約耳測試：邁向高品質的 12 個步驟

作者：周思博 (Joel Spolsky)
譯：Paul May 梅普華
編輯： Nick Wong
August 9, 2000

聽說過 SEMA 嗎？這是一套相當深奧的系統，可以測量軟體團隊的好壞。等一下！不要急著連過去看。光是要搞懂那東西大概就要花上六年了。所以我自己有一套無責任的簡易方法來衡量軟體團隊的品質。這套方法的好處是只要花 3 分鐘左右。省下的時間足夠讓你唸趟醫學院。

約耳測試 (Joel Test)

1. 你有使用原始碼控制系統嗎？
2. 你能用一個步驟建出所有結果嗎？
3. 你有沒有每天都重新編譯建立(daily builds)嗎？
4. 你有沒有問題追蹤資料庫(bug database)？
5. 你會先把問題都修好之後才寫新的程式嗎？
6. 你有一份最新的時程表嗎？
7. 你有規格嗎？
8. 程式人員有沒有安靜的工作環境？
9. 你有沒有用市面上最好的工具？
10. 你有沒有測試人員？
11. 有沒有在面試時要求面試對象寫程式？
12. 有沒有做走廊使用性(hallway usability)測試？

約耳測試的好處是每個問題都很容易回答**是**或**否**。你不必計算每天寫的程式行數或是每個轉折點的平均問題數量。只要答「是」就加 1 分。約耳測試的缺點是*絕對不能*用來確保核電廠的安全性。

得 12 分是完美，11 分勉強可接受，不過 10 分以下(含 10 分)就表示問題大了。事實上大部份軟體組織都只拿到 2 或 3 分，這些組織都*岌岌可危*，因為微軟隨時都是以 12 分的水準運作。

當然啦，這些並不是決定成敗的唯一因素：特別是當你的優秀團隊做些沒人要的產品時(對，沒人要)。另外也可能有那種「高手」團隊，即使完全不鳥這些東西卻還是能做出改變世界的夢幻軟體。不過除此之外其他人都一樣，如果你能把這 12 件事做好，就能建立一個能穩定出貨的紀律團隊。

1.你有使用原始碼控制系統嗎？

我用過一些商用原始碼制系統也用過免費的 CVS，我可以告訴你 CVS 相當*不錯*。不過如果你沒有原始碼控制系統，當需要程式人員合作時你就會被壓垮了。程式人員沒法子知道其他人做了些什麼。也不能輕易回復成出錯前的狀態。原始碼控制系統還有另一個優點，就是原始碼會被登出(check out)到各個程式人員的硬碟裡 -- 我還沒看過哪個用了原始碼控制的專案會遺失大量程式。

2.你能用一個步驟建出所有結果嗎？

我的意思是：從最新的原始碼快照開始，要花多少步驟才能建立出貨用的軟體？好的團隊會有單個腳本檔案，只要執行這個檔案，就會從頭登出所有檔案，編譯每一行程式，建立執行檔(包含所有不同版本，語言以及#ifdef 組合)，製作安裝程式，並且產生出最後要用的媒體形式 -- 光碟片編排，網站下載或是其他各種形式。

如果這個程序不只一個步驟就會容易出錯。另外當出貨時程緊逼時，修正「最後的」問題，製作最終執行檔等等的過程要能飛快地完成。如果程式編譯和安裝檔製作等動作要 20 個步驟才能完成，你一定會急瘋掉並且做出一些蠢事。

就是為了這一點，我前一家公司把原本用的 WISE 換成 InstallShield：我們需要能透過 NT 工作排程器，在晚上用描述檔自動執行的安裝製作程序，由於 WISE 不能透過工作排程器半夜執行，所以我們就把它丟掉了。(親切的 WISE 員工跟我保證他們的最新版一定會支援夜間執行。)

3.你有沒有每天都重新編譯建立(daily builds)嗎？

在使用原始碼控制工具時，有時候程式人員會不慎登入(check in)某些內容而導致編譯失敗。舉例來說，某人新增加了一個原始程式檔，整個程式在他的機器上都能正常編譯，可是卻忘記把新增的程式檔加到原始碼控製程式庫中。結果這位仁兄健忘並快樂地鎖上機器回家了。其他人都不能做事，所以也只好很不爽地回家。

導致編譯失敗非常糟糕(又經常發生)，這時每天重新編譯建立就很有幫助了。它能保證不會有漏網之魚。在大型的團隊中，要確保能立即修正編譯失敗的最佳方法就是每天下午(像是午餐時間)重新編譯。大家在午餐前儘可能的登入檔案。等大家回來的時候已經編譯完畢。如果結果正常，很好！大家可以登出最新版的原始碼繼續工作。如果有問題就去把它搞定，而其他人還可以用前一版沒問題的程式繼續幹活。

