低頻度不具合の再現テストにおける GUI テスト自動化ツールの導入効果

Effectiveness of a test automation tool in reproduction testing of rare problems

NEC パーソナルコンピュータ株式会社　PM グループ
NEC Personal Computers, Ltd., PM Group
森 一郎
Ichiro Mori

Abstract We make and sell NEC's personal computer products. We bundle 64bit OS and our original apps and ISV apps on our products. Our department's mission is to deliver products to our customers without any OS problems. OS problems found in the system test by development departments and test departments and defects found in QA division's audit need to be reproduced and fixed. Reproduction tests are done manually by external test sites. OS problems include rare and critical ones, less than 0.1% frequency rate. We need to repeat the same operation more than 1,000 times to reproduce the defect and analyze it. This will consume much man-power. We applied a GUI test automation tool to the reproduction testing of a rare OS problem. Then we found the tool reduced the man-power for reproduction testing. In this paper, we introduce how to apply the GUI test automation tool to reproduction testing for actual products, and explain the results.

1. はじめに

当社はパーソナルコンピュータ（以下、PC）の製造・販売を行っている。当社の PC 製品はオペレーティングシステム（以下、OS）として M 社製 64bit OS を搭載し、同 OS 上で動作する自社製および ISV（独立系ソフトウェアベンダ）製アプリをプリインストールしている。筆者が所属するソフトウェア・インテグレーション部門の任務は出荷判定までに、検出されたすべての OS 不具合の原因を究明し、処置方針を決定することにある。

開発部門・評価部門でのシステムテスト、および品質保証部門による出荷監査で検出された OS 不具合の再現テストは外部へ委託しており、基本的にはすべて人手で行っている。OS 不具合の中には発生率が 0.1%以下という低頻度で致命的な不具合も含まれるが、原因調査のため、このような低頻度で致命的な不具合を再現させるには、同じ操作を 1,000 回以上繰り返すことが必要となる。そのため、多大なテスト工数がかかっていた。

OS 起動時や終了時にハングアップやブルースクリーンなどの致命的な現象が発生したとき、当社ではこれまで自社製の連続再起動試験ツール「Aginger」を用いた再現テストを行っていた。Aginger は、OS 起動時に自動起動し、シャットダウン/休止/スリープを実行後、OS の内部タイマーを利用して一定時間後に OS を再起動させることができるツールである。

Aginger は、グラフィカルユーザインタフェース（以下、GUI）操作やキーボード入力の機能は備えていない。そのため、OS 再起動後のサインインなど何らかのユーザ入力を行った場合にのみ発生する現象の再現テストには使えず、人手で同じ操作を繰り返していた。その結果、多大なテスト工数がかかっていた。そこで、再現テストに市販の GUI テスト自動化ツールを活用することを考え、eggPlant 社の「eggPlant Functional」を導入した。

同ツールの導入効果を測るため、製品開発中に発生したある低頻度 OS 不具合について、人手でおこなう再現テストの工数と、eggPlant Functional を用いて自動化した再現テストの工数を比較した。その結果、再現テスト工数を約 17 分の 1 に削減でき、大きな効果が得られた。本論文ではその成果を報告する。

神奈川県横浜市西区みなとみらい 3-6-1 みなとみらいセンタービル
Minatomirai Center Building, 3-6-1, Minatomirai, Nishi-ku, Yokohama-shi, Kanagawa Japan
Tel: 080-3579-3187　　e-mail:imori@necp.co.jp
PM グループ　エキスパート

2 章では従来の再現テスト方法とその課題について述べ、3 章でテスト自動化の概要を説明する。4 章ではGUI テスト自動化ツール導入効果検証の全容を説明し、5 章で考察を加え、6 章で結論を述べる。

2. 従来の再現テスト方法

2.1. 人手による再現テスト

システムテストで、ハングアップ、ブルースクリーン、機能不全などユーザにとって致命的な現象が発生した場合、テスト部門は社内のバグトラッキングシステムにバグチケットを登録する。ソフトウェア・インテグレーション部門は、そのバグチケットの情報に基づき、OS 側からの原因調査を開始する。現象発生機と同一機種を準備し、再現手順通りに操作して再現を試みる。これが再現テストである。再現テストを行う目的には、現象の再現率、機種依存性、致命度の確認のほか、OS 側からの原因調査に必要な各種ログ/メモリダンプの取得、現象を回避する手順の調査も含まれる。現象の再現率は機種毎に、再現回数÷試行回数[%]で求める。

