一個Java Swing的測試案例產生器

鄭宗其　　林迺衛
國立中正大學　資訊工程學系
E-mail: {chengcc94, naiwei}@cs.ccu.edu.tw

摘要

軟體測試的目標在於確保軟體的品質。圖形使用者介面的程式碼在現代的應用程式中所占的比例非常高，所以圖形使用者介面的測試相當重要。因為圖形使用者介面上的元件數量通常很多，且元件間執行順序的限制非常少，所以需要的測試案例數量非常龐大。本篇論文實作一個針對Java Swing圖形使用者介面函式庫的半自動測試案例產生器，這個工具是奠基於UML的使用案例圖及活動圖。這個工具依據覆蓋標準及UML圖形自動產生測試路徑。這個工具再依據測試路徑與測試者所給予的測試資料自動產生測試程式碼。這個工具除了可以有系統的產生測試程式碼，也可以有系統的產生測試路徑供錄製與播放的測試技術參考。

關鍵詞：圖形使用者介面測試、黑箱測試、測試案例產生器。

1. 概論

圖形使用者介面(Graphic User Interface)的測試與一般軟體的測試方式，有相當大的不同，亦有不同的困難[3, 7, 8]。圖形使用者介面使得事件(event)來驅動執行，因為圖形使用者介面上的元件數量通常很多，且元件間執行順序的限制非常少，所以需要的測試案例數量非常龐大。例如滑鼠在介面上任意一個元件點選，可能都是一個合法的事件。因此，要有系統且有效率的執行圖形使用者介面的測試是非常困難且瑣碎的。

圖形使用者介面的測試一般都使用錄製與播放的技術來進行測試[11]。針對每一個測試案例，測試者先使用錄製工具將圖形使用者介面的操作過程錄製下來，成為腳本，往後實際測試時，則依據腳本使用播放工具重現操作過程進行測試。由於圖形使用者介面的測試案例數量非常龐大，測試者需要自行設計大量的測試案例。

本論文的目的在於依據使用者介面的規格，在圖形使用者介面程式開發之前，盡可能自動化地先產生測試案例，且明確判斷測試案例是否滿足覆蓋標準。因此，本論文開發一個針對Java Swing圖形使用者介面函式庫的半自動測試案例產生器。這個工具使用的規格是奠基於UML的使用案例圖(use case diagrams)及活動圖(activity diagrams)。這個工具依據覆蓋標準(coverage criterion)及UML圖形自動產生測試路徑(test paths)。這個工具再依據測試路徑與測試者所給予的測試資料(test data)自動產生JFCUnit測試程式碼。JFCUnit是奠基於JUnit的自動圖形使用者介面測試工具[5, 6]。

Ostrand et al.開發一個圖形使用者介面的測試工具TDE [11] 。測試者可以使用TDE錄製與重播測試者與被測試程式的互動，將工具紀錄下來的腳本儲存起來，並提供視覺化的介面讓測試者可以自行編輯這些腳本，測試者再依據這些腳本執行測試。

Memon et al.開發一個圖形使用者介面的測試工具PATHS [9] 。工具PATHS使用人工智慧規劃(AI planning)自動產生測試案例，測試者定義成對的起始狀態及目標狀態與規劃運算(planning operators)，PATHS產生一連串的動作到達目標狀態，這些動作串列即構成測試案例。

Memon et al.利用事件流程圖(Event Flow Graph)來產生測試案例[10] 。首先利用事件流程圖定義事件執行順序的限制，再從起始狀態，以圖形走訪的演算法走訪事件流程圖，當到達目標狀態時即完成一個測試案例。

設計者通常先利用使用案例圖來定義使用者與系統互動的需求[1]，使用案例圖的每個節點表示系統要達到的某些功能，透過一個視窗畫面完成，每個視窗畫面上有許多的元件，使用案例圖的邊線用來定義視窗畫面之間的從屬關係。使用者透過驅動視窗畫面上元件的事件來與系統互動，事件之間的執行順序的限制則使用活動圖來定義。

