行政院國家科學委員會專題研究計畫 成果報告

敏捷式例外處裡：程序，重構，與架構
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 96-2221-E-027-034-
執 行 期 間 ：96 年 08 月 01 日至 97 年 07 月 31 日
執 行 單 位 ：國立臺北科技大學資訊工程系(所)

計 畫 主 持 人：謝金雲
共 同 主 持 人：鄭有進
計畫參與人員：碩士班研究生-兼任助理人員：洪志忠
博士班研究生-兼任助理人員：陳建村

報 告 附 件 ：出席國際會議研究心得報告及發表論文

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 97 年 08 月 12 日

敏捷式例外處裡: 程序, 重構, 與架構 結案報告

計劃編號： 96-2221-E-027 -034

執行期限：民國 96年8月1日 至 民國 97年7月30日

主　持　人：謝金雲　國立台北科技大學資訊工程系副教授

共同主持人：鄭有進　國立台北科技大學資訊工程系教授

計畫參與人員： 陳建村、洪志忠 國立台北科技大學

資訊工程系研究生

摘　要

在此計畫中，我們完成一個支援 Java 語言的階段式例外處理模型，並分析符合該模型不同等級的例外處理標準所涵蓋的例外處理基本操作與成本。在該模型之下，我們可以討論在不同的軟體開發階段，採用不同等級的例外處理所需付出的成本，提供開發人員依據專案的特性加以取捨的依據。該模型可以同時適用於 Java checked 與 unchecked 例外。

關鍵詞：強健度模型、例外處理、Java

Abstract

This research proposes a four-level robustness model for exception handling. We present exception handling primitive operations which can be used to implement the model. The cost to achieve different levels of the robustness model is analyzed. With the proposed approach, developers can choose different exception handling strategies according to project needs. The model is applicable for Java checked and unchecked exceptions.

Keywords：Robustness models, exception handling, Java

一、前言

例外處理是一件困難且不易完成的工作。遵循「把大問題切割成若干較小的問題，再個個擊破」(divide and conquer）的基本原理，我們認為可以藉由分階段完成較小的例外處理設計活動，來達到最終例外處理設計的目的。

在本研究中，我們定義四個例外處理強健度等級，以作為例外處理設計的目標。依據文獻探討所整理的結果，我們將常見的例外處理方法所能達到的強建度等級區分為三個等級,如表 1 中的 G1, G2 與 G3。另外，我們增加了 G0 等級，用以表示一個系統的強健度尚未被評估的狀態。

如表 1 所示，我們以六個元素來描述軟體的強健度等級，分別為： name, program's capability to deliver the requested service, the state of the program after handling the exception, how application's lifetime is affected by the exception, known strategies to achieve the robustness level, 與 also known as。

二、三種不同等級的強健度等級

軟體開發是一連串取捨與決定的過程。對於例外處理而言，若是沒有辦法規範例外處理的等級 (level)，則開發人員將沒有可供取捨判斷的依據。也就是說，我們將無法判別一個軟體系統的例外處理屬於何種等級，作的好不好，是否足夠。因此，在此研究計畫中，我們探討並提出一個包含四個強健度等級的例外處理模型，以作為規劃與判斷軟體元件例外處理等級或能力的依據。

2.1 Goal level G0

我們將 G0 的等級稱為「undefined」，用以標記程式的例外處理能力尚未被分析的狀態。在此等級中，我們知道程式可能遇到例外狀況並且採用某種尚未定義的方式加以處理例外；此時程式的狀態可能是對的，也可能是錯誤的。

表 1: 四種強健度等級定義

Element	Goal level G0	Goal level G1	Goal level G2	Goal level G3
name	undefined	error-reporting	state-recovery	behavior-recovery
service	failing implicitly or explicitly	failing explicitly	failing explicitly	delivered
state	unknown or incorrect	unknown or incorrect	correct	correct
lifetime	terminated or continued	terminated	continued	continued
how-achieved	NA	(1) propagating all unhandled exceptions, and (2) catching and reporting them in the main program	(1) error recovery and (2) cleanup	(1) retry, and/or (2) design diversity, data diversity, and functional diversity
also known as	NA	failing-fast	weakly tolerant, organized panic	strongly tolerant, retry

