第二章 相關研究

Empty catch block

「Empty catch block」此壞味道意旨當程式發生例外並捕捉例外後，沒有對此例外進行處理，亦即 catch black 內為空的。

影響

此壞味道的例外處理機制方式為不處理，此作法會隱藏錯誤訊息，會使開發者往後若遇到例外狀況發生，除錯困難度將會提升。

範例:  
如下圖所式，第9行writeFile函式會創立一個新檔案接著把資料寫入該檔案中。根據Java 的官方 API文件說明，第12行的new FileWriter(“k:\\test.txt”) 有可能會發生IOException。若程式執行到第12行時發生IOException，此例外將會被第16行的catch block接住，但catch block 並未對其做處理，最終此壞味道將被忽略導致開發人員除措困難度將會提升



Dummy handler

定義:

「Dummy Handler」當程式發生例外並捕捉例外後，處理方式只印出或紀錄例外訊息，而沒有實質做處理;而如果在印出或紀錄壞味道的同時，也有將其例外丟出，則不算是壞味道。  
影響

此壞味道的例外處理機制方式為印出或紀錄例外，其效果也幾乎等同忽略例外，在程式執行時，開發者和使用者很難觀察到這些訊息。

範例

如下圖所式，第 writeFile函式會創立一個新檔案接著把資料寫入該檔案中。根據Java 的官方 API文件說明，第12行的new FileWriter(“k:\\test.txt”) 有可能會發生IOException。若程式執行到第12行時發生IOException，此例外將會被第16行的catch block接住，但catch block 僅記錄錯誤訊息。



Nested Try statement

「Nested Try Statement」此壞味道的意旨在程式碼中存在著巢狀的try block

影響

對開發者來說，此壞味道複雜的結構將會影響程式碼的可讀性、測試性以及維護性。

範例：如下圖所示，在第17行的finally block 會進行資源釋放的工作，然而很多關閉資原的函數都會丟出例外，用來代表式放資源失敗。因此在finally block中很容易發生巢狀 try statement 的情況，使得程式碼結構變得複雜集難以維護



Unprotected Main Program  
「Unprotected Main Program」此壞味道的意旨在程式碼中，主程式中的程式碼並沒有被 try block 包覆起來

影響  
當程式執行時發生例外，沒有被捕捉到的例向上把例外傳遞，最終傳到主程式而導致程式不預期的終止或產生錯誤

範例  
如下圖所示

Careless cleanup

「Careless cleanup」此壞味道的意旨在釋放資源前，發生例外導致資源無法被正常釋放

影響

此壞味道因為在釋放資源之前，因為例外導致資源無法正常被釋放或關閉，將導致資源耗盡並降低系統穩定度

範例:

如下圖所示，此writeFile函式會創立一個新檔案接著把資料寫入該檔案中，並釋放資源。根據Java 的官方 API文件說明，此函示在第14行 FileWriter函示、第15行 write、第16行flush均會丟出IOException，這些例外若發生在第17行釋放資源之前將會導致此close沒有被執行，導致資源持續被占用無法釋放



Exception thrown from finally block

「Exception thrown from finally block」, 此壞味道的特徵是在finally block中發生例外且此例外也被往外丟

影響  
此壞味道發生例外時，在 finally block 發生的例外會覆蓋 try block 或 catch block 所發生的例外，覆蓋的例外處理訊息會誤導程式人員，難以得知該例外是由哪一個block所造成

範例:

如下圖所示，此writeFile函式會創立一個新檔案接著把資料寫入該檔案中，並釋放資源。根據Java 的官方 API文件說明，此函示在第14行 FileWriter函示、第15行 write、第16行flush均會丟出IOException，此例外會被17行的catch block接住並往caller回傳此例外，但是再回傳之前會先進入19行的finally block，若此時21行的close函示也發生例外，此例外將會被22行的catch block接住並在23行將此例外往外丟，此時這個丟出去的例外將會把19行的例外覆蓋，導致開發者無從得知例外完整的例外狀況