活動圖描述使用案例中一連串的動作[2]，圖中的節點可能是另一張活動圖，或是代表一個元件，用<<元件型態>>的方式命名，該元件的事件用由節點發射出去的邊線表示，用[事件名稱]的方式命名，由起始點走訪至活動圖的結束點，一條路徑代表一個測試案例，走訪活動圖可產生多條路徑，在不同的走訪標準生成的測試案例，可滿足不同的覆蓋標準，有了測試路徑之後，將測試路徑轉換成測試程式，即可利用測試工具來測試。

本論文其餘章節簡述如下：第二節說明測試案例產生器的系統架構。第三節簡介如何使用UML的使用案例圖與活動圖描述圖形使用者介面的規格，並說明如何讀取UML圖形的檔案。第四節說明如何利用UML圖形產生符合不同覆蓋標準的測試路徑。第五節簡介JFCUnit工具，並說明如何利用測試路徑及測試者給予的測試資料產生JFCUnit測試程式。最後，第六節綜合本論文的結論與未來可以改進的方向。

2. 系統架構

測試案例產生器主要分成以下三個部份：UML圖形閱讀器 (UML Diagram Reader)、測試路徑產生器(Test Path Generator) 、測試程式產生器(Test Program Generator) ，如圖一所示。UML圖形閱讀

器讀取UML圖，將UML儲存的資料格式轉換成自行定義的資料結構。測試路徑產生器走訪分析UML圖形，根據不同的覆蓋標準產生測試路徑。測試程式產生器依照測試路徑產生器得到的測試路徑，生成測試案例的程式碼。

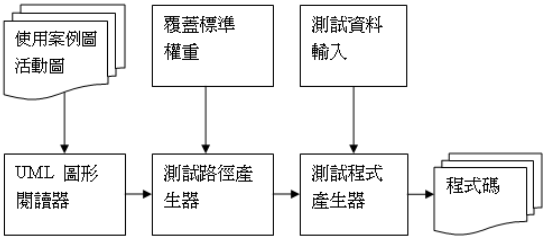

圖1. 系統架構

3. UML 圖形閱讀器

本節說明UML圖形閱讀器，我們先介紹如何利用使用案例圖及活動圖定義圖形使用者介面，我們接著說明UML圖形閱讀器的運作原理。

本篇論文以使用者(actor)節點、使用案例節點與從屬關係來描述系統的功能需求，使用者節點可以代表不同的使用者，使用案例節點表示使用者與系統互動的介面及功能。每個使用案例節點表示一個使用者畫面，邊線表示使用者畫面之間的從屬關係。

一個視窗畫面裡面的元件與每一個元件的事件，由活動圖來詳加敘述。活動圖中的開始與結束節點代表視窗畫面的開始與結束，一個動作節點表示視窗畫面上的元件，或者可能是另一張活動圖。動作節點的名稱若是<<型態>>，表示這個動作節點是一個 Java Swing 的元件，例如<<JTextField>>userName 可知視窗當中有一個JTextField，名稱為userName。當動作節點沒有<<型態>>就代表這個動作節點是由另一張的活動圖來描述，對應使用案例圖會是一個使用案例節點包含另一個使用案例節點。代表圖形使用者介面會是一個視窗畫面能夠開啟另一個視窗畫面，兩個視窗畫面由兩張不同的活動圖來描述，每一個視窗畫面上的元件詳細資訊，都紀錄在各張活動圖當中。視窗畫面中元件進行的事件以邊(活動流)來表示，例如 JTextField 元件可以由使用者鍵入資料，JButton 元件可以用滑鼠按下按鈕，不同元件有不同的動作。各種事件表示在節點發射出去的邊上面，加上[事件名稱]這樣的形式來代表從這個節點會進行的事件，在本篇論文的測試程式產生器來定義這些事件，例如<<JTextField>>節點有[keyInData]的事件等等。

