



國立臺北科技大學

資訊工程系碩士班

碩士學位論文

利用 AspectJ 搭配測試案例曝露
例外處理壞味道的影響

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

研究生：劉彥麟

指導教授：謝金雲教授、鄭有進教授

中華民國一百零七年七月

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.....	ii
ABSTRACT.....	iii
誌謝.....	iv
目錄.....	v
表目錄.....	viii
圖目錄.....	ix
第一章 緒論.....	1
1.1 研究背景與動機.....	1
1.2 研究目標.....	1
1.3 論文組織架構.....	2
第二章 背景知識與相關研究.....	3
2.1 例外處理壞味道.....	3
2.1.1 Dummy Handler.....	3
2.1.2 Empty Catch Block	4
2.1.3 Nested Try Statement.....	5
2.1.4 Unprotected Main Program.....	6
2.1.5 Exception Thrown From Finally Block	7
2.1.6 Careless Cleanup.....	8
2.2 Robusta	8
2.3 AspectJ	9
2.3 Abstract Syntax Tree Node(AST Node)	10

第三章 研究方法.....	11
3.1 曝露壞味道的方法.....	11
3.1.1 Dummy Handler.....	11
3.1.2 Empty Catch Block	12
3.1.3 Unprotected Main Program.....	13
3.1.4 Exception Thrown From Finally Block	14
3.1.5 Careless Cleanup.....	15
第四章 設計與實作.....	16
4.1 Dummy Handler Empty Catch Block	17
4.1.1 產生 AspectJ、測試檔案的素材收集	17
4.1.2 實作 Dummy Handler & Empty Catch Block 搜集素材	18
4.1.3 分析測試搭配 AspectJ 與程式碼的互動	21
4.2 Unprotected Main Program.....	22
4.2.1 產生 AspectJ、測試檔案的素材收集	22
4.2.2 實作 Unprotected Main Program.....	23
4.2.3 分析測試搭配 AspectJ 與程式碼的互動	26
4.3 Exception Thrown From Finally Block	27
4.3.1 產生 AspectJ、測試檔案的素材收集	27
4.3.2 實作 Exception Thrown From Finally Block	29
4.3.3 分析測試搭配 AspectJ 與程式碼的互動	32
4.4 Careless Cleanup.....	34
4.4.1 產生 AspectJ、測試檔案的素材收集	34
4.4.2 實作 Careless Cleanup.....	35
4.4.3 分析測試搭配 AspectJ 與程式碼的互動	38
第五章 案例分析.....	40

5.1 Unprotected Main Program 案例分析.....	40
5.1.1 偵測並分析 Unprotected Main Program.....	40
5.1.2 呈現 Unprotected Main Program 對系統的影響.....	40
5.1.3 消除 Unprotected Main Program.....	42
5.2 Exception Thrown From Finally Block 案例分析與實作	43
5.2.1 偵測並分析 Exception Thrown From Finally Block	43
5.2.2 呈現 Exception Thrown From Finally Block 造成的影響	43
5.2.3 消除 Exception Thrown From Finally Block	45
5.3 Dummy Handler 案例分析與實作.....	46
5.3.1 偵測並分析 Dummy Handler.....	46
5.3.2 呈現 Dummy Handler 造成的影響.....	46
5.3.3 消除 Dummy Handler 壞味道.....	48
第六章 結論與未來研究方向.....	49
6.1 結論.....	49
6.2 未來展望.....	49
參考文獻.....	50

表目錄

表 1 實作物件功能列表.....	16
-------------------	----

圖目錄

圖 2-1 Dummy Handler 壞味道範例	3
圖 2-2 Empty Catch Block 壞味道範例.....	4
圖 2-3 Nested Try Statement 壞味道範例.....	5
圖 2-4 Unprotected Main Program 壞味道範例	6
圖 2-5 Exception Thrown From Finally Block 壞味道範例	7
圖 2-6 Careless Cleanup 壞味道範例	8
圖 2-7 AspectJ 目標的程式碼.....	9
圖 2-9 嵌入 AspectJ 後運行程式碼的結果.....	10
圖 2-10 AST Node 結構之範例	10
圖 3-1 Dummy Handler 例外嵌入範例	11
圖 3-2 Empty Catch Block 例外嵌入範例.....	12
圖 3-3 Unprotected Main Program 例外嵌入範例	13
圖 3-4 Exception Thrown From Finally Block 例外嵌入範例.....	14
圖 3-5 Careless Cleanup 例外嵌入範例	15
圖 4-1 Dummy Handler 壞味道範例	17
圖 4-2 Dummy Handler 的 AspectJ 範例程式碼.....	17
圖 4-3 Dummy Handler 搭配 AspectJ 的測試案例.....	18
圖 4-4 組成 AspectJ 程式碼、測試案例相關的 Class Diagram.....	19
圖 4-5 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram	20
圖 4-6 含有 Dummy Handler 壞味道的程式碼	21
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

圖 4-9 Unprotected Main Program 壞味道範例	22
圖 4-10 Unprotected Main Program 的 AspectJ 範例程式碼.....	23
圖 4-11 Unprotected Main Program 搭配 AspectJ 的測試案例.....	23
圖 4-12 組成 AspectJ 程式碼、測試案例相關的 Class Diagram.....	24
圖 4-13 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram	25
圖 4-14 含有 Unprotected Main Program 壞味道的程式碼	26
圖 4-15 重現 Unprotected Main Program 壞味道發生例外的測試程式碼	26
圖 4-16 根據 Unprotected Main Program 所製作的 AspectJ 程式	27
圖 4-17 Exception Thrown From Finally Block 壞味道範例	27
圖 4-18 Exception Thrown From Finally Block 的 AspectJ 範例程式碼	28
圖 4-19 Exception Thrown From Finally 搭配 AspectJ 的測試案例.....	28
圖 4-20 組成 AspectJ 程式碼、測試案例相關的 Class Diagram.....	30
圖 4-21 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram	31
圖 4-22 含有 Exception Thrown From Finally 壞味道的程式碼	32
圖 4-23 重現 Exception Thrown From Finally 壞味道發生例外的測試程式碼	33
圖 4-24 根據 Exception Thrown From Finally 所製作的 AspectJ 程式.....	33
圖 4-25 Careless Cleanup 壞味道範例	34
圖 4-26 Careless Cleanup AspectJ 範例程式碼.....	34
圖 4-27 重現 Careless Cleanup 壞味道發生例外的測試程式碼	35
圖 4-28 組成 AspectJ 程式碼、測試案例相關的 Class Diagram.....	36
圖 4-29 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram	37
圖 4-30 含有 Careless Cleanup 壞味道的程式碼	38
圖 4-31 重現 Careless Cleanup 壞味道發生例外的測試程式碼	38
圖 4-32 根據 Careless Cleanup 所製作的 AspectJ 程式.....	39
圖 5-1 含有 Unprotected Main Program 壞味道案例程式碼	40

圖 5-2 程式正常運行結果	41
圖 5-3 Unprotected Main Program AspectJ 程式碼.....	41
圖 5-4 與 AspectJ 相同規則產生對應嵌入符號的程式碼.....	41
圖 5-5 與 AspectJ 搭配的測試案例.....	42
圖 5-6 測試案例 Failure 結果圖	42
圖 5-7 測試案例 Pass 結果圖	42
圖 5-8 含有 Exception Thrown From Finally 壞味道案例程式碼	43
圖 5-9 Exception Thrown From Finally Block AspectJ 程式碼	44
圖 5-10 與 AspectJ 相同規則產生對應嵌入符號的程式碼.....	44
圖 5-11 與 AspectJ 搭配的測試案例.....	45
圖 5-12 測試案例 pass 結果圖	45
圖 5-13 含有 Dummy Handler 壞味道案例程式碼	46
圖 5-14 Dummy Handler AspectJ 程式碼.....	47
圖 5-15 與 AspectJ 相同規則產生對應嵌入符號的程式碼.....	47
圖 5-16 與 AspectJ 搭配的測試案例.....	48
圖 5-17 測試案例 pass 結果圖	48

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。

```
9= public void writeFile() {  
10     FileWriter fw = null;  
11     try {  
12         fw = new FileWriter("k:\\test.txt");  
13     } catch (IOException e) {  
14         e.printStackTrace();  
15     }  
16 }  
17
```

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。

```
9 public void writeFile() {  
10     FileWriter fw = null;  
11     try {  
12         fw = new FileWriter("k:\\test.txt");  
13     } catch (IOException e) {  
14     }  
15 }  
16
```

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。

```
9 public void writeFile() {  
10     FileWriter fw = null;  
11     try {  
12         fw = new FileWriter("k:\\test.txt");  
13         fw.write("test");  
14         fw.flush();  
15     } catch (IOException e) {  
16         e.printStackTrace();  
17     } finally {  
18         try {  
19             fw.close();  
20         } catch (IOException e) {  
21             e.printStackTrace();  
22         }  
23     }  
24 }
```

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。

```
56 public static void main(String[] args) throws Exception {  
57     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
58     Injector injector = Guice.createInjector(new TomightyModule(), Js250.newJs250Module());  
59  
60     Tomighty tomighty = injector.getInstance(Tomighty.class);  
61     invokeLater(tomighty);  
62     TrayManager trayManager = injector.getInstance(TrayManager.class);  
63     invokeLater(trayManager);  
64 }
```

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。

```
11 public void writeFile() throws IOException {
12     FileWriter fw = null;
13     try {
14         fw = new FileWriter("k:\\test.txt");
15         fw.write("test");
16         fw.flush();
17     } catch (IOException e) {
18         throw e;
19     } finally {
20         try {
21             fw.close();
22         } catch (IOException e) {
23             throw e;
24         }
25     }
26 }
```

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。

```
11 public void writeFile() throws IOException {
12     FileWriter fw = null;
13     try {
14         fw = new FileWriter("k:\\test.txt");
15         fw.write("test");
16         fw.flush();
17         fw.close();
18     } catch (IOException e) {
19         throw e;
20     }
21 }
```

