

**資訊工程系碩士班**

**碩士學位論文**

**利用Robusta消除例外處理壞味道**

Removing Exception Handling Bad Smells Using Robusta.

**研究生：楊雅雯**

**指導教授：鄭有進教授、謝金雲教授**

**中華民國一百零七年六月**

# 摘　要

論文名稱：利用Robusta消除例外處理壞味道

頁數：43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：楊雅雯

指導教授：鄭有進教授、謝金雲教授

關鍵詞：例外處理、壞味道、強健度、重構

移除例外處理壞味道，可以提升軟體的強健度。然而，例外處理壞味道的移除，是相當具有挑戰性的工作，對於經驗不足的開發人員尤然。

針對各種不同的JAVA 程式例外處理壞味道，本論文提供一系列以重構為基礎的處理壞味道移除參考範例，以導引開發者移除例外處理壞味道。我們並將相關作業程序實作在Robusta中，開發人員可透過Robusta自動化重構程式碼來消除壞味道，以提升軟體的強健度。

# ABSTRACT

Title: Removing Exception Handling Bad Smells Using Robusta

Pages: 43

School: National Taipei University of Technology

Department: Information Engineering

Time: June, 2018

Degree: Master

Researcher: Ya-Wun Yang

Advisor: Chin-Yun Hsieh Ph.D., Yu Chin Cheng Ph.D.

Keywords: Exception Handling, Code Smells, Robustness, Refactoring

Removing exception handling bad smells can improve robustness of software. However, writing exception handling code without bad smells is challenging task, especially for inexperienced developers.

In order to handle different bad smells of exception for Java programs. In this thesis, we propose a series of refactoring-oriented example to remove bad smells in exception handling code in Java to lead developers to remove exception handling bad smells. We implement the related operating procedure in Robusta tool. Developers can use Robusta tool to automatically refactor the code which has exception handling bad smells and improve software’s robustness.

# 致謝

在軟體系統實驗室的這兩年，很感謝鄭有進老師、謝金雲老師提供了很好的學習環境，讓我有機會參與各種專案來增加自己的實作經驗，也能夠將課堂學到的知識應用在實作中。並且在專案或是論文遇到問題時，兩位老師也都提供我們許多寶貴的建議，讓我們能夠在迷茫中找到方向。

感謝李家政學長，當團隊開發遇到問題時，謝謝學長總是提供方向或是引導我們去找尋答案，讓功能能夠順利完成。

感謝實驗室的同學們，當我在專案或是課堂作業有問題的時候，總是協助我並且教導我。也在閒暇之餘一起出去吃飯、聊天、打桌遊，讓我有個充實的研究所生活。

感謝我的家人、親戚和朋友，謝謝家人支持我唸研究所，並提供生活費讓我能夠無憂無慮地完成這兩年的碩士學位；謝謝姑姑有回高雄的時候總是會送一些家裡的東西來給我；謝謝姨婆常常來看我，和煮好吃的料理給我吃；謝謝朋友偶爾陪我一起吃飯、聊天，雖然見面的時間不多，但是跟你們相處的時光很開心；謝謝哥哥總是在我生活遇到任何問題的時候來幫助我排除，也常常帶我出去玩。

最後，我想要謝謝隊友劉彥麟同學，在碩二的這一年你教了我很多，不論是軟工課的專案、Robusta、TCSE、或是找工作面試的一些問題，謝謝你一直帶著我，讓我在這兩年的研究所中學到了很多知識跟經驗。

# 目錄

[摘　要 i](#_Toc516084797)

[ABSTRACT ii](#_Toc516084798)

[目錄 iii](#_Toc516084799)

[表目錄 vi](#_Toc516084800)

[圖目錄 vii](#_Toc516084801)

[第一章 緒論 1](#_Toc516084802)

[1.1 研究背景與動機 1](#_Toc516084803)

[1.2 研究目標 1](#_Toc516084804)

[1.3 論文組織架構 1](#_Toc516084805)

[第二章 背景知識 3](#_Toc516084806)

[2.1 Robusta 3](#_Toc516084807)

[2.2例外處理壞味道 3](#_Toc516084808)

[2.2.1 Empty Catch Block 3](#_Toc516084809)

[2.2.2 Dummy Handler 4](#_Toc516084810)

[2.2.3 Unprotected Main Program 5](#_Toc516084811)

[2.2.4 Nested Try Statement 5](#_Toc516084812)

[2.2.5 Careless Cleanup 6](#_Toc516084813)

[2.2.6 Exception Thrown From Finally Block 6](#_Toc516084814)

[2.3強健度等級 7](#_Toc516084815)

[等級0：未定義(Undefined) 7](#_Toc516084816)

[等級1：錯誤回報(Error reporting) 8](#_Toc516084817)

[等級2：狀態回復(State recovery) 8](#_Toc516084818)

[等級3：行為重試(Behavior recovery) 9](#_Toc516084819)

[2.4 Abstract Syntax Tree 9](#_Toc516084820)

[第三章 研究方法 11](#_Toc516084821)

[3.1 壞味道消除方法介紹 11](#_Toc516084822)

[3.1.1 Dummy Handler & Empty Catch Block 11](#_Toc516084823)

[3.1.2 Unprotected Main Program 15](#_Toc516084824)

[3.1.3 Nested Try Statement 16](#_Toc516084825)

[3.1.4 Careless Cleanup 19](#_Toc516084826)

[3.1.5 Exception Thrown From Finally Block 20](#_Toc516084827)

[3.2 過去與現在Robusta快速修復與重構差異 21](#_Toc516084828)

[3.2.1 快速修復 21](#_Toc516084829)

[3.2.2 重構 22](#_Toc516084830)

[3.3 設計與實作 23](#_Toc516084831)

[3.3.1 快速修復 23](#_Toc516084832)

[3.3.2 重構 31](#_Toc516084833)

[3.4 壞味道的偵測、曝露及消除流程 38](#_Toc516084834)

[第四章 應用實例 40](#_Toc516084835)

[4.1 Dummy Handler應用實例 40](#_Toc516084836)

[4.1.1偵測Dummy Handler 40](#_Toc516084837)

[4.1.2產生曝露Dummy Handler壞味道的測試案例 41](#_Toc516084838)

[4.1.3消除Dummy Handler壞味道 42](#_Toc516084839)

[4.2 Careless Cleanup應用實例 44](#_Toc516084840)

[4.2.1偵測Careless Cleanup 44](#_Toc516084841)

[4.2.2產生曝露Careless Cleanup壞味道的測試案例 44](#_Toc516084842)

[4.2.3消除Careless Cleanup壞味道 45](#_Toc516084843)

[4.3 Exception Thrown From Finally Block應用實例 47](#_Toc516084844)

[4.3.1偵測Exception Thrown From Finally Block 47](#_Toc516084845)

[4.3.2產生曝露Exception Thrown From Finally Block壞味道的測試案例 47](#_Toc516084846)

[4.3.3消除Exception Thrown From Finally Block壞味道 48](#_Toc516084847)

[4.4 Unprotected Main Program應用實例 50](#_Toc516084848)

[4.4.1 偵測Unprotected Main Program 50](#_Toc516084849)

[4.4.2產生曝露Unprotected Main Program壞味道的測試案例 50](#_Toc516084850)

[4.4.3消除Unprotected Main Program壞味道 51](#_Toc516084851)

[第五章 結論與未來展望 53](#_Toc516084852)

[5.1 結論 53](#_Toc516084853)

[5.2 未來展望 53](#_Toc516084854)

[參考文獻 54](#_Toc516084855)

# 表目錄

[表3-1、Robusta快速修復功能差異 21](#_Toc516222814)

[表3-2、Robusta重構功能差異 22](#_Toc516222815)

# 圖目錄

[圖2-1、Empty Catch Block範例 4](#_Toc516222816)

[圖2-2、Dummy Handler範例 4](#_Toc516222817)

[圖2-3、Unprotected Main Program範例 5](#_Toc516222818)