本論文利用 Eclispe 開發環境的 Omondo EclipseUML工具來繪製UML圖形[4]。UML圖形檔案是使用XML的格式儲存[12]。UML圖形閱讀器是先以DOMParser讀取UML圖檔案，再將UML圖中的資訊轉換成自行定義的類別資料結構，UML圖形閱讀器與測試路徑產生器以自訂資料結構來傳遞溝通資料。

使用案例圖儲存檔案的副檔名是uud，活動圖副檔名是uad，本篇論文使用 Apache Xerces DOMParser來讀取UML圖的檔案，DOMParser會回傳DOM結構樹，資訊可從DOM結構樹中取得。

Omondo EclipseUML工具畫的圖是用XML資料結構來儲存，最外層元素表示圖的各種屬性，下一層元素是圖裡面的節點，下兩層的元素是表示從上一層元素也就是節點發射出去的邊。DOMParser回傳DOM結構樹的結構是圖的子孫為節點，節點的子孫是這個節點發射出去的邊。

為了走訪路徑方便，需要可隨時存取的資料結構，我們定義了 OriginalDiagram、OriginalDiagramEdges別、OriginalDiagramNodes類別。OriginalDiagram的欄位存放圖的所有資訊，Hashtable型態的欄位存放圖裡面所有的節點與邊，OriginalDiagramNodes類別中放置節點的各類資訊，除了畫面上節點大小、位置等等較不重要的資料外。邊的類別態是OriginalDiagramEdges，是在OriginalDiagram 的 Hashtable 欄位，OriginalDiagramEdges一樣會紀錄所有的邊資訊。

UML圖形閱讀器會將UML圖最終轉存為Diagram類別、DiagramEdges、DiagramNodes類別，Diagram存放走訪所需要的圖資訊，DiagramEdges類別存放邊的資訊、DiagramNodes類別存放節點的資訊。

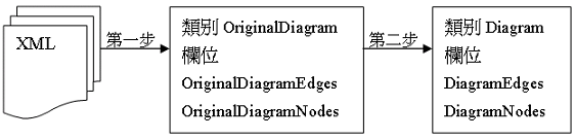

圖2. 資料結構轉換流程

轉換資料結構流程如圖2所示：第一個動作會將XML的資料對應存到OriginalDiagram類別當中，完全不會作任何處理，單純將資料複製過去，只有資料結構上的差異。第二個動作將OriginalDiagram類別中的資料轉換到另一個自訂的資料結構Diagram，Diagram類別有DiagramEdges、DiagramNodes資料結構，分別存放節點、邊的資料，轉換OriginalDiagram至Diagram的過程會將圖中的所有邊與節點都一一編號，每一個邊與節點都有不重複的編號，作為索引以供識別，DiagramEdges、DiagramNodes的資料欄位只有保留往後運作所需要的資料，沒有用到的資料在轉換過程會忽略，假使往後有需要更多的資料，在轉換過程新增進去即可。

這樣的設計將UML圖形閱讀器、測試路徑產生器分開來，彼此是用Diagram的資料結構溝通傳遞，對於UML圖的資料處理與轉換工作，完全放在UML圖形閱讀器來做，當Omondo EclipseUML工具版本改變或其資料結構改變，只需要更動UML圖形

閱讀器即可，不會影響到測試路徑產生器的運作。

4. 測試路徑產生器

本節說明測試路徑產生器的路徑產生方式，我們先介紹測試路徑的覆蓋標準，再介紹測試路徑產生的方式。

4.1. 覆蓋標準

本篇論文對於路徑走訪的覆蓋標準有All Nodes、All Edges、All Paths三種，路徑選擇的優先順序有依照事件權重高低與隨機選取兩種方式。

All Nodes需滿足圖中每一個節點至少走訪過一次，走訪路徑時盡量選擇尚未走訪過的節點，直到所有節點都有走訪過。All Edges需滿足圖中每一個邊至少走過一次，先選擇沒有走訪過的邊，直到所有的邊都走訪過才完成走訪。All Paths需滿足所有可能的路徑都走訪過。