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 運程式碼的結果呈現嵌入的程式碼。

```
7 public class Example {
8
9     public static void main(String[] args) {
10         FileWriter writer = null;
11         try{
12             File file = new File("Hello1.txt");
13             writer = new FileWriter(file);
14             writer.write("123");
15         }catch(IOException e){
16             e.printStackTrace();
17         }finally{
18             try {
19                 if(writer!=null)
20                     writer.close();
21             } catch (IOException e) {
22                 e.printStackTrace();
23             }
24         }
25     }
26
27
28 }
```

圖 2-7 AspectJ 目標的程式碼

```

4 public aspect HelloAspect {
5
6     before() throws IOException:(call(* *.write(..) throws IOException) && withincode(* Example.main(..))) {
7         System.out.println("In AspectJ");
8         throw new IOException();
9     }
10
11 }

```

圖 2-8 AspectJ 程式碼範例

```

In AspectJ
java.io.IOException
    at introduction.HelloAspect.ajc$before$introduction_HelloAspect$1$19032eff(HelloAspect.aj:8)
    at introduction.Example.main(Example.java:14)
|

```

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 ASTNode 來幫助降低靜態分析的難度。透過 ASTNode，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 ASTNode 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。

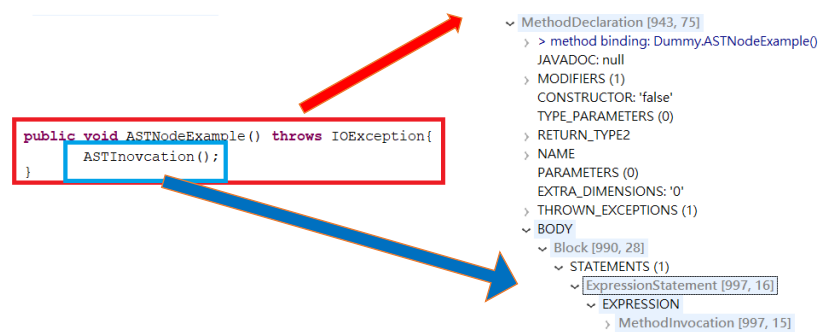


圖 2-10 AST Node 結構之範例

第三章 研究方法

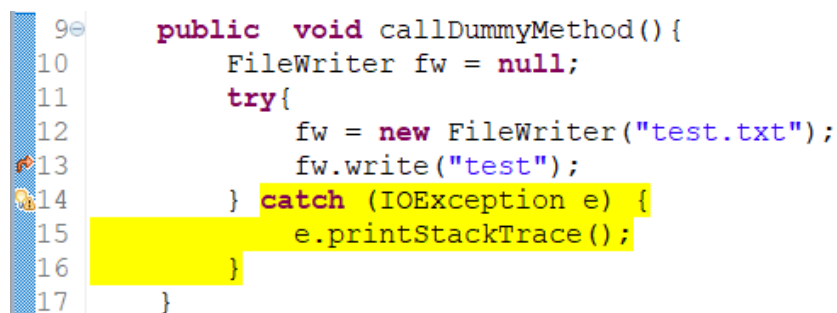
本論文以陳友倫[1]的研究為基礎，將其提出的方法實作在 Robusta 定義的壞味道中，讓所有的壞味道都能產生對應的 AspectJ 程式碼和測試，藉由測試搭配 AspectJ 嵌入例外來驗證例外處理行為是否正確，並將壞味道對於軟體的影響現形。

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。

若要让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。



```
9 public void callDummyMethod() {
10     FileWriter fw = null;
11     try {
12         fw = new FileWriter("test.txt");
13         fw.write("test");
14     } catch (IOException e) {
15         e.printStackTrace();
16     }
17 }
```

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。

若要让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。

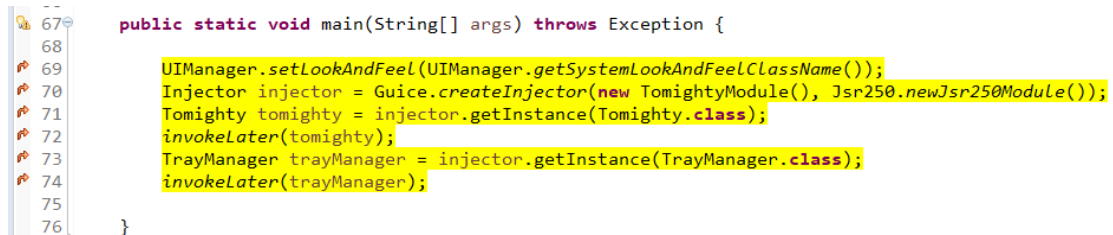
```
8 public class Dummy {
9     public void callDummyMethod() {
10         FileWriter fw = null;
11         try{
12             fw = new FileWriter("test.txt");
13             fw.write("test");
14         } catch (IOException e) {
15             }
16     }
17 }
```

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。



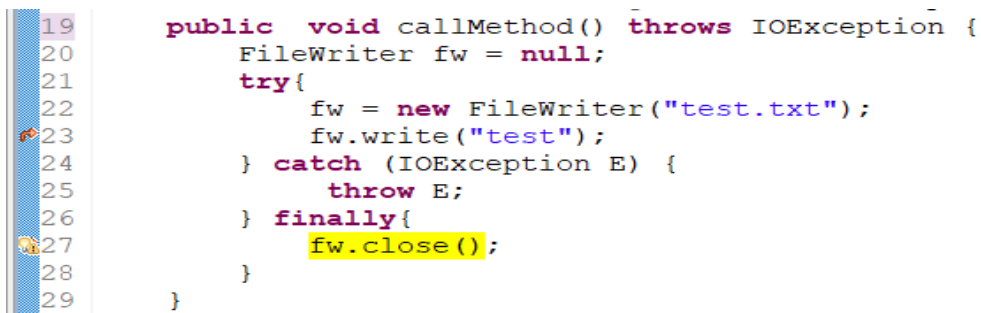
```
67 public static void main(String[] args) throws Exception {  
68  
69     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
70     Injector injector = Guice.createInjector(new TomightyModule(), Jsr250.newJsr250Module());  
71     Tomighty tomighty = injector.getInstance(Tomighty.class);  
72     invokeLater(tomighty);  
73     TrayManager trayManager = injector.getInstance(TrayManager.class);  
74     invokeLater(trayManager);  
75  
76 }
```

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。



```
19 public void callMethod() throws IOException {
20     FileWriter fw = null;
21     try{
22         fw = new FileWriter("test.txt");
23         fw.write("test");
24     } catch (IOException E) {
25         throw E;
26     } finally{
27         fw.close();
28     }
29 }
```

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。

```
166 @Override
167 public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException {
168     Iterator<ImageWriter> iterator = ImageIO.getImageWritersByFormatName("jpeg");
169     ImageWriter writer = (ImageWriter) iterator.next();
170     ImageWriteParam p = writer.getDefaultWriteParam();
171     ImageOutputStream ios = ImageIO.createImageOutputStream(outputStream);
172     Args.nullNotPermitted(bufferedImage, "bufferedImage");
173     Args.nullNotPermitted(outputStream, "outputStream");
174     p.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
175     p.setCompressionQuality(this.quality);
176     writer.setOutput(ios);
177     writer.write(null, new IIOMemorySource(bufferedImage, null, null), p);
178     ios.flush();
179     writer.dispose();
180     ios.close();
181 }
```

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
AddAspectMarkerResolution_[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

```
87 public static PieDataset readPieDatasetFromXML(InputStream in)
88     throws IOException {
89
90     PieDataset result = null;
91     SAXParserFactory factory = SAXParserFactory.newInstance();
92
93     try {
94         SAXParser parser = factory.newSAXParser();
95         PieDatasetHandler handler = new PieDatasetHandler();
96         parser.parse(in, handler);
97         result = handler.getDataset();
98     } catch (SAXException e) {
99         System.out.println(e.getMessage());
100     }
101     catch (ParserConfigurationException e2) {
102         System.out.println(e2.getMessage());
103     }
104     return result;
105 }
106 }
```

圖 4-1 Dummy Handler 壞味道範例

捕捉例外的類型

壞味道所在的 class 及 method 名稱

```
9 before() throws SAXException: (call(*(..) throws SAXException) ) && withincode(* DatasetReader.readPieDatasetFromXML(..)){
10
11     String name = thisJoinPoint.getSignature().getName();
12     String operation = AspectJSwitch.getInstance().getOperation(name);
13     if (operation.equals("f(SAXException)"))
14         throw new SAXException("This exception is thrown from DummyHandler's unit test by using AspectJ.");
15
16 }
17
18
19 Object around(): (call(*(..)||call(*.new(..))) && withincode (* DatasetReader.*(..)){
20     String method = thisJoinPoint.toString();
21     if (method.contains("Exception(Throwable)")){
22         return proceed();
23     } else {
24         return null;
25     }
26 }
```

圖 4-2 Dummy Handler 的 AspectJ 範例程式碼

```

14= @Test
15 public void testDueToNewSAXParserCatchBlockShouldThrowExceptionInReadPieDatasetFromXML() {
16     aspectJSwitch.initResponse();
17     aspectJSwitch.addResponse(newSAXParser/f(SAXException));
18     aspectJSwitch.toFirstResponse();
19     try{
20         DatasetReader.readPieDatasetFromXML(null);
21         Assert.fail("Exception is not thrown from the catch block.");
22     } catch (Exception e) {
23         String exceptionMessage = e.getMessage().toString();
24         Assert.assertTrue(exceptionMessage.contains("This exception is thrown from DummyHandler's unit test by using AspectJ."));
25     }
26 }
27 }

```

跟 catch block 捕捉相同例外

圖 4-3 Dummy Handler 搭配 AspectJ 的測試案例

4.1.2 實作 Dummy Handler & Empty Catch Block 搜集素材

當使用者選擇 Expose bad smell 之後，AspectMarkerResolutionForDummyHandlerAndEmptyCatchBlock 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

