Journal of Science and Engineering Technology, Vol. 14, No. 2, pp.47~56 (2018)

以優質測試案例管理機制提升軟體維護效率與品質

賴森堂
實踐大學資訊科技與管理學系
10462 台北市中山區大直街 70 號
stlai@mail.usc.edu.tw

摘 要

資訊技術與應用環境的快速演進，促使軟體系統必須具備完美的可維護與可擴充特性，才能有效延續軟體的生命期，提升用戶的市場競爭優勢。不斷提出的維護需求是導致軟體維護成本大幅增加的主要關鍵，為了克服維護與變動需求對軟體生命週期帶來的衝擊，IID （Iterative and incremental Development）製程以具備高度變更與擴增彈性，有效改善軟體開發與維護的變動風險。不過，IID 製程在反覆漸進的開發過程中，不僅耗時且需動用過多資源，不易及時應付功能擴增與需求變更的持續測試與持續整合，造成新版軟體系統的品質無法獲得適時驗證與確認（V&V）。為此，本文以優質測試案例為基礎，提出一套測試案例管理機制（Test Case Management Mechanism; TCMM），以優質測試案例結合完善管制程序，在有限資源下，適時完成軟體系統變動後的相關測試，有效改善維護測試耗用的資源，具體提升軟體維護效率與品質。

關鍵詞：維護需求，測試案例，IID，管理機制，迴歸測試

A High-Quality Test Case Management Mechanism for Improving the Efficiency and Quality of Software Maintenance

SEN-TARNG LAI
Department of Information Technology and Management, Shih Chien University
No.70, Dazhi St., Zhongshan Dist., Taipei City 10462, Taiwan, R.O.C.
stlai@mail.usc.edu.tw

ABSTRACT

Owing to the rapid evolution of information technology and application environments, software system must possess excellent maintainability and extensibility throughout its life cycle. Frequent software maintenance requests become a crucial challenge that result in high software maintenance costs and risk. To overcome high maintenance costs and risk, the IID process provides great flexibility and changeability to reduce frequent software maintenance requests. However, in maintenance operation or agile software development, the IID process is not only time consuming but also requires more human resources. Continuous testing, integration, and delivery with automation and

high-efficiency tests can perform the major tasks of software maintenance. In this paper, on the basis of a high-quality test case, the test case management mechanism（TCMM）is proposed. Combining high-quality test cases with the TCMM can facilitate timely completion of relevant tests of software system changes with limited resources; effectively reduce the resource consumption of continuous testing, integration, and delivery; and improve the efficiency and quality of software maintenance.

***Key Words*:** maintenance request, test case, IID, management mechanism, regression test

一、簡介

軟體開發過程中，受到人員不足且時程緊迫的影響，導致延續軟體生命期的可維護性經常被主管階層所忽視，使得維護成本持續不斷的攀升。1990 年代早期，軟體工程學者 Pressman 就曾經推測出軟體維護成本，在整個軟體預算中所佔的百分比將高達 70-80% [17]。Schach 在他的 Classical and Object-Oriented Software Engineering 書上也提到：整個軟體成本的三分之二（約 67%）被用在軟體維護上 [19]，許多企業與組織甚至投入 80%的時間及人力在軟體維護作業上 [7]。在激烈的市場競爭壓力下，電信公司為了迎合用戶的需求，必須持續推出各種新的服務項目，迫使現行軟體功能必須經常配合陸續推出的優惠或促銷方案進行修改、調整與擴充，才能提升市場競爭優勢。為此，每年必須投入很可觀的人力與資源進行系統的維護作業，以符合市場的需求。由上述內容可知，維護階段在整個軟體生命週期是最花時間且最昂貴的一個階段，更顯示出軟體維護的重要性。為此，如何提昇軟體維護效率以降低維護成本，已成軟體維護作業中值得深入探究的主題。

軟體開發過程中的需求變動，已被 IEEE 認定歸屬於維護階段的工作項 [19]。不過，一般的認定還是以軟體系統交付移轉給使用單位後，軟體系統才正式進入維護階段（Post-delivery Maintenance），進入維護階段後，使用單位一般提出的維護需求分為四種類型：（1）修改軟體系統隱含的錯誤—更正維護（Corrective Maintenance）。（2）擴充、更新或改善軟體系統功能—完善維護（Perfective Maintenance）。（3）將軟體系統安置於新的環境—安置維護（Adaptive Maintenance）。（4）對於可能或即將發生的異常狀況，事先採取防範作業—預防性維護（Preventive Maintenance） [8, 19]。成功的維護作業必須完成下面三項任務：

1. 依據維護需求的目的與時程要求，規劃軟體系統擴增、修訂或調整等維護任務。
2. 對於完成擴增、修訂或調整的軟體系統，必須制定一套完整的測試活動，對維護作業完成的產品文件進行確認與驗證。
3. 依據使用單位規劃的維護需求時程與預算如期完成新版的軟體系統。

每項維護作業在進行過程中，幾乎都必須參考、引用甚至修改開發階段完成的相關文件，其中又以更動作業後的各種測試步驟的測試案例最具影響力，因此，維護測試作業是否具備可管理與優質的測試案例，將成為影響維護作業效率與品質的關鍵因素。

