測試內嵌系統 — 您有沒有它的 GuT 呢？

Vincent Encontre

Rational Software 白皮書

TP 317, 11/01

Rational®
the software development company

目錄

簡介

本白皮書提供測試內嵌系統的一般簡介，然後討論內嵌系統的問題如何影響測試流程和技術，以及 Rational Test RealTime 如何提供這些問題的解決方案。

一般概念的定義

我們先從決定一般概念的一些定義開始。

什麼是測試？測試是規範式流程，包括檢查應用程式（包括其元件）的行爲、效能和健全性是否符合預期準則。其中一個主要標準（不過通常是隱含的）是應用程式要儘量無瑕疵。因此，預期的行爲、效能和健全性應該是可以正式說明及測量的。除錯照字面上解釋是指移除問題（錯誤），被視爲測試流程的一部分而已。

到底什麼是「內嵌系統」？要提出精確的定義並不容易，而且爭議性很高，因此，這裡舉了一些例子。內嵌系統存在於每一個「智慧型」裝置中，滲入我們的日常生活中，例如袋子裡的行動電話和它背後的所有無線基礎架構、桌面上的 Palm Pilot、傳送電子郵件的網際網路路由器、大螢幕家庭電影院、航空控制站，及它所監督的誤點飛機等等。現在軟體佔這些裝置價值的 90%。

圖 1 世界在內嵌軟體上運行

大部分（即使不是全部）內嵌系統都是「即時的」。「即時」和「內嵌」這兩個詞彙經常交替使用。在即時系統中，運算的正確性不只依賴其邏輯正確性，還依賴產生結果的時間。如果系統的計時限制不符，就會發生系統失效。對於已識別爲安全關鍵的某些系統而言，是不容許失效的。因此，***測試計時限制與內嵌系統的測試功能行爲一樣重要***。

即使內嵌系統的測試流程類似用於許多其他種類之應用程式的流程，內嵌系統的一些重要問題仍然會影響它：

- 應用程式開發和執行平台之間的分隔
- 各式各樣的執行平台和交互開發環境
- 廣泛的部署架構
- 不同實作參照範例的共存性

- 對執行平台的嚴格資源和計時限制
- 缺乏明確設計模型
- 新興的品質和認證標準

本白皮書結尾處有這些問題更詳細的討論。

這些問題大大影響測試和測量內嵌系統的能力。這也說明爲何測試這類系統如此困難，而成爲目前開發作法其中最脆弱的一環。這也難怪，根據最近一項研究指出（請參閱 [R1]），內嵌系統開發專案有 50% 以上都落後排程好幾個月，而只有 44% 的設計符合特性和效能預期的 20%—即使總開發資源當中有 50% 以上是投入於測試中。

在本白皮書其餘章節，我們將：

- 瀏覽一般測試反覆，從中可以衍生最小的一組測試技術
- 建立此反覆的實例來處理測試複雜系統，如內嵌系統中所發現的，我們並依據這些考量，來增加理想測試技術的功能
- 檢查使內嵌系統如此難以開發和測試的原因
- 評定這些問題如何新增至用來測試它們的技術所實現的特性清單

Rational Test RealTime 將作爲產品的範例，來實作此理想技術的大部分。

一般測試反覆

準備測試中粒度 (Granule Under Test, GuT)

在任何測試中，第一個必要步驟是要識別要測試的粒度。「粒度 (granule)」一字是用來避免使用其他像元件、單元或系統之類的字眼，這些字的定義比較不那麼通用。例如，在 UML 中，元件是指應用程式的一個組件，它有自己的控制項執行緒（亦即，作業或 UNIX 流程）。我也想使用「粒度」一字作爲「精度 (granularity)」的字根，它在可以測試的各種元素上有很好的翻譯 —從單一程式碼行到大型分散式系統。

識別要測試的粒度，然後將它轉換成可測試的粒度或測試中粒度 (granule under test, GuT)。此步驟包括隔離粒度與其環境，使它不靠 *Stub* 與 *配接器*的幫助。Stub 是程式碼組件，它模擬粒度和其餘應用程式之間的雙向存取。

然後，建置測試驅動程式。它以適當的輸入刺激 GuT，然後測量輸出資訊，並將它與預期的回應做比較。配接器是用來容許測試驅動程式刺激 GuT。刺激和測量是遵循 GuT 各道關卡的特定路徑：我們稱之爲控制與觀察點 (Points of Control and Observation, PCO)，這個詞彙直接來自電信工業。PCO 可位於 GuT 邊界或內部。

