1. 緒論
   1. 研究背景與動機
   2. 研究目標
   3. 論文組織架構
2. 背景知識

2.1 Robusta

2.2 例外處理壞味道

2.3 系統強健度等級

1. 研究方法

3.1消除壞味道流程

1. 實驗結果
2. 結論與未來展望

# **第二章 背景知識**

## **2.1 Robusta**

Robusta是一個靜態分析Java程式碼的工具，它能夠偵測出程式碼中例外處理的壞味道並且產生報表。在Eclipse的工具列中，點擊Help→Eclipse Marketplace…後，搜尋Robusta安裝即可。安裝完畢後，對專案點擊右鍵選擇Properties→Robusta Detecting Settings，能夠設定要偵測的壞味道類型、folders等。而在Robusta工具中定義了六種例外處理壞味道，分別為：

1. Dummy Handler
2. Empty Catch Block
3. Unprotected Main Program
4. Nested Try Statement
5. Careless Cleanup
6. Exception Thrown Form Finally Block

下一小節將會一一介紹。

## 2.2例外處理壞味道

### 2.2.1 Empty Catch Block

當程式碼中有會丟出Checked Exception的函式時，Java編譯器會要求開發者對這些函式作處理，處理的方法有兩種：在介面宣告會丟出例外和用try/catch將函式包覆起來。為了不造成「介面演進」的問題，普遍開發者會選擇用try/catch將函式包住。因此，當try裡的程式碼發生例外時，例外被catch捕捉後，如果catch裡不做任何事的話，會造成例外被掩蔽的現象，稱之為Empty Catch Block壞味道。此種忽略例外的作法會造成開發者除錯不容易，並且掩蔽例外發生的事實，讓程式碼帶著不正確的狀態繼續往下執行，而降低系統強健度。



圖 Empty Catch Block範例

### 2.2.2 Dummy Handler

Dummy Handler壞味道定義和Empty Catch Block相似，差別在於當程式在try裡發生例外時，例外被catch捕捉後，catch裡只有紀錄例外訊息，沒有再做其他的處理。雖然例外有被記錄下來，但如果在圖形化、網頁化界面等等情境下，使用者或開發者不容易直接看到例外訊息。而這種做法容易產生例外已經被處理的假象，但實際上卻沒有對例外進行修復。可能產生的影響為當系統發生錯誤時，例外訊息只有被印出，除了造成開發人員不容易除錯，也會讓程式碼繼續再不正確的狀態往下執行。因此Dummy Handler壞味道會增加開發者或除錯者除錯的困難度，同時也會降低系統強健度。



圖 Dummy Handler範例

### 2.2.3 Unprotected Main Program

當例外被丟出後都沒有被捕捉時，這些例外都會被往上回報。因此，這些例外最後會來到來最上層的主程式中。如果主程式不處理這些例外，將會使程式發生不預期的終止。因此，當主程式裡的程式碼沒有被try/catch包覆住，並在catch裡對例外做處理的話，稱之為Unprotected Main Program壞味道。程式不預期的終止程式會被使用者認為是一種軟體品質不佳的表現。



圖 Unprotected Main Program範例

### 2.2.4 Nested Try Statement

Nested Try Statement壞味道定義為當try、catch或finally區塊中存在巢狀結構的try statement，稱之為Nested Try Statement壞味道。雖然Nested Try Statement壞味道並不會對程式碼造成影響，也沒有程式邏輯上的錯誤，但是對開發者來說，複雜的巢狀結構將不容易閱讀，並且不容易測試和維護。



圖 Nested Try Statement範例

### 2.2.5 Careless Cleanup

Careless Cleanup壞味道定義為當程式碼執行時，在執行到釋放資源的程式碼之前發生例外狀況的話，會無法執行到釋放資源的程式碼，而造成資源沒有正確被釋放。Careless Cleanup壞味道會導致資源被消耗並且降低系統穩定度。如下圖所示，若程式碼執行到第14行發生例外時，則會進到第16行catch裡對例外進行處理，而造成程式沒有執行到第15行釋放資源的程式碼，導致資源沒有正確被釋放。