兩種選擇路徑走訪優先順序的方式，是針對不同種類的邊，圖形中有兩種邊，一種代表元件的事件，另一種是單純的事件流。對於節點是元件時，而且這個元件有多個不同的事件，可以給予每一個事件不同的權重高低。決定走訪路徑時，會以較高權重的事件優先走訪。若節點不是代表元件節點，並且有多個發射出去的邊，可以選擇隨機選取的方式來選擇走訪的邊。不使用這兩種模式的話，系統預設是依照畫圖時的加入順序來走訪。

4.2. 測試路徑選擇

測試路徑產生器走訪路徑的方式，從開始節點走至結束節點，表示完成一條走訪路徑，走訪時可能會有迴圈與無迴圈的狀況，需運用不同的方式走訪。本篇論文對於無迴圈的路徑採用深度優先搜尋，有迴圈的走訪情況則加上其他的方法來處理。

對於無迴圈的走訪方式，從圖中的起始點開始走訪，先走訪節點的子節點，不斷往下遞迴走訪，當走訪到終止節點時就得到一條路徑，再回溯上一層走訪另一個子節點，往下走訪至終止節點就得到另一條路徑，當遞迴走訪滿足所選擇的覆蓋標準後，即完成走訪路徑的工作。

當圖中有迴圈存在時，我們處理的步驟如下：

1. 找出圖中的所有迴圈。
2. 以自行定義的複合節點 loopNode 取代迴圈，便能以無迴圈的方式走訪整張圖。
3. 接著模擬走訪每一個迴圈節點集合可能的狀況。
4. 最後將模擬走訪迴圈的路徑取代複合節點 loopNode，就能得到完整的路徑。

要找出迴圈有下列步驟：(1)首先要找出各節點的Dominator，(2)再利用節點的Dominator找出各節點構成的backedge，便可以用backedge找出形成迴圈的節點集合。

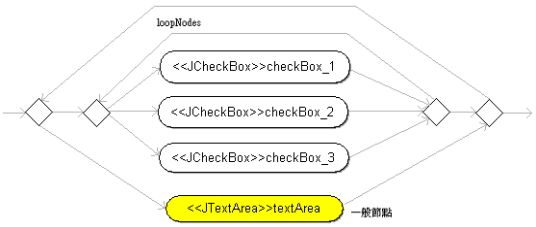

圖3. 複合節點

有了每一個迴圈的節點集合，我們可以將圖中所有的迴圈替換成自行定義的複合節點loopNode，就能將有迴圈路徑變成沒有迴圈的路徑，再以無迴圈方式的走訪整張圖，就能得到走訪路徑的結果。

模擬走訪迴圈的所有可能路徑，首先走訪這個迴圈節點集合的所有路徑，再將迴圈中所有路徑作組合，就可以得到走訪這個迴圈的所有可能路徑。走訪迴圈中的一條路徑是從迴圈的進入點走至迴圈的離開點，走訪完所有可能的路徑並記錄下來。

模擬走訪迴圈的方式，就像從箱中取出不重複的球，將迴圈視為一個箱子，每一條路徑視為箱中的有編號的球，從取出一個、兩個，直到所有球都取出，也就是每一種可能都會走訪，取出的路徑組合成走訪這個迴圈的路徑。

最後以組合完成的迴圈走訪路徑取代自行定義的複合節點loopNode，就能夠得到圖中走訪的所有路徑。

5. 測試程式產生器

本節說明測試程式產生器的運作原理，我們先介紹JUnit與JFCUnit，我們接著介紹測試程式產生的方式，最後我們針對本篇論文實作的元件，解說其測試資料的輸入畫面。

5.1. JUnit 與 JFCUnit

一般執行測試是使用JUnit[10]，JUnit提供一個可以自動且重複執行測試案例的架構，使用JUnit實施單元測試基本上會有三個過程，建立待測試的物件，呼叫物件的方法，確認結果是否正確。

