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. (實作方法)
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：在介面會告會丟出例外和用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，並且降低系統強健度。



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。雖然例外被記錄下來，但如果在圖形化、網頁化等等情境下，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，造成開發人員不容易除錯。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。



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。對使用者來說，程式不預期的終止程式會認為是一種軟體品質不佳的表現。



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裡存在巢狀結構的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。



Nested Try Statement範例

### 2.2.5 Careless Cleanup

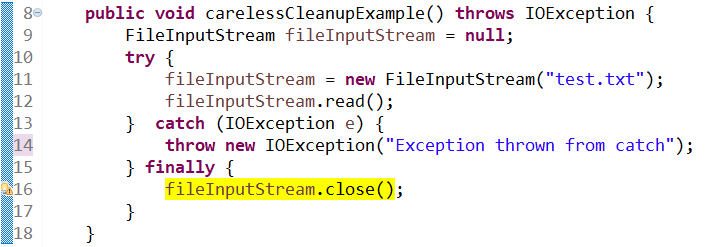
Careless Cleanup壞味道定義為當程式碼執行時，在執行到釋放資源的程式碼之前發生例外狀況的話，會無法執行到釋放資源的程式碼，而造成資源沒有正確被釋放。Careless Cleanup壞味道會導致資源被消耗並且降低系統穩定度。如下圖所示，若程式碼執行到第14行發生例外時，則會進到第16行catch裡對例外進行處理，而造成程式沒有執行到第15行釋放資源的程式碼，導致資源沒有正確被釋放。



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，對例外進行的處理方式為向上回報後，程式最後會執行finally裡的程式碼。如果當finally裡的程式碼也丟出例外的話，會覆蓋掉原先try或catch裡所丟出的例外，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，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。



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。

### 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，因此只要將例外網上一層丟出即可。

### 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處於正確的狀態。因此，例外發生後，系統還是能夠繼續執行。

### 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，還需要尋找其他方法來達成原本的任務，常用的方法為重試。

我主要關注在讓系統強健度從等級0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。

## 3.2 壞味道消除方法介紹

### 3.2.1 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。

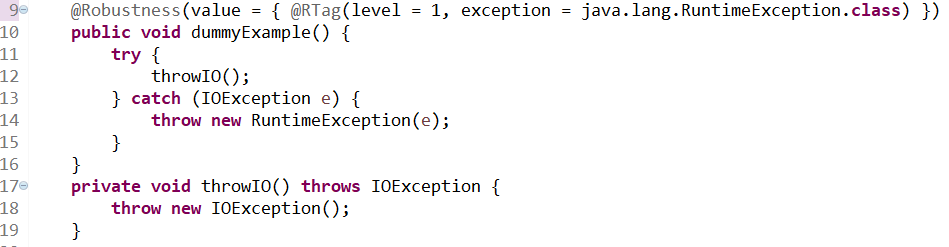


圖 Dummy Handler壞味道修復結果

* 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。

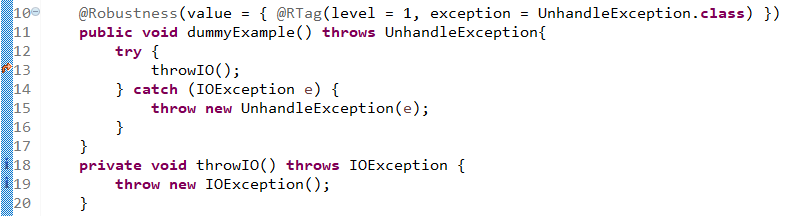


圖 Dummy Handler壞味道重構結果

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，因此我們認為方法2的修復方法­與方法3的重構方法所丟出的例外都是屬於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。

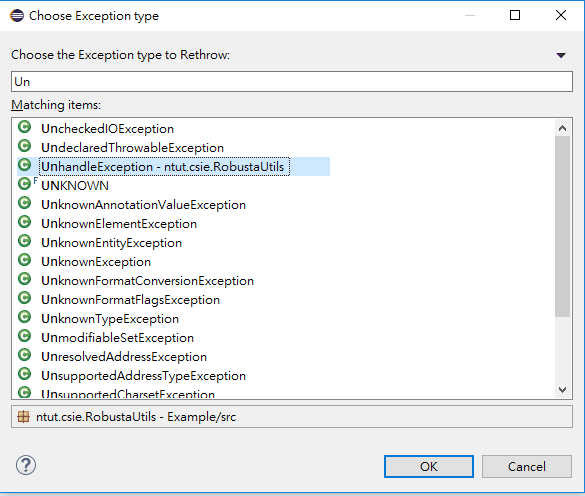


圖 選單提供的Unchecked Exception

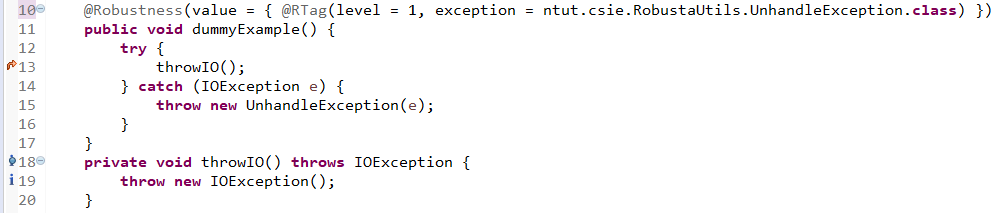


圖 Dummy Handler壞味道重新修正後的重構結果

### 3.2.2 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

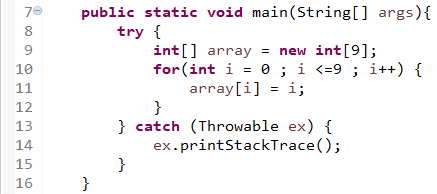


圖 Unprotected Main Program壞味道重新修正後的修復結果

### 3.2.3 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

### 3.2.4 Careless Cleanup