2.2 Goal level G1

例外處理最怕遇到的問題，就是例外被忽略 (ignoring) 或是被吞掉 (swallow)。因為若是元件隱藏例外，則當例外發生時，系統內部的狀態可能已經不正確，但是從外部看來該系統卻處於正確狀態。要找出此種錯誤非常困難。因此，我們所列出例外處理的第一個等級，就是要求元件不可以隱藏例外，如此一來將有助於早期發現程式中的錯誤，以便在程式正式釋出之前，找出大部分的錯誤 [4]。

2.3 Goal level G2

例外發生的原因並非都是由程式的錯誤所造成，有許多例外是因為執行環境 (runtime environment) 所產生。例如，磁碟空間不足、檔案不存在、網路斷線等等。因此，若元件只是單純的回報例外，而沒有將自己的狀態保持在正常的情況，則這些例外將無法由軟體系統或是使用者來執行恢復的動作。因此，我們訂定例外處理的第二個等級為 state-recovery (狀態恢復)，確保例外發生之後，元件還是處於正確的狀態。

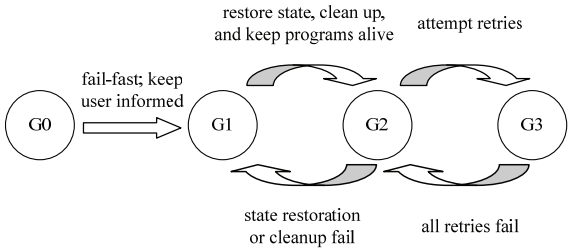

圖 1:強健度等級升級與降級

表 2：達成不同強健度等級之基本操作

Goal	Primitive	Meaning
G1	detecting	Detecting an error state of a component with program logic such as assertions, checks, and acceptance tests.
	signaling	Signaling or throwing an exception to indicate an error or a failure.
	catching	Catching an exception to start the handling process. Because catching checked exceptions and unchecked exceptions require different effort, they are distinguished and denoted by catching_c and catching_u respectively.
	wrapping	Transforming an exception into another type. This can imply a change of exception type.
	declaring	Declaring and/or documenting an exception to be thrown by a component. Similar to catching, declaring checked exceptions and unchecked exceptions are distinguished and denoted by declaring_c and declaring_u respectively. If the exception is designed up-front, it is denoted by $\text{declaring}_{c,\ pre}$ and $\text{declaring}_{u,\ pre}$.
	informing	Informing the users or developers of the occurrence of the exception. Possible actions include popping up an error dialog and logging.
G2	error-handling	Recovering the system from an error state. Common methods include backward recovery, forward recovery, and compensation.
	cleanup	Returning resources to the runtime environment. A cleanup operation is performed regardless of the occurrence of an exception.
	interface-changing	Changing the interface of a component due to declaring_c.
G3	fault-elimination	Removing the exceptional condition and preventing the fault from being activated again.
	alternative	Designing and implementing the alternatives. Common approaches include design diversity and data diversity.
	selection mechanism	Designing a mechanism for selecting suitable alternatives to replace the failed one regarding the raised exception. The selection mechanism may employ static, dynamic, or rule-based strategies.

2.4 Goal level G3

更進一步地，我們訂定例外處理的第三個等級為 behavior-recovery (行為恢復)，也就是當例外發生時，元件本身需要嘗試其他的方法以提供正常的服務。符合此等級的元件又稱為 self-repair、self-healing 或 resilience [1]。

表 3：實作不同強健度等級之成本公式

Goal	Cost Equation
G1	For try blocks not at the top level: *cost (error-reporting) = cost (declaring) +* *cost(detecting)+cost (signaling)* new exceptions 0 unchecked exceptions *cost(catching)+cost (wrapping)+cost (signaling)* checked exceptions For the top-level try block: *cost (error-reporting) = cost (informing)*
G2	*cost (state-recovery) = cost (error-reporting) + cost (error-handling) +cost (cleanup) + cost (interface-changing)*
G3	*cost (behavior-recovery) = cost (state-recovery) + cost (fault-elimination) + cost (alternative) + cost (selection mechanism)*

2.5 階段式例外處理

區分不同強健度等級的設計使得我們的方法可以支援階段式例外處理。軟體的強健度將隨著目標等級（goal levels）提高而增強，且較高目標等級將包含較低目標等級的強健度能力。附帶說明，當一個被設計為 G3 的軟體元件在程式執行時無法達到 G3，該元件可以降即以便滿足 G2（至少保持狀態正確），甚至只達到 G1（將錯誤回報給呼叫者）；如圖 1 所示。

