Life Cycle of System Development

系 統 開 發 的

生 命 週 期

姓名_____ 班別 _____ **ict**_____

目錄

系統開發的階段

系統開發生命周期的階段及項目初始化

系統開發生命周期是系統開發員建立高質素系統時所遵從的一個過程。
據美國的司法部門所定下的指引文件，共有 10 個官方的階段。

在高級程度電腦科的課程裡，將集中討論以下五個階段：

1. 系統分析
2. 系統設計
3. 系統實施、測試
4. 系統轉換、維修
5. 系統文件編製 (這階段是一個不斷進行的過程，滲透在其它的 4 個階段之內)

在系統開發生命周期的 10 個官方階段中，某些不在此討論，例如是程式初始化，而某些則合併在一起，例如開發階段、統合、測試階段，合併一起稱為「系統實施」，而計劃階段被歸納入「系統分析」。 請注意，系統分析及設計的教科書，可能並不跟隨這 10 個官方的階段，原因是對系統開發生命周期有不同的解譯。 舉例說，某些作者可能把系統轉換當作系統實施的一部分，並把維修視為一個獨立的階段。

無論如何，任何系統開發生命周期，總是由一個項目負責人開始，他通常是一個機構的高級管理人員，他看見有建立電腦信息系統的需要。 適合機構需要的現成套裝軟件，總會被認真考慮採用的可能性。 如果在市場上找不到合適的套裝軟件，而機構本身有內部的開發設計隊伍(IT 部門)，隊伍可能展開軟件開發的項目。 否則，機構可能僱用信息科技行業內的服務供應商(軟件公司)來建立系統。 這樣，機構便成為服務供應商的顧客。 項目的***利益相關者***包括批准項目的高級管理人員，項目初始人及建議系統的最終用戶。

1.*系統分析*

在系統分析這個階段中，主要的目標是確定將要開發的系統的_____。
收集方法有多種，如面談及調查等。 然後，項目在經濟、機構及技術方面的_____，將在可行性研究內探討。如果項目被批准，便需要(圖文)描述建議系統(新系統)的用戶需求的實現方法，而通過使用一些製模工具 model 可以減少不清晰的情況。

識別用戶需求 user requirements

系統必須達到用戶需求。 需求可分成功能性及非功能性的。

- ***功能性需求*** functional requirements
  指系統_____進行的事情或系統應包括的信息(因為沒有合適的信息或數據，系統便不能運作)。 例如，一個學校圖書館系統，應允許用戶，進行借還書手續、查閱紀錄、在網上搜尋書籍目錄，這是功能性的需求。

- *非功能性需求* non-functional requirements
  是關於系統的_____特性，如*性能*及*可使用性*。 舉例說，系統允許用戶使用萬維網瀏覽器接達，是一種非功能性需求。 另一個例子是用戶建議系統的服務可用性，如_____可用性。在系統設計的階段，亦常討論有關非功能性需求的事情。

基本上，最少有四個收集用戶需求的數據收集方法：

- *問卷調查*
  當需求信息所涉及的人數多，問卷是普遍的工具。 問卷是一系列書寫形式的問題，以便從個體中取得信息。 如果牽涉的人口非常多，可從人口中抽樣一組人(在可管理的範圍內)，並向他們派發問卷調查。 問卷及調查的問題，必須小心撰寫以避免不清晰。 有時候，在全面調查進行之前，會安排一個模擬測試給少量回應者，以偵測調查可能遇上的問題。 由於問卷及調查的_____性質，這個工具通常使用來收集一些一般的數據，而不是詳盡的用戶需求。使用問卷常用下列步驟引出信息：
  - *選取參與者 (或樣本)*
  - *設計問卷*
  - *管理問卷*
  - *跟進送回的問卷 (分析數據)*

- *面談*
  是收集用戶需求的最普遍使用的方法，因為它簡單_____。 通常面談是一對一的進行，向用戶提問他們對系統的_____。 有時候，由於時間限制或其它關係，採訪者可能進行小組面談，在同一時間內採訪數個用戶。 進行面談的步驟依次是：
  - *選取被採訪者*
    必須採訪所有項目的利益相關者。 特別是將經常使用被建議系統的最終用戶 end-user。亦應採訪將被建議系統所取代的現行系統之用戶。
  - *設計面談問題*
    封閉式問題(例如，對圖書館的每月使用量，你想要甚麼統計報告? Y-or-N)
    開放式問題(例如，在借閱櫃台上你希望有甚麼改善?)。
    探索式問題可從被採訪者中引出更多信息(例如，你可舉個例子嗎?)。
  - *準備面談*
    採訪者將制定一個面談計劃，包括面談問題的順序安排，預料的答案及跟進問題等
  - *進行面談*
    面談的首件事情不是發問問題，而是與被採訪者建立關係，取得信任，並願意真實回答問題。 採訪者應是一個獨立及專業的信息追尋員，並持不偏不倚的態度。 採訪者應先告知被採訪者面談的目的，及被採訪者為何被選取，然後進行面談。
  - *進行面談後的跟進*
    面談記錄應在面談後盡快完成。 記錄應以有用的形式記載，而非只是一系列未整理的面談筆記。 應讓被採訪者閱讀記錄，以肯定被收集的信息的正確性。

- ***觀察***

  是有效的方法，讓分析員了解用戶怎樣在_____環境下與現行系統互動，並取得第一手經驗。 就觀看用戶運作的過程，分析員看見操作的_____情況，因而發現任何被忽略或對用戶需求的錯誤敍述。 觀察方法可用作支援或反駁從面談取得的信息。

- ***文件審閱***

  讓分析員，檢查在機構內使用的文件，如_____、備忘錄及程序手冊等。 對那些資料詳盡的文件的關鍵性覆檢，可能展現用戶的期望。

使用以上介紹的一個或多個方法來識別需求後，必須在需求規格內制定詳盡的功能性或非功能性需求，成為一套對將開發的系統之需求的完整描述。

🛠 *活動*

- 對於下列的每個項目，決定收集相關信息的方法，令你能識別用戶的需求。
  - 現行的學校圖書館電腦系統，需要增加網上接達。
  - 體育科教師想開發一個系統，以便學生登記參加運動日的項目，編排運動會的項目，把學生分派到不同項目，記錄及操作結果。
  - 學校的學生組織打算提升網站，使它更容易使用及吸引所有的學生。
  - 學校管理計畫使用一個已裝置八達通咭閱讀器的系統，在學校入口自動點名
  - 開發一個學校資源管理系統，讓教師及學生可以預訂設備(場地租用)

可行性研究 (技術 IT/經濟$$$/機構-人)

在收集了用戶需求後，並於高級管理人員決定項目應否繼續進行之前，會先進行一個可行性研究，_____項目成功的機會率。 一個可行性研究應該陳列其他_____的方案，及推薦_____的方案。 此外，要識別項目涉及的_____，及在項目進行中該如何處理那些風險。 可行性研究的目標，是要評估系統開發的目的是否合理。

- ***技術可行性 IT***

  不但評估設計及開發系統所需的技術是否存在，而且估計在系統完成後，該機構是否有能力使用那技術。 技術上的可行性應從四個方面研究：

  – *科技的通曉能力*

    如果系統涉及一種該機構從未使用的科技，風險相應增加。

  – *應用系統的通曉能力*

    如果開發人員對該商業應用工作範疇不熟悉，開發人員很大機會曲解用戶的需求。
    如果用戶對應用系統不熟悉，可能導致更嚴重的問題發生。

  – *項目的規模*

    項目的規模可以理解為系統的功能數目，開發系統所需的時間，以及開發過程中所需要的人手等等。 項目的規模越大，項目的風險也就越大。

  – *兼容性*

    在一般情況下，一個新系統在實施之後，應與仍存在的技術或應用系統兼容。 如果

新系統未能融入現存的環境中，它對用戶機構的益處便有限了(e.g. ipad & flash)。但是，如新技術的採納很可能為公司帶來莫大的益處，則另當別論。

- ***經濟可行性 $$$***
  - 可以透過_____***分析***來研究，以測量一個項目所牽涉的財務風險。 詳細的經濟可行性分析，需要經濟學及金融學的知識，那是這課程範圍以外的領域。 基本上，經濟可行性是關於新系統所帶來的好處/效益，會否實質高於開發它所需的成本。
  - 開發成本(一次過的成本)及營運系統(循環不斷的成本)
  - 是對於開發成本(顧問費用、硬件及軟件上的投資)及營運成本(維修、軟件費用)的有形量度。 而成本可以是無形的，例如，工作流程的轉變，為人員帶來士氣上的衝擊。
  - 新系統所帶來的好處：***有形的好處***(節省人手及銷售額增加)及***無形的好處***(改善顧客服務及更高的產品質素)。 使用***財務分析***的概念及技巧 (例如是現金流、現淨值、收支平衡點及投資回報等)，來決定經濟可行性。
  - 項目不一定為公司帶來更多利潤。公司可能為免業務流失到競爭者手中，而決定推行項目。

- ***機構內的可行性*** (_____)

  *是*關於建議系統融入機構的持續業務中的順暢程度。 不像可量化的經濟可行性，機構內的可行性是不易量度的，但有兩個方法可用來執行評估：
  - *策略一致性* (e.g.開發者及用家)

    很多資訊科技項目是由機構內的資訊科技 IT 部門發起，但項目最終變成失敗。 這是因為由資訊科技人員發起的項目目的與其它部門的操作或目標 不一致。 項目的策略一致性越大，它的風險越低。 一個經濟及技術可行的項目，如果它不能符合機構的整體項目組合，可能被延遲或甚至不能推行，例如是項目組合內已有太多高風險的項目。 這種項目管理的方法稱為***組合管理***。
  - *利益相關者分析*

    是意圖量度利益相關者(例如是發起開發一個系統的高級管理人員，及系統用戶) 對支援開發及維修系統的準備程度。 支援可以是由高級管理人員，使用備忘錄的形式宣揚新系統的好處，或是鼓勵系統開發者與終端用戶有更密切的接觸，或是委派工作人員參與項目開發的隊伍等。

制定及評估其它方案

在進行透徹的可行性研究後，可能出現數個被建議的方案。 有時候，如果科技上有更多投資，項目在技術上會更可行，但經濟可行性有負面影響。 在某些情況下，技術及經濟可行性是有利的，但由於用戶不願採納改變，機構內的可行性便處於低位。 例如，對於八達通咭閱讀器的點名用途進行可行性研究後，發現很多教師也不喜歡這個意見，因為學生可使用他人的八達通咭，代其他人點名。 另一方案是使用手提個人數碼助理 PDA，然後透過無線局部區域網絡 LAN，把數據傳送到校務處。 在不同的方案上選出項目，常常涉及妥協。

客戶需求製模 **modeling** (把抽象概念→實質化/圖像化，表現出來)

一個信息系統的功能面的特點，是表現於系統保存的數據及處理那些數據的程序。 那些數據及程序特徵，可以透過數據製模及程序製模工具來表達。 大部分工具是可制作圖表的，除了文本外，以製模工具記錄需求的主要原因是避免不清晰。 其中一個最常用的數據製模工具大概是***實體關係圖*** ***(entity relationship diagram***，ERD)。
由於 ERD 概念已在「數據庫」的課題中描述，這裡我們只集中討論程序模型。
數據流程圖 (data flow diagram，DFD)是用來說明*程序製模*工具的重要部分。 請注意，數據製模及程序製模，除了可應用在新系統上，也可應用在現有系統上。 現存系統的模型可幫助系統開發員，更精確地了解現存系統的工作。

- ***用數據流程圖*** (data flow diagram，DFD)***作程序製模***
  程序模型是表示一個系統怎樣操作的正規方法。 它描述(一個商業系統的)程序或活動怎樣運作，及數據在它們之間怎樣移動，以達到系統的目標。 數據流程圖內使用四個符號。 它們是：
  - *處理程序* ：把輸入數據流程轉換成輸出 (子程式)
  - *數據流程* → ：顯示數據的移動(例：數據的輸入及輸出)
  - *數據儲存器* ：是數據的儲存(例：檔案 file 或數據庫表格 table)
  - *外部實體* ：是系統範圍以外的來源或終點 (例如：與系統互動的系統用戶，訂單，產品，顧客，...)

