國立屏東大學 資訊工程學系 物件導向軟體工程

3. 需求分析

需求是「該被開發出來的規格」，因此需求最終必須導出規格(Specification)。規格即為軟體系統開發的指引(guideline)。在後續系統設計與開發的過程中都必須遵循。需求工程(Requirements Engineering)是描繪出系統該有哪些功能的抽象層次規格並區分出其優先次序的工作，會出現在專案啟始和細部評估階段。需求不完整以及缺少使用者參與是導致專案失敗最主要的兩個原因，兩者的起因都是因為需求工程失敗所導致的。由於軟體系統之成敗取決於需求之滿足，良好的需求工程便成為軟體開發專案是否成功的重要關鍵。通常，需求工程包含需求擷取與需求分析兩大工作項目。

3.0.1 需求擷取

要瞭解需求必須從資料的搜集開始，由於資料的搜集主要是針對既有的系統或運作方式、流程等著手，所以這一部分的工作又稱為領域分析。

- 需求來自於
  - 系統的直接使用者；
  - 其他關鍵人員（例如，管理者、維護者、安裝人員）；
  - 與此系統互動之其他系統；
  - 與此系統互動之硬體設備；
  - 法律和管制限制；
  - 技術限制；
  - 商業目標。

3.0.1.1 領域分析與領域知識

- 領域分析是根據既有系統(或處理流程)及其開發的歷史、領域專家的知識、背後的理論，加以辨別，收集以及組織相關資訊的過程。
- 領域知識是指與特定領域相關的各項知識。
- 領域分析是系統開發前的預備知識。對於待開發產品之背景資料分析與蒐集、對於產品領域知識與操作越了解，越能在將來做出好的決策。
- 簡單地說就是也就是從各方面對目前既有的系統(或處理流程)進行資料蒐集。
  - 在資料搜集的過程中，不要限定範圍或是種類，舉凡是跟領域問題有關聯的都要蒐集起來。任何資料都能讓你對於計劃領域有更深一層的了解。
  - 針對特定的需求，決定你要蒐集的方向，也就是跟計畫密切相關的資料。
- 領域專家
  - 領域分析過程中，你會接觸到許多人，其中對領域有深度瞭解的人稱為領域專家(Domain expert)。這些人大都是軟體專案的客戶或是既有系統的使用者等。
  - 他們對企業的各項活動以及業務流程最熟悉。而這些人也將在系統開發的分析階段扮演著重要的角色。
- 了解領域知識的好處
  - 可以有效率的和客戶溝通
    - 一般來說，從事領域分析的大多是系統分析師。有時候，也包括負責專案的專案經理，他是開發團隊與客戶間主要的對話窗口。但是嚴格說來，參與專案的每一個成員都必須對相關的領域有基本的認識與瞭解。因為這可以增進開發團隊與客戶之間的溝通。

        - 了解企業既有的(As-is)處理流程，對於所要開發的系統會有實質的幫助。尤其對經常變動的企業邏輯 (business logic)或是商業規則(business rule)這方面尤其重要。
        - 預留未來新系統的開發空間
            - 在你為客戶開發系統的同時，你會對其業務有更深一層的瞭解。你甚至會瞭解到，客戶的許多既有的處理流程可以利用資訊系統的優點來改進，幫助企業改進執行的效率、提升生產力、降低成本開銷等。對於這些觀察，可以經由與客戶高階執行階層討論與提出建議，以進而向客戶提出計畫書。為另一個新的計畫做準備。

3.0.1.2 需求擷取的方式

在擷取使用者需求之前，必須瞭解系統之潛在使用者及可能之人機互動。需求擷取常用的方式如下：

- 企業既有的報表、表單、操作流程相關文件
    - 對於任何與開發中計畫相關之報表，表單之搜集。這些資料的主要來源為企業內既有的文件、業務流程說明、工作內容、作業描述等等。仔細閱讀這些資料以了解企業的核心流程，企業功能以及企業規則，幫助自己建立對該企業的一個普遍認識。對於人工作業時所採用之單據更需要仔細蒐集和保留。
- 訪談
    - 訪談是需求擷取方法中相當常用的一種方式。
    - 訪談是系統分析最有效且最普遍的資料蒐集方法，訪談時分析師親自與使用部門的主管或相關作業人員面對面討論實際作業的情況、報表和資訊需求等。
- 在訪談期間，系統分析師蒐集到的可能是事實、選擇或推測，並且可觀察到人們的肢體語言、情緒和他們對於現行系統之觀感等。
- 透過訪談你可以跟使用者有面對面討論與溝通機會，對於使用者的需求能夠有較真實的體驗。
- 訪談可分成兩種方式：
    - 開放式訪談(Open Interview)
        - 分析師事先不預定表格、問卷或固定的標準程序，訪談過程全由使用者自由談論其工作。
        - 適用於分析師對問題領域不熟悉或無法預期之情況。
    - 結構化訪談(Structured Interview)
        - 結構化訪談又稱為標準化訪談或導向式訪談，其訪談過程近似於詢問(Interrogation)而非交談 (Conversation)，所要求資訊的深度、專門程度亦較深。
        - 這種方式的特點是把問題標準化，然後由受訪者回答或選擇。所有的受訪者都回答同一形式的問題，其作法如下：
    - 每討論一個主題時，先由分析師依其現有瞭解的知識，提出簡短敘述。
    - 使用者針對主題作深度分析，但分析師應在適當知識深度時加以中止。
    - 對某個主題有初步瞭解，就可以進行另一主題的訪談。
- 訪談之問題依其性質可分成開放性與封閉性問題：
    - 開放性問題
        - 用來探索分析師無法預期的回答或無法明確詢問之問題。
        - 優點是能讓先前不知道的資訊浮現出來，分析師可以用一些非預期中的問題，不斷的來探究新的資訊。
        - 缺點是回答問題所花的時間較長，也較難作結論。
    - 封閉性問題
        - 適用於問題可預期且回答可明確描述之情況。
        - 優點：訪談的時間較短，問題可較廣泛。
        - 缺點：所列回答之選項未必包含受訪者所要回答之答案。
        - 封閉性問題可以有下列幾種設計形式：
            - 對與錯的選擇方式。
            - 多重選擇的方式。
            - Likert尺度的衡量方式。用一些級距來衡量受訪者意見的強弱程度，例如用很好、好、普通、差與很差五個等級。

