# 摘要

論文名稱：利用AspectJ搭配測試案例曝露例外處理壞味道的影響

頁數：五十一頁

校所別：國立臺北科技大學 資訊工程 研究所

畢業時間：一百零六學年度 第二學期

學位：碩士

研究生：劉彥麟

指導教授：鄭有進教授、謝金雲教授

關鍵詞：Robusta、例外處理、壞味道、AspectJ、測試案例

含有例外處理壞味道的程式碼，存在著降低軟體強健度的風險。Robusta 為一個以Java開發的程式碼靜態分析工具，能夠偵測Java程式中的例外處理壞味道。

為了讓Robusta能夠呈現壞味道對軟體的影響，本論文藉由測試搭配AspectJ來重現例外狀況。當測試執行到選定的程式碼時，便可透過AspectJ嵌入例外來重現含有例外處理壞味道的路徑，進而呈現壞味道對於軟體的影響。

我們並以JfreeChart、Tomighty兩個開源專案做實驗對象。結果顯示，Robusta針對壞味道特性產生的AspectJ程式碼和測試案例，可以呈現壞味道對程式碼帶來的影響，也驗證壞味道的存在確有降低軟體品質的風險。

# ABSTRACT

Title: Applying AspectJ and Test Cases to Expose the Impact of Exception Handling Bad Smells

Pages: 51

School: National Taipei University of Technology

Department: Computer Science and Information Engineering

Time: June, 2018

Degree: Master

Researcher: Yen-Lin Liu Advisor: Chin-Yun Hsieh, Yu Chin Cheng

Keywords: Robusta, Exception handling, Bad Smell, AspectJ, Test cases

Bad smells in exception handling code may decrease the robustness of a program. Robusta is a static analysis tool which has been shown to be very useful in detecting exception handling bad smells in Java programs.

The impact of a bad smell needs to be exposed so that we can know if it is a real bug. In this thesis, we propose a method for exposing the impact of a bad smell by using AspectJ, that is, to dynamically inject a code to make an exception be thrown at a desired place so that the effect of the exception to the program can be revealed.

In this study, an empirical study has also been conducted by applying the proposed method to two open source software - JFreeChart and Tomighty. The result shows the proposed method can successfully expose the impact of exception handling bad smells detected by Robusta.

# 誌謝

兩年在軟體系統實驗室的時光過的飛快，感謝謝金雲老師、鄭有進老師在論文上的指導以及提供學習氣氛很好的研究環境，也從老師們身上學到如何把一個軟體做好。

感謝陳建村老師在論文口試中提供了許多建議，在經由老師所提出的建議後並思考其可行性，並為其做修改，讓論文更完整。

感謝家政學長在我寫論文及開發的時候給予建議和引導我釐清問題。

感謝同屆的同學們，我們一起學習、寫作業、趕專案的demo、mob programming，我想這些時光我畢業後一定會很懷念，謝謝大家。

最後要感謝我的家人，讓我無後顧之憂的努力學習，當我背後的後盾，讓我順利取得碩士學位。也要感謝晏綸，總是當一個很好的觀眾，聽著我說所有的大小事，給我支持和鼓勵，妳的陪伴讓人覺得很溫馨。

# 目錄

[摘要 i](#_Toc518989724)

[ABSTRACT ii](#_Toc518989725)

[誌謝 iii](#_Toc518989726)

[目錄 iv](#_Toc518989727)

[表目錄 vii](#_Toc518989728)

[圖目錄 viii](#_Toc518989729)

[第一章 緒論 1](#_Toc518989730)

[1.1研究背景與動機 1](#_Toc518989731)

[1.2 研究目標 1](#_Toc518989732)

[1.3 論文組織架構 2](#_Toc518989733)

[第二章 背景知識與相關研究 3](#_Toc518989734)

[2.1例外處理壞味道 3](#_Toc518989735)

[2.1.1 Dummy Handler 3](#_Toc518989736)

[2.1.2 Empty Catch Block 4](#_Toc518989737)

[2.1.3 Nested Try Statement 5](#_Toc518989738)

[2.1.4 Unprotected Main Program 6](#_Toc518989739)

[2.1.5 Exception Thrown From Finally Block 7](#_Toc518989740)

[2.1.6 Careless Cleanup 8](#_Toc518989741)

[2.2 Robusta 8](#_Toc518989742)

[2.3 AspectJ 9](#_Toc518989743)

[2.3 Abstract Syntax Tree Node(AST Node) 10](#_Toc518989744)

[第三章 研究方法 11](#_Toc518989745)

[3.1曝露壞味道的方法 11](#_Toc518989746)

[3.1.1 Dummy Handler 11](#_Toc518989747)

[3.1.2 Empty Catch Block 12](#_Toc518989748)

[3.1.3 Unprotected Main Program 13](#_Toc518989749)

[3.1.4 Exception Thrown From Finally Block 14](#_Toc518989750)

[3.1.5 Careless Cleanup 15](#_Toc518989751)

[第四章 設計與實作 16](#_Toc518989752)

[4.1 Dummy Handler Empty Catch Block 17](#_Toc518989753)

[4.1.1 產生AspectJ、測試檔案的素材收集 17](#_Toc518989754)

[4.1.2 實作Dummy Hander & Empty Catch Block搜集素材 18](#_Toc518989755)

[4.1.3 分析測試搭配AspectJ與程式碼的互動 21](#_Toc518989756)

[4.2 Unprotected Main Program 22](#_Toc518989757)

[4.2.1產生AspectJ、測試檔案的素材收集 22](#_Toc518989758)

[4.2.2實作Unprotected Main Program 23](#_Toc518989759)

[4.2.3分析測試搭配AspectJ與程式碼的互動 26](#_Toc518989760)

[4.3 Exception Thrown From Finally Block 27](#_Toc518989761)

[4.3.1產生AspectJ、測試檔案的素材收集 27](#_Toc518989762)

[4.3.2 實作Exception Thrown From Finally Block 29](#_Toc518989763)

[4.3.3分析測試搭配AspectJ與程式碼的互動 32](#_Toc518989764)

[4.4 Careless Cleanup 34](#_Toc518989765)

[4.4.1產生AspectJ、測試檔案的素材收集 34](#_Toc518989766)

[4.4.2實作Careless Cleanup 35](#_Toc518989767)

[4.4.3分析測試搭配AspectJ與程式碼的互動 38](#_Toc518989768)

[第五章 案例分析 40](#_Toc518989769)

[5.1 Unprotected Main Program 案例分析 40](#_Toc518989770)

[5.1.1偵測並分析Unprotected Main Program 40](#_Toc518989771)

[5.1.2呈現Unprotected Main Program 對系統的影響 40](#_Toc518989772)

[5.1.3消除Unprotected Main Program 42](#_Toc518989773)

[5.2 Exception Thrown From Finally Block案例分析與實作 43](#_Toc518989774)

[5.2.1偵測並分析Exception Thrown From Finally Block 43](#_Toc518989775)

[5.2.2呈現Exception Thrown From Finally Block 造成的影響 43](#_Toc518989776)

[5.2.3消除Exception Thrown From Finally Block 45](#_Toc518989777)

[5.3 Dummy Handler案例分析與實作 46](#_Toc518989778)

[5.3.1偵測並分析Dummy Handler 46](#_Toc518989779)

[5.3.2呈現Dummy Handler 造成的影響 46](#_Toc518989780)

[5.3.3消除Dummy Handler壞味道 48](#_Toc518989781)

[第六章 結論與未來研究方向 49](#_Toc518989782)

[6.1 結論 49](#_Toc518989783)

[6.2 未來展望 49](#_Toc518989784)

[參考文獻 50](#_Toc518989785)

# 表目錄

[表1 實作物件功能列表 16](#_Toc518989786)