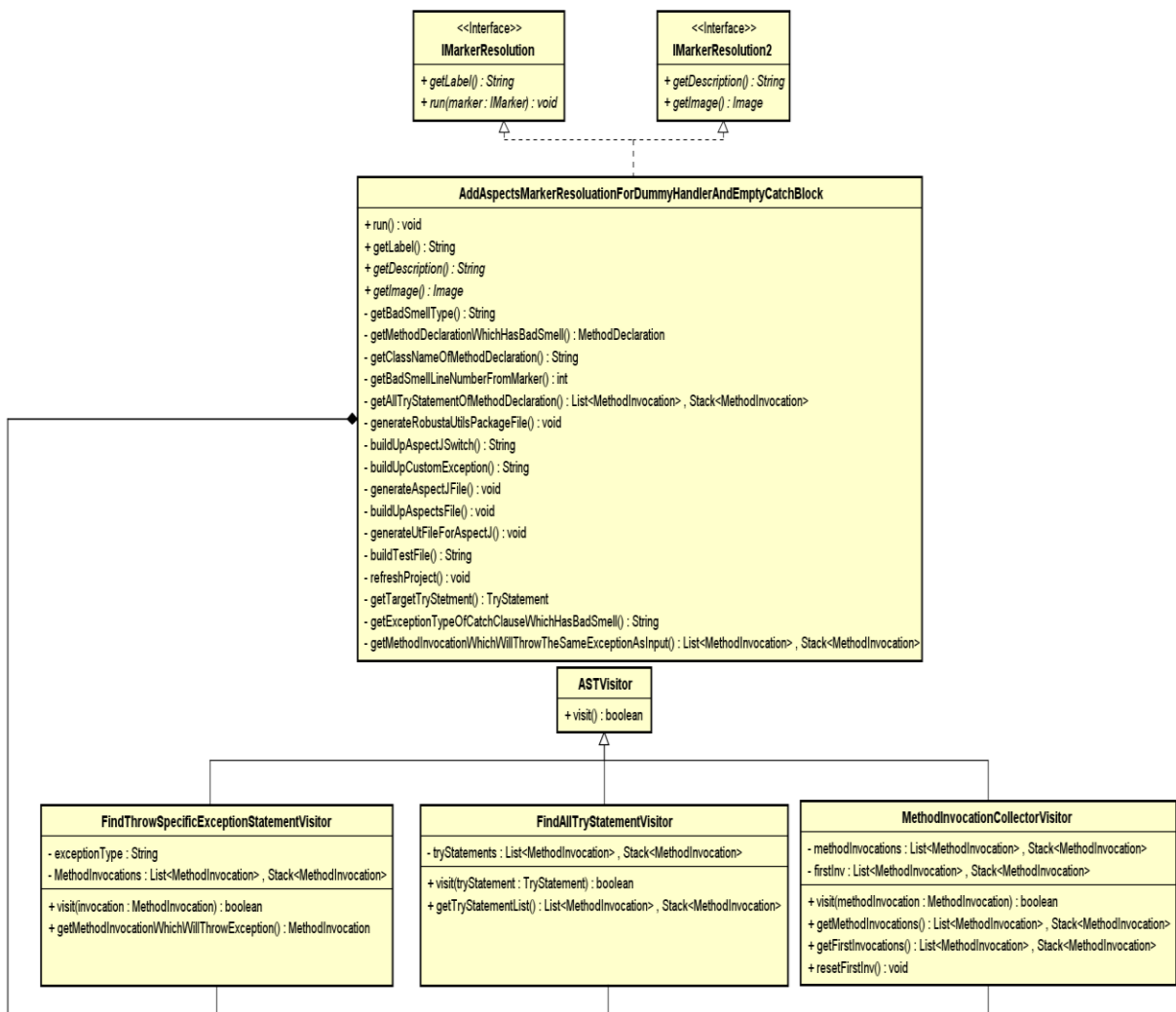


圖 4-4 組成 AspectJ 程式碼、測試案例相關的 Class Diagram

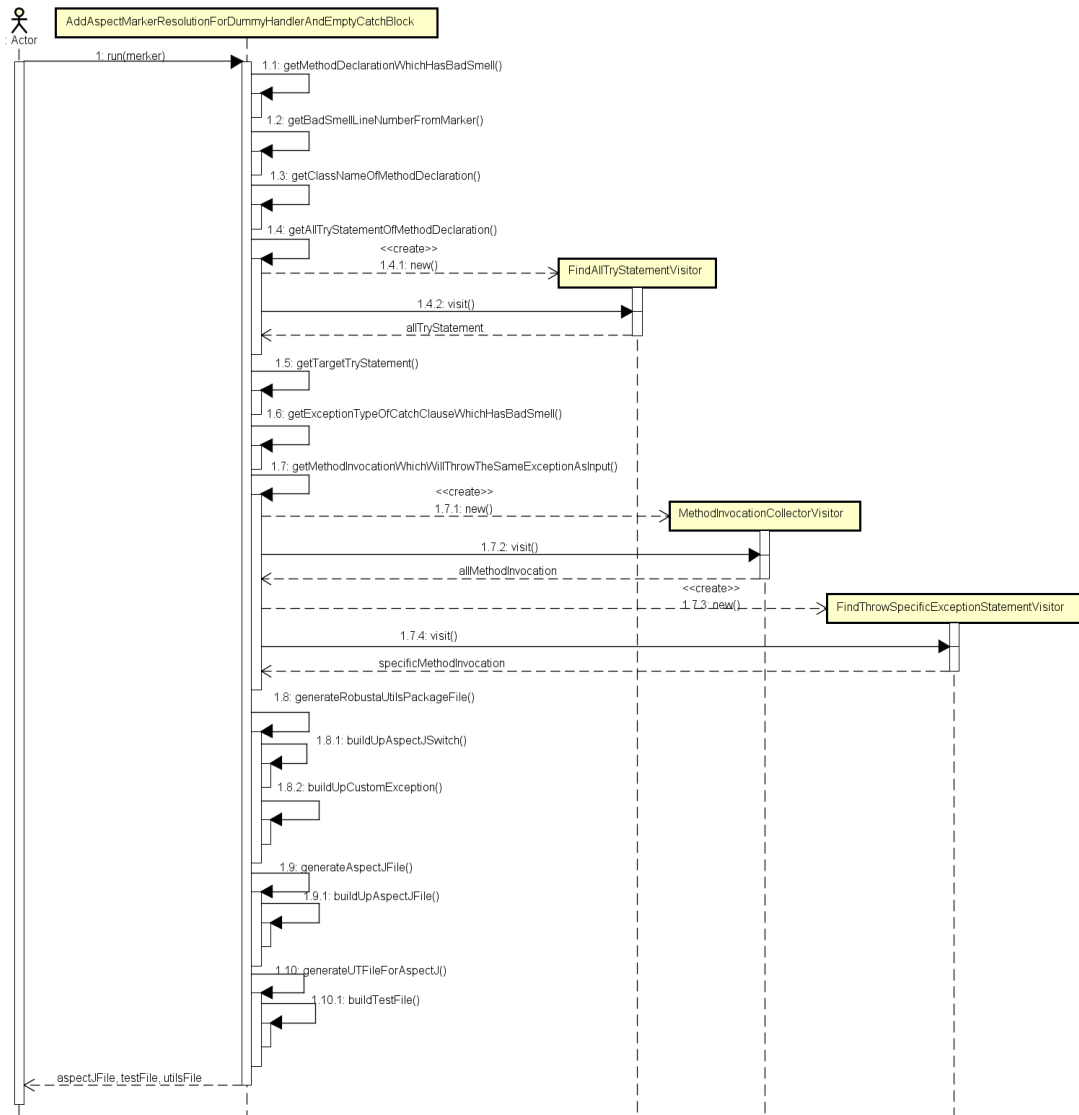


圖 4-5 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram

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。

```
87 public static PieDataset readPieDatasetFromXML(InputStream in)
88     throws IOException, SAXException {
89     PieDataset result = null;
90     SAXParserFactory factory = SAXParserFactory.newInstance();
91     try {
92         SAXParser parser = factory.newSAXParser();
93         PieDatasetHandler handler = new PieDatasetHandler();
94         parser.parse(in, handler);
95         result = handler.getDataset();
96     }
97     catch (SAXException e) {
98         System.out.println(e.getMessage());
99     }
100    catch (ParserConfigurationException e2) {
101        System.out.println(e2.getMessage());
102    }
103    return result;
104 }
```

圖 4-6 含有 Dummy Handler 壞味道的程式碼

```
14 @Test
15 public void testDueToNewSAXParserCatchBlockShouldThrowExceptionInReadPieDatasetFromXML() {
16     aspectJSwitch.initResponse();
17     aspectJSwitch.addResponse("newSAXParser/f(SAXException)");
18     aspectJSwitch.toFirstResponse();
19     try{
20         DatasetReader.readPieDatasetFromXML(null);
21         Assert.fail("Exception is not thrown from the catch block.");
22     } catch (Exception e) {
23         String exceptionMessage = e.getMessage().toString();
24         Assert.assertTrue(exceptionMessage.contains("This exception is thrown from DummyHandler's unit test by using AspectJ."));
25     }
26 }
27 }
```

圖 4-7 重現 Dummy Handler 例外處理壞味道發生例外的測試程式碼


```

9= before() throws SAXException: (call(* *(..) throws SAXException) ) && withcode(* DatasetReader.readPieDatasetFromXML(..)){
10
11 String name = thisJoinPoint.getSignature().getName();
12 String operation = AspectJSwitch.getInstance().getOperation(name);
13 if (operation.equals("f(SAXException)"))
14     throw new SAXException("This exception is thrown from DummyHandler's unit test by using AspectJ.");
15
16 }
17
18
19= Object around():(call(* *(..))||call(*.new(..)) && withcode (* DatasetReader.*(..)){
20     String method = thisJoinPoint.toString();
21     if(method.contains("Exception(Throwable)")){
22         return proceed();
23     }else{
24         return null;
25     }
26 }