- 訪談之前必須要有詳細規劃，訪談之後必須將其結果詳實地紀錄下來。
  - 訪談之前的規劃應該包括：
    - 決定訪談對象
    - 設定訪談的議題
    - 準備相關事宜
      - 基本上，你可以先列舉所要討論的問題在一張紙上。讓訪談有個目標，這樣才不至於讓訪談變成聊天。
- 訪談注意事項
  - 比如說，問題可以從＂你希望系統提供什麼樣的功能？＂開始，
  - 或者是＂目前你們對這項工作的作業流程是如何進行的？＂。
  - 從這些問題當中可以引領出許多與技術面，執行面，操作面等等相關的討論議題。
  - 另外，訪談對象的選取要跟訪談內容相關。比如說，如果你是要了解一項特定的作業程序，那麼，訪談的對象最好是執行此作業的人員，而不是資深經理。
  - 要仔細的聆聽受訪者的回答，同時將重點記下來，在得到允諾的情況下，還可以將訪問的內容用錄音機或錄影機錄下來，以有效的掌握訪談內容。
  - 訪談結束後，需在48小時之內將訪談內容整理出來，因為經過48小時之後，訪談的內容會慢慢的從記憶中消失。
  - 不管是面對開放性或是封閉性的問題，分析師不要強調問題答案的對或錯。
  - 在訪談時，不要對新的系統做任何預期的想法。需讓受訪者知道他們的意見會被仔細的考慮，並讓受訪者知道專案的完成需要很多的步驟，同時還有很多人的觀點都需要一起考慮。
  - 從系統潛在的使用者、管理者與對現行系統有經驗的專業人員等尋找各式各樣的觀點，以對新系統作全盤性的瞭解，如此才能設計出大多數使用者都可接受的系統。

- 開會討論
  - 開會討論與訪談的方式很類似。
  - 開會討論是一種很有效率的資料蒐集方式。使用者代表與系統開發人員聚集一堂，將所知道的事實、觀念說出，讓所有與會人員一起相互溝通意見。
  - 此方法的優點是較易獲得正確的資料，即使有不同的意見與觀念，經由眾人研究亦能加以修正。此外，亦可發揮腦力的效果。
  - 缺點是安排溝通與協調較費時。
  - 系統開發的過程中通常都會訂定有每週定期的開會時間表用以
  - 檢視計畫的進度
  - 確認工作細節
  - 確認需求的正確性
  - 各項相關任務的分派
  - 人力資源的調度等等事項。
  - 這些事項都是一個系統開發過程中的相當常見的變數，因此舉行定期的開會討論為一個有效的管理方式。
  - 除了管理層面的定期開會討論之外，有時候它也用在技術面的討論上。
- 其他需求擷取的方式還包括有：
  - 觀察
    - 實地觀察所獲得的資料正確性會比查閱文件為高，亦能驗證所蒐集資料之正確性及補充不完整的資料，透過觀察可以獲得第一手的資料。
    - 僅用觀察仍無法完整的反映出組織的真實情況與需求，例如被觀察者行為可能改變。
    - 選擇正常與例外情況之時機或對象來作觀察，可獲得更多的資料。
  - 問卷調查
    - 當潛在使用者太多或分布太廣時，可考慮以問卷之方式擷取需求。
    - 一般來說，問卷調查適合於大型企業或公眾資訊系統的設計，因為它所涉及的作業範圍或對象太廣，系統分析師無法逐一親自調查，故利用問卷方式來蒐集使用者需求較為可行。
    - 設計一份好的問卷需有相當的練習與經驗，因為問卷上的問題是以文字靜態的表達，故

問題之語意與邏輯必須很清楚且有條理。
- 問卷設計時也可以用各種不同的方法來問同一個問題，以觀察各種可能的答案。
- 正式問卷調查前需有先導測試。
- 經由先導測試之檢討與回饋可進一步修飾問卷，及早發現問卷可能之問題，對提升問卷之效能有很大之幫助。