# 圖目錄

[圖2-1 Dummy Handler 壞味道範例 3](#_Toc518989787)

[圖2-2 Empty Catch Block 壞味道範例 4](#_Toc518989788)

[圖2-3 Nested Try Statement 壞味道範例 5](#_Toc518989789)

[圖2-4 Unprotected Main Program 壞味道範例 6](#_Toc518989790)

[圖2-5 Exception Thrown From Finally Block壞味道範例 7](#_Toc518989791)

[圖2-6 Careless Cleanup壞味道範例 8](#_Toc518989792)

[圖2-7 AspectJ目標的程式碼 9](#_Toc518989793)

[圖2-9嵌入AspectJ後運行程式碼的結果 10](#_Toc518989794)

[圖2-10 AST Node結構之範例 10](#_Toc518989795)

[圖3-1 Dummy Handler 例外嵌入範例 11](#_Toc518989796)

[圖3-2 Empty Catch Block例外嵌入範例 12](#_Toc518989797)

[圖3-3 Unprotected Main Program例外嵌入範例 13](#_Toc518989798)

[圖3-4 Exception Thrown From Finally Block例外嵌入範例 14](#_Toc518989799)

[圖3-5 Careless Cleanup例外嵌入範例 15](#_Toc518989800)

[圖4-1 Dummy Handler 壞味道範例 17](#_Toc518989801)

[圖4-2 Dummy Handler的AspectJ 範例程式碼 17](#_Toc518989802)

[圖4-3 Dummy Handler搭配AspectJ的測試案例 18](#_Toc518989803)

[圖4-4 組成AspectJ程式碼、測試案例相關的Class Diagram 19](#_Toc518989804)

[圖4-5 組成AspectJ程式碼、測試案例相關的Sequence Diagram 20](#_Toc518989805)

[圖4-6 含有Dummy Handler壞味道的程式碼 21](#_Toc518989806)

[圖4-7 重現Dummy Handler例外處理壞味道發生例外的測試程式碼 21](#_Toc518989807)

[圖4-9 Unprotected Main Program壞味道範例 22](#_Toc518989808)

[圖4-10 Unprotected Main Program的AspectJ範例程式碼 23](#_Toc518989809)

[圖4-11 Unprotected Main Program搭配AspectJ的測試案例 23](#_Toc518989810)

[圖4-12組成AspectJ程式碼、測試案例相關的Class Diagram 24](#_Toc518989811)

[圖4-13組成AspectJ程式碼、測試案例相關的Sequence Diagram 25](#_Toc518989812)

[圖4-14含有Unprotected Main Program壞味道的程式碼 26](#_Toc518989813)

[圖4-15重現Unprotected Main Program壞味道發生例外的測試程式碼 26](#_Toc518989814)

[圖4-16根據Unprotected Main Program所製作的AspectJ程式 27](#_Toc518989815)

[圖4-17 Exception Thrown From Finally Block壞味道範例 27](#_Toc518989816)

[圖4-18 Exception Thrown From Finally Block的AspectJ 範例程式碼 28](#_Toc518989817)

[圖4-19 Exception Thrown From Finally搭配AspectJ的測試案例 28](#_Toc518989818)

[圖4-20組成AspectJ程式碼、測試案例相關的Class Diagram 30](#_Toc518989819)

[圖4-21組成AspectJ程式碼、測試案例相關的Sequence Diagram 31](#_Toc518989820)

[圖4-22含有Exception Thrown From Finally壞味道的程式碼 32](#_Toc518989821)

[圖4-23重現Exception Thrown From Finally壞味道發生例外的測試程式碼 33](#_Toc518989822)

[圖 4-24 根據Exception Thrown From Finally所製作的AspectJ程式 33](#_Toc518989823)

[圖4-25 Careless Cleanup壞味道範例 34](#_Toc518989824)

[圖4-26 Careless Cleanup AspectJ範例程式碼 34](#_Toc518989825)

[圖4-27重現Careless Cleanup壞味道發生例外的測試程式碼 35](#_Toc518989826)

[圖4-28組成AspectJ程式碼、測試案例相關的Class Diagram 36](#_Toc518989827)

[圖4-29組成AspectJ程式碼、測試案例相關的Sequence Diagram 37](#_Toc518989828)

[圖4-30含有Careless Cleanup壞味道的程式碼 38](#_Toc518989829)

[圖4-31重現Careless Cleanup壞味道發生例外的測試程式碼 38](#_Toc518989830)

[圖4-32根據Careless Cleanup所製作的AspectJ程式 39](#_Toc518989831)

[圖5-1 含有Unprotected Main Program壞味道案例程式碼 40](#_Toc518989832)

[圖5-2 程式正常運行結果 41](#_Toc518989833)

[圖5-3 Unprotected Main Program AspectJ 程式碼 41](#_Toc518989834)

[圖5-4 與AspectJ 相同規則產生對應嵌入符號的程式碼 41](#_Toc518989835)

[圖5-5 與AspectJ搭配的測試案例 42](#_Toc518989836)

[圖5-6 測試案例Failure結果圖 42](#_Toc518989837)

[圖5-7 測試案例Pass結果圖 42](#_Toc518989838)

[圖5-8含有Exception Thrown From Finally壞味道案例程式碼 43](#_Toc518989839)

[圖5-9 Exception Thrown From Finally Block AspectJ 程式碼 44](#_Toc518989840)

[圖5-10與AspectJ 相同規則產生對應嵌入符號的程式碼 44](#_Toc518989841)

[圖5-11與AspectJ搭配的測試案例 45](#_Toc518989842)

[圖5-12測試案例pass結果圖 45](#_Toc518989843)

[圖5-13含有Dummy Handler壞味道案例程式碼 46](#_Toc518989844)

[圖5-14 Dummy Handler AspectJ 程式碼 47](#_Toc518989845)

[圖5-15與AspectJ 相同規則產生對應嵌入符號的程式碼 47](#_Toc518989846)

[圖5-16與AspectJ搭配的測試案例 48](#_Toc518989847)

[圖5-17測試案例pass結果圖 48](#_Toc518989848)

# 第一章 緒論

## 1.1研究背景與動機

本篇論文以陳友倫[1]、廖振傑[2]的論文為研究背景，該兩篇論文指出由北科大軟體系統實驗室所開發的靜態分析工具Robusta[3]，能分析程式碼中的例外處理壞味道[4][5]，幫助開發者在開發Java程式時，提升其品質及強健度[4][5]。經由Robusta分析之後所得到之壞味道，若能夠證明確實是問題，便可以提升Robusta的價值。因此，陳友倫提出了以Aspect揭露導因於例外處理的程式缺陷的方法，以呈現因例外處理壞味道對軟體的影響。

Aspect[6]允許開發者在不修改程式碼的狀況下對程式嵌入例外程式碼，使其強制曝露出例外處理壞味道帶來的影響，但如果開發人員對Aspect語言特性不熟悉，則需額外花費成本來學習此語言。

若Robusta所定義的例外處理壞味道，都可以產生其對應揭露例外的方法，使開發者便可藉由強制丟出例外的過程，觀察因為例外處理壞味道對程式造成的影響。本篇論文將上述構想實作於Robusta上使其自動產生AspectJ程式碼，對於開發者來說學習AspectJ的門檻將大幅降低，也提升了Robusta的價值。

## 1.2 研究目標

本論文的目標即接續陳友倫針對Dummy Handler產生對應AspectJ提出的做法，將Robusta定義的壞味道設計出AspectJ嵌入例外來曝露壞味道的方法，且搭配測試案檢驗例外處理機制是否正確，並將其實作在Robusta上。讓Robusta能藉由自動產生測試案例和AspectJ程式呈現壞味道對於程式的影響，提升Robusta的價值。

