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：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兩個開源專案來做實驗對象。結果顯示，Robusta針對壞味道特性產生的AspectJ程式碼和測試案例，可以曝露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：

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 |

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

Exception handling bad smells are the symptom in the source code of a program that possibly decrease the software robustness.Robuats, a programmatic bad smell detection tool build on Java, it is capable of detecting exception bad smell handling for Java code.

In order to expose the impact of Exception Handling Bad Smells, in this thesis we apply AspectJ and Test cases to show the Exceptional circumstances. When Test cases execute to a specific code which has the bad smell. By using AspectJ to inject throw excpetion code to display the Excpetion bad smells handling path so that show the impact of Excpetion handling bad smells on software.

In this experiment, we apply Robusta to analyze two popular open source projects, JFreeChart and Tomighty.The result indicates that Robusta can expose the impact of exception bad smells handling by generating the specific AspectJ and Test cases associated to the Excpetion Bad Smell type define by robusta. Also verify that bad smells in the code possibly decrease the software robustness.

目錄

[摘要 1](#_Toc515560163)

[ABSTRACT 2](#_Toc515560164)

[第一章 緒論 5](#_Toc515560165)

[1.1研究背景與動機 5](#_Toc515560166)

[1.2 研究目標 5](#_Toc515560167)

[1.3 論文組織架構 5](#_Toc515560168)

[第二章 背景知識與相關研究 6](#_Toc515560169)

[2.1 例外處理壞味道 6](#_Toc515560170)

[2.1.2 Dummy Handler 6](#_Toc515560171)

[2.1.1 Empty Catch Block 6](#_Toc515560172)

[2.1.3 Nested Try Statement 7](#_Toc515560173)

[2.1.4 Unprotected Main Program 7](#_Toc515560174)

[2.1.5 Exception Thrown From Finally Block 8](#_Toc515560175)

[2.1.6 Careless Cleanup 8](#_Toc515560176)

[2.2 Robusta 9](#_Toc515560177)

[2.3 AspectJ 9](#_Toc515560178)

[第三章 研究方法 11](#_Toc515560179)

[3.1暴露壞味道的方法 11](#_Toc515560180)

[3.1.1 Dummy Handler 11](#_Toc515560181)

[3.1.2 Empty Catch Block 11](#_Toc515560182)

[3.1.3 Unprotected Main Program 12](#_Toc515560183)

[3.1.4 Exception Thrown From Finally Block 12](#_Toc515560184)

[3.1.5 Careless Cleanup 13](#_Toc515560185)

[第四章 設計與實作 14](#_Toc515560186)

[4.1 Dummy Handler Empty Catch Block 14](#_Toc515560187)

[4.1.1 產生AspectJ、測試檔案的素材收集 14](#_Toc515560188)

[4.1.2 實作Dummy Hander & Empty Catch Block搜集素材 15](#_Toc515560189)

[4.1.3 分析測試搭配AspectJ與程式碼的互動 17](#_Toc515560190)

[4.3 Unprotected Main Program 18](#_Toc515560191)

[4.3.1產生AspectJ、測試檔案的素材收集 18](#_Toc515560192)

[4.3.2實作Unprotected Main Program 19](#_Toc515560193)

[4.3.3分析測試搭配AspectJ與程式碼的互動 21](#_Toc515560194)

[4.2 Exception Thrown From Finally Block 22](#_Toc515560195)

[4.2.1產生AspectJ、測試檔案的素材收集 22](#_Toc515560196)

[4.2.2 實作Exception Thrown From Finally Block 23](#_Toc515560197)

[4.2.3分析測試搭配AspectJ與程式碼的互動 25](#_Toc515560198)

[4.4 Careless Cleanup 27](#_Toc515560199)

[4.4.1產生AspectJ、測試檔案的素材收集 27](#_Toc515560200)

[4.4.2實作Careless Cleanup 28](#_Toc515560201)

[4.4.3分析測試搭配AspectJ與程式碼的互動 30](#_Toc515560202)

[第五章 案例分析 31](#_Toc515560203)

[5.1 Unprotected Main Program 案例分析 31](#_Toc515560204)

[5.1.1偵測並分析Protected Main Program 31](#_Toc515560205)

[5.1.2呈現Unprotected Main Program 對系統的影響 32](#_Toc515560206)

[5.1.3消除Unprotected Main Program 33](#_Toc515560207)

[5.2 Exception Thrown From Finally Block案例分析與實作 33](#_Toc515560208)

[5.2.1偵測並分析Exception Thrown From Finally Block 34](#_Toc515560209)

[5.2.2呈現Exception Thrown From Finally Block 造成的影響 34](#_Toc515560210)

[5.2.3消除Exception Thrown From Finally Block 35](#_Toc515560211)

[5.3 Dummy Handler案例分析與實作 36](#_Toc515560212)

[5.3.1偵測並分析Dummy Handler 36](#_Toc515560213)

[5.3.2呈現Dummy Handler 造成的影響 36](#_Toc515560214)

[5.3.3消除Dummy Handler壞味道 37](#_Toc515560215)

[第六章 結論 38](#_Toc515560216)

[6.1 結論 38](#_Toc515560217)

[6.2 未來展望 38](#_Toc515560218)

[參考文獻 38](#_Toc515560219)

# 第一章 緒論

## 1.1研究背景與動機

本論文根據陳友倫***以Aspect揭露導因於例外處理的程式缺陷***的論文為研究背景，其論文指出目前北科大軟體系統實驗室，正在開發一個靜態分析工具Robusta，它能分析出程式碼中的例外處理壞味道，幫助開發者在開發Java程式時，提升其品質及具備更好的強健度，因此如果在分析之後，分析出很多壞味道，如果不能證明這些壞味道是問題，開發者可能也不會正視例外處理壞味道所帶來的影響，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，來呈現例壞處理壞味道的方法。

跟據其研究，Aspect雖然可以允許開發者在不修改原始碼的狀況下，透過Aspect的特性，對程式嵌入例外，使其強制暴露出例外處理壞味道帶來的影響，但開發人員若是對Aspect語言特性不熟悉的話，需要額外花費更多程本來學習此語言。

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，讓開發者可以藉由強制丟出例外的過程，看到因為例外處理壞味道而對程式的影響。除此之外，將其實作在Robusta上使其自動的產生Aspect程式碼，對於開發者來說學習Aspect的門檻將大幅降低，也提升了Robusta的價值。

## 1.2 研究目標

本論文的目標將接續陳友倫針對Dummy Handler產生對應Aspect提出的做法，將剩餘的壞味道設計出其Aspect的嵌入壞味道的方法，並將其實作在Robusta上，讓Robusta定義的壞味道都能藉由自動產生的Aspect程式呈現對於程式的影響，讓開發人員正視例外處理壞味道所帶來的影響，對開發人員在撰寫例外處理程式時更有幫助。

## 1.3 論文組織架構

本論文分為六個章節，第一章節說明本論文的研究動機。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。第四章會說明本論文依據不同壞味道種類分門別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Java專案，呈現實作後的功能操作及暴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可改善及研究的方向。