三、強健度等級實作

隨著強健度等級增加，例外處理也變的更加複雜。表 2 列出了我們用來實作不同強健度等級的例外處理基本操作（primitive operations）。

這些例外處理基本操作在執行時也可能會失敗，此時我們需要知道程式該如何面對此狀況。在我們的模式中，程式語言中直接支援屬於 G1 的基本操作，因此這些基本操作原則上不會失敗。例如，丟出例外（exception signaling）和傳遞例外（propagation）很少會失敗。然而，萬一些基本操作失敗，那麼系統將丟出一個 unchecked 例外來表達嚴重錯誤。

G2 和 G3 的基本操作與一般程式碼沒有甚麼不同，因此有較高的機會發生例外。當這些基本操作發生錯誤時，程式應

該要自行丟出一個 unchecked 例外。

四、估算例外處理成本分布

根據表 2 的例外處理基本操作，表 3 列出實作每一個強健度等級的成本公式。

五、應用範例

我們以一個軟體開發範例來說明如何應用表 3 的公式來估算例外處理的成本。

假設我們要開發一個稱之為 JCIS 的應用軟體，以支援 Java 程式的持續整合。三個月之後將有一個軟體展覽，我們希望在該軟體展中展示 JCIS。在市場上相關的軟體競爭非常激烈，因此我們不能承擔若是無法準時在軟體展中推出 JCIS 所失去商機的風險。另一方面，使用者也期待我們的軟體必須要有不錯的品質，因此，長期而言例外處理對 JCIS 而言將是一個不可被忽視的重點。

依據專案的特性，我們的開發策略很簡單：在軟體開發初期，我們將專注在建立 JCIS 的核心功能；我們將此階段稱之為 time-to-market (TTM) iterations。為了趕上軟體展的期限，我們在此階段將不主動處理例外；我們只會回報並紀錄例外的發生 (G1)。在完成主要核心功能之後，我們才會開始思考如何增進 JCIS 的例外處理能力。我們將後續的階段稱之為 robustness iterations。

在此我們考慮一個 JCIS 的使用案例：從 CVS (軟體專案程式庫) 中下載專案以供整合之用。該使用案例對應到的系統結構如圖 2 所示。

使用者發出了一個 checkout 指令，該指令被一個圖型使用者介面的元件 PIC 所接收。PIC 將該指令轉送給 CVSController，其負責管理所有的 CVS 指令。為了完成 checkout 指令，CVSController 產生另一個 CVSCheckout 元件去處理。CVSCheckout 元件的實作方式使用到一個開放原始碼的 JavaCVS 元件。透過 JavaCVS 去連接到 CVS repository 並執行實際下載的動作。現在，我們考慮一個錯誤發生狀況：

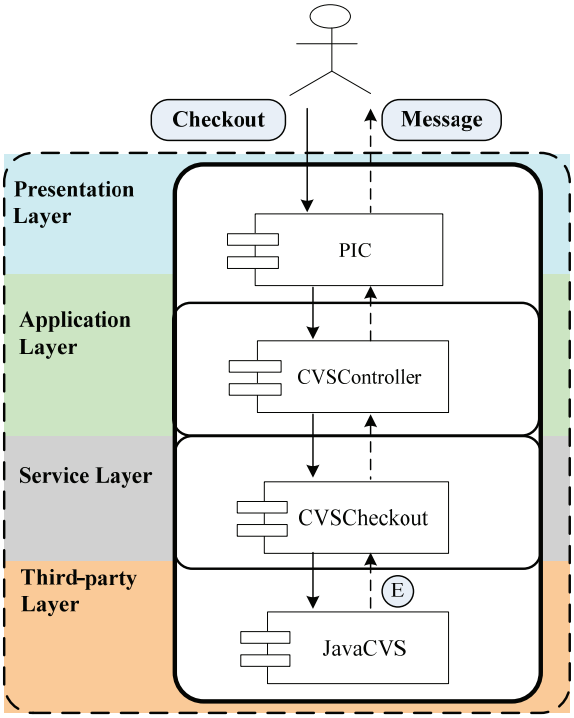

圖 2：JCIS 的階層式架構

網路連線中斷：JavaCVS 元件連線到某個 CVS repository 並且正在下載檔案。網路連線突然中斷導致 JavaCVS 丟出一個 `ResponseException`。