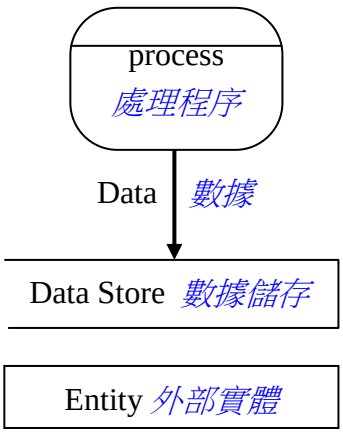
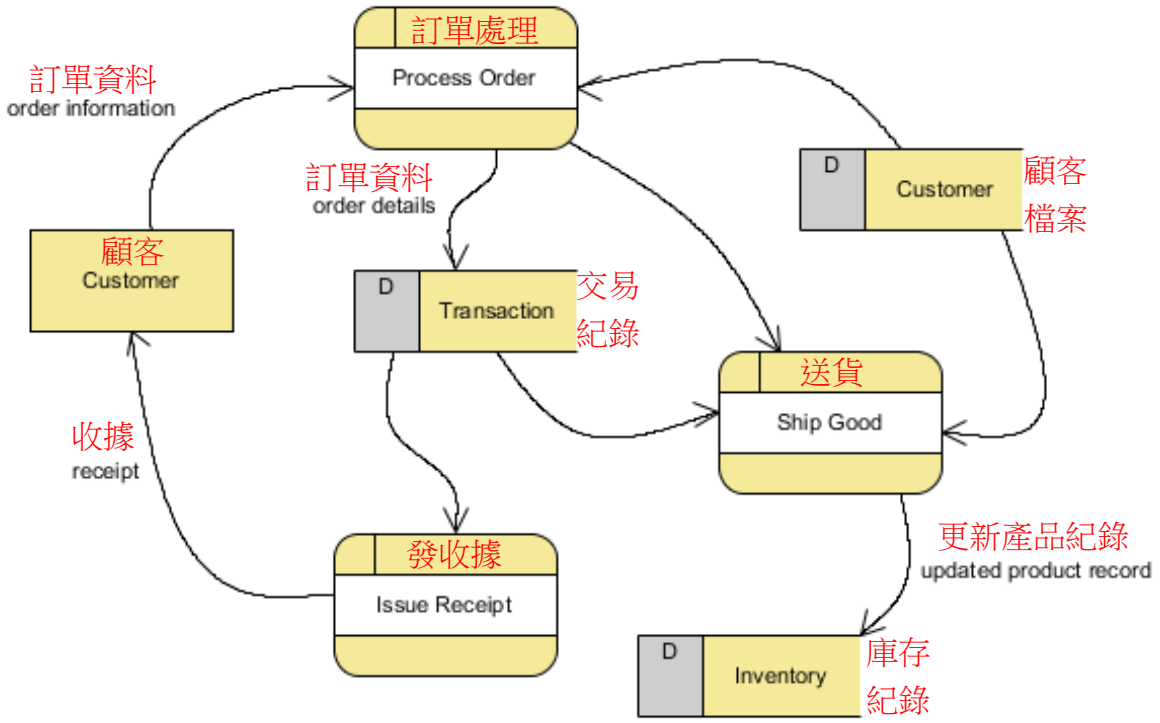


圖 1 數據流程圖標記法。

圖 1 的數據流程圖以 Gane 及 Sarson 形式的符號表示。
而另一個常用的形式稱為 DeMarco 及 Yourdon 標記法。



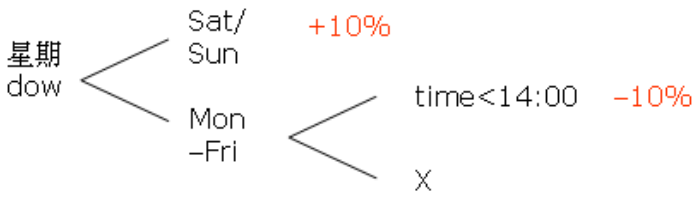
在單一個數據流程圖 DFD 上顯示商業程序是非常複雜的。 數據流程圖在一個階組內描述系統處理方面的不同細緻程度，以支援不同類型的用戶。 低層次的數據流程圖，是相對的高層次數據流程圖的一部分之詳細描述。 例如，一個大學圖書館管理員，有興趣知道整個圖書館信息系統能夠提供的關鍵功能，而非那些功能是怎樣實現的。 因此他/她只對高層次的數據流程圖有興趣。 而借閱櫃台的職員，只需要知道他們利用借閱子系統所需做的工作，因此，關於借閱子系統操作的低層次數據流程圖，便切合他們的需要。

子數據流程圖更詳盡地顯示母數據流程圖的一部分。 數據流程圖的最高層次是*背景圖*，它只描述單一個程序(整個系統)。 數據流程圖的每個程序可擴展為較低層次的數據流程圖，其中可有數個程序、數據流程及或許也有(內部的)數據儲存。 具體而言，由背景圖內的程序擴展的數據流程圖被指為*層次 0 圖*。 由層次 0 圖擴展出來的數據流程圖稱為*層次 1 圖*。 由層次 *n* 圖擴展出來的數據流程圖稱為*層次(n+1)圖*。 因此，描述一系列的數據流程圖只有一個背景圖及一個層次 0 圖，及多個層次 *n* 圖，而 $n \geq 1$。

教學附註

- 有多個規則規定怎樣保持一系列的數據流程圖的一致性。 那些規則的簡介及一些簡單的例子，可在這個數據流程圖網頁內找到。 但是，讀者請留意，網頁命名層次 *n* 圖(當 $n \geq 1$)的方法，有別於一般常用的方法。 www.umsl.edu/~sauter/analysis/dfd/dfd_intro.html

程序描述，通常是文字的，對每個最低層的程序的描述是需要的(也被稱為*功能性基本指令*)。 它提供相聯的功能性基本指令的程序邏輯的詳情。 如果程序本身的邏輯較複雜，程序詳情可以*判定樹* *decision tree* 或*判定表* *table* 的形式描述，以避免不清晰。例



1. 若在週末(星期六、日)進入，收費會加10%
2. 若在平日(星期一至五)進入，而進入時間為14:00或以前，收費會減30%

	time<14:00	time>=14:00
Sat/Sun	Fee	Fee
Mon-Fri	Fee	Fee

Game of Life

neighbours=	0-1	2	3	4-8
cells[i][j]=0				
cells[i][j]=1				

cells[i][j]附近若有
0-1鄰居：死亡
2-3鄰居：仍然生存
=3鄰居：新生命誕生
4-8鄰居：死亡

- ***數據字典*** data dictionary

數據資源庫描述的數據元素(如數據流程，及儲存的數據)，稱為*數據字典*。 數據字典是一系列元數據_____、數據_____及數據的_____法。 所有在程序模型內使用的數據，必須在數據字典內描述。 建立數據字典的原因是維持檔案及表格的數據之間的一致性。 舉例說，數個檔案內可能也有學生名字及身份證號碼，它們可能有

不同的名稱或甚至不同的格式，數據字典的使用，可維持名字及身份證號碼欄的一致性。一個好的數據字典，能加速電腦化信息系統的設計、實施及維修。

數據字典由軟件系統使用的數據欄的描述組成，包括欄的意義、有效性規則及組成規則等。 數據字典使用以下的符號描述數據欄的組合：

= 等於 + 及 and [] 兩者選一 [L|M|C|S] or () 選擇性輸入 optional	例如，「學生」這個數據可以下列組合代表： Student = Name + Student ID + Date of Birth + Sex + (Age)
Name char(__) Student ID int (__) Date of Birth date Sex char(__) Age int	=*任何在 50 個字符內的字符串* =取錄年份 + *任何六位數字* =*任何(<=today)有效日期* (YYYYMMDD 的形式) =[M | F] =任何整數，由今日及出生日期，計算出來的年齡

項目規劃 project management

為了項目可在限期內完成，在項目計劃時須設定里程碑及相對的限期，以促進項目管理。 在每個相應里程碑可完成的事應明確地定義。***關鍵路徑方法*(critical path method**，CPM)是一個項目管理工具，幫助項目經理在項目進行期間監察項目進度。

一個龐大的項目，在系統開發的每個階段中，應分拆成細小的任務 task。 在每個項目中，均有一些不可延遲的_____任務，否則整個項目可因此被延遲。 而對於項目其它的非關鍵任務，可容許延遲一定的時間，但這取決於它與其它任務的依賴關係。 關鍵路徑方法幫助項目經理分辨那個任務是關鍵性的，即是：在關鍵路徑內的元素、及識別每個非關鍵任務，在不影響整體項目的完成時間下，可接受最長延遲的時限。

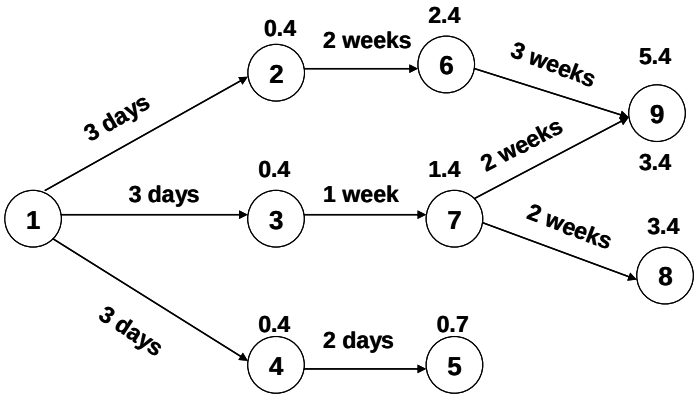
關鍵路徑方法的步驟如下：

1. 列出所有任務及其開始的最早時間、預計所需的時間及任何之前的任務。例子：

任務	最早開始	所需時間	之前的任務
1. 與用戶面談	週 1	3 天	
2. 系統分析	週 1	2 星期	1
3. 用戶界面設計	週 1	1 星期	1
4. 硬件選取	週 1	2 天	1
5. 硬件安裝	週 2	1 星期	4
6. 程式編寫(系統邏輯)	週 3	3 星期	2
7. 程式編寫(用戶界面)	週 2	2 星期	3
8. 用戶培訓	週 4	2 星期	7
9. 系統測試	週 3	4 星期	6 及 7
10. 文件編製	週 5	3 星期	

2. 以圓形及箭咀表示任務。例子：
描述任務 3(用戶界面設計)需要 1 星期的時間完成，及它需要在任務 7(程式編寫(用戶界面))開始之前完成。

相應以上例子的關鍵路徑圖顯示如下。

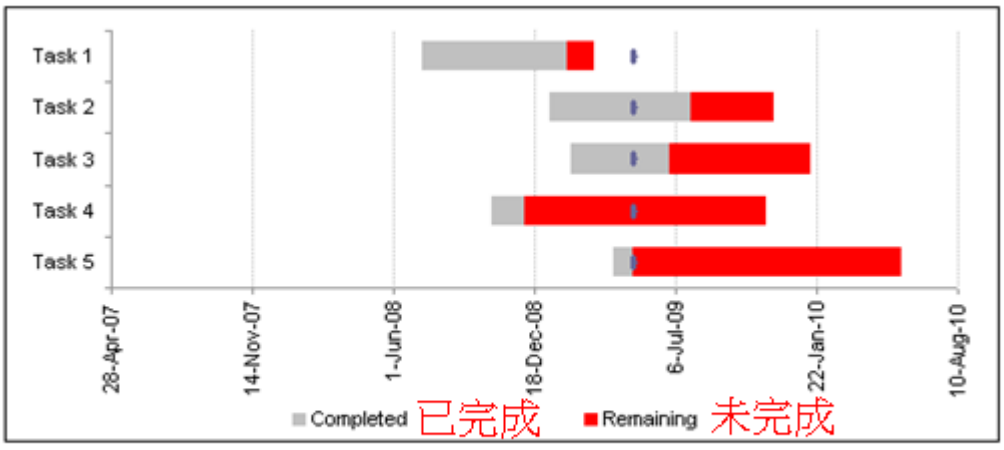


請留意，在每個圓形之上的數字，是相應任務可開始的最早時間(以星期計算)。 例如，在路徑 1-2-6(需 5 星期及 3 天完成)及路徑 1-3-7(需 3 星期及 3 天完成)上的所有任務，需在任務 9 開始之前完成。 因此，任務 9 的最早開始時間為 5 星期及 3 天(約 5.4 星期)。 如果項目經理想盡早開始任務 9，他必須加快完成任務 2 或任務 6。

