敏捷式例外處理方法：以 Scrum 為例
Agile Exception Handling in Scrum

謝金雲
國立臺北科技大學資訊工程系
hsieh@csie.ntut.edu.tw

陳建村[1]
美超微電腦股份有限公司
teddy.chen.tw@gmail.com

摘　要

本論文以 Scrum 為例，探討如何在採行敏捷方法的專案中，妥善地規劃例外處理設計。例外處理設計可增強系統強健性，屬於非功能需求的一種。傳統上軟體工程要求非功能需求應該儘量在程式撰寫之前就被設計完成，然而這一點並不符合敏捷方法所倡導的演進式設計以及反覆式開發與逐步成長的精神。本論文基於例外處理強健度模型、強健度故事、例外處理重構與工具等技術，提出一套完整的方法，以期解決上述問題。本論文所提出之方法已經過採行 Scrum 之軟體開發團隊實際驗證其可行性與有效性。

關鍵字：Scrum、例外處理、敏捷方法

一、前言

例外是許多當代的程式語言，例如 C++、Java、與 C#用來表達錯誤 (error) 的一種方式。相對的，例外處理則是程式語言所提供用以處理錯誤的一種機制，以避免錯誤變成系統層次的失敗 (failure)。妥善的處理例外將有效增進系統的強健度 (robustness) 並提高系統的可靠度 [2,5,8]。

然而，例外處理卻是一件困難且不易正確完成的工作。研究顯示一個系統中例外處理程式碼佔了所有程式的三分之二 [6]，且許多程式缺陷 (fault) 來自於例外處理程式。因此，如何有效地處理例外，便成為提昇軟體品質與可靠度的重要課題。由於近年來軟體大量的使用於各種設備之上，例如消費性電子裝置與手機等，軟體品質因素對於人們的影響也日益擴大，使得例外處理變得越加重要。

除此之外，例外處理本身也是一種程式設計，因此牽涉到軟體開發流程的問題。由於例外處理解決的是非功能面需求，因此在軟體專案中通常不是優先被考慮的因素，甚至完全被忽略。此一問題已被許多學者所提出，並建議應該在軟體開發早期 (需求分析階段) 就訂定例外處理需求 [3,5,15]，並依據此定義清楚的例外處理需求來分析與設計例外處理程式。**這些做法基本上假設軟體需求定義相當明確且穩定**，但對於使用敏捷方法或是演進式設計 (evolutionary design) 的軟體專案而言 [10]，並不容易直接採用。由於敏捷方法強調歡迎改變、以程式碼為主的設計、較短的軟體釋出週期、透過軟體重構來改善設計、撰寫自動化測試等等，因此，我們認為對於敏捷方法應該有另一套系統化且全面性的方法來支援例外處理設計。本論文將以近年來廣為被採行的敏捷方法 Scrum [13]為例子，說明在 Scrum 框架中如何無縫地做好例外處理設計以達到逐步提昇軟體系統強健度的目的。