軟體開發不重視維護品質，經常造成階段開發文件不齊全且缺乏可維護性，維護作業必須投入大量人力與時間才能勉強達成使用單位所提出的各項維護需求，使得軟體維護成本持續不斷的大幅升高。其中，維護作業完成前，必須進行多項的測試活動，以確認維護作業滿足變動需求。因此，為了提升維護作業的效率與品質，測試案例應具備優質的文件、配置完善的管理制度，且能結合測試工具達成測試自動化。有效改善軟體系統測試案例（Test Case; TC）的品質與管理制度是降低維護成本的必要條件，本文以優質測試案例為基礎，提出一套測試案例管理機制（TCMM），以優質 TC 結合制度化的 TCMM，以有限資源適時完成軟體系統維護更動後的相關測試，有效改善維護測試耗用的資源，具體提升軟體維護效率與品質。本文共分六個章節，第二節將探討軟體維護需求的類型且從維護需求作業剖析出重要的工作項。第三節從測試案例的重要性說明 TC 的生命週期與關鍵品質。第四節提出一套 TC 品質量測模式與改善流程。第五節將以優質 TC 為基礎，提出一套 TCMM。第六節將對本文主題作個結論，且針對 TCMM 帶來的優勢進行說明。

二、軟體維護作業之探討

軟體維護是延續軟體系統生命期的關鍵作業，軟體作業的工作時程與產品品質更是影響維護計畫成敗的主要因素。

（一）軟體維護需求的類型

軟體系統交付移轉給使用單位後，軟體隨即進入維護階段，為了配合應用需要與環境的變遷，使用單位提出維護需求的三種狀況如下：（1）更正維護（Corrective Maintenance）—修改軟體系統未被發現的錯誤。（2）完善維護（Perfective Maintenance）—擴充、更新或強化軟體系統現有的功能。（3）安置維護（Adaptive Maintenance）—軟體系統安置於新的運作環境 [8, 19]。不過，軟體開發過程中，維護品質並不受到開發人員的重視，各個發展階段的文件不僅不完整，而且內容更缺乏正確性及一致性，使得使用單位所提出各項維護需求，受到非常嚴重的考驗。配合維護需求所進行的軟體維護作業，是軟體生命週期能否有效延續的主要因素，每項維護作業在進行過程中，幾乎都必須參考、引用甚至修改開發階段的多項產品文件，其中維護作業完成前，必須進行多項的測試活動，以確認維護作業的品質。因此，為了提升維護作業的效率與品質，測試活動應配置優質測試案例及完善的管理制度，且能結合測試工具達成測試自動化。

IID 是目前很受重視的開發方法 [13]，物件導向軟體系統經常採用的 UP（Unified Process）開發模式 [12]，以及非常注重開發風險的螺旋開發模式（Spiral Model） [3]都是以 IID 製程進行軟體開發作業。反覆（Iterative）開發是指反覆使用同一系列的操作或步驟進行的開發作業，漸增（Incremental）開發是指每次遞增會在現有的軟體系統上，增加新的功能、模組或單元等。IID 方法於每一次反覆開發作業，且完成可運作的系統版本後，立即測試且評估此版本，及早發現開發製程與產品的問題與缺失，協助適時進行改善措施，降低軟體開發風險。IID 製程很適合用於維護需求頻繁的軟體系統，反覆的持續測試活動 [9, 18] 則是 IID 製程必須克服的挑戰。

（二）維護作業的主要工作項

軟體維護作業是依據使用單位提出的維護需求進行工作與時程的規劃，為了成功的達成軟體維護計畫制定的目標，維護各項工作必須依擬定的時程如期完成且滿足使用單位的品質要求。為此，維護作業必須適時達成三項階段性的關鍵任務（參閱圖 1 所示）：

1. 前置作業階段：深入剖析維護需求的目標且依建構管理（Configuration Management; CM）程序 [14]及交互引用關係表，詳細列出需要修改、擴增或調整各個階段文件及程式碼。
2. 更動作業階段：依維護需求的任務與目標，完成各階段文件及程式碼的新增、修改或更正等作業。
3. 確認與驗收作業階段：已修改或新增完成的階段文件或程式碼必須透過書面審查作業或測試活動逐一確認，以確保維護作業的產品品質。未能通過審查作業或測試活動的相關文件，必須退回修改。最後再對已完成的維護需求項目進行驗收測試作業，確認維護作業應邀請使用單位代表參與，以確保都能滿足使用單位的需求目標。

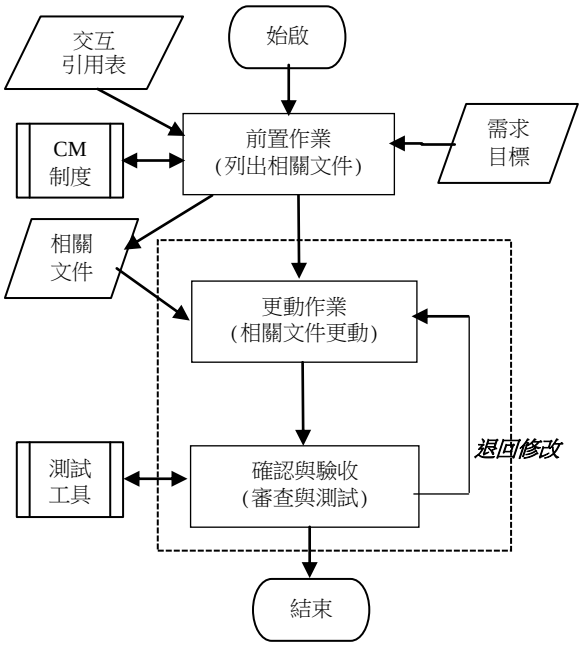

圖1．維護作業三個階段示意圖

維護作業的三個階段在執行過程中，必須參考、引用甚至修改相關階段的產品文件，因此當開發文件出現不完整、不齊全或不一致等品質缺失，勢必大幅衝擊整個維護作業的效率與品質。此外維護作業最為關鍵且耗用最多資源工作的就是各項任務完成前的測試活動 [20]，包括單元、整合、功能、系統及迴歸測試等五個測試步驟：