## 1.3 論文組織架構

本論文共有六個章節，第一章說明本論文的研究動機與目的。第二章為本論文知識背景說明。第三章會說明本論文跟據陳友倫以Aspect揭露導因於例外處理的程式缺陷的研究，增加AspectJ對不同壞味道揭露例外的方法加以實現及設計。第四章會說明依據不同壞味道種類設計與實作嵌入例外的AspectJ程式碼及存在例外處理壞味道程式碼發生例外狀況時，例外處理是否正確的測試。第五章則是依據開源專案，呈現實作後的功能操作及曝露壞味道後對程式帶來的影響。第六章則是本論文的結論及未來展望。

# 第二章 背景知識與相關研究

## 2.1例外處理壞味道

### 2.1.1 Dummy Handler

定義:

「Dummy Handler」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。然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[4][5]。

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，開發者和使用者很難觀察到這些訊息。

範例:

如圖2-1所示， writeFile函式會創立一個新檔案接著把資料寫入該檔案中。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。



圖2-1 Dummy Handler 壞味道範例

### 2.1.2 Empty Catch Block

定義:

「Empty Catch Block」壞味道意指程式發生例外並捕捉例外後，忽略此例外[4][5]。

影響:

此作法會隱藏潛在問題，使開發者若遇到例外狀況發生，將難以除錯。

範例:

如圖2-2所示，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。若第12行的new FileWriter(“k:\\test.txt”) 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，此壞味道將被忽略，將會導致開發人員除錯困難度提升。



圖2-2 Empty Catch Block 壞味道範例

### 2.1.3 Nested Try Statement

定義:

「Nested Try Statement」壞味道意指在程式碼中存在著巢狀的Try Statement [4][5]。

影響:

對開發者而言，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:

如圖2-3所示，在第17行的Finally Block 會進行資源釋放的工作，然而很多關閉資源的函式都會丟出例外，用來代表釋放資源失敗。因此在Finally Block中會發生巢狀Try Statement 的情況，使得程式碼結構變得複雜及難以維護。



圖2-3 Nested Try Statement 壞味道範例

### 2.1.4 Unprotected Main Program

定義:

「Unprotected Main Program」壞味道意指在程式碼中，主程式或執行緒沒有捕捉由下傳遞至自己身上的例外[4][5]。

影響:

當程式執行時發生例外，主程式或執行緒沒有捕捉由下傳遞至自己身上的例外，則主程式或執行緒會不預期的終止或產生錯誤。

範例:

如圖2-4所示，56行為程式的main program，57到63行為main program的程式碼，若其中的method出現例外，因為沒有被Try Catch 包覆住，當程式發生未預期的錯誤時會導致系統被迫中止。



圖2-4 Unprotected Main Program 壞味道範例

### 2.1.5 Exception Thrown From Finally Block

定義:

「Exception Thrown From Finally Block」壞味道的特徵是在Finally Block中發生例外且此例外也被往外丟[4][5]。

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使開發人員不知道其它Block發生了例外。

範例:

如圖2-5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。此函式在第14行 FileWriter函式、15行 write、16行flush均會丟出IOException。此例外會被17行的Catch Block接住並向外拋出例外，但在拋出例外前會先進入19行的Finally Block，若此時21行的close函式也發生例外，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。



圖2-5 Exception Thrown From Finally Block壞味道範例

### 2.1.6 Careless Cleanup

定義:

「Careless Cleanup」壞味道意指在釋放資源前，發生例外導致資源無法被正常釋放[4][5]。

影響:

此壞味道因為在釋放資源之前發生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:

如圖2-6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。此函式在第14行 FileWriter函式、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。



圖2-6 Careless Cleanup壞味道範例

## 2.2 Robusta

Robusta為靜態分析Java程式碼的開源工具，協助開發人員偵測Java程式中存在的例外處理壞味道。目前已定義的六種例外處理壞味道，即為前述的Empty Catch Block、Dummy Handler 、Nested Try Statement、Unprotected Main Program、Careless Cleanup、Exception Thrown From Finally Block。Robusta會將偵測的結果以統計報表呈現。

## 2.3 AspectJ

AspectJ[6]是基於Java實做Aspect-Oriented Programming的程式語言，讓開發者能在不改變原有的程式碼狀況下額外增加程式碼的行為或改變狀態。將設計完的AspectJ程式與程式碼一起編譯並執行後，當程式執行到特定的函式時，會觸發AspectJ 嵌入程式碼。圖2-7為寫檔的範例，若想要在第14行writer.write執行前利用AspectJ嵌入程式碼，AspectJ程式碼需要撰寫如下圖2-8，需要以下步驟

* 時機點:before表示在執行目標函式前會嵌入AspectJ程式碼
* 目標:Call表示目標函式
* 目標所在位置:withincode表示目標函式在檔案中的所在位置
* 嵌入內容:期望利用AspectJ嵌入的程式碼

根據上面的步驟，圖2-8為AspectJ程式碼範例，嵌入的時機點為before，則表示在執行writer.write之前會嵌入程式碼，並且目標的範圍在Example.main中，最後嵌入的程式碼內容在圖2-8中的7~8行。圖2-7為 AspectJ目標的程式碼，執行main程式第14行時，AspectJ會嵌入了程式碼，如圖2-9運行程式碼的結果呈現嵌入的程式碼。



圖2-7 AspectJ目標的程式碼

圖2-8 AspectJ程式碼範例



圖2-9嵌入AspectJ後運行程式碼的結果

## 2.3 Abstract Syntax Tree Node(AST Node)

在Robusta[3]分析程式碼時採用第三方套件AST Node來幫助降低靜態分析的難度。透過AST Node，Robusta可以將程式碼解析成以樹狀結構來表示的抽象語法結構。樹狀的節點都對應程式碼中的一種結構。

以下為本論文中會廣泛出現到的名詞。

* MethodInvocation:函式中，呼叫程式執行的程式碼
* MethodDeclaration:函式中包含多個MethodInvocation、判斷式、變數宣告

圖2-10為AST Node結構之範例，ASTNodeExample為一個MethodDeclaration Node，包覆著一個呼叫的函式，ASTInvocation為一個呼叫的函式，其AST Node結構就是MethodInvocation。



圖2-10 AST Node結構之範例

# 第三章 研究方法

本論文以陳友倫[1]的研究為基礎，將其提出的方法實作在Robusta定義的壞味道中，讓所有的壞味道都能產生對應的AspectJ程式碼和測試，藉由測試搭配AspectJ嵌入例外來驗證例外處理行為是否正確，並將壞味道對於軟體的影響現形。

## 3.1曝露壞味道的方法

### 3.1.1 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。

若要讓壞味道對於軟體的影響現形(圖3-1第14~16行為Dummy Handler壞味道)，可利用測試搭配AspectJ在Try Block嵌入例外程式碼，使程式碼在Try Block中發生例外狀況並讓Catch Block來捕捉對應的例外狀況，藉此檢驗程式碼例外處理是否正確。



圖3-1 Dummy Handler 例外嵌入範例

### 3.1.2 Empty Catch Block

「Empty Catch Block」壞味道意指當程式發生例外時，會將例外捕捉卻以忽略例外的方式進行處理，此做法會隱藏潛在問題，會讓除錯難度提升也會降低程式的品質。

若要讓壞味道對於軟體的影響現形(圖3-2第14~15行為Empty Catch Block壞味道)，可利用測試搭配AspectJ在Try Block嵌入例外程式碼，讓程式碼在Try Block中發生例外狀況並讓Catch Block來捕捉對應的例外狀況，藉此檢驗程式碼例外處理是否正確。



圖3-2 Empty Catch Block例外嵌入範例

### 3.1.3 Unprotected Main Program

「Unprotected Main Program」壞味道意指在程式碼中，主程式或執行緒由下傳遞至身上的例外，因此當未被捕捉的例外往上傳遞，最終傳到主程式或執行緒而導致程式不預期的終止執行。

