Reg4J: 一個支援敏捷式程序的前端迴歸測試選擇工具[1]

Reg4J: A Front-End Regression Test Selection Tool for Agile Process

謝金雲　李文棋　陳建村　鄭有進
國立臺北科技大學資訊工程系
Email: {hsieh, s2598012, s1669021, yccheng}@ntut.edu.tw

摘　要

在敏捷式軟體開發流程中，程式開發人員必須頻繁地執行迴歸測試以支援軟體重構與快速開發。隨著軟體系統的成長，測試案例隨之增多，執行測試案例之時間亦增加。此現象可能導致：(1) 程式開發人員無法忍受等待執行測試的時間，因此選擇性避免執行迴歸測試，而違反敏捷開發流程的精神；(2) 程式開發人員忍受冗長的執行測試時間，因而導致生產力降低。為解決此問題，本論文以 Java 語言與 Eclipse 技術實作一個前端迴歸測試案例選擇工具，藉由自動挑選出適當的測試案例，有效地減少前端迴歸測試等待時間，以支援敏捷軟體開發流程。

關鍵詞：前端迴歸測試、敏捷式程序、Eclipse

一、前言

在軟體開發過程中，迴歸測試 (regression test) 可用以確保程式開發人員在修改軟體之後並未破壞該系統原有的行為，也就是沒有產生新的錯誤。近年來廣為流行的敏捷式軟體開發程序（Agile Software Process）[1, 4]，例如 Extreme Programming (XP) [2] 則更進一步鼓勵程式開發人員持續且頻繁地執行迴歸測試，以期儘早發現因為撰寫、修改、或重構（refactoring）[6] 程式所造成的問題。

基本上迴歸測試的執行模式可分為下列兩種:

- **後端迴歸測試**：又稱為批次迴歸測試（back-end/batch regression test）。採用此模式的組織通常會要求所有程式開發人員在每日固定時間，例如下午五點下班之前，將所有的程式交付到集中式的原始碼控管系統中。接著自動化建構系統（build system）便會開始執行一系列的建構活動，其中包含編譯與執行迴歸測試等。目前較著名的實務作法有微軟公司所提倡的每日建構（daily build）[11]，或是 XP 所鼓吹的持續整合（continuous integration）[2]。
- **前端迴歸測試**：近年來由於 JUnit 單元測試工具的成功 [9, 10]以及廣泛地被各種主流的整合開發環境所支援（例如 Eclipse、NetBeans 與 Visual Studio.NET），因此程式設計人員逐漸改變開發程式與執行測試的習慣。也就是說，程式開發人員自行撰寫單元測試程式並在其個人開發環境下執行單元測試。此外，由於 JUnit framework 具擴充性與開放式的設計，因此在搭配諸如 mock object 或 stub 等技術的應用之下，我們亦可使用 JUnit 來驅動整合測試與功能測試 [9]。因此，傳統後端迴歸測試的若干工作便可利用程式開發人員的電腦來執行。在本文中我們將程式開發人員在其個人開發環境下所進行的迴歸測試稱之為**前端迴歸測試**（front-end regression test），用以和傳統的後端迴歸測試加以區別。

隨著軟體系統功能逐步地發展，其測試案例的數量與種類亦隨之增多，且其複雜度亦提高。因此，執行前端迴歸測試所需的時間也逐漸增加，因而導致程式開發人員等待前端迴歸測試完成的時間變長；我們將此時間稱為**前端迴歸測試等待時間**。若在每次執行前端迴歸測試時，只選擇部份相關的測試案例來執行，則可有效地降低每次執行前端迴歸測試的等待時間，並增進軟體開發的生產力及提昇軟體品質。

傳統後端迴歸測試採用以原始程式碼為基礎的選擇技術 (source code based regression test selection) [8, 12]，其基本精神如下：首先探討測試案例涵蓋原始程式的程度，其次計算修改後的程式與原始程式的異動範圍，最後選擇出可涵蓋該異動範圍的測試案例並加以執行。此方法可應用在後端迴歸測試的執行環境中，例如在整合機器（integration machine）[2] 上執行整合測試，但卻不適合應用於前端迴歸測試。其主要原因為：