2.2. 連続再起動試験ツールによる再現テスト

当社 PC はコンシューマ向け、ビジネス向け合わせて 24 機種あり、機種毎に異なるデバイス、搭載アプリ、出荷時設定等の組合せがある。OS 機能に関しては全機種で同じテスト項目を実施する。このうち、システム安定性の確認を目的とした連続再起動試験では、OS のシャットダウンと再起動、休止と復帰、スリープと復帰という電源操作をそれぞれ 1,000 回以上繰り返す。この操作を人手でやるには多大なテスト工数がかかるため、連続再起動試験ツールを用いて自動化している。連続再起動試験ツールとして自社製の「Aginger」を用いている。

Aginger は、OS 起動時に自動起動し、シャットダウン/休止/スリープを実行後、OS の内部タイマーを利用して一定時間後に OS を再起動させることができるツールである。

当社では、不具合検出後にその原因調査目的で実施する再現テストにおいても、Aginger を使用している。

2.3. 再現テストにおける課題

Aginger は OS のシャットダウンと再起動、休止と復帰、スリープと復帰という電源操作を指定回数分、自動実行できるが、テスターによる GUI 操作やキーボード入力は受け付けていない。したがって例えば、再起動後のサインインなど何らかのユーザ入力を行った場合にのみ発生する現象は、Aginger 単独では再現できず、人手が必要となる。その結果、このような現象の再現テストに多大な工数がかかっていた。この課題を解決すべくテスト自動化に取り組んだ。

3. テスト自動化

3.1. テスト自動化の概要

前述の課題を解決するには、GUI 操作やキーボード入力を自動化する必要がある。そこで、市販の GUI テスト自動化ツール[1]を活用することを考えた。GUI テスト自動化ツールは、テスターが一度行った操作手順を記録してテストスクリプトの形で保存しておき、二度目からはテストスクリプト実行により、操作手順を再現できるツールである。

3.2. GUI テスト自動化ツールの選定

GUI テスト自動化ツールの選定においては、テスト種類により最適なツールを選択すべきであり、使い勝手や簡単さなども考慮する必要がある[2]。筆者は複数ツールのスペックを比較検討の上、eggPlant 社の「eggPlant Functional」（以下、eggPlant）を採用した。eggPlant は LAN 経由で解析 PC からテスト対象 PC にリモート接続し、解析 PC から操作した結果をテストスクリプトとして出力し、そのテストスクリプト実行により同じ操作手順を自動実行できる、キャプチャ＆リプレイツール[3]の１つである。図 1 に eggPlant の操作画面例を示す。

eggPlant の特長として、図 2 に示す通り、習得が容易であること、画像と文字を両方認識できるこ

と、対象が広いことが挙げられる。特に、画像認識と文字認識の両方の技術を駆使して、テストスクリプトが自動生成できる点は、当社での再現テストにおいて非常に有用である。例えば、テスターがどのボタンを押したかをキャプチャしてリプレイするのに、画面上の座標を用いる方式では、画面の解像度が異なる装置では同じテストスクリプトが使えない。しかし、画像認識を用いる方式では同じボタンのアイコンを探してクリックすることができ、文字認識を用いる方式では同じボタン名の文字列を探してクリックすることが可能である。したがって画面の解像度が異なる装置で同じテストスクリプトを利用できる。さらに、あるテストケースを実行した結果、成功時の画面が表示されたのか、失敗時の画面が表示されたのかを、画像認識あるいは文字認識により判定できる。自動生成されたテストスクリプトを人手で修正する必要はあるものの、テスト実行結果の判定までを自動化可能である。