Robusta

Robusta 為一個以Java開發的開源專案，是一個靜態程式碼分析工具，主要的功能是用來協助開發人員偵測出其Java程式中淺藏的例外處理壞味道，來幫助開發者找出程式中的例外處理壞味道。只要在**eclipse** 中的marketplace 正確安裝robusta，就能使其偵測目前已定義的六種例外處理壞味道，分別為Empty catch block、Dummy handler 、Nested Try statement、Unprotected Main Program、Careless cleanup，除此之外還可以藉由產生報表的方式，迅速的定位出例外處理壞味道的位置以及壞味道的數量，降低人工檢查大量程式碼的成本。

AspectJ

**AspectJ** 是一種基於java實做AOP(Aspect-Oriented Programming)的程式語言，

開發人員可以在不改動原有的程式碼下，額外增加原始碼的行為或改變狀態。  
透過 再透過 before、after、around 決定要嵌入程式的時機

將設計完的aspectJ程式與原始碼一起編譯並執行後，當程式執行到被鎖定的函式時，就會在特定的時機觸發AspectJ 嵌入程式碼。本研究將會以AspectJ搭配單元測試來驗證使用者例外處理的機制，是否符合我們所預期

第三章

根據陳友倫的研究:以aspectJ接露導因於例外處理的程式缺陷，此論文利用利用aspectJ的特性，在不更動原有程式碼下，在特定的程式碼中嵌入發生例外的程式碼，強制使其產生例外狀況，進而呈現壞味道對於軟體的影響。其研究結果發現，利用robusta產生的aspectJ程式碼可以讓例外處理壞味道對於軟體的影響被暴露出來。

本論文將從陳友倫的研究為基礎，將其提出的方法應用於robusta定義的壞味道中，讓所有的壞味道都能產生對應的AspectJ程式碼，並藉由aspectJ在特定程式碼的位置嵌入例外藉此程現出壞味道對於系統的影響

3.1暴露壞味道分析

3.1.1不同壞味道暴露其例外的方式

下表為 robusta 目前定義的壞味道以及對應的壞味道定義，此章節將將針對定義的壞味道逐一設計期暴露例外的方式，藉此呈現出壞味道對系統的影響

|  |  |
| --- | --- |
| 壞味道種類 | 壞味道定義 |
| Dummy Handler | 當程式發生例外並捕捉例外後，處理方式只印出或紀錄例外訊息，而沒有實質做處理; |
| Empty Catch Block | 程式發生例外並捕捉例外後，沒有對此例外進行處理，亦即 catch black 內為空的 |
| Exception Thrown From Finally Block | 此壞味道的特徵是在finally block中發生例外且此例外也被往外丟 |
| Careless cleanup | 此壞味道的意旨在釋放資源前，發生例外導致資源無法被正常釋放 |
| Unprotected Main Program | 此壞味道的意旨在程式碼中，主程式中的程式碼並沒有被 try block 包覆起來 |

Dummy handler

Dummy handler壞味道為當程式發生例外並捕捉例外後，處理方式只印出或紀錄例外訊息，而沒有實質做處理，因而掩蓋了錯誤訊息。接下來我們接介紹如何在dummy handler的狀況下，讓此壞味道對於軟體的影響現行(第xx行為dummy handler壞味道)。首先我們將鎖定該壞味道的catch block，並依據catch block 接取的例外做為要強制拋出例外的類別，以下圖為例ｘｘｘ，此壞味道的catch block 接取的例外類別為 xx exception，因此我們將用此例外類別做為嵌入例外的類別。下一步我們將回到try block中，尋訪try block 中的所有method invocation，並鎖定最先直行姊與嵌入例外為相同類型的method invocation，並將此鎖定的method invocation視為即將丟出例外的點並對其嵌入例外，藉由這種方式我們就可以將dummy handler這個壞味道所帶來的影響給暴露出來，最後我們薛要蒐集 xx method name、method name、exception type、class name、method 類型，以便生程aspectJt程式