[圖2-4、Nested Try Statement範例 5](#_Toc516222819)

[圖2-5、Careless Cleanup範例 6](#_Toc516222820)

[圖2-6、Exception Thrown From Finally Block範例 7](#_Toc516222821)

[圖2-7、等級0：未定義 7](#_Toc516222822)

[圖2-8、等級1：錯誤回報 8](#_Toc516222823)

[圖2-9、等級2：狀態回復 8](#_Toc516222824)

[圖2-10、等級3：行為重試成功 9](#_Toc516222825)

[圖2-11、等級3：行為重試失敗 9](#_Toc516222826)

[圖2-12、分析Java檔後的AST view樹狀結構 10](#_Toc516222827)

[圖3-1、Dummy Handler壞味道範例 11](#_Toc516222828)

[圖3-2、Dummy Handler壞味道快速修復結果 12](#_Toc516222829)

[圖3-3、Dummy Handler壞味道重構結果 12](#_Toc516222830)

[圖3-4、Dummy Handler壞味道重新修正後的快速修復結果 13](#_Toc516222831)

[圖3-5、選單提供的Unchecked Exception 14](#_Toc516222832)

[圖3-6、Dummy Handler壞味道重新修正後的重構結果 14](#_Toc516222833)

[圖3-7、Unprotected Main Program壞味道快速修復結果 15](#_Toc516222834)

[圖3-8、錯誤物件的繼承架構 16](#_Toc516222835)

[圖3-9、Unprotected Main Program壞味道改善後的快速修復結果 16](#_Toc516222836)

[圖3-10、選取Eclipse 提供的Refactor→Extract Method…功能 17](#_Toc516222837)

[圖3-11、在Extract Method視窗替函式命名 17](#_Toc516222838)

[圖3-12、Nested Try Statement壞味道重構結果 18](#_Toc516222839)

[圖3-13、Robusta提供的Extract Method功能 18](#_Toc516222840)

[圖3-14、Careless Cleanup壞味道快速修復結果 19](#_Toc516222841)

[圖3-15、Exception Thrown From Finally Block壞味道原先的重構結果 20](#_Toc516222842)

[圖3-16、Exception Thrown From Finally Block壞味道改善後的重構結果 21](#_Toc516222843)

[圖3-17、Dummy Handler和Empty Catch Block QuickFix功能的Class Diagram 24](#_Toc516222844)

[圖3-18、Dummy Handler和Empty Catch Block QuickFix功能的Sequence Diagram 25](#_Toc516222845)

[圖3-19、Unprotected Main Program QuickFix功能的Class Diagram 27](#_Toc516222846)

[圖3-20、Unprotected Main Program QuickFix功能的Sequence Diagram 28](#_Toc516222847)

[圖3-21、Careless Cleanup QuickFix功能的Class Diagram 30](#_Toc516222848)

[圖3-22、Careless Cleanup QuickFix功能的Sequence Diagram 31](#_Toc516222849)

[圖3-23、Dummy Handler和Empty Catch Block Refactoring功能的Class Diagram 33](#_Toc516222850)

[圖3-24、Dummy Handler和Empty Catch Block Refactoring功能的Sequence Diagram 33](#_Toc516222851)

[圖3-25、Nested Try Statement Refactoring功能的Class Diagram 35](#_Toc516222852)

[圖3-26、Nested Try Statement Refactoring功能的Sequence Diagram 36](#_Toc516222853)

[圖3-27、Exception Thrown From Finally Block Refactoring功能的Class Diagram 37](#_Toc516222854)

[圖3-28、Exception Thrown From Finally Block Refactoring功能的Sequence Diagram 38](#_Toc516222855)

[圖3-29、壞味道偵測、曝露及消除流程圖 39](#_Toc516222856)

[圖4-1、JFreeChart中Dummy Handler壞味道範例 41](#_Toc516222857)

[圖4-2、JFreeChart Dummy Handler壞味道測試案例 42](#_Toc516222858)

[圖4-3、JFreeChart Dummy Handler壞味道測試失敗 42](#_Toc516222859)

[圖4-4、Robusta提供自動化消除Dummy Handler壞味道 43](#_Toc516222860)

[圖4-5、JFreeChart Dummy Handler壞味道消除的結果 43](#_Toc516222861)

[圖4-6、JFreeChart readPieDatasetFromXML函式正確處理例外後測試成功 43](#_Toc516222862)

[圖4-7、JFreeChart中Careless Cleanup壞味道範例 44](#_Toc516222863)

[圖4-8、JFreeChart Careless Cleanup壞味道測試案例 45](#_Toc516222864)

[圖4-9、JFreeChart Careless Cleanup壞味道測試失敗 45](#_Toc516222865)

[圖4-10、Robusta提供自動化消除Careless Cleanup壞味道 46](#_Toc516222866)

[圖4-11、JFreeChart Careless Cleanup壞味道消除的結果 46](#_Toc516222867)

[圖4-12、JFreeChart encode函式正確處理例外後測試成功 46](#_Toc516222868)

[圖4-13、JFreeChart中Exception Thrown From Finally Block壞味道範例 47](#_Toc516222869)

[圖4-14、JFreeChart Exception Thrown From Finally Block壞味道測試案例 48](#_Toc516222870)

[圖4-15、JFreeChart Exception Thrown From Finally Block壞味道測試失敗 48](#_Toc516222871)

[圖4-16、Robusta提供自動化消除Exception Thrown From Finally Block壞味道 49](#_Toc516222872)

[圖4-17、JFreeChart Exception Thrown From Finally Block壞味道消除的結果 49](#_Toc516222873)

[圖4-18、JFreeChart saveChartAsPNG函式正確處理例外後測試成功 49](#_Toc516222874)

[圖4-19、Tomighty工具 50](#_Toc516222875)

[圖4-20、Tomighty中Unprotected Main Program壞味道範例 50](#_Toc516222876)

[圖4-21、Tomighty Unprotected Main Program壞味道測試案例 51](#_Toc516222877)

[圖4-22、Tomighty Unprotected Main Program壞味道測試失敗 51](#_Toc516222878)

[圖4-23、Robusta提供自動化消除Unprotected Main Program壞味道 51](#_Toc516222879)

[圖4-24、Tomighty Unprotected Main Program壞味道消除的結果 52](#_Toc516222880)

[圖4-25、Tomighty main program正確處理例外後測試成功 52](#_Toc516222881)

[圖5-1、兩層Try Statement的Careless Cleanup壞味道範例 53](#_Toc516222882)

# 第一章 緒論

本章節將先介紹本論文的研究背景與動機；接著描述本論文的預期目標；最後介紹本論文的組織架構。

## 1.1 研究背景與動機

例外處理為程式碼在執行過程中，遇到例外狀況時所做的處理方式。在Java程式碼中，常見的例外處理機制為try-catch-finally。當try、catch或finally中含有壞味道時，容易因為不適當的例外處理而導致錯誤發生，降低系統的強健度。正確的例外處理設計及實作能夠讓軟體在遭遇例外時，也能正常執行功能。然而，對經驗不足的開發人員而言，容易因為沒有相關知識而設計出含有例外處理壞味道[1]的程式碼。

撰寫出正確的例外處理程式碼是一件困難的事，因此為了協助開發人員能夠正確處理例外行為，根據陳建村等人的研究[2]，提出以強健度等級作為例外處理等級或能力的依據，透過重構程式碼[3]來消除例外處理壞味道，並將其功能實作於Robusta[4]中且使其自動化。藉由Robusta來幫助開發人員發掘程式碼中例外處理壞味道，並且以自動化重構的方式來消除這些壞味道，進而提升軟體品質。

## **1.2 研究目標**

在前人的研究中，Robusta裡定義的壞味道不斷地被修正及更新，其偵測壞味道的功能也愈來愈準確。然而，部分例外處理壞味道被偵測出來後，其對應的重構方法尚未被實作，或是還有能夠改善的地方。因此，本論文將對Robusta重構的功能進行改善，並將部分壞味道尚未實作的重構功能補齊，讓Robusta藉由重構消除例外處理壞味道的功能更加齊全。