若要讓壞味道對於軟體的影響現形(圖3-3第69~74行為Unprotected Main Program壞味道)，可利用測試搭配AspectJ在main program嵌入例外程式碼，使main program中發生例外，藉此檢驗程式碼例外處理是否正確。



圖3-3 Unprotected Main Program例外嵌入範例

### 3.1.4 Exception Thrown From Finally Block

「Exception Thrown From Finally Block」壞味道意指在Finally Block中發生例外且被往外丟，造成 Finally Block 丟出的例外會覆蓋 Try Block 或 Catch Block 所丟出的例外，而使開發人員不知道其它Block發生了例外。

若要讓壞味道對於軟體的影響現形(圖3-4第27行為Exception Thrown From Finally Block壞味道)，首先可利用測試搭配AspectJ在Try Block嵌入例外程式碼，使程式碼在Try Block或是Catch Block中發生例外狀況；接著利用AspectJ在Finally Block嵌入例外程式碼，使程式碼在Finally Block也使其發生例外狀況，藉此檢驗程式碼例外處理是否正確。



圖3-4 Exception Thrown From Finally Block例外嵌入範例

### 3.1.5 Careless Cleanup

「Careless Cleanup」壞味道意指在釋放資源前發生例外，導致資源無法被正常釋放。

若要讓壞味道對於軟體的影響現形(圖3-5第177~178行為 Careless Cleanup壞味道)，可利用測試搭配AspectJ在釋放資源函式執行前嵌入例外程式碼，使程式碼在釋放資源前發生例外，讓釋放資源的函式無法被執行，藉此檢驗程式碼例外處理是否正確。



圖3-5 Careless Cleanup例外嵌入範例

# 第四章 設計與實作

本章節將會設計及實作第三章提出的方法，會新增以下物件，表1所示為實作物件功能列表說明。其中，AddAspectMarkerResolution\_[badSmellName]是繼承自Eclipse plugin的IMarkerResolution[7]而來，繼承之後允許開發者實作按下Eclipse 提示選單後的流程。Visitor皆是繼承自ASTVisitor[8]，因應不同程式碼結構來尋訪，幫助收集AspectJ、測試所需要的素材。

表1 實作物件功能列表

|  |  |
| --- | --- |
| 類別名稱 | 類別用途 |
| AddAspectMarkerResoluation\_[badSmellName] | 根據Visitor所蒐集到的資料建立AspectJ程式碼。 |
| FindAllTryStatementVisitor | 利用Visitor的尋訪，找出給予特定區塊程式碼中的所有Try Block。 |
| MethodInvovationCollectorVisitor | 利用Visitor尋訪，找出給予特定區塊的MethodInvocation。 |
| FindThrowSpecificExceptionStatementVisitor | 利用Visitor尋訪，找出給予特定區塊相同例外類型的MethodInvocation。 |

## 4.1 Dummy Handler Empty Catch Block

### 4.1.1 產生AspectJ、測試檔案的素材收集

根據3.1.1提出的方法，如圖4-1為含有Dummy Handler壞味道的範例，為了要產生圖4-2會嵌入例外程式碼的AspectJ和圖4-3搭配AspectJ的測試檔案，需要以下步驟來收集所需的素材。

1. 取得Dummy Handler壞味道的Catch Block捕捉例外的類型，如圖4-1第98行中Catch Block捕捉的SAXException
2. 取得Try Block中與步驟一會發生相同例外類型的MethodInvocation[9]，如圖4-1第93行的newSAXParser會拋出與Catch Block捕捉相同的例外類型
3. 取得該壞味道名稱，如圖4-1第87行的readPieDatasetFromXML
4. 取得該壞味道所在MethodDeclaration所屬的Class名稱



圖4-1 Dummy Handler 壞味道範例

壞味道所在的class及method名稱

捕捉例外的類型

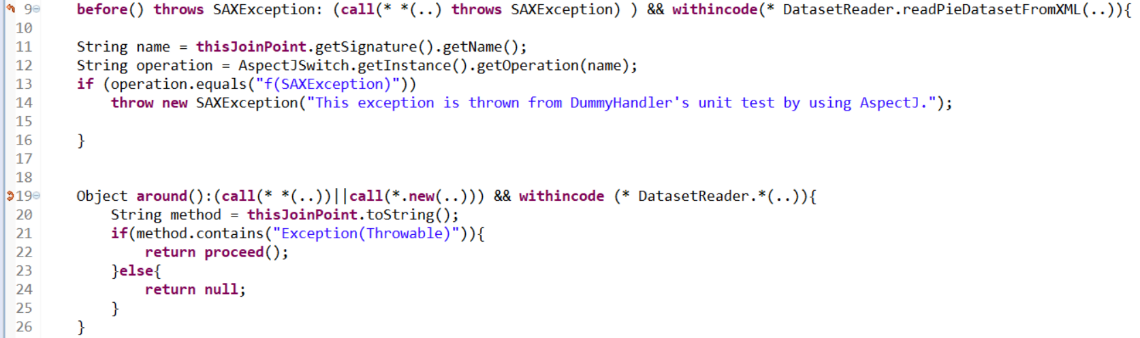


圖4-2 Dummy Handler的AspectJ 範例程式碼



跟catch block捕捉相同例外的MethodInvocation

圖4-3 Dummy Handler搭配AspectJ的測試案例

### 4.1.2 實作Dummy Hander & Empty Catch Block搜集素材

當使用者選擇Expose bad smell之後，AspectMarkerResolutionForDummy  
HandlerAndEmptyCatchBlock會透過visitor尋訪程式碼結構的過程，收集產生AspectJ和測試所需的素材。

1. 透過標記的IMarker[7]，如圖4-1第98行的燈泡，取得壞味道MethodDeclaration[9]、Dummy Handler壞味道所在程式碼的位置
2. 藉由取得壞味道在程式碼中所屬的Class名稱
3. 取得該MethodDeclaration內所有Try Statement
4. 藉由壞味道的位置取得所有Try Statement中，含有壞味道的目標Try Statement
5. 取得該目標Try Statement中Catch Block捕捉的例外處理類型
6. 利用Catch Block拿到的例外處理類型去找尋目標Try Statement內相同例外的MethodInvocation[9]
7. 從步驟六取得與Catch捕捉相同例外類型MethodInvocation

