測試驅動的開發

就其對軟件質量和程序員生產率的影響而言，TDD是最具爭議的敏捷實踐之一。經過十多年的研究，陪審團仍未確定其有效性。 TDD承諾：提高質量和生產率，並通過不斷增長的測試庫的安全網絡支持嶄新的，乾淨的設計。更重要的是，該食譜聽起來非常簡單：不要在未通過測試的情況下編寫代碼。

在這裡，我們將重新審視TDD承諾的證據。但是，在繼續之前，請先停下來思考以下核心問題的答案：什麼是TDD？

讓我們猜測：您的響應很可能是這樣的：“ TDD是一種在代碼之前編寫測試的實踐。”這個名稱強烈暗示著對測試優先動態的強調，這可能是大多數（即使不是全部）有關TDD爭議的根源。不幸的是，經常互換使用“ TDD”和“測試優先”是一個普遍的誤解。測試優先只是TDD的一部分。系統中還有許多其他齒輪可能會使TDD滴答作響。

如何處理小任務，保持紅綠重構週期短而穩定，只編寫通過失敗測試所需的代碼並進行重構呢？如果我們告訴您這些齒輪中的某些對實現TDD的承諾比測試實施的順序有更多的貢獻，該怎麼辦？ （提示：您應該事先提供證據。）

15年（矛盾的）證據

早在2003年，當軟件開發範式開始發生不可逆轉的變化（更好）時，肯特·貝克（Kent Beck）就根據傳聞提出了主張，並為軟件工程研究人員鋪平了道路：

尚無研究明確證明TDD與質量，生產率或娛樂性的許多替代方案之間的差異。但是，軼事證據是壓倒性的，其次要作用是顯而易見的。

從那以後，許多研究（例如實驗和案例研究）都對TDD的有效性進行了調查。這些研究在輔助研究中定期進行合成（見表1），只是為了揭示整個主要研究中相互矛盾的結果。這項研究還表明，TDD並不能帶來一致的整體收益，特別是對於整體生產率和質量子組而言。

為什麼結果不一致？除了表1中列出的研究內容的差異外，其他原因可能是：

•TDD的嵌齒輪過多，

•它的有效性在很大程度上受上下文的影響（例如，現有任務或個人技能），

•齒輪彼此之間高度相互作用，並且

•大多數研究只關注測試優先方面。

識別不一致的來源對於設計進一步控制這些來源的研究很重要。

MatjažPancˇur和Mojca Ciglaricˇ推測，研究結果表明TDD優於後測試方法，這是由於大多數實驗都採用了較接近瀑布法的粗粒度測試後方法作為對照組。這在處理之間的粒度上產生了很大的差異，有時甚至在對照中甚至完全缺乏測試，從而導致了不公平，誤導性的比較。最後，僅當與粗粒度開發過程相比時，TDD才可能表現更好。

行業採用（或缺乏）

關於TDD的討論很普遍，而且通常很激烈。但是在實踐中使用TDD有多常見？至少不是，至少這是證據所暗示的。

例如，在監測了416個開發人員超過24,000小時的開發活動之後，研究人員報告說，在聲稱使用TDD的項目中，只有12％的開發人員遵循了TDD。我們在與專業開發人員的合作中也觀察到了類似的模式。確實，如果有可能重新分析僅考慮此方面的所有現有證據，那麼事物的形狀可能會發生重大變化（無論好壞）。我們將成為魔鬼的擁護者，並提出疑問，如果來自TDD愛好者的證據是否基於非TDD活動中錯誤的個人經驗呢？

同樣，最近的一項研究分析了GitHub中所有（Java）項目的2015年9月快照。研究人員使用啟發式方法來識別類似TDD的存儲庫，發現只有0.8％的項目遵守了TDD協議。此外，通過將這些項目與對照組進行比較，該研究報告說，在以下方面，兩組之間沒有差異

•提交速度作為生產力的度量，

•修正缺陷的次數作為缺陷數量的指標，並且

•為質量報告的項目報告的問題數量。

此外，將拉取請求的數量和每個作者的提交分佈進行比較，並不表示對開發人員協作有任何影響。

Adnan Causevic及其同事確定了限制TDD在行業中使用的七個因素：

•增加開發時間（提高生產力），

•TDD經驗或知識不足，

•設計不足，

•開發人員測試技能不足，

•對TDD協議的依從性不足，

•特定於域和工具的限制，以及

•舊代碼。

不足為奇的是，這三個因素與開發人員遵循TDD的能力及其遵循的嚴格性有關。

是什麼真正使TDD滴答作響？

對TDD進行更精細的研究不僅涉及生產代碼和測試代碼的編寫順序，而且還涉及開發週期的平均持續時間，持續時間的一致性和重新投入工作。最近對39位專業人員的研究報告說，短開發週期的穩定節奏是提高質量和生產率的主要原因。確實，考慮到短而穩定的周期的影響，測試優先的影響完全消失了。這些發現與早期研究相吻合，表明TDD專家的周期長度比創新的短得多，並且變化少。阿利斯泰爾·考克本（Alistair Cockburn）解釋了大像生牛肉片（Elephant Carpaccio）的概念，並指出：“敏捷開發人員在其工作中應用了微甚至納米級的增量開發。” 15