圖 Careless Cleanup範例

### 2.2.6 Exception Thrown Form Finally Block

有一段try/catch/finally的程式碼，當程式在try或catch發生例外時，若對例外進行的處理方式為向上層呼叫者回報後，程式最後會執行finally裡的程式碼。如果當finally裡的程式碼也發生例外並向上層回報的話，會覆蓋掉原先try或catch裡所回報的例外，產生「例外蓋台」的現象，稱之為Exception Thrown Form Finally Block壞味道。這個壞味道會誤導開發者關注在finally發生的例外，而忽略了原先try或catch所發生的例外。如圖所示，當try的第12行發生例外時，程式會來到第13行的catch裡將例外回報，最後再執行finally裡第16行釋放資源的程式碼。但如果第16行發生例外時，會覆蓋掉原來在第14行回報的例外，而改回報第16行的例外。



圖 Exception Thrown Form Finally Block範例

## **2.3強健度等級**

系統強健度等級可以分為四級，分別為：

### 等級0：未定義(Undefined)

當開發人員還沒對他的系統貼上強健度等級的標籤時，則該系統的強健度等級為0。在這個等級的系統發生例外時，處理的方法可能有印出例外訊息讓開發人員知道發生例外，也有可能是不做任何事直接忽略例外。可能產生的問題為當系統發生錯誤時，開發人員無法確切得知系統在哪裡發生錯誤。

如圖所示，當Component A發生例外E時，由於Component A沒有定義強健度，造成Component B無法得知Component A是執行成功還是將例外掩蔽起來，而無法做後續的處理，因此可能造成系統crash或繼續執行，無法確切的掌握。



圖 等級0：未定義

### 等級1：錯誤回報(Error reporting)

錯誤回報為當例外發生時，一定要讓呼叫者知道，讓呼叫者了解有例外發生，因此只要將例外往上一層丟出即可。

如圖所示，當Component A發生例外時，如果不進行任何處理，直接將例外回報給Component B，如果Component B也不處理，則繼續往上層回報，直到回報到最上層的Component D。最後，由最上層Component處理例外或紀錄例外訊息。



圖 等級1：錯誤回報

### 等級2：狀態回復(State recovery)

達到這個等級時，一定要先滿足等級1，並且在錯誤發生後讓系統回復到原本正確的狀態。因此，例外發生後，系統還是能夠繼續正常執行。

如圖所示，Component A發生例外後，將例外回報給Component B；接著，Component B再將例外回報給ComponentC。由於Component C接到例外後造成狀態錯誤，因此先將狀態回復成讓程式能夠繼續正常提供服務後，再將例外回報ComponentD，由最上層Component 處理例外或紀錄例外訊息。



圖 等級2：狀態回復

### 等級3：行為回復(Behavior recovery)

系統強健度要到達等級3，必須先滿足等級2的條件外，還要重試例外發生前的行為或是尋找其他方法來達成原本的任務。如果重試行為失敗或是尋找的其它方法也發生例外，則要將例外向上層回報。

如圖所示，Component A發生例外後，會先將狀態回復之後，再重試行為，如果重試成功後，則程式繼續往下執行。如果重試失敗，如圖所示，則會將例外向上層回報，由最上層Component處理例外或紀錄例外訊息。



圖等級3：行為回復

本篇論文主要關注在讓系統強健度從等級0提升到等級1的階段。

# 第三章 研究方法

本章節中，會先介紹我所提供的例外處理壞味道消除流程；接著再介紹要如何消除這些例外處理壞味道。而消除例外處理壞味道的方法為延續洪哲瑋的論文「例外處理程式壞味道的自動化偵測與重構」，在Dummy Handler、Ignore Checked Exception和 Unprotected Main Program壞味道中，會先介紹洪哲偉論文中所提供的方法是如何消除這些壞味道，瞭解消除的方法後，再介紹我們發現哪些功能與當時提供的方法不一致或不夠完美，因此重新修正或改善這些功能；在Nested Try Statement壞味道的修復方法為洪哲瑋論文所提供的方法；最後，本論文增加Careless Cleanup、Exception Thrown Form Finally Block壞味道的消除方法，並將以上的功能實作於Robusta工具中，使其能夠自動化消除程式碼中例外處理的壞味道。