JFCUnit[11]是JUnit的一個擴充，用來測試以Java Swing開發的應用程式，使用JFCUnit的方法非常類似JUnit。一般應用程式可能是由幾個部份構成，最外層的圖形使用者介面、中間的流程運作與程式邏輯運算、再連結資料庫，最後呈現在最外層的圖形使用者介面上，在測試的時候除了最外層的圖形使用者介面，都能輕易使用JUnit來做單元測試，亦能得到相當好的結果。

但是我們發現在整個程式測試過程中，單元測試是針對方法的呼叫來作測試，在真正執行程式時，通常會先操作圖形使用者介面之後才會呼叫方法，所以對於測試圖形使用者介面無法直接套用，而JFCUnit是測試圖形使用者介面程式的工具，待測試程式是Java Swing建立的圖形使用者介面，可以讓我們藉由模擬使用者的方式來測試圖形使用者

介面，如同使用者真正與圖形使用者介面互動一樣，且JFCUnit也提供許多API讓測試者使用。步驟(1)開啟視窗元件，(2)尋找視窗上要測試的元件的位置，(3)驅動已被定位的元件，例如按下按鈕、輸入文字等。

5.2. 測試程式產生

測試程式產生器運作需要有測試資料輸入與測試路徑，測試路徑由測試路徑產生器，測試資料需要測試者來輸入，測試者可藉由本系統的介面針對測試類別或測試方法，給予不同的測試資料。測試程式產生器產出的結果會有固定的格式，一個視窗畫面對應一個測試類別，一條測試路徑代表一個測試方法，測試類別裡面有許多的測試方法，也就是一個視窗畫面可能有很多測試路徑。

由測試路徑產生器，產生不同覆蓋標準的路徑，路徑上面的節點是由PathNodes類別構成，PathNodes的欄位紀錄節點的編號與名稱、事件的名稱、測試資料輸入的畫面等等多種資料。

測試者需給予的測試資料，分成測試類別需要的資料與測試方法需要的資料兩種。

測試類別需要的資料用於建立測試環境，就是每一個方法共用的資料，需要輸入的測試資料，分成package、field與setUp、tearDown四個部份，類別檔案命名方式是以圖的名稱加上XXXTest.java。

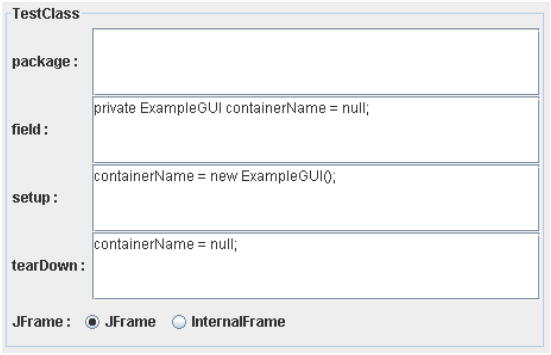

圖4. 類別測試資料輸入畫面

測試者需要在package部份填入被測試程式的套件名稱，這樣測試程式才能找到被測試的程式，格式為”import xxx.xxx.xxx;”。

在field部份宣告被測試程式，格式為”private ExampleGUI containerName = null;”，”ExampleGUI”是被測試程式的類別名稱。

setUp部份需要填寫被測試程式的建構程式碼，”containerName = new ExampleGUI();”。

tearDown部份填寫銷毀setUp建構的被測試程式物件程式碼，”containerName = null;”，還有選擇畫面視窗是JFrame或InternalFrame。產生測試程式碼會生成固定的類別框架，”containerName”是被測試程式的變數名稱，變數名稱是本系統定義的不能改變。

```
import java.awt.*;
import javax.swing.*;
import java.util.List;
import junit.framework.Test;
import junit.framework.TestSuite;
import junit.extensions.jfcunit.*;
import junit.extensions.jfcunit.eventdata.*;
import junit.extensions.jfcunit.finder.*;                    Package 部份
public class ExamplesTest extends JFCTestCase {
        private ExampleGUI containerName = null;             Field 部份
    public ExamplesTest(String name) {
        super(name);
    }
    public void setUp() throws Exception {
        super.setUp();
        this.setHelper(new JFCTestHelper());
        containerName = new ExampleGUI();                    setup 部份
        containerName.setVisible(true);
        containerName.pack();
    }
    public void tearDown() throws Exception {
        containerName = null;
        TestHelper.cleanUp(this);                            teardown 部份
        super.tearDown();
    }
    public void testPath_0() throws Exception {              測試函式部份
```