Empty catch block

「Empty catch block」此壞味道意旨當程式發生例外並捕捉例外後，沒有對此例外進行處理，亦即 catch black 內為空的。因而掩蓋了錯誤訊息。接下來我們接介紹如何在Empty catch block的狀況下，讓此壞味道對於軟體的影響現行(第xx行為Empty catch block壞味道)。首先我們將鎖定該壞味道的catch block，並依據catch block 接取的例外做為要強制拋出例外的類別，以下圖為例ｘｘｘ，此壞味道的catch block 接取的例外類別為 xx exception，因此我們將用此例外類別做為嵌入例外的類別。下一步我們將回到try block中，尋訪try block 中的所有method invocation，並鎖定最先執行與嵌入例外為相同類型的method invocation，並將此鎖定的method invocation視為即將丟出例外的點並對其嵌入例外，藉由這種方式我們就可以將Empty catch block這個壞味道所帶來的影響給暴露出來，最後我們薛要蒐集 xx method name、method name、exception type、class name、method 類型，以便生程aspectJt程式

Exception Thrown From Finally Block

「Exception thrown from finally block」, 此壞味道的特徵是在finally block中發生例外且此例外也被往外丟，因此在 finally block 發生的例外會覆蓋 try block 或 catch block 所發生的例外，覆蓋的例外處理訊息會誤導程式人員，難以得知該例外是由哪一個block所造成。

接下來我們接介紹如何在Exception Thrown From Finally Block的狀況下，讓此壞味道對於軟體的影響現行(第xx行為Exception Thrown From Finally Block壞味道)。為了要呈現這個壞味道帶來的影響，我們需要製造兩個例外，這樣才可以重現出這個壞味道帶來的影響，首先我們將鎖定該壞味道的finally block，找尋在finally block第一個會丟出例外的method invocation，並將此鎖定的method invocation視為即將丟出例外的點並對其嵌入它本身的例外，如此一來我們就可以重現從finally block 丟出例外的狀況了，接下來我們需要找尋try block第一個會丟出例外的methodinvocation，將此鎖定的method invocation視為即將丟出例外的點並對其嵌入例外，以上步驟都完成後我們就可以嵌入兩個例外，並運用此方式將Exception Thrown From Finally Block這個壞味道所帶來的影響給暴露出來，最後我們薛要蒐集 xx method name、method name、exception type、class name、method 類型，以便生程aspectJt程式。

Careless cleanup

「Careless cleanup」此壞味道的意旨在釋放資源前，發生例外導致資源無法被正常釋放。接下來我們接介紹如何在careless cleanup的狀況下，讓此壞味道對於軟體的影響現行(第xx行為 careless cleanup壞味道)。為了要呈現這個壞味道帶來的影響，我們需要再釋放資源的methodInvocation前遷入例外，在釋放資源之前若發生例外，則在此try block內的釋放 method ivovcation 則不會被執行，讓資源無法正常被釋放，這樣才可以重現出這個壞味道帶來的影響，首先我們將鎖定該壞味道的 try block，並找尋此try block 內的釋放資源methodinvocation ，以此點當做起點，往使用資源的物件宣告點尋找，找出梨釋放資源methodinvocation 最遠且有可能發生例外的methodInvocationn視為即將丟出例外的點並對其嵌入例外，並運用此方式將 careless cleanup這個壞味道所帶來的影響給暴露出來，最後我們要蒐集 xx method name、method name、exception type、class name、method 類型，以便生程aspectJt程式。

Unprotected Main Program

「Unprotected Main Program」此壞味道的意旨在程式碼中，主程式中的程式碼並沒有被 try block 包覆起來，因此當程式執行時發生例外，沒有被捕捉到的例向上把例外傳遞，最終傳到主程式而導致程式不預期的終止或產生錯誤。接下來我們接介紹如何在Unprotected Main Program的狀況下，讓此壞味道對於軟體的影響現行(第xx行為 Unprotected Main Program壞味道)。為了要呈現這個壞味道帶來的影響，我們需要先掃描 main function是否存在 try block，若存在try block則找尋try block之外，最先會被執行到的methodinvoacation 視為即將丟出例外的點並對其嵌入例外。若不存在 try block 則以 main function中第一個methodinvocation 視為即將丟出例外的點並對其嵌入例外，為了呈現這個壞味道帶來的影響，我們採用的作法為不管methodinvocation 是否會發生例外，直接指定嵌入的例外為runtimeexception，並運用此方式將 Unprotected Main Program這個壞味道所帶來的影響給暴露出來，最後我們要蒐集 xx method name、method name、exception type、class name、method 類型，以便生程aspectJt程式。