◦ 抽樣調查
- 當需要瞭解的使用者人數過多時，可以抽樣方式進行。

◦ JAD會議。
- 聯合開發(Joint Application Development, JAD)主要之精神，是透過一個二至五天的集會，讓開發者與顧客能夠快速有效，而且深入的檢討需求並取得共識。
- 聯合開發的具體結果是產生完整的需求文件。
- 具體步驟
  - 範圍界定
    - 首先，由專案出資單位的高階主管定義專案的範圍，並以文字記載，且由高階主管和JAD的召集人一起簽訂契約。這個步驟使JAD的召集人得到授權來進行需求分析，對於目標與範圍也有了約定。
  - 關鍵人員的熟悉
    - JAD的召集人要花一些時間訪談關鍵性的使用者及管理人員，以瞭解專案的背景資料及重要的需求。
  - 會議準備
    - JAD會議事前的準備工作應包括下列項目：
      - 整理需求文件草稿。
      - 分送需求文件草稿。
      - 安排助理人員。
    - 準備會議室。
  - 會議進行
    - 會議進行時，召集人引導大家充分利用各種視覺上的輔助工具如貼紙、白板、投影片、圖表等，將需求表達出來並做有效的溝通及共識的達成。
  - 文件產生
    - 最後階段將JAD會議所蒐集的需求整理成需求文件，為達到會議的效果，需求文件的準備要非常快速，例如二、三天。最後再召開一次審查會議，以確認需求文件的內容。

◦ 小組討論
◦ 腦力激盪

3.0.2 需求分析

利用各種需求擷取的方式，你獲得了許多跟即將開發的系統相關的資料。這些資料可能很凌亂，雜亂無章，因此你必須將這些資料做整理與分析。

3.0.2.1 需求的種類

- 需求應該僅說明系統該做什麼(What)而已，而不是直接說明系統該怎麼做(How)
  - 對於所擷取之需求資料可以分為兩大類：
    - 功能的需求 (functional requirement)
      - 系統該提供什麼樣的行為
      - 功能的需求主要是在描述系統該做什麼。也就是系統要提供給使用者的服務項目。
      - 對於系統所提供之功能的描述可以包括什麼樣的輸入是這個功能所必須的、這個功能的處理流程與步驟、以及經過資料處理後，這個功能的輸出為何等等。
      - 功能性需求是在描述系統應該要做什麼，它是一段系統功能的描述。例如：
        - 自動提款機系統會檢查插入的金融卡之有效性。

        - 自動提款機系統會驗證客戶所輸入的PIN碼。
    - 非功能的需求(non-functional requirement)
        - 系統的特性或限制
        - 非功能的需求是指跟系統的執行效率，效能之需求，且可以量度的(measurable)的項目。下面所列為RUP的軟體需求文件中所列舉之非功能性需求參考項目：
            - 反應時間(response time) 對於使用者所觸發之事件的執行所將花費之時間。
            - 使用性(usability)描述對於一個正常的使用者所需之訓練時間。
            - 可靠度(reliability)可靠度可包含許多要項，最常見的是描述系統的失敗率。
            - 效能(performance)表現度可包含許多要項，最常見的是描述系統在每秒鐘可以處理的交易量。
            - 維護性(maintainability)描述任何可以增進系統維護之相關項目。比如說，編碼的準則，命名的標準等等。
            - 範例
                - 自動提款機系統會限制每一張金融卡24小時內領出之金額不得超過250美元
                - 自動提款機系統會用C++開發。
                - 自動提款機系統會用256位元的加密方式與銀行連線。
                - 自動提款機系統會在3秒內驗證完一張金融卡。
                - 自動提款機系統會在3秒內驗證完PIN碼。

3.0.2.2 需求分析描述

- 對於需求擷取所獲得的資料，你應該定義出所要解決的問題。定義問題的原則是：儘量使用淺顯易懂的句子，不要長篇大論，也不要太抽象，相關之流程敘述要記載明確。問題的敘述最好是從使用者的觀點出發。
- 範例
    - 對於一個電影院(火車、客運等等)訂票系統來說，我們可以將使用者的需求定義如下：
        - 使用者必須登入以使用此系統。
        - 使用者可以利用瀏覽器找尋相關電影並預訂電影票。
    - 如果你用“ 實現訂票流程自動化 ”來描述這個系統的功能，那就把問題的範圍定義變得太廣，很抽象。對於接續的分析工作也較無實質的幫助。(基本上，上面對於系統的描述並不是錯誤，但是它出現的地方應該是在計畫書中，而不是在需求分析描述。)
- 需求描述中定義需求的技巧
    - 由使用者的觀點來看系統。也就是當你是使用者時你會希望系統提供什麼樣的功能。盡量把自己放在一個使用者的位置或是角色。
    - 對於功能性需求的描述，你應該可以將它們歸納成事件(Event)
    - 範例
        - 對於“ 使用者搜尋音樂CD”這個功能，系統允許我們輸入關鍵字以作為搜尋的依據。
        - 要知道，只是輸入關鍵字根本上不會發生什麼事。只有當你按下了，比如說一個按鈕，系統才會開始做事。 ” 搜尋音樂CD”就是一個事件，這個事件(藉由按了按鈕)觸發系統去處理或是執行某些特定的工作，並且對於這個事件，系統會將處理結果回應給使用者。
    - 範例(ATM)
        - 大部分的人都使用過自動提款機(ATM)來從事提款，轉帳等事項。如果你仔細地觀察一下ATM的畫面，你會發現到一部ATM的主畫面以及其他可用的選項都會對應到一個事件上。比如說：提款，存款，餘額查詢，轉帳等等。這些不都是一部提款機所提供給使用者的功能嗎？你，一個使用者，必須製造某些事件來觸發或是告知系統你到底想要它做什麼。
- 功能性需求(事件)的描述方式