圖5. 類別測試程式碼

測試方法需要的資料是每一條路徑本身用到的，路徑上的每一個元件所需要的初始值、輸入值、預期值三種。測試者針對路徑上的每一個元件輸入元件所需要的測試資料。本系統的輸入介面畫面上，每一個元件基本上有初始值的欄位，元件動作所需的值或驅動動作條件欄位，元件動作後預期的結果。本系統會依據路徑自動產生測試的測試方法，多少路徑就會有多少測試方法，路徑上節點是一個元件與元件的動作，程式會把確定元件位置的程式碼、比對元件狀態的程式碼、驅動元件的程式碼、確認動作使是否符合預期的程式碼，這些程式碼加入測試方法中。

本系統除了只針對一條路徑上的元件輸入測試資料，測試者亦可以針對整個圖所有的路徑來輸入測試資料，本系統列出測試類別中所有路徑用到的基本元件全部種類，讓測試者對元件種類給予測試資料，本系統會將測試類別中的所有路徑上，每個元件依照元件種類給予測試資料，節省測試者輸入測試資料的時間。

5.3. 基本元件

本篇論文使用的Java Swing基本元件總共十一種，產生JFCUnit可執行的程式碼，對元件作三個動作，(1)尋找元件在視窗中的位置，(2)找到元件位置便能對元件作驅動，(3)檢查元件動作後的狀態是否與預期的相同。

表1. 各元件事件

元件名稱	事件名稱
JTextField	keyInData(鍵入資料)
JTextArea	keyInData(鍵入資料)
JCheckBox	isChecked(是否選取)
JRadioButton	isSelected(是否選取)
JList	selectedIndex(以索引選取項目)與 selectedName(以名稱選取項目)

JComboBox	selectedIndex(以索引選取項目)與 selectedName(以名稱選取項目)
JTable	selectRowIndex(選取 Row 索引的位置)與 selectRowAndColumnIndex(選取 Row 索引與 Column 索引的交會位置)
JTabbedPane	selectedIndex(以索引選取頁面)與 selectedNameAndIndex(以名稱與索引選取頁面)
JButton	click(按下按鈕)
JDialog	isDisplayed(呈現對話視窗)
JFileChooser	selectFile(選取檔案)

各元件需輸入的測試資料如下：

JTextField與JTextArea需要填入的資料有(1)初始值，用來比對元件開啟呈現時，JTextField裡面的值有無符合初始值，(2)填入值，元件欲鍵入的值，(3)預計值，檢查元件的結果是否與預計值相同，畫面上輸入資料是利用JTextField元件。

JCheckBox與JRadioButton在輸入測試資料畫面上以JCheckBox來輸入(1)初始是否選取(2)元件選取與否(3)檢驗是否選取。

JList與JComboBox輸入測試資料的畫面需要(1)初始選取的項目，(2)元件選取的項目，(3)預期選取的項目，有兩種事件分別是以索引與項目名稱作為選取的條件。selectedIndex是利用索引來找尋項目，selectedName利用項目名稱來尋找。

JTable事件有兩種，第一種是以Row的索引來選擇資料，第二種是Row的索引與Column索引來選擇，在填入測試資料時，要有(1)初始選取的資料位置(2)元件選取的資料位置(3)預計選取到的資料位置。

