Cloud OS 之持續整合
Continuous Integration on Cloud OS

賴建億
Chien-Yi Lai

中文摘要

Cloud OS是工研院研發的雲端作業系統，其中模組眾多且模組之間交互作用複雜，為了有效的對系統進行整合測試，維護軟體品質，在測試方法與測試案例以及整個軟體開發流程上做了許多改進，讓Cloud OS的開發流程達到持續整合(Continuous Integration)，進而改善整個軟體品質，並減少整合測試所需的時間與人力。

Abstract

Cloud OS is a cloud operating system developed by ITRI to provide IaaS service. During developing, we need a process to ensure software quality. We work on testing automation and improve on software development process. As a result, we apply continuous integration on Cloud OS development process so the time and cost on integration test is greatly reduced.

關鍵詞(Key Words)

持續整合 (Continuous Integration；CI)
測試自動化 (Testing Automation)
軟體生命週期 (Software Develop Life Cycle；SDLC)
雲端作業系統 (Cloud OS)

1．前言

軟體開發方法的演進一直都有許多理論，從早期的Waterfall、CMMI，到後來的Agile還有Continuous Integration[1]。但軟體專案的延遲與bug也一直沒有好的解決方法。在研發Cloud OS[8]的過程中我們也遇到相同的問題。為了解決這些問題，我們做了許多嚐試，將最新的理論，也就是CI (Continuous Integration)導入Cloud OS的研發。測試Cloud OS的主要問題是一次完整測試花費時間太長，這會造成CI流程推動的困難。因此我們對流程做了修改，選取完整項目的子集合作為Acceptance Test，通過Acceptance Test的結果才能正式進入完整測試，讓CI流程在效率與正確性之間得到平衡，才能順利推動CI。本文對Cloud OS如何進行整合測試做介紹。

2．Cloud OS 整合測試之挑戰

2.1 Cloud OS 概述

Cloud OS[8] 是工研院雲端中心研發的雲端運算系統，目的是提供 IaaS 服務。主要核心技術為伺服器虛擬化、儲存裝置虛擬化及網路虛擬化等三個層次，以彈性配置CPU、記憶體、儲存及網路等資源來提供雲端運算服務。Cloud OS主要軟體功能有資源管理與虛擬叢集配置、分散式儲存系統、資料備援與回復、網路與系統管理，以及雲端服務系統安全防護等

五個面向之雲端作業系統管理技術。資源管理與虛擬叢集配置技術管理四大系統資源：中央處理器(CPU)、記憶體、磁碟儲存及網路等之最佳化配置。主要軟體模組有：Physical Data Center Management(PDCM)、Virtual Data Center Management(VDCM)、VM Management(VMM)、Distributed Storage(DMS/DSS)、Resource Provision Management(RPM)、Security、Directory Server、Eucalyptus Node Controller(NC)、Hypervisor(Xen)等。這些模組之間互相溝通才能達到上述Cloud OS的軟體功能。Cloud OS架構如圖1所示。

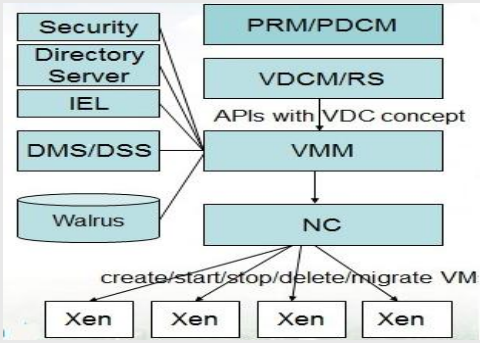

圖 1　Cloud OS 架構

2.2 挑戰

在這麼複雜的軟體系統上維持軟體品質是一項挑戰，主要困難包括:

(1) 模組數量，每個模組都有各己的測試需求。
(2) 模組之間交互作用複雜。
(3) 某些測試步驟複雜且費時。
(4) 軟體變動頻繁，開發者無法完全保證每個版本的品質。