## **1.3 論文組織架構**

本論文分為五個章節，第一章是緒論，描述本論文的背景與動機。第二章會介紹與本論文相關的背景知識。第三章為研究方法，會介紹消除例外處理壞味道的方法，並說明本論文是如何設計及實作於Robusta中。第四章則會運用本論文提供的壞味道消除方法在實際案例應用以提升程式的強健度。最後一章為本論文的結論與未來展望。

# **第二章 背景知識**

## **2.1 Robusta**

Robusta[4]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，也能夠以自動化修改程式碼的方式消除這些壞味道。在Eclipse[5]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。而在Robusta定義了六種例外處理壞味道[6]，分別為：

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

Java編譯器要求開發者對程式碼中會丟出Checked Exception的函式做處理，處理的方法有兩種：在介面宣告會丟出例外或用try/catch將函式包覆起來。為了不造成「介面演進」[1]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，會造成例外被掩蔽的現象，稱之為Empty Catch Block壞味道[6]。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2-1、Empty Catch Block範例

### 2.2.2 Dummy Handler

Dummy Handler壞味道[6]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裡只有記錄例外訊息，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被印出的例外訊息，也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，除了會造成開發人員不容易除錯，也會讓程式碼繼續在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2-2、Dummy Handler範例

### 2.2.3 Unprotected Main Program

當例外被丟出後都沒有被catch捕捉時，這些例外都會被往上層回報。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，稱之為Unprotected Main Program壞味道[6]。對使用者而言，程式不預期地終止是軟體品質不佳的表現。



圖2-3、Unprotected Main Program範例

### 2.2.4 Nested Try Statement

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[6]。雖然Nested Try Statement不會對程式碼造成影響，也沒有程式邏輯的錯誤，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2-4、Nested Try Statement範例

### 2.2.5 Careless Cleanup

Careless Cleanup壞味道[6]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，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會導致資源被消耗並且降低系統穩定度。如下圖2-5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2-5、Careless Cleanup範例

### 2.2.6 Exception Thrown From Finally Block

一段有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，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 From Finally Block壞味道[6]。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2-6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，接著執行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2-6、Exception Thrown From Finally Block範例

## **2.3強健度等級**

在陳建村的研究[7]中，提出包含四個強健度等級的例外處理模型，作為規劃與判斷軟體元件例外處理等級或能力的依據。強健度的四個等級，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[7]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。因此當例外發生時，可能導致系統發生錯誤也可能使系統繼續正常執行，無法確實掌握。

如圖2-7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。



圖2-7、等級0：未定義

### 等級1：錯誤回報(Error reporting)

錯誤回報[7]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2-8所示，當Component A發生例外時，如果不進行任何處理，會直接將例外回報給上層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處理例外或記錄例外訊息。



圖2-8、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。[7]

如圖2-9所示，Component A發生例外後，將例外回報給上層Component B；接著，Component B再將例外回報給上層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，再將例外回報給上層ComponentD，由最上層Component 處理例外或記錄例外訊息。



圖2-9、等級2：狀態回復

### 等級3：行為重試(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。[7]

如圖2-10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2-11所示，則會將例外向上層回報，由最上層Component處理例外或記錄例外訊息。



圖2-10、等級3：行為重試成功



圖2-11、等級3：行為重試失敗

## **2.4 Abstract Syntax Tree**

Abstract Syntax Tree簡稱AST[8]，是Eclipse JDT[9]裡的一部份，用來表達Java檔案的結構。另外，Eclipse[5]提供AST view[10]的plugin，能夠將Java檔案轉換為樹狀結構，表達Java程式碼的語法架構。本論文消除壞味道的快速修復功能和重構功能，都是透過AST提供的ASTParser[11]來分析Java程式碼的結構，藉由AST提供的走訪方式來走訪節點，並使用ASTRewrite[12]來對目標節點進行修改，最後寫回Java檔中來變更程式碼內容。常見的AST節點有：

* Method Invocation：method的節點在AST中稱為Method Invocation。
* Try Statement：當程式碼遇到可能丟出例外的Method Invocation時，Java編譯器會要求開發者將該Method Invocation用try/catch保護起來，或是在介面宣告該Method Invocation會丟出的例外型別。如果開發者用try/catch保護的話，此try/catch的節點在AST中稱為Try Statement。Try Statement包含了try block、catch clause和finally block。
* Method Declaration：一段程式碼的區塊，包含程式碼界面和區塊中程式碼執行的內容，在AST中稱為Method Declaration，而Method Declaration index為Method Declaration在該Java文件的排序。

如圖2-12所示，為AST在程式碼中的範圍和AST view樹狀結構的範例。



圖2-12、分析Java檔後的AST view樹狀結構

# 第三章 研究方法

本章節中，會先介紹要如何消除這些例外處理壞味道；然後，整理了Robusta[4]過去與現在消除壞味道功能的差異；接著，將本論文提出的新增功能與改善的方法實作於Robusta中；最後，介紹本論文所提供的例外處理壞味道消除流程。

## 3.1 壞味道消除方法介紹

消除例外處理壞味道的方法為延續洪哲瑋的論文[3]。在Dummy Handler、Empty Catch Block、 Unprotected Main Program和Nested Try Statement中，會先介紹洪哲偉論文中所提供的方法是如何消除且實作於Robusta中。瞭解消除的方法後，介紹這些壞味道在Robusta中，哪些功能與當時提出的方法不一致或可以修正和改善的地方，並對其重新修正和改善；在Careless Cleanup，本論文增加它的消除方法；在Exception Thrown From Finally Block，會先介紹Robusta提供的重構功能，再介紹我們對此功能進行了什麼改善。最後，將本論文提出的功能實作於Robusta中，使其能夠自動化消除程式碼中例外處理的壞味道。

### 3.1.1 Dummy Handler & Empty Catch Block

根據洪哲瑋論文中提到的Ignore Checked Exception，在楊智傑的論文[13]中被正式定義為Empty Catch Block。而Empty Catch Block的消除方式與Dummy Handler一樣，於是本論文將Dummy Handler 和 Empty Catch Block放在本小節中一起敘述。如圖3-1所示，以下將以Dummy Handler為範例。

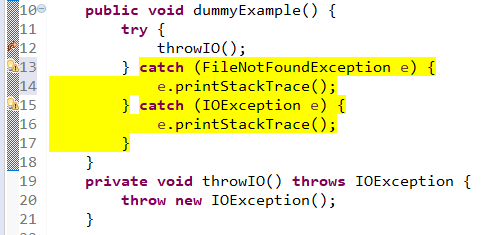


圖3-1、Dummy Handler範例

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，提供的功能為快速修復功能及重構功能。而快速修復和重構兩種功能的差異點在於重構功能比較具有彈性，部分的參數能夠讓使用者自己決定，而快速修復的功能比較沒有彈性，使用者沒有辦法做更動。

* 快速修復功能：

如圖3-2第15、17行所示，當catch捕捉到例外後，透過Robusta[4]的快速修復功能，自動於catch區塊中捕捉到的例外轉換為RuntimeException，將例外回報給上一層。

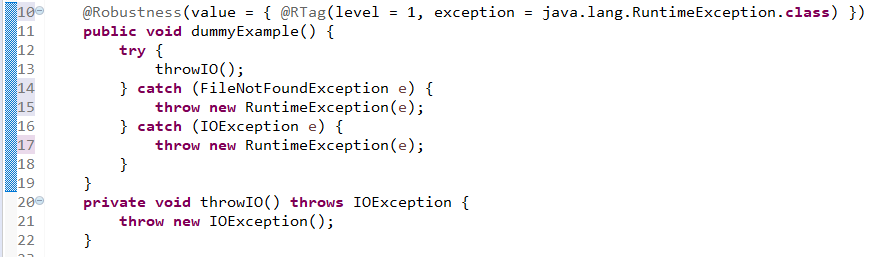


圖3-2、Dummy Handler壞味道快速修復結果

* 重構功能：

如圖3-3第16、18行所示，當使用者希望能夠丟出的例外型別為自己所定義的，而非上述所提到的RuntimeException時，洪哲瑋的論文說明能夠透過Robusta所提供的選單，來選擇自己所希望拋出的例外型別來重構。