# 第二章 背景知識與相關研究

## 2.1 例外處理壞味道

### 2.1.2 Dummy Handler

定義:

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

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

範例

如下圖2所示，第12行 writeFile函式會創立一個新檔案接著把資料寫入該檔案中。根據Java 的官方 API文件說明，第12行的new FileWriter(“k:\\test.txt”) 有可能會發生IOException。若程式執行到第12行時發生IOException，此例外將會被第16行的Catch Block接住，但Catch Block 僅記錄錯誤訊息。



圖2 Dummy Handler 壞味道範例

### 2.1.1 Empty Catch Block

定義:

「Empty Catch Block」此壞味道意旨當程式發生例外並捕捉例外後並忽略此例外

影響:

此壞味道的例外處理機制方式為不處理，此作法會隱藏潛在問題，會使開發者往後若遇到例外狀況發生，增加除錯的困難度。

範例:  
如下圖1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。第12行的new FileWriter(“k:\\test.txt”) 有機會發生IOException。若程式執行到第12行時發生IOException，此例外將會被第16行的Catch Block接住，但Catch Block 並未對其做處理，最終此壞味道將被忽略導致開發人員除錯困難度將會提升。



圖1 Empty Catch Block 壞味道範例

### 2.1.3 Nested Try Statement

定義:

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

影響:

對開發者來說，此壞味道複雜的結構將會影響程式碼的可讀性、測試性以及維護性。

範例：如下圖3所示，在第17行的Finally Block 會進行資源釋放的工作，然而很多關閉資原的函數都會丟出例外，用來代表釋放資源失敗。因此在Finally Block中很容易發生巢狀 Try Statement 的情況，使得程式碼結構變得複雜及難以維護。



圖3 Nested Try Statement 壞味道範例

### 2.1.4 Unprotected Main Program

定義:  
「Unprotected Main Program」此壞味道的意旨在程式碼中，主程式或執行緒沒有捕捉由下傳遞至自己身上的例外。

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



圖4 Unprotected Main Program 壞味道範例

### 2.1.5 Exception Thrown From Finally Block

定義:

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

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

範例:

如下圖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，此例外會被17行的Catch Block接住並往caller回傳此例外，但是再回傳之前會先進入19行的Finally Block，若此時21行的close函式也發生例外，此例外將會被22行的Catch Block接住並在23行將此例外往外丟，此時這個丟出去的例外將會把19行的例外覆蓋，導致開發者無從得知例外完整的例外狀況。



圖6 Exception Thrown From Finally Block壞味道範例

### 2.1.6 Careless Cleanup

定義:

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

影響:

此壞味道因為在釋放資源之前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:

如下圖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。根據Java 的官方 API文件說明，此函式在第14行 FileWriter函式、第15行 write、第16行flush均會丟出IOException，這些例外若發生在第17行釋放資源之前將會導致此close沒有被執行，導致資源持續被占用無法釋放。



圖5 Careless Cleanup壞味道範例

## 2.2 Robusta

Robusta 為一個以Java開發的開源專案，是一個靜態程式碼分析工具，主要的功能是用來協助開發人員偵測出其Java程式中存在的例外處理壞味道，來幫助開發者找出程式中的例外處理壞味道。目前已定義的六種例外處理壞味道，分別為Empty Catch Block、Dummy handler 、Nested Try Statement、Unprotected Main Program、Careless cleanup，除此之外還可以藉由產生報表的方式，統計壞味道的數量並提供例外處理壞味道的位置，讓使用者可以快速的查找被偵測的例外處理壞味道，大幅降低人工檢查大量程式碼的成本。

## 2.3 AspectJ

AspectJ 是一種基於Java實做AOP(Aspect-Oriented Programming)的程式語言，開發人員可以在不改動原有的程式碼下，額外增加原始碼的行為或改變狀態，再透過 before、after、around 決定要嵌入程式的時機。

將設計完的AspectJ程式與原始碼一起編譯並執行後，當程式執行到特定的函式時，就會在特定的時機觸發AspectJ 嵌入程式碼。本研究將會以AspectJ搭配測試暴露例外處理壞味造成的影響。

下圖7為一個寫檔的程式，若我們想要在第14行writer.write執行前利用AspectJ來嵌入程式碼，我們的AspectJ程式碼需要撰寫如下圖8，我們需要以下步驟

* 時機點:before表示在執行目標函式前我們會嵌入AspectJ程式碼
* 目標:Call表示我們的目標函式
* 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:為我們利用AspectJ想嵌入的程式碼

根據上面的步驟，我們再來回顧圖八，我們嵌入的時機點為before，意指在writer.write之前會嵌入程式碼，並且目標的範圍限定在Example.main中，最後嵌入的程式碼在圖8中的7~8行。

回到圖7，我們執行main程式後，發現在執行到14行時，Aspect幫我們嵌入了程式碼，並運行如圖9，結果呈現我們嵌入的程式碼，如此一來我們就可以不更改程式碼得狀態下嵌入程式。



圖7 Aspect目標的源始碼



圖8 AspectJ程式碼範例



圖9 嵌入AspectJ後運行程式碼的結果

# 第三章 研究方法

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

本論文將從陳友倫的研究為基礎，將其提出的方法應用於Robusta定義的壞味道中，讓所有的壞味道都能產生對應的AspectJ程式碼和測試，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暴露壞味道的方法

### 3.1.1 Dummy Handler

「Dummy Handler」壞味道為當程式發生例外並捕捉例外後，處理方式只印出或紀錄例外訊息，而沒有實質做處理。

若要讓壞味道對於軟體的影響現形(圖10 第14~16行為Dummy Handle壞味道)。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。

完成上面的步驟，我們就可以讓 Dummy Handler這個壞味道對於軟體的影響現形。



圖10 Dummy Handler 例外嵌入範例

### 3.1.2 Empty Catch Block

「Empty Catch Block」壞味道意旨當程式發生例外時，會將例外捕捉但選擇以忽略例外的方式進行處理，這樣的做法會隱藏潛在問題，會讓除錯難度提升也會降低程式的品質。  
 若要讓壞味道對於軟體的影響現形(圖11 第14~15行為Empty Catch Block壞味道)。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。

完成上面的步驟，我們就可以讓Empty Catch Block這個壞味道對於軟體的影響現形。



圖11 Empty Catch Block例外嵌入範例

### 3.1.3 Unprotected Main Program

「Unprotected Main Program」此壞味道的意旨在程式碼中，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13第69~74行為Unprotected Main Program壞味道)。利用測試搭配AspectJ在main program嵌入例外程式碼，使main program中使其發生例外，讓main program處在沒有try catch包覆的狀態下並發生例外狀況，藉此檢驗程式碼例外處理是否正確。