JTabbedPane需要(1)初始選取的Tab頁面(2)選取的Tab頁面(3)預計選取的Tab頁面，一種是利用索引，另一種是索引加上Tab名稱來選取。

JButton只有一個事件，就是按下按鈕，不需要填入任何測試資料。

JDialog要輸入呈現的對話視窗的title名稱。程式利用title名稱來找對話視窗，確認對話視窗是否開啟。

JFileChooser要有(1)欲選擇的檔案路徑，(2)給予檔案名稱，(3)檢查開啟的檔案名稱。程式檢查選擇檔案路徑開啟的檔案，檔案的名稱是否正確。

6. 結論與未來展望

在本論文中，我們開發一個奠基於使用案例圖與活動圖的半自動測試案例產生器，在圖形使用者介面完成之前產生測試案例自動產生測試路徑，手動輸入測試路徑的輸入和預期輸出資料，自動產成可執行的JFCUnit測試程式。

我們利用UML的使用案例圖與活動圖，定義圖形使用者介面上的十七種基本元件，各基本元件的事件互動的組合，以UML來產生測試案例作測試。讓開發圖形使用者介面可以運用測試先行的開發流程，先產生測試案例，再來開發程式與測試。經由使用案例圖與活動圖自動產生不同覆蓋標準的測試案例。測試案例經由測試人員給定測試資料，系統會根據測試資料自動產生測試程式碼。數量龐大的測試案例，系統自動產生程式碼，不需要重複撰寫相同的程式碼。半自動產生大量的測試案例程式碼，大幅減少人工作為，節省大量的時間。

本篇論文使用的編輯UML的工具是Omondo EclipseUML，產生的測試程式碼可經由JFCUnit執行，讀入與產生的格式是單一的，未來可以針對不同的UML編輯工具都能夠讀取，並且產生不同的測試工具的程式碼。實作可以產生測試程式的Java Swing元件的個數不夠多，未來可以針對更多的元件來自動產生測試程式。我們提出的測試工具是缺少元件靜態屬性的分析，往後能夠加入靜態屬性分析讓工具更加完整。除了自動產生測試案例與測試程式外，測試資料的自動產生我們尚未達成，為了能達到完全自動化的測試，需要能增加測試資料的自動產生，來實現完整的測試自動化。

7. 參考文獻

[1.] J. Almendros-Jimenez and L. Iribarne, “Designing GUI Components from UML Use Cases,” In *Proceedings of 12th IEEE International Conference on the Engineering of Computer-Based Systems*, pp. 210–217, April 2005.

[2.] J. Almendros-Jimenez and L. Iribarne, “Describing Use Cases with Activity Charts,” In *Proceedings of Metainformatics Symposium*, Salzburg, Austria, Oct. 2004.

[3.] T. Daboczi, I. Kollar, G. Simon, and T. Megyeri, “How to Test Graphical User Interfaces,” *Instrumentation & Measurement Magazine*, IEEE, Vol. 6, pp. 27–33, Sept. 2003.

[4.] Eclipse. http://www.eclipse.org/.

[5.] JFCUnit. http://jfcunit.sourceforge.net/.

[6.] JUnit. http://www.junit.org/.

[7.] P. Gerrard, “Testing GUI Applications,” In *Proceedings of EuroSTAR’97*, November 1997.

[8.] A. M. Memon, “GUI Testing: Pitfalls and Process,” *Computer*, Vol. 35, No. 8, pp. 87–88, 2002.

[9.] A. M. Memon, *A Comprehensive Framework for Testing Graphical User Interfaces*. Ph.D. Thesis, 2001.

[10.] A. M. Memon, M. E. Pollack, and M. L. Soffa, “Hierarchical GUI Test Case Generation Using Automated Planning,” *IEEE Transactions on Software Engineering*, Vol. 27, No. 2, pp. 144–155, 2001.

[11.] T. Ostrand, A. Anodide, H. Foster, and T.

Goradia, “A Visual Test Development Environment for GUI Systems,” *SIGSOFT Software Engineering Notes*, Vol. 23, No. 2, pp. 82–92, 1998.

[12.] XML. http://www.w3.org/XML/.