## 3.1 壞味道消除方法介紹

### 3.1.1 Dummy Handler & Empty Catch Block

根據洪哲瑋論文中提到的Ignore Checked Exception 壞味道，在楊智傑的「Robusta－一個對於Java例外處理壞味道的偵測工具」論文中被正式定義為Empty Catch Block。而Empty Catch Block壞味道的移除方式與Dummy Handler壞味道是一樣的，於是我將這兩種壞味道放在本小節中一起敘述。如圖所示，以下將以Dummy Handler為範例。



圖 Dummy Handler修復前範例

根據洪哲瑋的論文，這兩種壞味道的消除方法都是當catch捕捉到例外後，如果沒有要處理例外，應該將例外往上一層回報，因此提供的方法為修復方法及重構方法。而修復和重構兩種方法的差異點在於重構具有比較彈性的功能，部分的參數能夠讓使用者自己決定，而修復的方法為固定的，使用者沒有辦法做更動。

* 修復方法：

如圖第14行所示，當catch捕捉到例外後，透過工具的「快速修復」功能，自動於catch區塊中捕捉到的例外轉換為RuntimeException，並回報給上一層。

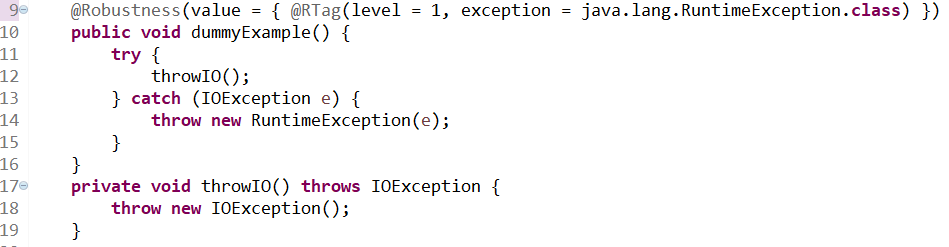


圖 Dummy Handler壞味道修復結果

* 重構方法：

如圖第15行所示，當使用者希望能夠丟出的例外型態為自己所定義的，而非上述所提到的RuntimeException時，洪哲瑋的論文敘述能夠透過工具所提供的來選擇自己所希望拋出的例外類型來重構。

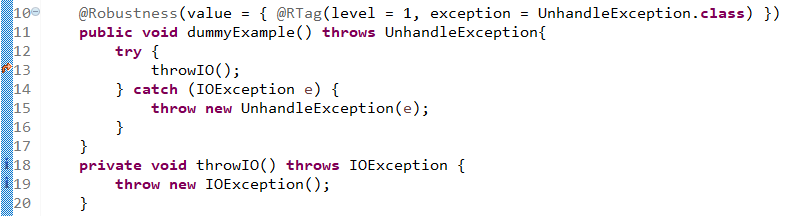


圖 Dummy Handler壞味道重構結果

我們發現工具提供消除壞味道的功能與當時論文所提出的不太一致，提供的功能有三種，分別為：

1. 修復方法­：丟出例外
2. 修復方法­：丟出RuntimeException
3. 重構方法：丟出Unchecked Exception

在方法3的重構方法中，丟出的Unchecked Exception預設為RuntimeException，我們認為方法2的修復方法­與方法3的重構方法丟出的例外都屬於Unchecked Exception，因此功能相似，所以將方法2拿掉，留下快速修復方法與重構方法，並根據現有功能來重新修正這兩種方法。