一般做法，一個不需時間完成的人造任務，可被加到以上的圖中，令圖中任何路徑的最後任務，即任務 5，8 及 9，可連接到此任務。 這使項目經理能計算出項目所需的時間。

關鍵路徑方法的主要優點是它幫助計算項目_____，及監察項目_____。
如果任務的完成時間不同於計畫的時間，相應的關鍵路徑圖將會被更新，以反映與原計劃的時間偏差將怎樣影響整體項目進程。 有了關鍵路徑方法，關鍵任務可被識別出來，在項目進度落後計劃的進程時，項目經理可以作出資源調配。

由於在項目進行期間，維持一個關鍵路徑圖可以是繁複的，軟件工具如 Microsoft Project 是設計來解決這個問題。



項目管理：甘特圖 GANTT Chart

教學附註

- 關鍵路徑方法不是識別關鍵路徑的唯一工具。 Gantt 圖表是另一個工具。 而程式評估及覆核技巧(Program Evaluation and Review Technique)，PERT，是關鍵路徑方法的變型。
- 這網站提供另一個例子說明關鍵路徑方法的概念。 它顯示 Excel 可怎樣計算關鍵路徑方法(參閱例子 1)。 此外，那裡也說明關鍵路徑方法的一個變型，它考慮項目的成本問題(參閱例子 2)。 另一個關鍵路徑方法的網站也值得觀看。 那裡總結了關鍵路徑方法及程式評估及覆核技巧(PERT)的不同之處。

2.系統設計

每個功能性需求必須由某些_____實現。 分析員界定功能需求，並以文件記錄每個程式的功能。 請注意，實現程式的功能需要透過適當的_____及_____。

例如，要描述一個學生的個人資料，以 SQL/C 語言編碼的相應數據結構可以是如下：

```
create table student (
    studID  int(6),
    name    char(20),
    dob     date
)
```

```
struct emp{         // 顧員資料紀錄
    char name[20];// 姓名
    int    age;     // 年齡
    float  salary;  // 薪金
};
struct emp e;  // an employee record
```

分析員可以如下的*偽代碼* pseudo code *制定算法 algorithm*：

```
if (noOfRenew<=3) then
    perform the renewal and
    increases noOfRenew by 1
otherwise
    display message " renewal rejected "
```

```
_____ (續借次數<=3)_
    進行續借
    增加續借次數

_____
    顯示信息 "錯誤：續借被拒"
```

實際的編碼 programming，留待程式編寫員在實施的階段才進行。

用戶界面(版面)user interface (使用者界面/人機界面)

定義一個_____與_____間的互動，及系統接受、產生的輸入和輸出的性質。
如果用戶界面設計差劣，將影響系統的可用性及用戶對系統的接受。 因此，不能輕視。

例如，由 WebSAMS 時間表模組產生主時間表，所支援的用戶調較網頁界面設計，界面程式只允許用戶在同一時間內觀看一個時間表，但是，要編排時間表，最少要同時觀看三個時間表，分別是班別、班房及教師的時間表。無論時間表系統功能怎樣強大，不善的用戶界面設計，會嚴重影響時間表編排系統的可用性。(另一失敗例子: SBA 2010 簡單複雜化)

用戶界面的設計(版面設計)，要考慮六個原則。

- *屏幕，表格及報告的*_____
  是指屏幕及文件怎樣組織，以致可用及一致。
- *內容關注*
  是指用戶界面輕易把用戶的_____力，吸引至特定內容的能力。
- *美學標準*
  是指用戶界面設計，如何令人感覺愉快，例如使用較_____的顏色。
- *用戶經驗*(習慣)
  是指把用戶的經驗用作設計用戶界面之參考。 例如，有經驗的用戶較喜歡鍵盤的輸入界面，因它支援有效率的數據輸入，而新手用戶則較喜歡滑鼠推動的界面程式。
- *一致性*
  這是一個重要的因素，令系統容易使用，及允許用戶可預見將發生的事情。 例如，Microsoft Office 的所有應用產品，均在屏幕的上方設有選單欄。 統一的圖示，如「儲存」 (🖫)，均在所有的應用軟件內使用。(劃一設計、容易上手)
- 容易操作
  是指_____設計，使用戶容易操作，例如是最少的滑鼠點擊次數。

用戶界面的種類

用戶界面有三個主要的種類：

- *指令推動界面 Command Line Interface (CLI)*
  它透過文本與系統互動。 指令是由鍵盤輸入的一系列文本。 系統的反應也以文本形式顯示。 最常見的指令推動界面大概是 MS-DOS 指令界面。 下圖顯示複製一個檔案的指令。 如果用戶不是專家，使用指令推動界面 CLI 將是_____的，因為用戶需記著所有指令，而且界面並不容易使用。 指令推動界面原本是_____及 Linux 系統設計的一種用戶界面。

- *選單推動界面 Menu Driven*
  顯示一系列的選擇，用戶從中選取其中的一個。
  較指令推動界面更易使用，因為它不需用戶記下所有_____。
  在設計選單推動界面時，必須注意選單項目是根據邏輯歸類。
  下圖顯示一個選單推動界面的設計初稿，圖書館借閱櫃台的職員，可輸入數值 1、2、3 或 4 以執行相應的功能：

  圖書館借閱系統
  1. 借出
  2. 還書
  3. 書籍續借
  4. 離開

  選擇： __

  選單項目常會被有系統組織起來，因而容易使用。 以上例子列出的選單項目，滿足了群組的兩個條件 ： _____的及依_____頻率排列。 例如，把

「書籍續借」放在選單項目的開頭是有些奇怪的。 雖然，文本模式的選單推動界面，時至今日，已不再被廣泛應用，但是，它的設計概念被多數的圖形用戶界面採納。

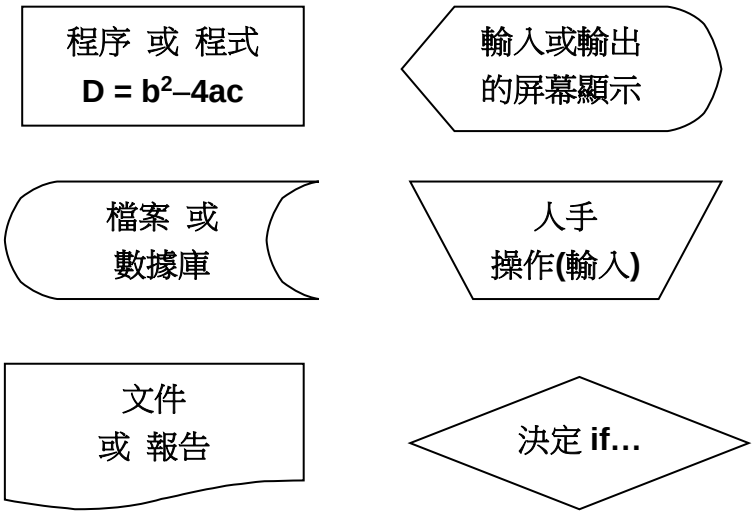
- *圖形用戶界面 Graphical user interface (GUI 看圖辨識)*
  GUI 透過圖形圖示與用戶互動。 舉例說，印表機圖示(🖷)用來代表列印行動。 現在，圖形用戶界面是十分流行的。 它允許用戶除了可使用鍵盤執行輸入任務外，還可使用指標器，如滑鼠或觸式屏幕。圖形用戶界面的特點可以用首字母縮略字_____概括，它代表 *W*indows(視窗)、*I*cons(圖示)、*M*enus(選單)及 *P*ointing device(指標器)。

🛠 活動
- 縮放用戶界面(ZUI)是圖形用戶界面(GUI)的進化發展。 在網上查找甚麼是 ZUI。
- 要體驗立體圖形用戶界面可提供的東西，觀看 http://lgscope.hp.infoseek.co.jp/lgamazon/videoA.wmv 的影像短片。
  而關於立體圖形用戶界面的詳情，請到訪 http://www.sgi.com/fun/freeware/3d_navigator.html 及 http://www.sun.com/software/looking_glass/的網站。

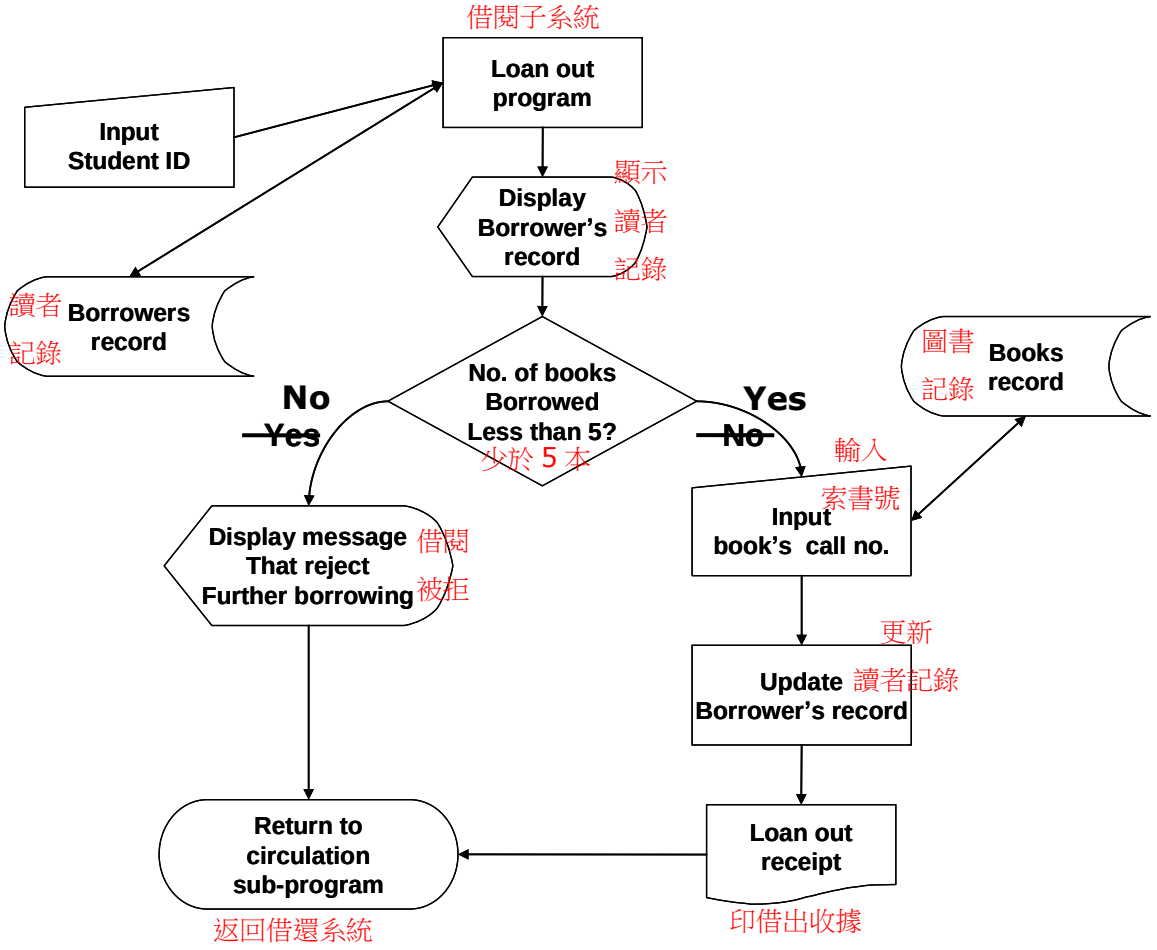
系統流程圖及結構圖

程式設計的傳統方法，是利用結構圖把分析結果(如數據流程圖)轉變為設計模型。
系統流程圖也被廣泛使用以展示系統怎樣操作的概覽。 它代表文件的流程，
及把_____轉變成_____的_____，即電腦程式或人手操作的步驟。