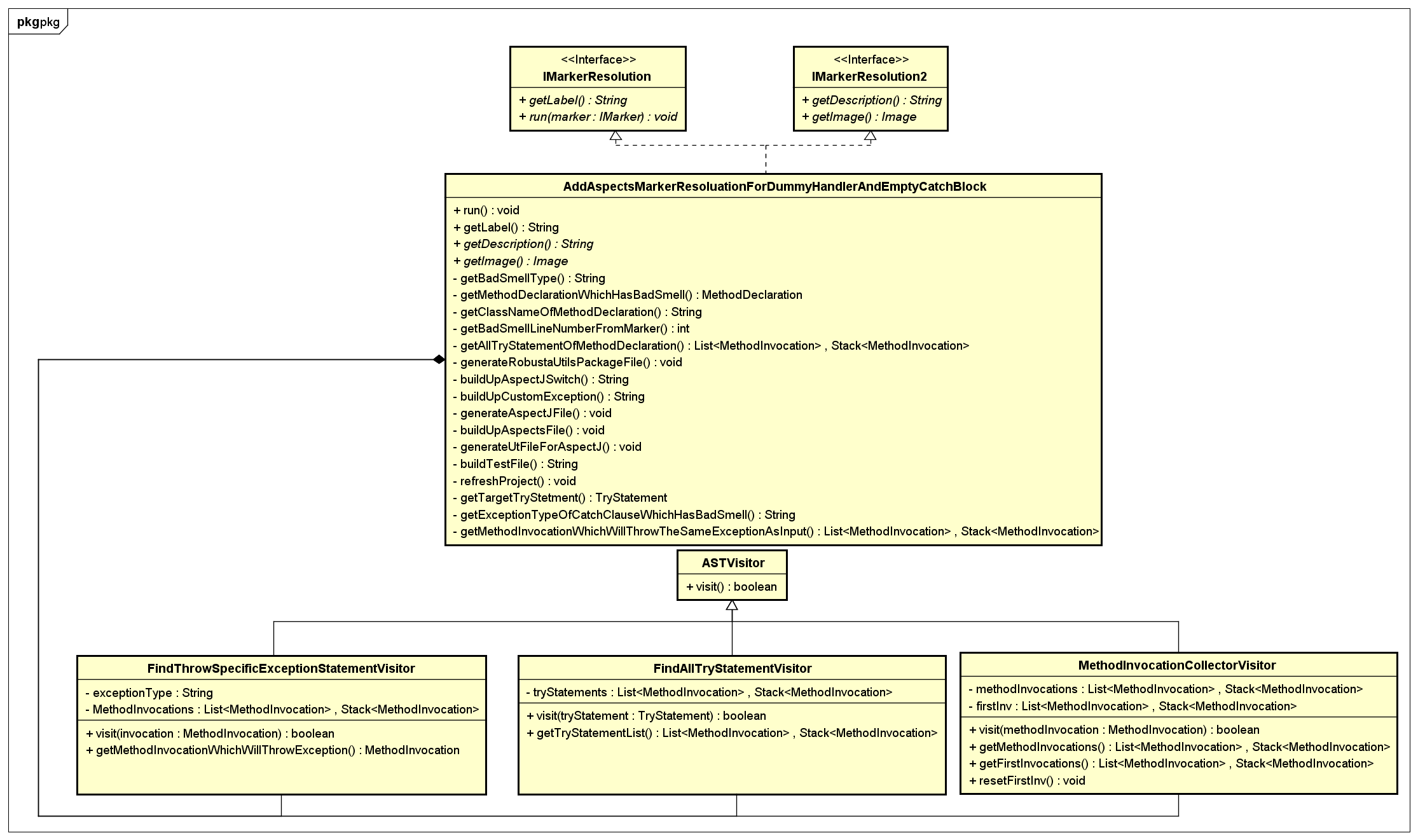


圖4-4 組成AspectJ程式碼、測試案例相關的Class Diagram

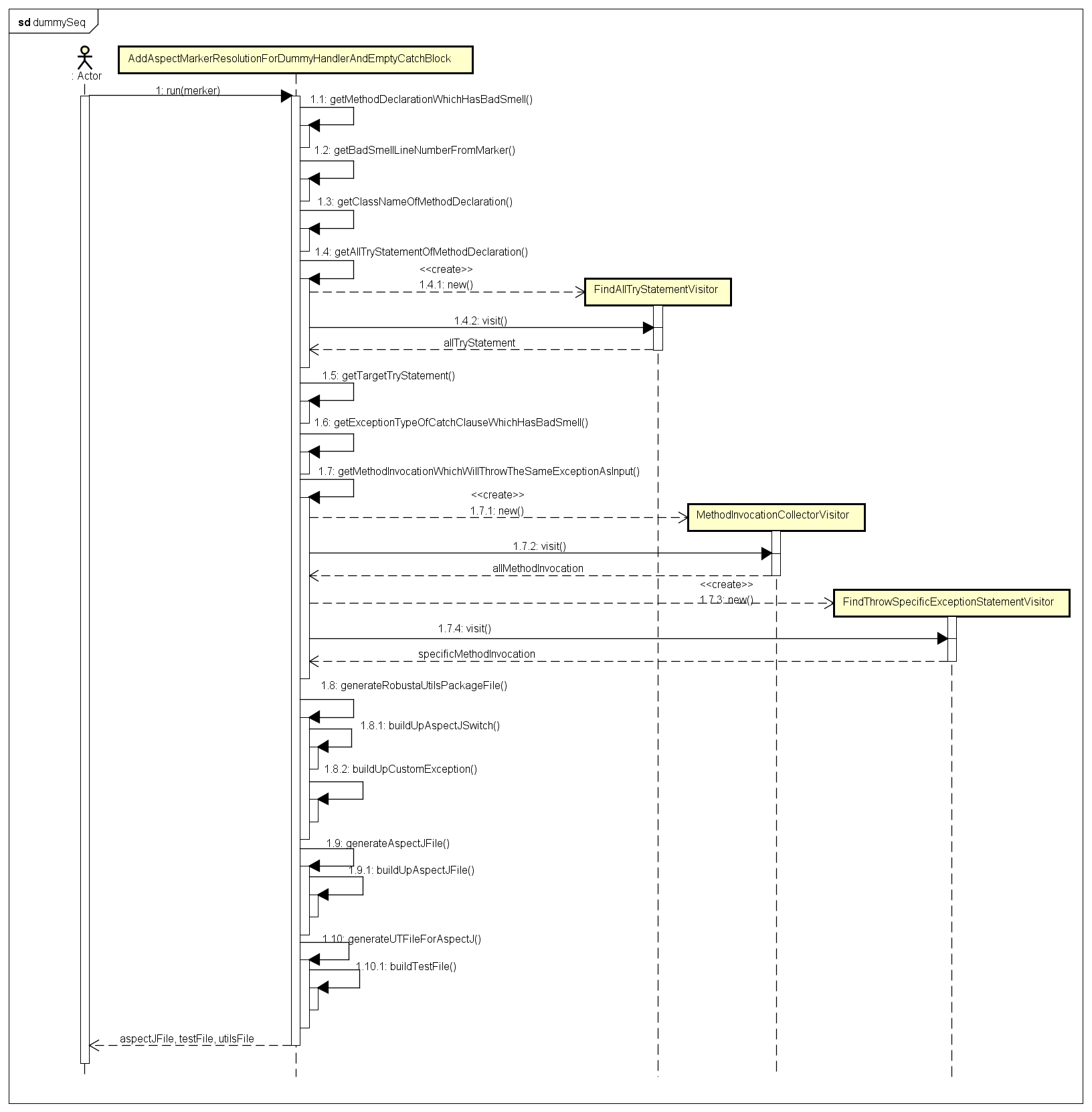


圖4-5 組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.1.3 分析測試搭配AspectJ與程式碼的互動

圖4-6為含有Dummy Handler壞味道的程式碼，若Dummy Handler、Empty Catch Block壞味道沒有被消除，表示程式在發生例外狀況時，採取的方式為印出錯誤訊息、忽略例外。

如圖4-8利用AspectJ在Try Block中第一個與Catch Block相同例外類型的MethodInvocation[9]嵌入例外程式碼，並運用AspectJ略過其他函式來到執行嵌入例外的函式。當測試案例如圖4-7執行時，會啟動AspectJ並嵌入例外程式碼，但含有例外處理壞味道的程式碼沒有將例外拋出來，測試因此Fail。

此方法可以還原程式碼發生例外的情境並檢驗例外處理的正確性。根據Robusta的建議，程式碼捕捉例外後需將例外拋出去，若拋出的例外會被測試案例的Catch Block捕捉，表示有對Dummy Handler、Empty Catch Block壞味道做處理，因此測試案例通過。



圖4-6 含有Dummy Handler壞味道的程式碼



圖4-7 重現Dummy Handler例外處理壞味道發生例外的測試程式碼

圖4-8 嵌入SAXException的AspectJ程式

## 4.2 Unprotected Main Program

### 4.2.1產生AspectJ、測試檔案的素材收集

根據3.1.3提出的方法，如圖4-9為含有Unprotected Main Program壞味道的範例，為了要產生圖4-10會嵌入例外程式碼的AspectJ和圖4-11搭配AspectJ的測試檔案，需要以下步驟來收集所需的素材。