本文其它章節組織如下。第二節簡介研究背景，包含問題分析與 Scrum 介紹。第三節說明本論文提出的方法。第四節說明兩個實際應用案例。最後為結論與未來研究方向。

二、研究背景

2.1 問題分析

從軟體開發流程的角度來看，要把例外處理做好是件困難的工作，其原因可歸納為下面幾點：

- **屬於非功能性需求的例外處理很容易被忽略**：大體上軟體開發的順序，是先從功能性需求開始做起，然後再考慮像是可用性，安全性，強建性，易用性等非功能性需求。在軟體開發專案普遍面臨開發時間不夠的問題，因此像是例外處理這類非功能性需求就經常成為被忽略與犧牲的對象。開發人員常常會對自己說：『我先把功能做出來，之後再來處理例外』。可是，通常等功能寫好之後，這些暫時被列為 TO DO 的例外，就永遠被遺忘了。
- **有些例外是需求面看不到的，要到實做時才會出現**：使用案例（use cases）方法建議在需求分析時撰寫 failure scenarios 來規劃當例外發生時系統應該如何因應。但是，有很多例外是和實做方法或是選擇的軟體元件有關，因此這種與實做相關的例外就無法被列在需求分析中（因為在問題領域中該例外並不存在或是不容易看出來）。
- **缺少例外處理實作知識**：由於訓練或相關知識不足，在實務上開發人員經常會遇到不知該如何處理例外的情況。例如，在 Java 程式中呼叫與 IO 相關的 methods 時，會丟出 `IOException` 這個 checked exceptions。Checked exceptions 表示 Java 編譯器會檢查收到這類例外的程式有沒有遵循 catch or declare 這個規則 [15]：如果你使用到某個會丟出 checked exceptions 的程式，那麼你必須 (1) 寫一個 `try` block 把這個例外抓下來，或

[1] 第二作者為 Certified ScrumMaster。

是 (2) 把這個例外宣告在你自己程式的介面上繼續往外傳遞。由於在需求分析時通常不會規範這類的實做例外要如何處理，因此只能依靠程式設計師的判斷能力視情況個別處理。有人會反射性的把例外捕捉之後直接忽略，然後假裝沒事繼續做其他功能；有人把例外往外丟，將這個難題交給下一個人；有人會把例外紀錄到日誌檔然後認為這樣就算是把例外處理好了；有的人捕捉這個例外然後丟出一個新的例外；有的人捕捉例外之後，嘗試修復例外造成的問題，但是通常越補問題越多。簡單的說，在大部分的情況下，由於不知道例外要如何處理，不同的程式設計師通常依據自己的喜好隨機做出決定，而這樣的決定將會降低而非提昇系統強健度。

上述問題在敏捷方法中依然存在。在敏捷方法中一個較大的需求可以切成若干個 stories，然後在不同的 sprints (iterations) 中逐一完成 [12]。然而如同絕大部分的非功能需求，**例外處理也是一種 cross-cutting concerns (http://en.wikipedia.org/wiki/Aspect-oriented_software_development)，通常不會順著我們切割 stories 的方向存在**。例如，你要撰寫一個網頁查詢股價的功能，在第一個 sprint 你開發使用基本欄位來查詢單一公司股價的功能，在第二個 sprint 你開發進階股價查詢功能，在第三個 sprint 規劃實做例外處理。雖然例外處理被規劃在第三個 sprint，但這並不表示在前兩個 sprints 中我們就不會遇到例外處理的問題。例如，查詢資料過程中可能會發生無法登入到資料庫的例外，資料庫連線中斷例外。如果這些例外因為在前兩個 sprints 中沒有被規劃所以被開發者隨意忽略，則有可能在第三個 sprint 中不容易找到這些例外處理漏洞而永久被遺漏，直到測試人員或使用者發現系統行為不正常而回報 bugs。

2.2 Scrum 簡介

Scrum 被稱為一種框架（framework），它主要規範一個軟體專案中的角色及其責任、必要的活動、產出物等三項內容 [13]。雖然 Scrum 並沒有規範必須實施哪些軟體開發實務作法，但實務上許多 Scrum 團隊普遍套用 XP 所倡導的實務作法，例如自動化測試、持續整合、重構等等。

Scrum 定義了三種角色，分別為：Product Owner、Scrum Master 及 Developer (Team)。Product Owner 為 Stakeholders 代表，收集與了解產品需求，訂定需求重要性（優先順序），並確認專案團隊是否正確地完成需求。Scrum Master 為負責指引專案團隊正確地執行 Scrum。Developer 即為軟體專案開發團隊成員。

Scrum 所規範的主要活動與產生出如圖 1 所示。Stakeholders 對於要開發的產品擁有一個願景，並對於這樣的願景而產生出對於產品的需求。Product Owner 會收集這些的需求並將其寫成 stories。專案所有的 stories 都被放在一個 Product Backlog 中（排定優先順序的 stories 列表）。

