

Ch02: 錯與除錯

- 1.1 軟體危機
- 1.2 軟體品質
- 1.3 品質模型
- 1.4 品質控制與確保
- 1.5 案例

```
graph TD
  A[開始] --> B{決定};
```



弘德皇帝與程式設計師

弘德皇帝與程式工程師



2.1 臭蟲與錯誤

2.1.1 臭蟲

1947 年 9 月 9 日下午 3 點 45 分，**Grace Murray Hopper** 在她的筆記本上記下了史上第一個電腦 bug ——在 Harvard Mark II 電腦裡找到的一隻飛蛾，她把飛蛾貼在日記本上，並寫道「First actual case of bug being found」。這個發現奠定了 Bug 這個詞在電腦世界的地位，變成無數苦逼程式設計師的噩夢。從那以後，bug 這個詞在電腦世界表示電腦程式中的錯誤或者疏漏，它們會使程式計算出莫名其妙的結果，甚至引起程式的崩潰。

Grace Murray Hopper 是 Harvard Mark I 上第一個專職程式設計師，創造了現代第一個編譯器 A-0 系統，以及第一個高級商用電腦程式語言「COBOL」，被譽為「COBOL 之母」，被稱為「不可思議的葛麗絲（Amazing Grace）」。

這是流傳最廣的關於電腦 bug 的故事，可是歷史的真相是，bug 這個詞早在發明家湯瑪斯·愛迪生的年代就被廣泛用於指機器的故障，這在愛迪生本人的 1870 年左右的筆記裡面也能看得到。而電氣電子工程師學會 IEEE 也將 bug 這一詞的引入歸功於愛迪生哪些臭名昭彰的軟體 bug 名留青史？

事實上，口語上常用 Bug, 但光是一個 bug 很難表達所有的狀況。

犯錯、錯誤、失效

當工程師 *犯錯*(error)，把不對的程式邏輯寫到程式碼中，程式碼內就有了 *錯誤* (fault)，當有 fault 的程式編譯成為執行碼開始執行後，就可能導致系統的 *失效*(failure)。失效不一定要是系統當機，如果和我們預期的結果不同，我們就會說他失效。有錯誤不一定會產生失效，只要程式剛好不會執行到錯誤的地方，就不會產生失效。

犯錯 (make errors) 導致錯誤 (fault)，錯誤導致失效 (failure)。

Fault 錯誤一詞過於通用，在資訊領域比較常用的口頭說法是 Bug。但習慣上我們會誤解「Bug」所指的工程師的編碼錯誤，而且也過於淡化錯誤所帶來的影響。許多的錯誤來自於規格或設計錯誤，因此比較精準的說法是「缺陷」(defect) - 表示系統的錯誤可能來自規格的錯誤、設計的錯誤或是編碼的錯誤。

只有工程師會犯錯嗎？其實不然，我們所採用的編譯器 compiler，框架framework 或工具也可能出錯，都會造成錯誤的產生。錯誤一定會造成失效嗎？未必，有的程式已經運行了十幾年也沒有發生失效，可以說是運氣好，也可以說是運氣差- 因為越晚發現的錯誤越難修復。

Error

依據 IEEE Standard Glossary of Software Engineering Terminology" (standard 610.12, 1990):

Error An error is a mistake, misconception, or misunderstanding on the part of a software developer.

這裡的 developer 包含 軟體工程師、程式撰寫工程師、系統分析師與測試工程師。例如開發者可能會誤解設計符號、程式撰寫工程師可能打錯一個變數名稱等。

Fault

Faults (Defects)

A fault (defect) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.

錯誤或缺失有時被稱為「bug」。缺失一詞更能凸顯需求規格或是設計文件的錯誤。缺失可以在審查的階段被發現並加以矯正。

所有的失效都因為有 fault 嗎？也不盡然，例如網路的異常造成系統的失效就不能說是系統的錯誤。也就是說例外（exception）會造成系統的失效。但是工程師應該要能夠降低例外所造成的衝擊，例如有訊息的提示，系統狀態的儲存等。

關於布林值最棒的一點是，即使你搞錯了，也只差一點點（一個位元，雙關語）。

The best thing about a boolean is even if you are wrong, you are only off by a bit.