```

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 MethodInvocation 名稱，如圖 4-9 第 68~73 行的 MethodInvocation
2. 取得 Unprotected Main Program 壞味道的 MethodDeclaration[9]名稱，圖 4-9 第 67 行的 main program
3. 取得 Unprotected Main Program 壞味道 MethodDeclaration 所屬的 Class 名稱

```

67= public static void main(String[] args) throws Exception {
68     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
69     Injector injector = Guice.createInjector(new TomightyModule(), Js250Module());
70     Tomighty tomighty = injector.getInstance(Tomighty.class);
71     invokeLater(tomighty);
72     TrayManager trayManager = injector.getInstance(TrayManager.class);
73     invokeLater(trayManager);
74 }

```

圖 4-9 Unprotected Main Program 壞味道範例

```

6 public aspect TomightyAspectException {
7     before() : (call(* *(..) ) ) && within(Tomighty) && withincode(* main(..) ){
8         String name = thisJoinPoint.getSignature().getName();
9         String operation = AspectJSwitch.getInstance().getOperation(name);
10        if (operation.equals("f(RuntimeException)"))
11            throw new RuntimeException("Main Program is not surround with try/catch.");
12    }
13 }
14

```

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

```

12 @Test
13 public void testSetLookAndFeelThrowExceptionInMain() {
14     aspectJSwitch.initResponse();
15     aspectJSwitch.addResponse("setLookAndFeel/f(RuntimeException)");
16     aspectJSwitch.toFirstResponse();
17     try{
18         String[] args={};
19         Tomighty.main(args);
20     }catch (Throwable e) {
21         Assert.fail(e.getMessage());
22     }
23 }

```

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，AddAspectMarkerResolutionForUnprotectedMain 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、Unprotected 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

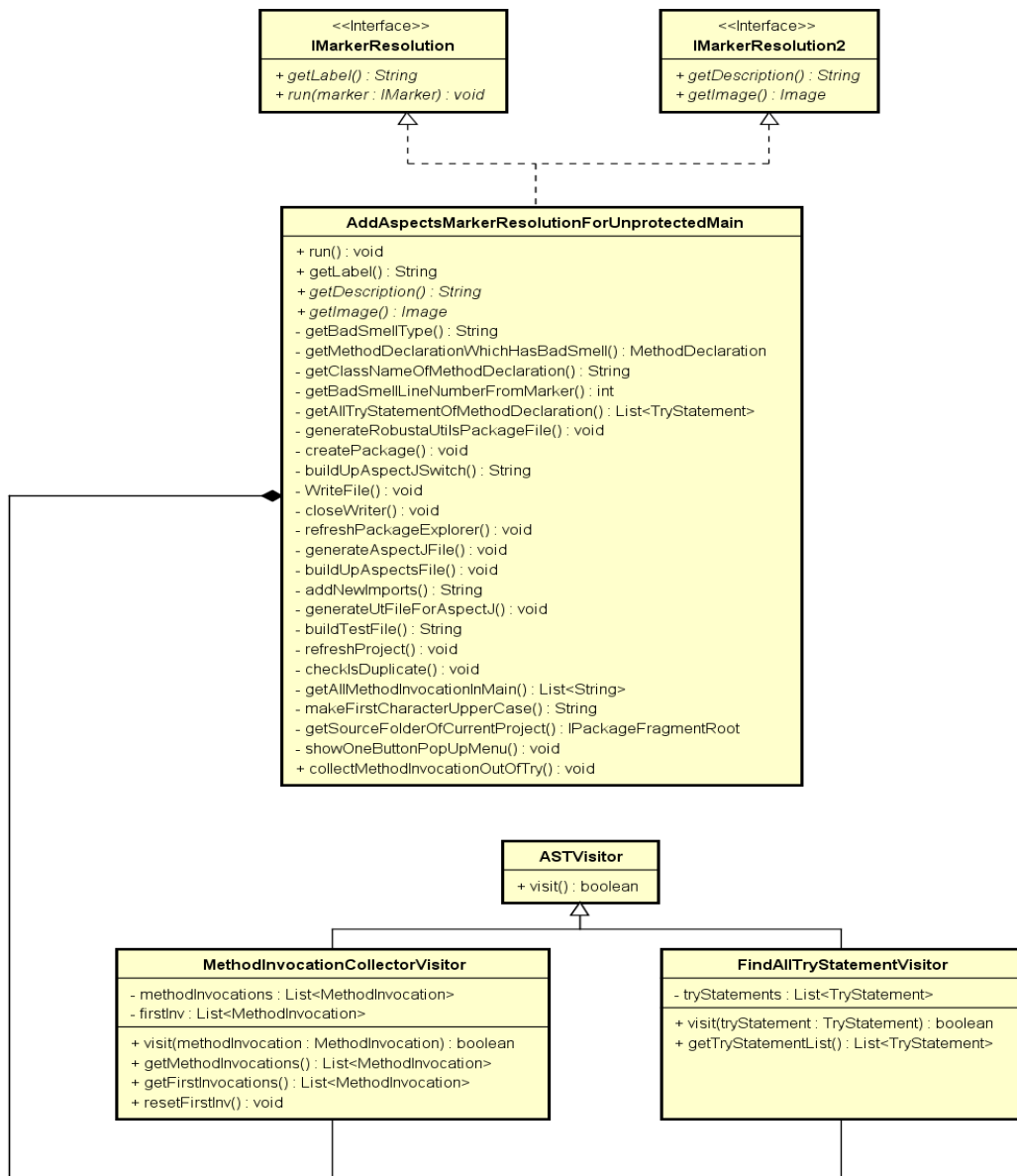


圖 4-12 組成 AspectJ 程式碼、測試案例相關的 Class Diagram

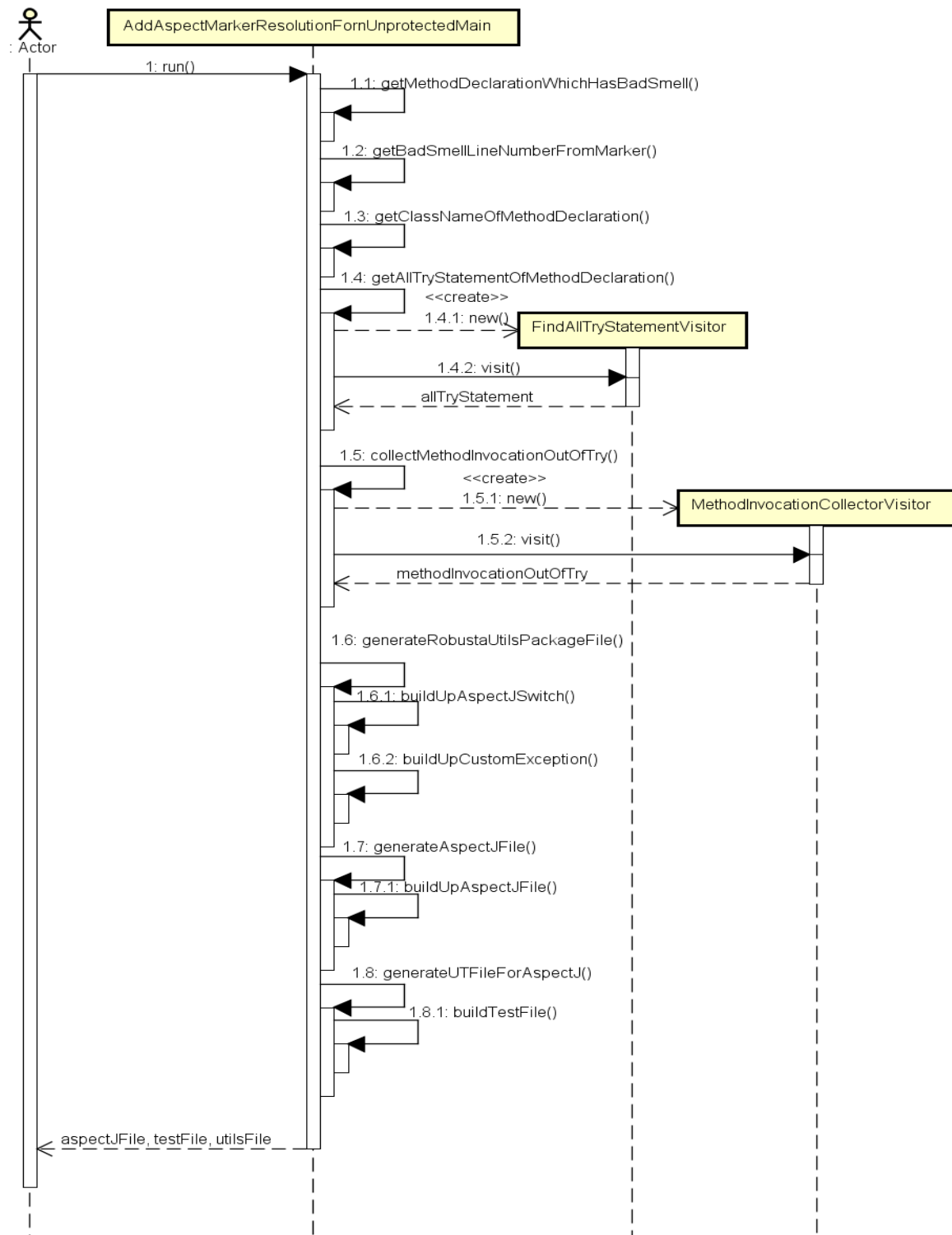


圖 4-13 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram

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。

```
67 public static void main(String[] args) throws Exception {
68     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
69     Injector injector = Guice.createInjector(new TomightyModule(), Js250.newJs250Module());
70     Tomighty tomighty = injector.getInstance(Tomighty.class);
71     invokeLater(tomighty);
72     TrayManager trayManager = injector.getInstance(TrayManager.class);
73     invokeLater(trayManager);
74 }
```