我們 Excel 團隊有個規定，導致編譯失敗的人必須從此負責重新編譯的動作(作為處罰)，一直到有其他人出錯為止。這是個讓人不要導致編譯失敗的好誘因，同時是個讓大家輪流處理重新編譯的好方法，這樣大家都會知道怎麼做。

我這篇文章裡有更多每日重新編譯的資料：Daily Builds are Your Friend。

4.你有沒有問題追蹤資料庫(bug database)？

不管你說什麼。只要你在寫程式(只有一個人寫也一樣)，如果沒有一套良好的資料庫列出程式中所有的問題，一定會產生品質低劣的程式碼。很多程式人員自認能把問題清單記在腦裡。才怪。我從來沒法子一次記住超過二或三個問題，而且會在第二天早上或是趕著出貨時把它們全部忘掉。你一定要正式的記錄問題。

問題資料庫可大可小。一個最簡化的有效問題資料庫必須包含每個問題的下列資料：

- 重現問題的完整步驟
- 應該看到的行為
- 實際看到的(有問題的)行為
- 被指派的負責人
- 是否已修正

如果你是覺得問題追蹤軟體太複雜才不追蹤問題，建個 5 欄的表，填上這些重要欄位*然後開始用吧*。

想深入瞭解問題追蹤，請參閱無痛錯誤追蹤。

5.你會先把問題都修好之後才寫新的程式嗎？

古早第一版的 Microsoft Word for Windows 被視作為「死亡行軍」型專案。進度一直落後。整個團隊的工作時間長得離譜，專案卻一延再延三延，大家都承受到無比的壓力。拖了幾年後那個鬼東西終於上市了，微軟就把整個團隊送到 Cancun(墨西哥著名海灘)渡假，然後再坐下來做些深度反省。

他們發現專案經理過度堅持要保持「進度」，結果程式人員只能趕工寫出爛程式，而且正式的時程並沒有包含錯誤修正的階段。沒有人試圖要減少問題數量。而且實際上剛好相反。有個程式人員要寫程式計算一行文字的高度，結果他只寫了"return 12;"並等問題報告出爐說這個函數功能不對。時程表變成一份等著被轉換成問題的功能列表。事後檢討時稱之為「無窮錯誤法」。

為了修正這個問題，微軟全面採用所謂的「零錯誤作法」。很多公司裡的程式人員都不禁竊笑，因為聽起來像是管理階層認為能用行政命令降低錯誤數量。實際上「零錯誤」是指無論何時都要*先*修正錯誤才能寫新程式。原因如下。

一般來說，愈晚修正錯誤，修正所付出的成本(時間及金錢)愈高。

舉例來說，當你打錯字或出現編譯器會發現的語法錯誤，要修正只是小事一件。

當你的程式第一次執行出錯時，應該也能立即改正，因為整個程式還在你腦海裡。

如果要為幾天前寫的程式除錯，應該需要回想好一陣子，不過當裡重讀所寫的程式之後，就會記起所有細節並在適當時間內把問題修好。

不過如果你要為*幾個月*前寫的程式除錯，很可能已經忘掉了一大半，要修正就是難上加難。你也可能正在替*別人的*程式除錯，而當事人可能正在阿盧巴渡假，這時候除錯就像科學一樣：你得條理分明小心翼翼地慢慢來，而且也無法確定要多久時間才能解決。

另外如果要為*已出貨*的程式除錯，要修正問題的代價可是難以估算的。

要立即修正問題的理由之一，就是因為這樣做能少花點時間。另一個理由是寫新程式的時間還比修正現有錯誤的時間較易估計。舉例來說，如果要你估計寫個串列排序的程式需要多久，你應該能估算得相當準確。不過如果說你的程式在裝了 Internet Explorer 5.5 之後有問題，要估計需要多久才能修好這個問題，恐怕你連*猜*都不會，因為你不知道(當然不知道)問題*哪兒*來的。要找出問題可能要花 3 天，也可能只花 2 分鐘。

我的意思是如果你的計劃時程裡有很多錯誤待修正，這種時程是不太可靠的。不過如果把*已知*的錯誤都修好了，所剩的就只要新程式了，那麼你的時程就會變得非常準確。

把錯誤數量維持在零還有另外一個優點，就是面對競爭時反應更快。有些程式人員認為這樣做能讓產品*隨時能推出*。所以如果競爭者推出某個殺手級新功能來搶客戶，你只要把那個功能加上去就可以立即出貨，不必去修正累積下來的大量問題。

6.你有一份最新的時程表嗎？

我們在這裡要談談時程表。如果你的程式對公司非常重要，有太多理由可以說明預知程式完成時點有多麼重要。程式人員不愛訂定時程可是惡名昭彰。他們會對業務大叫：「該完成的時候就會完成！」