- *系統流程圖 System Flow Chart*
  以圖型代表組成系統的不同程式、檔案、數據庫及人手程序。 像程式流程圖般，符號被標記及連接起來。 以下是一些常用的系統流程圖符號。



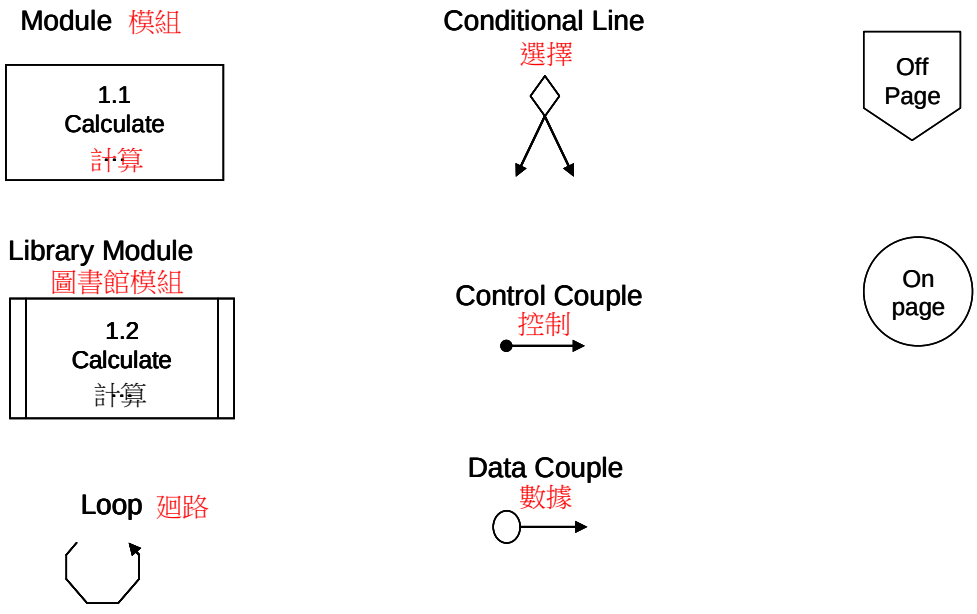
以下的系統流程圖，是學校圖書館系統的借閱子系統，其中借出步驟的簡化版本。



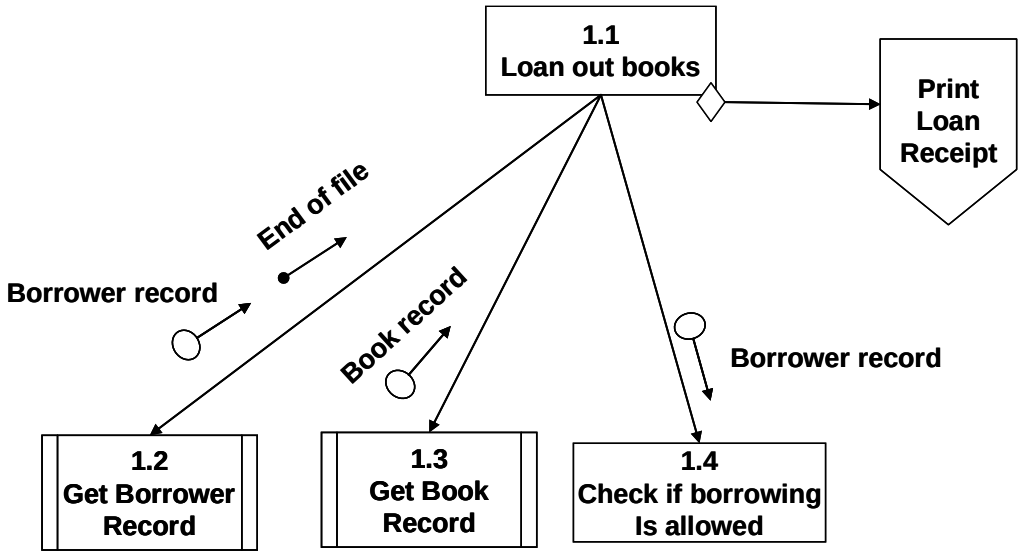
- 結構圖 structure chart

描述系統的結構性關係，及函數與子函數交換的信息。 它以階層的次序顯示程式模組之間的關係。 結構圖有三個組件：*模組*、模組之間的*連接*及模組之間的*數據溝通*。 常用的結構圖元素顯示如下：

http://www.ablmcc.edu.hk/~scy/home/javascript/images/structure-chart.png



以下是書籍借出模組的簡化結構圖：



教學附註

- 結構圖現今已甚少被使用，因它以*結構性設計法*用於系統開發。 結構性設計法強調功能性分解，它以逐步求精法分拆複雜的系統，變成很多單一功能的任務，從而達到有效率的實施。 但是，近年的系統開發大多採用物件導向法。
- 評估結構性設計質素的兩個重要概念是*耦合(coupling)*及*結合(cohesion)*。 這些概念是在高級程度電腦科的課程範圍以外，但是，有興趣的讀者可閱覽這個網上資源，參考這些概念的精簡介紹。

3.系統實施 implementation

當設計階段完成，系統設計便可實施。 實施階段最為人所知的元素是_____。
編碼是系統設計的抽象模型的認知與實現。實施的另一個重要元素是_____。

為什麼需要測試?

如果程式編寫被視為寫作程序，測試便是編輯程序。 一個作家甚少在沒有編輯的情況下出版他的著作。 同樣，一個新開發的軟件系統，不應在有錯誤的情況下發放給用戶。 對功能性及非功能性的系統需求，兩者的滿意度的徹底測試是必須的，並須_____錯誤及_____之處。

測試的原理是比較_____結果與_____結果。 若測試結果符合預期，測試便通過，否則，它被視為失敗。
請注意，測試只可_____錯誤，它不能_____錯誤不存在。

測試不只在軟件系統編碼後才進行，而應在開發的不同_____執行，
因為越早_____錯誤，修復它們的_____越低。

測試的種類

為確保系統的質素，必須有最少三種的測試。

- ***單元測試*** unit test

  目標是確保每個_____如規格定義般執行功能。 它不會揭示性能問題或任何系統性的錯誤。 進行單位測試的測試員通常是系統分析員或開發模組的程式編寫員。 單位測試的兩個常見方案是：

  – *白盒測試* ***white-box***

  執行白盒測試的測試員，必須可全面接達模組的_____，及具有執行某功能的程式碼的相關知識。 例如，我們需要準備的測試實例，須確保模組內的每個_____，最少被執行一次。 而分枝覆蓋測試需要確認沒有程式碼的分枝，可導致應用出現不正常行為。 測試員有責任確保模組的組件的功能是可靠及完整。 一般而言，白盒測試只用於複雜的模組。 為精簡課題，不在此討論其它的白盒測試策略。

  – *黑盒測試* ***black-box***

  在黑盒測試中，測試員只可透過模組的界面與它接達，而不用理解實際的來源碼。 測試員建立 測試_____test cases，檢查模組是否符合規格內定明的_____。 兩個常用的黑盒測試策略是

  - 範圍測試 – 測試在指定輸入範圍 range(月份)的數據，檢查程式的穩定性。
  - 邊界測試 – 測試在指定輸入範圍的上限及下限 boundary 的數值，保證可接受的輸入上限及下限，產生正確的輸出。

  黑盒測試一般可由任何瞭解測試性質的人(用戶)執行。 有關黑盒測試的更全面討論，包括測試策略，可參閱由 Ball State University 所做的演示簡報片。

- ***系統測試*** system test

  是評估整個系統是否符合特定的要求，測試是在統合個別單位或子系統後進行。 系統測試的重點不在_____，而是在測試更廣泛的範圍，包括系統在大負載下的可用性、_____及性能表現。 在系統測試期間，可進行很多不同類型的測試，例如是需求測試、可用性測試、保安測試、性能測試及文件編製測試等。 系統測試通常由系統分析員以_____的取向進行，因為它是系統傳送至用戶手中作接受測試前的最後階段測試。 某些研究指出，大多數的軟件錯誤會在系統測試中被發現。

- ***接受(驗收)測試*** acceptance test

  主要由用戶進行的測試，它確保系統符合_____需求。 用戶使用黑盒方法檢查系統(因用戶不須知道系統的內部運作)。 當接受度測試成功完成後，負責項目的顧客為系統的接受度署名(驗收)。 接受度測試可分兩階段執行：

  – *Alpha 測試*

  在 alpha 測試中，通常使用經_____的測試數據。

  – *Beta 測試*

  在通過 alpha 測試後，系統被稱為 beta 版本(測試版本)，而且可進行 beta 測試。 在 beta 測試中，_____數據會被用來測試系統，使錯誤的數目減至最少。

測試計畫 Test Plan

Wikipedia 對*測試實例*給予以下的定義。

> *測試員以一系列的測試實例 test cases 為基礎，決定一個應用軟件的需求是否部分或完全被滿足。 ...每個需求必須有至少一個測試實例 ...*
>
> *(摘自 Wikipedia)*

一般錯誤的概念，認為一系列的測試實例便是一個測試計畫。
事實上，一個測試計劃是對系統的一系列完整的測試。
每個測試，均有明確目標，而且包括一系列特定的測試實例，以檢驗及定義預期的結果。

例如，如規格定明，一個讀者可借出的書籍數目是：0 至 5，測試員應開發最少__個測試實例：其中三個實例為有效的輸入，包括可接受範圍的周邊數值_____及_____，及在它們中間的一個數值；其餘兩個實例為非法的輸入，它們應是僅僅超越可接受範圍的數值，即是_____及_____。
當然，也可執行數據類型檢驗 (A-Z/0-9)，以確保輸入是一個數值。 測試實例並不限於數據輸入，有時候，系統對指令或事件的組合的反應，也必須被測試。 下圖顯示一個用來規劃測試計畫的簡化表格。

測試計畫

程式編號：_____ **版本號碼：**_____
測試員 ： _____ **進行日期：**_____
結 果： ❑ **通過** ❑ **失敗 (詳情)：**_____
_____
測試編號：_____
針對的用戶需求：_____
目標：_____
_____

測試實例

	界面編號	數據欄	輸入值	預期結果	實際結果
1.	_____	_____	_____	_____	_____
2.	_____	_____	_____	_____	_____
3.	_____	_____	_____	_____	_____
4.	_____	_____	_____	_____	_____
5.	_____	_____	_____	_____	_____

手稿
_____
_____

教學附註

- 更詳盡的測試描述，可在 http://en.wikipedia.org/wiki/Test_plan 取得。
- 美國電機及電子工程師學會定明 **829 軟件測試的標準文獻(IEEE 829-1998)**可在 http://www.ruleworks.co.uk/testguide/IEEE-std-829-1998.htm 取得。

🛠 *活動*

- 列出用作測試一個計算數字(n)的平方根的函數 sqrt(n)之測試值。
- 討論即使當所有組成模組通過單位測試，為什麼仍需要系統(統合)測試。

4.系統轉換及維修 conversion & maintenance

在系統實施及測試後，使用適當的*系統轉換*策略使系統投入運作。 以下是四個常用的策略：

直接轉換 direct cutover

直接轉換的策略，是使舊系統遷移到新系統的過程在很短的時間內完成。 如其名，在一特定的日子，關閉舊系統而同時使新系統開始運作。 為確保順利及成功的遷移，在直接轉換前必須完成徹底_____。 直接轉換被視為_____風險的策略，有以下的優點及缺點。