3.1.2 一般壞味道呈現方式

根據程有綸的論文研究，在尚未使用 aspect 來暴露例外之前，若想要在執行階段讓例外發生，如下圖所示xxx函式會丟出ioexception，為了重現壞味道，我們必須在xxx method 強制寫入會丟出例外的函式.但強制謝入會丟出例外的函釋後，揭序的程式碼編譯器則會跳出unreachable code 提醒使用者該行程式碼因為被強制丟出例外的關係，所以將不會被執行且無法編譯。最後程式碼會如下圖所是，程式以經無法正常被執行，若要讓程式可以正常被執行，則須將強制寫入例外的程式碼下方的程式用住姊的方式註解掉，為了讓軟體可以恢復尚未暴露例外之前的狀態，我們必須把強制的例外程式移除並把住姊的部分拿掉，這樣的方式雖然可以達到暴露例外的效果，蛋在操作起來並不容易且消耗成本也高

3.2利用aspectJ程現壞味道的方法

到目前為止，我們知道程式碼中若有壞味道存在時，通常就是程式有bug存在的表徵。這些潛伏於例外處理中的不正確程式碼，經常不容易在測試的時候被發現，因為例外處理行為往往正是被理解的最少且測試程度最為不足的部分.

為了讓例外處理壞味道的影響可以呈現，根據程有綸的論文研究，我們使用AspectJ[23]來模擬例外發生。AspectJ是一種實作aspect oriented programming(AOP)技術的程式語言，允許在既有的Java程式內動態插入新的程式結構，如此我們便能夠在不改變原始碼的情況下，把程式的正常行為

與例外行為加以分隔。

3.2.1aspectj使用說明

3.2.2aspectj處發時機

根據aspectJ的特性，只要撰寫出的aspectJ程式碼符合語法規則且指定需要遷入程式碼的對象、時機、作用範圍，即可進行遷入程式碼的動作。當指定的methodinvocation被執行時,aspectJ程式碼就會被啟動並達到遷入程式碼的動作，本論文的啟用時機以陳友倫的論文做改善及調整，如下圖所是，

陳友倫的觸發時機為指定的methodinvocation被執行時,aspectJ程式碼就會被啟動並達到遷入程式碼的動作，蛋這樣的啟動時機點會造成不管任何情況下都匯嵌入程式碼並丟出例外，會導致正常的程式碼無法啟動，所以其論文需要再aspectj的程式碼中而外加入一個switch的開關來手動控制例外是否要被丟出。  
本論文根據其想法，將其觸發時機限定於執行測試時自動啟動，相較於以往的手動調整更彈性了一些

3.2.3 aspect 依據不同壞味道的材料收集

當robusta完成掃描後，我們可以得知例外處理壞味道的範圍(如圖中螢光筆畫線的位置)，robusta會將壞味道的資訊紀錄在marker中(圖中的燈泡位置) ，當我們按下add aspect 選項後robusta會針對選定的壞味道案例蒐集資料，而蒐集資料的流程下一個小節，我將針對不同的壞味道進行介紹和分析

1.Dummy handler

1.取得壞味道的範圍

根據robusta的功能，在掃描完壞味道後，robusta會將壞味道的程式碼畫上螢光筆來告知使用者壞味道的範圍，此外robusta會將壞味道的相關資訊技入在該壞味道的marker中(圖中的燈泡),marker中記錄了該壞味道的methodInvication和 methodDeclration 如此一來，就可以蒐集到產生aspectJ的相關素材了