但是問題不可能就這樣算了。業務人員有太多的計劃決策必須遠在程式出貨之前做決定：展示，商展，廣告等等。而做決定的唯一方法就是定出時程並隨時更新。

擁有時程的另一個重點是逼你決定要製作哪些功能，並且能逼你*剔除*最不重要的功能而避免功能過度膨脹(featuritis，又名 scope creep)。

要維護時程表並不困難。請參閱我的文章 Painless Software Schedules，文中敘述建立好用時程表的簡單方法。

7.你有規格嗎？

寫規格像用牙線：大家都同意這是好事，卻沒有人真的在做。

我不知道為什麼，或許是因為大多數程式人員都討厭寫檔案吧。所以當全是程式人員的團體面對問題時，自然傾向用程式碼而非檔案來表示答案。他們寧願跳進去寫程式也不願先寫規格。

在設計階段發現問題時，只要改幾行就能輕易修正。等程式寫出來之後，修正的代價就高得多了，代價包含了情感(人們討厭拋棄程式碼)和時間，所以會抗拒修正問題。通常未依據規格製作的軟體 最後的設計都很糟，而且進度完全無法控制。 這似乎就是發生在 Netscape 上的問題。它的前四版變得一團亂，結果管理階層愚蠢地決定 把程式丟掉重新開始。然後他們在 Mozilla 上又重蹈覆轍，造出了一個無法控制的怪物，而且耗了*幾年*才進入 alpha 測試階段

我的拿手方法是把程式人員送去上密集的寫作課程，讓他們變得不那麼排斥寫作，就可以解決這個問題。另一個方法是僱用聰明的專案經理來寫規格。不管用哪一種方法，你都應該強制執行「沒有規格不寫程式」這個簡單的規則。

你可以由我寫的四篇系列文章學到所有關於規格的內容。

8.程式人員有沒有安靜的工作環境？

有大量的檔案記載，為知識工作者提供空間安靜及隱私可以提昇產能。軟體管理經典 Peopleware 大量記錄了這種產能上的增益。

這就是問題所在。我們都知道知識工作者進入「狀況」(flow，也被稱作 in the zone)時工作效果最佳，這時候他們會完全與環境脫離，全心專注在工作上。他們忘記時間並透過絕對專注產出極佳成果。他們所有豐富的產出也都是在這個時候完成的。作家，程式人員，科學家，甚至籃球球員都會告訴你進入「狀況」的情形。

問題是要進入「狀況」不是那麼容易。如果你有試著計時，平均大概要 15 分鐘才能開始全速工作。有時如果你累了或是那一天已經有很多創造性的成果，會根本無法進入「狀況」，然後看看網頁玩玩俄羅斯方塊打混過完一天。

還有一個問題就是很容易*脫離*「狀況」。噪音、電話、同事的中斷(*特別是這一點*)都會讓你脫離「狀況」。假設有個同事問了一個問題讓你中斷了 1 分鐘，實際上卻會讓你完全脫離「狀況」，得再等半個小時才能回復生產力，結果你的整體產能都出問題了。如果你身在一個喧鬧的 BULLPEN 環境中(像那些一窩蜂(caffeinated)網路公司最愛營造的典型)，行銷部門在程式人員旁對著電話大喊，你的產能就像一直被中斷的知識工作者一樣顛簸，永遠無法進入「狀況」。

這對程式人員來說更加嚴重。生產力多寡在於是否能在短期記憶體中處理大量的細節。任何一種中斷都會讓這些細節完全消失。等你轉回來工作時就完全不記得任何細節(比如正在使用的區域變數名稱或是搜尋演算法寫到哪了)，必須把剛剛的東西找出來，於是速度就放慢下來一直到你回復為止。

這裡有個簡單的算術。我們可以說(依照陳述所暗示的)雖然僅僅打斷程式人員一分鐘，事實上是去掉了 15 分鐘的產能。以此為例，假設有兩個程式人員 Jeff 和 Mutt，把他們安排在一個標準呆伯特(Dilbert，美國漫畫)養牛場裡相鄰的開放隔間中。Mutt 忘記了 strcpy 函數的 Unicode 版本拼法。他可以花 30 秒自己查出來，也可以花 15 秒問 Jeff。由於人就坐在旁邊，所以他問 Jeff。Jeff 分心所以就損失了 15 分鐘的產能(替 Mutt 省了 15 秒)。

現在把他們搬到兩間有牆有門的獨立房間裡。如果 Mutt 忘記那個函數的拼法，他可以花 30 秒查出來，也可以花 45 秒過去問 Jeff(就典型程式人員的身裁來說離開位置並不輕鬆)。結果他就自己查了。於是 Mutt 損失 30 秒的產能，不過卻替 Jeff 省下 15 分鐘。哎呀呀呀！