C 函數粒度的 PCO 範例如下：

- 在粒度內的觀察點：函數中特定程式碼行的涵蓋面
- 在粒度邊界的觀察點：函數傳回的參數值
- 在粒度內的控制點：區域變數的變更
- 在粒度邊界的在控制點：含實際參數的函數呼叫

Stub 和測試驅動程式可以是應用程式的其他組件（如果有的話），不一定是爲了測試 C 函數或 C++ 類別而開發。GuT 可透過應用程式的另一個組件存取或刺激，再由該組件扮演 Stub 或測試驅動程式的角色。 Stub 和測試驅動程式構成 GuT 的*測試控制工具*環境。

說明測試案例

說明測試案例取決於對下列的理解：

- 適當的 PCO — 這視要執行的測試種類而定，例如功能、結構、負荷量等等
- 及如何利用它們 — 要傳送什麼資訊，預期可以透過此資訊接收到什麼，還有它的順序

測試的類型以及傳送或預期的資訊，取決於針對 GuT 設定的需求。在安全性關鍵系統的案例中，正式和精確需求是開發流程的重要一環。正式需求是測試的重要動機來源，它們不一定能夠明確識別可探查系統重要缺點的測試。使用正式需求時，以及當缺乏特定需求時針對較不重要的應用程式，測試者必須考慮執行適當的測試集（亦稱爲「測試計劃」），繪製一些需求來測試 GuT。這些需求必須轉換成正式測試案例來利用 PCO 的優點。所謂「正式」，簡而言之，就是 *可執行的*。

通常，需求本身並不正式，不會自動轉換成正式測試案例。此轉換流程通常產生錯誤，使測試案例無法正確反映需求。自從引進 UML 之後，規格語言變成更正式了，現在可以表達以正式需求爲基礎的測試案例，來避開轉換陷阱。Rational QualityArchitect 和 Rational Test RealTime 的一部分提供使用這類模型測試技術的理想範例。

很可惜，並非所有需求都是使用 UML 來描述的，尤其是在內嵌系統中：測試案例最常見的正式說明技術就是使用程式語言，例如 C 或 C++。雖然 C 和 C++ 已眾所周知（所以學習曲線縮短），但它們在考量測試案例的需求上成效不佳，例如 PCO 定義或預期的 GuT 行爲。因此，它們無法讓您寫出綜合性的測試案例。此問題已由特定高階測試語言的設計來處理，這些語言非常適合特定測試領域，例如資料密集型或交易型測試。Rational Test RealTime 提議混合原生的 3GL 和專用的高階 Scripting 語言，呈現兩者最好的一面—縮短學習曲線和提高撰寫測試案例的效率。

撰寫測試案例另一個非常重要而有生產力的方式就是使用階段作業錄製器。當 GuT 受到刺激（不論是以手動方式或經由它的未來環境）時，特定觀察點記錄 GuT 的輸入和輸出資訊，然後進一步自動轉換成適當的測試案例，並於稍後重播。這種階段作業錄製器的範例，可在 Rational Rose RealTime 中找到，其模型執行導致產生反映追蹤執行的 UML 序列圖，然後作爲 Rational QualityArchitect RealTime 的測試案例模型。

每一個測試案例必須使 GuT 進入特定啓動狀態，來容許測試執行。測試案例的這個部分即所謂的前言。在實際測試結束時，不論結果是什麼，測試案例 Script 必須使 GuT 進入最後穩定狀態，容許下一個測試案例執行。測試案例的這個部分叫作後序。

部署和執行測試

測試案例會轉換及整合成測試驅動程式和 Stub 的資訊部分（相對於可操作的部分）。請特別注意，Stub 說明是完整測試案例的一部分。測試案例是在測試控制工具執行期間執行。

觀察測試結果

測試執行所產生的結果透過觀察點來監督。

在粒度邊界上，典型的觀察點包括：

- *函數傳回的參數*或已接收的訊息
- *廣域變數值*
- *資訊排序和計時*

在粒度內，典型的觀察點包括：

- *程式碼涵蓋面*，提供關於涵蓋 GuT 的哪一個軟體部分之詳細資料

- *控制曲線*，遵循已執行的各種邏輯分支
- *資訊流程*，將 GuT 不同組件的時間之相關資訊的交換視覺化。一般而言，這種流程是以 Rational Test RealTime 中的 UML 序列圖來表示。
- *資源使用*，顯示非功能性資訊，例如在 GuT 的不同組件所花的時間、記憶體儲存區管理或事件處理效能。