每個模組都有各自的單元測試，但是整合之後，每個模組提供的功能仍需要在整合好的環境再進行測試，確認整合之後功能仍然正常。整合之後有些功能會受到其他模組的影響，例如所有網路連線都會受到L2 Directory Server的影響、NC需要所有模組都正常運作才能順利開啟VM。同時，如果有些測試失敗，也有可能會造成後續測試連續失敗，例如HA (High Availability)的測試失敗之後可能讓整個系統都處於異常的狀態，造成後續測試都失敗。

這些問題讓每次的測試都需要花上許多時間，更有甚者，測試順序排在後面的測試項目往往因為先進行的測試失敗而到最後因為時間不夠而無法進行或無法得到正確的結果。最後，因為一次完整測試的時間太長，開發部門發布新版本的時候沒有辦法確保基本品質，可能會給測試部門不良的版本(Bad Release)。測試這些不良版本造成人力與時間的浪費，更進一步造成測試部門的負擔。

3 · Cloud OS 整合測試流程改進

針對這些挑戰，我們做了許多的改進，根據每個模組的功能，我們花費大量時間蒐集與產生測試項目。為了更有效的運用時間，我們致力於測試自動化。為了避免開發部門發佈不良版本，我們進行了發佈流程的改進。為了在版本正確性與發部流程的時效之間達到平衡，我們對所有測試案例做了檢視，移除重複測試的測試案例並選出最基本的測試案例給開發部門做為發佈版本的必要流程。以下針對各分項做更詳細的介紹：

3.1測試案例的蒐集

自從2011年開始，針對Cloud OS的測試案例開始有系統的蒐集，第一年蒐集約400個案例，到2013年總共蒐集了1300個案例。到2013年為止的案例主要是由工研院內部針對Cloud OS所有功能所撰寫。2013年之後隨著Cloud OS出貨給客戶，也從客戶端得到回饋，客戶的UAT案例與壓力測試以及使用上遇到的問題也被蒐集到我們的測試案例庫之中。

3.2測試案例的自動化

隨著測試案例的蒐集，很明顯的趨勢是這些案例不可能全由手動執行。因此我們同步開始測試案例的自動化。我們使用Jenkins[5][6]做為CI (Continuous Integration)的主要工具，測試的腳本則是使用Selenium[2]與Robot Framework，對Cloud OS的Web介面進行操作，這樣就可以將大部份需要人介入的操作自動化。

在選擇測試自動化工具時有兩種主要不同方式，使用 Record-Replay 的方式或是 Script

程式撰寫的方式。採用 Record-Replay 的方式好處是產生測試案例比較容易，只需要操作 UI 介面即可。撰寫腳本的方式相對複雜許多，許多簡單的動作(例如:Click Button 並等待某個元件狀態改變) 就需要上許多程式碼。但是最終我們還是選擇使用撰寫程式碼的方式。主因有幾個:首先是在自動化的過程中發現，許多的測試都有共同的步驟，因此程式碼可以重複使用這些共同步驟。Record-Replay 的方式就無法有效利用這些重複的部份。其次是我們的測試中，有許多變數的部份(例如每次得到的 VM IP)都不同。程式碼的方式可以取得這些變數，所以每次案例執行就可以設定不同的數值給變數。Record-Replay 的方式也不容易做到這點。最後是有些測試案例需要在某個步驟之後呼叫外部的程式(例如呼叫 Python Script 連線並登入到 VM 並做某些檢查)，使用 Record-Replay 方式達到這個目的也有困難。針對撰寫程式碼的方式的主要困難點，也就是用程式去模擬真人操作時步驟很繁瑣的部份，我們將許多基本步驟寫成獨立的函式，因此雖然在一開始進度很慢，但是在底層函式庫建立之後，再用這些函式組合出新的測試案例速度就加快許多。因此我們可以讓測試案例保有彈性與再利用性，同時也不致於在撰寫測試案例時花費太多時間。