2.取得被標記的catch block 所接住的例外類別

根據Dummy handler 壞味道 robusta 會在 catch block的地方標記此處為 dummy handler壞味道，因此我們就可以知道若咬暴露這個壞味道帶來的影響，我們需要在aspectJ嵌入哪種例外，讓其例外發生

3.取得會發生與catch block中相同例外的 method invocation

再拿到catch block中會處理的了例外類型後，緊接著我們要來尋找try block中，一樣也會丟出與該壞味道相同例外類型的method invocation，用來當作嵌入對象

4.分析 try block 找出先被執行且與該catch block相同例外類型的 method invocation

　根據上一個了流程，我們若在try block中找到許多與catch block中會接住相同例外的的method invocation，我們只要針對第一個會丟出與catch相同例外且會被執行的method invocation 當做嵌入對象即可。取得該method invocation得時做方法為為利用visitor 將 try block中的method invocation都走訪一變，visitor 在尋訪每個astnode 時，能狗取得該node的程式結構，我們可以利用裡面結構的資訊，來得到該node是否會丟出例外以及他的例外類型

5.解析methodInvocation 相關資訊

根據上一個步驟，我們已經取得第一個會丟出與catch相同例外且會被執行的method invocation，接下來我們將進一步解吸，取得該methodinvoaction的函式名稱、丟出例外類型

6.解析methodDeclaration 相關資訊

接下來我們要把methoddeclaration的類別名稱、methoddeclaration名稱等相關資訊

7.組裝aspectJ 程式碼

完成以上步驟之後，我們就蒐集好組程aspectJ的素材，將其根據Dummy handler的特性下去拼裝，如同下圖程式碼所是

//note  
本研究提供一個方法，藉由產生測試案例的方式幫助使用者驗證例外處理壞味道是否符合預期的正常被處理。產生對應的測試案例之前，有以下幾個步驟需要準備。第一步:找出軟體中的壞味道;第二部:針對找到的壞味道設計測試案例;

第三步:修復程式碼中的味壞到直到測試通過。

3.1偵測壞味道

Java程式中，例外處理主要用於因應程式執行期間所碰到的例外狀況，其處理的方式跟程式強韌度有密切的關係。若在例外處理的實做沒有重視處理的方式，程式便有可能再遭遇例外的時候無法達成交付的任務.因此若可以找到程式中的例外處理壞味道，並將此壞味道到消除，將可提升程式的強見度

但是當程式的架構日益龐大，對開發人原來說查找程式碼中的例外處理壞味道的存在是件非常耗費人力的一件事，所以我們需要利用靜態分析工具來幫助開發者找到軟體中的壞味道

常見的靜態分析壞味道工具有 FindBugs,PMD,Robusta。根據陳奕帆的研究論文，robusta在偵測壞味道的精準度高於 findbugs 集 pmd，因此在設計單元測試之前將以robusta來偵測程式碼中的壞味道，除此之外robusta還能夠以報表的方式紀錄壞味在程式碼中的確切位置，方便使用者做快速的查找集檢視。

3.2壞味道設計測試案例

根據上一節的結果，我們可以透過robusta找到壞味道，本節將會介紹如何設計測試案例來驗證例壞處理壞味道的處理機製，跟據陳友綸的研究，

如之前所述，在利用壞味道偵測工具Robusta完成受測程式掃瞄之後，下一步工作便是檢視

Robusta 所標示的壞味道出現位置以及肇因區域。其次，透過丟出例外來揭露例外對程式產生

的影響；最後，比較ezScrum在遭遇例外時的行為與沒有例外出現時的正常行為。若前者會造

成程式無法正確運作而且沒有顯示錯誤訊息，我們便可判定該壞味道為程式缺陷。

為了讓例外處理壞味道的影響可以呈現，我們使用AspectJ[23]來模擬例外發生。AspectJ是

一種實作aspect oriented programming(AOP)技術的程式語言[24]，允許在既有的Java程式內

　　動態插入新的程式結構，如此我們便能夠在不改變原始碼的情況下，把程式的正常行為

與例外行為加以分隔。