- 主詞+動詞+受詞的形式。
- 你可以把事件當作是需求描述的精簡版或是摘要(summary)。

3.0.2.3 Event Diagram

我們可以利用事件圖(Event Diagram)來思考辨識事件，如figure 1。

Fig. 1: 事件圖思考方式

除了一般的事件圖外，亦可使用CRUD圖。CRUD是由create、retrieve、update與delete的第一個字母所組成的縮寫，利用我們對資料的動作來捕捉事件，對於發覺隱性的功能或是使用者相當有用。

- 假設系統需要提供「使用者查詢音樂CD、、、使用者加入會員」、「使用者訂購音樂CD」等等顯而易見的功能。我們可以就所發現之資料以及動作先在紙上嘗試著畫出如figure 2之基本草圖：

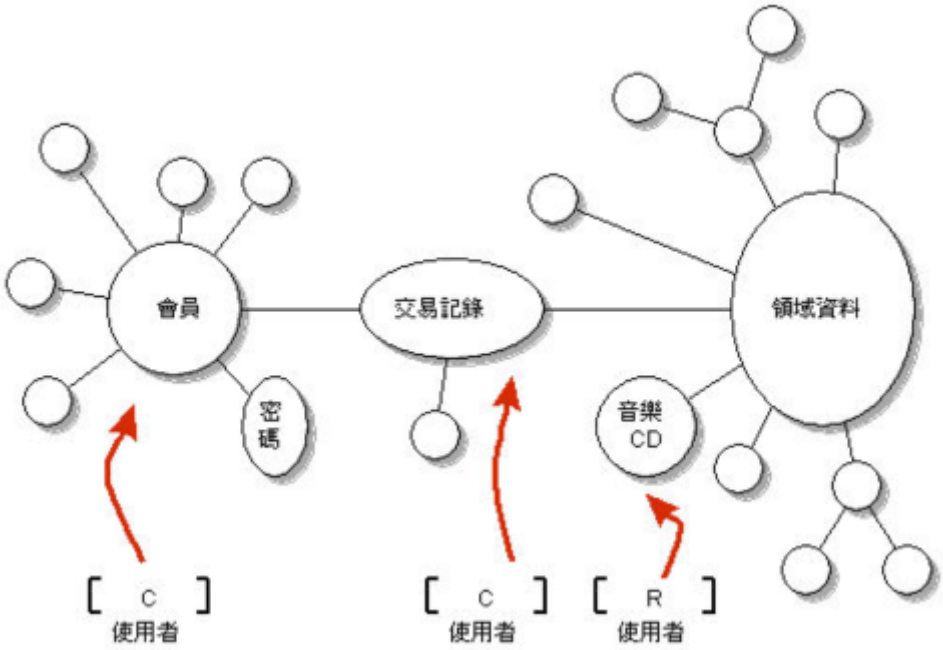

Fig. 2: CRUD使用範例之1

- 在figure 2中，在使用者上方我們以方框標記出目前為止我們發現到對於該資料的動作（所以，方框中的符號可能包括C、R、U、D四個字）。接著，我們對於每項發覺之資料，嘗試著詢問：還有誰可以執行其他的動作呢？例如說我們知道使用者可以查詢音樂CD的資料（上圖最右邊之箭頭所示），所以我們發現了R，那麼對音樂CD，誰可以執行CUD呢？也就是說音樂CD的資料是誰在負責建立，誰可以更新，誰可以刪除呢？這時候，管理者的概念就會跑出來了。
- 而我們的事件需求就會增加下列幾個項目：「管理者新增音樂CD資料」、「管理者刪除音樂CD資料」、「管理者更新音樂CD資料」。管理者只負責這些事項嗎？如果不是，還有哪些呢？所以，導引出管理者這個角色之後，我們更可以從該新發現角色之角度來思考、探究其所需之功能，這部分

與領域知識相關，如圖3-3中右邊圓圈框起來的區域。

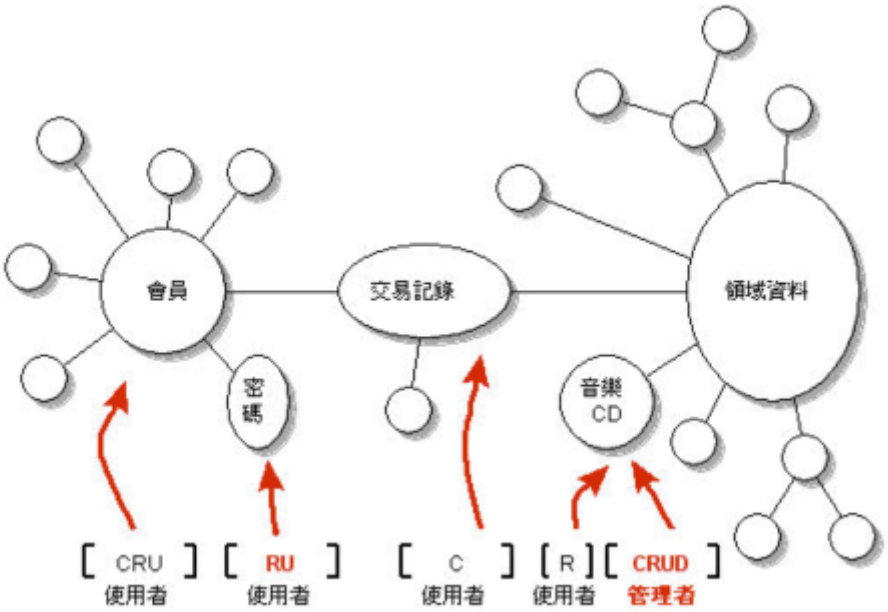

Fig. 3: CRUD使用範例之2

- 另外，對於細微的資料也要仔細辨識。例如從分析中我們知道會員可以登入到系統，登入時要提供帳號以及密碼。密碼雖然是很小的資料，但是從我們的經驗我們知道提供「使用者更改密碼」是一個有用且必須的功能。因此，對於密碼這個資料，我們有U的動作。因為密碼是與會員相關，所以我們可以將密碼從會員中獨立出來。從了U的動作，系統應該提供查詢密碼的功能，因此，對於密碼這個資料，使用者可以對其進行R□U的動作；另外，你曾經使用過那個系統它有提供刪除密碼的功能呢？至少作者從來沒有見過任何系統有提供這項功能。因此，並不是所有的資料都會帶有CRUD的動作。