完成上面的步驟，我們就可以讓Unprotected Main Program這個壞味道對於軟體的影響現形。



圖13 Unprotected Main Program例外嵌入範例

### 3.1.4 Exception Thrown From Finally Block

「Exception thrown From Finally Block」壞味道意旨當在Finally Block中發生例外且此例外也被往外丟，因此在 Finally Block 發生的例外會覆蓋 Try Block 或 Catch Block 所發生的例外，覆蓋的例外處理訊息會誤導程式人員，難以得知該例外是由哪一個Block所造成。

若要讓壞味道對於軟體的影響現形(圖12第27行為Exception Thrown From Finally Block壞味道)。首先，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。

完成上面的步驟，我們就可以讓Exception Thrwon From Finally Block這個壞味道對於軟體的影響現形。



圖12 Exception Thrown From Finally Block例外嵌入範例

### 3.1.5 Careless Cleanup

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

若要讓壞味道對於軟體的影響現形(圖14 第177~178行為 Careless cleanup壞味道)。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。

完成上面的步驟，我們就可以讓Careless Cleanup這個壞味道對於軟體的影響現形。



圖14 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，如表1實作物件功能列表說明，AddAspectMarkerResolution\_[badSmellName]是繼承自Eclipse plugin的IMarkerResolution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，因應不同程式碼結構來做尋訪期內容，尋訪的過程也可以幫我們收集AspectJ、測試所需要的素材。

表1 實作物件功能列表

|  |  |
| --- | --- |
| 類別名稱 | 類別用途 |
| AddAspectMarkerResoluation\_[badSmellName] | 根據Visitor所蒐集到的資料  ，建立AspectJ程式碼。 |
| FindAllTryStatementVisitor | 利用Visitor的尋訪，找出給予特定區塊程式碼中的所有Try Block。 |
| MethodInvovationCollectorVisitor | 利用Visitor的尋訪，找出給予特定區塊的MethodInvocation。 |
| FindThrowSpecificExceptionStatementVisitor | 利用Visitor的尋訪，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，如下圖15為含有Dummy Handler壞味道的範例，為了要產生圖16會嵌入例外程式碼的AspectJ和圖17搭配AspectJ的測試檔案，我們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15第98行的Catch Block捕捉例外類型
2. 取得Try Block中與步驟一會發生相同例外類型的MethodInvocation，如圖15第93行的MethodInvocation會拋出與Catch Block捕捉相同的例外類型
3. 取得該壞味道MethodDeclaration 名稱，如圖15第87行的MethodName
4. 取得該壞味道所在MethodDeclaration所屬的Class名稱

做完上述的步驟後，就可以收集到組成AspectJ、測試所需的素材。



圖15 Dummy Handler 壞味道範例

捕捉例外的類型

壞味道所在的class及MethodDeclaration名稱



圖16 Dummy Handler AspectJ 範例程式碼

 圖17 Dummy Handler搭配AspectJ的測試案例

跟catch block捕捉相同例外的MethodInvocation

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

當使用者選擇Expose bad smell時，AddAspectMarketResolutionForDummyHandlerAndEmptyCatchBlock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，如圖15第98行的燈泡，來取得壞味道在該Java文件的MethodDeclaration、Dummy Handelr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. 藉由例外處理壞味道的位置取得所有Try Statement中，含有例外處例壞味道的目標Try Statement
5. 取得該目標Try Statement的Catch Block
6. 從Catch Block中拿到捕捉的例外處理類型
7. 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
8. 從步驟七取得的MethodInvocation中拿取第一個與Catch捕捉相同例外類型的MethodInvocation