1. 單元測試：針對新增、修改或變更的單元程式碼進行完整性測試。
2. 整合測試：針對已通過單元測試的單元程式進行整合測試，測試重點著重介面的一致性。
3. 功能測試：針對擴增、修訂或強化後的功能進行功能測試，確認各項功能都能符合維護項目。
4. 系統測試：依據維護需求規格對新版軟體系統進行系統測試，確認新版軟體系統運作已滿足使用單位的預期目標。

5. 迴歸測試：為確保其他未受維護影響的功能，維護後都能正常運作，各項測試活動中還必須考量且實施大規模的迴歸測試 [1, 5]。

這些繁複的測試活動是確保維護作業品質的必要步驟，卻經常需要動用過多人力與時間資源，成為影響維護作業效率與品質的主要關鍵。

三、測試案例的生命週期與關鍵品質

（一）測試案例的生命週期

測試案例主要包括測試資料與預期結果兩部份，測試案例是測試作業的重心更是軟體系統的重要資產，善用測試案例的可再用性是提升測試作業自動化的關鍵。測試驅動開發（Test Driven Development; TDD）製程依據 user story 描繪的內容，設計且擬訂單元測試案例，在程式還未撰寫前就完成測試案例設計與規劃，最大的優勢就是確認需求項目的完整性與明確性 [15]。當測試案例的設計與規劃無法具體且有效達成時，表示 user story 描繪的內容存在不完整或不明確的現象，開發人員必須與利益關係人及使用單位進一步的討論與確認。以下針對透過 user story 描繪的內容，完成測試案例設計與規劃的後續作業探討測試案例的生命週期（如圖 2 所示）：

1. 測試案例設計/重設計：依據新增 user story 描繪的內容，設計（重設計）與規劃單元測試案例，黑箱與白箱測試是兩種常用的測試策略，以下從兩種策略說明測試案例的準備與設計過程 [16]：
   (1) 黑箱測試的測試案例：黑箱測試是依據 user story 描繪的需求，設計與規劃各種狀況的測試資料與預期結果，如需求規定範圍內資料，需求規定範圍外資料，型態錯誤的資料與臨界點前後的資料等，且以執行結果來判斷程式碼是否依需求規格完成撰寫，不需要理會程式碼執行測試所涵蓋的敘述、分支或路徑。
   (2) 白箱測試的測試案例：白箱測試是注重於程式碼測試涵蓋度的蒐集，因此，設計與規劃白箱測試案例時，必須深入剖析程式碼，才能精確設計出可以執行特定路徑的測試資料。白箱測試案例可以協助程式測試執行過程中，利用涵蓋度分析器蒐集程式碼的敘述、分支或路徑等關鍵涵蓋度，用以評估測試案例的完整性，且可以改善程式執行的效能。

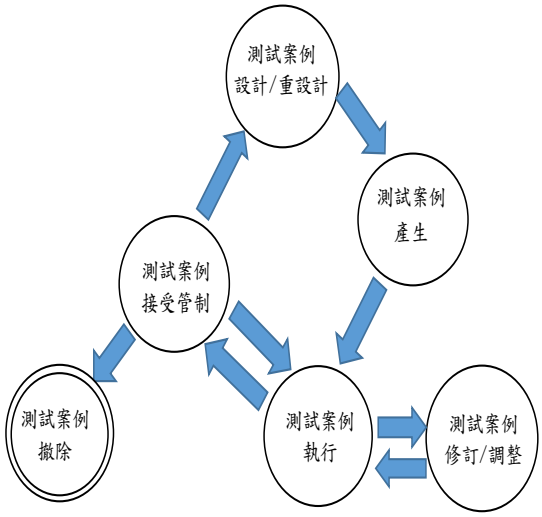

圖 2. 測試案例生命週期狀態圖

2. 測試案例產生：依設計（或重新設計）與規劃的內容產生包含測試資料與預期結果的完整測試案例。測試案例的內容應包含測試案例 Id、設計者、建立日期、類別名稱（class name）、測試對象（如 method name）、測試目的、測試輸入資料描述、輸入資料及預期結果等項目。
3. 測試案例執行：當程式碼撰寫或修改完成後，輸入測試案例資料執行程式測試，且以預期結果判斷程式是否通過測試。
4. 測試案例接受管制：程式通過測試後，測試案例也須依 CM 程序要求，搭配 user story 與程式碼一起繳交且存放到資源庫接受管制，經核准，測試案例才能提領修改。測試案例具備高度可再用性，更可支援維護作業後續的迴歸、整合及系統等測試作業，應妥善管理與使用。
5. 測試案例修訂與調整：使用單位提出維護需求且修改 user story 描繪的內容，則程式碼與測試案例也必須從資源庫提領出且配合 user story 修改內容重新進行規劃與設計測試案例。
6. 測試案例撤除：當軟體系統過於老舊或維護成本過高，使用單位決定放棄繼續維護此系統時，測試案例即功成身退，從資源庫中移除。