在 Scrum 中一個開發週期(iteration)稱之為 Sprint，當 Sprint 開始時，Product Owner、Scrum Master 及 Developer 一起參與 Sprint Planning Meeting，挑選出該 Sprint 所要實做的 stories，並針對完成每一個 story 所需的工作再細分出若干個 tasks。這些屬於這個 Sprint 的 stories 與 tasks 就稱為 Sprint Backlog。

在 Sprint 進行中，每天團隊舉行一個 15 分鐘的 Daily Scrum Meeting。當 Sprint 結束時，團隊進行 Sprint Review Meeting，在該會議中 Developer 向 Product Owner 展示在此 Sprint 中所完成的功能，藉此從 Product Owner 得到立即的回饋。最後，在 Retrospective Meeting 中團隊討論關於如何持續改善開發流程與軟體品質的議題。整個專案反覆若干個 Sprint 一直到專案完成（產品釋出）為止。

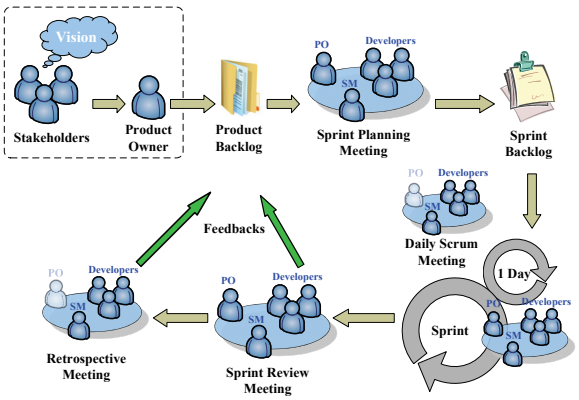

圖 1 Scrum 流程圖

三、敏捷例外處理方法

3.1 設計原理

本研究所提出的方法遵循下列幾項原則：

支援演進式設計：傳統在需求確定下所採行的大量前期設計（big up-front design）方法已不適用於敏捷專案之中 [10]。本研究所提出的方法必須要支援演進式設計，讓開發者可依據專案時程與資源的特性自行決定系統所需的強健度，且可以允許開發者在不同的 iteration 逐步增強系統強健度而不會導致設計混亂或是大量不必要的重工（rework）。

為例外處理需求訂定可操作式定義：傳統採用列表或是 constraint cards [12] 紀錄非功能需求的方式容易被遺忘且不易被落實到個別的 story 中。例如，假設專案定義了『當例外發生時 100% 不可以造成任何資料錯誤』這種強健度需求。理論上這是合理的要求，但是當開發人員實做每一個 story 而需要操作（落實）此強健度需求時，又會出現『我先把功能做出來，之後再來處理例外』的症狀。要解決此問題，除了強健度需求的定義要能夠簡單明

瞭且容易被落實之外，還需要支援以**逐步成長**的方式來增強系統強健度，如此才有可能讓專案團隊自行決定當時程緊迫時(例如所開發的軟體在三個月之後必須參展）可接受較低的強健度，而在後續的開發週期中再依據實際需要決定是否提昇強健度。

符合 Scrum 框架：即使開發團隊有了支援演進式設計以及可操作的例外處理需求，如果沒有一套系統化的機制讓開發團隊可以定期在『開發新功能』與『增強系統強健度』之間做出取捨，則增強強健度這種非功能需求還是極可能被遺忘。本研究所提出的方法必須能夠直接融入 Scrum 框架，利用 Scrum 的回饋機制來防止非功能需求被遺忘。

相容現有敏捷實務作法：敏捷團隊經常利用設計樣式、軟體重構、自動化測試等實務作法。本研究所提出的方法必須能夠與現行流行之敏捷實務作法相容，以便減少敏捷團隊學習的時間並增加其採用的意願。

提供工具支援：許多敏捷團隊所採用的實務作法，例如軟體重構或自動化單元測試，雖然不需要工具支援也可實施，但是有了工具支援之後，將會使得這些作法實施起來更方便且更有生產力。本研究所提出的方法應考慮到自動化與工具支援的因素。