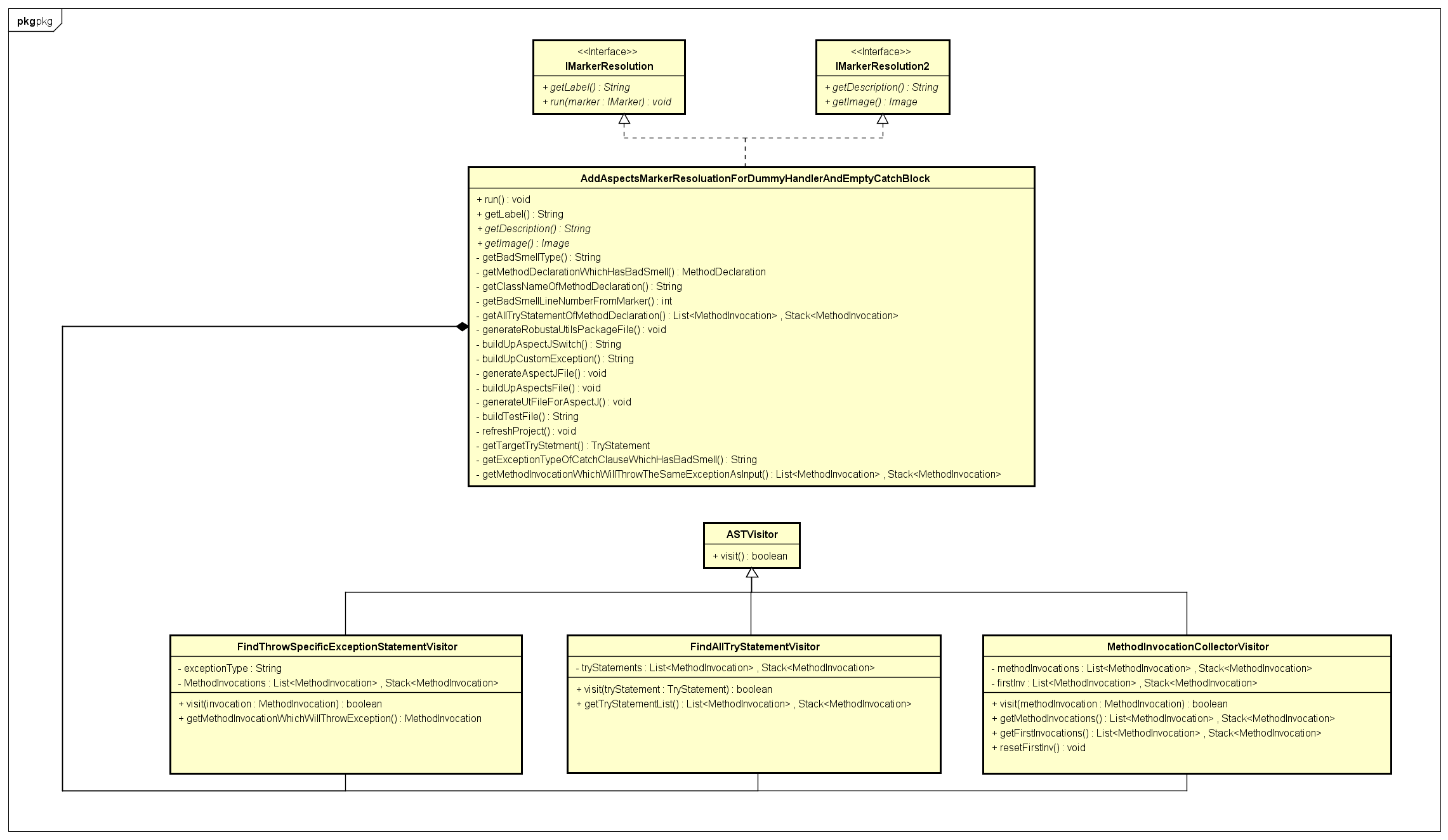


圖18 組成AspectJ程式碼、測試案例相關的Class Diagram

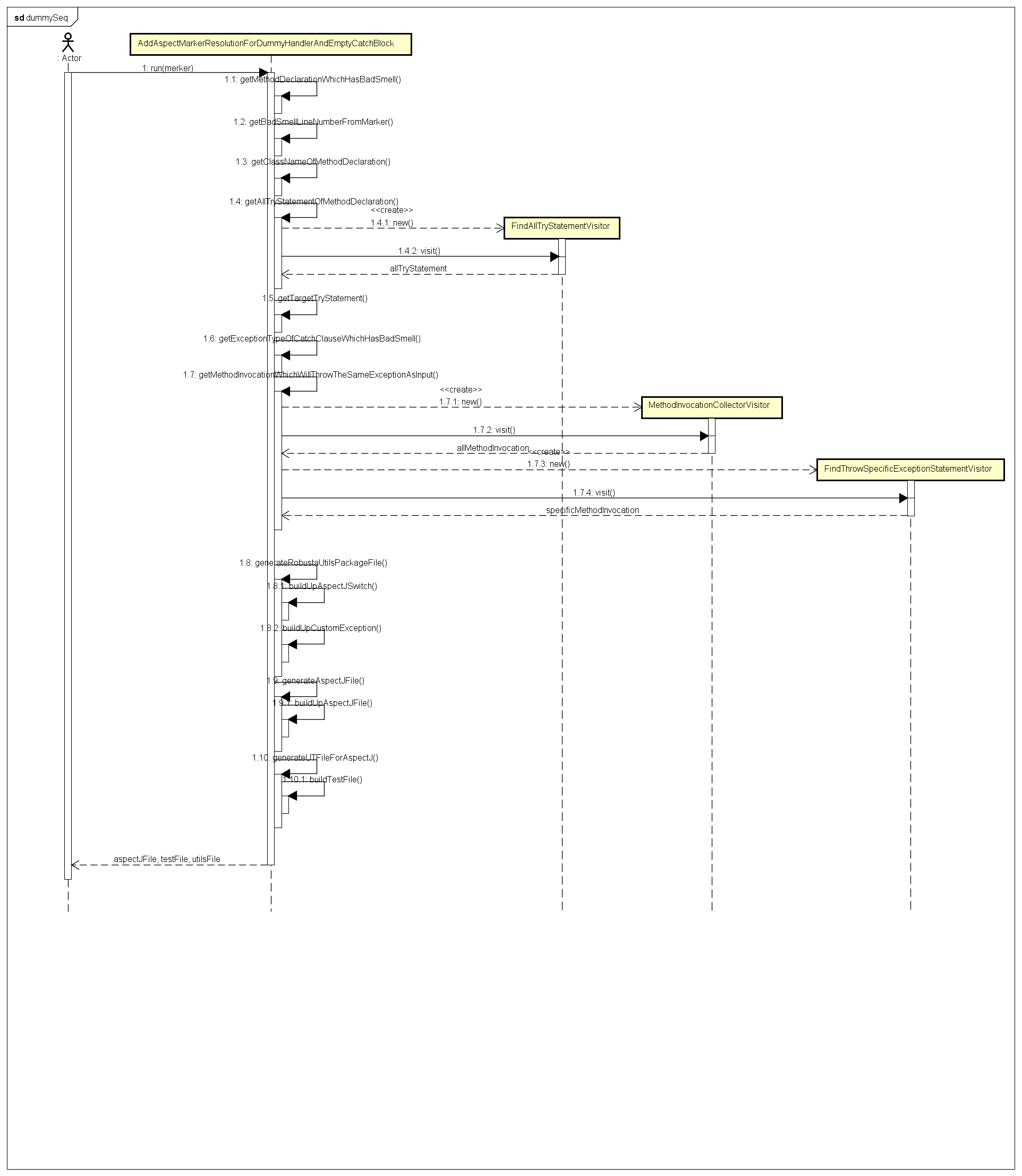


圖19 組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.1.3 分析測試搭配AspectJ與程式碼的互動

根據下圖20的程式碼，若Dummy Handler、Empty Catch Block的壞味道沒有被消除，則表示該程式在發生例外狀況的時候，採取的方式只有印出錯誤訊息、忽略例外。則此測試案例實作結果如圖21所示，我們利用AspectJ在Try Block中嵌入例外，因此在執行測試的時候AspectJ便會被啟動並嵌入例外，但含有例外處理壞味道的程式碼並沒有把例外拋出來，便會來到測試中Assert Fail的地方，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，則表示該程式在遭遇例外狀況的時候，根據Robusta建議方式，程式碼捕捉到例外後需將例外拋出去，因此在測試中被拋出的例外會被測試案例的Catch Block捕捉，則表示該段程式碼有對Dummy Handler、Empty Catch Block做處理，因此測試案例就通過了。



圖20 含有Dummy Handler壞味道的源始碼



圖21 重現Dummy Handler例外處理壞味道發生例外的測試程式碼



圖22 嵌入SAXException的AspectJ程式

## 4.3 Unprotected Main Program

### 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，如下圖31為含有Unprotected Main Program壞味道的範例，為了要產生圖32會嵌入例外程式碼的AspectJ和圖33搭配AspectJ的測試檔案，我們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31第68~73行的MethodInvocation
2. 取得Unprotected Main Program壞味道的 MethodDeclaration名稱，圖31第67行的MethodName

3. 取得Unprotected Main Program壞味道MethodDeclaration所屬的Class名稱



圖31 Unprotected Main Program壞味道範例

例外處理壞味道所在的MethodDeclaration



嵌入RuntimeException讓程式碼在任何情況下皆會發生例外

圖32 Unprotected Main Program AspectJ範例程式碼

Main Program 第一個要被嵌入RuntimeException 的 MethodInvocation



例外處理壞味道所在的MethodDeclaration

圖33 Unprotected Main Program搭配AspectJ的測試案例

### 4.3.2實作Unprotected Main Program

當使用者選擇Expose bad smell時，AddAspectMarketResolutionForUnprotectedMain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來取得壞味道在該Java文件的MethodDeclaration、Unproetced Main Program所在程式碼的位置。
2. 藉由取得壞味道所在的MethodDeclaration拿到所屬的Class名稱
3. 取得該MethodDeclaration內所有的MethodInvocation
4. 取得MethodDeclaration中所有的Try Statement
5. 將不在Try Statement內的MethodInvocation收集起來