3.0.2.4 Event Table

當你歸納出事件之後，可以將之紀錄於表格中，成為事件表。 事件表是用來記錄系統功能很有用的工具。不要擔心功能到底需要怎麼被實現出來，先把系統當成一個黑盒子。這樣做的好處是讓參與計畫的人員能將焦點放在系統高層次的觀點，從外部來看系統，而不是系統內部的運作情況。在很多的經驗中我們發現人們一般都把焦點放在系統的How, 而不是系統的What□這一點要值得注意。

- 事件列表格式 (可自訂)
  - 事件表欄位說明
    - 事件(Event)□定義事件名稱，利用主詞+動詞+(受詞)的型式。
    - 觸發器(Trigger)□系統是如何得知事件的發生? 也就是指事件是如何被觸發的，例如特定的操作、特定資料內容的改變或輸入、特定的情況或條件或是特定時間。
    - 來源(Source)□資料來源
    - 活動(Activity)□事件發生時， 系統要執行的任務。活動要以動詞為開端。 例如：查詢(Retrieve)...□更新(Update)...□產生(Create)...□取消(Delete)...□
    - 回應(Response)□如果需要的話，系統該產生什麼樣的輸出？
    - 目的地(Destination)□輸出到哪裡去。
    - [optional]優先權(Priority)□此數值是指該需求在所有需求中的優先順序，普遍用來指定優先順序的是MoSCoW代碼[1]。
      - Must have

        - Should have
        - Could have
        - Want to have
        - [optional]截止時間(Due Date)：該需求必須完成的時間。
    - 範例
        - 一個線上購物系統，最明顯的事件就是＂顧客下訂單＂。我們可以分析出：事件名稱就是顧客下訂單。訂單是顧客下的，所以事件的來源來自於顧客。這個事件被觸發是因為訂單資料的到來，這是來自來源的一種要求。對於這個事件，系統應該要產生一筆新的訂單記錄，以記錄這個活動。所以，＂產生一筆訂單＂是活動的名稱。而＂訂單編號＂與所產生的訂單是這個活動的回應，此回應的目的地是負責出貨業務的部門。
        - 另外，我們也假設系統必須在每個月5日產生前一月份的月報表。

Event	Trigger	Source	Activity	Response	Destination	Priority	Due Date
顧客下訂單	資料(訂單資料輸入)	顧客	在資料庫中產生一筆新訂單	1. 訂單編號 2. 訂單	出貨部門	M	2014/10/1
產生月報表	時間(每月5號)	系統	依據資料庫內的資料產生前一個月份的報表	月報表	經理	S	每月5號

Tab. 1: Event Tabale範例

- 將需求描述轉換為事件表
    - 思考方向
        - 嘗試著鑑別有哪些事情是必須由系統來自動化執行。
        - 嘗試著鑑別所有想要從系統取得某些東西的外部實體。
        - 嘗試著鑑別所有想要從系統取得某些東西的內部實體。
        - 嘗試著鑑別所有需要被儲存的資料。
    - 由建立(Create)、存取(Retrieve)、更新(Update)以及刪除(Delete)這四個動作來檢視系統可能發生的事件行為。這四個動作統稱為CRUD 。
        - 想一想有沒有以資料為導向的事件。
        - 想一想有沒有以時間為導向的事件。
        - 利用主詞+動詞+(受詞)的型式來歸納出事件。
        - 嘗試著鑑別各事件稍後會使用到的任何資訊。

3.0.2.5 Glossary(詞彙表)

事件表固然記錄了系統所應提供的功能，可是，在任何事件的過程中都會牽涉到許多領域中的資料，概念。你要利用詞彙表將它紀錄、整理下來。詞彙表將在後面有很大的作用。對於不清楚的詞彙也要一併記錄下來。然後在與相關客戶人員討論以求得更進一步的了解

- 詞彙表內容
    - 捕捉所有出現於需求描述的所有名詞。
    - 捕捉所有出現於事件表中的所有名詞，以及伴隨著的資料項目。
    - 捕捉在既有的系統，現行程序及現行報告、表單中的相關資訊。

詞彙	解釋
登入資料	帳號、密碼
訂單	訂單包含有客戶資料、訂購項目
客戶資料	客戶編號、姓名、住址、電話、email
訂購項目	編號、音樂CD摘要
音樂CD	音樂CD摘要、曲目
音樂CD摘要	專輯名稱、演唱者、類型、單價、出版商
曲目	歌曲名稱、時間長度、作詞者、作曲者、歌詞

詞彙	解釋
月報表	（格式待查）
客服建言	發言者、主題、內容、日期

Tab. 2: 詞彙表範例

3.0.3 軟體需求規格

- 軟體需求規格(Software Requirement Specification)並無固定的內容，但通常會包含下列的項目
  - 系統目標
  - 系統範圍
  - 系統整體描述
  - 功能需求
  - 非功能需求
  - 系統所需提供之使用者、軟體、硬體等切面之需求
  - 其他資料