3.2 方法內容

針對 3.1 小節所描述的五點原則，我們分別提出下列作法。

採用四種強健性等級以支援支援演進式設計：為了支援演進式例外處理設計，我們提出定義軟體強健性等級方法 [4]。這個強健性等級可用來作為定義例外處理需求。有了需求，開發人員遇到例外的時候就知道要如何處理才能滿足這個需求。以下簡述這四個強健性等級的意義。

- G0：如果你還沒有幫你的需求或程式定義強健性等級，那麼你的系統就屬於 G0 等級。G0 表示當某個 service (可以想成整個系統，呼叫某個元件，SOA 中的 service call 或是一般的 function call) 發生錯誤的時候，可能會讓呼叫者知道錯誤發生，也有可能會假裝沒事 (failing implicitly or explicitly)。也就是說使用該 service 的人，其實是無法確切得知它是否有成功達成任務。而當錯誤發生的時候，service 處於不明或是錯誤狀態 (state) [2]。例外發生時系統可能會終止也可能繼續執行 (terminated or continued)。
- G1：G1 表示當某個 service 發生錯誤的時候，一定要讓呼叫者知道，絕對不能假裝沒事 (failing explicitly)。因此，使用該 service 的人便可確切得知它是否有成功達成任務。而當錯誤發生的時候，service 可能處於不明或是錯誤的狀態。例外發生時系統要終止執行（因為此時狀態已經不明，所以繼續執行下去可能會讓整個系統錯得更離譜，所以要立刻終止）。要達到 G1 強健度等級很簡單，就是把所有的例外都往外丟，然後在主程式 (main program) 捕捉所有的例外並回報給使用者知道。G1 又稱為 failing-fast。
- G2：和 G1 一樣，G2 要求當某個 service 發生錯誤的時候，一定要讓呼叫者知道，絕對不能假裝沒事 (failing explicitly)。和 G1 不同，G2 要求當錯誤發生之後，service 必須保證還是處於正確狀態 [2]。由於整個系統的狀態還是正確的，因此例外發生時系統可以繼續執行（continued)。要達到 G2 強健度等級就要多做兩件事情。第一件事情就是 service 要想辦法回復到別人呼叫它之前的正確狀態 (error recovery)。例如，如果該 service 修改了資料庫裡面的資料，當例外發生時就要執行 rollback（簡單說就是要 undo 之前修改過的狀態）。第二件事情就是釋放資源 (cleanup)。例如，把要來的記憶體，file handlers，connections 等資源釋放。G2 又稱為 weakly tolerant。
- G3：此等級的系統要求使命必達。因此，當某個 service 發生錯誤的時候，要另外想辦法排除困難達成任務。和 G2 相同，G3 要求當錯誤發生之後，service 必須保證還是處於正確的狀態。 由於整個系統的狀態還是正確的，因此例外發生時系統可以繼續執行（continued)。要達到 G3 強健度等級除了要做到 G2 的 error recovery 和 cleanup 之外，還需要想其他方法達成原本的任務。這些其他方法包含 retry，design diversity, data diversity, functional diversity 等等。G3 又稱為 strongly tolerant。

如圖 2 所示，強健性等級具有逐步成長（incremental）與包含（inclusive）的特性。具有逐步成長特性是因為隨著等級提昇強健性也跟著提昇；具有包含特性是因為高一級的強健性等級包含下一級的所有作法。這兩個特性使得我們的方法具有支援演進式設計的能力。