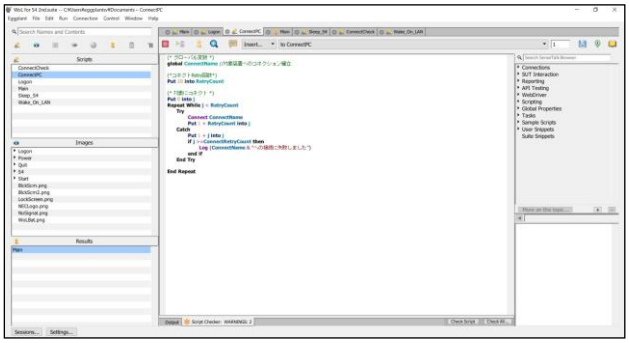

図 1 eggPlant の操作画面例

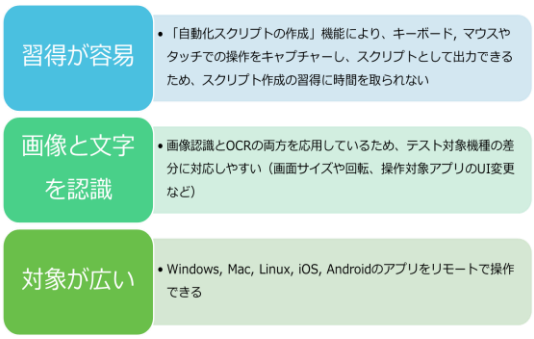

図 2 eggPlant の特長

3.3. GUI テスト自動化ツールの過去の成果

eggPlant を用いてソフトウェアテストを自動化する試みはこれまでに何度か社内で実施してきた。当初は ISV 製アプリのすべての機能を人手で操作する受入テストの効率化を狙っていた。

人手で行う従来テストの場合、ある事務系アプリのテスト工数は 524 人 H であった。これに対し、eggPlant で自動化したテスト工数の見積りは 85 人 H となった。従って、受入テスト工数を約 1/6 に効率化できることが見込まれ、大きな効果が得られた。

ただし、この結果は eggPlant に精通したテスト自動化専任者を想定した見積りである。製品適用のためにはテスト委託先にテスト自動化専任者を置けるかというリソース面の課題が残る。また、事務系アプリでは自動化できたテスト項目が多かったが、TV や動画編集等の AV 系アプリでは、操作結果を目や耳で確認しなければならないなど、自動化できる項目が限られると予想している。

この試みから、受入テストを自動化するにあたり、

課題①eggPlant に精通したテスト自動化専任者のアサイン

課題②自動化対象アプリの選定

という 2 つの大きな課題をクリアしなければならないことがわかっていた。

3.4. テスト自動化専任者のアサイン

今回、OS 不具合の再現テストを eggPlant で自動化するには、テストスクリプトを正しく書いて実行させるというプログラミングスキルがテスターに求められる。そこで当社の不具合調査担当チームの中で、最もプログラミング経験のあるテスターをアサインした。プログラミング経験があるとはいえ、eggPlant 用のテストスクリプト作成は初めてである。そこで eggPlant の正規販売代理店であり、自らも第三者検証サービスを提供している株式会社アドックインターナショナルに協力を要請した。すなわち、eggPlant の有償技術サポートの枠で、当社テスターのテストスクリプト作成を支援していただくことにしたのである。

この施策によって、社内に eggPlant に精通したテスターがいなくとも、テスト自動化を進めることができた。

3.5. 自動化対象アプリの選定

テスト自動化には必ず、テストスクリプト作成工数がかかる。そこで、一度作ったテストスクリプトを何度も使い回すことができれば効果的である[1]。その点において、事務系アプリの受入テストではメニューを開いて一つを選択する、といった単純な操作を何度も繰り返す。また、各テスト項目をすべての出荷対象機種に対して、繰り返す。そのために高い効果が見込めた。繰り返し回数が多ければ多いほど、自動化の効果が出る。

今回、低頻度不具合の再現テストに着目した理由は、繰り返し回数が非常に多いからであった。システムテストのフェーズで1機種あたり 1,000 回以上行われる連続再起動試験において、例えば1回だけ、OS のハングアップが発生したとする。それが当社のバグ管理システムに登録されると、今度は再現テストとして同じ操作を数千回繰り返すことになる。低頻度不具合の場合、同じ手順を 1,000 回試行すれば、必ず1回再現するというものではないからである。

また再現後も、原因分析のためのログ収集や、他機種でも再現するかどうかの確認、さらに修正モジュール入手後の不再現確認などを行っていると、合計で何万回というオーダーで繰り返すことになる。テストスクリプト作成にある程度の工数がかかったとしても、これだけの回数、繰り返し使うことができれば、テスト自動化が工数削減につながるであろうと期待できる。