* 重新修正後的修復方法：

顧名思義為「快速修復」的功能，因此不論catch所捕捉的例外為Checked Exception或Unchecked Exception，都直接將所接到的例外丟出來向上層回報，並在介面宣告所丟出的例外，以最快速的方法來消除Dummy Handler 和 Empty Catch Block這兩種壞味道。如圖第15、17行所示，不論catch捕捉什麼例外，都直接丟出向上層回報，以達到快速修復來消除壞味道的目的。



圖 Dummy Handler壞味道重新修正後的修復結果

* 重新修正後的重構方法：

我們發現工具在這兩種壞味道所提供的選單中，選單中只會顯示Unchecked Exception的類型，因此使用者只能選擇自己定義的Unchecked Exception。並且在選擇Unchecked Exception丟出後，介面也不會宣告所回報的例外。因此我們將重構方法功能修正為丟出Unchecked Exception來進行回報。如圖所示，選單會提供所有的Unchecked Exception，並根據使用者的輸入來從選單中來對Unchecked Exception進行篩選，圖中的UnhandleException為使用者自己定義的Unchecked Exception。如圖所示，透過回報例外來消除壞味道後不會在介面進行宣告。

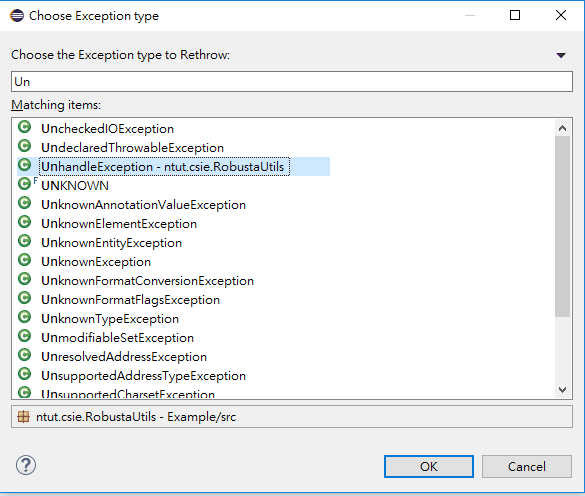


圖 選單提供的Unchecked Exception

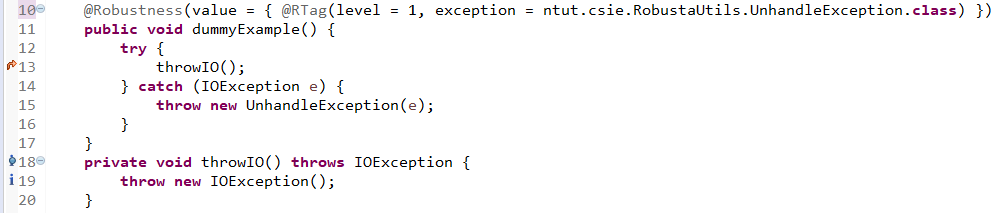


圖 Dummy Handler壞味道重新修正後的重構結果

### 3.1.2 Unprotected Main Program

根據洪哲瑋的論文，因為消除Unprotected Main Program壞味道的方法是固定的，因此Unprotected Main Program壞味道的消除方法只提供了修復方法。如圖所示，修復方法為在main program中去捕捉Exception型態的例外，並在catch區塊中標示//TODO: handle exception來提醒使用者要處理這個例外。



圖 Unprotected Main Program壞味道修復結果

根據洪哲偉的論文，我們發現會產生兩種問題，第一種問題為：雖然透過修復方法將Unprotected Main Program壞味道消除後，將會產生Empty Catch Block壞味道。要消除Unprotected Main Program壞味道的方法為用try/catch將main program保護住，如果還不清楚catch要做什麼例外處理的話，能夠先將catch區塊中處理方式用顯示錯誤並記錄到日誌檔中，最後結束程式執行。因此我們將catch區塊中的//TODO: handle exception拿掉，改為ex.printStackTrace();，讓catch區塊中有做處理。第二種問題為：