優點：

- 實施是簡單和直接。
- 與其它轉換策略比較，_____最低。
- 用戶不會感到混亂，因為在任何時間內，只有一個系統在運作。

缺點：

- 用戶必需快速_____新系統，因為無法轉回舊系統。(只許成功，不許失敗)
- 任何錯誤的出現，可能帶來系統的_____，而某些任務可能沒有其它替代方案。
- 沒有新、舊系統的並行運作，沒法_____兩個系統的運作結果，較難發現新系統的潛在錯誤。

平行轉換 parallel

主要是解決因舊系統突然終止而產生的缺點。 轉換以漸進的方式進行，新舊系統在一段時間內_____操作。 數據輸入兩個系統，交叉校驗新舊系統的輸出。若新舊系統特性相近，平行轉換可有最佳的結果，特別是以電腦系統取代人手操作的系統。 平行轉換的_____較低，因為如新系統失敗，可允許商業操作返回舊系統。

優點:

- 可以校驗/比較新舊系統運作的結果，容易發現新系統的_____。
- 當新系統不能正常操作時，舊系統可作為_____。 這可給予用戶安全感。

缺點:

- 在某時段內讓兩個系統_____，帶來額外的費用。
- 用戶的_____量實際上倍增，因為每個任務均需執行兩次，增加用戶負擔。
- 兩個系統的同時運作，容易產生混亂。(在舊系統輸入了...以為在新系統也輸入了...)
- 用戶通常對熟悉的系統感覺較好，因此可能用上(較實際需要)更長時間去接受新系統。

分階段轉換 phased (模組 A,B,C,D,E,...)

結合「直接轉換」及「平行轉換」的優點，並減少風險及缺點。 在這個策略中，新系統的模組逐一被使用。 當新的模組運作良好，舊系統的相應(相容)模組的運作被結束。 所以，在整體上它的運作如平行轉換般，但在每個階段或時期，個別模組的轉換是直接的。 請注意，以階段轉換小型及非複雜的系統的成本校益並不顯著。 階段轉換有下列的優點及缺點。

優點:

- 減少新系統出錯所帶來的_____影響，因為系統是以模組為單元逐一使用的。
- 任何錯誤的出現，不需維持長久的_____，錯誤便可被偵測及修正。
- 成本____遠_____於平行轉換，因為用戶不需重複每個任務(新舊系統各做一次)。

缺點:

- 可能需要很_____時間，使新系統安裝完成及運作順暢。
- 在軟件開發項目裡若有新功能，新舊模組較難_____。
- 轉換階段中一旦出現錯誤，會令用戶發現對系統整體有不良的印象/信任(不滿/抗拒/反感)。

實驗轉換 pilot(先導/試點)

是指新系統在不同地點安裝的情況。 例如，相同的銀行系統必須在不同分行內使用。 這個策略是在一個(或數個)場地安裝整個新系統，作為實驗測試的一部分，所以有實驗轉換的命名，而其它的場地仍以舊系統運作。 當轉換在實驗場地實施成功，其它場地便可進行遷移。但是，機構須接受在某段時間內，在不同場地出現不一致性，才可採用實驗轉換。

優點:	*缺點*:
● 錯誤只發生在實驗場地，不會對整個機構造成有害影響。 ● 其它場地可從實驗場地的經驗中學習，尤其是操作方面的經驗。	● 即使在實驗場地成功進行轉換，這並不表示其它場地的安裝也是順利的，因為每個場地可能有本身的特質。 ● 在所有場地安裝新系統，需要較長的時間。

評估轉換策略的選擇

表面看來，沒有特定的轉換策略比其它的優勝(各有所長)。 有時候，需要使用_____的策略， 例如，整個系統的實驗轉換可與實驗場地的平行轉換混合實施。

選擇轉換策略時，除了它所帶來的好處外，還要考慮三個因素：____

無論是使用那個轉換策略或混合的轉換策略，也應制定應變計畫。

🛠 活動

- 繪畫一個圖表，比較以上的四個轉換策略的風險、成本及所需時間。
  使用*低*、*中*、*高*、*短*及*長*等值作比較。
- 在以下的每個系統轉換的個案中，考慮風險、成本、好處及所需時間，
  選擇一個合適的轉換策略，或策略的組合，並且解釋你的選擇。
  - 學校圖書館系統原本是人手操作的，它將被電腦系統取代。
  - 一個以電腦為本的學校圖書館系統，將轉換成一個網上系統，
    令用戶可透過互聯網，在圖書館的非開放時間接達系統。
  - 在一間診所使用的電腦系統，有數個模組：預約模組、病人記錄模組及會計模組。
    整個系統將提升至較容易使用的界面，以信用卡付款的新功能會加到會計模組中。
  - 新的銷售點(POS)系統，將安裝到連鎖超級市場的分店中。

系統維修 maintenance

是系統運作成功後，對系統的持續_____。 除了_____錯誤外，系統_____與_____，也是維修的主要工作內容。

系統維修包括七項的主要活動：

- ***修正錯誤***
  無論系統設計、實施及測試是怎樣好，系統仍有機會存在錯誤。 原因通常是：
  - 用戶需求的溝通問題/誤差
  - 設計缺點
  - 出現非預期的情況 (沒有被測試)
  - 用家沒有正確使用系統

  錯誤修正是第一優先，因它對整個系統可引起嚴重問題，並且動搖用戶對系統的信心。
- ***系統性事故後的復原*** (recovery)
  系統性事故可以是經常發生的，原因可能是一個被棄置的程式、數據損失或硬件損壞。
  因此，應提供系統復原的支援及防止系統性事故的的再度發生。
- ***系統改善***
  當系統開始運作，用戶常常要求系統更完善或需要更多的功能，要求不同程度的修改。
- ***系統統合***
  不同的系統可能個別被開發， 但機構有可能想統合系統以精簡運作。
- ***基本軟件或網絡的改變***

新的操作系統 OS 及數據庫管理系統 DBMS (及甚至是硬件)，
可能對現存的應用系統引起問題。 因此，也有需要對應用系統作出改變。

- *協助用戶*
  對終端用戶的支援也是一種系統維修工作。 無論用戶有怎樣良好的培訓，或文件編製是怎樣完善，用戶仍有可能尋求協助。 對用戶提供協助的最常見方法是：
  - 在系統內以內置說明屏幕形式出現，網上支援，或把常見問題(FAQs)貼在網上。
  - 負責支援的人員透過電話或親身解答用戶的問題。
    求助台 help desk 可在 2 個層面上操作：
    - 層面 1 的支援職員，擁有廣泛電腦技術，可回覆不同範疇的求助、一般查詢。
    - 層面 2 的支援職員，解決層面 1 的支援職員不能處理的求助。 層面 2 的支援職員必須對部分或整個系統有較深入的知識，可提供專業的意見。
- *系統重組*
  因運作或商業模式的一些改變，機構可能需要改寫系統。 重要的改變通常被當作個別的項目。

教學附註

- 系統維修的經典例子是所謂的「千年蟲」問題。 在六十及七十年代的早期，由於儲存成本昂貴，所以年份的首兩個數字不會被儲存，例如，1993 年 3 月 15 日會以 YYMMDD(年月日)的形式被儲存為 930315。 同樣，2000 年 4 月 24 日會被儲存為 000424。 當 930315 及 000424 這兩個日期被使用及運算，可產生荒謬的結果，例如是計算一個人的年齡(2000−1993=7)可產生的錯誤計算(00−93=-93)；計算過期還書、食品...等。在接近2000的九十年代末期，很多公司為解決這個問題而花了很多錢，系統的修正目的，是使系統時鐘在跳至 2000 年 1 月 1 日時能符合公元二千年數位標準。 有關「千年蟲」的詳情，可在 Coping with the Year 2000 Rollover, *Software Development*. 4, no.8 (August 1996), 23-24, 26, 28. Keuffel, Warren.內找到。

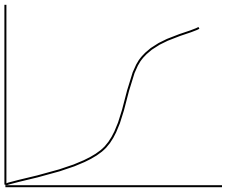
用戶培訓

用戶培訓是十分重要的，因為如用戶不懂操作，一個系統便沒有價值及_____。

- 培訓目標
  - 培訓應集中於幫助用戶完成他們的任務，而不只是了解系統。 培訓必須集中圍繞系統的活動及系統本身。 培訓後，用戶應能夠了解系統怎樣配合他們的____。
  - 培訓應集中於用戶需要做的事，而不是系統能夠做的。 大多數系統均提供遠超過用戶需要的功能。 所以，培訓的重點應是用戶定期_____的任務。
- 培訓方法
  - 培訓方法有多種。 最常見的方法是課堂培訓，由一個指導員同一時間培訓多個用戶。 如果財政上可行及有特別的需要，也可安排一對一的培訓，由一個指導員在特定的時間內培訓一個用戶。
  - 由於資訊科技(IT)的廣泛使用，_____培訓，也越來越普及。 教材可以是鐳射光

碟內或網上的文本、圖像、聲音、影像及動畫等。 用戶也可透過電腦培訓提供的互動_____，在閱覽教材後，檢測是否明白內容。 電腦培訓的開發通常是_____的，但用作廣泛培訓卻十分便宜。

5.系統文件編製 System Doc



為什麼需要編製系統文件?

當一個系統在運作，便需要_____。 在信息服務行業裡，常有人才_____。 一般，系統開發與維修是不同組別的人員，因此，維修人員需要花時間_____系統及克服學習曲線的問題。 書寫良好及定期更新的文件編製，肯定可縮短新的工作人員學習系統的時間。

文件編製，對不同的人有不同的意義。 通常人們認為，文件編製只是描述系統及它的組成部分的_____之文本。 但事實上，分析員在系統開發時，積累了大量系統的信息，例如屏幕格式/設計、數據庫及檔案設定等，成為編製文件內容的一部分。

編製系統文件是持續不斷的過程

考試要取得成功，穩定及持續的溫習，總比臨急抱佛腳有效，成功的文件編製也是一樣。因此，文件編製應在系統開發生命周期的不同階段中_____進行，而不是留待項目的完結才開始。 如要產生高質素的編製文件，持續過程是必須的。

不幸地，很多系統，不是沒有編製文件，便是所編製文件不足，分析員總是不願投資時間在書寫文件上，或是他們缺乏良好的書寫技巧，及用上不必要(不易理解) 的技術性詞彙。

請注意，良好的編製系統文件可：

- 使_____容易及有助系統的修改；
- 為系統提供永久的_____，即使開發系統的人員離開，也不影響系統的維修；
- 為系統的所有利益相關者，提供良好的溝通渠道；及
- 減少培訓的_____。

系統文件的種類

一般來說，文件編製可分成 3 個主要的種類：

- *系統文件編製 System Doc*
  系統文件編製應以較廣的層面看系統。 包括幫助分析員及程式編寫員，理解一個系統是怎樣建造的信息。 詳細的信息，如用戶_____、設計_____及_____等，主要在分析及設計的階段記錄下來，它們形成系統文件編製的基本部分。 當系統需要_____時，系統文件編製對展現項目便十分有用。 下圖顯示了系統文件編製的簡單概括。

I 系統分析	**II** 系統設計	**III** 系統實施
1. 用戶需求 2. 問題描述 3. 可行性研究 **...**	1. 屏幕及報告格式 2. 數據字典 3. 數據庫模式及檔案格式 4. 程式規格 **...**	1. 測試計劃、測試數據 2. 轉換計劃 **...**

- *技術性文件編製 Technical Doc*
  主要是一系列_____、_____、_____及應用協定界面等的描述。 事實上，一些電腦輔助軟件工程工具，可編輯及產生簡單的技術性文件編製。如系統需要_____，程式編寫員很倚靠技術性文件編製，對系統的組件作出改變。