撰寫強健度故事來定義可操作式例外處理需求：敏捷方法中所稱的 story，一般是用來紀錄功能性需

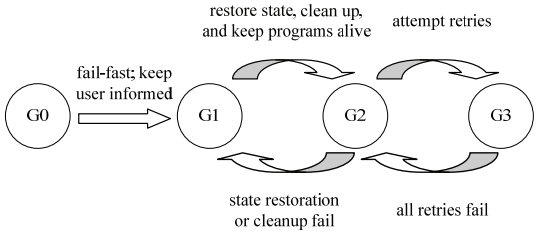

圖 2 例外處理強度的升級與降級

求。由於例外處理是屬於 cross-cutting concerns，因此在功能性需求中，或是非功能性需求中，都會遇到需要處理例外的情況。為此，我們建議：

- 定義 story 的強健度等級：由於達到不同強健度等級所需完成的工作十分明確（例如，達到 G2 需要作到狀態回復與錯誤回報），因此一旦定義了強健度等級則開發人員在實做 story 時若需要處理例外便有明確的依據（可操作性）。此外，藉由定義每一個 story 的強健度等級，可以促進 product owner 與開發人員之間的對話，以便釐清需要投入多少時間與資源來達成每一個 story 的強健性。此定義可視為該 story 之 definition of done 的一個項目。
- 撰寫強健度故事（robustness story）：將以條列方式或是 constraint cards 紀錄非功能需求的方式具體化，改寫成 robustness story，與其他正常功能的 story 一起放在 product backlog 中一起排定優先順序，以防止其被遺忘。以下為針對開放原始碼檔案同步軟體 SyncFree [16] 所撰寫的 robustness story 範例：『身為使用者，我希望使用 SyncFree 同步檔案時若使用者操作檔案的權限不足，不會造成同步資料錯誤』。完成此 robustness story 之後使用者便可知道該軟體不會因為權限不足所發生的例外導致資料損毀。

符合 Scrum 框架：有了 story 與 robustness story 並為其定義強健度等級之後，便可無縫地融入 Scrum 框架。我們將這兩種 stories 都一起放入 product backlog 中，讓 product owner 來決定實做的優先順序。在 sprint planning meeting 中，product owner 與 developer 藉由互動過程共同估算每個 story 的 story point（把該 story 的強健度等級考慮進去），之後將每個 story 細分為若干個 tasks。在 daily scrum 與 sprint 進行中，developer 若是對於達成某個強健度等級的作法有任何疑問或困難，便可隨時與其他 developer 討論、尋求 scrum master 協助、甚至要求 product owner 調整該強健度等級（降級或是升級）。在 sprint review meeting 中 developer 展示每個 story 已達到其所要求的強健度等級。最後，在 retrospective meeting 中，developer 可分享好的強健度等級實做方法與有待改進的作法，以作為日後持續改善的依據。

相容現有敏捷實務作法：到目前為止我們提到了如何從需求與流程（Scrum 框架）上考量例外處理的問題，接下來我們將說明如何實踐這些例外處理需求。由於例外處理實做也是一種『寫程式』，因此如果沒有良好的設計能力與實做技巧，則就算是開發團隊為每一個 story 都定義了強健度等級也是枉然，甚至會因為例外處理不良而產生更多的例外處理錯誤。我們建議套用下列作法來滿足開發團隊所定義的強健度等級。

- 例外處理樣式與實務作法：現有研究已提出許多關於例外處理的樣式與實務作法[1,15]。我們認為開發團隊除了學習設計樣式增強對於正常功能設計的能力之外，應該也要學習如何設計與撰寫例外處理程式（例外處理設計之議題在大學課程與業界中經常被忽略）。
- 例外處理壞味道：為了避免與找出程式碼中的例外處理問題，開發團隊應該要瞭解如何辨識例外處理壞味道 [4]。
- 例外處理重構：瞭解了例外處理壞味道便可採用例外處理重構技術來重整程式碼改善例外處理設計的品質 [4]。
- 例外處理測試：針對程式的例外處理能力應撰寫自動化測試案例加以驗證。參考目前的主流方式，我們建議先針對 robustness stories 撰寫自動化驗收測試（acceptance tests），再針對每一個 method 撰寫自動化單元測試 [9]。

提供工具支援：例外處理工具支援的範圍涵蓋很廣，包含例外註記 [18]、例外偵測 [18]、例外處理壞味道偵測與修復 [4]、自動化例外處理重構 [4, 17]、例外處理程式碼視覺化分析 [18]、自動產生例外處理測試案例、以靜態或動態程式碼分析找出例外處理錯誤 [7,18] 等等。目前我們所開發的 Eclipse 外掛工具針對 Java 語言已可涵蓋上述前五項範圍。