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

```
12 @Test
13 public void testDueToSetLookAndFeelMainShouldNotThrowAnyException() {
14     aspectJSwitch.initResponse();
15     aspectJSwitch.addResponse("setLookAndFeel/f(RuntimeException)");
16     aspectJSwitch.toFirstResponse();
17     try{
18         String[] args={};
19         Tomighty.main(args);
20     }catch (Throwable e) {
21         Assert.fail(e.getMessage());
22     }
23 }
```

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

```

6 public aspect TomightyAspectException {
7     before() : (call(* *(...) ) ) && within(Tomighty) && withincode(* main(..) ){
8         String name = thisJoinPoint.getSignature().getName();
9         String operation = AspectJSwitch.getInstance().getOperation(name);
10        if (operation.equals("f(RuntimeException)"))
11            throw new RuntimeException("Main Program is not surround with try/catch.");
12    }
13 }
14 }

```

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

```

301 public static void saveChartAsPNG(File file, JFreeChart chart,
302     int width, int height, ChartRenderingInfo info)
303     throws IOException {
304
305     Args.nullNotPermitted(file, "file");
306     OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
307     try {
308         ChartUtils.writeChartAsPNG(out, chart, width, height, info);
309     }
310     finally {
311         out.close();
312     }
313 }

```

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

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

```

10 before() throws IOException : (call(* *.close(..) throws IOException) && within(ChatUtils){
11 String name = thisJoinPoint.getSignature().getName();
12 String operation = AspectJSwitch.getInstance().getOperation(name);
13 if (operation.equals("f(IOException)"))
14     throw new IOException("This Exception is thrown from finally block, so it is a Exception Thrown From Finally Block bad smell.");
15 }
16
17 before(): (call(* *(..) throws IOException) && withincode(* ChatUtils.saveChartAsPNG(..)){
18 String name = thisJoinPoint.getSignature().getName();
19 String operation = AspectJSwitch.getInstance().getOperation(name);
20 if (operation.equals("f(CustomRobustaException)"))
21     throw new CustomRobustaException("This Exception is thrown from try/catch block, so the bad smell is removed.");
22 }
23
24 Object around():(call(* *(..)||call(*.new(..))) && withincode (* ChatUtils.*(..)){
25     try{
26         return proceed();
27     }catch(Exception e){
28         return null;
29     }
30 }

```

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

```

15 @Test
16 public void testDueToCloseFinallyShouldNotThrowAnyExceptionInSaveChartAsPNG() {
17     aspectJSwitch.initResponse();
18     aspectJSwitch.addResponse("writeChartAsPNG/f(CustomRobustaException)");
19     aspectJSwitch.addResponse("close/f(IOException)");
20     aspectJSwitch.toFirstResponse();
21     try{
22         ChartUtils.saveChartAsPNG(null, null, 0, 0, null);
23     } catch (CustomRobustaException e) {
24         e.printStackTrace();
25         Assert.assertEquals("This Exception is thrown from try/catch block, so the bad smell is removed.", e.getMessage());
26     } catch (Exception e) {
27         e.printStackTrace();
28         Assert.fail("Exception is thrown from finally block.");
29     }
30 }

```

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

例外處理壞味道所在的

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，AddAspectMarkerResolutionForThrownFromFinally 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