- *用戶文件編製 User Doc*
  如用戶手冊、培訓手冊及網上說明系統，旨於在_____的層面上幫助用戶，即怎樣_____系統。 用戶文件編製可組織成三個主要的類型：
  - *輔導*指引：引導新用戶透過每個步驟完成工作。
  - *程序手冊*：描述怎樣執行特定工作(例:更新記錄)，作為中級用戶的一般指引。
  - *參考文件*：用戶的最後憑藉，當用戶欲啟動一項功能，並在參考其它兩種用戶文件編製後仍未能成功，參考文件便是用戶最終的資訊工具，因此，*參考文件*須蓋括相應系統功能的所有詳細資料。

在線 on-line 系統文件逐漸流行及重要，用戶不需離開系統，即時可接達說明內容。
但是，傳統的硬本手冊仍然十分重要，因硬本手冊容易翻閱及取得文件大概的結構。

教學附註

- T. Barker. (2003). *Writing Software Documentation*. Boston: Allyn & Bacon，是開發文件編製的一個很好的參考。

– 完 –

系統開發的基本概念

甚麼是系統? System

「系統」這個名詞幾乎每日出現在我們的生活中。 當你在地鐵車箱內經歷列車延誤時，你可能聽到以下的廣播：「由於訊號系統出現故障…」。 當使用「系統」這個名詞時，很多人會認為它是指複雜的東西。 這些印象是有理由的，因為系統就是一組互動元素，它們一起工作以完成特定的任務或功能。

系統的特性

組成一個系統的***元素***是互相關聯及互相倚賴的。
人體是一個系統，它由不同的器官(元素)組成，而它們一起工作以維持生命(功能)。
而圖書館資訊系統也由不同的元素組成，以支援它的借還、採購及編製目錄等功能。

🛠 *練習*

- 一部汽水售賣機可被看成一個系統。 列出它的組成部分及描述它們的功能 *(例如投入硬幣、確認、選擇物件、握緊及放開等.)*。
- 一套音響裝置系統也可被看成一個系統。 列出它的組成部分及描述它們的功能 *(例如擴音器、揚聲器、雷射唱片播放器及均衡器等)*。

每個系統必須有定義清晰的***界限***，它把系統與其***環境***分開。 任何物理系統的界限及環境，如人體及學校園區，均可容易被分辨出來。 但是，資訊系統的界限不一定是那麼明確。 例如，對於圖書館的借還部分，是否需要支援用戶的網上預約書籍需求，這沒有單一的正確答案。 這取決於個別項目計畫的目標。 因此，資訊系統的界限與系統範圍的概念息息相關。 系統的***範圍***告知系統在完成時，跟據用戶的需要，應達到甚麼要求。 環境基本上是影響系統的連串因素，它們可能影響系統的形式、開發及行為。 例如，自動櫃員機(ATM)與允許使用自動櫃員機服務的銀行客戶互動。 總括來說，資訊系統的環境，可被看成那些影響資訊系統行為的因素或各方面的集成體。

界面定義兩個實體之間的通訊界限。 一個資訊系統的***系統界面***允許資訊在系統與其環境之間流動。 例如，數據必須透過特定的界面設備，如鍵盤，進入電腦系統；而系統輸出將會顯示在輸出設備上，如打印機。 其實，***輸入***及***輸出***設備為用戶提供連接資訊系統的界面。

有時候，一個系統的規模太大，或功能過於複雜時，系統便會被分成較小的系統組件，稱為***子系統*** Sub-system。我們稱它為子系統(或簡單地稱為*系統*)，而不是系統元素，因為子系統提供多個功能。 例如，圖書館的借還部分提供書籍預約、書籍推介、書籍借出及歸還等服務。 系統的元素及子系統也是透過它們的界面而通訊的。 例如，書籍的推介是由借還部指

示給採購部作採購參考的。管理部分是另一個子系統。

最後，幾乎所有系統均有***儲存器***，記錄一些重要的資訊，例如是系統狀況。 一個簡單的例子是電視的開關器。 我們不能知道按下開關器的結果，除非我們知道電視機現在是否開著了。另一個例子是關於自動櫃員機怎樣回應無效的登入。 機器容忍兩次無效的個人識別號碼，但不允許第三次。 這些例子顯示一個系統，可能根據現狀，對相同的輸入有不同的回應。 那些狀況資訊便是保存在儲存器之內。

教學附註

- 圖書館的採購部可設立櫃台，直接從圖書館用戶收集書籍的推介，而不是透過借還部代為收集。 這樣減少了兩個部門(子系統)之間的一個界面，但在採購部與圖書館用戶(環境)之間需要一個新的界面。 在用戶的角度來說，減少系統界面的數目可能是一個好提議，用戶因此不需要記著何時採用那個系統界面。

🛠 *練習*

- 就圖書館的(i)物理特徵*(例如是入口、借還處、查詢處、書架及閱覽區等)*及(ii)功能*(例如是書籍採購、編目、預約及借還等)*，找出一所公共圖書館的子系統。
- 對於一部在有標路面行駛的自動駕駛車輛來說，它的子系統是甚麼*(例如是道路識別、操舵控制及摩打控制等)*? 在子系統之間流動的數據可能包括甚麼呢*(道路的虛擬圖像、操舵控制信號及速度控制信號等)*?

資訊系統

Wikipedia 對資訊系統有以下的描述。

> *一個系統，無論是自動化或人手操作的，由人、機器及/或方法組成，並收集、處理及傳播代表用戶資訊的數據。* *(摘自 Wikipedia)*

資訊系統展示系統的特性。 由多個元素(人、*機器*、*及/或方法*)組成，有清晰的界限作為定義，並配以界面。 資訊系統收集並處理輸入的數據，使它們變成對系統用戶有意義的資訊。

讓我們拿學校圖書館的資訊系統作為說明。 系統的輸入包括借出或歸還的圖書之編號、書籍預約及圖書館用戶編號等。 系統可能會保留狀況數據以支援圖書館的功能。 例如，系統需要記錄讀者所借出的書籍數目，以確保用戶不能借出多過他/她允許借出的書籍數目。 此外，為了檢查已借出的書籍有否逾期，系統可能需要每天掃瞄用戶記錄。 當已逾期的書籍還未歸還時，提示資訊將會輸出到借還部的電腦屏幕上。

學校圖書館的資訊系統用戶，包括圖書館職員及校內的讀者。 但是，用戶可能也包括其他有權接達圖書館系統所保持的資訊的有關人等。 例如，政府的審計員可能被允許接達關於學校圖書館資源使用率的統計，而圖書館資訊系統的報告功能可能支援這些統計。

用戶透過不同的電腦用戶界面，查閱電腦目錄、讀者借還記錄、及網上預約書籍等服務，從而與圖書館資訊系統互動。 當用戶在圖書館櫃台要求服務時，圖書館職員可能會使用儲存在圖書証的數碼資訊。

除了注意上述的學校圖書館系統的特性外，我們還需留意***環境***與***問題域***。

在學校圖書館的例子中，環境是學校管理員，或甚至提供撥款及定立學校圖書館應怎樣運作的規則的政府。
問題域定義系統所處理的問題的性質及特性。 在學校圖書館的內容中，這可能包括用戶種類、給予每個用戶種類的不同圖書館功能、預算費用及實施排程等。

在上述的學校圖書館資訊系統中，大部分描述均可應用在人手操作及電腦化的圖書館中。 當然，在自動化圖書館中，由於電腦的使用，一些輸入及輸出、界面及儲存器是有所不同。

以下總結了一個資訊系統的普遍特性。
- 每個資訊系統均有它的用戶。
- 每個資訊系統均有它的用途及提供給用戶的功能。
- 每個資訊系統均有界限及定義清晰的範圍。
- 資訊系統可能由數個子系統組成。 每個子系統的用途應被清楚定明。
- 在相同的資訊系統之內的子系統，最少會在某些情況下互動。
- 資訊系統會與它的環境互動。

🛠 *練習*
- 找出一些你所熟悉的資訊系統，包括人手操作的及自動化的。 對於每個系統，列出系統的用戶、系統用途、系統組件(及子系統)、系統輸入及輸出、系統界限及環境，以及系統儲存器所保留的數據。
- 對於一部在有標路面行駛的自動駕駛車輛來說，它的子系統是甚麼(*例如是道路記認、操舵控制及摩打控制等*)? 在子系統之間流動的可能數據包括甚麼呢(*道路的虛擬圖像、操舵控制信號及速度控制信號等*)?

開發新系統的好處與壞處

假設有一位校長考慮把人手操作的圖書館自動化。 這個決定需要衡量多個因素，包括自動化圖書館的好處及在財務限制下的資源承諾。

重新設計及實施一個系統的好處是滿足大部分(全部)用戶的需求。 但是開發一個新的系統並不是唯一的解決方案。 另一個可行的方案是在市場採購適合的***軟件套裝***。 例如，公司甚少選擇重新開發文字處理軟件，因為市面上有很多良好的文字處理套裝。 就算一間公司決定開發一個新的軟件，它仍需要在***內部開發***或***外判***開發之間作出選擇。 決定採用那種軟件策略之前，需要考慮以下因素：

- 一個合適的軟件_____是否存在；
- 公司是否有合適的_____ (在知識和經驗等方面)開發軟件；
- 內部開發軟件是否帶來_____；
- 能否找出可靠及有經驗的軟件_____承擔外判計畫；
- 當軟件推出時，公司是否有合適的人員_____軟件系統；及
- 計畫的預算_____等等。

即使開發新的系統在技術上及經濟上是可行的，仍需要考慮其它因素。 例如，如果新的系統可節省人力資源，多餘的勞動力可怎樣處理? 系統用戶會喜歡軟件嗎? 要記著，一個軟件系統的推出，可能為用戶帶來工作流程的改變，而用戶可能不歡迎那些改變。 除此之外，***系統轉換***不一定是順暢的。 例如，地鐵將軍澳線開始運作初期，也曾碰到困難。 此外，如新系統運作不正常，系統可能會有完全停頓的風險。

香港國際機場 1998 年 7 月 6 日啟用初期，一度出現過混亂，客運大樓的電腦出現故障，以致航班資料無法顯示，行李輸送系統亦出現了錯誤，客運大樓亦多次發生過供電及供水停頓等情況；機場快線亦出現過班次延誤及未能成功靠站的問題；貨運服務更一度接近中斷，超級一號貨站的交收系統停頓，大量貨物積壓在貨站。 http://zh.wikipedia.org/
英國倫敦希斯羅機場 March 5, 2008 Heathrow Airport, London Chaos 1. 0400 - Both passengers and staff have trouble locating car parks 找不到停車場 2. 0400 - Delayed opening of check-in desks results in long queues 辦理登機櫃台延遲開放 3. 0442 - First passengers arrive early but wait an hour for luggage 等待行李(1 hr) 4. All morning - Clogged conveyor-belt leads to long wait for luggage 行李輸送帶堵塞 5. 1630 - Baggage system failure; all check-in at T5 suspended 行李系統癱瘓，5 號大樓停用 6. 1700 - After long queues form at "fast bag drop" desk, BA suspends check-in of all luggage 7. 300 flights cancelled in the first 5 days http://news.bbc.co.uk/2/hi/uk_news/7318568.stm

因此，在考慮開發新的系統時，衡量其好處與壞處極為重要。 如果，最終的決定是建立一個電腦操作的系統，以代替人手操作的系統或老化的電腦系統，整個系統開發生命周期，都需要引入專門的技術。

計畫的潛力及風險，通常與計畫需求的性質有關。

- *程序自動化*
  - 開發新的系統代替部分人手操作，組織的基本運作保留不變
  - 集中改善效率
  - 把風險程度減至最低，但同時也減低了計畫的潛在優勢
- *程序改善*
  - 適度改變組織的操作，集中改善現有系統的程序效益，而不是程序效率
  - 有適度的風險及適度的潛在計畫優勢
- *程序重組*
  - 運用新概念及新科技的優點，例如是電子商貿，改變對組織的根本操作
  - 有高度的風險但龐大的潛在計畫優勢