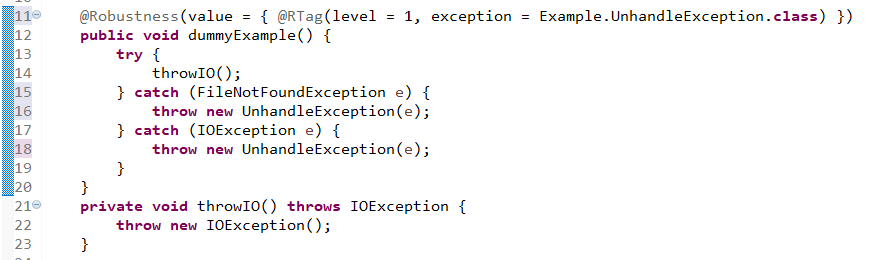


圖3-3、Dummy Handler重構結果

我們發現Robusta提供消除壞味道的功能與當時洪哲瑋論文所提出的不太一致，Robusta提供的功能有三種，分別為：

1. 快速修復功能­：丟出例外
2. 快速修復功能­：丟出RuntimeException
3. 重構功能：丟出Unchecked Exception

在方法3的重構功能中，丟出的Unchecked Exception預設為RuntimeException，我們認為方法2的快速修復功能­與方法3的重構功能丟出的例外都屬於Unchecked Exception，因此功能相似，所以將方法2拿掉，留下快速修復功能與重構功能，並根據現有功能來重新修正提供的功能。

* 重新修正後的快速修復功能：

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。如圖3-4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3-4、Dummy Handler重新修正後的快速修復結果

* 重新修正後的重構功能：

我們發現Robusta[4]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Dummy Handler和Empty Catch Block提供的重構功能為丟出Unchecked Exception來進行回報。如圖3-5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CustomRobustaException為使用者自己定義的Unchecked Exception。如圖3-6所示，透過回報Unchecked Exception來消除壞味道後不會在介面進行宣告。

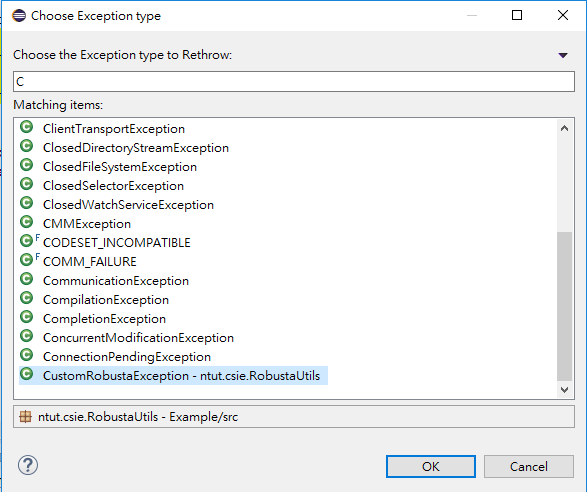


圖3-5、選單提供的Unchecked Exception

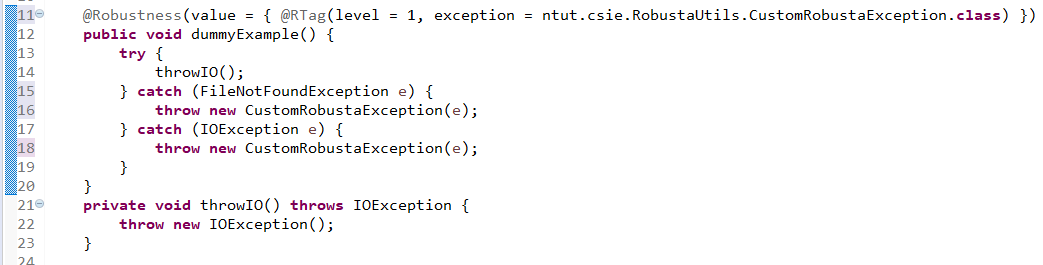


圖3-6、Dummy Handler重新修正後的重構結果

### 3.1.2 Unprotected Main Program

根據洪哲瑋的論文[3]，消除Unprotected Main Program[6]的方法是利用快速修復的功能用try/catch將其main program保護起來。如圖3-7所示，快速修復功能為在main program中用try/catch將main program保護起來，並且在捕捉Exception[14]型別的例外，最後在catch區塊中標示//TODO: handle exception來提醒使用者要處理這個例外。



圖3-7、Unprotected Main Program壞味道快速修復結果

根據洪哲偉的論文，我們發現此壞味道的快速修復功能會產生兩種問題。

第一種問題為：雖然透過快速修復功能將Unprotected Main Program消除了，但會產生Empty Catch Block。消除Unprotected Main Program的方法為用try/catch將main program保護住，如果還不清楚catch要如何處理例外的話，能夠先將catch區塊中例外處理以顯示錯誤或記錄到日誌檔的方式來處理，最後結束程式執行。因此我們將catch區塊中的//TODO: handle exception改為將例外訊息記錄到日誌檔中，讓main program的catch區塊有記錄例外訊息。

第二種問題為：根據Unprotected Main Program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3-8為錯誤物件的繼承架構圖，雖然catch有去捕捉Exception型別的例外，但如果系統發生Error[15]時，main program還是會因為沒有捕捉Error[15]而造成系統異常終止。因此我們將catch捕捉的Exception[14]改為Throwable[16]，如圖3-9為Unprotected Main Program改善後的快速修復結果。



圖3-8、錯誤物件的繼承架構

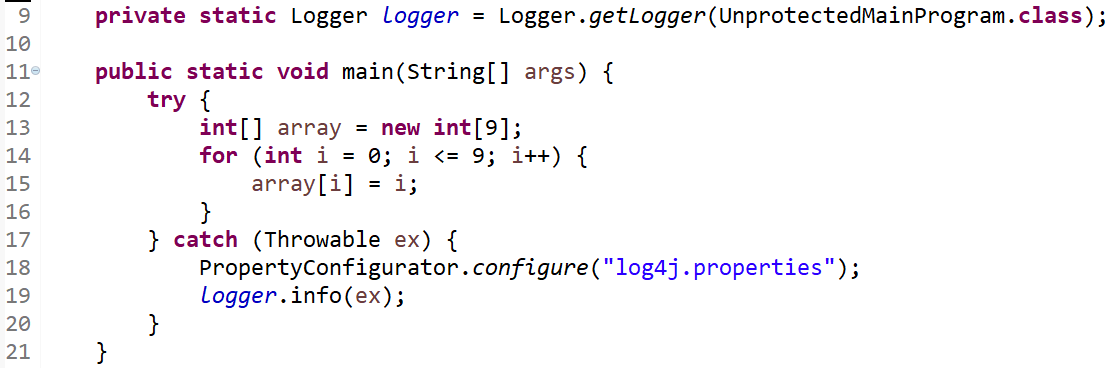


圖3-9、Unprotected Main Program改善後的快速修復結果

### 3.1.3 Nested Try Statement

Nested Try Statement[6]為Try Statement[8]裡的try、catch或finally區塊中又有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[3]，因為Eclipse已經提供重構功能，所以不在Robusta中提供重構功能。因此如果要消除Nested Try Statement，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命名要抽成的函式名稱，點擊OK後Nested Try Statement就消除了。如圖3-10所示，選取第9~15行的Try Statement結構，[8]選取Extract Method；接著如圖3-11所示，在視窗中的Method name填寫要命名的函式名稱；最後如圖3-12所示，Nested Try Statement就被消除了。