- 例外註記：以程式語言所提供的註記機制（例如 Java annotations 或 C# custom attributes）直接將程式強健度等級紀錄在程式碼中。直接將強健度等級紀錄在程式碼的好處符合敏捷方法以程式碼為主的設計的精神，並可支援例外處理自動化工具分析。
- 丟出例外提醒：自動偵測程式設計師所呼叫的 methods 是否會回傳例外，以提醒使用者應注意例外處理問題。在 Java 語言中，編譯器對於 checked exceptions 提供工具支援，但是對於 unchecked exceptions 並沒有。雖然 Java 的設計者並不建議使用者處理 unchecked exceptions，然而由於有許多廣為使用的開放原始碼軟體，例如 Spring，Hibernate，Eclipse SWT 都以 unchecked exceptions 來代替 checked exceptions 的使用，因此這類的工具越來越有其必要性。
- 例外處理壞味道偵測：許多現有系統對於例外處理並不完善，因此極可能存在大量的例外處理壞味道。自動化例外處理壞味道偵測與修復工具將可大大縮短開發人員找出問題的時間。
- 自動化例外處理重構：藉由自動化例外處理重構可減少程式設計師手動重構的時間並減低重構所可能發生的錯誤。
- 例外處理程式碼視覺化分析：當程式結構或例外傳遞很複雜，或是開發人員對於所要改善的

程式碼不是很瞭解的情況下，藉由視覺化工具將可協助開發人員從例外處理的角度來瞭解所要分析的程式碼。常見的工具有例外傳遞圖形與程式碼 call chain 分析圖。

- 靜態或動態程式碼分析：透過程式碼分析技術，可以由靜態或動態分析找出程式中潛在的例外處理問題。
- 自動產生例外處理測試案例：藉由撰寫強健度等級，合約（contract），與靜態程式碼分析，可自動產生例外處理測試案例以減輕開發人員撰寫測試案例的負擔並提高測試涵蓋率。

由於強健度等級具有支援演進式設計的特性，若敏捷團隊真的沒有時間處理例外，至少可採取 G1 等級，以確保所以發生的例外都被回報且沒有任何例外被忽略。由於達成 G1 的實做方法非常簡單，相較於任由強健度處在 G0 的不確定狀態，我們認為本方法整體而言不但不會額外增加敏捷團隊的開發負擔，而可以確保系統的例外處理設計達到一定的品質，並為日後提昇系統強健度立下良好基礎。

四、應用經驗

本論文所提出的敏捷例外處理方法可適用於開發新專案與維護既有系統，在此我們簡述實際應用經驗。

4.1 開發新專案

我們在一個實際採行 Scrum 的業界團隊中應用本方法，其導入作法建議如下：

A. 花二至四小時教導團隊強健度等級觀念以及符合每個等級所需的基本例外處理方法。
B. 在沒有特別規定之下，預設所有的 classes 與 methods 一定要達到 G1。如此一來，在開發階段經由各種測試我們便可盡量找出應該處理而沒有被處理的例外。在做這個規定之前，有許多例外都被忽略了因此使用者介面上看不到錯誤，可是此時系統的狀態已經不對，導致除錯變得更加困難。乍看之下 G1 好像很不負責的把所有的例外都往外丟，但是反而可以在開發階段發現問題並加以修復。此時 developer 便可機動與 product owner 討論針對此例外情況是否必須由 G1 提昇至 G2 或 G3，因此整體而言提昇軟體的強健度。
C. 對於特定的操作，例如資料庫處理，由於有內建的交易處理控制機制，很容易可達到 G2，因此這類的操作一開始實做就必須達到 G2。
D. 除非 product owner 要求，或是設計過程中發現某個操作不達到 G3 會變得很難使用，否則 developer 不應該自行嘗試把程式提升到 G3。