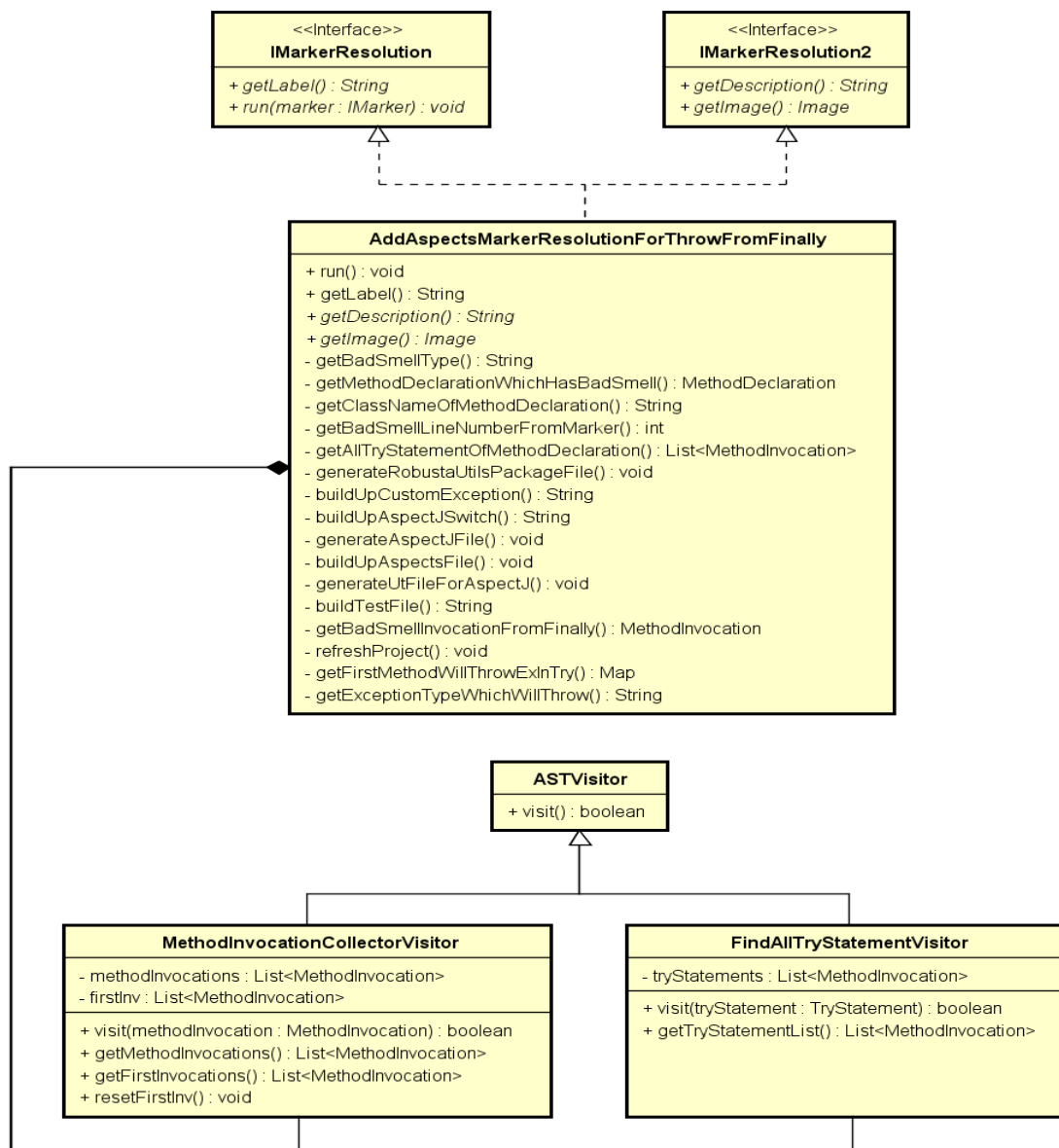


圖 4-20 組成 AspectJ 程式碼、測試案例相關的 Class Diagram

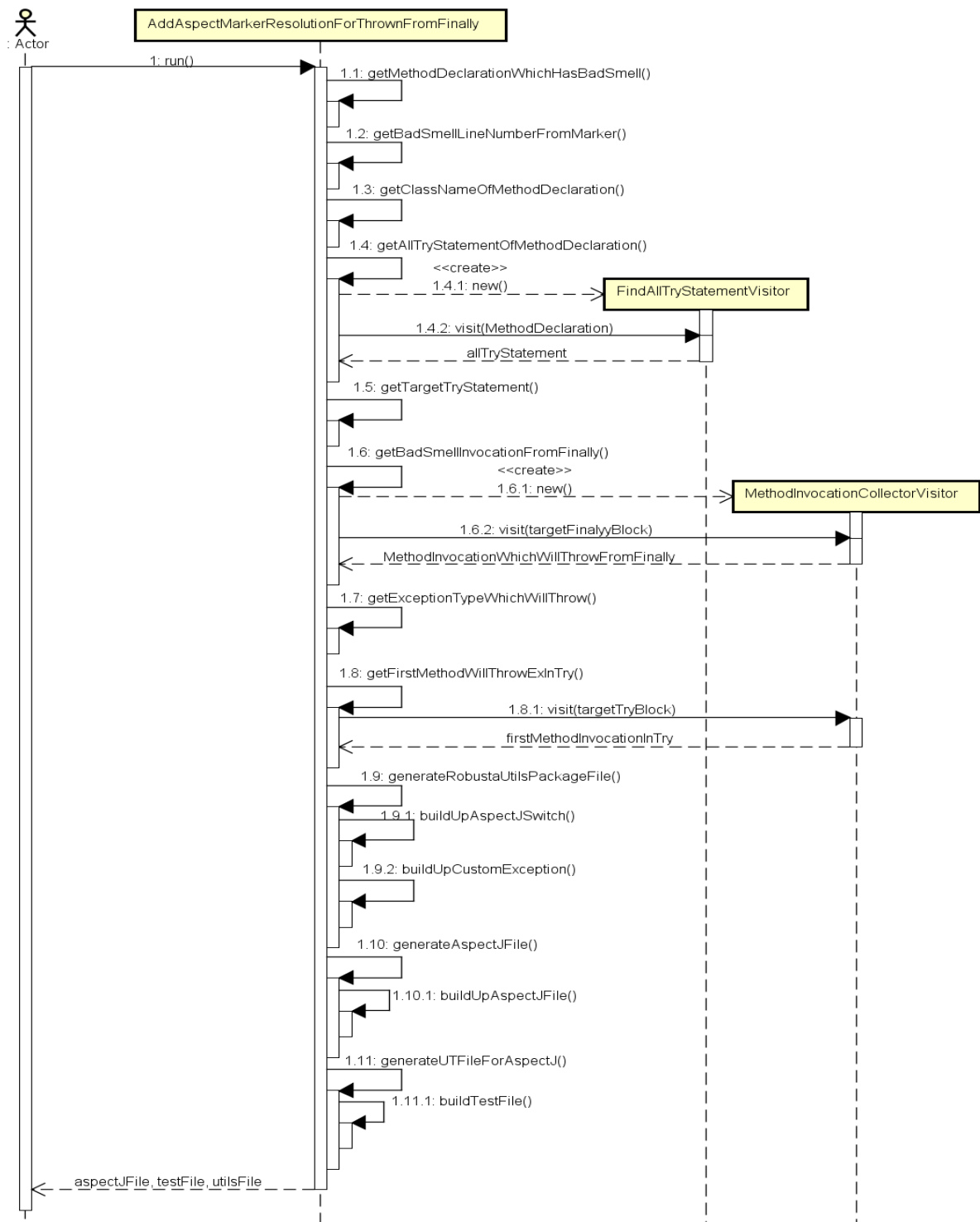


圖 4-21 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram

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。

```
304 public static void saveChartAsPNG(File file, JFreeChart chart,
305     int width, int height, ChartRenderingInfo info)
306     throws IOException {
307     Args.nullNotPermitted(file, "file");
308     OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
309     try {
310         ChartUtils.writeChartAsPNG(out, chart, width, height, info);
311     }
312     finally {
313         out.close();
314     }
315 }
```

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

```

15@
16 @Test
17 public void testDueToCloseFinallyShouldNotThrowAnyExceptionInSaveChartAsPNG() {
18     aspectJSwitch.initResponse();
19     aspectJSwitch.addResponse("writeChartAsPNG/f(CustomRobustaException)");
20     aspectJSwitch.addResponse("close/f(IOException)");
21     aspectJSwitch.toFirstResponse();
22     try{
23         ChartUtils.saveChartAsPNG(null, null, 0, 0, null);
24     } catch (CustomRobustaException e) {
25         e.printStackTrace();
26         Assert.assertEquals("This Exception is thrown from try/catch block, so the bad smell is removed.", e.getMessage());
27     } catch (Exception e) {
28         e.printStackTrace();
29         Assert.fail("Exception is thrown from finally block.");
30     }
31 }

```

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

```

10@
11 before() throws IOException : (call(* *.close(..) throws IOException)) && within(ChartUtils){
12     String name = thisJoinPoint.getSignature().getName();
13     String operation = AspectJSwitch.getInstance().getOperation(name);
14     if (operation.equals("f(IOException)"))
15         throw new IOException("This Exception is thrown from finally block, so it is a Exception Thrown From Finally Block bad smell.");
16 }
17@
18 before(): (call(* *(..) throws IOException)) && withincode(* ChartUtils.saveChartAsPNG(..)){
19     String name = thisJoinPoint.getSignature().getName();
20     String operation = AspectJSwitch.getInstance().getOperation(name);
21     if (operation.equals("f(CustomRobustaException)"))
22         throw new CustomRobustaException("This Exception is thrown from try/catch block, so the bad smell is removed.");
23 }
24@
25 Object around(): (call(* *(..)) || call(*.new(..))) && withincode (* ChartUtils.*(..)){
26     try{
27         return proceed();
28     } catch (Exception e){
29         return null;
30     }
31 }

```

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

```
166 @Override
167 public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException {
168     Iterator iterator = ImageIO.getImageWritersByFormatName("jpeg");
169     ImageWriter writer = (ImageWriter) iterator.next();
170     ImageWriteParam p = writer.getDefaultWriteParam();
171     ImageOutputStream ios = ImageIO.createImageOutputStream(outputStream);
172     Args.notNullNotPermitted(bufferedImage, "bufferedImage");
173     Args.notNullNotPermitted(outputStream, "outputStream");
174     p.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
175     p.setCompressionQuality(this.quality);
176     writer.setOutput(ios);
177     writer.write(null, new IIOMemorySource(bufferedImage, null, null), p);
178     ios.flush();
179     writer.dispose();
180     ios.close();
181 }
```

圖 4-25 Careless Cleanup 壞味道範例

```
9 after(): (call(* *.close(..) throws IOException)) && withincode(* SunJPEGEncoderAdapter.*(..)){
10     String name = thisJoinPoint.getSignature().getName();
11     String operation = AspectJSwitch.getInstance().getOperation(name);
12     if (operation.equals("AOPCheckResources"))
13         AspectJSwitch.getInstance().checkResource();
14 }
15
16 before() throws IOException: (call(* *.write(..) throws IOException) ) && withincode(* SunJPEGEncoderAdapter.encode(..)){
17     String name = thisJoinPoint.getSignature().getName();
18     String operation = AspectJSwitch.getInstance().getOperation(name);
19     if (operation.equals("f(IOException)"))
20         throw new IOException("This Exception is thrown from Robusta.");
21 }
22
23
24 Object around():(call(* *(..))||call(*.new(..))) && withincode (* SunJPEGEncoderAdapter.*(..)){
25     try{
26         return proceed();
27     }catch(Exception e){
28         return null;
29     }
30 }
```

圖 4-26 Careless Cleanup AspectJ 範例程式碼

```

15 @Test
16 public void testReleaseMethodShouldExecuteInEncode() {
17     aspectJSwitch.initResponse();
18     aspectJSwitch.addResponse("write/f(IOException)");
19     aspectJSwitch.addResponse("close/AOPCheckResources");
20     aspectJSwitch.toFirstResponse();
21     try{
22         SunJPEGEncoderAdapter object = new SunJPEGEncoderAdapter();
23         object.encode(null, null);
24         Assert.assertTrue(aspectJSwitch.isResourceCleanup());
25     }catch (Exception e) {
26         Assert.assertTrue(aspectJSwitch.isResourceCleanup());
27     }
28 }