藉由visitor做完上述的步驟後，就可以收集到組成AspectJ、測試所需的素材。



圖34 組成AspectJ程式碼、測試案例相關的Class Diagram

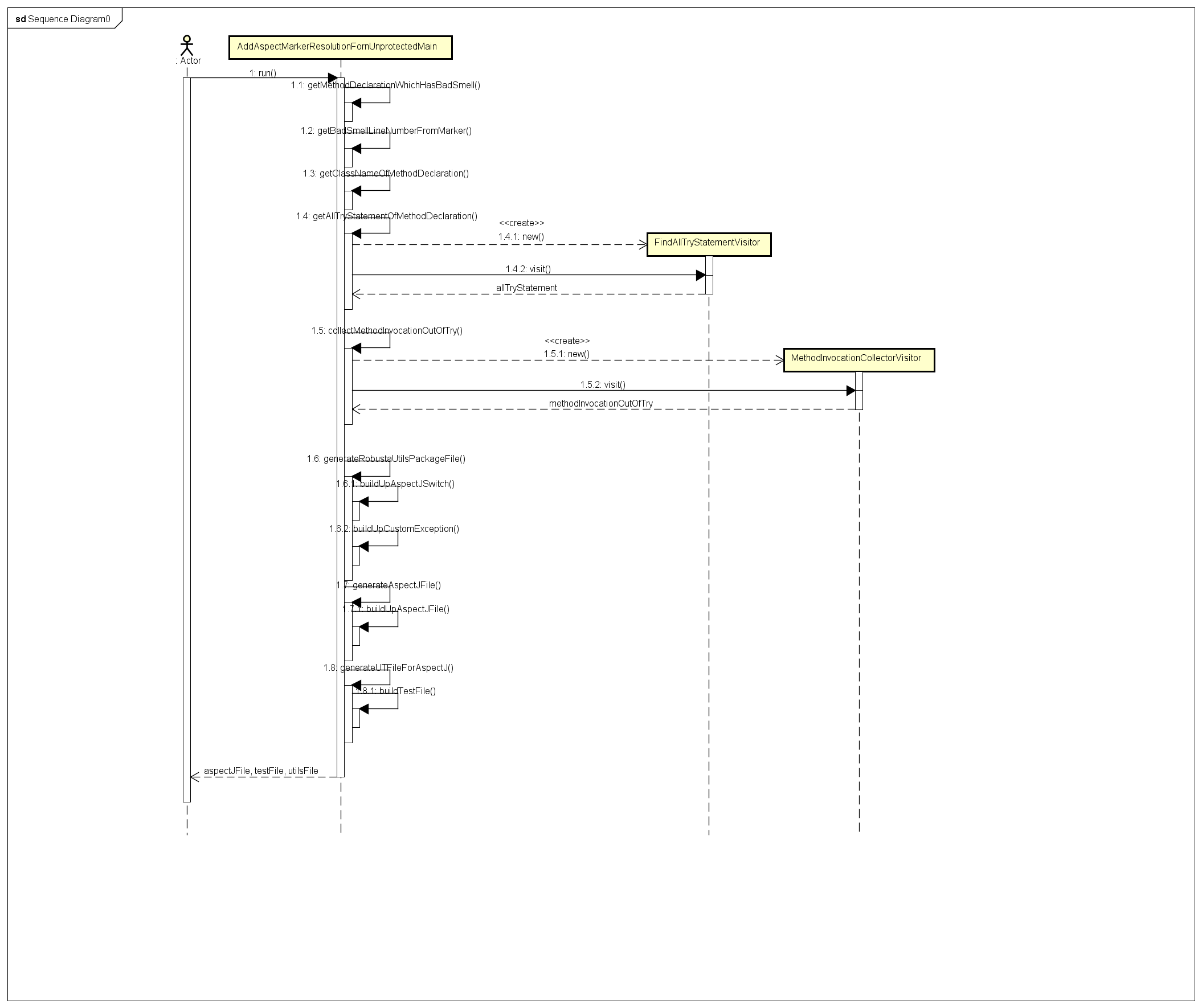


圖35組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.3.3分析測試搭配AspectJ與程式碼的互動

根據下圖36的程式碼，若Unprotected Main Program的壞味道沒有被消除，則表示主程式未捕捉從下拋出例外到主程式的時候，則該程式會不預期的終止。  
 測試案例實作結果如圖37所示，利用AspectJ在main program中對沒有Try Block包覆起來的MethodInvocation嵌入例外，因此在執行測試的時候AspectJ便會被啟動並嵌入例外，若存在著壞味道，則例外會被拋出來，來到該測試Catch Block的地方，因此就會出現Assert Fail的狀況，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，因此就不會來到測試的Catch Block，根據我們設計的測試案例若程式有對其壞味道做處理，這個測試案例就通過了。



圖36含有Unprotected Main Program壞味道的源始碼



圖37重現Unprotected Main Program壞味道發生例外的測試程式碼



圖38根據Unprotected Main Program所製作的AspectJ程式

## 4.2 Exception Thrown From Finally Block

### 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，如下圖23為含有Exception Thrown From Finally Block壞味道的範例，為了要產生圖24會嵌入例外程式碼的AspectJ和圖25搭配AspectJ的測試檔案，我們需要以下步驟來收集所需的素材。

1. 取得Exception Thrown From Finally Block壞味道中Finally Block 會丟出例外的MethodInvocation名稱及例外類型，如圖23第311行具有會丟出例外的MethodInvocation

2. Try Block中會丟出例外的MethodInvocation名稱，如圖23第308行會丟出例外的MethodInvocation

3. 取得該壞味道MethodDeclaration所屬的class名稱



圖23 Exception Thrown From Finally壞味道範例

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



圖24 Exception Thrown From Finally AspectJ 範例程式碼



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

例外處理壞味道所在的MethodDeclaration

圖25 Exception Thrown From Finally搭配AspectJ的測試案例

### 4.2.2 實作Exception Thrown From Finally Block

當使用者選擇Expose bad smell時，AddAspectMarketResolutionForThrownFromFinally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來取得壞味道在該Java文件的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中含有例外處例壞味道的目標Try Statement
5. 從目標Try Statement的Finally Block中拿到壞味道所在的MethodInvocation
6. 從步驟五取得的MethodInvoation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