並且根據Unprotected Main Program壞味道的定義，當主程式發生例外後，軟體會發生不預期的終止。如圖所示，因此雖然catch有去捕捉Exception型態的例外，但如果系統發生Error時，main program還是會因為沒有捕捉Error而造成系統異常終止。因此我們將catch捕捉的Exception改為Throwable。



圖 錯誤物件的繼承架構

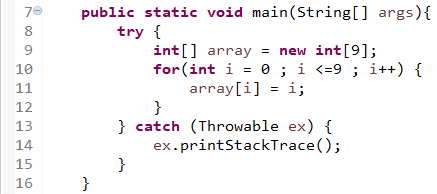


圖 Unprotected Main Program壞味道重新修正後的修復結果

### 3.1.3 Nested Try Statement

Nested Try Statement壞味道為try statement裡又有try statement，多層的巢狀結構容易讓開發人員或維護人員不容易閱讀程式碼。消除Nested Try Statement壞味道的方法為將巢狀結構的try statement抽成一個函式，降低程式碼的複雜度。根據洪哲偉的論文，因為Eclipse已經提供重構功能，所以不在工具中提供重構功能。因此如果要消除Nested Try Statement壞味道，步驟為先將要抽成函式的try statement圈選起來，點擊滑鼠右鍵選取Refactor→Extract Method…按鈕；接著在Extract Method的視窗中替選取的try statement命名要抽出的函式名稱，點擊OK後Nested Try Statement壞味道就消除了。如圖所示，選取第9~15行的try statement結構，選取Extract Method；接著如圖所示，在視窗中的Method name填寫要取的函式名稱；最後如圖所示，Nested Try Statement壞味道就被消除了。



圖 選取Eclipse 提供的refactor功能



圖 在Extract Method視窗替函式命名



圖 Nested Try Statement壞味道重構結果

我們發現在工具中有提供Nested Try Statement壞味道的重構功能，雖然重構後的結果是一樣的，但工具與Eclipse的Extract Method做連結，使Nested Try Statement壞味道的重構功能自動化了，因此使用者不用再圈選出要獨立抽成函式的try statement。如圖所示，點擊工具的「Refactor==>Extract Method」後，工具會自動連結Eclipse的Extract Method功能，而顯示Method name的視窗，填寫函式名稱後，Nested Try Statement壞味道就被消除了。



圖 工具提供的Extract Method功能

### 3.1.4 Careless Cleanup

Careless Cleanup壞味道為當程式碼執行到釋放資源之前發生例外，會造成釋放資源的程式碼不會被執行。因此，消除Careless Cleanup壞味道的步驟為：將try statement增加finally block；接著，在finally block裡加入try statement，保護程式碼在執行釋放資源時如果發生意外，不會讓例外從finally block丟出，而造成Exception Thrown From Finally壞味道，並在catch block終將捕捉到的資源釋放例外記錄到日誌檔中；最後，在try block裡將釋放資源的程式碼放入，並在執行釋放資源之前，檢查物件是否為空。如圖所示，為Careless Cleanup壞味道修復後的結果，雖然會產生Nest Try Statement壞味道，但再使用工具的自動修復功能來消除即可。



圖 Careless Cleanup壞味道修復結果

### 3.1.5 Exception Thrown Form Finally Block

當finally block裡的程式碼丟出例外時，會覆蓋掉原先try/catch block所丟出的例外，而誤導開發人員或維護人員忽略try/catch block所丟出的例外，稱為Exception Thrown Form Finally Block壞味道。

要消除這個壞味道的方法，就是不要讓finally block丟出例外。因此，要消除Exception Thrown Form Finally Block壞味道的步驟為：先將finally block會丟出例外的程式碼用try/catch保護住；接著，當釋放資源失敗而在catch 捕捉到例外後，將例外寫入日誌檔中；最後，將finally裡的這個try statement獨立抽成一個函式即可。如圖所示，為Exception Thrown Form Finally Block壞味道的重構結果。雖然消除Exception Thrown Form Finally Block壞味道後會產生Dummy Handler壞味道，但我們認為在Exception Thrown Form Finally Block跟Dummy Handler這兩種壞味道的權衡之下，留下Dummy Handler壞味道相對來說對程式碼的影響是比較小的，因此才會在Exception Thrown Form Finally Block壞味道消除後而殘留下Dummy Handler壞味道。