(Anonymous)

2.1.2 規格導致的缺陷

並不是所有的錯誤都是因為「寫程式」造成的缺陷，許多的情況規格的問題。針對一個計算機，以下的計算結果或現象是不是錯誤？

- $5/2 = 2$
- $1/3 * 3 = 0.999999$
- 我需要次方的功能，但卻沒有
- 平方根的功能根本不需要，但系統卻有此功能
- 大數字的相乘，例如 $88888888 * 88888888$ ，系統沒有顯示數字。
- $1/0$ 產生系統崩潰
- 顏色的配色好醜


 我前方沒有規格，錯誤在我身後形成

考慮以下三個規格：

- *規格一*：設計一個除法器，使用者可以輸入被除數，除數，並且呈現出結果，小數點下兩位四捨五入。
- *規格二*：設計一個除法器，使用者可以輸入被除數，除數，並且呈現出結果，小數點下兩位四捨五入。使用者不得輸入被除數為 0。（規格沒有說明萬一輸入 0 的處理方式）
- *規格三*：設計一個除法器，使用者可以輸入被除數，除數，並且呈現出結果，小數點下兩位四捨五入。使用者不得輸入被除數為 0，若輸入為 0, 清除結果欄位，並出現提示要求使用者重新輸入。

很顯然地三個規格比前兩者較好。

:::SUCCESS

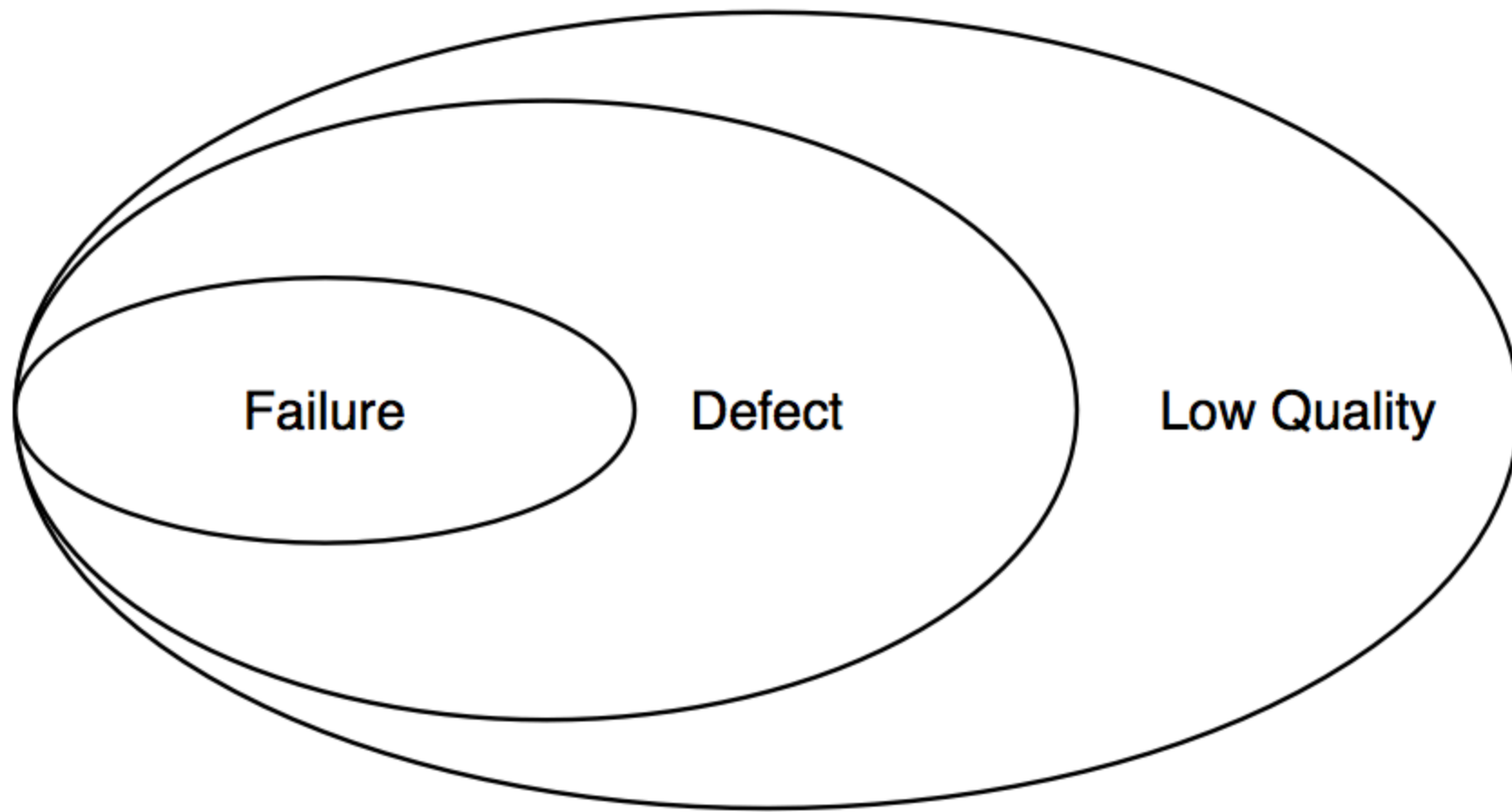
Ex  有一個程式使用者輸入三個數字，並判斷其為何種三角形，請寫出其規格

```
int checkTriangle(int a, int b, int c)
```