5.1 開發情境

如圖 3(1)所示，在 TTM iterations，CVSCheckout 捕捉由 JavaCVS 所丟出的 checked `ResponseException` 並將其轉換成 unchecked `UnhandledException` 以便於直接回報給最上層的 PIC 元件。我們將一個 checked 例外轉成 `UnhandledException` 的動作相當於在程式中留下一個 exception hole；而該 exception hole 將在下一個階段 (robustness iterations) 中被修復。

當專案進行到 robustness iterations，我們將主動處理例外。首先，我們修復遺留在 CVSCheckout 元件的 exception hole。由於 CVSCheckout 是一個 command 物件 [3]，其缺乏足夠的 context information 以執行自我修復，因此我們將 CVSCheckout 的例外處理能力定義在 G2 使得當例外發

表 4：例外處理成本分佈

Iter.	Primitive operation in CVSCheckout	Primitive operation in CVSController	Cost value	Percentage
1	wrapping, signaling, catching$_c$		3	4.8%
2	declaring$_c$, wrapping, error-handling, cleanup,	catching$_c$, wrapping, signaling	29	46.0%
3		declaring$_c$, wrapping, selection, alternative,	31	49.2%

生時它可以被其他物件繼續使用。因此，我們宣告了一個 checked exception `CVSCheckoutException` 在 CVSCheckout 的介面上。接著，我們參考表 2 中所列的例外處理基本操作來修改 CVSCheckout 使其具有達到 G2 的能力。其結果如圖 3(2) 所示，此時原本的 exception hole 被往上推到 CVSController 元件中。

經過上述的演進，CVSController 現在擁有一個 exception hole。因為 CVSController 是一個應用程式階層的元件，且具備足夠的 context information 來執行重試 (retrying)，因此我們將它的例外處理等級由 G1 提升到 G3。同樣地，我們參考表 2 中所列的例外處理基本操作來修改 CVSController 使其具有達到 G3 的能力，其結果如圖 3(3) 所示。我們在 CVSController 介面宣告了另一個 checked exception `RepositoryException` 用以表示 CVSController 無法成功達到 G3 的錯誤。由於此時 CVSController 已經處於 G3，PIC 自動變成了 G2（因為 PIC 的例外處理能力僅與 CVSController 相關，也就是 PIC 的例外處理能力主要由 CVSController 來決定)。因此，當使用者看到 PIC 所顯示的錯誤對話盒時，使用者將可安全的重試 (重新執行) 原本失敗的功能 (因為此時雖然有錯誤發生，系統還是處於正確的狀態)。

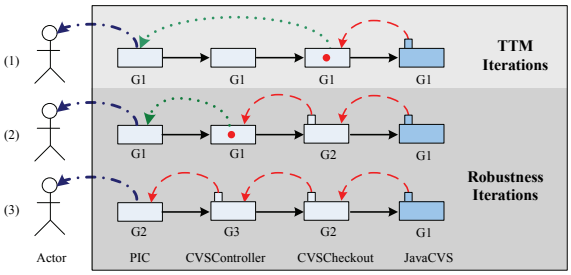

圖 3：例外處理演進

5.2 例外處理成本評估

表 4 列出了本範例中例外處理成本分布的情況。其例外處理基本操作的成本預估值如表 5 所示。在此我們要觀察的是例外處理成本在不同開發階段的分布，而不是基本操作的值是如何估算。換句話說，在給定一組基本操作預估值之後，我們便可評估在軟體開發過程中，例外處理的成本將會發生在哪一個階段。以此範例而言，在 TTM iterations 時，我們只花了總例外處理成本的 4.8%，以便滿足快速上市的需求。由於我們知道每一個軟體元件的強健度等級，因此雖然在 TTM iterations 我們沒有妥善地處理例外，但是後續的補強工作並不會因此而造成混亂。

六、結論與未來展望

本研究提出一個包含四個等級的例外處理強健度模型，並分析達成不同強健度等級所需的例外處理基本操作。我們以實際範例說明如何採用本研究所提出的方法來實作例外處理設計以及評估例外處理的

成本分布。

未來研究方向將探討以自動化的方式來估算軟體系統中，特定執行路徑的強健度等級，以及達到特定強健度等級所需的成本，以做為軟體開發成本估算之參考。

表 5：例外處理基本操作預估值

wrapping	1
signaling	1
catching_c	1
catching_u	5
declaring_u	5
declaring_c	5
$\text{declaring}_{c,pre}$	10
cleanup	5
selection	5
error-handling	15
alternative	20

參考文獻

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, Jan-Mar 2004.