4. GUI テスト自動化ツール導入効果検証

4.1. 検証対象とする OS 不具合

低頻度不具合の再現テストへの eggPlant 導入効果を検証するにあたり、検証対象とする OS 不具合を選定した。検証当時の製品開発において再現テストが難航していた OS 不具合の1つに「休止状態から復帰後にサインインすると、ディスプレイ全面に青画面が表示される」という現象があった。本現象はシステムテスト中に検出されたが、再現率が 4,000 分の 1 と極めて低頻度であった。また、現象発生時には未保存のユーザデータを失うという点で致命的でもあった。このように低頻度かつ致命的な OS 不具合の再現テストを自動化する手法が確立できれば、今後も役立つため、この OS 不具合を検証対象とした。

この OS 不具合は当初、Aginger で再現を試みたものの、一度も再現しなかった。Aginger による再現テストでは、休止状態からの復帰が電源ボタン操作によるものではなく、内部タイマーによるものであることや、サインイン操作をスキップしていることが原因していると考えられる。Aginger では人手による操作手順を忠実に再現できない。そこで Aginger を用いない自動再現方法を考案することとした。

4.2. 検証方法

今回構築したテスト自動化環境のシステム構成図を図 3 に示す。

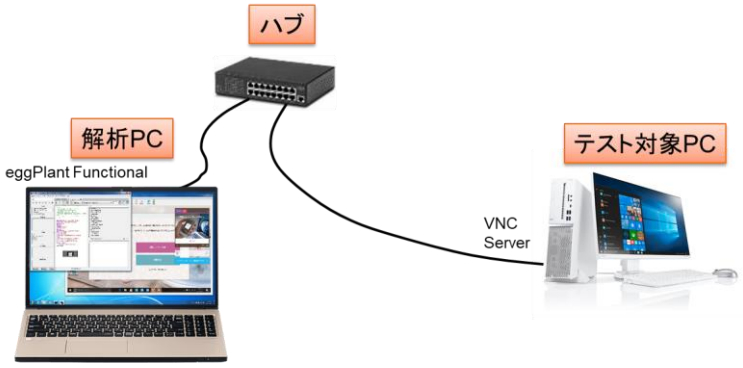

図 3 テスト自動化環境のシステム構成 A

図 3 を参照するとテスト対象 PC には、ハブを介して、eggPlant を導入した解析 PC が有線 LAN で接続されている。

eggPlant を導入した解析 PC を用いて、リモート接続したテスト対象 PC の GUI を操作し、その結果をテストスクリプトとして出力しておくと、その後、同じ操作手順を自動実行できる。検証対象 OS 不具合を再現させる手順は①休止→②電源ボタンを押して復帰→③サインイン画面をクリック→④デスクトップ画面表示の判定という 4 つのステップの繰り返しである。テスト自動化のためにはこれら 4 つのステップをそれぞれ自動化すればよい。

手順①の休止については OS 標準のメニューを操作し、テストスクリプトを自動生成すればよいので、eggPlant で容易に自動化できる。

手順④のデスクトップ画面表示の判定には eggPlant の画像認識機能を用いた。毎回同じ壁紙が表示されるデスクトップ画面の一部をキャプチャしておき、その画像が見つかれば正常、それ以外は異常と判定する。この判定のためのテストスクリプトは自動生成されないため、人手で作成した。

手順②と③には特別な工夫が必要であった。eggPlant はソフトウェアであり、手順②の電源ボタンを押して復帰という物理的な操作は自動化できない。Aginger を用いれば内部タイマーによる復帰は可能だが、本現象は Aginger を用いると再現できないことがわかっていたため、代わりに Wake On LAN を用いることとした。Wake On LAN は同一ネットワーク上にある別の PC からマジックパケットを送出することにより、テスト対象 PC の電源を入れることができる OS 標準機能である。同一ネットワーク上に接続した Wake On LAN トリガー用 PC からマジックパケットを送出する操作を eggPlant のテストスクリプトとして自動化した。