可以針對單一測試案例收集和/或一組測試案例聚集所有這些觀察。

決定下一步

當收集並合成所有測試資料之後，有兩個結果：一或多個測試案例失敗，或所有測試都通過。

測試案例可能因爲一些理由而失敗：

- *不符合需求*（包括隱含的零當機需求） — 您必須回到實作，甚至回到原始設計來解決 GuT 的這個問題。
- *測試案例錯誤* — 其發生率遠高於您的想像。測試就像軟體一樣，第一次使用時不一定能夠如預期運作。修改測試案例來解決此問題。
- *測試案例無法執行* — 同樣地，就像軟體一樣，一切看似正常，但您就是無法部署或啓動或連接測試控制工具到 GuT。

如果通過所有測試，您可以考慮採取下列行動：

- *重新評估測試* — 如果您還在測試流程初期，可對測試的價值和目標提出懷疑。測試應該要找出問題，尤其是在開發工作上，如果您一無所獲，那麼，…
- *增加測試案例數* — 這樣應該可以增加 GuT 的可靠性。您可以反對，因爲可靠性是需求的一部分，話是沒錯。然而，透過整組測試案例，可靠性的層次通常直接與 GuT 的涵蓋面層次相關。
- *程式碼涵蓋面是最廣泛使用的涵蓋面類型* — 如 Rational Test RealTime 中的實作。以程式碼涵蓋面爲基礎的測試有助於定義其他測試案例，將涵蓋面層次提高到與需求一致的層次。此測試部分通常稱爲結構測試—測試案例是以 GuT 的內容爲基礎，而不是直接取決於它的需求。
- *聚集粒度來增加測試範圍* — 如下面的段落所述，您可以將此一般流程套用到大部分系統。

測試何時停止？

這是軟體工作者的常有的疑問，但它並不是本白皮書要解決的問題。然而，有一種探索方式是考量測試中系統的安全性。系統可以視爲安全關鍵嗎？對於非安全關鍵系統，可以依據一些主觀準則（例如市場時效性、預算和「夠好了」）來停止測試。然而，對於不能失效的安全關鍵系統而言，不能以這樣的準則來做出停止測試的決定。在接近本白皮書結束之處，有一節的標題是*新興的品質和認證標準*，裡面有對於處理此問題的一些建議。

一般測試技術的需求

從上述的一般測試反覆來看，我們可以推斷出測試工具必須實施的最小特性集。它們必須：

- 幫助定義及隔離 GuT
- 提供測試案例表示法（不論是 3GL 或視覺化或高階 Scripting），支援 PCO、傳送至及預期來自 GuT 的資訊、和前言/後序的定義。
- 幫助從需求或測試計劃中正確衍生測試案例
- 提供替代方式，使用階段作業錄製器來實作測試案例

- 支援測試案例部署和執行
- 報告觀察
- 評定成功或失敗

Rational Test RealTime 支援這些特性而且超越這些需求，來處理內嵌系統領域中所發現的複雜系統的測試。

測試複雜系統的六個漸進式步驟

複雜系統的一般架構和實作

內嵌系統是複雜的系統，可由非常不同的架構組成，從迷你 8 位元微控制器，到包含多重處理器平台的大型分散式系統。然而，這些系統有三分之二是執行在即時作業系統 (RTOS) 上，不論是買來現用的或是內部製造的，並且實作執行緒概念，這些執行緒已延伸到 RTOS 作業或流程。（執行緒是獨立於控制流程外的粒度）。在 UML 中，此概念稱爲元件，而節點是指執行 RTOS 管理的一組作業的獨立處理單元。節點之間的任何通訊通常是使用傳訊通訊協定（例如 TCP/IP）來執行。