圖 Exception Thrown Form Finally Block壞味道重構結果

## 3.2 設計與實作

在工具偵測完例外處理壞味道後，會標示警告訊息提醒使用者要消除壞味道，並針對每種不同的壞味道提供快速修復或重構的功能。

### 3.2.1 快速修復

當程式碼偵測完例外處理壞味道後

## 3.3 與學長的差異

## 3.4 壞味道的偵測、暴露及消除流程

要消除例外處理壞味道，我們提供了一套偵測、暴露及消除壞味道的流程，總共有四個步驟。第一步：對專案點擊右鍵­→Robusta→Detect smell，來偵測專案有哪些例外處理壞味道，產生例外處理壞味道的報表，並透過報表來到某一段有例外處理壞味道的程式碼，程式碼旁邊會有Robusta對壞味道的標記；第二步：產生暴露程式碼壞味道的測試案例，根據劉彥麟論文提供的利用測試案例來暴露壞味道所帶來的影響，執行測試案例，如果程式碼存在壞味道，則測試會失敗；第三步：藉由本論文介紹的壞味道消除方法，消除程式碼的壞味道；第四步：再執行一次剛剛失敗的測試案例，測試通過代表該例外處理壞味道已經消除，強健度因此提升。如圖所示，為我們提供的壞味道暴露及消除流程。



圖 壞味道偵測、暴露及消除流程

# 第四章 應用實例

本章節會以不同的開源專案：JFreeChart和tomighty為範例，將第三章介紹的壞味道消除方法實作在Robusta工具中並使其自動化，並透過壞味道消除流程來進行分析及實作。

JFreeChart由Java語言所撰寫，是Java平台上的免費的圖表庫，能夠支援多種圖表，並應用於客戶端(例如：JavaFX或Swing)或伺服器端(匯出各種圖表格式，例如：PNG、SVG)。本論文將用JFreeChart來分析Dummy Handler及Exception Thrown Form Finally Block兩種壞味道。

tomighty是一個採用「番茄鐘工作法」的桌面的時間管理工具，由Java語言所撰寫，來幫助使用者管理時間，提高工作效率。

## 4.1 Dummy Handler應用實例

### 4.1.1 偵測Dummy Handler

如圖所示，第99、102行為JFreeChart專案中偵測出來的Dummy Handler壞味道。這是一段讀XML檔中Data set的程式碼，catch block會捕捉兩種例外，分別為SAXException和ParserConfigurationException，我們將以SAXException來進行分析與介紹。當第93行的newSAXParser()函式發生SAXException時，會到第98行被catch捕捉住並印出例外訊息，因此這是一個Dummy Handler壞味道，最後程式會回傳null。



圖 JFreeChart Dummy Handler壞味道範例

### 4.1.2產生暴露Dummy Handler壞味道的測試案例

我們認為當程式碼在93行發生例外時，程式碼已經在不正確的狀態，被catch捕捉後卻沒有對狀態進行回復或任何處理，只有印出例外訊息後讓程式碼繼續帶著不正確的狀態執行下去，如圖所示，藉由劉彥麟同學的論文產生暴露壞味道影響的測試案例後，再將其測試案例相關的設定補齊，如圖所示，測試案例會因為壞味道存在而造成測試失敗。