- **Lack of Controllability**: 在 IDE 中，程式開發人員應該能夠控制測試案例的執行順序、數量以及每次的執行時間，並且可以隨時終止迴歸測試的執行。此外，迴歸測試選擇技術應該要避免某些特定的測試案例永遠或鮮少被執行的現象(test case starvation)。
- **Lack of Context**: 在採用測試驅動開發（test-driven development）[3] 的情況之下，程式開發人員經常需要來回修改原始程式與測試程式。此時若採用傳統的 source code based regression test selection 技術，將由於並未考慮程式開發的實況（context），而使得測試案例選擇演法的設計與實作變的更加複

[1] 發表於第三屆台灣軟體工程研討會，June 8-9, 2007.

雜，可能會選擇出許多沒有必要的測試案例而增加測試等待時間，造成生產力降低。

- **Inversion of Coverage**: 與 code-based regression-test-selection 技術相反，在軟體開發過程中，程式開發人員經常是因為修改測試程式或是修改測試資料而執行前端迴歸測試，並非因為修改原始程式而執行前端迴歸測試。在此條件之下，我們所關注的是，如何一併選擇出與本次所修改之測試案例相關的其他測試案例。也就是說儘可能找出所有被此次修改之測試案例所影響的其他測試案例，而非傳統的測試案例涵蓋原始測試的程度。

對於一個實行 XP 或其他 agile software process 的軟體開發團隊而言，為了支援重構（refactoring）與測試驅動開發（test-driven development, TDD），能夠在開發環境中自動、快速且有效地頻繁執行測試案例將是極為重要的一項先決條件[3, 6]。因此，Saff 提出了持續測試（continuous testing）的概念 [13, 14]。藉由利用 CPU 閒置的時間在背景模式下持續執行迴歸測試，則可降低批次執行迴歸測試的等待時間。然而由於 Saff 在研究中所使用的測試案例其執行時間均非常之短，因此其持續測試之作法與批次測試之比較效益並非十分明顯。此外，Saff 的作法持續執行全部的測試案例，因此當特定測試案例需要耗費大量計算資源時，將會顯著地降低電腦反應時間，影響程式開發人員之工作。

本論文考量前端迴歸測試選擇模式之特殊性與實用性，提出一個前端迴歸測試選擇參考模型 (front-end regression test selection reference model)。此模型將同時討論以原始程式以及測試程式作為選擇測試案例的參考依據，並可讓程式開發人員依據測試案例與原始程式的相依性、原始程式本身的相依性、測試案例執行時間、測試案例執行結果、最大可忍受之迴歸測試等待時間、測試案例執行頻率、測試案例或原始程式修改次數與時間等條件來選擇測試案例。程式開發人員並可自行設定前端迴歸測試選擇方法，達到客製化的目的。此外，我們將以 Java 語言實做一個可支援上述模型的前端迴歸測試選擇工具－Reg4J。為了驗證本模型之有效性並增加其實用性，我們將 Reg4J 設計為 Eclipse [5] 整合開發工具的一個外掛程式 (plug-ins)。Eclipse 是一個廣為使用的開放原始碼 Java IDE 環境與工具整合平台，藉由將 Reg4J 整合進 Eclipse 的平台中，將使得前端程式開發人員的程式撰寫與測試流程更加緊密的結合在一起，以期提高其生產力與軟體品質。

本文其他章節組織如下。第二節提出一個支援前端迴歸測試選擇的模型。第三節說明 Reg4J 所支援的測試案例選擇策略。第四節說明系統架構。第五節以實際案例說明 Reg4J 之使用。最後為結論與未來工作。