Hint: 非三角、三角、正三角、等腰三角、等腰直角三角、直角。注意等腰直角三角的判斷條件。

如果沒有規格，寫的再好都會被強辭奪理成 bug。同理，明明是錯誤也會被強辭奪理成不是錯誤。

然而，如果一味的要求規格書要寫完整所有的規格，在現實環境也是很困難的。在沒有爭議，一看便能夠確認是錯誤的情況下，我們是可以直接認定為錯誤的。當有模糊不確定的時候，我們再讓規格書來做確認。



fig_失效，缺陷與低品質

上圖說明失效、缺陷和低品質的關係。沒有失效並不代表系統沒有缺陷。沒有缺陷的系統也只是表示符合規格所定義的，規格書很難寫清楚的東西包含 非功能性需求，例如效

2.1.3 編碼錯誤

以下列出一些常見的編碼錯誤。

- 算數錯誤
 - 分母為 0 的錯誤;
 - 運算超載;
 - 精準度錯誤。
- 邏輯錯誤
 - 無窮迴圈;
 - 差一個 Off-by-one bug (OBOB) ◦ %, counting one too many or too few when looping ◦

```
for (i = 0; i <= a.length; i++)  
{  
    /* Body of the loop */  
}
```

資源相關臭蟲

- 使用 Null pointer;

```
People p[3] = new People[3];  
p[1].sleep();
```

- 使用沒有初始值的變數;
- 資源漏出 Resource leaks, 有限的空間因為不斷被分配使用（卻沒有釋放），導致資源耗盡所產生的錯誤。See (https://en.wikipedia.org/wiki/Resource_leak).
- 緩衝區溢位 Buffer overflow, 程式企圖儲存超過它容量的資料。
- 超量的遞迴，即便是邏輯正確，也會造成堆疊易位溢位。
- 釋放後使用 Use-after-free error, 使用一個 pointer, 它原先所指的空間已經釋放了。

FIG: Race condition in multiple threads

Thread 1	Thread 2	Integer value
		0

多執行緒程式臭蟲

- 死結, 工作 A 必須等到 B 完成才能繼續，但同時 B 也要等到 A 完成才能繼續。
- 競爭, 程式執行的順序與開發者想像的不同，資料的共用沒有有效控制所產生的錯誤

介面臭蟲

- 不正確的 API 使用，例如參數個數的錯誤、順序錯誤、資料型態錯誤等。

2.1.4 錯誤的預防

錯誤的處理通常有三種方法

- *預防*。預防勝於治療，透過教育訓練或是謹慎的軟體工程工法來避免錯誤的發生。
- *偵測，減緩，容錯與移除*。當錯誤已經進入你的系統，你要在發佈（release）之前把它偵測出來並且解決。
- *處理*。當你的系統已經發佈，錯誤已經發生，你就要考慮發生錯誤的處理。注意一旦含有錯誤的系統發佈給使用者，其處理的代價就會大很多。

預防與偵測有以下的作法：

- *寫程式的模式*。例如採用 Defensive programming 的方式避免邏輯性的程式錯誤。Bug 通常會造成內部資料的不一致。我們寫程式的時候隨時的檢查是否有不一致的情況，發現不一致時可以終止程式的執行或是提供訊息給程式員。



2.2 除錯

2.2.1 觀念

不要想著不會錯，要想著會出錯：

- 避免使用試探法，「碰碰運氣修改程式看看能不解決問題」，通常沒有辦法找出問題的根源；
- 錯誤會群聚，如果這個地方有錯，相同的模組、相同的人、相同的時間所開發的程式也該檢查一下；
- 不要只改徵兆，要找出原因，修改原因，動動腦分析思考與錯誤徵兆有關的資訊；
- 小心改一個錯誤又帶來其他錯誤；
- 修改也是一種 programming 活動，所有 programming 的原則都適用；
- 修改前一定要做版本的 check in，中間的過程如果具備意義也要做 check in。
- 除錯的工具只是輔助的手段。工具不能取代思考；
- 避開死胡同。很久都解不開時，可以休息一下，跳脫原思考的疆界。問人家都會有一些新的靈感。