Wake On LAN トリガー用 PC を追加したテスト自動化環境のシステム構成図を図 4 に示す。

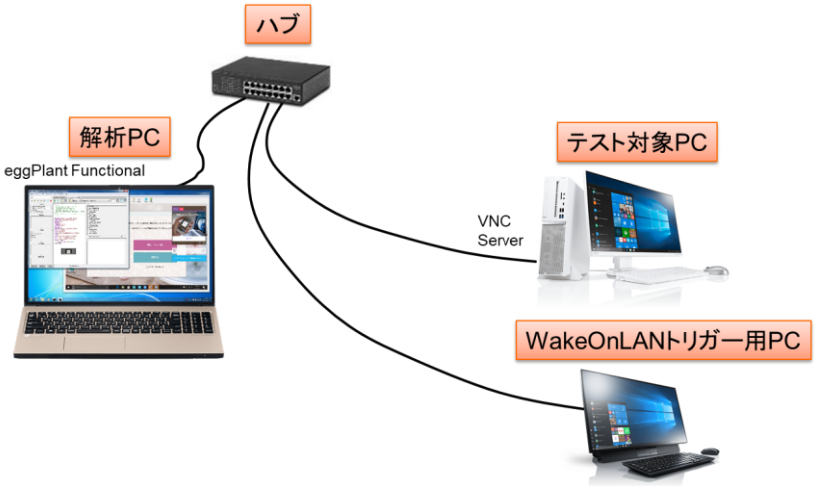

図 4 テスト自動化環境のシステム構成 B

手順③のサインイン画面クリックは eggPlant のみでは自動化できなかった。サインイン画面というのは OS がデスクトップ画面を表示する前に出す画面であり、この時点ではまだ eggPlant がテスト対象 PC にリモート接続できないためである。そこで、Adder 社のリモートスイッチ「Adderlink Digital ipeps」を導入した。本リモートスイッチをテスト対象 PC とハブとの間に挿入し、テスト対象 PC の入力インタフェース（マウスとキーボード）を横取りして、eggPlant から直接入力できるようにした。リモートスイッチの導入により、サインイン画面でのクリックを自動化することに成功した。

リモートスイッチを追加したテスト自動化環境のシステム構成図を図 5 に示す。

図 5 テスト自動化環境のシステム構成 C

4.3. 作成したテストスクリプト

今回の再現テストのために作成したテストスクリプトの内容と規模を表 1 に示す。eggPlant を使用するのは初めてのテスターが、試行錯誤しながら作成した。コマンドの使い方がわからないときには、eggPlant 技術サポート担当にメールで質問をして解決した。最終的には、eggPlant 技術サポート担当によるコードレビューを経て、完成版とした。完成したテストスクリプトの全コードを図 6 に示す。

図 5 のテスト自動化環境で、図 6 のテストスクリプトを実行することにより、これまで手動で行っていた低頻度不具合の再現テストを自動で何千回、何万回と繰り返せるようになった。このテストスクリプトを夜間に実行させておいたところ、208 回目で青画面となる現象を再現できた。

表 1　作成したテストスクリプトの内容と規模

区分	テストスクリプト名	処理	行数
メインルーチン	Main.script	1.検証PC上のeggPlantからテスト対象PCにリモート接続 2.テスト対象PCを休止状態にする 3.Wake On LANトリガー用PCからマジックパケットを発行することにより、テスト対象PCを遠隔起動 4.テスト対象PCにサインイン 5.サインイン成功時は回数を数え、サインイン失敗時はエラー出力 6.処理2～5を10,000回繰り返す	64
サブルーチン	ConnectPC.script	検証PC上のeggPlantから指定のPCにリモート接続	20
	ConnectCheck.script	画面上に期待する画像があるかチェック	33
	Sleep_S4.script	休止状態に入る	27
	Wake_On_LAN.Script	Wake On LANトリガー用PCに接続後、マジックパケット発行のバッチファイルを実行	33
	Logon.script	サインインする	57
合計			234