圖3-10、選取Eclipse 提供的Refactor→Extract Method…功能



圖3-11、在Extract Method視窗替函式命名



圖3-12、Nested Try Statement重構結果

我們發現在Robusta[4]中有提供Nested Try Statement[6]的自動化重構功能，雖然重構後的結果是一樣的，但Robusta會與Eclipse[5]的Extract Method功能做結合，使Nested Try Statement的重構功能自動化，因此使用者不用再圈選出要獨立抽成函式的Try Statement。如圖3-13所示，點擊Robusta的「Refactor==>Extract Method」後，Robusta會自動連結Eclipse的Extract Method功能，而顯示如圖3-11的Extract Method視窗，填寫函式名稱後，如圖3-12所示，Nested Try Statement就被消除了。



圖3-13、Robusta提供的Extract Method功能

### 3.1.4 Careless Cleanup

Careless Cleanup[6]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。因此要消除Careless Cleanup，首先在有壞味道的Try Statement增加finally 區塊；接著，為了避免程式碼在執行釋放資源時發生意外，讓例外在finally區塊被丟出，而造成Exception Thrown From Finally，因此在finally block裡加入try/catch保護程式碼，並在catch block中將捕捉資源釋放所發生的例外訊息記錄到日誌檔中；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3-14所示，為Careless Cleanup[6]快速修復後的結果，雖然會產生Nested Try Statement，但再使用Robusta[4]提供的重構功能來消除即可。



圖3-14、Careless Cleanup快速修復結果

### 3.1.5 Exception Thrown From Finally Block

當finally block裡的程式碼丟出例外時，會覆蓋掉原先try或catch 區塊中所丟出的例外，而誤導開發人員或維護人員關注在finally block[8]丟出的例外，而忽略try或catch 區塊中所丟出的例外，稱為Exception Thrown From Finally Block。[6]

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 From Finally Block，首先將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，印出例外訊息；最後，將finally裡的這個Try Statement獨立抽成一個函式即可。如圖3-15所示，為Robusta對Exception Thrown From Finally Block的重構結果。消除Exception Thrown From Finally Block後，雖然會產生Dummy Handler，但在Exception Thrown From Finally Block跟Dummy Handler這兩種壞味道對程式碼影響的權衡之下，我們認為留下Dummy Handler對程式碼的影響相對來說是比較小的，因此才會在Exception Thrown From Finally Block消除後而殘留下Dummy Handler。



圖3-15、Exception Thrown From Finally Block原先的重構結果

雖然透過Robusta消除了Exception Thrown From Finally Block[6]，但衍生的Dummy Handler仍然會在Robusta[4]提供的壞味道分析報表中顯示。為了避免使用者對殘留下來的Dummy Handler抱有疑慮，如圖3-16所示，我們將殘留的Dummy Handler增加一些註解，向使用者解釋為何我們在這裡留下Dummy Handler，降低使用者的困惑。此外，我們也將印出例外訊息的部分修改為將例外訊息記錄到日誌檔中。

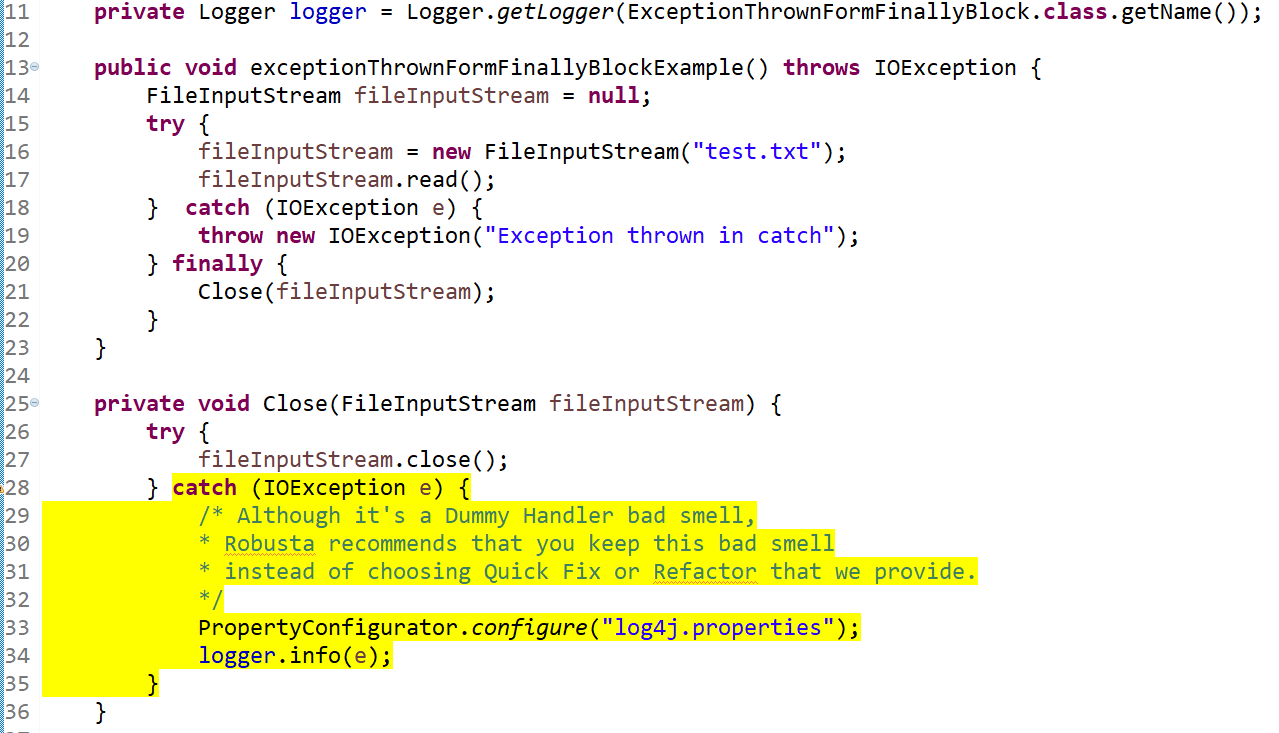


圖3-16、Exception Thrown From Finally Block改善後的重構結果

## 3.2 過去與現在Robusta快速修復與重構差異

本小節整理Robusta提供的快速修復和重構功能，將Robusta原本存在的快速修復與重構功能，和本論文新增或改善的快速修復與重構功能做比較。[3]

### 3.2.1 快速修復

表3-1、Robusta快速修復功能差異

|  |  |  |
| --- | --- | --- |
| 壞味道種類 | 原本的Robusta | 本論文改善後提供的方法 |
| Dummy Handler  &  Empty Catch Block | catch捕捉到的例外後丟出RuntimeException來進行回報。 | 將catch捕捉到的例外型別直接丟出來進行回報。 |
| Unprotected Main Program | 產生try/catch保護主程式，且catch捕捉Exception類別後不做任何事。 | 產生try/catch保護主程式，catch捕捉Throwable類別並將例外寫入日誌中。 |
| Careless Cleanup | 無 | 增加快速修復功能。 |

### 3.2.2 重構

表3-2、Robusta重構功能差異

|  |  |  |
| --- | --- | --- |
| 壞味道種類 | 原本的Robusta | 本論文改善後提供的方法 |
| Dummy Handler  &  Empty Catch Block | 使用者能夠選擇自己定義的例外類別來將例外丟出。 | 使用者能夠選擇Unchecked例外類別，包含使用者定義的Unchecked例外，來將例外丟出。 |
| Nested Try Statement | 與Eclipse的Extract Method做連結，自動化重構程式碼來消除壞味道。 | 無 |
| Exception Thrown From Finally Block | 將釋放資源的函式用try/catch包住，並且Extract Method，讓使用者自己定義獨立出來的函式名稱，自動化重構程式碼來消除壞味道。 | 改善Robusta自動化重構功能，將獨立出來的函式修正為catch捕捉例外後，將例外寫入日誌檔中，並增加註解，解釋為什麼這裡留下Dummy Handler。 |