```

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

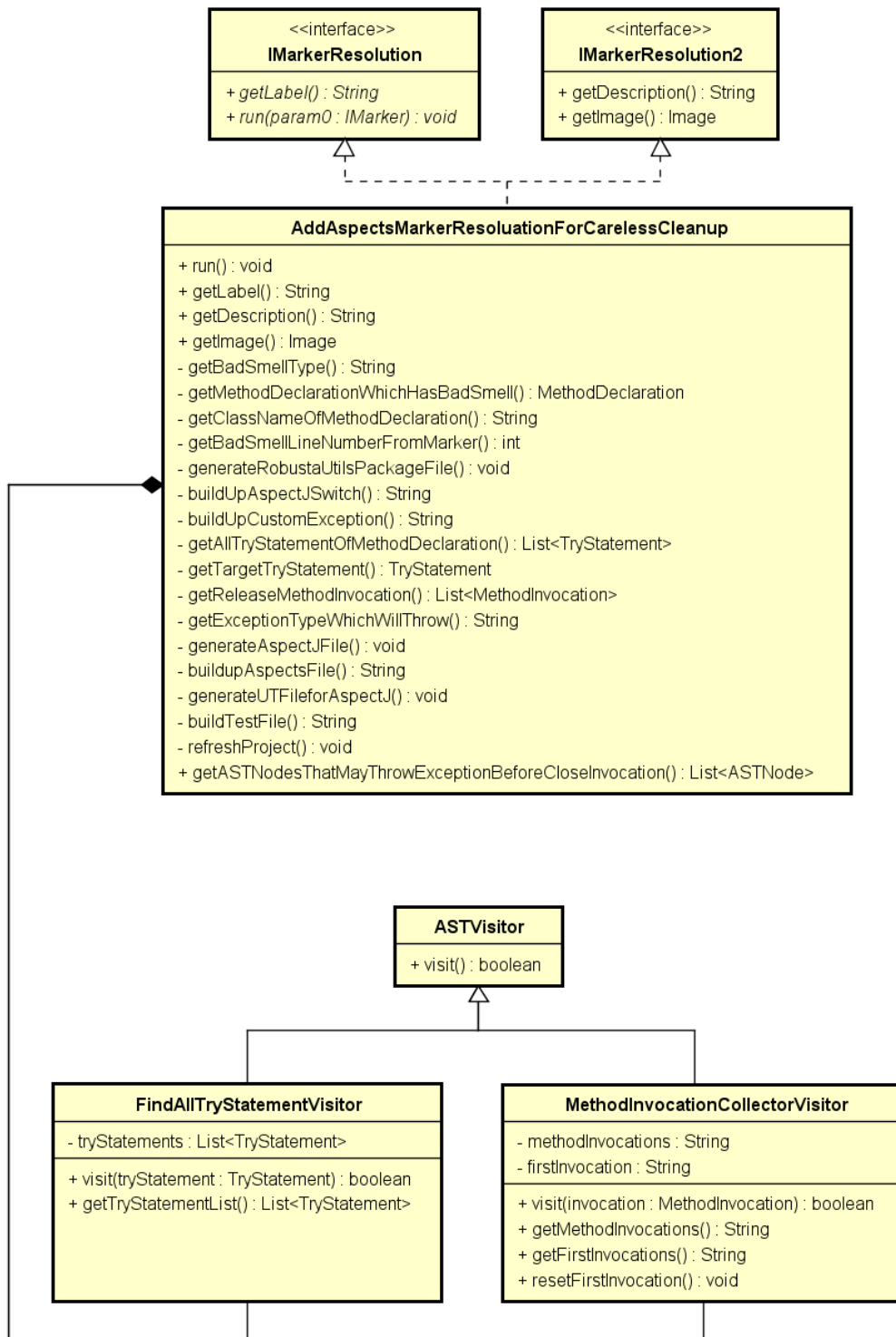


圖 4-28 組成 AspectJ 程式碼、測試案例相關的 Class Diagram

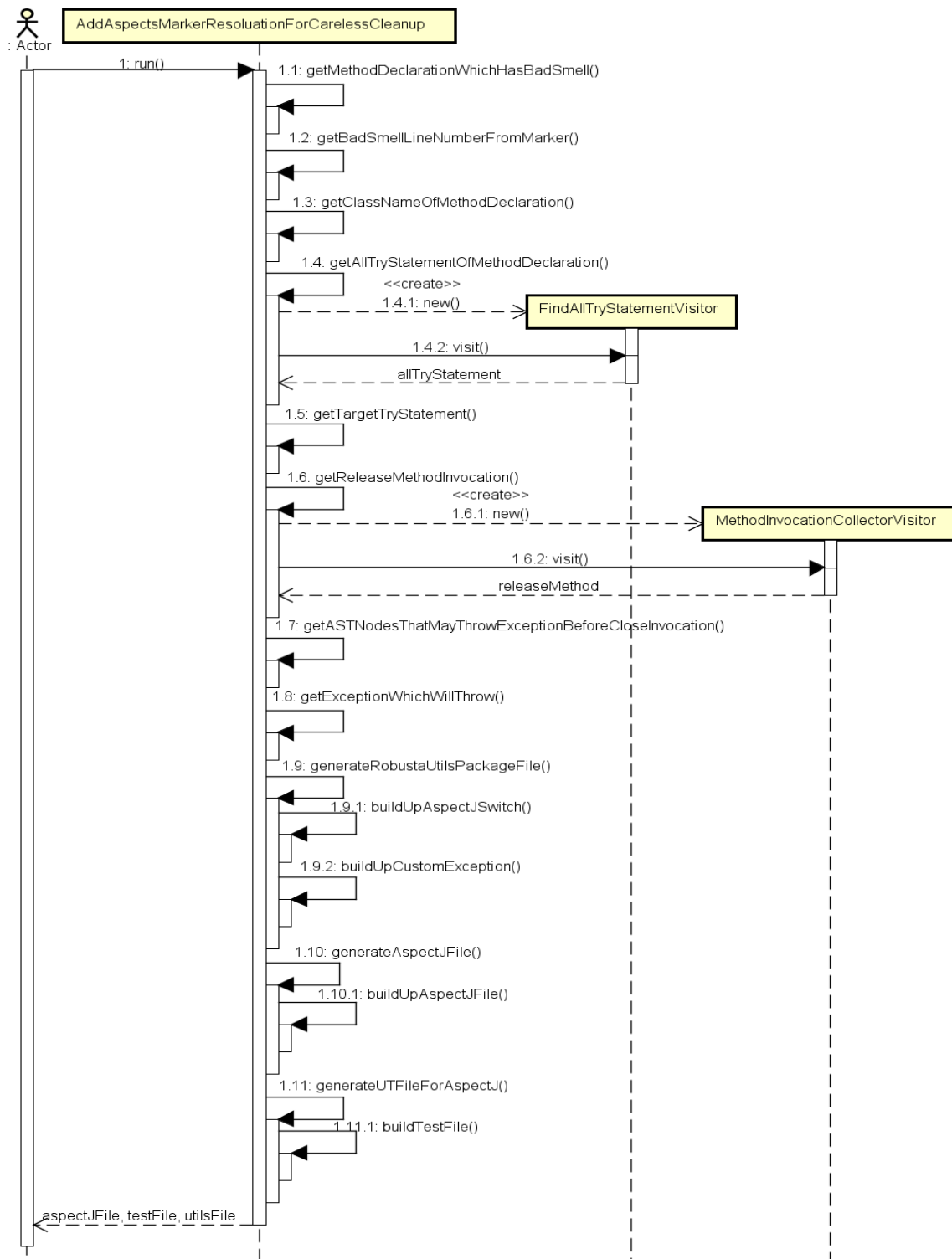


圖 4-29 組成 AspectJ 程式碼、測試案例相關的 Sequence Diagram

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。

```
169 @Override
170 public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException {
171     Iterator iterator = ImageIO.getImageWritersByFormatName("jpeg");
172     ImageWriter writer = (ImageWriter) iterator.next();
173     ImageWriteParam p = writer.getDefaultWriteParam();
174     ImageOutputStream ios = ImageIO.createImageOutputStream(outputStream);
175     Args.nullNotPermitted(bufferedImage, "bufferedImage");
176     Args.nullNotPermitted(outputStream, "outputStream");
177     p.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
178     p.setCompressionQuality(this.quality);
179     writer.setOutput(ios);
180     writer.write(null, new IIOMetadata(bufferedImage, null, null), p);
181     ios.flush();
182     writer.dispose();
183     ios.close();
184 }
```

圖 4-30 含有 Careless Cleanup 壞味道的程式碼

```
15 @Test
16 public void testReleaseMethodShouldExecuteInEncode() {
17     aspectJSwitch.initResponse();
18     aspectJSwitch.addResponse("write/f(IOException)");
19     aspectJSwitch.addResponse("close/AOPCheckResources");
20     aspectJSwitch.toFirstResponse();
21     try{
22         SunJPEGEncoderAdapter object = new SunJPEGEncoderAdapter();
23         object.encode(null, null);
24         Assert.assertTrue(aspectJSwitch.isResourceCleanup());
25     }catch (Exception e) {
26         Assert.assertTrue(aspectJSwitch.isResourceCleanup());
27     }
28 }
```

圖 4-31 重現 Careless Cleanup 壞味道發生例外的測試程式碼

```

9  after(): (call(* *.close(..) throws IOException)) && withincode(* SunJPEGEncoderAdapter.*(..)){
10  String name = thisJoinPoint.getSignature().getName();
11  String operation = AspectJSwitch.getInstance().getOperation(name);
12  if (operation.equals("AOPCheckResources"))
13      AspectJSwitch.getInstance().checkResource();
14  }
15
16  before() throws IOException: (call(* *.write(..) throws IOException) ) && withincode(* SunJPEGEncoderAdapter.encode(..)){
17  String name = thisJoinPoint.getSignature().getName();
18  String operation = AspectJSwitch.getInstance().getOperation(name);
19  if (operation.equals("f(IOException)"))
20      throw new IOException("This Exception is thrown from Robusta.");
21  }
22
23
24  Object around():(call(* *(..))||call(*.new(..))) && withincode (* SunJPEGEncoderAdapter.*(..)){
25      try{
26          return proceed();
27      }catch(Exception e){
28          return null;
29      }
30  }

```

圖 4-32 根據 Careless Cleanup 所製作的 AspectJ 程式

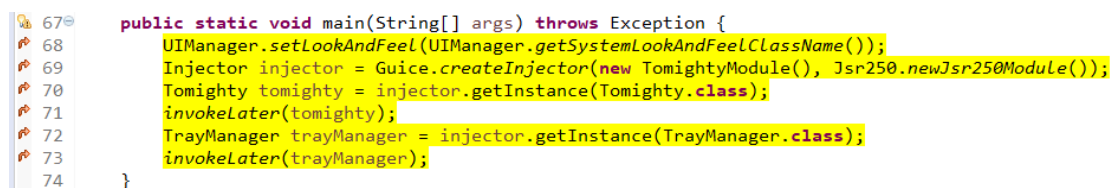
第五章 案例分析

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。



```
67 public static void main(String[] args) throws Exception {
68     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
69     Injector injector = Guice.createInjector(new TomightyModule(), Jsr250.newJsr250Module());
70     Tomighty tomighty = injector.getInstance(Tomighty.class);
71     invokeLater(tomighty);
72     TrayManager trayManager = injector.getInstance(TrayManager.class);
73     invokeLater(trayManager);
74 }
```

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

```

7  before() : (call(* *(..) ) ) && withincode(* Tomighty.main(..) ){
8      String name = thisJoinPoint.getSignature().getName();
9      String operation = AspectJSwitch.getInstance().getOperation(name);
10     if (operation.equals("f(RuntimeException)"))
11         throw new RuntimeException("Main Program is not surround with try/catch.");
12 }

```

圖 5-3 Unprotected Main Program AspectJ 程式碼

```

67 public static void main(String[] args) throws Exception {
68     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
69     Injector injector = Guice.createInjector(new TomightyModule(), Jsr250.newJsr250Module());
70     Tomighty tomighty = injector.getInstance(Tomighty.class);
71     invokeLater(tomighty);
72     TrayManager trayManager = injector.getInstance(TrayManager.class);
73     invokeLater(trayManager);
74 }

```

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。

```

12 @Test
13 public void testDueToSetLookAndFeelMainShouldNotThrowAnyException() {
14     aspectJSwitch.initResponse();
15     aspectJSwitch.addResponse("setLookAndFeel/f(RuntimeException)");
16     aspectJSwitch.toFirstResponse();
17     try{
18         String[] args={};
19         Tomighty.main(args);
20     }catch (Throwable e) {
21         Assert.fail(e.getMessage());
22     }
23 }