（二） 測試案例的品質

為了改善維護作業的資源困境與效率風險，維護測試活動必須具備高度自動化，優質測試案例是提高自動化測試比例的必要條件 [2, 6]，適時將案例優化文件、簡明性、可管

理性及可維護性等四項關鍵品質融入測試案例，使測試案例可以配合維護相關作業進行調整、修改及自動化測試，稱此案例為優質測試案例（High Quality Test Cases; HQTC），以HQTC 可提升維護作業的效率且改善資源困境，HQTC 融入的四項品質特性說明如下：

1. 高品質案例文件（Documents Quality）：每一項測試案例首先必須制訂明確、易於存取的識別代號、規範統一的文件格式且具完整、正確、一致等特性：
   (1) 統一格式：明確識別代號及統一格式的測試案例文件可以提高可再用性。
   (2) 正確性：確保文件描繪的內容是正確的。
   (3) 完整性：依文件格式項目的資料內容都已完整的描繪。
   (4) 一致性：文件描繪的資料項目與所引用規格的項目資料已達成一致。
2. 簡明性（Simplicity）：每筆測試案例都必須擬定具體性、明確性與唯一性的測試目標：
   (1) 具體性：測試案例只規劃特定或單一目標以找出未被發現的缺失或錯誤。
   (2) 明確性：每一測試案例應明確定義預期目標，協助找出未被發現的缺失或錯誤。
   (3) 唯一性：每一測試案例應採取唯一的測試策略與測試方法，避免出現重複或類似的測試案例。
3. 可管理性（Manageability）：測試案例應具備易於存取、版本管制與交互引用等可管理特性：
   (1) 易於存取：每一測試案例都有唯一的識別碼（Identifier），且應具備簡易且快速的存取機制，提高測試案例的存取效率。
   (2) 版本管制：各階段的規格隨著需求變動與環境演進而調整，測試案例也必須配合變動與演進更動，測試案例應建立完善的版本制度以有效管理測試案例的版本變動。
   (3) 交互引用：測試案例是依據各階段的規格進行設計，且搭配已完成的程式碼進行測試，因此，測試案例（項目）與相關文件之間應建立一套完善交互引用關係。
4. 可維護性（Maintainability）：測試案例應具備可變動性、修改彈性與可結合性等可維護特性：
   (1) 可變動性（Changeability）：預期結果與實際執行結果的差異比對，易於更正設計錯誤的測試案例。
   (2) 修改彈性（Flexibility）：需求變動與環境演進促使測試案例也必須配合調整與修改，案例中的測試資料與預期結果對於規格文件應有完整對照關係，以強化測試案例的修改彈性。
   (3) 具結合性：可配合程式修改調整，可依據測試步驟的要求進行擴增與整合。

四、案例品質量測模式與改善流程

個別品質因子必須量化與結合後，才能具體識別 TC 的品質問題與缺失，進而提出品質的改善作業。

（一）測試案例品質量測模式

品質因子量化過程可以分成兩個步驟：首先，依相關文獻與軟體測試實務，詳細規劃且設計一套品質因子量化查核表（Checklist）。接著，採取問卷與訪談方式，針對從事多年軟體測試人員與資深軟體工程師進行意見與數據的蒐集，再將數據彙整成量化值。然而，個別因子或測量值只能測量或評估特定的品質特性。為了有效地監測和評估測試案例的品質，單獨的因子或測量值應該進行適當的組合[4, 10, 11]。兩種度量組合模式分別為線性組合模式（LCM）[10, 11]與非線性組合模式（NLCM）[4, 10]，NLCM 具有比 LCM 更高的精度測量。然而，LCM 具有高的靈活性，比 NLCM 更易於擴增與調整。為此，本文以 LCM 模式用於測試案例品質測量。不同領域的軟體專案開發活動採取不同的品質度量。因此，在應用線性組合模式之前，必須對品質因子的量化值進行正規化。因子正規化的量化值將介於 0 到 1 之間，最佳品質量化值接近 1，最差品質接近 0。測試案例應考慮四項特性的量化品質，包括文件，簡明性，可管理性及可維護性品質。採用 LCM 模式，具相關性的品質因子可以結合成基層度量，然後將歸屬同一品質特性的基層度量可以結合成品質測量值。最後，結合四個關鍵品質測量值且產生測試案例品質指標。五個公式描述如下：

1. 文件品質測量值是由統一的文件格式（Uniform Format）、文件正確性（Correctness）、完整性（Completeness）及一致性（Consistency）等四項品質度量結合而成，結合公式如公式（1）：

DQM: Document Quality Measurement
UFM: Uniform Format Metric W_1*: Weight of UFM*

CrM: Correctness Metric　*W_2: Weight of CrM*
CmM: Completeness Metric　*W_3: Weight of CmM*
CnM: Consistency Metric　*W4: Weight of CnM*

$$DQM = W_1 * UFM + W_2 * CrM + W_3 * CmM + W_4 * CnM$$
$$W_1 + W_2 + W_3 + W_4 = 1 \quad (1)$$

2. 簡明性品質測量值是由案例具體性（Specificity）、明確性（Clarity）及唯一（Uniqueness）性等三項品質度量結合而成， 結合公式如公式（2）:

SQM: Simplicity Quality Measurement
SM: Specificity Metric　*W_1: Weight of SM*
CM: Clarity Metric　*W_2: Weight of CM*
UM: Uniqueness Metric　*W_3: Weight of UM*

$$SQM = W_1 * SM + W_2 * CM + W_3 * UM$$
$$W_1 + W_2 + W_3 = 1 \quad (2)$$

3. 可管理性品質測量值是文件可存取性（Accessibility）、版本管制（Version Control）特性及交互參考（Cross-Reference）特性等三項品質度量結合而成，結合公式如公式（3）:

MgQM: Manageability Quality Measurement
DAM: Doc. Accessibility Metric　*W_1: Weight of UFM*
VCM: Version Control Metric　*W_2: Weight of CrM*
CRM: Cross-Reference Metric　*W_3: Weight of CmM*

$$MgQM = W_1 * DAM + W_2 * VCM + W_3 * CRM$$
$$W_1 + W_2 + W_3 = 1 \quad (3)$$

4. 可維護性品質測量值是由案例可變動性（Changeability）、修改彈性（Flexibility）及測試案例可結合性（Combinability）等三項品質度量結合而成， 結合公式如公式（4）:

MtQM: Maintainability Quality Measurement
DCM: Doc. Changeability Metric W_1: Weight of DCM
RFM: Revision Flexibility Metric W_2: Weight of RFM
TCCM: TC Combinability Metric W_3: Weight of TCCM

$$MtQM = W_1 * DCM + W_2 * RFM + W_3 * TCCM$$
$$W_1 + W_2 + W_3 = 1 \quad (4)$$

5. 最後，測試案例品質指標是由文件品質（Document Quality）、簡明性（Simplicity）、可管理性（Manageability）及可維護性（Maintainability）等四項品質量測值結合而成， 結合公式如公式（5）

TCQI: Test Case Quality Indicator
DQM: Document Quality Measurement
W_{dq}: Weight of DQM
SQM: Simplicity Quality Measurement
W_{sq}: Weight of SQM
MgQM: Manageability Quality Measurement
W_{mg}: Weight of MgQM
MtQM: Maintainability Quality Measurement
W_{mt}: Weight of MtM

$$RTCQI = DQM * W_{dq} + SQM * W_{sq} + MgQM * W_{mg} + MtQM * W_{mt}$$
$$W_{dq} + W_{sq} + W_{mg} + W_{mt} = 1 \quad (5)$$

敏捷開發模式的 IID 製程，持續測試（Continuous Testing; CT）效率與品質受到 TC 品質的衝擊最為明顯。為了改善與提升 TC 品質，TCQM 模式採取多層次的結合步驟，蒐集 TC 的基層品質因子且完成量化，第一層先將相關的基層品質因子結合成 13 項品質度量，第二層再將相關的品質度量結合成 4 項品質量測值，最後，第三層將 4 項品質量測值結合成 TC 品質指標（TC 品質量測值），至於權位值是針對基層度量（Metric）對於各項量測值的影響程度而設定，不同的軟體開發專案著重品質項目與應用領域都有差異，因此權位值應具備可以配合實際狀況調整的彈性。本文稱此量測過程為測試案例品質量測（TCQM）模式，模式架構如圖 3 所示。TC 品質指標是相對的判斷機制，可透過結合步驟與品質因子協助識別 TC 的品質缺失，進而提出 TC 品質改善與矯正措施，具體提高測試案例可再用品質，也間接強化 CT 效率與品質。

（二） 測試案例的品質改善流程

TC 品質量測模式除了可以協助識別 TC 品質缺失，更可以透過各項品質特性的量測公式找出 TC 基層品質因子的不足，進而協助提出 TC 相關項目內容的品質改善機制。TC 品質改善作業適合採取 Rule-based 的判斷作業（參閱圖 4）：以 TC 品質門檻（threshold）搭配法則式判斷，先找出 TC 存在的品質問題與缺失，再以品質量測模式的結合公式推導出基層品質因子的缺陷，具體識別 TC 品質缺失的原因，再提出有效的改善作業，協助矯正與改善 TC 品質，且將通過品質確認的 TC 交付版本管制系統（Version Control System; VCS）進行管制作業。