測試工程師要擁有偵探的特性：

- 好奇心。對於小問題都好奇的想去解決；
- 耐心。能夠反覆的搜查，驗證自己的想法；
- 觀察力。能夠細心的看到、聽到一些現象；
- 學習力。不斷的學習累積專業知識或生活知識，才能做出判斷；
- 推理能力。依據現有蒐集的資訊做出推理，再不斷的驗證；
- 經驗。這是少不了的，很多類似的情節可以馬上判斷出犯人的手法。
- 第六感。就是覺得哪裡怪怪的。



2.2.2 方法

- 從錯誤的外部形式入手，找出錯誤的地方。要閱讀錯誤訊息，不要因為是英文就跳過不看。
- 研究程式，找出原因；
- 修改程式，排除錯誤；
- 測試。要注意是否引入新的錯誤；
- 重複上述動作直到沒有錯誤。

研究程式，找出原因的方法：

- 強行除錯：watch variable; memory dump; print variable；
- 回朔法：沿著錯誤的症狀，一路辦隨著程式的流程回朔到可能出錯的源頭；
- 追蹤法：透過追蹤程式的執行，觀察其路徑與預計的是否相同，找出問題的源頭；
- 注意異常：發現有異常即立即處理，避免異常擴大到無法檢測的地步。（使用斷言）
- 歸納演繹法：根據現象列出所有的假設，在逐步的實驗、分析以否決不對的假設，逐步的縮小可能的假設，找出真正問題的根源。

2.2.3 邏輯推演與除錯

除錯的過程中需要從現象找出原因，這需要應用邏輯推演來協助。

$$(p \Rightarrow q) \not\Rightarrow (q \Rightarrow p)$$

例如我們觀察到 cache 有開的時候，資料就會產生錯誤; 如今資料發生錯誤了，就推斷是開了 cache，這是錯誤的推理，會導致在除錯時一些誤導。也要注意，「不開 cache 就不會產生錯誤」也是一個錯誤的推理：

$$(p \Rightarrow q) \not\Rightarrow (\neg p \Rightarrow \neg q)$$

當多個原因造成一個現象：

$$p_1 \vee p_2 \vee p_3 \Rightarrow q$$

表示 p_1, p_2, p_3 只要其中一個為真，就會造成 q 現象。這也表示當 q 現象沒有發生， p_1, p_2, p_3 都不可能。亦即：

$$\neg q \Rightarrow (\neg p_1 \wedge \neg p_2 \wedge \neg p_3)$$

:::success

Ex 🏈 已知格式錯誤且住址長度超過50 以上，會產生Err101 的錯誤。以下的推斷是否正確：

目前沒有產生Err101 錯誤，而且我們確定格式有錯，因此可以斷定字串長度小於 50

說明：目前沒有產生Err101 錯誤，表示格式沒有錯誤 或 長度沒有50。因為目前我們確定格式有錯，我們因此可以否定長度超過 50，亦即字串長度小於 50。

:::SUCCESS

Ex 🏀 有時候我們需要從一些線索（或現象）來推斷因果。例如，由一些使用者的回報現象如表，系統失效可能原因是什麼？

OS	Memory	TSR	Version	Result
Win 8	2G	K	2.5	Normal
Win 8	1G	no	2.5	Normal
Win 8	2G	K	2.4	Normal
Win 8	1G	K	2.5	Normal
Win 10	2G	K	2.5	Abnormal
Win 10	1G	K	2.4	Abnormal
Win 10	2G	K	2.4	Abnormal
Win 10	1G	no	2.5	Normal
Win 10	2G	K	2.3	Abnormal
Win 10	1G	no	2.5	Normal

我們初步斷定只要有安裝卡巴防毒軟體（K）並且運行在 Window 10 作業系統時，系統就會產生異常（列印會當掉）。異常的情況與軟體版本、記憶體沒有關係，我們可以獲得此式：

$$installK \wedge onWindow10 \Rightarrow Abnormal$$

當然，資料量越多，得到的推論會越正確。但注意若有人反應：系統異常了，而且安裝了 window 10 的版本，是否可以斷定該使用者安裝 K？依據邏輯規則是不一定的，但我們卻常常犯了此錯誤。

寫小程式時完全不感覺寫程式是困難的，也覺得「軟體品質」和自己沒有關係，但程式一旦大到一定程度，就會難以控制。

AI 輔助除錯

2.3 除錯工具

除錯工具- Java 實作

2.4 防禦性編程

開車遇到綠燈你會直接闖過去嗎？多數的時候我們還是會減慢速度，因為我們不知道對方會不會闖紅燈，這樣的開車方式是一種「防禦性」開車方式。寫程式也有同樣的觀念。防禦性編程（Defensive programming）是一種撰寫程式的態度與方法。

本節介紹兩種技巧：

- 斷言 assertion
- 例外處理 exception handling

一個好的工程師是那種過單行道馬路都要左顧右盼的人

A good programmer is someone who always looks both ways before crossing a one-way street.