圖 JFreeChart Dummy Handler壞味道測試案例



圖 JFreeChart Dummy Handler壞味道測試失敗

### 4.1.3消除Dummy Handler壞味道

如圖所示，接著透過本論文在工具提供的自動化快速修復或重構功能來消除壞味道，本論文以重構方法為例。如圖所示，原先失敗的測試案例在壞味道消除後會成功。



圖 Robusta工具提供自動化消除Dummy Handler壞味道



圖JFreeChart Dummy Handler壞味道消除的結果



圖 JFreeChart Dummy Handler壞味道消除後測試成功

## 4.2 Exception Thrown Form Finally Block應用實例

### 4.2.1 偵測Exception Thrown Form Finally Block

如圖所示，工具偵測JFreeChart後，發現專案中saveChartAsPNG的函式含有Exception Thrown Form Finally Block壞味道。這段程式碼是將圖表存成PNG的圖片格式，當308行發生例外時，會將例外像上一層回報，接著執行finally block裡第311行釋放資源的程式碼，如果第311行發生例外，將會覆蓋第308行原本要回報的例外，改丟出第311行的例外，因此這是一個Exception Thrown Form Finally Block壞味道。



圖 JFreeChart Exception Thrown Form Finally Block壞味道範例

### 4.2.2產生暴露Exception Thrown Form Finally Block壞味道的測試案例

如圖所示，藉由劉彥麟同學的論文產生暴露壞味道影響的測試案例，並將saveChartAsPNG函式所需要的參數準備好後，接著執行測試案例，如圖所示，測試案例會因為壞味道存在而造成測試失敗。



圖 JFreeChart Exception Thrown Form Finally Block壞味道測試案例



圖 JFreeChart Exception Thrown Form Finally Block壞味道測試失敗

### 4.2.3消除Exception Thrown Form Finally Block壞味道

如圖所示，接著透過本論文在工具提供的自動化修復功能來消除壞味道。如圖所示，會將原本finally block釋放資源的程式碼抽成一個函式，並用try/catch保護住，讓finally block不再丟出例外。最後，再執行一次測試案例，如圖所示，原先失敗的測試案例在壞味道消除後會成功。



圖 Robusta工具提供自動化消除Exception Thrown Form Finally Block壞味道



圖 JFreeChart Exception Thrown Form Finally Block壞味道消除的結果



圖 JFreeChart Exception Thrown Form Finally Block壞味道消除後測試成功

## 4.3 Unprotected Main Program實驗結果

如圖所示，為tomighty工具正常啟動時的畫面。



圖 tomighty工具

### 4.3.1 偵測Unprotected Main Program

如圖所示，為tomighty專案的main program所在位置。透過工具的偵測顯示它具有Unprotected Main Program壞味道。因此，當main program發生例外時，程式會發生不預期的終止，而被使用者認為是軟體品質不佳的表現。



圖 tomighty Unprotected Main Program壞味道範例

### 4.3.2產生暴露Unprotected Main Program壞味道的測試案例

如圖所示，藉由劉彥麟同學的論文產生暴露壞味道影響的測試案例，接著執行測試案例，如圖所示，測試案例會因為壞味道存在而造成測試失敗。



圖 tomighty Unprotected Main Program壞味道測試案例



圖 tomighty Unprotected Main Program壞味道測試失敗

### 4.3.3消除Unprotected Main Program壞味道

如圖所示，接著透過本論文在工具提供的自動化修復功能來消除壞味道。如圖所示，原先失敗的測試案例在壞味道消除後會成功。



圖 Robusta工具提供自動化消除Unprotected Main Program壞味道

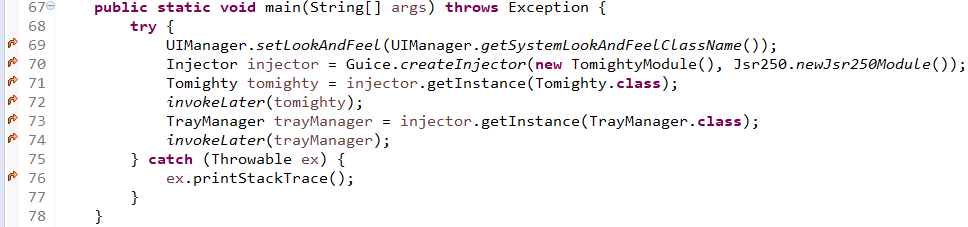


圖 tomighty Unprotected Main Program壞味道消除的結果



圖 tomighty Unprotected Main Program壞味道消除後測試成功