## 3.3 設計與實作

在Robusta[4]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，並針對不同的壞味道來提供快速修復或重構的功能。

### 3.3.1 快速修復

在Dummy Handler、Empty Catch Block、Unprotected Main Program和Careless Cleanup中，有提供快速修復(QuickFix)的功能來消除壞味道。當使用者點擊警告訊息要消除壞味道時，RLQuickFixer會根據使用者所選擇的壞味道種類來提供對應的快速修復方法。

#### 3.3.1.1 Dummy Handler & Empty Catch Block

當使用者點擊Dummy Handler 和 Empty Catch Block的Quick Fix功能時，RLQuickFixer類別會觸發DummyQuickFix類別，執行快速修復的功能來消除壞味道。

1. 透過標記的IMarker[17]來取得壞味道在該Java文件的MethodDeclaration index。
2. 設定AST[8]相關資訊取得CompilationUnit。
3. 藉由取得的MethodDeclaration index和CompilationUnit取得壞味道所在的MethodDeclaration。
4. 取得該MethodDeclaration中指定壞味道所在的catch資訊。
5. 將MethodDeclaration的介面宣告指定catch所接住的例外型別。
6. 檢查catch是否有印出例外訊息的方法，若有則移除。
7. 將catch捕捉到的例外向上層回報出去。

完成上述步驟即可完成Dummy Handler 和 Empty Catch Block[6]自動化快速修復功能。



圖3-17、Dummy Handler和Empty Catch Block QuickFix功能的Class Diagram



圖3-18、Dummy Handler和Empty Catch Block QuickFix功能的Sequence Diagram

#### 3.3.1.2 Unprotected Main Program

當使用者點擊Unprotected Main Program[6]的Quick Fix功能時，RLQuickFixer類別會觸發UMQuickFix類別，執行快速修復的功能來消除壞味道。

1. 透過標記的IMarker[17]來取得壞味道在該Java文件的MethodDeclaration index。[8]
2. 設定AST相關資訊取得CompilationUnit。
3. 藉由取得的MethodDeclaration index和CompilationUnit取得壞味道所在的MethodDeclaration，此MethodDeclaration即為主程式。
4. 產生try/catch將主程式保護住，在這一步驟中，首先會先蒐集主程式內所有的Try Statement，而將主程式分為兩種情境：主程式內有Try Statement和主程式內沒有Try Statement。
   * 主程式內沒有Try Statement：
5. 產生try/catch，並在catch捕捉Throwable類別[16]，並將捕捉到的例外記錄日誌檔中。
6. 將主程式內的程式碼移入try區塊裡。
   * 主程式內有Try Statement：[8]
7. 檢查catch是否有捕捉Throwable[16]類別或是Exception[14]類別，如果沒有，則增加捕捉Throwable類別的catch區塊，並將例外捕捉後記錄日誌檔中。
8. 將不在try/catch裡的程式碼移入try/catch中。

完成上述步驟即可完成Unprotected Main Program[6]自動化快速修復功能。



圖3-19、Unprotected Main Program QuickFix功能的Class Diagram



圖3-20、Unprotected Main Program QuickFix功能的Sequence Diagram

#### 3.3.1.3 Careless Cleanup

當使用者點擊Careless Cleanup[6]的Quick Fix功能時，RLQuickFixer類別會觸發CCQuickFix類別，執行快速修復的功能來消除壞味道。

* 1. 透過標記的IMarker[17]來取得壞味道在該Java文件的MethodDeclaration index。[8]
  2. 設定AST[8]相關資訊取得CompilationUnit。
  3. 藉由取得的MethodDeclaration index和CompilationUnit取得壞味道所在的MethodDeclaration。
  4. 分析IMarker[17]，取得壞味道的行數。
  5. 藉由壞味道的行數和MethodDeclaration取得釋放資源的函式。
  6. 判斷MethodDeclaration內是否Try Statement，如果有Try Statement，則

1. 取得釋放資源函式會丟出的例外型別。
2. 在MethodDeclaration的介面宣告釋放資源函式會丟出的例外。
   1. 產生Try Statement，並將釋放資源的函式放入finally block中。
   2. 將原來釋放資源的函式移除。
   3. 將MethodDeclaration內剩下的函式移入Try Statement。

完成上述步驟即可完成Careless Cleanup[6]自動化快速修復功能。



圖3-21、Careless Cleanup QuickFix功能的Class Diagram



圖3-22、Careless Cleanup QuickFix功能的Sequence Diagram

### 3.3.2 重構

在Dummy Handler、Empty Catch Block、Nested Try Statement和Exception Thrown From Finally Block[6]中提供了壞味道消除的重構的功能。當使用者點擊警告訊息要消除壞味道時，RLQuickFixer會根據使用者所選擇的壞味道種類來提供對應的重構方法。

#### 3.3.2.1 Dummy Handler & Empty Catch Block

當使用者在Dummy Handler和Empty Catch Block[6]選擇警告訊息提供的重構方法時，RLQuickFixer會執行RethrowUncheckExAction的類別，啟動重構頁面讓使用者設定相關參數和要回報的例外型別等。RethrowExWizard繼承Eclipse的RefactoringWizard類別，因此能夠提供預覽畫面協助使用者設定重構的相關資訊。在使用者設定完成後，RethrowExInputPage會將使用者設定的資訊傳到RethrowExRefactoring類別，對Dummy Handler和Empty Catch Block進行重構。

* 1. 新增RethrowExWizard類別，並產生RethrowExInputPage類別，提供Unchecked Exception選單畫面讓使用者選擇。
  2. 使用者選擇Unchecked Exception。
  3. 將使用者設定的重構資訊傳入RethrowExRefactoring類別進行重構程式碼。

完成上述步驟即可完成Dummy Handler 和 Empty Catch Block自動化重構功能。



圖3-23、Dummy Handler和Empty Catch Block Refactoring功能的Class Diagram



圖3-24、Dummy Handler和Empty Catch Block Refactoring功能的Sequence Diagram

#### 3.3.2.2 Nested Try Statement

當使用者在Nested Try Statement[6]選擇警告訊息提供的重構方法時，RLQuickFixer會執行NTMarkerResolution的類別，與Eclipse[5]結合來啟動重構頁面讓使用者設定相關參數和要回報的例外型別等。連結Eclipse的ExtractMethodWizard類別，它繼承了RefactoringWizard類別，提供預覽畫面協助使用者設定重構的相關資訊。在使用者設定完成後，連結Eclipse的ExtractMethodInputPage，將使用者設定的資訊傳到Eclipse的ExtractMethodRefactoring類別，對Nested Try Statement進行重構。

1. 新增ExtractMethodWizard類別，並產生ExtractMethodInputPage類別，提供Extract Method畫面讓使用者設定。
2. 使用者設定Extract Method資訊。
3. 將使用者設定的重構資訊傳入ExtractMethodRefactoring類別進行重構程式碼。

完成上述步驟即可完成Nested Try Statement自動化重構功能。



圖3-25、Nested Try Statement Refactoring功能的Class Diagram



圖3-26、Nested Try Statement Refactoring功能的Sequence Diagram

#### 3.3.2.3 Exception Thrown From Finally Block

當使用者在Exception Thrown From Finally Block[6]選擇警告訊息提供的重構方法時，RLQuickFixer會執行TEFBExtractMethodMarkerResolution的類別，啟動重構頁面讓使用者設定相關參數和要回報的例外型別等。CodeSmellRefactoringWizard繼承Eclipse的RefactoringWizard類別，因此能夠提供預覽畫面協助使用者設定重構的相關資訊。在使用者設定完成後，ExtractMethodInputPage會將使用者設定的資訊傳到TEFBExtractMethodRefactoring類別，對Exception Thrown From Finally Block進行重構。

* 1. 新增CodeSmellRefactoringWizard類別，並產生ExtractMethodInputPage類別，提供Extract Method畫面讓使用者設定。
  2. 使用者設定Extract Method資訊。
  3. 將使用者設定的重構資訊傳入TEFBExtractMethodRefactoring類別進行重構程式碼。