1. 取得具有Unprotected Main Program壞味道的main 函式中的MethodIn

vocation名稱，如圖4-9第68~73行的MethodInvocation

1. 取得Unprotected Main Program壞味道的 MethodDeclaration[9]名稱，圖4-9第67行的main program
2. 取得Unprotected Main Program壞味道MethodDeclaration所屬的Class名稱



圖4-9 Unprotected Main Program壞味道範例



例外處理壞味道所在的MethodDeclaration

嵌入RuntimeException讓程式碼在任何情況下皆會發生例外

圖4-10 Unprotected Main Program的AspectJ範例程式碼

Main Program 第一個要被嵌入RuntimeException 的 MethodInvocation



例外處理壞味道所在的MethodDeclaration

圖4-11 Unprotected Main Program搭配AspectJ的測試案例

### 4.2.2實作Unprotected Main Program

當使用者選擇Expose bad smell時，AddAspectMarkerResolutionForUnprotect

edMain會透過visitor尋訪程式碼結構的過程，收集產生AspectJ和測試所需的素材。

1. 透過標記的IMarker取得壞味道在該Java文件MethodDeclaration、Unproetced Main Program所在程式碼的位置。
2. 藉由取得壞味道所在的MethodDeclaration拿到所屬的Class名稱
3. 取得該MethodDeclaration內所有的MethodInvocation
4. 取得MethodDeclaration中所有的Try Statement
5. 將不在Try Statement內的MethodInvocation收集起來



圖4-12組成AspectJ程式碼、測試案例相關的Class Diagram



圖4-13組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.2.3分析測試搭配AspectJ與程式碼的互動

圖4-14所示為含有Unprotected Main Program壞味道的程式碼，若Unprotected Main Program壞味道沒有被消除，則表示主程式未捕捉從下傳遞到自身的例外，則該程式會不預期的終止。

如圖4-16利用AspectJ在main program中對第一個MethodInvocation嵌入例外程式碼，並運用AspectJ略過其他函式來到執行嵌入例外的函式。因此，當測試案例如圖4-15執行的時候AspectJ便會啟動並嵌入例外。若存在壞味道則例外會被拋出來，來到該測試Catch Block的地方，測試因此Fail，此方法能夠還原程式碼發生例外的情境，藉此檢驗例外處理的正確性。若此壞味道被消除後，儘管在Main Program中出現例外狀況，但是因為Main Program被Try Catch包起來，將其例外進行捕捉，因此不會到達測試的Catch Block，根據設計的測試案例若程式有對其壞味道做處理，這個測試案例就通過了。



圖4-14含有Unprotected Main Program壞味道的程式碼



圖4-15重現Unprotected Main Program壞味道發生例外的測試程式碼



圖4-16根據Unprotected Main Program所製作的AspectJ程式

## 4.3 Exception Thrown From Finally Block

### 4.3.1產生AspectJ、測試檔案的素材收集

根據3.1.4提出的方法，如圖4-17為含有Exception Thrown From Finally Block壞味道的範例，為了要產生圖4-18會嵌入例外程式碼的AspectJ和圖4-19搭配AspectJ的測試案例，需要以下步驟來收集所需的素材。

1. 取得Exception Thrown From Finally Block壞味道的 MethodDeclaration名稱，圖4-17第301行的saveChartAsPNG
2. 取得Exception Thrown From Finally Block壞味道中Finally Block 會丟出例外的MethodInvocation名稱及例外類型，如圖4-17第311行具有會丟出例外的close函式
3. Try Block中會丟出例外的MethodInvocation名稱，如圖4-17第308行會丟出例外的writeChartAsPNG
4. 取得該壞味道MethodDeclaration所屬的Class名稱



圖4-17 Exception Thrown From Finally Block壞味道範例



Finally Blcok 會丟出例外的MethodInvocation及例外丟出類型

圖4-18 Exception Thrown From Finally Block的AspectJ 範例程式碼



例外處理壞味道所在的MethodDeclaration

Try、Finally Blcok會丟出例外的MethodInvocation和嵌入例外類型

圖4-19 Exception Thrown From Finally搭配AspectJ的測試案例

### 4.3.2 實作Exception Thrown From Finally Block

當使用者選擇Expose bad smell時，AddAspectMarkerResolutionForThrown  
FromFinally會透過visitor尋訪程式碼結構的過程，收集產生AspectJ和測試所需的素材。

1. 透過標記的IMarker取得壞味道MethodDeclaration、Exception Thrown From Finally Block壞味道所在程式碼的位置。
2. 藉由取得壞味道所在的MethodDeclaration程式碼中所屬的Class名稱
3. 取得該MethodDeclaration內所有Try Statement
4. 藉由例外處理壞味道的位置，取得所有Try Statement中含有例外處理壞味道的目標Try Statement
5. 從目標Try Statement的Finally Block中拿到壞味道所在的MethodInvocation
6. 從步驟五拿取其MethodInvocation Name、會丟出的例外類型
7. 從目標的Try Statement取得第一個會丟出例外的MethodInvocation



圖4-20組成AspectJ程式碼、測試案例相關的Class Diagram



圖4-21組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.3.3分析測試搭配AspectJ與程式碼的互動

根據圖4-22的程式碼，若是在進入Finally Block前發生了例外，且Finally Block本身也會發生例外，則造成Finally Block的例外會掩蓋Finally Block前所發生的例外。

如圖4-24利用AspectJ在程式碼內Try Block中writeChartAsPNG嵌入客製的CustomRobustaException，並在Finally Block中的close也嵌入例外。接著運用AspectJ略過其他函式來到執行嵌入例外的函式，測試案例實作結果如圖4-23所示，將會設計兩個Catch Block，一個為自定義的CustomRobustaException、另一個為Exception 類別的Catch Block。因此，當Try Block所產生的例外被Finally Block的例外覆蓋住時，會來到Exception的Catch Block造成測試Fail。此方法可以還原出程式碼壞味道發生例外的情境，藉此檢驗例外處理的正確性。若將程式碼的壞味道消除後，Robusta建議使用者不要在Finally Block中丟出例外，這個案例因為Try Block中丟出了例外，且沒有被Finally Block裡發生的例外覆蓋，所以被客製的Catch Block捕捉，因此這個測試案例就通過了。



圖4-22含有Exception Thrown From Finally壞味道的程式碼



圖4-23重現Exception Thrown From Finally壞味道發生例外的測試程式碼



圖 4-24 根據Exception Thrown From Finally所製作的AspectJ程式

## 4.4 Careless Cleanup

### 4.4.1產生AspectJ、測試檔案的素材收集

根據3.1.5提出的方法，如下圖4-25為含有Careless Cleanup壞味道的範例，為了要產生圖4-26會嵌入例外程式碼的AspectJ和圖4-27搭配AspectJ的測試案例，需要以下步驟來收集產生的素材。

1. 取得具有Careless Cleanup壞味道的函式中的釋放資源的MethodInvocation名稱，如圖4-25第180行的close
2. 取得Careless Cleanup壞味道的 MethodDeclaration名稱，圖4-25第167行的encode
3. 取得Careless Cleanup壞味道MethodDeclaration所屬的Class名稱



圖4-25 Careless Cleanup壞味道範例



圖4-26 Careless Cleanup AspectJ範例程式碼



圖4-27重現Careless Cleanup壞味道發生例外的測試程式碼

### 4.4.2實作Careless Cleanup

當選擇Expose bad smell時AddAspectMarkerResolutionForCarelessCleanup會透過visitor尋訪程式碼結構的過程，收集產生AspectJ和測試所需的素材。