大部分內嵌系統開發人員使用 C、C++、Ada 或 Java 作爲程式語言（2002 年有 70% 使用 C，60% 使用 C++，20% 使用 Java，5% 使用 Ada，請參閱 [R1] 的附註）。內嵌系統中有多個語言並不足爲奇，尤其是 C 和 C++ 在一起，或 C 和 Java 在一起。C 應該更有效率而且更接近平台的詳細資料，而由於物件導向概念，Java 或 C++ 應該更有生產力。然而，要注意，內嵌系統的程式設計師並不是物件狂熱者！

在內嵌系統的環境定義中，粒度可以是下列其中之一。此清單是按漸進的複雜性排序。

- C 函數或 Ada 程序
- C++ 或 Java 類別
- C 或 Ada 模組（的集合）
- C++ 或 Java 類別叢集
- RTOS 作業
- 節點
- 完整系統

對於最小內嵌系統，完整系統只包含一組 C 模組，不整合任何 RTOS 相關程式碼。對於最大系統（分散式系統），網路通訊協定又增加了複雜性。

下一節顯示這個一般架構如何影響不同的測試步驟。

測試的六個漸進式步驟

視粒度類型而定，並根據業界的*一般*用法（本節稍後會討論），需要六個測試步驟來檢查應用程式的行爲、效能和健全性是否符合預期的準則。這些步驟如下：

- 軟體單元測試
- 軟體整合測試
- 軟體驗證測試
- 主機測試
- 系統整合測試
- 系統驗證測試

軟體單元測試

GuT 是隔離的 C 函數或 C++ 類別。視 GuT 的用途而定，測試案例可能包含：

- *資料密集測試* — 對函數參數值套用大範圍的資料變式，或
- *情境型測試* — 運用不同的 C++ 方法呼叫序列來執行需求中所有可能的使用案例

觀察點是指傳回的值參數、物件特性評量和程式碼涵蓋面。白箱測試法是用來測試單元，表示測試者必須熟悉 GuT 的內容。單元測試由開發人員負責。

由於要向下追蹤複雜內嵌系統的普通錯誤並不容易，因此，必須在單元測試層次全力尋找並移除它們。

軟體整合測試

GuT 現在是一組函數或一個類別叢集。驗證介面是整合測試的精華。同一種控制點也適用於單元測試（資料密集 main 函數呼叫或方法呼叫序列），而觀察點的焦點則放在使用資訊流程圖的低階粒度之間的互動。

等到 GuT 開始有意義時—也就是當端對端測試狀況可套用至 GuT 時—就可以執行效能測試，它們應該可以提供架構有效性的適當指示。至於功能測試，則越快越好。所以，每一個即將到來的步驟將包括效能測試。白箱測試法也是此步驟期間使用的方法。軟體整合測試由開發人員負責。

軟體驗證測試

GuT 就是元件內所有的使用者程式碼。這可視爲軟體整合的最後步驟。使用案例現在較貼近一般使用者狀況，而遠離實作詳細資料。觀察點包括資源使用評估，因爲 GuT 是整體系統的重要組件。最後，同樣地，我們把這個步驟當作白箱測試法。軟體驗證測試也是由開發人員負責。

主機測試

現在 GuT 是完整系統元件；亦即，在軟體驗證測試期間測試的使用者程式碼，加上所有與 RTOS 和平台相關的組件：作業機制、通訊、中斷等等。控制點通訊協定不再是對函數或方法的呼叫，而比較像是使用 RTOS 訊息佇列傳送或接收的訊息。

傳訊參照範例中的對稱安排暗示測試驅動程式和 Stub 之間的區別在此階段是不相關的。我們稱之爲*虛擬測試器*，因爲每一個都可以取代及作爲另一個系統元件，相對於 GuT。「模擬器」和「測試器」是「虛擬測試器」的同義字。虛擬測試器技術應該具有多用途，能夠適應龐大的 RTOS 和網路通訊協定。從現在開始，測試腳本通常會使 GuT 進入所要的起始狀態，然後產生依序的訊息範例序列，並比較訊息內容與預期的訊息及比較接收日期與計時限制來驗證接收到的訊息。測試腳本會分散及部署在不同虛擬測試器上。系統資源受到監督，以評定系統維持內嵌系統執行的能力。從這個步驟開始，灰箱測試法是較好的測試方法。唯一需要的是要瞭解 GuT 的介面。根據組織而定，主機測試可能由開發人員或由指派的系統整合團隊負責。