二、前端迴歸測試選擇工具參考模型

圖 1 為我們所提出前端迴歸測試參考模型，其重要概念敘述如下：

- Developer: 程式開發人員，負責程式開發。Developer 會編輯並修改某個 Workspace 中的若干個 Projects。
- Workspace: 工作區。Workspace 是用來管理 Project 的物件。一個軟體開發環境中只會存在一份 Workspace，但一個 Workspace 可包含多個 Projects。
- Project: (軟體)專案，其中包含原始程式碼 (source code)與測試程式 (test code)。
- Project History: 專案歷史紀錄。Project History 包含原始程式碼與測試程式的修改紀錄（例如，修改日期，異動程式名稱等），以及歷次執行回歸測試的歷史紀錄（例如，測試是否成功，測試執行時間等）。Project History 中的資料會提供給 Test Case Selector 作為選擇測試案例之參考。
- Resource Monitor: 資源監督器。Resource Monitor 負責監督 Developer 對於所有 Projects 中的原始程式與測試程式之新增、修改與刪除等操作，並將異動資料紀錄於 Project History。
- Test Case Selector: 測試案例選擇器。Test Case Selector 會參考 Project History 的資料，依據不同的 Test Selection Strategy 至目前工作中的 Project 選擇測試案例，並將選擇後的測試案例交給 Test Runner 執行。
- Test Selection Strategy: 測試選擇策略。Test Selection Strategy 代表多種選擇測試案例的策略（演算法）。Test Case Selector 可依據 Developer 的設定（例如，選擇上次執行失敗的所有測試案例），或是程式開發的情境（例如，目前 Developer 正在修改原始程式或是測試程式）使用適當的策略來選擇測試案例。
- Test Runner: 測試執行器。Test Runner 接受由 Test Case Selector 所選擇的測試案例並加以執行。
- Test Listener: 測試監聽器。Test Listener 向 Test Runner 註冊要求得到測試執行之結果回報，並依據 Test Runner 所回報之資料，更新 Project History 並產生 Test Result。

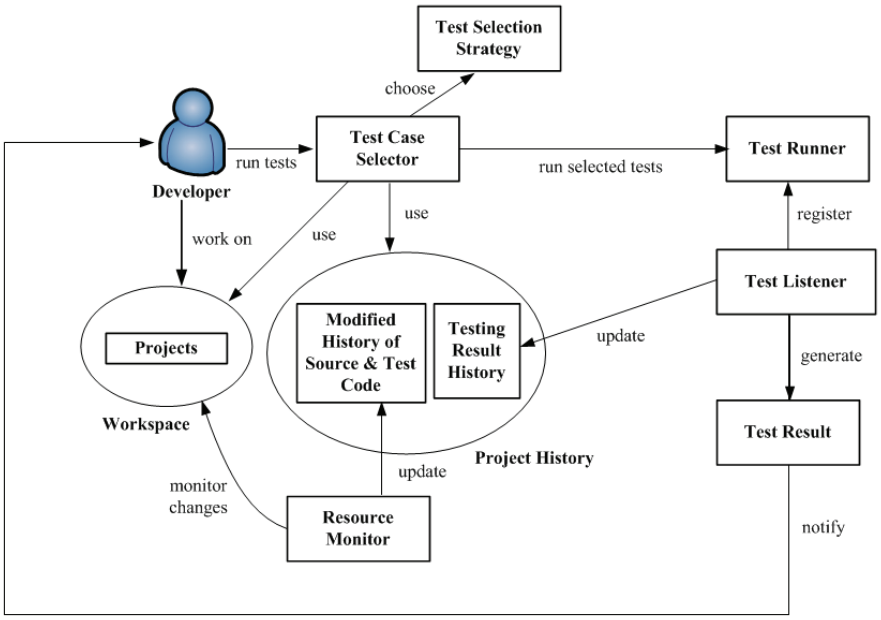

圖 1 前端迴歸測試選擇模型

- Test Result: 測試結果。Test Result 用來表示本次迴歸測試之結果,該結果會回饋給 Developer 作為修定原始程式或測試程式之參考。

三、 測試案例選擇策略

我們將測試選擇策略（test selection strategy）的條件分成三大類，分別為 Test Selection（測試選擇）、Test Constraint （測試限制）與 Test Ordering（測試順序）。也就是說圖 1 上方的 test selection strategy 將由這三大類的選擇條件來實作完成。圖 2 為測試案例選擇策略的概念圖，其中 Project History 與 Test Case Selector 的用途請參考第 2 節圖 1 的說明。在此我們專注於其餘三個概念：

