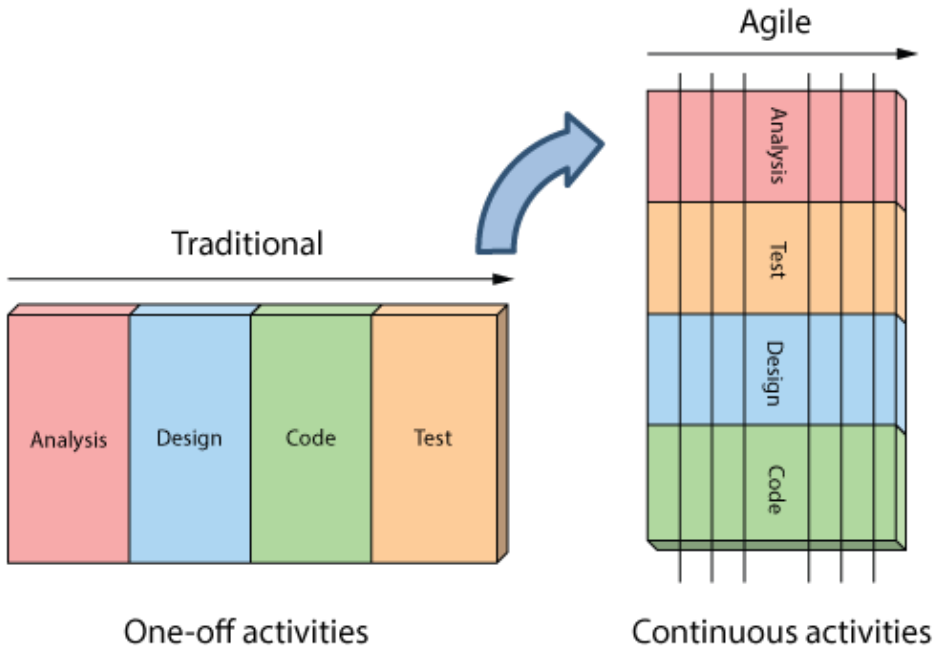
1. Test-first Development：在還沒有開始寫整個 project 的 source code 之前，就先把 test code 寫出來。目的是要確立需要寫哪些部分、哪些功能的 code，利用測試來推動整個開發的進行。此種開發著重於需求分析與產品質量的控制，避免在產品 source code 完成後沒有達到原定的目標。此方法是敏捷開發的核心觀念。
   Waterfall development model：project 的開發過程是通過一系列的階段而完成。從計畫→需求分析→設計→撰寫程式→測試，若在任一階段發現了問題，應該要返回上一階段去思考並做適當的修改。若成功地推向下一階段沒有發生問題，可以看作開發過程「流動」到了下一個階段，這也是此種開發 model 會被稱作 waterfall 的原因。

   Difference：以下圖去說明：



   簡而言之，左方(Waterfall)是將開發切割成「不同型式的階段」，一個階段完成後才會去做下一個；右方(TDD)是將 project 的目標切割成「不同開發目標的階段」，但每個階段都是從測試出發，然後才開始真正的撰寫 code。
   Waterfall 的方式可能造成撰寫完 code 開始進行測試時才發現問題點很多，要花很多時間去修復，也就是所謂 80/20 定理。而 waterfall 也會有產品目標改變時，需要花很多時間去處理的缺點，甚至有可能整個 project 重來。

Case Study A：
使用 TDD 的方式去開發會比較好。主要的原因在於 Program Manager's expertise 的專業程度。Program Manager's expertise 在每一部份功能開始開發時就要設計並測試好才可以下去做，在 Program manage 時會需要更高的水準，而 Case A 剛好有符合了這樣的特性。
Table3 可以讓我們知道，雖然時間花費會增加 25~35%，但 defect density 提高了很多。是適合使用 TDD 開發的。

Case Study B：
使用 TDD 的方式去開發為比較好，主要的原因在於 Test LOC/ Source LOC 的比值已經達到 0.88。雖然這題沒有看到 Program Manager's expertise 有突出的表現，但開發人數較多，且開發性質上 Test 佔的比例也不低，很適合套用在 TDD 上面。
由 Table6 我們可以看到，在換成 TDD 開發後，defects density 暴增成了 4.6 倍，而且改由 TDD 開發所付出的額外時間只有 15%，其實效果非常顯著。