該團隊採用本方法已有一年半時間，實務上相當程度提昇軟體強健度。主要原因為強健度等級這個觀念簡單易懂，而且也很符合敏捷方法的精神。因為強健度等級基本上就是秉持著『階段性，逐步改善例外處理設計』的作法，在許多情況下，正常功能還沒有全部完成時，是不容易決定例外處理到底應該做在整個系統的那一層。此時過早、過於精細的例外處理設計不見得有用，反而可能造成時間上的浪費。

4.2 維護既有系統

SyncFree 是一個由北科大資工系軟體系統實驗室所開發的檔案同步軟體。SyncFree在2003與2004年獲得由國科會自由軟體計畫贊助，分別於2004年與2005年釋出了1.0 與2.0版軟體。在2009年底我們獲得使用者回報關於使用 SyncFree 2.0 版所發生的若干錯誤（bugs），我們便著手採用本論文所提出的方法重新評估與改善 SyncFree 的強健度。目前此實驗尚在進行中，在此我們說明前期成果。以下為我們應用本方法的流程：

A. Product owner 針對使用者所回報的bugs或是使用者特別關注的系統強健度功能，撰寫 robustness stories，並設定其重要性。
B. Developers 於 sprint planning meeting 時挑選 robustness stories，為某一個 robustness stories 安排以下幾個 tasks。
    i. 使用 robot framework [14] 撰寫自動化驗收測試以確定該 robustness story 可正確執行或是處於錯誤狀態。
    ii. 使用我們所開發的 call chain analysis 工具，畫出實做該 story 的 call chains 以協助分析 call chain 上每一個 method 的強健度。圖3為使用該工具所畫出『身為使用者，我希望當來源或目的磁碟機容量不足時，執行雙向同步不會造成檔案同步資料錯誤』這個 robustness story 的 call chain。
    iii. 分析 call chains 上的每一個 method，並定義其強健度等級。
    iv. 為每一個 method 撰寫單元測試用以測試其是否滿足上一步驟所設定的強健度等級。若測試失敗則套用例外處理重構技術 [4] 修改程式直到測試通過為止。
    v. 最後，使用我們所開發的例外處例壞味道偵測工具 [18] 來自動找出該 call chains 所涵蓋程式碼中的壞味道，並加以修復。
    vi. 再次確認步驟 i 與步驟 iv 所撰寫的自動化測試都通過。
C. 於 Sprint Demo Meeting 時展示自動化驗收測試證明該 robustness stories 已經實做完成。