藉由visitor做完上述的步驟後，就可以收集到組成AspectJ、測試所需的素材。



圖26組成AspectJ程式碼、測試案例相關的Class Diagram

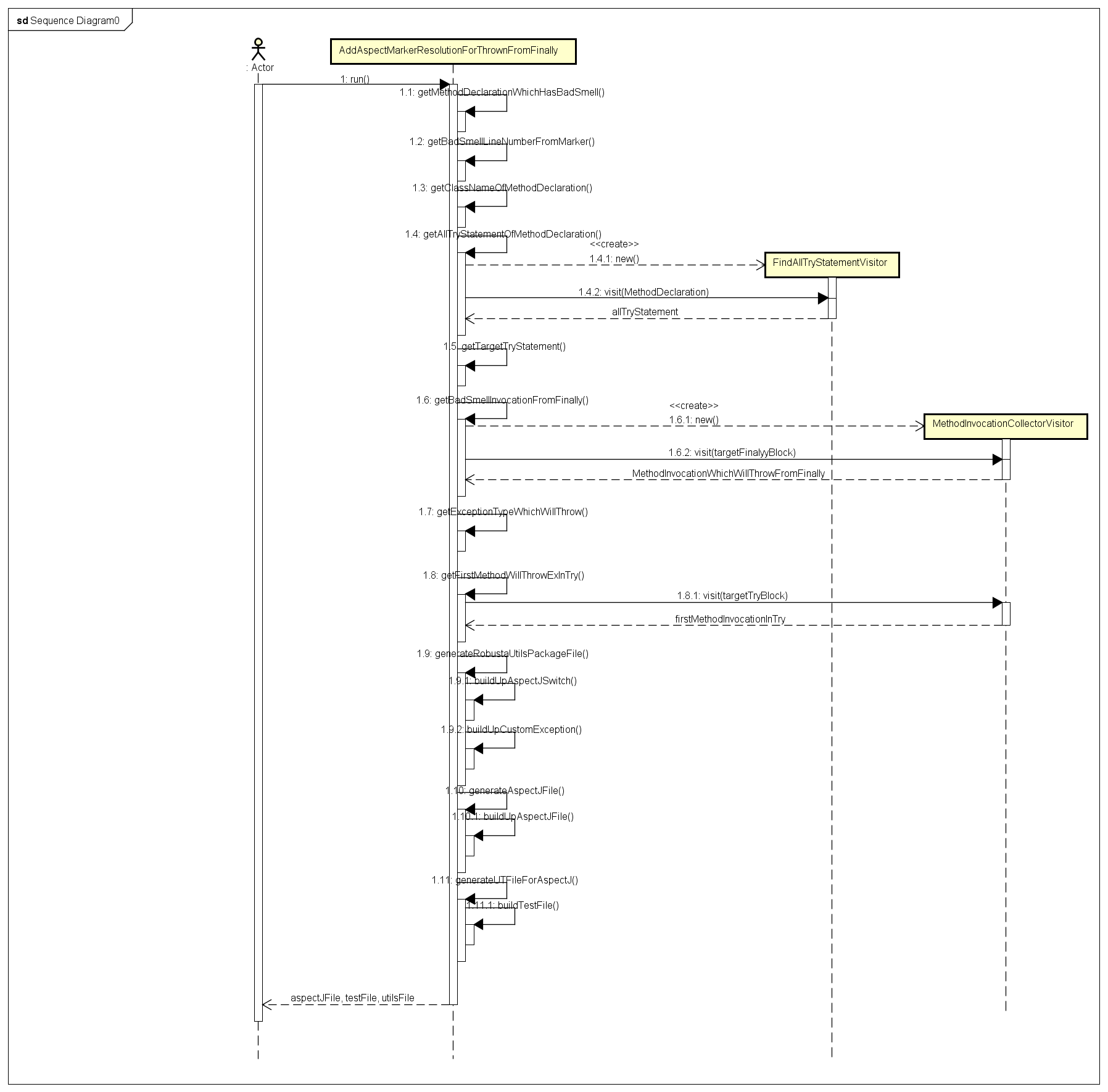


圖 27組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.2.3分析測試搭配AspectJ與程式碼的互動

根據下圖28的程式碼，若是在進入Finally Block前發生了例外，且Finally Block本身也會發生例外的話，則會造成Finally Block的例外會掩蓋Finally Block前所發生的例外。

測試案例實作結果如圖29所示，我們會設計兩個Catch Block，一個為自定義的CustomRobustaException、一個Exception 類別的Catch Block，此測試案例會因為AspectJ在原始碼中的Try Block中writeChartAsPNG嵌入客製的CustomRobustaException，並在Finally Block中的close也嵌入例外，如此一來當Try Block所產生的例外若被Finally Block的例外覆蓋住的話，變會來到Exception的Catch Block，因此就會出現Assert Fail的狀況，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，Robusta建議使用者不要在Finally Block中丟出例外，這個案例因為Try Block中丟出了例外沒有被Finally Block裡會發生的例外覆蓋，所以被客製的Catch Block捕捉，因此這個測試案例就通過了。



圖28含有Exception Thrown From Finally壞味道的源始碼



圖29重現Exception Thrown From Finally壞味道發生例外的測試程式碼



圖30 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39為含有Unprotected Main Program壞味道的範例，為了要產生圖40會嵌入例外程式碼的AspectJ和圖41搭配AspectJ的測試檔案，我們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39第180行的MethodInvocation