系統整合測試

GuT 是從單一節點內的一組元件開始，最後包含所有系統節點，最後發展成一組分散式節點。PCO 混合了與 RTOS 和網路相關的通訊協定，例如 RTOS 事件和網路訊息。除了元件之外，虛擬測試器也可以扮演節點的角色。至於軟體整合，其焦點在於驗證不同介面。灰箱測試法是較好的測試方法。系統整合測試由系統整合團隊負責。

系統驗證測試

GuT 最後成爲整體完成內嵌系統。最後步驟的目標如下：

- *符合一般使用者的功能需求*。請注意，一般使用者可能是電信網路中的一個裝置（假設內嵌系統是網際網路路由器），或是一個人（如果系統是消費性裝置）或是兩者（可由一般使用者管理的網際網路路由器）。
- *執行最終非功能性測試，例如負荷量和健全性測試*。虛擬測試器可複製來模擬負荷量，並設計成在系統中造成失效。

- *確保與其他連接設備的交互作業能力*。檢查是否符合適用的互聯標準

詳述討論這些目標已超出本白皮書的範圍。黑箱測試法是較好的方法—測試者專注於頻繁而危險的使用案例。

決定如何排序這些步驟

現在我們有六個漸進式步驟，該如何使用它們呢？以下有許多用來決定的準則：

- 這些步驟是否全都適用於您的系統
- 所有已選取的步驟是否要對系統的全部或只有一部分執行。
- 已選取的步驟應該套用至已選取的部分的次序

以收集此準則爲準礎，決策主要依據您要開發的內嵌系統的種類而定：安全關鍵與否、市場時效性、部署很少或很多等等。未來將撰寫提供準則協助您完成此選取流程的文件。屆時您可以透過 Rational Developer Network 來存取。

處理這些測試步驟的另一個方式是將它們合而爲一！驗證測試可視爲對更大的 GuT 的單元測試。在驗證測試期間也要檢查整合。它取決於您要如何存取 GuT、您可以插入的 PCO 種類、您可以從測試案例中何處及如何挑出 GuT 的特定部分（可能距離很遠）。但我們留待以後再討論…

複雜系統測試技術的其他需求

爲了處理測試複雜內嵌系統的難題，測試技術必須新增下列功能：

- *管理多個類型的控制點* — 若要刺激 GuT，請透過不同語言連結（例如 Ada、C、C++ 或 Java）來使用函數呼叫、方法呼叫和訊息傳遞或遠端程序呼叫 (RPC)，以達成此目的。
- *提供各種觀察點*例如參數和廣域變數檢查；確認、資訊和控制路徑追蹤；及程式碼涵蓋面錄製或資源使用監督。每一個這些觀察點都應該提供預期與實際評量功能。

內嵌系統的問題如何影響測試流程和技術

在此章節中，我們將強調內嵌系統的特定問題，並以 Rational Test RealTime 作爲測試工具來評定它們如何影響用來測試它們的技術。

應用程式開發和執行平台之間的分隔

內嵌系統的多個定義之一：

> *內嵌系統是任何必須設定在平台上的軟體系統，該平台不同於應用程式預定要部署及鎖定目標的平台。*

在開發方面，平台通常代表作業系統，例如 Windows、Solaris 或 HP-UX。請注意，在內嵌系統領域中，與其他更多 IT 系統領域相比，UNIX 和 Linux 使用者的百分比更高（40%，請參閱 [R1] 的附註）。至於目標，平台將包括前述任何裝置。

爲何有此限制？因爲目標平台已最佳化且專爲一般使用者量身訂作（可能是真的人或是另一組裝置），所以開發不需要執行必要元件（例如鍵盤、網路、磁碟等等）。

若要處理這個雙重平台問題，測試工具必須以最透明而有效的方式，提供從開發平台存取該執行平台的權限。事實上，不要讓使用者看到這種存取的複雜性。存取包括測試案例資訊下載、測試執行遠端監督（啓動、同步化、停止），還有測試結果和觀察上傳。在 Rational Test RealTime 中，所有目標平台存取是由目標部署技術控制。

此外，Rational Test RealTime 可使用於 Windows、Solaris、HP 和 Linux，這些是裝置、內嵌系統和基礎架構工業的先進公司所使用的開發平台。

大量而多變化的執行平台和交互開發環境

執行平台的範圍很廣，有以迷你 8 位元微控制器爲基礎的主機板，也有大型分散式網路系統。因爲晶片和系統供應商眾多，所有平台需要不同工具進行應用程式開發。在同一個內嵌系統內使用多個平台，已經越來越普遍。