9.你有沒有用市面上最好的工具？

用編譯語言撰寫程式是一般家用電腦還無法瞬間完成的最後幾件事之一。如果你的編譯過程超過數秒，去找台最新最棒的電腦可以替你省點時間。如果編譯需要超過 15 秒，程式人員覺得無聊就會跑去看線上新聞 The Onion，然後陷在裡面耗掉幾個鐘頭的產能。

在單螢幕系統上替 GUI 程式除錯並非絕不可能，不過用起來有夠痛苦。如果你在撰寫 GUI 程式，弄兩台螢幕會讓你輕鬆許多。

大部份程式人員到最後都得修整圖示或工具列所用的圖，可是大部份人都沒有一個好用的圖形編輯器。用微軟的小畫家修圖簡直是笑話，不過卻是大多數程式不得不做的事。

在我前一家公司，系統管理員會一直送些垃圾信給我，抱怨我在伺服器上使用了超過「220 MB」的硬碟空間。我說依據現在硬碟的價格，這點空間的費用還遠比不上我所用的*衛生紙*。即使只花 10 分鐘清理目錄也是生產力的極大浪費。

一流的開發團隊不會虐待他們的程式人員。即使工具不好所引起的挫敗很小，累積起來都會讓程式人員心情不爽脾氣暴躁。而不爽的程式人員就等於無生產力的程式人員。

除此之外...程式人員也是很容易用最酷最新的東西賄賂的。這可遠比再增加薪水叫他們工作 要便宜多了！

10.你有沒有測試人員？

如果你的團隊沒有專門的測試人員(至少每兩到三個程式人員要配一名)，你要不是推出問題很多的產品，就是浪費錢叫時薪 100 美元的程式人員去做測試員(時薪 30 美元)做的事。省測試員絕對不是真省，這實在是再明顯不過了。我實在很驚訝很多人卻還認不清這一點。

看看 Top Five (Wrong) Reasons You Don't Have Testers 吧，這是我針對這個題目所寫的文章。

11.有沒有在面試時要求面試對象寫程式？

你可能不叫魔術師先表演幾招就直接僱用嗎？當然不會。

你可能不先嘗嘗菜就決定自己婚宴的餐廳嗎？我很懷疑。(除非是 Marge 姑姑，如果不讓她弄一道「頂級」碎牛肝餅，她會恨你一輩子)。

儘管如此，現在程式人員是否錄用都還是要看履歷是否突出，或是因為主試人員面談聊得很高興，或是回答些查檔案就知道的瑣碎問題(比如 CreateDialog() 和 DialogBox()間的差異是什麼？)。你根本不會管他們能否背出幾百條有關程式設計的瑣事，你真正在意的是他們能否寫出程式。更糟的情況是問那種「啊！我懂了！」的問題：就是那種知道答案時理所當然，可是不知道答案時卻莫名其妙的問題(譯註：像是腦筋急轉彎)。

拜託*別再這樣做了*。隨便你想怎麼面試都行，不過記得一定要讓面試者寫些程式。(需要更多建議時可以看我寫的 Guerrilla Guide to Interviewing。)

12.有沒有做走廊使用性(hallway usability)測試？

*走廊使用性測試*是說到走廊攔住下一位經過的人，然後逼他試用你剛寫好的程式。如果你做夠五個人，就可以發現程式中 95%應注意的使用性問題。

良好的使用者介面設計並沒有想像中那麼困難，在吸引客戶中意並購買產品時又是極為重要的。你可以參閱我寫的免費線上 UI 設計書，是針對程式人員的短篇入門書。

不過處理使用者介面時有一點最重要：如果你把程式展示給少數幾個人看(事實上五或六個就夠了)，就能快速地發現一般人會遇到的主要問題。Jakob Nielsen 的文章 中有解釋原因。即使你的 UI 設計技巧不足，只要強逼自己實行不花什麼工夫的走廊使用性測試，就會讓你的 UI 水準大幅提昇。

約耳測試的四種用法

1. 對你自己的軟體組織評分，再把分數給我作為講八卦的題材。
2. 如果你是一個程式設計團隊的經理，可以用它來確保團隊能在最佳狀態工作。等拿到 12 分之後，就可以把程式人員放著不管，專心去避免業務的干擾就好了。
3. 如果你正在決定是否接受一份程式設計的工作，可以問問未來可能的僱主他們能拿幾分。如果分數太低時要先確定你有權修正這種問題。否則你將會灰心喪氣而且一事無成。
4. 如果你是個正在評估某個程式設計團隊價值的投資者，或是你的公司正考慮與其他公司合併，這個測試可以提供快速的判斷方法。