2取得Unprotected Main Program壞味道的 MethodDeclaration名稱，圖39第167行的MethodName

3取得Careless Cleanup壞味道MethodDeclaration所屬的Class名稱

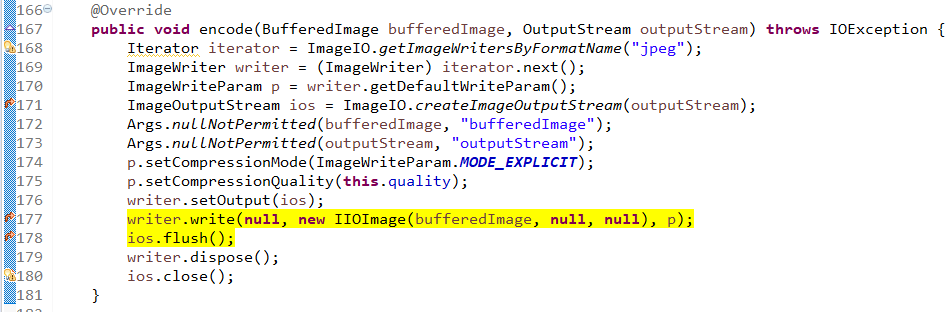


圖39 Careless Cleanup壞味道範例

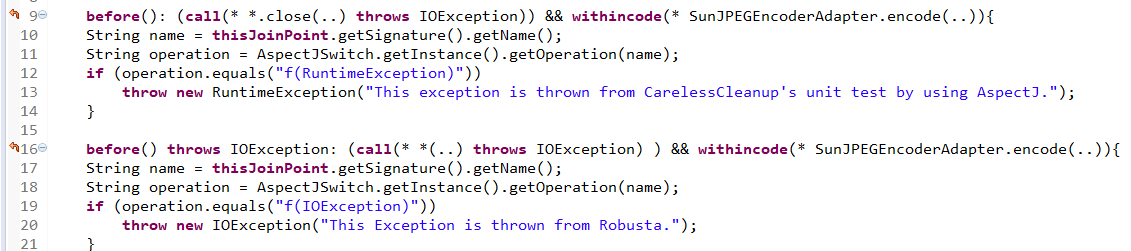


圖40 Careless Cleanup AspectJ範例程式碼

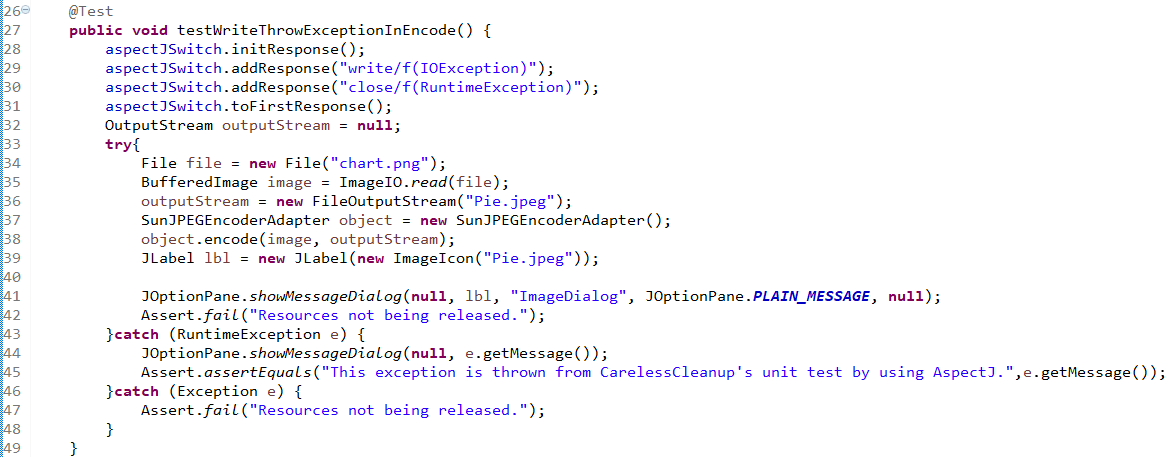


圖41重現Careless Cleanup壞味道發生例外的測試程式碼

### 4.4.2實作Careless Cleanup

當使用者選擇Expose bad smell時，AddAspectMarketResolutionForCarelessCleanup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來取得壞味道在該Java文件的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，含有例外處例壞味道的目標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

藉由visitor做完上述的步驟後，就可以收集到組成AspectJ、測試所需的素材。



圖42組成AspectJ程式碼、測試案例相關的Class Diagram

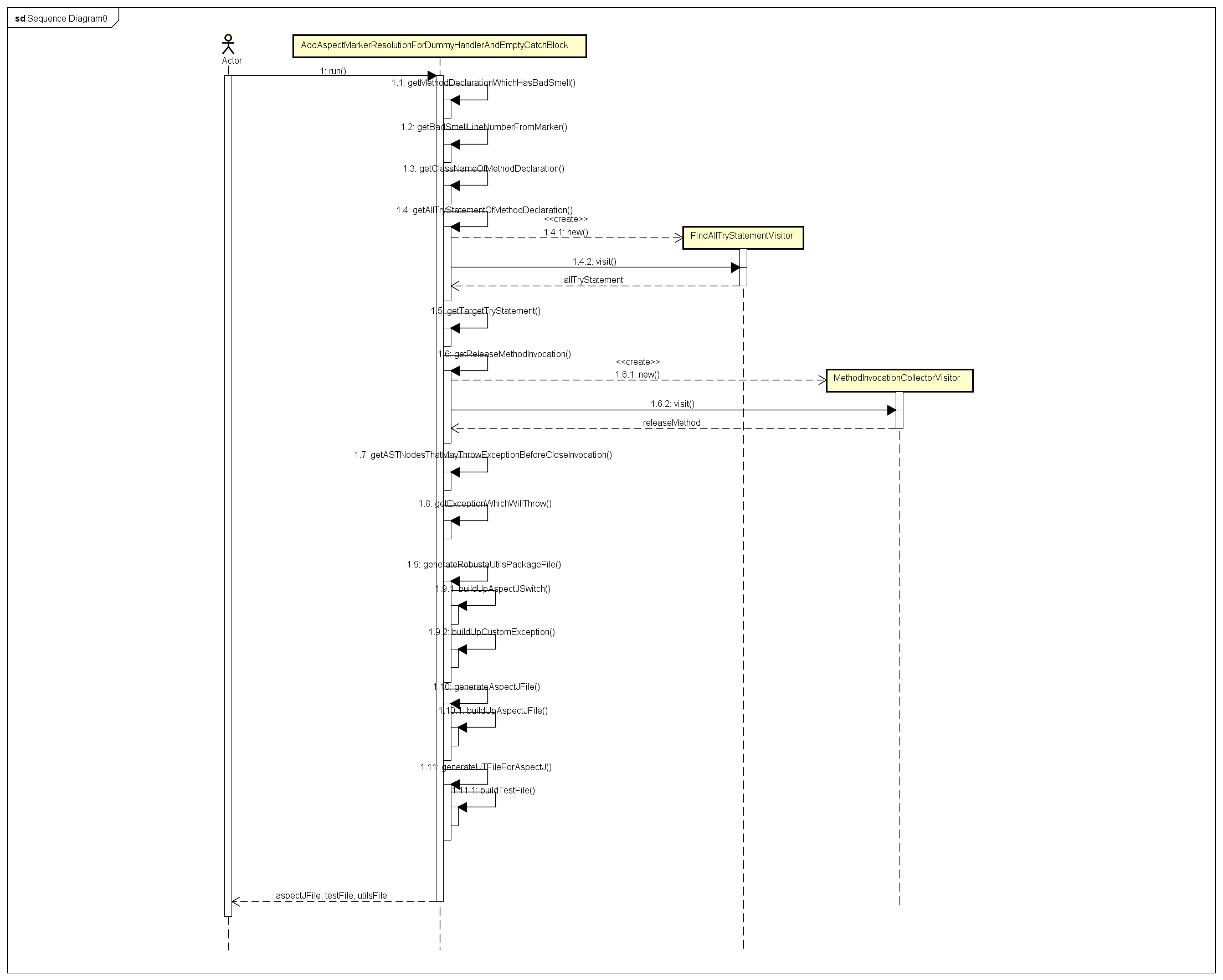


圖43組成AspectJ程式碼、測試案例相關的Sequence Diagram

### 4.4.3分析測試搭配AspectJ與程式碼的互動

根據下圖44的程式碼，我們可以發現若這樣的程式碼在close之前發生了例外，則會導致close不會被執行，而讓資源不會被正常釋放，所以根據這樣的情境我們設計了測試案例，我們的AspectJ會在離close method最遠且會發生例外的Method Invocation 嵌上其對應的例外，以及在close method 後嵌上檢查close是否被執行的判斷函式，在這樣的況下，若壞味道沒有被消除，則會來到我們自定義的Exception的Catch Block，因此就會出現Assert Fail的狀況，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，則最終我們會得到由Finally Block中release method所丟出來的例外，並被自定義例外的Catch Block給接起來，因此這個測試案例就通過了。



圖44



**圖45**

# 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暴露壞味道所帶來影響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Protected Main Program

如圖46所示，為Tomighty.java中的程式碼截圖。此段程式碼為程式的main program，根據Robusta的分析，此main program 並沒有被Try Block包覆起來，如果主程式或執行緒沒有捕捉由下拋出至身上的例外，則主程式或執行緒會不預期地終止執行，所以這是一種Unprotected Main Program的壞味道。