完成上述步驟即可完成Exception Thrown From Finally Block自動化重構功能。



圖3-27、Exception Thrown From Finally Block Refactoring功能的Class Diagram



圖3-28、Exception Thrown From Finally Block Refactoring功能的Sequence Diagram

## 3.4 壞味道的偵測、曝露及消除流程

要消除例外處理壞味道，我們提供了一套偵測、曝露及消除壞味道的流程[18]，總共有五個步驟。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[1]的程式碼，程式碼旁邊會有Robusta[4]對壞味道的標記；第二步：產生曝露程式碼壞味道的測試案例，根據劉彥麟論文[19]提供的利用測試案例來曝露壞味道所帶來的影響；第三步：執行測試案例，如果程式碼沒有正確的處理例外，則測試會失敗；第四步：藉由本論文介紹的壞味道消除方法，消除程式碼的壞味道並正確的處理例外；最後：再執行一次剛剛失敗的測試案例，測試通過代表例外已經被正確的處理，強健度因此提升。如圖3-29所示，為我們提供的壞味道曝露及消除流程圖。



圖3-29、壞味道偵測、曝露及消除流程圖

# 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[4]中並使其自動化，再透過壞味道消除流程[18]來對案例進行分析與應用。

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 From Finally Block兩種壞味道。[6]

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。本論文將用Tomighty來分析Unprotected Main Program壞味道。[6]

## 4.1 Dummy Handler應用實例

### 4.1.1偵測Dummy Handler

如圖4-1所示，Robusta[4]偵測JFreeChart後，發現專案中readPieDatasetFromXML的函式第99、102行含有Dummy Handler。這是一段讀XML檔中Data set的程式碼，catch block[8]會捕捉兩種例外，為SAXException和ParserConfigurationException，我們將以SAXException為例來進行分析與介紹。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，最後程式會回傳null。



圖4-1、JFreeChart中Dummy Handler範例

### 4.1.2產生曝露Dummy Handler的測試案例

我們認為當程式碼在圖4-1的第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4-2所示，藉由劉彥麟同學的論文[19]產生曝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4-3所示，測試案例因為程式碼沒有正確處理例外而造成測試失敗。



圖4-2、JFreeChart Dummy Handler測試案例



圖4-3、JFreeChart Dummy Handler測試失敗

### 4.1.3消除Dummy Handler

如圖4-4所示，接著透過本論文在Robusta[4]提供的自動化快速修復或重構功能來消除壞味道，本論文以重構功能為例。如圖4-5所示，第100行為當catch捕捉到例外後以RuntimeException將例外向上層回報。如圖4-6所示，原先失敗的測試案例因為例外被正確處理而成功。



圖4-4、Robusta提供自動化消除Dummy Handler



圖4-5、JFreeChart Dummy Handler消除的結果



圖4-6、JFreeChart readPieDatasetFromXML函式正確處理例外後測試成功

## 4.2 Careless Cleanup應用實例

### 4.2.1偵測Careless Cleanup

如圖4-7所示，Robusta[4]偵測JFreeChart後，發現專案中encode的函式第180行含有Careless Cleanup壞味道[6]。這是一段將圖片以JPEG格式來進行編碼並輸出的程式碼。當第177、178行的write ()函式或flush函式發生IOException時，例外會被向上層回報，導致程式碼不會執行到第180行釋放資源的程式碼，因此這是一個Careless Cleanup壞味道。

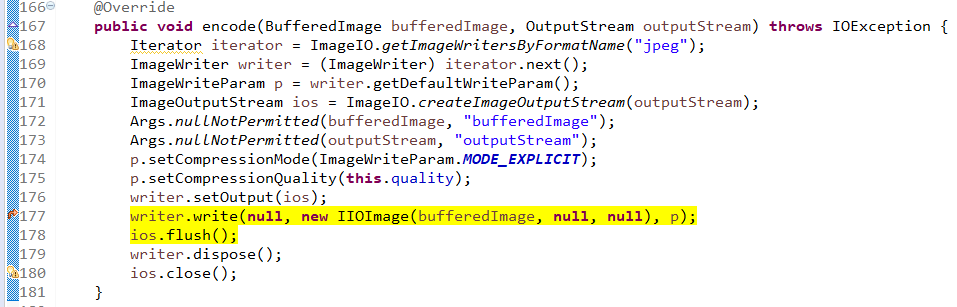


圖4-7、JFreeChart中Careless Cleanup範例

### 4.2.2產生曝露Careless Cleanup的測試案例

如圖4-8所示，藉由劉彥麟同學的論文[19]產生曝露壞味道影響的測試案例，並將encode函式所需要的參數準備好後，接著執行測試案例。如圖4-9所示，測試案例因為程式碼沒有正確處理例外而造成測試失敗。



圖4-8、JFreeChart Careless Cleanup測試案例

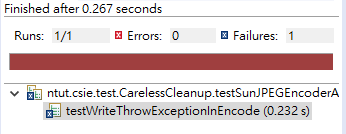


圖4-9、JFreeChart Careless Cleanup測試失敗

### 4.2.3消除Careless Cleanup

如圖4-10所示，接著透過本論文在Robusta[4]提供的自動化快速修復功能來消除壞味道。如圖4-11所示，產生try/finally來保護程式碼，並且在finally裡用try/catch將釋放資源的例外保護住，避免產生Exception Thrown From Finally Block。[6]如圖4-12所示，原先失敗的測試案例因為例外被正確處理而成功。

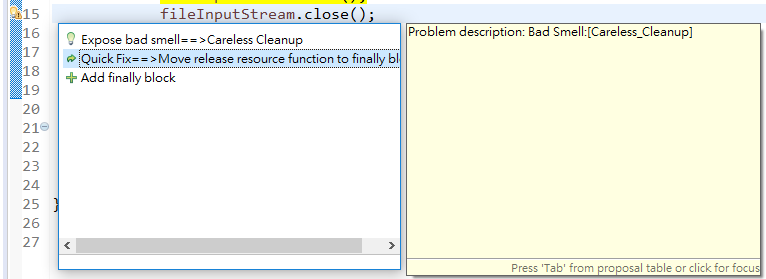


圖4-10、Robusta提供自動化消除Careless Cleanup

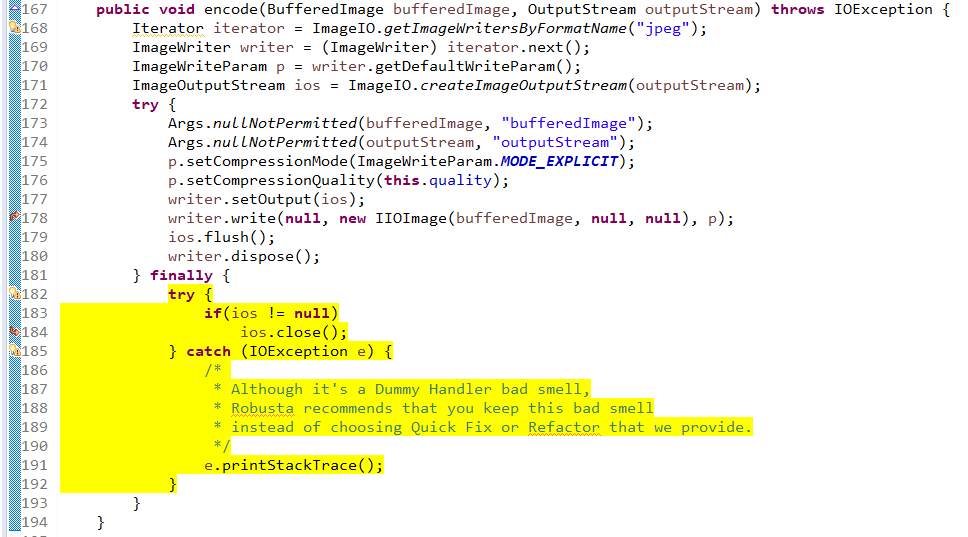


圖4-11、JFreeChart Careless Cleanup消除的結果

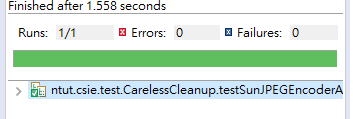


圖4-12、JFreeChart encode函式正確處理例外後測試成功

## 4.3 Exception Thrown From Finally Block應用實例

### 4.3.1偵測Exception Thrown From Finally Block

如圖4-13所示，Robusta[4]偵測JFreeChart後，發現專案中saveChartAsPNG的函式含有Exception Thrown From Finally Block。[6]這段程式碼是將圖片存成PNG的格式。當308行發生例外時，會將例外向上層回報，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，改向上層回報第311行的例外，因此這是一個Exception Thrown From Finally Block。