-- by [Doug Linder](#)

[斷言](#)- Java 實作

[例外處理](#)- Java 實作

[日誌](#)- Java 實作

2.5.2 Lala 語錄

為大虎保險公司開發的保險系統已經有一年，許多模組陸續上線測試，也頻頻出現問題，大虎的專案經理「福哥」似乎已經按捺不住，找雄太反應：

「雄太，系統有問題沒有關係」福哥說：「但是要處理，處理的態度也很重要」

「難道，Lala 處理的不好？」雄太說。

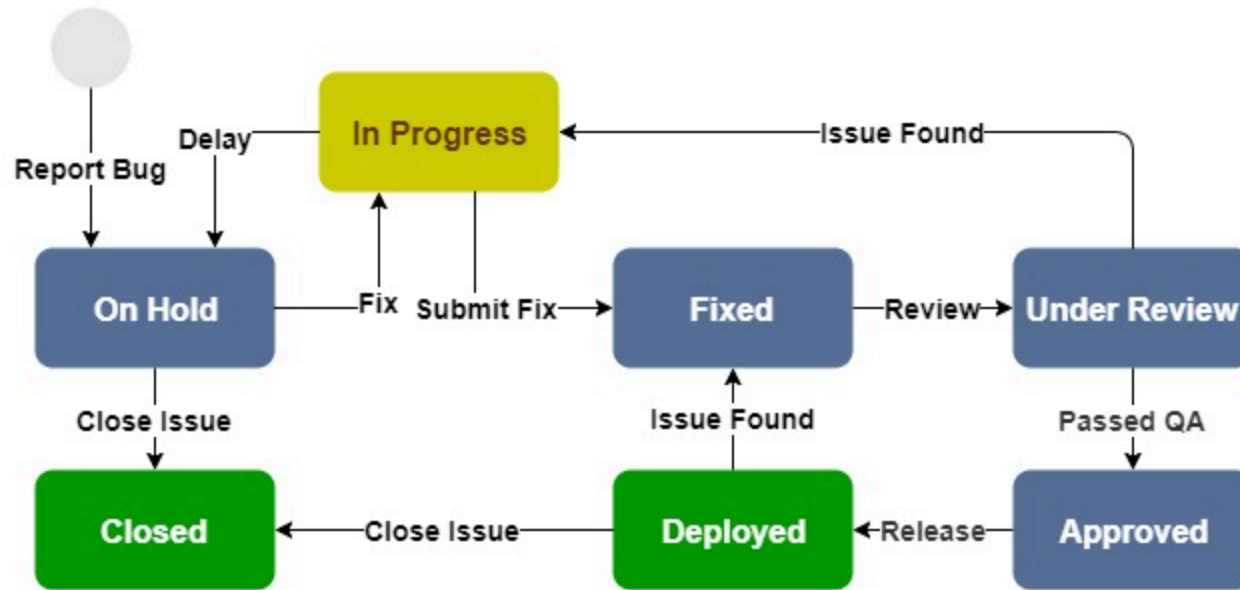
「她的回應感覺起來是不認錯，好像在推責任，難道我們還會污賴她嗎？最糟糕的是，今天講了這個錯誤，下星期測試又冒出來，真懷疑她到底有沒記下來？有沒有認真在除錯？你看看，這是員工們私下整理 lala 常常說的話」：

- 之前不會這樣啊！
- 昨天明明會動的啊！（皺眉頭）
- 這一定是你們機器的問題。
- 你剛剛到底做了什麼！（生氣狀）

它是在你資料有問題（鐵口直斷）

2.5.3 議題管理系統

? 市面上有哪些常用的 BTS/IMS ? 有哪些功能 ? IMS 的流程為何 ?



發生缺陷並不嚴重，嚴重是去處理錯誤，沒有一套缺陷處理的機制。

缺陷追蹤系統，有時候又叫議題追蹤系統（issue tracking system）是一套管理缺陷或議題的資訊系統。他可以協助客戶通報缺陷，程式員知道要修正哪些缺陷，並且讓品質確