3.3測試案例的刪減

測試案例的蒐集與自動化的數量在2013年達到最高峰，約有1300項。此時我們發現新的問題，儘管有測試自動化，全部執行完這些測試案例的時間仍然太久。檢討之後發現原本在蒐集這些案例的時候是各個小組分別針對他們所需要的項目去產生，其中有需多測試可能有重複(或是部分重複)或是有些案例可以利用其他的案例做為假設狀態(Pre-condition) 。這些情況下重複的案例可以刪除或合併，有先後順序的測試案例可以藉由流程的安排加速測試的速度。因此從2013開始，就開始全面的檢討測試案例的必要性與關聯性。從2013年最多的時候約1300項持續的縮減案例數量，到2015年只留下900項必須的項目。測試需要的人力由最多的六個人到2015年只需要一個人執行測試即可完成。

3.4發布流程的改進

除了改進測試的自動化之外，另一個問題是減少開發部門發出不良版本的機會。在開發過程中發現RD單位因為專注於新功能的開發，常常忽略對原有功能的檢查。也就是沒有做Regression Test，造成的結果是一個Bug常常在Close與Reopen之間來回，同時造成溝通的問題。測試人員與RD討論時常常因為版本不同而浪費許多時間。

因此，我們重新設計了每次Release的必要步驟。我們從900個測試案例中選出最少需要通過的測試項目稱為Acceptance Test。RD發佈的版本必須要通過Acceptance Test才會被認為是良好的版本(Good Release) ，同時也導入CI的概念，所謂CI的觀念可以用圖2來表示，也就是在開發與測試是持續進行的過程，而不是發佈之後再開始進行測試。這個概念就是，開發完成(Programming)之後先做單元測試(Build/Unit test) ，通過之後再做整合測試。如果測試失敗，就回到RD重新開發。

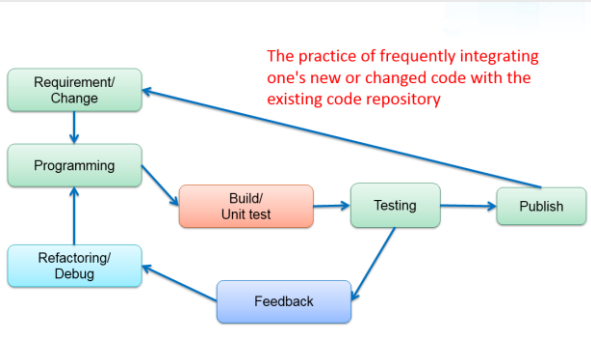

圖 2　Continuous Integration Concept

導入這個流程的目的就是希望RD在開發過程可以持續的進行整合測試，只有經過驗證的Good Release才會進入完整測試的步驟。基於這個概念，套用到Cloud OS開發流程之後我們得到一個適用於Cloud OS的流程。其中最重要的

就是在RD發布版本之前，加入一個測試的步驟(Acceptance Test)，通過這個測試才算良好版本(Good Release)，之後才進行完整的測試。在選擇Acceptance Test我們面臨的挑戰則是如何在完整性與時效性之間取得平衡。Acceptance Test太多則影響RD開發的時效，太少則失去這個測試做為把關者的意義。目前我們的Acceptance Test約耗時4個小時，也就是當RD需要發佈新版本時，需要花半天的時間通過最基本的測試，目前看來是一個可以接受的成果。圖3為目前的完整流程:

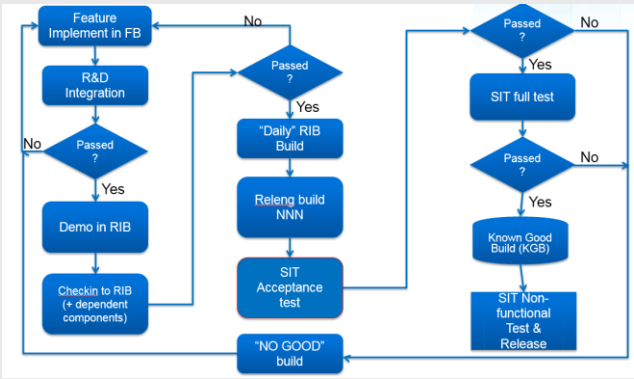

圖 3　適用於 Cloud OS 的 CI 流程

另外也加入靜態程式碼分析，每個版本都必須通過Coverity[7]的檢查，確認內部開發的程式碼都沒有High impact的錯誤，才可以發佈。

3.5其他工具的使用

除了測試案例的建立之外，還有其他輔助工具也對於整合測試很有幫助。其中一個是Bug Tracking System，我們使用的是JIRA[9]。一個好的Bug Tracking System可以幫助擔任不同角色的人員對同一個Bug溝通與瞭解。在Cloud OS的開發過程中，我們使用JIRA做Bug的追蹤，同時也使用JIRA做新功能開發的追蹤。當一個Bug或是新功能進入JIRA系統之後，這個JIRA項目會指派給RD的Team Lead，再由Team Lead指派給負責的RD。當RD完成工作項目之後，JIRA[9]項目會回報給發現Bug的測試人員，測試人員預期在下一版的Release看到這個項目被解決。舉報與修正Bug的流程如圖4:

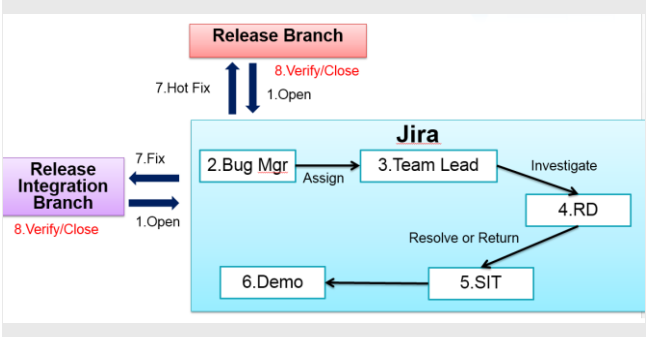

圖 4　JIRA Issue Flow

另一個使用的工具是軟體版本控制。在Cloud OS中使用的是SVN，目前比較流行的版本控制是Git。但是當Cloud OS開始開發的時候還是以SVN[10]比較成熟，因此延用下來Cloud OS就一直維持在SVN。但是有些RD會採用折衷的辦法，也就是在RD自己的電腦上使用Git，因為Git有支援Git-SVN之間的溝通。可以將SVN上的程式碼下載到local電腦成為Git的Repository，這樣就可以使用Git的較為強大的Local Version Control，等確認功能完成之後再當成一次Commit回傳到SVN。好處是Git可以在每次小規模的變動就Commit，鼓勵RD在開發過程中持續的Commit，方便在RD自己的電腦上做版本控制。但是這些小的Commit不會回到SVN，所以SVN上不會有太多片段的Commit紀錄。

4．整合測試流程改進成果

經過第三章描述的測試流程改進之後，整合測試的效果得到大幅改進。測試所需時間由每個Release 15個人日減少到每個Release 5個人日即可完成測試。演進過程如圖5所示:

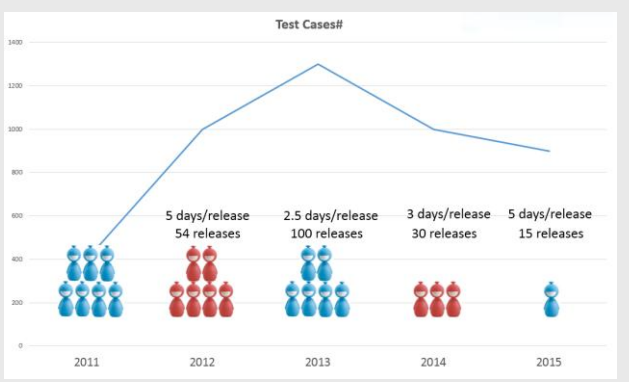

圖 5　CloudOS 測試案例與所需人力的演進

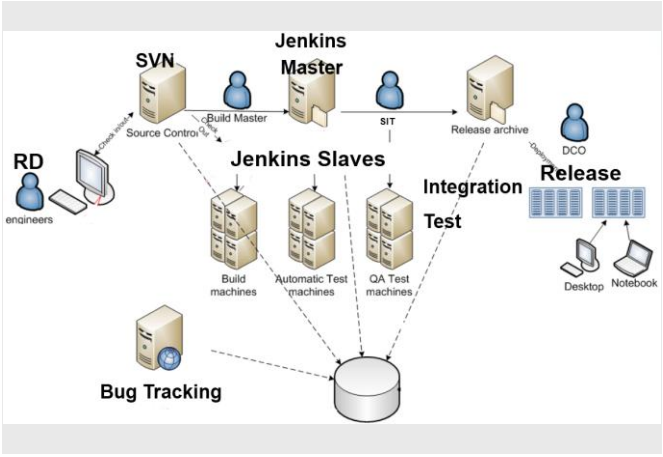

圖 6　CloudOS 開發與測試使用環境

圖6所示則是完整的軟體開發流程，包括Server組成，不同角色在流程中扮演的任務。我們有獨立的Bug Tracking與SVN伺服器。重點在Build與Testing的部份，為了讓許多人能同時使用Jenkins系統[5][6]，所以使用的是Master-Slave方式。這個模式下需要預先建立許多Jenkins Slave(圖6中的 Build machines, Automatic Test machines等等)，每個Slave被指派不同任務，例如是Build Slave或是Testing Slave。使用者都連到同一個Jenkins Master，由Jenkins Master指派工作到不同的slave，同一時間只要有足夠的Jenkins Slave能提供使用者需要的服務即可。這種架構就可以支援大量工作與大量使用者同時上線進行開發與測試的需求。

5．結論

本文介紹Cloud OS的測試與開發流程的演進，從幾乎沒有開發流程規範時遇到的問題與挑戰到目前有良好流程情況得到的改進。並且概略描述目前使用的流程。目前的流程就是將CI (Continuous integration)的觀念導入，並且強調測試的自動化，在導入的過程中，也根據Cloud OS特性與人員工作流程做了一些調整，得到目前的成果。經過這樣流程讓軟體品質得到大幅度的增進。

經由檢視這一路以來的演進，Cloud OS提供了一個實驗的平台，可以發現在大型的軟體開發過程中，良好的軟體開發流程對於軟體品質的幫助十分明顯。良好的流程可以幫助RD在早期就發現問題所在，避免RD發布不良的版本給整合測試部門。當測試部門得到版本都是良好的版本(Good Release)時，就可以更快的時間完成測試，盡快給開發部門回饋，讓開發與測試之間形成良性循環。

同時在導入CI的過程另外得到的結論是，當計劃進行到某個規模之後再來導入CI 會遭遇到許多困難與阻力，例如我們花費許多的人力在完成足夠的測試案例。這是因為Cloud OS已經有許多功能都已經完成之後才開始導入CI，所以這些測試案例都需要額外的人力與工作專門來撰寫。自動化的過程與開發流程的變更也導致開發部門的工作量增加，造成導入的阻力。我們得到的結論是CI的導入最好是在軟體開發計畫的初期就開始，再隨著軟體的複雜度增加而調整，如此一來每次的變動都是漸進式的，遇到的困難與阻力都會減少。而且實務上並沒有所謂完美的流程，期望一次導入完美的流程也是不需要的，因此越早導入良好的開發與測試流程會得到越好的成效。

參考文獻

[1] Continuous integration, https://en.wikipedia.org/wiki/Continuous_integration
[2] Selenium, http://www.seleniumhq.org/
[3] Mark Fewster & Dorothy Graham, Software Test Automation, Addison-Wesley Professional (September 4, 1999)
[4] Paul Duvall, Steve Matyas, and Andrew Glover, Continuous Integration Improving Software Quality and Reducing Risk, Addison-Wesley Professional; 1 edition (July 9, 2007)
[5] Jenkins, https://jenkins.io/
[6] John Ferguson Smart, Jenkins: The Definitive Guide Continuous integration for the masses, O'Reilly Media, July 2011
[7] Coverity, http://www.coverity.com/
[8] ITRI Cloud OS, https://www.itri.org.tw/eng/Content/MsgPi

c01/Contents.aspx?SiteID=1&MmmID=620651706136357202&MSid=621024023620073056
[9] Jira, https://www.atlassian.com/software/jira
[10] SVN, https://subversion.apache.org/

作者簡介

賴建億

工研院資通所/資料中心系統軟體組/資深工程師。研究領域包括軟體測試方法。