Main.script

```
//グローバル変数
global ConnectName , KVM , WOLPC //対象装置へのコネクション確立
global Start_Buttom , Power_Buttom , S4_Buttom //Sleep_S4用
global Logon_Buttom , LockScreen , LogonCheck //Logon(サインイン)用
global WOLBAT //Run_WOL用
global TRC , i //試験設定及び実施回数カウンタ用

//コネクション名の定義コナ
Put "KVM" into KVM
Put "WOLPC" into WOLPC

//試験設定回数を設定
set TRC = 10000

//試験実施回数初期値を設定
set i = 0

//Sleep_S4の準備
Put "Start¥Start_KVM" into Start_Buttom
Put "Power¥Power_KVM" into Power_Buttom
Put "S4¥S4_KVM" into S4_Buttom

//Logonの準備
Put "Logon¥Logon_KVM" into Logon_Buttom
Put "Logon¥Lock_KVM" into LockScreen
Put "Logon¥LogCheck_KVM" into LogonCheck

//WOLの準備
Put "WoLBat" into WOLBAT

//KVMに接続
set ConnectName = KVM
Run "ConnectPC" //スクリプト名"ConnectPC"を実行
Run "ConnectCheck" //スクリプト名"ConnectCheck"を実行
Run "Logon" //スクリプト名"Logon"を実行

//変数"i"が10000未満の限り試験対象を繰返し実行する
Repeat While i < TRC
    //S4に入ったKVMとの接続を切る
    Run "Sleep_S4" //スクリプト名"Sleep_S4"を実行
    Disconnect ConnectName //ConnectNameで指定した接続先から切断
    wait 60 //1分待機

    //WoL用PCに接続
    Run "Wake_On_LAN" //スクリプト名"Wake_On_LAN"を実行
    wait 60 //1分待機

    //KVMに接続
    set ConnectName = KVM //ConnectNameをKVMに設定
    Run "ConnectPC" //スクリプト名"ConnectPC"を実行
    Run "ConnectCheck" //スクリプト名"ConnectCheck"を実行

    //ログイン試験
    Run "Logon" //スクリプト名"Logon"を実行

    if i > TRC then //変数"i"がTRCを上回る場合にエラーメッセージを残す
        LogError ("青画面出現")
    else        //成功回数をログに記録し、変数"i"をカウントアップ
        Log(i + 1 & "回成功")
        set i = i + 1
    end if
end repeat
```

ConnectPC.script

```
//グローバル変数
global ConnectName //対象装置へのコネクション確立

//コネクトRetry回数を10に設定
Put 10 into RetryCount

//対象にコネクト
Put 0 into j
Repeat While j < RetryCount //eggPlantから対象装置への接続可否判定
    Try //成功すれば処理終了。
        Connect ConnectName
        Put 1 + RetryCount into j
    Catch //失敗時の処理。一定回数分失敗でログ表示。
        Put 1 + j into j
        if j >=ConnectRetryCount then
            Log (ConnectName & "への接続に失敗しました")
        end if
    End Try
End Repeat
```

ConnectCheck.script

```
//グローバル変数
global ConnectName
global Start_Buttom , Power_Buttom , S4_Buttom //Sleep_S4用
global Logon_Buttom , LockScreen , LogonCheck
global Start_Buttom
global ConnectName
global WOLBat
global TRC , i

//ConnectCheckカウンタ初期化
set CC = 0

//CCが10未満である限り画像認識によるチェックを実行
repeat while CC < 10
    if ImageFound(Image:LockScreen,WaitFor :1)then
        set CC = 10
    else if ImageFound(Image :"NECLogo",WaitFor:1)then
        set CC = 10
    else if  ImageFound(Image:Logon_Buttom,WaitFor:1)then
        set CC = 10
    else if ImageFound(Image:Logon_Buttom & "2",WaitFor:1)then
        set CC = 10
    else if ImageFound(Image :"WoLBat",WaitFor:1)then
        set CC = 10
    else if ImageFound(Image:LogonCheck,WaitFor:1)then
        set CC = 10
    else
        Disconnect ConnectName
        wait 5
        Run "ConnectPC"
        set CC = CC + 1
    end if
end repeat
```