```

圖 5-5 與 AspectJ 搭配的測試案例

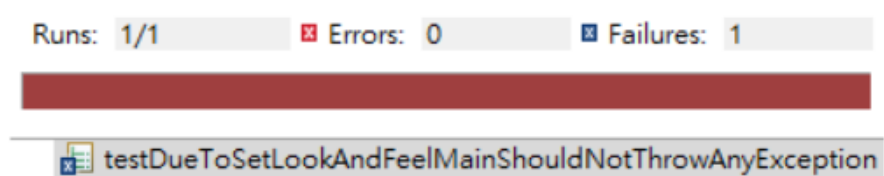


圖 5-6 測試案例 Failure 結果圖

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。

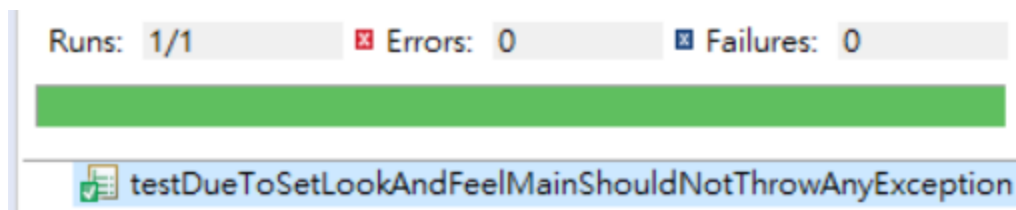
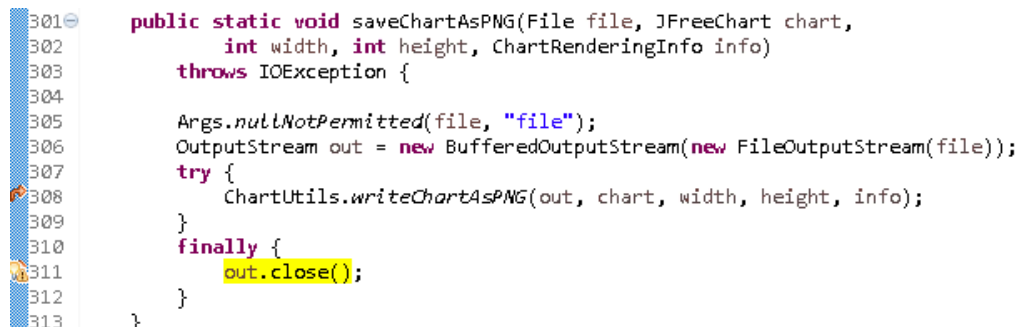


圖 5-7 測試案例 Pass 結果圖

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。



```
301 public static void saveChartAsPNG(File file, JFreeChart chart,
302     int width, int height, ChartRenderingInfo info)
303     throws IOException {
304
305     Args.nullNotPermitted(file, "file");
306     OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
307     try {
308         ChartUtils.writeChartAsPNG(out, chart, width, height, info);
309     }
310     finally {
311         out.close();
312     }
313 }
```

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。


```

10 before() throws IOException : (call(* *.close(..) throws IOException)) && within(ChartUtils){
11     String name = thisJoinPoint.getSignature().getName();
12     String operation = AspectJSwitch.getInstance().getOperation(name);
13     if (operation.equals("f(IOException)"))
14         throw new IOException("This Exception is thrown from finally block, so it is a Exception Thrown From Finally Block bad smell.");
15 }
16
17 before(): (call(* *(..) throws IOException)) && withincode(* ChartUtils.saveChartAsPNG(..)){
18     String name = thisJoinPoint.getSignature().getName();
19     String operation = AspectJSwitch.getInstance().getOperation(name);
20     if (operation.equals("f(CustomRobustaException)"))
21         throw new CustomRobustaException("This Exception is thrown from try/catch block, so the bad smell is removed.");
22 }
23
24 Object around(): (call(* *(..)) || call(*.new(..))) && withincode (* ChartUtils.*(..)){
25     try{
26         return proceed();
27     } catch (Exception e){
28         return null;
29     }
30 }

```

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

```

304 public static void saveChartAsPNG(File file, JFreeChart chart,
305     int width, int height, ChartRenderingInfo info)
306     throws IOException {
307     Args.nullNotPermitted(file, "file");
308     OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
309     try {
310         ChartUtils.writeChartAsPNG(out, chart, width, height, info);
311     }
312     finally {
313         out.close();
314     }
315 }

```

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。

```

15@ @Test
16 public void testDueToCloseFinallyShouldNotThrowAnyExceptionInSaveChartAsPNG() {
17     aspectJSwitch.initResponse();
18     aspectJSwitch.addResponse("writeChartAsPNG/f(CustomRobustaException)");
19     aspectJSwitch.addResponse("close/f(IOException)");
20     aspectJSwitch.toFirstResponse();
21     try{
22         ChartUtils.saveChartAsPNG(null, null, 0, 0, null);
23     } catch (CustomRobustaException e) {
24         e.printStackTrace();
25         Assert.assertEquals("This Exception is thrown from try/catch block, so the bad smell is removed.",e.getMessage());
26     } catch (Exception e) {
27         e.printStackTrace();
28         Assert.fail("Exception is thrown from finally block.");
29     }
30 }

```

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。

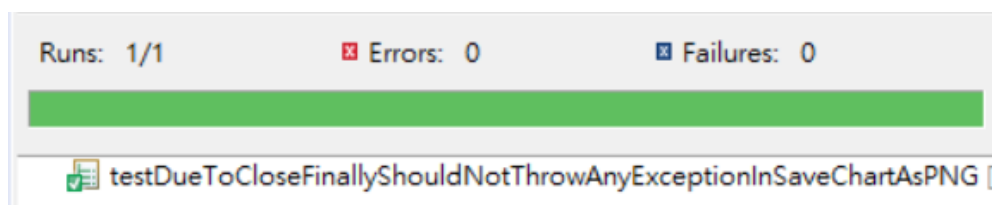
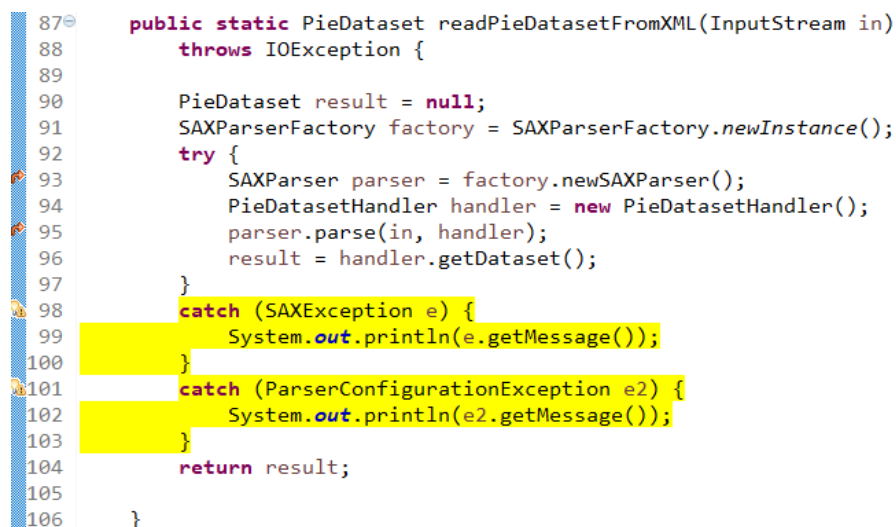


圖 5-12 測試案例 pass 結果圖

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。



```
87 public static PieDataset readPieDatasetFromXML(InputStream in)
88     throws IOException {
89
90     PieDataset result = null;
91     SAXParserFactory factory = SAXParserFactory.newInstance();
92     try {
93         SAXParser parser = factory.newSAXParser();
94         PieDatasetHandler handler = new PieDatasetHandler();
95         parser.parse(in, handler);
96         result = handler.getDataset();
97     }
98     catch (SAXException e) {
99         System.out.println(e.getMessage());
100     }
101     catch (ParserConfigurationException e2) {
102         System.out.println(e2.getMessage());
103     }
104     return result;
105 }
106 }
```

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。

```

9= before() throws SAXException: (call(* *(..)) throws SAXException) ) && withcode(* DatasetReader.readPieDatasetFromXML(..)){
10
11 String name = thisJoinPoint.getSignature().getName();
12 String operation = AspectJSwitch.getInstance().getOperation(name);
13 if (operation.equals("f(SAXException)"))
14     throw new SAXException("This exception is thrown from DummyHandler's unit test by using AspectJ.");
15
16 }
17
18
19= Object around():(call(* *(..))||call(*.new(..))) && withcode (* DatasetReader.*(..)){
20 String method = thisJoinPoint.toString();
21 if(method.contains("Exception(Throwable)")){
22     return proceed();
23 }else{
24     return null;
25 }
26 }

```

圖 5-14 Dummy Handler AspectJ 程式碼

```

87= public static PieDataset readPieDatasetFromXML(InputStream in)
88     throws IOException, SAXException {
89     PieDataset result = null;
90     SAXParserFactory factory = SAXParserFactory.newInstance();
91     try {
92         SAXParser parser = factory.newSAXParser();
93         PieDatasetHandler handler = new PieDatasetHandler();
94         parser.parse(in, handler);
95         result = handler.getDataset();
96     }
97     catch (SAXException e) {
98         System.out.println(e.getMessage());
99     }
100    catch (ParserConfigurationException e2) {
101        System.out.println(e2.getMessage());
102    }
103    return result;
104 }

```

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。

```

14@
15 public void testDueToNewSAXParserCatchBlockShouldThrowExceptionInReadPieDatasetFromXML() {
16     aspectJSwitch.initResponse();
17     aspectJSwitch.addResponse("newSAXParser/f(SAXException)");
18     aspectJSwitch.toFirstResponse();
19     try{
20         DatasetReader.readPieDatasetFromXML(null);
21         Assert.fail("Exception is not thrown from the catch block.");
22     } catch (Exception e) {
23         String exceptionMessage = e.getMessage().toString();
24         Assert.assertTrue(exceptionMessage.contains("This exception is thrown from DummyHandler's unit test by using AspectJ.));
25     }
26 }
27 }

```

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。

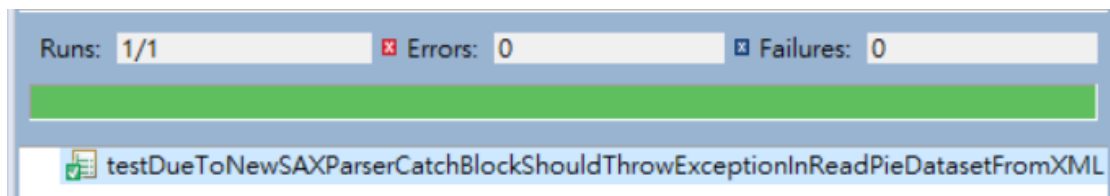


圖 5-17 測試案例 pass 結果圖

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。

2. 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 Marketplace, Available:
<https://Marketplace.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