[2] C.-T. Chen, Y. C. Cheng, C.-Y. Hsieh, and I-L. Wu, "Exception Handling Refactorings: Directed by Goals and Driven by Bug Fixing," (to appear) *Journal of Systems and Software*, 2008. (DOI: 10.1016/j.jss.2008.06.035.).

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] J. Shore, “Fail Fast,” *IEEE Software*, vol. 21, no. 5, 2004.

計畫成果自評

在此計畫中我們完成例外處理強健度等級模型、例外處理基本操作、以及其成本模式評估，成功達到原本計畫書中所規劃的目標。本計畫的成果目前已發表一篇 SCI 國際期刊論文 [2]。

行政院國家科學委員會補助國內專家學者出席國際學術會議報告

96 年 12 月 8 日

附件三

報告人姓名	鄭有進	服務機構及職稱	國立台北科技大學 資訊工程系副教授
會議時間 會議地點	民國 96 年 12 月 3 日 至 民國 96 年 12 月 7 日 日本名古屋	本會核定補助文號	NSC 96-2221-E-027-034
會議名稱	（中文）第 14 屆亞太軟體工程研討會 （英文）Fourteenth Asia-Pacific Software Engineering Conference (APSEC)		
發表論文題目	（中文） 1. ezContract: 以標記函式庫及 Bytecode 處理的方式遂行 Java 之依合約設計 2. Pseudo Software: 一個迭代式需求發展與驗證的概念 （英文） 1. ezContract: Using Marker Library and Bytecode Instrumentation to Support Design by Contract in Java 2. Pseudo Software: a New Concept for Iterative Requirement Development and Validation		

告內容應包括下列各項：

一、參加會議經過

APSEC 2007 為亞太地區最重要之軟體工程學術研討會，本次會議為第 14 屆，主辦國為日本(2006、2005、2004 年分別由印度、台灣、韓國主辦)。APSEC 論文集由 EI、IEL 收錄。本年度錄取論文 66 篇，錄取率約為 33%，。本人於 11 月 30 日出發，在 3 天私人行程後，於 12 月 3 日到達名古屋，參加 workshop、tutorial，並於主要會議會中 12 月 5 日與 7 日發表論文共兩篇。

二、與會心得

亞太軟體工程會議已成為一個全球性的會議。本次論文與參加者來源國家除台灣、中國、韓國、日本、泰國、新加坡、越南、澳洲、美國等亞太地區國家以外，更有英國、愛爾蘭、瑞典、挪威、德國、義大利、西班牙等。本次研討會主軸為。本人出版論文兩篇 ezContract(由本人宣讀)與 Pseudo Software(由東海大學周忠信教授宣讀)順利完成發表，並獲得與會人士許多寶貴意見，對後續研究極具參考價值。此二篇論文均被排在會議大廳發表，其中，Pseudo Software 一場到場聆聽人數超過 100 人。此外，有日籍教授 Katsuhiko Gondow 對 ezContract 的工具極感興趣，本人已應允於明年工具完成後將軟體提供其使用。

此次會議亦可看出軟體工程界對於量測(measurement)的重視程度正大幅提升。由於專案的報價、品質、生產力等績效指標均須以量測與估計的方式獲得，因此，各種量測的方法、模型等持續被提出。由於台灣近年來積極推行軟體工程之流程改善(CMMI)，在漸有公司進展至 ML 4(量化管理)之際，量測與分析更顯得重要。此領域極值得產、官、學界共同重視，以有效了解並改善台灣在軟體生產上的績效。

會期間，12 月 4 日由 M. Jackson 教授主講之 tutorial “Problem Frames”為軟體需求工程中，相當受重視的一個方法，使用者日漸廣泛。惜 Jackson 教授在 tutorial 中因身體不適而告終止，僅完成 1/3 之內容，至為可惜。

三、考察參觀活動(無是項活動者省略)

無。

四、建議

主辦單位十分的用心，租用環境與設施極優的 Midland Square(5F)，地點位於名古屋火車站正對面，極為方便，此外負責學校 Nanzan University 人員在 M. Aoyama 教授領導下全程積極參與，為會議成功打下基礎。建議國內辦理 International conference 時，亦能優先選擇優質場地。

五、攜回資料名稱及內容

第 14 屆亞太軟體工程研討會會議論文集一份(紙本)

六、其他

無