目前我們已經採用上述方法撰寫了十七個 robustness stories 並實際完成其中兩個實做。

五、結論與未來展望

本論文提出一套完整的敏捷式例外處理方

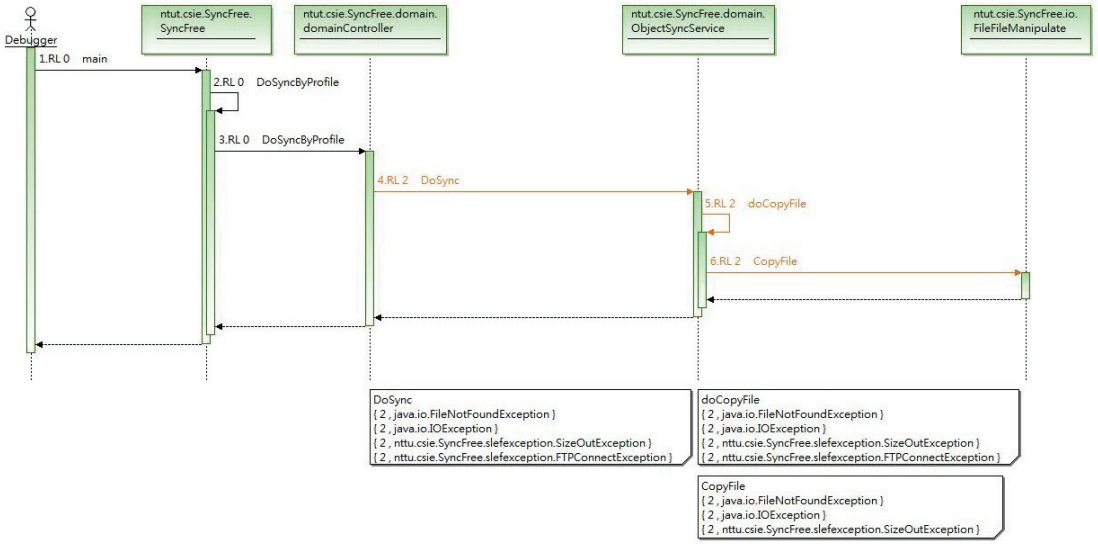

圖 3 由工具自動產生之 SyncFree call chain 循序圖

法，並以 Scrum 為例實際說明如何應用於新專案開發與舊專案維護。本方法已經應用於一個業界的 Scrum 團隊中，實際驗證藉由階段式改善例外處理的方式可提昇軟體強健度同時並可維持程式碼設計的品質。目前我們正在實驗將本方法應用於維護專案，藉由兩位非原始開發者應用本方法來提昇一個在 2005 年 7 月所釋出的既有系統，以驗證本方法之有效性與實用性。初步結果顯示本方法可同時適用於新專案開發與既有系統維護。

未來研究方向包含將本方法推廣到不同的專案類型以及探討例外處理自動化測試工具的開發。

致謝

本論文由國科會計劃 NSC98-2221-E-027-052 所補助，特此致謝。

參考文獻

[1] A. Haase, “Java Idioms: Exception Handling,” in Proceedings of the EuroPLoP 2003 conference, Irsee, Germany, July 2003.

[2] B. Meyer, Object-Oriented Software Construction, 2nd, Prentice-Hall, 1997.

[3] C. M. Rubira, R. de Lemos, G. R. M. Ferreira, and F. Castor Filho, “Exception Handling in the Development of Dependable Component-Based Systems,” Softw. Pract. Exper., John Wiley & Sons, 2005, pp. 195-236.

[4] C.-T. Chen, Y. C. Cheng, C.-Y. Hsieh, and I-L. Wu, “Exception Handling Refactorings: Directed by Goals and Driven by Bug Fixing,” Journal of Systems and Software, volume 82, issue 2, 2009, pp. 333-345.

[5] F. C. Filho, P. H. da S. Brito, and C. M. F. Rubira, “Specification of Exception Flow in Software Architectures,” Journal of Systems and Software, 2006, pp. 1397-1418.

[6] F. Cristian, Exception handling. Technical Report RJ5724, IBM Research, 1987.

[7] FindBugs, http://findbugs.sourceforge.net/.

[8] L. Bass, P. C. Clements, and R. Kazman, Software Architecture in Practice, 2nd edition, Addison-Wesley, 2003.

[9] L. Crispin and J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley, 2009.

[10] M. Fowler, Is Design Dead?, http://martinfowler.com/articles/designDead.html, 2004.

[11] Manifesto for Agile Software Development, http://agilemanifesto.org/, accessed 2010.

[12] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley, 2004.

[13] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley, 2009.

[14] Robot Framework , http://code.google.com/p/robotframework/.

[15] S. Stelting, Robust Java: Exception Handling, Testing and Debugging, Prentice Hall PTR, 2005.

[16] 謝金雲，SyncFree: 一個使用Java技術開發之開放原始碼個人資料同步軟體, 國科會自由軟體專案研究計劃，計畫編號 92-2218-E-027-020.

[17] 余翠瑱, 陳建村, 謝金雲, “擴充 Eclipse 例外快速修復功能已實現利外處理策略,” 第二屆台灣軟體工程研討會, 2006.

[18] 洪哲瑋, 陳建村, 鄭有進, 謝金雲, “例外處理程式壞味道的自動化偵測與重構, ” 第五屆台灣軟體工程研討會, 2009.