- Test Selection：決定要執行哪些測試案例的選擇方法；例如，選擇上次執行測試之後修改過的測試案例（ModificatonDate）、目前游標位置所在的測試案例（CurrentMethod）、以及在過去測試中錯誤次數累積已達若干次以上的測試案例（FailCount）等。表 1 列出所有 Test Selection 方法。圖 2 中的 Test Selection 有一個自我參考（self link），也就是說 Test Selection 可以透過 And、Or、Not 關係運算來產生一個更大的 Test Selection。我們將 And、Or、Not 關係運算稱之為 TestCombination。Test Selection 與 Test Combination 可用 Composite pattern [7] 來表示。
- Test Constraint：選擇測試案例的限制條件，用以限制 Test Selection 的參數。例如，我們以 FailCount（測試失敗次數）當作 Test Selection 方法，然後限制只選擇出失敗次數大於 3 次的測試案例，其表示方法為 FailCount(Greater(3))。表 2 列出所有的 Test Constraint 篩選條件。
- Test Ordering：當使用 Test Selection 與 Test Constraint 篩選出測試案例之後，我們可進一步地安排這些測試案例的執行順序。例如，先測試最常出現錯誤的測試案例（Frequently Failed First）或是先測試最近有更動的測試案例（Recently Changed First）。設計此功能的目的在於讓使用者能夠決定不同性質測試案例的執行優先順序，以便於可在錯誤發生時決定是否提前終止迴歸測試的執行，以縮短前端迴歸測試等待時間並儘早修復程式錯誤。表 3 為所有 Test Ordering 之列表。

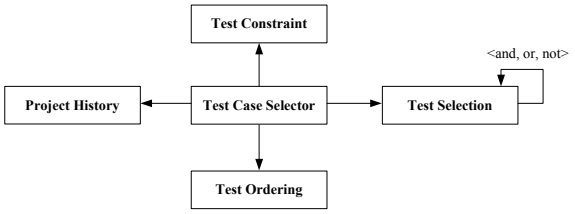

圖 2 測試案例選擇策略概念圖

接下來我們以一個範例說明 Test Selection 與 Test Combination 的組合用法。假使我們想要選擇出符合下列條件的測試案例：

(1) 測試案例執行失敗次數超過 3 次，且
(2) 執行日期在 2005 年 10 月 12 日之後，或
(3) 測試案例的檔案名稱以 Copy 開頭，或
(4) 測試案例之執行時間小於 1 秒（1000 毫

秒）

於是我們可如表 3 中的四種選擇條件，以找出前述(1)、(2)選擇條件的測試案例交集，以及與(3)、(4)選擇條件的聯集，即為我們所要執行的測試案例。圖 4 為 Test Selection 與 Test Combination 採用 Composite pattern 所表示的樹狀結構圖，反應出上述選擇條件的組合關係。

上述選擇方法可讓使用者自行建立並儲存為個人化的設定檔（profile），以達到重複使用的目的。使用者可透過圖形使用者介面自行勾選不同的組合條件，並選擇所需的參數。選擇完畢之後，使用者將可透過預覽的功能，了解其依據所設定的測試案例選擇方法與參數將會選出哪些符合條件的測試案例。預覽功能將可協助使用者在真正執行測試案例選擇方法之前評估其所設定方法之正確性。

表 1：Test Selection 方法列表

篩選方法	說明
Included Pattern	判斷測試的名稱中是否有某字串來做篩選
Fail Count	以測試的失敗次數作篩選
Last Status	以測試最後一次的執行結果作篩選
Run Count	以測試執行過的測試作篩選
Modification Count	以測試的被異動次數作篩選
Modification Date	以測試的被異動日期作篩選
Run Date	以測試的執行日期作篩選
Execution Time	以測試的執行時間作篩選
Success Count	以測試的成功次數作篩選
Excluded Pattern	判斷測試的名稱中是否沒有某字串來作篩選
Top Selection	以所有測試的最前面的幾個作篩選
Random Selection	隨機選擇測試
Dependency Level	依據原始程式之間的相依性階層作篩選
Current File	目前所開啟的檔案
Current Method	目前所開啟的檔案的方法