一般而言，這種環境的開發稱爲「交互開發環境」。各式各樣的執行平台暗示著有相對大量的開發工具可用，例如編譯器、連結器、載入器和除錯器。

它的第一個結果，就是觀察點技術只能以程式碼爲基礎。相對於 Rational PurifyPlus 系列所使用的目的碼插入技術，可供一小組原生編譯器使用的 Rational Test RealTime，使用程式碼強化功能來處理數字因素。此技術的十年經驗，獲得高度效率的程式碼強化功能。

此變化的另一項直接結果，是交互開發工具的供應商傾向於提供整合性開發環境 (IDE)，來隱藏複雜性，使開發人員更加得心應手。任何其他工具與相對應的 IDE 要密切整合，這是一項重要需求。例如，Rational Test RealTime 完美整合到 WindRiver 的 Tornado 或 GreenHills 的 Multi IDE。

由動態電腦晶片工業驅動的另一個性質，就是新的執行平台和相關聯的開發工具之發行非常頻繁。這使得在錄製期間，測試技術必須很有彈性並能夠適應這些新架構。新目標平台的 Rational Test RealTime 目標部署通常不到一週（通常在兩天以內）就可以達成。

對執行平台的嚴格資源和計時限制

根據定義，內嵌系統對應用程式應該執行的資源有更大的限制。這對於可用的 RAM 少於 1 KB 的迷你平台更是如此；開發環境的連線只能使用 JTAG 探針、模擬器或序列連結來建立；或微處理器的速度剛好能夠處理此工作。測試工具面臨一項困難的取捨—將測試資料放在開發平台上，並以犧牲效能作爲代價（這代價通常太大，令人無法接受）來透過連線連結傳送該資料，或者，由目標平台上的測試驅動程式來解譯測試資料。

Rational Test RealTime 使用的技術是要將測試控制工具內嵌到目標系統。其作法是在測試控制工具內編譯先前已轉換成應用程式設計語言（C、C++ 或 Ada）的測試資料，然後連結此測試控制工具物件檔到應用程式的剩餘部分。此建置鏈對使用者而言是透明的，因爲它在 Make 檔內使用 Rational Test RealTime 命令行介面。最佳化產生程式碼、智慧型鏈結對映配置和低記憶體覆蓋區，這些全都是爲了使目標平台上的測試控制工具所需要的資源減至最少而運用的，同時又能提供下列優點：

- 計時精確度已改進，並使用目標位置減少對效能的即時影響。
- 在測試案例執行期間避免在連線功能鏈結上循環任何資料資訊，可使主機目標通訊量減至最少。當測試案例確實結束之後，必要的話，觀察資料會儲存在 RAM，並可透過除錯器或模擬器的協助上傳。

雖然交互開發環境越來越有親和力，目前出貨的內嵌系統有大部分還是很難應付來開發和測試令人頭痛的難題。Rational Test RealTime 的目標是要簡化內嵌系統開發人員棘手的例行程序。

缺乏明確設計的視覺化模型

內嵌系統開發人員喜歡程式碼！很可惜，專科畢業生在視覺化建模方面的訓練不夠，他們比較相信程式碼才是「真材實料」。雖然透過 UML 之類的語言，視覺化建模在將內嵌知識併入設計方面已有長足的進步，但大部分內嵌系統程式設計師仍然喜歡使用一般舊式程式語言來完成他們的大部分工作。Java 並未將太多人帶入設計中！

爲了協助開發人員以他們喜好的方式工作，Rational Test RealTime 將焦點放在協助他們設計以應用程式程式碼爲基礎的測試案例，它提供一些精靈，例如測試範本產生器和 API 包裝函式。確定測試可套用至應用程式，是此程式碼測試建置流程的主要優點。其缺點是不保證測試案例會反映它應該檢查的需求。顯然，更廣泛採用視覺化建模可縮小需求和測試案例之間的差異。 Rational Test RealTime 也爲此技術舖路，它提供 Rational Rose RealTime UML 序列圖給測試案例編譯器。

新興的品質和認證標準

對於特定種類的內嵌系統—安全關鍵系統 —不容許發生失效狀況。我們在核能業、醫療業和航空業中都有發現這些系統。以很久以前設定的零故障爲目標，航空業和政府機關（例如美國聯邦航空署）共同發表這份聲明：*Software*