請注意，新系統的功能越多，與它相聯的風險越高。
但是，一個多功能的系統所帶來的改善，包括生產力、效率及盈利，將會更多。

> 🛠 *練習*
> - 一間小型餐廳的經理計畫使用電腦科技改善運作。 請提議在餐廳內可使用的電腦科技，及那些科技將帶來的潛在優勢。 同時也找出潛在的計畫風險。

系統開發生命周期 (System Development Life Cycle)

系統開發生命周期(SDLC)，是描述在資訊系統開發計畫中的階段之概念性模型。 現今有不同種類的系統開發生命周期方法論，例如是*瀑布模型*及*應用系統迅速發展法(RAD - Rapid Application Development)*、*原型開發(Prototyping)*等。 這些方法論是基於計畫的性質引導計畫的開發。 某些方法論在特定的計畫種類上有比較好的應用，而有時候可能採用數個方法論的混合方法在特定的計畫上。

1. 瀑布模型 Waterfall Model

瀑布模型是最早期的開發的系統開發生命周期方法論，並在數十年來被廣泛使用。 它很流行並被看成是一種典型的系統開發生命周期方法論。 瀑布模型描述系統開發活動是_____及_____的。 它把系統開發工作分成多個_____，而每個階段均有定義清晰的起點和終點。每個階段也有獨特的目標以及輸出。 當完成一個階段，計畫開發將前進到下一個階段，而它從不會返回之前的階段。 瀑布模型通常分成 4 個主要的階段，描述如下：

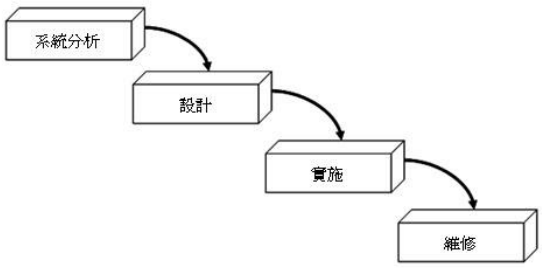


圖 1 瀑布模型的主要階段。

瀑布模型的四個主要階段可簡單地描述如下：

- *分析* – 評估現有的系統(現有的電腦系統或人手操作系統)，及找出所有用戶需求。如評估結果是可行的，建議新系統，而它的規格被制定成這階段的輸出。
- *設計* – 在這階段，擬定有關整體系統架構、數據、程式、及人機界面等設計計畫。
- *實施* – 在這階段，為電腦的解決方案編碼及測試。 當新的系統完成時，選擇適當的系統轉換策略，使系統運作。 在這階段也需考慮用戶培訓。
- *維修* – 一旦系統開始運作，便需要展開持續的系統維修。

有了瀑布模型，每個階段的輸出及文件將呈獻給客戶批核。 一旦客戶批核一個階段的工作，這個階段便完結，而下一個階段即將開始。 雖然在瀑布模型內可返回之前的階段(例如是從設計返回分析)，這情況甚少發生，而牽涉的工作將十分龐大，因為在每個階段都有大量的完成輸出及文件。

其它的系統開發方法論有所不同，但大部份也採用了瀑布模型的四個階段。 以上階段及其它系統開發生命周期方法的詳情，將在個別的章節探討。

✍ *練習*
- 瀑布模型有一些內在的問題，導致系統開發生命周期模型在近年的普遍性降低。 你知道那些問題是甚麼嗎?
- 鑑於那些問題，你認為瀑布模型應永不在任何軟件開發系統計畫內採用嗎? 為什麼?

瀑布模型的限制與不足

至今，我們對**系統開發生命周期**的方法論之討論，只集中於瀑布模型。 瀑布模型是基於一個簡單及有系統的結構性方法，這方法的系統開發活動，實質上是連續地排列的。 瀑布模型被廣泛使用的原因如下：

- 用戶需求在程式編碼前已被識別
- 不鼓勵用戶改變需求，使系統開發生命周期順利進行
- 清楚定明每階段的開始及結尾，加強了每階段的文件編製
- 每個階段依次序進行，而沒有任何重疊或反覆的步驟，這使管理上的控制更有效率

但瀑布模型不是完美的，他有以下的缺點：

- 顧客在軟件開發項目開始時，對系統開發通常只有一個_____的概念；
  因此，用戶需求未能在程式編碼前被識別
- 由於階段的_____之本質，若對之前「已完成」的階段作出改變，便變得困難
- 任何在初期階段的錯誤，要等到系統_____時方可能被發現，
  而且，修正那些錯誤將會是_____及困難的
- 在完成系統建議至實行任何可用系統期間，是一段頗長的時間，
  因此，設計錯誤將需要很長的時間才能發現

以上提及的缺點與瀑布模型的線性有莫大的關係。 瀑布模型缺乏彈性，
如果模型在需要時允許階段反覆，這問題便可舒緩。

🛠 *活動*
- 討論階段反覆在軟件開發系統的重要性。 舉出需要反覆的開發方法系統的例子。
- 閱覽這網頁學習瀑布模型的由來，及它在過去數個年代那麼普遍的原因。

教學附註
- 為了解決「純」瀑布模型的問題，推出了很多修改的瀑布模型。 有興趣的讀者可參閱 Steve McConnell 著作名為 *Rapid Development: Taming Wild Software Schedules* 的書中有關「生命周期計畫」(lifecycle planning)的章節。

其他系統開發方法

除了瀑布模型外，還有很多系統開發的方法。 其中兩個介紹如下。

2. 原型開發 Prototyping

系統用戶通常知道他們需要甚麼，但他們不知道建議的系統怎樣滿足他們的需求。 此外，很多系統用戶對系統分析員未能清晰描述他們所有的需求，因為他們常認以為分析員對系統環境已十分熟悉。 有些需求可能被忘掉。 而有些需求可能會被分析員誤解。 很多時候，用戶及分析員之間在言語上、或甚至文字上的溝通，也沒有發現那些問題。 如有任何問題，用戶通常要延至系統運作，及看見它的輸出時才察覺，但這暗示採納瀑布模型會有長時間的延遲。 **原型開發**方法論便解決了這個問題。

在原型開發方法中，建立了建議系統(原型)的_____版本，演示給用戶以收集早期的意見。意見將會用作改善原型，然後再次演示給用戶以收集意見。 這個程序連續地進行，直至用戶同意原型符合他們的需要。 當肯定了用戶的需求，原型可進一步開發至一個完備的系統。 這種方法，反覆地插入分析、設計及實施的階段。

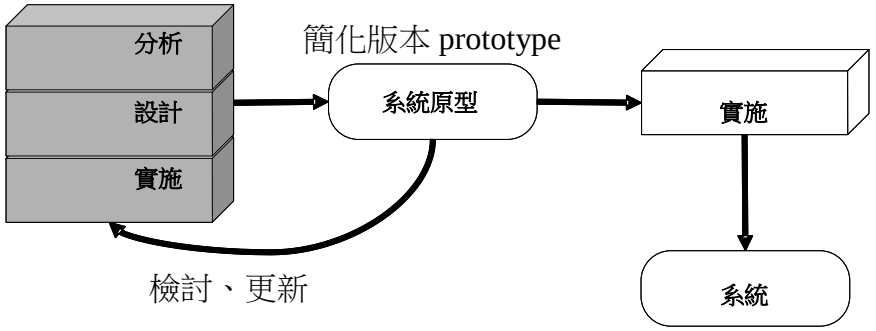


圖 1。 原型開發方法論的實施。

但是，如果項目隊伍決定開發一個全新的系統，而不是進一步開發原本的原型，原型便會被棄置。 由於原型旨於盡快從用戶中收集意見，設計及實施所花的時間有一定的限制，因此，原型的程式編碼可能不好。 所以，重新開始再建立系統可能更好。 在這情況下，原型只作為易於收集用戶需求的工具。 這種原型開發的方法稱為*棄置式原型開發*。

值得強調的是，原型只捕捉系統最重要的特徵。 否則，原型可能需要很多時間開發，而會有損原型開發方法的好處。

原型開發方法的另一個變型稱為*紙上原型開發*。 它的焦點是從系統的用戶界面反映用戶需求，例如是屏幕設計。 紙上原型開發的最簡單實施是把所有建議系統的重要屏幕顯示在紙上。另一個實施是以程式主要的用戶與系統間的互動。 但是它必須有一些程式編寫工作。 圖 2 模擬是一個說明紙上原型開發概念的例子。

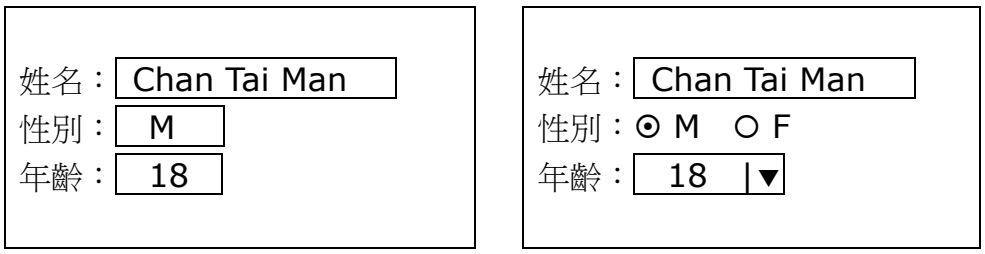


圖 2。 (左) 需要用戶鍵入文本作輸入的用戶界面原型。
(右) 根據用戶意見改善的用戶界面。

由於原型開發方法論，允許用戶在初期評估一個建議系統的效用，它使項目後期的改變較少，而作出這些改變往往是昂貴的。

原型開發方法論的特徵是其反覆性及彈性，它也解決了瀑布模型的主要缺點。 然而，原型開發方法論也有它本身的潛在問題。 其中一個潛在問題是分析員及用戶兩者，太集中於用戶界面的設計，而太小時間花在製造擁有所需功能的系統。 此外，用戶可能不斷提供次要功能的意見，或甚至在與分析員互動時加入新的需求 (即是*範圍蔓延*)。 實際上，意見收集周期不能無止境地進行。 何時停止意見收集周期，開始評估結果以完善系統也是一個潛在的問題。此外，在分析員與用戶或顧客多輪的互動後，原型最後可能已作出了重大的改變(面目全非)，令原本的設計理念變成差劣，因為一些基本的問題之前沒有被完全識別出來。

教學附註

- *範圍蔓延*，即是項目範圍不受控制的改變，大概是項目管理最重大的挑戰。 因此一個項目經理必須知道何時拒絕用戶引入新系統需求的要求。 處理用戶要求的一個方法是在建議系統的下一版本加入新的需求。
- 原型開發不只是應用在軟件開發上。 當一個玩具製造商打算設計一款新的玩具，他甚少立刻設計、製造及推銷它。 玩具製造商常會製造新玩具的原型，然後進行市場調查 (這個背景的用戶可能是批發分銷商或顧客)。 收集意見後，新款玩具的原型將重新設計，而另一個原型會產生，然後演示作另一輪市場調查。 只有當市場調查指示正面的結果，玩具製造商才正式大量製造新款的玩具。

3. 應用系統迅速發展法 (Rapid Application Development)

RAD 是一個相對快速及達至高質素的系統開發方法論。 它相對原型開發方法更為優勝。
RAD 常會使用 CASE TOOLS 電腦輔助軟件工程工具，來協助軟件的開發及維修。
當分析員輸入基本系統結構的資訊，到電腦輔助軟件工程工具中，那些工具將產生部分所需的程式碼及系統文件編製。

(類似 Raptor: 流程圖→C 程序)