建議至少應包含整體系統的描述與到功能性需求與非功能性需求，可使用下列項目表達：

```
* Context Diagram
*功能性需求
* Event Table
* UML - Use Case Diagram/Form
* SysML Requirement Diagram
*非功能性需求
* SysML Requirement Diagram
```

3.0.4 Context Diagrams

- Context diagrams are very useful for a high level overview of the system
- It is a representation of the system boundary and all the actors that interact with the system
- It helps visualize:
  - What is inside and what is outside the system boundary
  - Who are the actors that interact with the system
  - Which actors are human and which are external systems
  - What are the basic information flows into and out of the system
- Example

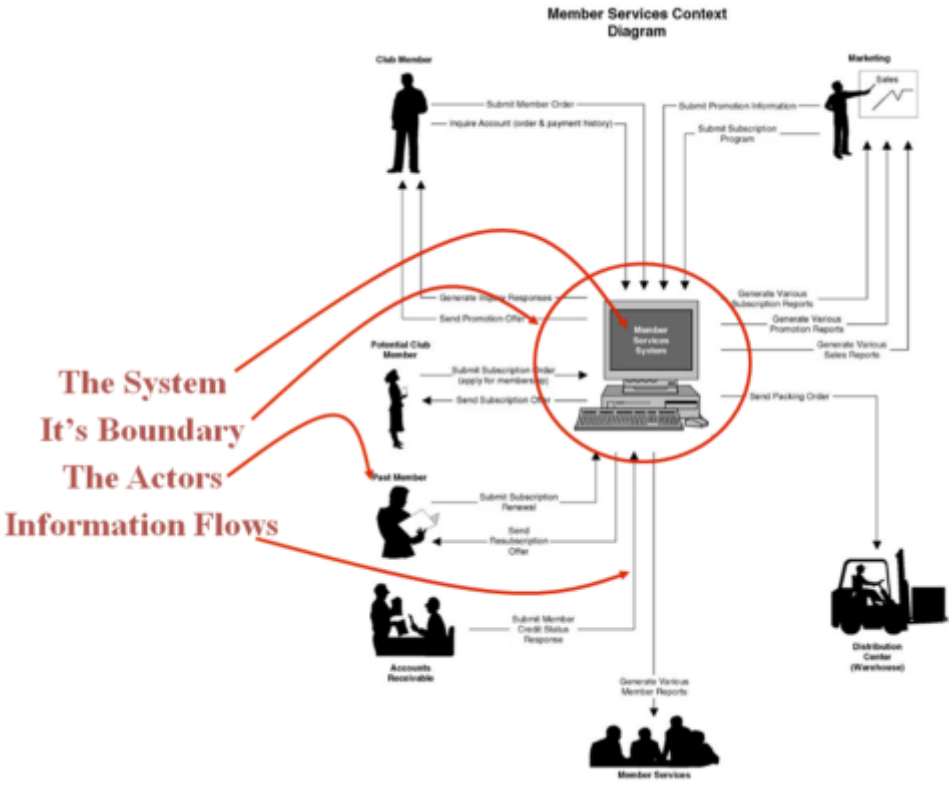

Fig. 4: 一個Context Diagrams範例

figure 4為一個簡單的示範，但在圖示方面可以參考以下的建議：

- System and its boundary: Oval
- Actors: Human mark
- Information Flows : lines with arrows
- Arrows from actor to the system represent actors supplying information to the system
- Arrows from system to the actor represent the system returning information to the actor
- Double-headed arrows represent both actor and system exchanging information with each other
- The arrows should contain text with a brief description of the information flow being exchanged