1. 透過標記的IMarker取得壞味道MethodDeclaration、壞味道所在程式碼的位置。
2. 藉由取得壞味道所在的MethodDeclaration拿到所屬的Class名稱
3. 取得MethodDeclaration 中所有的Try Statement
4. 藉由例外處理壞味道的位置取得步驟3所有Try Statement中，含有例外處理壞味道的目標Try Statement
5. 利用目標的Try Statement和壞味道的位置取的釋放資源的MethodInvocation
6. 取得目標的Try Statement中第一個會丟出例外的MethodInvocation name、例外類型



圖4-28組成AspectJ程式碼、測試案例相關的Class Diagram



圖4-29組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.4.3分析測試搭配AspectJ與程式碼的互動

根據下圖4-30的程式碼，若在close之前發生了例外，會導致close不會被執行，而讓資源不會被正常釋放。

如圖4-32利用AspectJ[6]在程式碼中離釋放資源最遠且會發生例外的MethodInvocation嵌上其對應的例外，另外在釋放資源的MethodInvocation後嵌上檢查close是否被執行的判斷函式，並運用AspectJ略過其他函式來到執行嵌入例外的函式，測試案例實作結果如圖4-31所示，若壞味道存在則會來到自定義Exception的Catch Block而造成測試Fail。此方法可以還原出程式碼發生例外的情境，藉此檢驗例外處理的正確性。若將程式碼的壞味道消除後，儘管在釋放資源之前發生了例外，但仍然會進入Finally Block中執行釋放資源的函式，並嵌入AspectJ來確認釋放資源的動作沒有發生例外。根據設計的測試案例若程式有對其壞味道做處理，因此這個測試案例就通過了。



圖4-30含有Careless Cleanup壞味道的程式碼



圖4-31重現Careless Cleanup壞味道發生例外的測試程式碼



圖4-32根據Careless Cleanup所製作的AspectJ程式

# 第五章 案例分析

本章節將會以開源專案為範例，使用第三章所介紹的曝露壞味道的方法及第四章設計與實作後的成果來做實驗。

## 5.1 Unprotected Main Program 案例分析

### 5.1.1偵測並分析Unprotected Main Program

如圖5-1所示，為Tomighty.java[10]中的程式碼截圖。此段程式碼為程式的main program，根據Robusta的分析，此main program 並沒有被Try Block包覆起來，如果主程式或執行緒沒有捕捉由下拋出至身上的例外，則主程式或執行緒會不預期地終止執行，所以這是一種Unprotected Main Program的壞味道。



圖5-1 含有Unprotected Main Program壞味道案例程式碼

### 5.1.2呈現Unprotected Main Program 對系統的影響

當Tomighty正常啟動時，會開啟主程式並正常運作該程式，如圖5-2所示，為程式正常啟動後的結果。藉由Robusta產生了一個AspectJ程式來嵌入例外程式碼，強制曝露出壞味道並觀察其帶來的影響[1]。如下圖5-3，在第6行的地方定義丟出例外的時機點、要嵌入的Class、嵌入例外的method，第10行會丟出RuntimeException來曝露這個壞味道。完成AspectJ程式碼後，可以看到圖5-4的程式中產生與AspectJ規則相對應的符號，在對應的符號處會搭配測試案例來嵌入例外。



圖5-2 程式正常運行結果

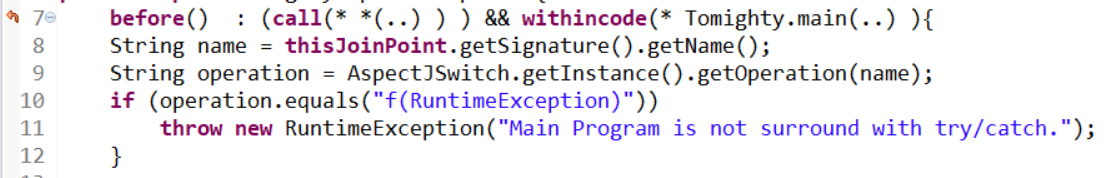


圖5-3 Unprotected Main Program AspectJ 程式碼



圖5-4 與AspectJ 相同規則產生對應嵌入符號的程式碼

藉由Robusta自動產生的測試能夠啟動AspectJ丟出例外，圖5-5第15行為要嵌入的MethodInvocation以及要丟出例外的類型。當測試呼叫了main程式而執行到圖5-4中的第58行時，AspectJ程式嵌入了RuntimeException使得Tomighty.main[10]主程式因為沒有對例外進行處理，所以程式便異常終止。便可將Unprotected Main Program這個壞味道帶來的影響給曝露出來，此外測試案例因沒有正確處理Unprotected Main Program壞味道而導致測試失敗。



圖5-5 與AspectJ搭配的測試案例

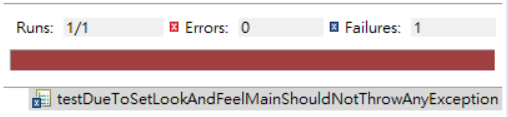


圖5-6 測試案例Failure結果圖

### 5.1.3消除Unprotected Main Program

根據楊雅雯論文[11]提供的快速修復功能移除壞味道，再次執行此測試，因為壞味道被消除了，儘管程式被嵌入例外程式碼，但因為外層被Try Statement包住，捕捉所有例外，並視需要記錄或在畫面上顯示清楚錯誤訊息，此外原先失敗的測試案例因為例外被正確處理而成功。

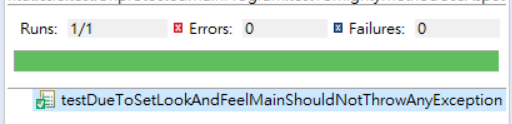


圖5-7 測試案例Pass結果圖

## 5.2 Exception Thrown From Finally Block案例分析與實作

### 5.2.1偵測並分析Exception Thrown From Finally Block

圖5-8所示為ChartUtils.java[12]中的程式碼截圖。此段程式碼為幫助使用者把chart轉存為png的檔案格式，根據Robusta的分析，此saveChartAsPNG 函式存在著Exception Thrown From Finally Block的壞味道。根據文獻探討，很多釋放資源的函式都以丟出例外的方式來代表釋放資源失敗，根據下圖5-8的程式碼範例，若Try Block 或是 Catch Block發生例外，而Finally Block也發生例外的話，Finally Block的例外則會掩蓋住Try Block 或是 Catch Block發生的例外，所以這是一種Exception Thrown From Finally Block壞味道。



圖5-8含有Exception Thrown From Finally壞味道案例程式碼

### 5.2.2呈現Exception Thrown From Finally Block 造成的影響

為了觀察例外發生對壞味道造成的影響，藉由Robusta產生AspectJ程式來嵌入例外程式碼，強制曝露出壞味道帶來的影響[1]，如下圖5-9，在第10、17行定義丟出例外的時機點、method的類型、要嵌入的Class、嵌入例外的method，第14行會丟出IOException，第21行會丟出CustomRobustaException來曝露出這個壞味道造成的影響。



圖5-9 Exception Thrown From Finally Block AspectJ 程式碼



圖5-10與AspectJ 相同規則產生對應嵌入符號的程式碼

藉由Robusta自動產生的測試能夠啟動AspectJ丟出例外，圖5-11第22行為要嵌入的MethodInvocation以及自定義的例外，此外在23行嵌入IOException的例外類型。當程式執行到圖5-10中的308行時，AspectJ程式嵌入了自定義的CustomRobustaException例外，接著執行Finally Block中311行的釋放資源函式，AspectJ嵌入了IOException程式碼，讓Try Block、Finally Block都發生例外，如此一來便會發生「例外蓋台」。因此藉由Robusta自動產生測試案例來執行含有壞味道的程式碼，若函式需要參數，則藉由分析資料型類別(Primitive type)為其產生參數的產生對應的預設值，讓測試可以呼叫此函式，如圖5-11為完整重現壞味道情境的測試案例，將Exception Thrown From Finally Block這個壞味道帶來的影響給曝露出來，且測試案例會因沒有正確處理Exception Thrown From Finally Block壞味道而導致測試失敗。