教學附註

- 電腦輔助軟件工程工具的討論可在 http://en.wikipedia.org/wiki/CASE 找到。

請注意，軟件開發的 OOP 物件導向方法，也促進應用系統迅速發展法的使用，因在其它軟件項目建立的軟件組件可_____ (reuse)，而這明顯有助減少系統開發的工作。
例如，某些為普通科診所開發的軟件物件，可在開發牙醫診所的軟件系統時再用。
軟件物件的例子：病人記錄及預約管理。

由於應用系統迅速發展法也包括捕捉用戶需求，它與原型開發方法論具有同樣的潛在問題。
用戶在看見已有改善的原型時，可能也提升用戶期望。因此，管理用戶期望是一件棘手的事。

教學附註

- 除了使用應用系統迅速發展法工具，跟據 Wikipedia，應用系統迅速發展法 RAD 有其它六個特徵：
  - ❖ *原型開發*
    在短時間內建立一個功能及規模減少的應用(程式)，以引出用戶的需求。
  - ❖ *反覆開發*
    每個版本將會推出更多功能的應用(程式)，而用戶的意見會反映在下一個版本中。
  - ❖ *時間限制* (否則永無止境)
    管理技巧是把限期固定但可調整個別目標，並把部分功能推遲到將來的版本中，這使反覆周期更快完成。
  - ❖ *隊員*
    隊伍 Team 應是小型的，但隊員必須有經驗、具多項功能技巧及有積極性的。
  - ❖ *管理方法*
    管理層應參與，保持開發周期短促及強制限期。

🛠 *活動*

- 在網上尋找其它系統開發生命周期方法論，
  並拿它們與瀑布模型、原型開發方法論及應用系統迅速發展法作比較。

– 完 –

問題定義 problem definition

在編寫程式以解決問題之前，清楚地定義問題是十分重要的。
對於大多數的軟件項目，系統分析員會接觸用戶，
以收集用戶需求及定義系統旨於解決的問題。 他們通常會考慮以下的問題：

- _____是甚麼?
- 支援解決辦法的過程需要甚麼_____?
- 解決辦法的預期_____是甚麼?
- 人們現時怎樣_____問題?
- 問題或部分問題可否_____地以軟件的辦法解決?

系統分析員/設計員製作設計規格，specification
再交程式編寫員 (根據設計規格文件) 寫程式。program

設計規格描述系統的程式輸入和輸出及程式的功能性。
程式輸入可由用戶透過任何輸入設備鍵入，
或從數據儲存庫(的檔案或數據庫)中提取，
而程式輸出可被儲存、列印或顯示在屏幕上等等。
驗證程式輸入、輸出及功能性是否正確。

習作一：

1. 某老師利用 OMR 批改 MC，並將結果儲存於試算表

```
Answer:-AABBCCDDA...
1A01 30 ABCDBACAB...
1A02 25 BCDABCDBA...
```

2. 其後發現其中一題出錯
3. 問如何在不用重改的前題下，把分數更正

習作二：

Chan Tai Man → Chan TM
Chan Wai → Chan W

試寫文字把以上問題定義，並設計算法解決以上問題。

計劃及設計方案 system design

從上而下的方案 Top-down Approach

軟件設計的從上而下方案可追溯回 1970 年代。
它以問題最廣泛及最宏觀的層面作開始，然後往下一層面工作，那裡有更多專門的詳情。
步驟可能在詳細的層面內重複，直至最底層的描述足以配對所選的程式編寫語言作實施。
很自然地，從上而下的設計方案會形成一個系統組件階組。

人們經常使用從上而下的方案在日常生活中解決問題。
讓我們利用計劃一個海外假期作說明。
在高的層面，我們需要決定假期的日期及目的地。
在下一個較低的層面，我們需要向僱主申請假期，決定前往目的地需乘搭的交通工具。
那些目標可再向下層伸延，例如取得、填寫及遞交假期申請表，
及向旅行社預定所選的交通工具及酒店住宿。

電腦模組/部件：
CPU
輸入設備
輸出設備
儲存設備
通訊設備

圖 1 顯示那個階組。 有些人可能爭議我們應計劃本地的交通，
例如是乘巴士到達機場。
那個動作是否包括在內視乎計劃任務的範圍。

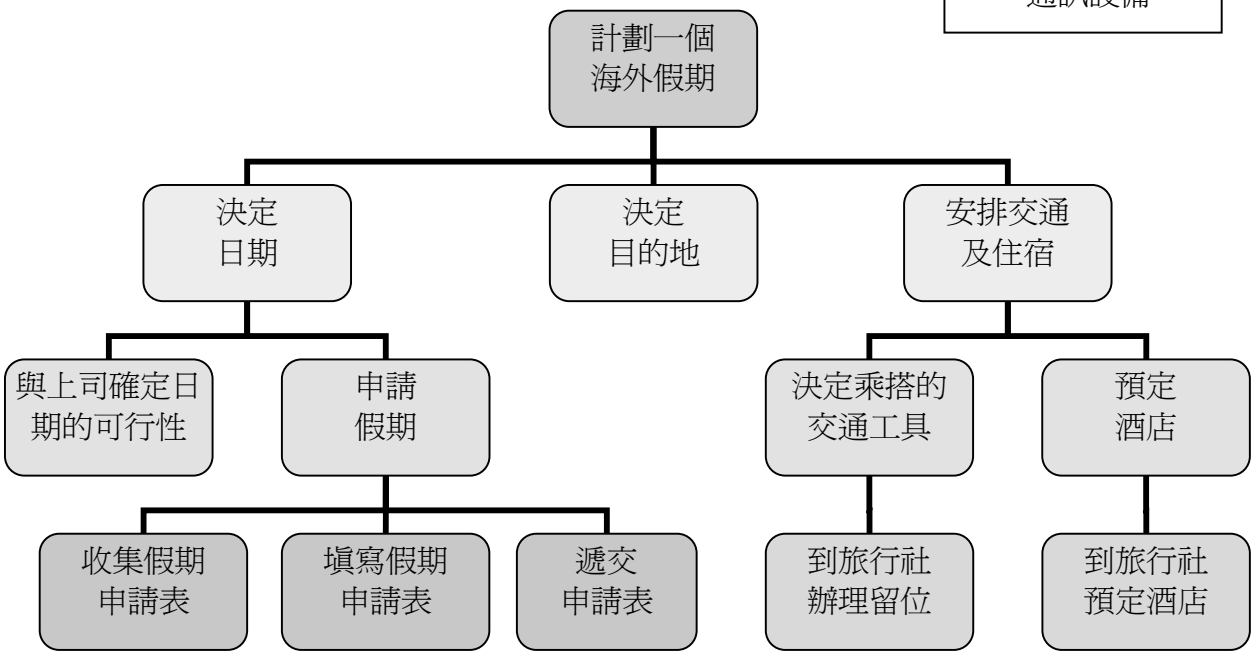


圖 1。 計畫一個海外假期的從上而下方案。

在程式開發中，其中一個最有名的從上而下開發方案是功能性方案或設計。功能性設計在 Wikipedia 裡被形容爲「*…確保一個電腦程式的每個模塊部分只有一個責任，及對其它部分產生最少的負面影響下執行那個責任*」。 這方案集中於一個軟件系統的功能性方面，並建議不同的

軟件組件實施那些功能。 爲替代實施龐大及複雜的組件，那些組件被分成較少的合作組件以達到軟件的目標。 明顯地，母組件需要協調子組件的工作。 例如，在安排交通及酒店前，應決定假期的目的地。

從上而下的方案步驟如下：
1. 清楚_____問題
2. 把問題分成細小的_____(sub-problem)
3. 爲每個子問題重複步驟____，直至它小得足以被軟件實施 (寫_____)解決

很自然地，功能性方案會導致軟件開發的模組方案。 這方案經數十年來在實際的軟件設計上被驗證爲有效，它把大型複雜的軟件開發項目分拆成可管理的組件，並由一隊程式編寫員處理。但是，隨著軟件系統的規模增加，當使用從上而下的方案時，在軟件組件間找出類似的地方是困難的。 軟件的可再用性在這情況下難以達到。

Wikipedia 指出從上而下方案的另一個潛在問題：

> *從上而下方案強調計劃及對系統的全面了解。 所以它本身是要最少在系統的某部分設計中達到一定程度的詳情，才可開始編碼。 但是這樣延遲了一個系統的最終功能性單位的測試，直到主要的設計完成後才可進行測試。*

從下而上的設計 **Bottom-up Approach**

簡單地說，從下而上的設計指明個別系統組件的詳情。
然後組件會被連接一起形成較大的部分，而它們會不斷連接直至形成一個完整的系統。

> *從下而上強調編碼及早期的測試，它們在定明第一個模塊後便可開始。 但是，這個方案有以下的風險，模組可能在它們不清楚怎樣與系統的其它部分連接的情況下被編碼，而那些連接可能不如第一次想像般容易。 編碼的可再用性 **reuse** 是從下而上方案的其中一個主要優點。* Wikipedia

人類甚少以從下而上的方法解決問題。 有趣地，他們也甚少只以從上而下的態度解決問題。人們在解決問題時通常會穿插於從上而下及從下而上的方案之間。 普遍地，我們嘗試以從上而下的方式了解問題及解決辦法的組件。 但是，我們經常尋找機會在其它問題上再用上解決辦法，並以此作爲所關注的問題的部分解決辦法。 這其實也是在大多數現代軟件項目中所採用的策略，如在 Wikipedia 裡的描述：

> *現代軟件設計方案通常結合了從上而下及從下而上的方案。 雖然好的設計需要對整個系統的了解，理論上會導致一個從上而下的方案，大多數的軟件項目在某程度上會嘗試使用現有的代碼。*

從下而上的方案不會像從上而下的方案般得到那麼多關注。 問題是由於建立可再用 reuse 的軟件組件是十分困難的。 這種情況一直維持，直到軟件開發的物件導向方案普及起來。 物件導向語言，如 C++及 Java，利用如 inheritance(繼承)

(見 http://en.wikipedia.org/wiki/Inheritance_%28computer_science%29)及 polymorphism(多形性) (見 http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming) 的特徵，使程式編寫員可容易地實習從下而上的設計。 像從上而下的方案，從下而上的方案強調模組軟件開發。

模組化 (子系統/子程式)

模組性是關於把程式分拆爲細小的單位，稱爲模組 module。 模組化的目標，是允許從簡單有凝聚性及鬆散地連結一起的模組，建立複雜的系統。 凝聚力是一個在模組內的量度，它量度來源碼一起運作以提供一個定義好的用途的協調程度。 連結是每個程式組件互相依靠的程度。 一般原則是達到一個高凝聚力 Cohesion (單用途的)及鬆散地 Coupling 連結一起的模組設計。在軟件維修時，能夠限制(減少)軟件系統內對較少個模組作出改變的影響。

電腦資訊系統： 輸入模組 輸出模組 report 系統管理 admin 部分 數據處理 process 模組	員工管理系統： (1) 資料查詢 Enquiry (2) 資料更新 Update (3) 報告/統計 Report

逐步求精法 step-wise refinement

逐步求精法是一個從上而下的程式設計技巧，那裡僞代碼語句會不斷擴展，直至我們到達一個層面，那裡的「正確」程式編寫語言碼的翻譯是「明顯」的(Wirth, 1971)。 有些文獻把逐步求精法相等於從上而下的程式編寫語言方案。

逐步求精法的例子(以 C 程式編寫語言寫成)可在此找到。
http://www.cs.odu.edu/~zeil/cs451/Lectures/05design/stepwise/stepwise_htse1.html#QQ1-2-1

結構化程式編寫 structured programming

結構化程式編寫是一個程式編碼的技巧，由兩個數學家在 1960 年代建議，它使用一個模組階組，其中每個模組均有一個入口及一個出口點，而控制是經結構的無條件分支向下傳送到結構的較高層面上。 換句話說，「GOTO」語句的使用是不被鼓勵的。

那裡有三種控制流程 (結構)：
_____sequence、_____decision 及_____iteration。

經常採用從上而下的設計模型。

– 完 –