するものと考えられる．

(注1)：csolib(http://wilsonmar.com/1winrun.htm),
func_ra(http://www.itechnologist.com/tech/) 等．

文　献

[1] 唐島めぐみ，烏山幸嗣，“ソフトウエアのテスト自動化手法，”OMRON TECHNICS, Vol.39, No.4, pp.330(90)-333(93), Jan.2000.

[2] “WinRunnerカスタマイズ・ガイド Version7.01，”マーキュリー・インタラクティブ・ジャパン株式会社, 2001.

[3] “WinRunnerTSLリファレンス・ガイドVersion7.01，”マーキュリー・インタラクティブ・ジャパン株式会社, 2001.

[4] “TestSuite7.0 製品概要説明書，”株式会社アシスト