2. NO
因為 TDD 會需要切割成很多個階段去獨立執行。會衍生出以下幾種特性：獨立測試性、測試驅動開發、緩慢發展。
除此之外，TDD 強調的是不斷地進行討論、整合，並且希望在面對各種變化時，大家都能有應變、change design 的能力。
以下會說明有些會造成 project 不適合 TDD 的例子：
首先是獨立測試性，這部份我們不以整個軟體的開發去看，而是假定一個軟體設計的過程：複雜資料庫的欄位設計為例：資料庫需要有 foreign key 的功能，也就是此欄位的值會和同表格其他欄位或是其他表格的某個欄位有關連性，不能兩者非同步地做資料的新增、修改、刪除。在一個好的資料庫欄位設計當中，foreign key 的相互連結數會是非常多的，在這樣的情況下常常在結束設計前，都沒有辦法拆開成一個個小階段去測試，而是直接一次做完後再慢慢測試、debug 會比較優。
再來是緩慢發展，如果顧客要求的是即時性得到產品的這樣一個 case，用 TDD 會花費較多的時間，顧客交叉比對了應徵的開發者開出的條件，當然會選用原本開發方法的開發者，因為他們並看不到一個好的軟體需要各階層的人不斷互動、溝通，他們只想得到開出 spec 與迅速取得想要的產品。
最後是討論、整合的困難點，這部分常會因為各公司文化的不同而出現很多不適合點。可能有的公司階級分明、多說多錯，這樣子只有上下級交代任務，並沒有什麼前瞻的想法可以拿來密切的討論，TDD 的精神也跟著失去。其實常常會有很多人在為公司開發產品的過程中，覺得多一事不如少一事，這樣就很有可能會捨棄 TDD 這樣密切討論，一步步整合成好產品的過程。

3. 我們想要設計一個 AES 加密解密程式，而在加密解密時我們可以選擇看到某幾個循環階段完成後加密文字長怎麼樣，而非一般我們使用 AES library 的函式直接得到最終結果，不知道中間進行了什麼。

   因為要知道加密中間的過程，等於我們要自己參考 AES 加密法的 spec，從頭實作出 AES 加密解密器。依照網路上的介紹，我們會需要實作以下幾個功能：

   (0) 將 16bytes 的 input 和 key 存成 4*4 的 byte 矩陣
   (1) GF256_mult_x() //傳進此函數的 uint_8 往左 shift 一位，溢位則要作 mod
   (2) GF256_mult() //傳進此函數的兩個 uint_8 進行相乘，溢位則要作 mod
   (3) GF256_inv() //回傳傳進此函數的 uint_8 的乘法反元素
   (4) affineTrans() //回傳傳進此函數的 unit_8 經過 affine trans. 的結果。
   (5) subBytes() //AES 每個循環階段會需要做的第一步，和(1)~(4)有關連
   (6) shiftRows() //每個循環階段將經過 subBytes()的內容作 row 的平移
   (7) mixColumn() //每個循環階段將經過 shiftRows()的內容乘上某個矩陣
   (8) keyExpansion()//算出每個循環階段經過 mixColumn()的內容要 XOR 的東西
   (9) extractRoundKey() //利用 round 值取得 keyExpansion 算出的結果。
   (10) addRoundkey()//每個循環階段經過 mixColumn()的內容要 XOR RoundKey
   而其中(4)~(7)因為 AES 加密法的設計，解密會是相反的流程跑回去，故會另外為這四個函數設計 inverse function。

   我們以函式開發的角度去看，我們可以將整個開發流程分成 7 個階段去進行開發：(0)(1)(2)(3)→(4)(5)→(6)→(7)→(8)(9)(10)→整合→inverse func.開發。
   第一階段：普通運算。按照 GF256 的規格設計，直接測試結果是否相符。
   第二階段：AES 講求經過 10 個循環階段之結果，測試時一定要先測過第一個循環階段才可以往下執行。此時如果出現之結果不符合應有結果，就可確定是此階段之實作有問題。因為此階段等同是加諸於第一階段的補充功能。
   第三階段、第四階段：仍舊停留在第一個循環階段去作測試。以加密法的設計流程來看，階段的設計等於是明文經過一個一個 system 而產出的結果。所以同階段之 input 在之前已被確認是對的，但 output 和我們寫出的測試不相符，則一定是 source code 的撰寫上出了問題。
   第五階段：和二、三、四階段是完全獨立的。可以在第一階段完成後就交辦此部份的設計與測試，才能及早完成 source code 的完成。
   第六階段：沒有 design 的部分，但會利用 AES library 裡的 function 所求出的結果作為測試是否符合設計的指標。此階段就是將前面五階段的 code 進行整合。通過此階段也等同完成加密部分。
   第七階段：也沒有 design 的部分，只是函式的轉換。利用解密結果比對即可得知 source code 是否符合我們想要的設計。