四、 系統架構

Eclipse 提供完善的介面、許多的延伸點和完整的 API，讓程式開發人員可以根據自己的需求，開發所需要的外掛程式（plug-in），整合至 Eclipse 中。Reg4J 即為建構在 Eclipse 上的外掛程式。圖 4 為 Reg4J 的系統概念圖，綠色粗線框起來的部份，即為整個 Eclipse 的內容。紅色粗線框起來的內容，即 Reg4J 的四個主要的模組。這些模組實作圖 1 中，前端迴歸測試選擇模型的主要功能。

表 2：Test Constraint 方法列表

篩選條件	說明
Greater	表示大於之關係
Lesser	表示小於之關係
ValueEqual	表示等於之關係
GreaterOrEqual	表示大於或等於之關係
LesserOrEqual	表示小於或等於之關係
Between	表示介於…之間之關係
StringStartsWith	表示以某字串開頭之關係
StringEndsWith	表示以某字串結尾之關係
StringContains	表示包含某字串之關係

表 3：Test Ordering 方法列表

排序方法	說明
Frequently Failed First	最常出錯誤的測試最優先執行
Recently Failed First	最近有出現錯誤的測試先執行
Recently Changed First	最近有更動過的測試先執行
Shortest Running Time First	測試執行時間最短的測試先執行
Random Order	隨機執行測試
Longest Running Time First	測試執行時間最長的測試先執行

表 4：測試案例組合範例

Test Selection	Test Relation
FailCount	Greater(3)
RunDate	Greater(“2005/10/12”)
IncludedPattern	StringStartsWith(“Copy”)
ExecutionTime	Lesser(1000)

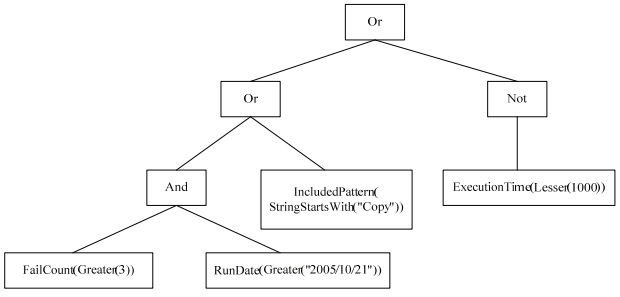

圖 3 Test Selection 與 Test Combination 展開後的樹狀結構範例

- Resource Monitor：負責監控程式開發人員對於程式碼所做的異動。
- Test Case Selector：實作在第二節中所描述的各種測試案例選擇策略，以作為程式開發人員選擇測試案例的依據。
- Test Launcher：負責執行 Test Case Selector 所挑選出來的測試案例。

- Test Run Listener：紀錄測試的結果，並產生測試結果報表。

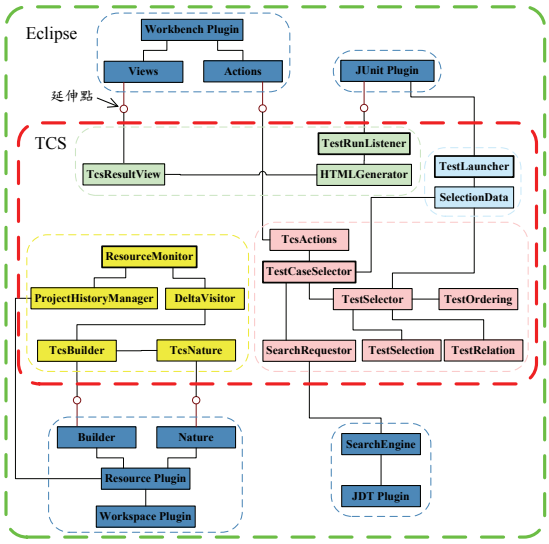

圖 4 Reg4J 系統概念圖

五、使用範例

5.1 選擇測試某函式的所有測試案例

假設程式開發人員撰寫 IntOracleTest 測試案例用以測試 IntOracle 函式。假設該測試案例執行失敗，以下為程式開發人員使用 Reg4J 來執行迴歸測試的步驟：