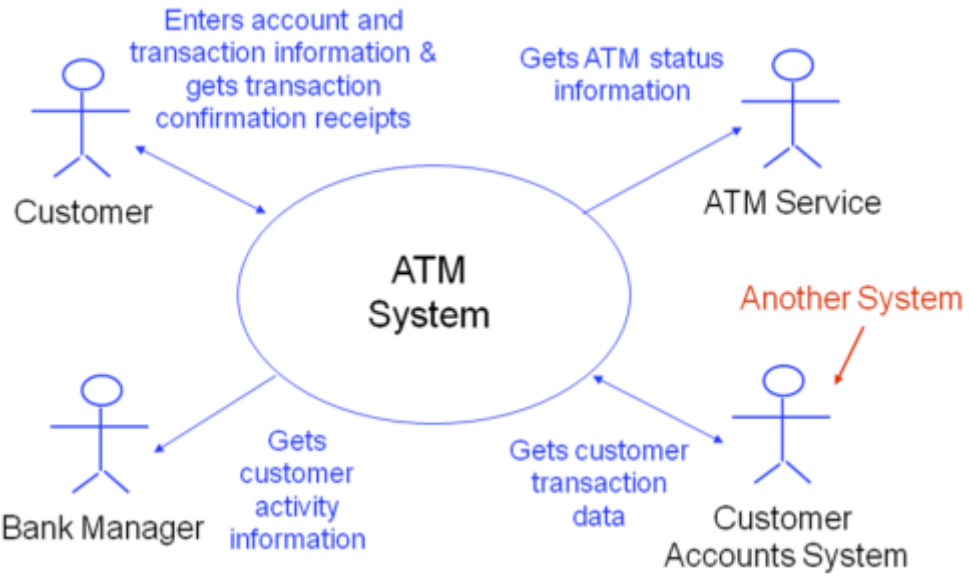

Fig. 5: 採用標準圖示的Context Diagram範例

以我們欲開發的ezERD為例，其Context Diagram可以表達如figure 6

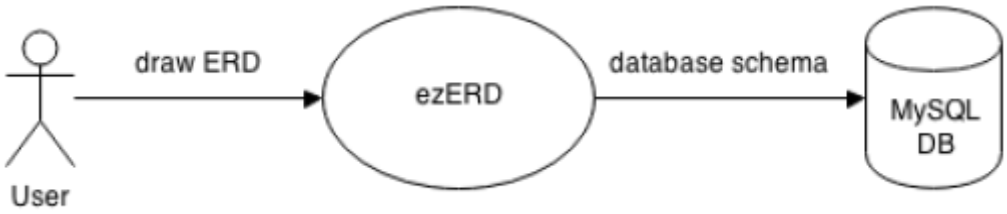

Fig. 6: ezERD的Context Diagram

figure 6顯示了使用者可以透過ezERD完成ERD的設計，並可輸出對應的資料庫綱要給MySQL資料庫。

3.0.5 Use Case Diagrams

- Introduced in 1986 by Ivar Jacobson.
- An important aspect of the UML
- Help us to find and record the system's behavioral (i.e., functional) requirements (not all)
- Use cases describe a system from the actors' perspective
- Use case modeling
  - Identify system boundaries
  - Identify system actors:
    - something (human or non-human) that has behavior and interacts with the system
  - Identify actors' goals for the system
  - Identify business events that fulfill these goals
  - Model (success/failure) scenarios for these events
  - Write use cases to document these scenarios

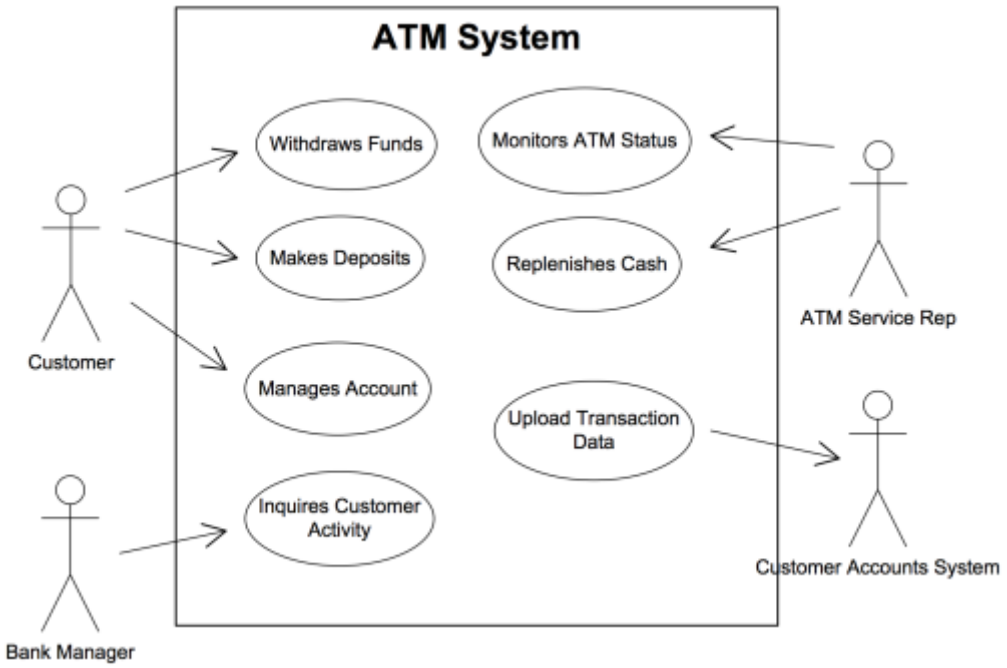

Fig. 7: Use Case Diagram範例

3.0.5.1 Actors

- An actor is "anything outside of the system that exchanges information with it, including users and other systems"

- An actor is “an entity that interacts with the system for the purpose of completing an event
- Actors play roles (e.g., client, salesperson, etc.)
- An actor is not a person, but the role the person plays
- A person can have many roles (e.g., user, manager)
- And a role can be played by many persons (e.g., clients)
- An actor can be a person or any (non-human) external entity (e.g., external system, device, external service organization) that the system will interact with
- Types:
    - Primary Actor: one who receives value from the system
        - You need to identify all primary actors upfront because the goal of your system is to deliver value to these actors
    - Secondary Actor: one who provides service to the system in one or more use cases
        - i.e., helps create value for primary actors
        - i.e., would not exist without primary actors or system
        - You can discover these later because your system doesn’t need to fulfill their goals
- Personalities
    - Initiator: an actor who initiates events that trigger a use case (e.g., customer places an order)
    - External Server: an actor who provides a service to the system in the use case (e.g., query to a credit bureau to process a loan)
    - Receiver: an actor that receives information from the use case (e.g. IRS receives corporate tax return)
    - Facilitator: an actor that supports another actor’s interaction with the system (e.g., data entry clerks)
- Actor Specification Cards which are defined as follows:

Actor Specification		
Actor Name:		
Type: (Primary/Secondary)	**Personality:** (initiator, external, receiver or facilitator)	**Abstract:** (Yes/No)
Role Description:		
Actor Goals: (no need to fill this box for secondary actors)		
Use Case Involved with: (events that fulfill goals above; a goal will map to one or more use cases; a use case will map to one or more primary actor goals; no need to fill this box initially, only after use cases have been identified.		