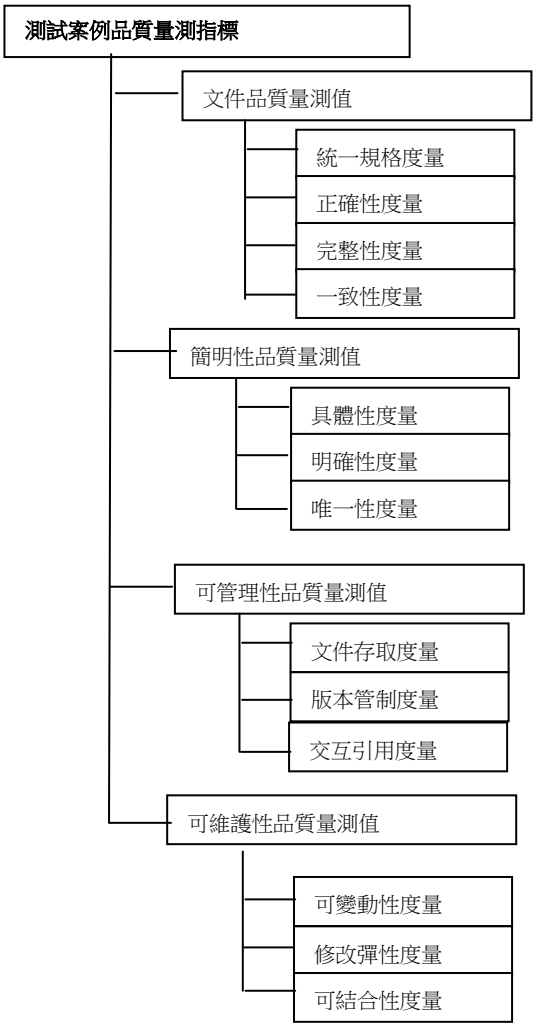

圖 3. TCQM 模式架構圖

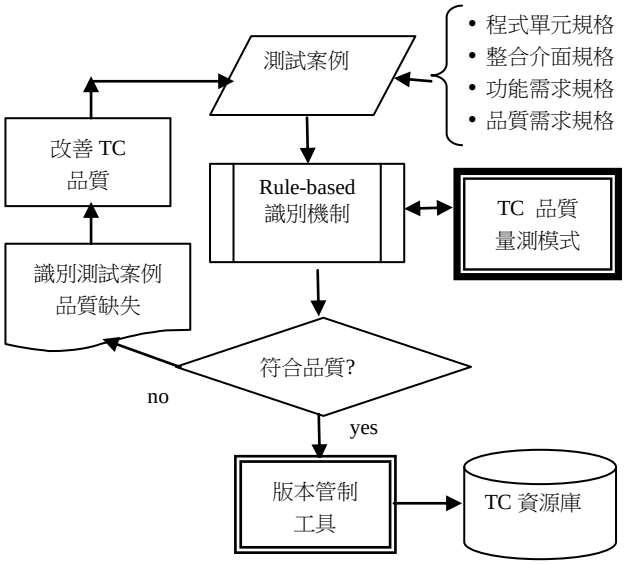

圖 4. TC 品質確認流程

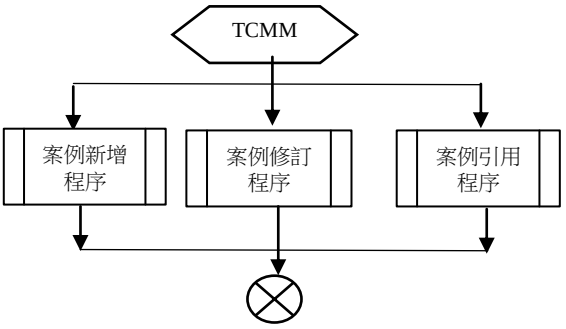

圖 5. TCMM 架構圖

五、測試案例管理機制

優質 TC 具備可管理性與可再用性是改善持續測試效率的關鍵因素。

（一）TC 管理機制的作業流程

測試案例的生命周期狀態是隨軟體系統開發與維護之作業流程進行變動調整，因此，測試案例管理機制必須考量案例的新增、修訂及引用等三項作業流程（參閱圖 5）：

1. 案例新增程序（參閱圖 6）有四個步驟：

步驟 1：配置易於存取的測試案例識別代號，識別代號是由測試層級加上測試項目文件代號，測試層級分為單元測試、整合測試、功能測試、系統測試及驗收測試等。

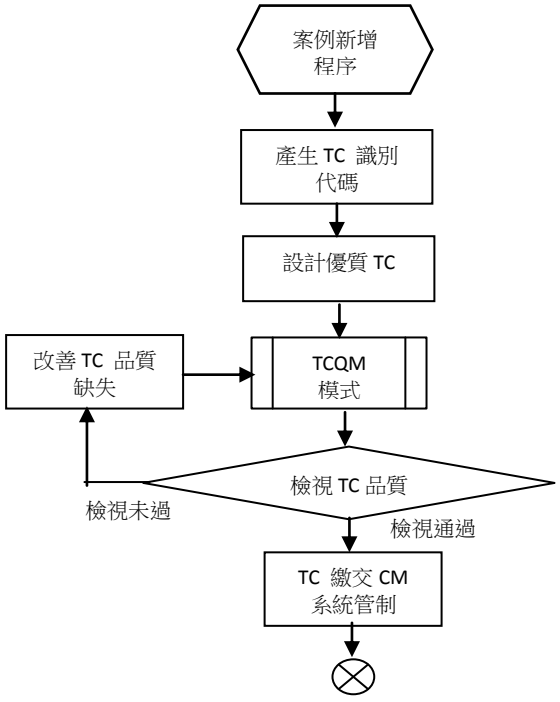

圖 6. TC 新增程序流程圖

步驟 2：建立測試案例與相關文件交互引用關係

- 擴增的需求項目勢必對應產生新的設計文件與程式碼。
- 依架構設計產生整合與功能等測試案例。
- 依細部設計產生各單元 TC。

步驟 3：檢視且確認優質測試案例的品質特性

- 新增的測試案例必須融入文件品質、簡明性、可管理性及可維護性等特質。
- 針對未能符合品質要求的 TC，應找出品質缺失且進行矯正，再次檢視與確認。

步驟 4：以識別代號配合 CM 制度存入資源庫。

2. 案例修訂程序（參閱圖 7）有四個步驟

步驟 1：持續測試作業可以依測試案例識別代號取得測試案例檔，以提升測試效率。

步驟 2：依測試步驟與工作任務修改測試案例，且仍須保持案例應有的品質特性。

步驟 3：檢視且確認變更或調整後優質 TC 的品質特性（同案例新增程序步驟 3）

步驟 4：依版本管制作業記錄修訂細節且配合 CM 制度存入資源庫。

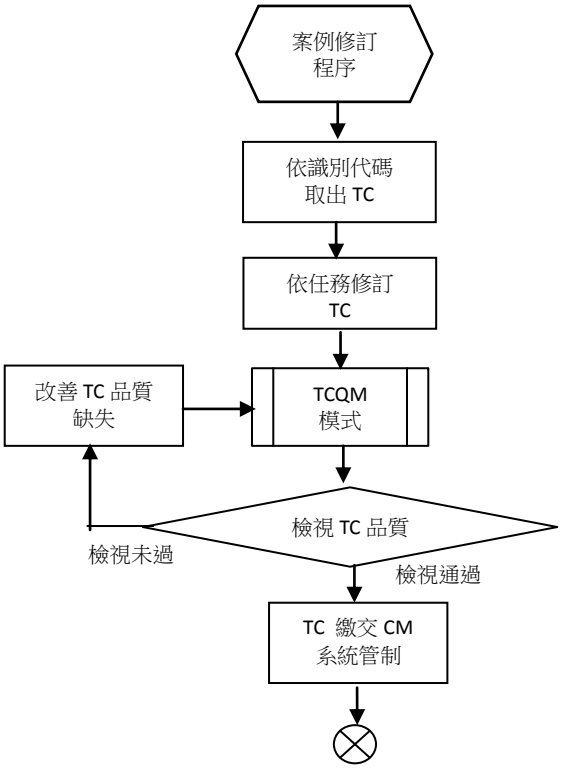

圖 7. 測試案例修訂程序流程圖

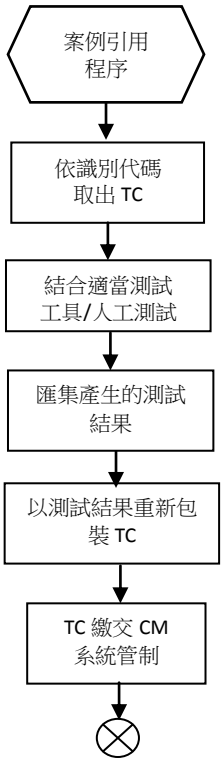

圖 8. 測試案例引用程序流程圖