圖4-13、JFreeChart中Exception Thrown From Finally Block範例

### 4.3.2產生曝露Exception Thrown From Finally Block的測試案例

如圖4-14所示，藉由劉彥麟同學的論文[19]產生曝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4-15所示，測試案例因為程式碼沒有正確處理例外而造成測試失敗。



圖4-14、JFreeChart Exception Thrown From Finally Block測試案例



圖4-15、JFreeChart Exception Thrown From Finally Block壞味道測試失敗

### 4.3.3消除Exception Thrown From Finally Block

如圖4-16所示，接著透過本論文在Robusta[4]提供的自動化重構功能來消除壞味道。如圖4-17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4-18所示，原先失敗的測試案例因為例外被正確處理而成功。



圖4-16、Robusta提供自動化消除Exception Thrown From Finally Block



圖4-17、JFreeChart Exception Thrown From Finally Block消除的結果



圖4-18、JFreeChart saveChartAsPNG函式正確處理例外後測試成功

## 4.4 Unprotected Main Program應用實例

如圖4-19所示，為Tomighty工具正常啟動時的畫面。



圖4-19、Tomighty工具

### 4.4.1 偵測Unprotected Main Program

如圖4-20所示，為Tomighty專案的main program所在位置。透過Robusta[4]的偵測顯示它具有Unprotected Main Program。[6]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4-20、Tomighty中Unprotected Main Program範例

### 4.4.2產生曝露Unprotected Main Program的測試案例

如圖4-21所示，藉由劉彥麟同學的論文[19]產生曝露壞味道影響的測試案例，接著執行測試案例，如圖4-22所示，測試案例會因為程式碼沒有正確處理例外而造成測試失敗。



圖4-21、Tomighty Unprotected Main Program測試案例



圖4-22、Tomighty Unprotected Main Program測試失敗

### 4.4.3消除Unprotected Main Program

如圖4-23所示，接著透過本論文在Robusta[4]提供的自動化快速修復功能來消除壞味道。如圖4-24所示，將main program用try/catch保護起來，且catch捕捉Throwable[16]類別，最後將例外訊息寫入日誌檔中。如圖4-25所示，原先失敗的測試案例因為例外被正確處理而成功。



圖4-23、Robusta提供自動化消除Unprotected Main Program

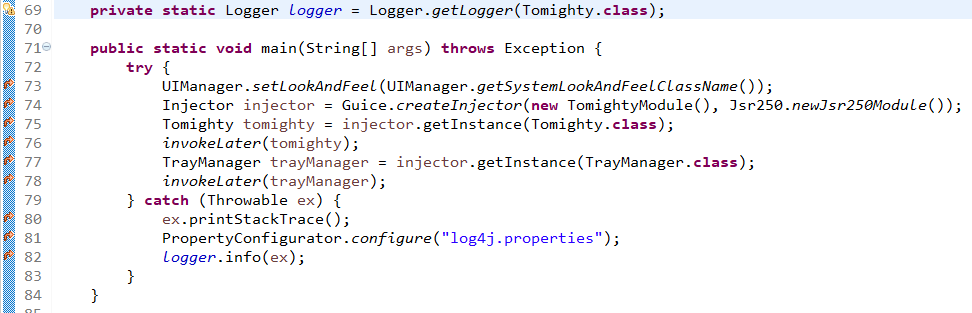


圖4-24、Tomighty Unprotected Main Program消除的結果



圖4-25、Tomighty main program正確處理例外後測試成功

# 第五章 結論與未來展望

## 5.1 結論

本論文將Robusta[4]原有的壞味道消除方法重新修正及改善，並且將Careless Cleanup[6]的快速修復功能實作於Robusta中；最後，將Robusta應用於開源專案，且成功的消除程式碼中例外處理的壞味道，進而提升專案的系統強健度。

## 5.2 未來展望

根據壞味道消除的功能，目前有以下幾點功能能夠改善。

第一點：在Careless Cleanup的快速修復功能中，目前只提供對一層Try Statement[8]來進行快速修復，如果釋放資源的程式碼在超過一層的巢狀結構，如圖5-1所示，則無法消除Careless Cleanup。因此如果能夠讓Careless Cleanup能夠在多層巢狀結構下快速修復的話，消除壞味道的功能將會更完善。

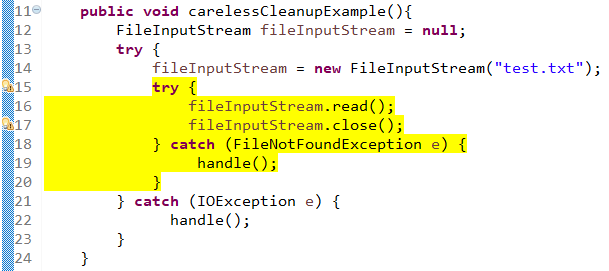


圖5-1、兩層Try Statement的Careless Cleanup範例

第二點：目前Careless Cleanup在快速修復後，會衍生出Nested Try Statement，如果Careless Cleanup能夠提供重構功能，讓使用者能夠填寫要獨立出來的函式名稱，並將衍生的Nested Try Statement自動獨立成一個函式，對使用者將會方便很多。

# 參考文獻

1. Chien-Tsun Chen. 例外處理設計的逆襲. 悅知文化，2014
2. 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
3. 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
4. Robusta at Eclipse Marketplace, <https://marketplace.eclipse.org/content/robusta-eclipse-plugin> [Accessed 01 June 2018]
5. Eclipse, <http://www.eclipse.org/> [Accessed 01 June 2018]
6. 陳友倫、以Aspect揭露導因於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
7. Chien-Tsun Chen, Yu Chin Cheng, Chin-Yun Hsieh and I-Lang Wu, “Exception Handling Refactorings: Directed by Goals and Driven by Bug Fixing,” Journal of Systems and Software, vol.82, no.2, pp.333-345, Feb. 2009.
8. ASTNode document, <https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTVisitor.html> [Accessed 01 June 2018]
9. Eclipse JDT, <https://www.programcreek.com/2011/01/best-java-development-tooling-jdt-and-astparser-tutorials/> [Accessed 01 June 2018]
10. AST View at Eclipse Marketplace, <https://marketplace.eclipse.org/content/ast-view> [Accessed 01 June 2018]
11. ASTParser document, <https://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTParser.html> [Accessed 01 June 2018]
12. ASTRewrite document,

<https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2Frewrite%2FASTRewrite.html> [Accessed 01 June 2018]

1. 楊智傑，Robusta- 一個對於Java例外處理壞味道的偵測工具，碩士論文，國立臺北科技大學資訊工程系碩士班，台北，2013.
2. Exception document, <https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html> [Accessed 01 June 2018]
3. Error document, <https://docs.oracle.com/javase/7/docs/api/java/lang/Error.html> [Accessed 01 June 2018]
4. Throwable document, <https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html> [Accessed 01 June 2018]
5. Eclipse IMarker document,

<http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fresources%2FIMarker.html> [Accessed 01 June 2018]

1. 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
2. 劉彥麟 利用AspectJ搭配測試案例曝露例外處理壞味道的影響，碩士論文，國立臺北科技大學資訊工程系碩士班，台北，2018