Considerations in Airborne Systems and Equipment Certification，當作 RTCA 的 DO-178B 標準的參考，如 [R2] 所述。這是全球航空業的安全關鍵軟體開發的主要標準。作爲最具威信的軟體開發標準之一，它也漸漸被製造生命關鍵系統的其他產業局部採用，例如汽車業、醫療業（FDA 剛發表一項接近 DO-178B 的標準）、國防工業或運輸業。

DO-178B 將軟體分類成五個重要性等級，它們與導致或造成系統功能失效的異常軟體行爲相關。最重要的是 A 級設備，其失效將導致整個系統發生災難性失效狀況。 DO-178B 包括非常精確的步驟，以確保 A 級設備夠安全，尤其是在測試領域中。Rational Test RealTime 符合所有等級的所有強制的 DO-178B 測試需求，包括 A 級設備在內。

摘要

在本白皮書中，我們已概述六個用來測試內嵌系統的漸進式步驟的流程。考量此流程（從超小型到超大型系統的整個範圍都列入考量）及內嵌系統的特定性質和限制，我們推斷出一組需求，那是理想技術必須擁有才能解決內嵌系統測試的需求。Rational Test RealTime 是在內嵌系統領域中最新的 Rational 供應項目，它舉例說明此理想技術的大部分內容。

本白皮書已提供測試內嵌系統的簡介，此後以不同討論主題爲焦點的其他文章將遵循之。

專有名詞

本節作者爲 Paul Szymkowiak。

很可惜，軟體工程使用的許多詞彙將有不同的定義。本白皮書使用的詞彙可能是在即時內嵌系統領域中工作的您最熟悉的。然而，軟體測試還使用其他相等和相關的詞彙。我們提供下面的對映表。

本白皮書使用的詞彙	相等/相關詞彙
測試中粒度 (GuT)：基於測試用途而與其環境隔離的系統元素。	*相關的詞彙*：測試項目、目標、測試目標
控制和觀察點：測試中的特定時間點，此時會錄製測試環境的觀察，或做出關於測試控制流程的決策。	最接近的相等詞彙：**驗證點**
後序：爲了使系統進入特別穩定的終止狀態而採取的動作，是執行下一個測試所需要的。	後置條件、重設
前言：爲了使系統進入特別穩定的啓動狀態而採取的動作，是執行測試所需要的。	前置條件、安裝
測試控制工具：安排一或多個測試驅動程式、特定測試的 Stub 和合併的強化功能，以達到執行一系列相關測試的用途。	*相關詞彙*：測試套組或測試驅動程式、測試 Stub
虛擬測試器：(1) GuT 的外部元素，它在測試期間與粒度互動。 (2) 執行測試的實例，通常代表某人的動作。	*相關詞彙*：測試腳本或測試驅動程式、測試 Stub、模擬器、測試器

參照

[R1] *"Critical Issues Confronting Embedded Solutions Vendors", Electronics Market Forecast,* http://www.electronic-forecast.com/, April 2001.

[R2] DO-178B, *Software Considerations in Airborne Systems and Equipment Certification,* Radio Technical Commission for Aeronautics — RTCA, http://www.rtca.org/, January 1992.

關於作者

Vincent Encontre 是 Embedded and RealTime, Automated Testing Business Unit 的主任，該單位總部設在法國 Toulouse 的 Rational 最新工程中心。當 Vincent 沒有旅行、出席會議或答覆數不盡的電子郵件時，他會把 Rational Test RealTime 當作 Rational 產品以及下一代 Rational 產品可重複使用的技術來看待。Vincent 在內嵌建模和測試技術及最佳作法方面有豐富的經驗。在 Rational 和 ATTOL 之前，Vincent 花了 13 年在 Philips，然後在 Verilog，從事設計、行銷和支援軟體工程工具，他相信這些工具有助於更快地建立更好的軟體。

在閒暇時間，Vincent 會與他的三個兒子和其他人踢足球，自由自在地享受法國南部美好的生活（以免太遲）。

Rational
the software development company

兩個總公司：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
電話：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
電話：(781) 676-2400

免付費專線：(800) 728-1212
電子郵件：info@rational.com
網址：www.rational.com
國際辦事處：www.rational.com/worldwide