圖5-11與AspectJ搭配的測試案例

### 5.2.3消除Exception Thrown From Finally Block

根據楊雅雯論文[11]提供的修復功能移除壞味道後，再次執行此測試，因為壞味道被消除了，儘管Finally Block被嵌入例外程式碼，但Finally Block所產生的例外不會被往外拋，因此便不會造成例外蓋台的現象，並視需要記錄或在畫面上顯示清楚錯誤訊息。按此做法就可以消除此壞味道，此外原先失敗的測試案例因為例外被正確處理而成功。

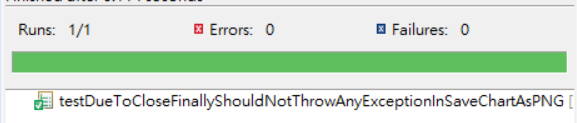


圖5-12測試案例pass結果圖

## 5.3 Dummy Handler案例分析與實作

### 5.3.1偵測並分析Dummy Handler

本範例是Dummy Handler壞味道程式碼，以JFreeChart[12]開源專案為例，如圖5-13為DataSetReader.java中的程式碼截圖。此段程式碼是幫助使用者在XML中，把Pie Chart格式的DataSet讀取出來，並回傳給使用者。根據Robusta的分析，程式碼中的93行會丟出 SAXException，並進入Catch Block中，但對其例外處理的方式為顯示錯誤訊息，並未對其做處理，所以這是一種Dummy Handler壞味道。



圖5-13含有Dummy Handler壞味道案例程式碼

### 5.3.2呈現Dummy Handler 造成的影響

為了觀察例外發生時因壞味道造成的影響，取得壞味道Catch Block中捕捉例外的類型，並根據捕捉例外的類型在Try Block找到對應的MethodInvocation並嵌入例外使其發生例外，藉由Robusta產生的AspectJ程式來嵌入例外，強制曝露出壞味道帶來的影響。如圖5-14，在第10行的地方定義丟出例外的時機點、method的類型、嵌入的Class、例外的method，第15行會丟出SAXException，來曝露出壞味道造成的影響。



圖5-14 Dummy Handler AspectJ 程式碼



圖5-15與AspectJ 相同規則產生對應嵌入符號的程式碼

藉由Robusta自動產生的測試能夠啟動AspectJ[6]丟出例外，圖5-16第31行為目標嵌入的MethodInvocation以及要丟出例外的類型。當程式執行到圖5-15第93行時，AspectJ程式嵌入了SAXException，使得readPieDatasetFromXML函式進入圖5-15第98行 Catch Block，便能夠把Dummy Handler這個壞味道帶來的影響給曝露出來。若函式需要參數，則藉由分析資料型類別(Primitive type)為其產生參數的產生對應的預設值，讓測試可以呼叫此函式，如圖5-16為完整重現壞味道情境的測試案例。當readPieDatasetFromXML函式發生例外時，處理的方式為印出錯誤訊息，並沒有做實質上的處理且掩蓋例外的發生，造成最終的result為null，並將此錯誤的狀態傳遞下去，導致圖5-16測試案例中需要使用到該result的SaveChartAsJPEG無法正常運作，進而存到一張資訊錯誤的圖。此外測試案例會因沒有正確處理Dummy Handler壞味道所以導致測試失敗。



圖5-16與AspectJ搭配的測試案例

### 5.3.3消除Dummy Handler壞味道

根據楊雅雯論文[11]提供的修復功能移除壞味道後，再次執行此測試，因為壞味道被消除了，所以儘管Try Block被嵌入例外程式碼，但因為Catch Block將捕捉到的例外往外拋，讓外層去處理該例外，用此做法就可以消除此壞味道。因此被程式碼拋出的例外會進到圖5-16中測試案例Catch Block並進行驗證，原先失敗的測試案例會因為例外被正確處理而成功。

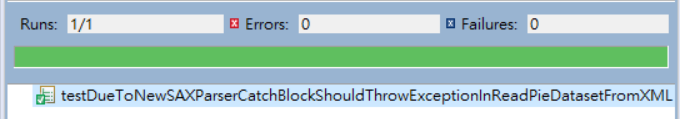
****

圖5-17測試案例pass結果圖

# 第六章 結論與未來研究方向

## 6.1 結論

根據陳友倫的論文[1]，提出以Robusta[3]產生的AspectJ[6]程式碼能發揮功效，讓例外處理壞味道對於軟體的影響被揭露出來。其論文也指出，透過Robusta的協助，儘管不熟悉AspectJ撰寫方式，也幫助開發者產生對應的AspectJ程式碼。本論文把除了Dummy Handler以外的例外處理壞味道，分別為Empty Catch Block、Careless Cleanup、Exception Thrown From Finally Block、Unprotected Main Program的壞味道，設計並實作其對應的AspectJ程式搭配測試案例，更改啟動的AspectJ的使用時機，因此不會對正在運行的程式碼造成影響，而限制在測試階段即可完成。

本論文已經實做在Robusta中，並對開源軟體JFreeChart[12]、Tomighty[10]進行曝露壞味道，且能夠成功的揭露壞味道帶來的影響。

## 6.2 未來展望

目前Robusta能夠分析Java程式中是否含有例外處理壞味道、產生失敗的測試案例提供重構功能，未來則有以下兩點可以被實作：

1. 將Robusta工具化在不同編輯器上

目前Robusta為在Eclipse編輯器上的工具，但撰寫Java的編輯器還有許多種，以IntelliJ IDEA為例，它也提供了很多plugin供開發者使用，若可以讓Robusta支援其他知名的編輯器，將可以廣泛的推廣Robusta。

1. 將Robusta概念開發在其他語言上。

目前Robusta是針對Java程式語言的例外處理壞味道做分析，若Robusta的概念運用在有例外處理機制的程式語言中，將可以支援更多程式語言。

# 參考文獻

1. 陳友倫，以Aspect揭露導因例外處理的程式缺陷，碩士論文，國立臺北科技大學資訊工程系碩士班，台北，2016.
2. 廖振傑，透過偵測及移除例外處理壞味道提升軟體強健度:以ezScrum為例，碩士論文，國立臺北科技大學資訊工程系碩士班，台北，2016.
3. Robusta at Eclipse Markerplace, Availbale: <https://Markerplace.eclipse.org/content/robusta-eclipse-plugin> [Accessed: 2017-06-02].
4. 陳建村，爪哇例外處理:模型、重構、與樣式，博士論文，國立臺北科技大學機電科技研究所博士班，台北，2008
5. Chien-Tsun Chen. 例外處理設計的逆襲. 悅知文化，2014
6. AspectJ at Eclipse, Available: https://eclipse.org/aspectj/ [Accessed: 2017-06-02].
7. Eclipse IMarker Document, Available: http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fresources%2FIMarker.html[Accessed: 2017-06-02].
8. ASTVisitor Document, Available: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTVisitor.html[Accessed: 2017-06-02].
9. AST Method document, Available: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FAST.html[Accessed: 2017-06-02].
10. Tomighty, Available: http://tomighty.org/ [Accessed: 2017-06-02].
11. 楊雅雯，利用Robusta消除例外處理壞味道，碩士論文，國立台北科技大學資訊工程系碩士班，台北，2018
12. JFreeChart, Available: <http://www.jfree.org/index.html> [Accessed: 2017-06-02].
13. Throwable document, Available: https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html [Accessed:2017-06-02].
14. 洪哲瑋，例外處理程式壞味道的自動化偵測與重構，碩士論文，國立臺北科技大學資訊工程系碩士班，台北，2009