Sleep_S4.script

```
//グローバル変数
global ConnectName //対象装置へのコネクション確立
global Start_Buttom , Power_Buttom , S4_Buttom //Sleep_S4用

//S4RepeatCount設定
Put 10 into S4RepeatCount

Put 0 into j
put 0 into k

Repeat While j < S4RepeatCount //S4に入るまでは繰り返し。
    if ImageFound(Image :"NoSignal",WaitFor:1)then
    //NoSignalが表示されれば S4に入ったと判断し処理終了。
        set j = j + S4RepeatCount
    else if ImageFound(Image S4_Buttom,WaitFor:1)then
    // 休止状態が表示されていればクリック。
        Click S4_Buttom
    else if ImageFound(Image:Power_Buttom,WaitFor:1)then
    //スタートメニューの機能(電源ボタン)が表示されていればクリック。
        Click Power_Buttom
    else
        set k = k + 1
        if k < 10
            TypeText WindowsDown,WindowsUp
            //Windowsキー操作によるスタートメニューの表示
        else
    Run "Logon"
    set k = 0
        end if
    End if
End Repeat
```

Wake_On_LAN.script

```
//グローバル変数
global ConnectName , KVM , WOLPC
global WOLBAT
global TRC , i

//カウンタ初期化
Put 0 into WOL_RC

set ConnectName = WOLPC

Repeat While WOL_RC < 10
    Run "ConnectPC"
    Run "ConnectCheck"
    if ImageFound(Image:WOLBAT,WaitFor:1)then //マジックパケットの実行
        click WOLBAT
        TypeText Return
    end if
    Disconnect ConnectName
    set ConnectName = KVM
    Run "ConnectPC"
    Run "ConnectCheck"
    if ImageFound(Image:"NoSignal",WaitFor:1)then
        Disconnect ConnectName
        set ConnectName = WOLPC
    else
        set WOL_RC = 10
    end if
End Repeat
```

Logon.script

```
//グローバル変数
global Logon_Buttom , LockScreen , LogonCheck
global Start_Buttom
global ConnectName
global TRC , i

//ログイン試験繰り返し回数
Put 100 into LRC

Put 0 into j
put 0 into k
put 0 into bl

//サインインに成功しない限り繰り返す。成功すれば処理を抜ける。
Repeat While j < LRC
    if ImageFound(Image:LogonCheck,WaitFor :1)then
        set j = j + LRC
    else if ImageFound(Image :"Logon¥BlueBack_KVM",WaitFor:1)then
    //デスクトップの一部が表示されているか。表示されていれば処理終了している。
    if k < 10
        set k = k + 1
        DisConnect ConnectName
        wait 10
        Run "ConnectPC"
    else
        set i = i + TRC
        set j = i
    end if
    else if ImageFound(Image:Logon_Buttom,WaitFor:1)then
        TypeText Return
    else if ImageFound(Image:Logon_Buttom & "2",WaitFor:1)then
        TypeText Return
    else
        set bl = bl + 1
        if bl < 100
            if bl > 5
                Disconnect ConnectName
                wait 20
                Run "ConnectPC"
            end if
            TypeText Return
        else
            log ("青画面停止")
            set i = i + TRC
            set j = i
        end if
    End if
    wait 1
End Repeat
```

図 6　作成したテストスクリプトの全コード

4.4.　検証結果

OS 不具合の再現テストの自動化前と、作成したテストスクリプトによる自動化後とを比較して、どのぐらいの工数削減効果が得られるかを測定した。

まず、検証対象 OS 不具合の再現手順①～④を人手で 1,000 回繰り返した場合の、テスト工数を測定した。このテスト工数には電源ボタンを押す時間や、OS 起動時間、サインインでのユーザ入力時間なども含めた。

次に、eggPlant による自動化後のテスト工数を測定した。自動化後テスト工数には、自動化テスト担当者が、テスト自動化環境の構築や、再現手順のテストスクリプト化に要する時間も含めた。

測定結果を表 2 に示す。自動化前は 1,000 回の試行に 47 人 H 必要であるのに対し、自動化後は 1,000 回の試行が 21 人 H で済んでいる。今回の OS 不具合調査では出荷判定のために 12,000 回の試行が必要だったので、その合計で比較すると、人手では 553 人 H かかる再現テストが 32 人 H で完了でき、再現テスト工数を約 17 分の 1 に効率化できた。

表 2　テスト自動化前後のテスト工数

指標	単位	自動化前	自動化後
a.環境構築	人H	1	8
b.スクリプト作成	人H	0	12
c.試行(1,000回試行時)	人H	46	1
d.テスト工数合計(=a+b+c)	人H	47	21
e.試行(12,000回試行時)	人H	552	12
f.テスト工数合計(=a+b+e)	人H	553	32