3. 案例引用程序（參閱圖 8）有四個步驟

步驟 1：持續測試作業可以依測試案例識別代號取得測試案例檔，以提升測試效率。

步驟 2：依測試步驟與任務判斷可以結合的測試工具（如 Junit, 整合、系統等測試工具），再依測試腳本提供測試資料與預期結果，配合測試工具進行自動化測試。

步驟 3：若無適合的測試工具，則採取人工測試作業。首次人工測試必須完整記錄測試資料與測試結果，以協助後續迴歸測試的自動化。

步驟 4：將產生的測試結果重新包裝測試案例，可提供後續維護測試自動化。

（二）TCMM 的效益評估

維護作業必須依使用單位要求如期完成，才能達成使用單位提出維護需求的最高效益。不過，為了確保維護作業品質能夠滿足維護需求規格，多項繁複的測試活動必須逐一確實完成。測試活動經常需要動用過多人力與時間資源，成為影響維護作業效率與品質的主要關鍵。HQTC 配合完善的 TCMM，可大幅提昇維護作業的測試效率與自動化能力，且確保測試活動品質，以下從四項特性說明 TCMM 的優勢：

1. 案例快速存取：優質測試案例具備明確的識別代號與交互引用關係，測試案例可以配合開發/維護作業快速取得測試案例且以版本管制系統進行存放。
2. 優質案例文件：測試案例融入完整且齊全的品質特性，可以快速配合新增、修改及引用等程序有效管理測試案例。
3. 提升自動化測試：測試案例可依據測試腳本（Test Script）存放測試資料與預期結果，再配合測試工具的實作提供測試資料、比對測試結果，可具體達成不需人力介入的自動化測試。特別是測試最頻繁的單元測試及迴歸測試。
4. 易於維護修改：測試案例具備完整優質文件、明確性、可管理性及可維護性等品質特性，對於測試案例的擴增、修改及強化等更動工作可以大幅提升效率。對於案例本身的調整與擴充，或將基層案例組構成高階案例都可以快速且有效的完成。

測試案例不重視品質特性又缺乏完善的管理方法，勢必會影響測試活動的效率。傳統檔案系統管理方式與版本管制方式是兩種常使用的測試案例管理方法，此兩種方法缺乏優質 TC 與 TCMM，只採取版本管制、建構管理與部份測試工具，無法具體且有效整合，不易達成高效率與自動化的測試。以案例識別與存取、案例修改與調整、案例擴充與組合及案例自動化測試等四個項目，對 TCMM 及兩種常用的管理方法進行評比（參閱表 1）：