1. 程式開發人員依據錯誤訊息，開啟測試程式碼，發現錯誤發生於測試案例的第 25 行，如圖 5 所示。如果 IntOracle 函式所接受的參數相等，應該要傳回 “-1”。
2. 程式開發人員開啟原始程式碼，將第 22 行的傳回值改為“-1”，如圖 6 所示。
3. 由於程式開發人員修改了 IntOracle 的原始碼，因此所有用來測試 IntOracle 函式的測試案例都應該被執行。此時程式開發人員選擇 Reg4J 的「Run test cases testing the selected method」功能。此時 Reg4J 會找出所有測試 IntOracle 函式的測試案例，並執行之。執行結果如 8 所示，顯示測試執行成功。

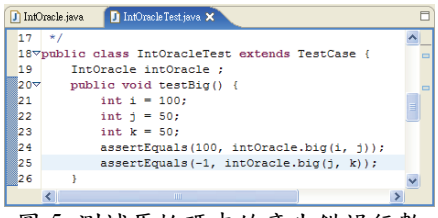

圖 5 測試原始碼中的產生錯誤行數

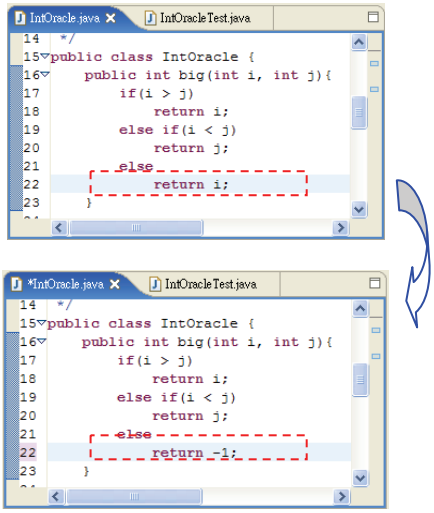

圖 6 修改原始程式碼

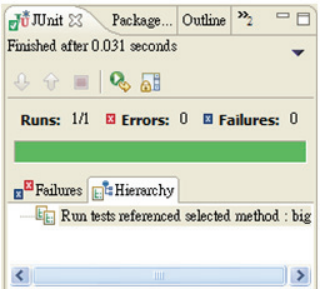

圖 8 測試成功

5.2 使用自定篩選條件

假設我們要選擇同時符合以下條件的測試案例：(1)在一個禮拜以內被修改過的測試，且(2)在三天內被執行過的測試，且(3)測試成功的次數小於 10 次；或(4)測試執行時間小於 100 毫秒的測試。此外，我們排序所選擇出的測試案例，讓最常失敗的最先執行。以下為程式開發人員使用 Reg4J 來執行迴歸測試的步驟：

1. 程式開發人員選擇 Reg4J 的「Custom test case selection」。
2. 系統跳出一個自定篩選條件的對話框，程式開發人員依序將 ModificationDate、RunDate 和 SuccessCount 的欄位打勾，選擇要比較的關係運算子，並填入各個篩選條件的比較值。因為是要同時符合這三個條件，因此按下 AND 按鈕，可以看到在選擇清單中，列出我們所設定的篩選條件組合。
3. 接著，將 ExecutionTime 的欄位打勾，選擇 Lesser 的關係運算子，填入 100，然後按下 OR 按鈕。同時也選擇測試執行的順序依據 Frequently Failed First，之後在這一個自定測試篩選條件的說明中，輸入 “Run tests as I want” 。完成結果如圖 9 所示。
4. 按下「Run Tests」的按鈕執行所選擇的測試。