4.5.　出荷判定のための再現テスト

今回の OS 不具合は 4,000 分の 1 という極めて低い頻度で発生していたため、原因調査は困難を極めた。その間にも商品開発は進み、OS、アプリ、デバイスドライバのバージョンが新しくなると、ついには再現できなくなった。このようなケースの場合、筆者は品質保証部門と連携し、再現率、ユーザ影響、回避策/復旧策の有無等を総合的に判断して、出荷判定を行う。今回の OS 不具合については、12,000 回試行で一度も再現しないことを確認の上、原因調査の継続を条件として出荷可と判定した。

この 12,000 回の試行においても、今回作成したテストスクリプトが有効に機能し、出荷判定の期日までに再現テストを終わらせることができた。

再現テストを自動化できるということは、すなわち、再現しないことを証明するための作業も自動化できるということであり、大きな意義があると考える。

5. 考察

5.1. 結果分析の視点

当社の製品開発で GUI テスト自動化ツールを活用したのは今回が初めてであるが、大きな工数削減効果が得られた。テスト自動化を成功に導くために重要な 2 つの視点は「スコープ」と「ROI(投資対効果)」である[4]。そこで、今回の結果をこの 2 つの視点で分析する。

5.2. スコープ視点での結果分析

テスト自動化のスコープとして、GUI テスト自動化ツールを、低頻度不具合の再現テストという同じ操作を何度も繰り返すテストに適用したことが、大きな工数削減効果を生んだと考えている。

5.3. ROI 視点での結果分析

eggPlant の 1 年間のライセンスと、アドックインターナショナルによる有償技術サポートを合わせたツール維持費用を、すべてテスト工数に換算すると、約 520 人 H となる。これは今回の OS 不具合の再現テストで削減できた工数に相当する。すなわち本ツールの維持費用を、わずか 1 件の OS 不具合の再現テストで回収できたことになる。

筆者は大小あわせて年平均 120 件の OS 不具合を処置している。ただし、この中には調査の結果、アプリやデバイスドライバが原因と判明したものも含んでいる。このうち連続再起動試験で検出される低頻度不具合が、年平均 12 件ある。この年平均 12 件の低頻度不具合の再現テストをすべて自動化できれば、本ツールの ROI は大と考えてよいだろう。

5.4. テストスクリプト作成における工夫

OS 不具合再現用のテストスクリプトは今回新規作成したものである。プログラミング経験はあるが、テストスクリプト作成経験のないテスター１名が、eggPlant の正規販売代理店であり、自ら第三者検証も行っているアドックインターナショナルの有償技術サポートを受けることで、実用に耐えるテストスクリプトを完成させることができた。

テストスクリプト作成にかかる費用や、ツールの使い方の教育費用も ROI を悪化させる要因となり得る[1]。しかし今回作成したような数百行規模のテストスクリプトであれば、すべてを自社で作成しようとせず、ツールベンダの有償技術サポートを活用することで費用を最低限に抑えられる。今回のテストスクリプト作成の支援にかかった費用、ツールの使い方教育費用は、すべて 5.3 節のツール維持費用に含まれており、追加でかかった費用はない。

6. 結論

低頻度不具合を再現させる再現テストに、市販の GUI テスト自動化ツールを適用することにより、再現テスト工数の削減に成功した。

今後は、今回作成したテストスクリプトをカスタマイズして、類似の OS 不具合の再現テストへの適用件数を増やす予定である。加えて、不具合の起こりやすい、OS のリカバリ操作をテストスクリプト化するなど、再現テスト自動化のバリエーションを増やすことによって自動化範囲を広げ、さらなる工数削減につなげたい。

参考文献

[1] 菅野 正行, “GUI 操作自動化ツールを用いたテスト効率化手法”, ソフトウェアテストシンポジウム 2016 東京

[2] 田中 桂三,“テスト種類に着目した最適な自動テスト支援ツールの選定方法と実践”, ソフトウェア品質シンポジウム 2012

[3] ASTER テストツール WG, “テストツールまるわかりガイド(入門編)”, 2012

[4] テスト自動化・TABOK 研究会, “なれる！Test Automator”, ソフトウェアテストシンポジウム 2013 東京