1. 案例識別與存取：TCMM 結合優質測試案例，可以利用明確的識別代號、交互引用關係及版本管制等特質快速完成案例識別與存放。其他管理方法的識別與存取效率有限。
2. 案例修改與調整：優質測試案例融入完整且齊全的品質特性，TCMM 可以配合新增、修改及調整等程序有效管理測試案例。其他管理方法並未考量測試案例的品質特性。
3. 案例擴充與組合：TCMM 搭配的優質測試案例融入簡明性與可維護性等品質，對於本身的調整與擴充，或將基層案例組成高階案例都可以快速且有效的完成。其他管理方法並未考量測試案例的相關品質特性，案例擴充與組合的能力有限。
4. 案例自動化測試：優質測試案例可依據測試腳本（Test Script）存放測試資料與預期結果，再配合測試工具的實作提供測試資料、比對測試結果，可具體達成不需人力介入的自動化測試。特別是測試最頻繁的單元測試及迴歸測試。其他方法的測試案例並未考量測試腳本（Test Script）存放，很難達成案例自動化測試。

表1．測試案例管理方法效益評估表

管理方法 評比項目	TCMM	傳統方式	版本管制方式
案例識別與存取	高	中	中
案例修改與調整	高	低	中
案例擴充與組合	高	低	中
案例自動化測試	高	低	低

六、結論

資訊技術與應用環境的快速演進，促使軟體資訊系統必須具備完善的可維護性，才能有效延續系統的生命期，不斷提出的變動需求與維護需求是導致軟體維護成本大幅增加的主要原因。為了克服維護需求帶來軟體維護風險，軟體維護製程必須具備高度變更與擴增彈性，大幅提升自動化測試，才能克服軟體維護需求的效率與資源困境。頻繁的維護作業，不僅耗時且需動用過多資源，在有限的時間與人力資源下，不易及時應付功能擴增與需求變更的持續測試，造成軟體系統的版本品質無法獲得適時確認。為此，本文以高品質測試案例為基礎，提出一套測試案例管理機制（Test Case Management Mechanism; TCMM），以測試案例再用性結合完善的測試工具，以有限的時程及資源完成軟體系統變更後的相關測試，有效改善維護測試耗用的資源，具體提升軟體維護效率與品質。TCMM 在軟體維護作業可以具體提升以下的效率:

1. 融入優質文件品質：可以協助維護作業快速存取、新增及修訂各個層級的測試案例。
2. 版本管制與交互引用：利用交互引用表可以快速取得受影響的測試案例，且利用版本管制紀錄完整修改內容，以提供隨時復原及後續的檢視與審查。
3. 增加自動化測試比率：測試案例依據測試腳本（test script）存放測試資料與預期結果，再配合測試工具的實作提供測試資料，有效增加自動化測試比率，具體提升軟體維護效率與品質。

誌謝

本研究接受科技部 105 學年度「專題研究」計畫（MOST 105-2221-E-158-002）補助。

參考文獻

1. Biswas, S., R. Mall, M. Satpathy, and S. Sukumaran (2011). Regression test selection techniques: A survey. *Informatica*, 35(3).
2. Bobera, D. (2008). Project risk mitigation by test cases. *Management*, 3(2), 025-028.
3. Boehm, B. W. (2000). *Spiral development: Experience, principles and refinements, report special report cmu*. SEI-2000-SR-008, Carnegie Mellon Software Engineering Institute.
4. Boehm, B. W. (1981) *Software engineering economics*, Prentice-Hall, New Jersey.
5. Briand, L. (2011). An introduction to regression testing. *Simula Research Laboratory*, 1-39.
6. Cem Kaner, J. D., (2003) What is a good test case? Retrieved August 20, 2016, from http://www.kaner.com/pdfs/GoodTest.pdf.
7. Christa, S., V. Madhusudhan, V. Suma, and J. J. Rao (2017). Software maintenance: from the perspective of effort and cost Requirement. In: *Proceedings of the International Conference on Data Engineering and Communication Technology*, Springer, Singapore. 756-768.
8. Christa, S., and V. Suma (2012) Software maintenance: An overview. *CSI Communications*, 26.
9. Duvall, P. M., S. Matyas, and A. Glover (2007) Continuous integration: Improving software quality and reducing risk. Pearson Education.
10. Fenton, N. E. (1991) *Software metrics - A rigorous approach*, Chapman & Hall.
11. Galin, D. (2004) *Software quality assurance - From theory to implementation*, Pearson Education Limited, England.
12. Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.
13. Larman, C. and V. R. Basili (2003) Iterative and incremental development: A brief history, *IEEE Computer*, 36(6), 47-56.
14. Leon, A. (2015) *Software configuration management handbook*, Third Edition, Artech House
15. Madeyski, L. and Ł. Szała, (2007) The impact of test-driven development on software development productivity - an empirical study. In *European Conference on Software Process Improvement*, Springer, Berlin, Heidelberg, 200-211.
16. Myers, G. J., C. Sandler and T. Badgett (2011) *The art of software testing*. John Wiley & Sons.
17. Pressman, Roger S. (2005) *Software engineering: a practitioner's approach*. Palgrave Macmillan.
18. Saff, D. and M. D. Ernst, (2003) Reducing wasted development time via continuous testing. In *Software Reliability Engineering, ISSRE 14th International Symposium on* IEEE, 281-292.
19. Schach, S. R. (2008) *Object-oriented software engineering*, McGraw-Hill.
20. Yusop, O. M., and S. Ibrahim (2011) Software maintenance testing approaches to support test case changes–A review. *International Conference on Digital Information and Communication Technology and Its Applications*, Springer, Berlin, Heidelberg, 33-42.

收件：107.03.22　修正：107.05.22　接受：107.07.03