圖 10 為執行結果報表，我們可觀察到其執行順序和設定的相同。

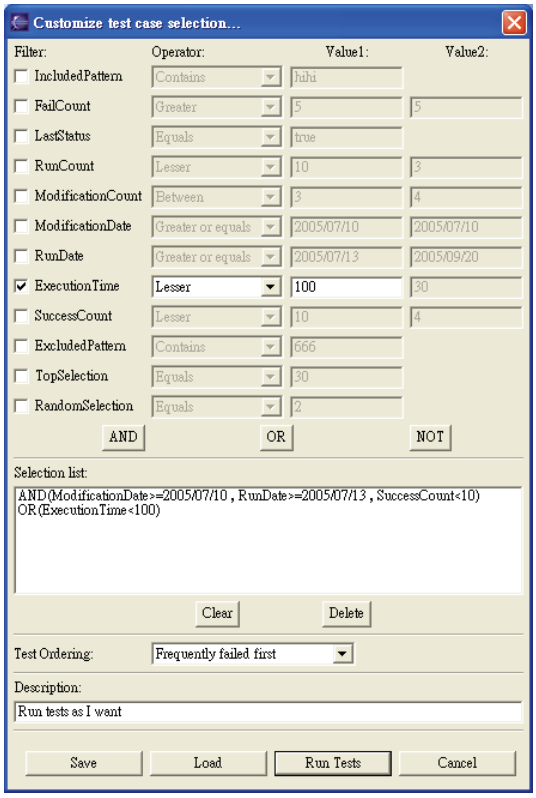

圖 9 自定篩選條件對話框

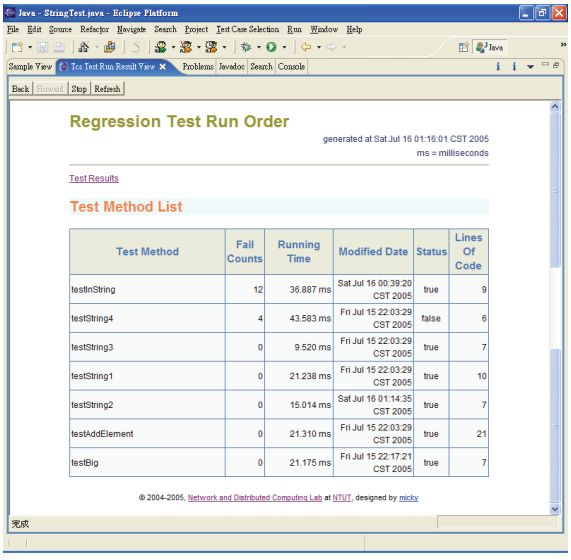

圖 10 觀看自定篩選測試執行的結果報表

六、結論與未來展望

本論文提出一個前端迴歸測試選擇模型與數個實用的測試案例選擇方法，搭配篩選條件和關係運算，能夠幫助程式開發人員在有限的測試執行時間中自動篩選出所需要的測試案例，以降低在敏捷式軟體開發流程中，頻繁執行前端迴歸測試所需的時間。我們在 Eclipse 平台上實作工具來支援所提出的測試案例選擇方法。藉由將工具整合進 Eclipse 平台中，使得程式開發人員的程式撰寫與測試流程更加緊密的結合在一起，以期提高其生產力與軟體品質。

未來發展方向包含擴充新的篩選條件，如依據測試程式與待測程式的行數、程式異動程度等。此外，本工具與測試驅動開發方法的互動關係亦是另一個研究的重點。

致謝

本研究由國科會計劃 NSC95-2221-E-027-047 補助，特此致謝。

參考文獻

[1] S. W. Ambler and R. Jeffries, *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, Wiley, 2002.

[2] K. Beck, *Extreme Programming Explained*, Addison-Wesley, 1999.

[3] K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2002.

[4] A. Cockburn, *Agile Software Development*, Addison Wesley, 2002.

[5] Eclipse.org, http://www.eclipse.org.

[6] M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[8] M. J. Harrold, A. Orso, J. A. Jones, M. Pennings, T. Li, S. Sinha, D. Liang, S. A. Spoon, and A. Gujarathi, “Regression Test Selection for Java Software,” *OOPSLA*, ACM, 2001.

[9] T. Husted and V. Massol, *JUnit in Action*, Manning, 2003

[10] JUnit, http://www.junit.org/.

[11] S. McConnell, *Rapid Development*, Microsoft Press, 1996.

[12] G. Rothermel, M. Harrold, J. Dedhia, “Regression Test Selection for C++ Software,” *Journal of Software Testing, Verification, and Reliability*, vol. 10, no. 2, June 2000.

[13] D. Saff, M. D. Ernst, “Reducing wasted development time via continuous testing,” Proceedings of the 14th International Symposium on Software Reliability Engineering, 2003.

[14] D. Saff, M. D. Ernst, “An Experimental Evaluation of Continuous Testing During Development,” Proceedings of the 2004 International Symposium on Software Testing and Analysis, 2004.