圖46 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，如下圖47所示，為程式正常啟動後的結果。



圖47 程式正常運行結果

為了觀察例外發生時壞味道造成的影響，我們藉由Robusta產生了一個AspectJ程式來幫助我們注入例外，強制暴露出壞味道帶來的影響，如下圖48，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行會幫我們丟出RunTimeException來幫我們暴露出這個壞味道造成的影響，完成AspectJ程式碼後，可以看到圖49的程式中產生了與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48 Unprotected Main Program AspectJ 程式碼



圖49 與AspectJ 相同規則產生對印嵌入符號的源始碼

為了讓例外在測試的狀態如期發生，我們還需要搭配測試才可以丟出例外，圖50第15行為要注入的MethodInvocation以及要丟出例外的類型。當測試呼叫了main程式因而執行到圖49中的第58行時，AspectJ程式注入了RunTimeException，此例外發生後使得Tomight.main主程式因為沒有對例外進行處理，所以程式便異常終止了，如此一來我們就可以把Unprotected Main Program這個壞味道帶來的影響給暴露出來，此外我們的測試案例會因為壞味道存在所以導致測試fail，讓使用者知道壞味道仍存在。



圖50 與AspectJ搭配的測試案例



圖51 測試案例fail結果圖

### 5.1.3消除Unprotected Main Program

根據楊雅雯論文提供的修復功能移除壞味道後，再次對此測試重新驗證一次，因為壞味道被消除了，所以儘管程式被注入例外，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，此外我們的設計案例會因為壞味道被消除所以測試案例通過，讓使用者知道壞味道已經消除了。



圖52 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

如圖53所示為ChartUtils.java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Thrown Exception From Finally的壞味道，根據文獻探討，很多關閉資源的函數都會丟出例外，用來代表釋放資源失敗，根據下圖53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的例外則會掩蓋住前Try Block 或是 Catch Block發生例外，所以這是一種Thrown Exception From Finally的壞味道。



圖53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

為了觀察例外發生時對壞味道造成的影響，我們藉由Robusta產生了一個AspectJ程式來幫助我們注入例外，強制暴露出壞味道帶來的影響，如下圖54，在第10、17行的地方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行會幫我們丟出IOException，

第21行會幫我們丟出自定義CustomRobustaExxception來幫我們暴露出這個壞味道造成的影響



為了讓例外在測試的狀態如期發生，我們還需要搭配我們的測試才可以丟出例外，第22行為要注入的methodInvocation以及自定義的例外，此外在

23行注入IOException的例外。如此一來，當程式執行到圖中的308行時，aspect程式注入了CustomRobustaException自定義的例外，使其產生例外，此實程式碼因為有Finally Block的關係，接續執行311行的釋放資源，同一時間aspect注入了IOException，根據此發法我們讓Try Block、Finally Block都使期發生例外，如此一來finally產生的例外便會覆蓋Try Block裡面所發生的例外，便可以將Exception Thrown From Finally Block這個壞味道帶來的影響給暴露出來，此外我們的設計案例會因為壞味道存在所以導致測試fail，讓使用者知道壞味道仍存在。





### 5.2.3消除Exception Thrown From Finally Block

根據楊雅雯論文提供的修復功能移除壞味道後，再次對此測試重新驗證一次，因為壞味道被消除了，所以儘管Finally Block被注入例外，但因為此層被Try Statement包覆住，捕捉例外不讓它在finally中將例外丟出去，並視需要記錄或在畫面上顯示清楚錯誤訊息，用此做法我們就可以消除此壞味道，此外我們的設計案例會因為壞味道被消除所以測試案例通過，讓使用者知道壞味道已經消除了。



## 5.3 Dummy Handler案例分析與實作

### 5.3.1偵測並分析Dummy Handler

本範例程式碼是從JFreeChart開源專案中取出，如圖所示，DataSetReader.java中的程式碼截圖。此段程式碼是幫助使用者去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，並進到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的壞味道。



### 5.3.2呈現Dummy Handler 造成的影響

為了觀察例外發生時對壞味道造成的影響，我們需要先取得壞味道catch中對應的check exception type，並根據此check exception type 在Try Block 找到對應的method invocation 並嵌入例外使其發生例外，藉由Robusta產生了一個AspectJ程式來幫助我們注入例外，強制暴露出壞味道帶來的影響，如下圖，在第10行的地方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15行會幫我們丟出SAXException，來幫我們暴露出這個壞味道造成的影響。



為了讓例外在測試的狀態如期發生，我們還需要搭配我們的測試才可以丟出例外，第31行為要注入的methodInvocation以及要丟出例外的類型。當程式執行到圖中的93行時，aspect程式注入了SAXException，此例外發生後使得readPieDatasetFromXML函式進入98 catch區塊，如此一來我們就可以把Dummy Handler這個壞味道帶來的影響給暴露出來，因此該method拿到的result為null，並將此錯誤的狀態傳遞下去，以此範例來說此結果會導致savechartasjpeg無法正常運作，進而存到一張資訊錯誤的圖，此外我們的設計案例會因為壞味道存在所以導致測試fail，讓使用者知道壞味道仍存在





### 5.3.3消除Dummy Handler壞味道

根據楊雅雯論文提供的修復功能移除壞味道後，再次對此測試重新驗證一次，因為壞味道被消除了，所以儘管try 區塊被注入例外，但因為catch將捕捉到的例外往外拋，讓外層去處理該例外，用此做法我們就可以消除此壞味道，此外被原始碼拋出的例外會進到圖中測試案例catch的區塊並進行assert，測試會因為壞味道被消除所以測試案例通過，讓使用者知道壞味道已經消除了。

****

# 第六章 結論

## 6.1 結論

根據陳友倫***以Aspect揭露導因於例外處理的程式缺陷***的論文，其論文提出以Robusta產生的AspectJ程式碼真的能發揮功效，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Aspect撰寫方式，也幫助開發者產生輕鬆的產生對應的Aspect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 Thrwon From Finally Block、Unprotected Main Program的壞味道，都對其設計並實做產生對應的AspectJ程式，並更改啟動的AspectJ的使用時機，如此一來將不會對正在運行的程式碼造成影響，而式限制在測試階段即可完成，除此之外，本論文的想法已經實做在robusta中，並對開源軟體JfreeChart、Tomighty進行暴露其壞味道造成的影響，可以成功的揭露壞味道代來的影響，讓使用者正視例外處理壞味道的處理方式。

## 6.2 未來展望

目前有以下幾點功能值得將來被實作

# 參考文獻

[1] Robusta at Eclipse Marketplace, https://marketplace.eclipse.org/content/robusta-eclipse-plugin [Accessed 20 December 2016]

[2] Aspectj at Eclipse, https://eclipse.org/aspectj/ [Accessed 20 December 2016]

[3] 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

[4] 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

[5] 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

[6] 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

[7] Chien-Tsun Chen. 例外處理設計的逆襲. 悅知文化，2014

[8] AspectJ plugin install guide, http://o7planning.org/web/fe/default/en/document/7174/install-aspectj-development-tools-into-eclipse

[9] ASTVisitor document,

https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTVisitor.html

[10] ASTNode document,

https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTVisitor.html