Elephant Carpaccio與TDD處理小任務的概念有關的另一個說法是，敏捷開發人員可以快速交付“不是因為我們如此之快，我們可以[開發]速度是其他人的100倍，而是我們訓練了自己，要求最終用戶可見的功能要比大多數其他人小100倍。” 15為了測試這一點，我們進行了一些實驗，在這些實驗中，我們對任務描述的框架進行了控制（細粒度的用戶存儲與粗粒度的用戶存儲粒度的通用說明）。我們觀察到，任務描述的類型和任務本身是在TDD上下文中影響軟件質量的重要因素。

簡而言之，在短而穩定的開發週期中處理細小，定義明確的任務，對質量和生產率的影響比對測試實施順序的影響更大。

與測試優先口頭禪的背離

即使我們考慮只關注TDD測試優先性質的研究，仍然存在與TDD過程相符的問題。 TDD並非二分法，您要么每次都要先認真地編寫測試，要么始終在事後進行測試。 TDD是這兩個極端之間的連續頻譜，開發人員傾向於動態跨越這一頻譜，並根據需要調整TDD流程。據報導，在工業環境中，時間壓力，缺乏紀律以及對TDD利益的認識不足，導致開發人員偏離了這一過程。12

為了獲得更多見識，在一項基於民族學的研究中，研究人員通過包括音頻記錄和註釋在內的人工製品，更加緊密地監控和記錄了TDD開發過程。16他們得出結論，開發人員認為實施是最重要的。階段，並沒有嚴格遵循TDD流程。特別是，開發人員編寫了超出必要數量的生產代碼，常常省略了重構，並且沒有根據生產代碼的進展情況使測試用例保持最新狀態。即使開發人員遵循測試優先原則只是，他們在為下一個功能編寫測試之前就考慮了生產代碼（不一定是設計）應該如何。換句話說，也許我們應該簡單地將這種現象命名為“代碼驅動測試”？

TDD的內部和外部動態比編寫測試的順序更為複雜。沒有令人信服的證據表明，TDD的性能始終優於任何其他開發方法，至少是那些迭代的方法。並且有足夠的證據質疑TDD是否能夠兌現其承諾。

那麼，您如何決定是否以及何時使用TDD？ TDD的次要效果如何？

與往常一樣，上下文是關鍵，而TDD的任何潛在好處都可能不是由於編寫測試和開發人員遵循的順序而定。有現實的期望而不是崇拜或丟棄TDD是有意義的。關注開發節奏；例如，在短而穩定的開發週期中處理小任務，而不用理會測試訂單。另外，請記住，在“ TDD能力”方面，某些任務比其他任務更好（適合）。

這並不意味著您應該避免嘗試或停止使用TDD。例如，如果您認為TDD為您提供了為每個小功能編寫測試的自律性，那麼遵循“測試優先”的原則肯定會阻止您採用跳過測試的快捷方式。在這種情況下，堅持規則的價值是意味著不要在單元測試失敗的情況下不編寫任何生產代碼。但是，您應該首先考慮這些測試的質量（不要過度關注覆蓋率）17，而不是固定自己是否在代碼之前編寫它們。儘管TDD確實導致了更多的測試1,6，但缺乏對測試質量的關注12，包括可維護性和與生產代碼的共同發展，16可能令人震驚。

只要您知道並滿意生產率與可測試性和質量之間的潛在權衡（也許是長期的回報？），使用TDD就可以了。如果您在執行TDD時只是享受樂趣並感覺良好，而沒有任何明顯的缺點，那也很好。畢竟，證據表明，快樂的開發人員可以提高生產率並產生更好的代碼！18

分工：（4人）

1.**（庭誼） 會議記錄**

2.**（宗芳）決定專案軟體要用哪個＋解釋**

* 用Azure DevOps Services＋
* 說明他的優點
* 介紹他的操作（截圖）

3.**（睿芊）心得：文章主旨心得＋說明**（1page)

* 會議的價值：

1. 實體or線上（傳訊息就能解決就傳訊息就好了）
2. 不同專題的教授有不同meeting方式

* 人力資源的分配

1. 草創時期人太多，意見太複雜，冗員
2. 大頭會v.s.全體會（梅竹黑客松）

* 時間分割的殺傷力

1. 思考＋討論必須分開（工作性質混亂、PM的價值：責任分配不要一人包攬）

4.**（奕方）小組討論**：優缺點＋時間管理、學習成效（1page)

ex: 婷婷、通識小組