- Example

Actor Specification		
Actor Name: Customer		
Type: Primary	**Personality:** initiator	**Abstract:** No
Role Description: A customer is a person who has opened an account with the bank. The customer is the main reason for the existence of ATM machines. The customer interacts with the ATM to obtain convenient banking services at times when he/she cannot or is not convenient to obtain such services at the bank's premises.		
Actor Goals:	* Withdraw cash	
	* Deposit funds	
	* Make transfers	
	* Inquire balances	

Use Case Involved with:	* Withdraw funds
	* deposit funds
	* manage account (make transfers, inquire balances)

以ezERD為例，其使用者的Actor Specification可以寫做如下：

Actor Specification		
Actor Name: 使用者		
Type: Primary	**Personality:** initiator	**Abstract:** No
Role Description: 使用者為ezERD軟體的主要操作者，透過使用ezERD軟體，使用者可以繪製新的ERD並與MySQL資料庫連結建立對應的資料庫綱要		
Actor Goals:	* 繪製ERD	
	* 連結MySQL伺服器	
	* 產生關聯式資料庫綱要	
Use Case Involved with:	* ERD編輯	
	* MySQL資料庫連結	
	* 產生資料庫綱要 (含database選擇或建立)	

3.0.5.2 Use Cases

- Is a complete set of artifacts of the use cases describing how actors interact with a system
- Artifacts are any diagrams, models, cards, tables, documents and other descriptions that define and describe the system requirements
- We will focus on two artifacts initially: diagrams and textual descriptions
- The use case model should fully describe the entire “functional scope” of the application
- Each use case within a use case model has its own smaller functional scope, which encompasses the functional responsibility of the use case
- Collectively, all use cases combined should encompass the entire functional scope of the system (i.e., the whole is equal to the sum of the parts).
- Related Artifacts
    - Context Diagram: not really part of a formal Use Case Model, but it is often included because it provides a great high level representation of the system
    - Use Case Diagram: shows the system boundary, actors and all use cases involved
    - Activity Diagrams: use case descriptions are sometimes diagrammed using this type of UML artifact
    - Use Cases: text descriptions that describe all possible scenarios of how actors interact with the system to deliver the required system functionality (i.e., functional scope).
- Finding Use Cases
    - List all goals for all primary actors
        - (e.g., obtain a loan, obtain banking services)
    - Identify business events that will accomplish these goals (e.g., apply for a loan, withdraw cash)
    - Each business event or goal that yields observable value to a primary actor maps to a use case
    - Actor specification cards are useful for this
    - Goals are not always fulfilled by the system (e.g. ATM has no cash), so Use Cases need to model “sunny day” (i.e., optimistic) and “rainy day” (i.e., pessimistic) scenarios.

3.0.5.3 Use Case Diagram Symbols

- System boundary: rectangle
- Use cases: Ovals
- Actors: Human mark
- Relationships: Line with or without arrows
- Meaning of the direction of arrows
- Arrows have a different meaning in Use Case Diagrams than in context diagrams
- Direction of arrows do not represent information flow.
- Instead, they indicate who initiates the use case.

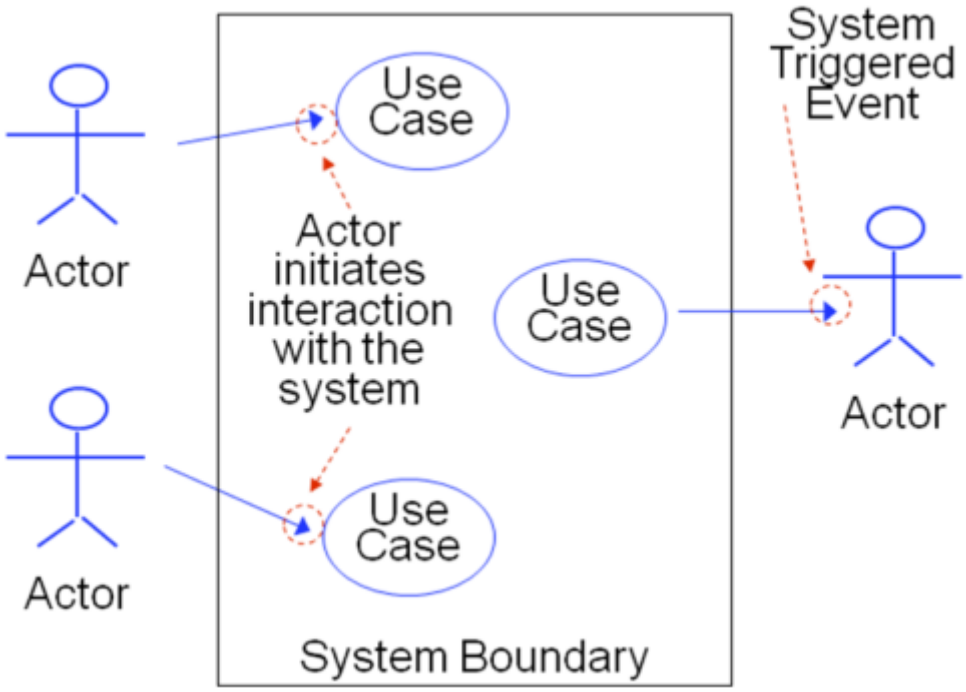

Fig. 8: 使用Use Case的範例1

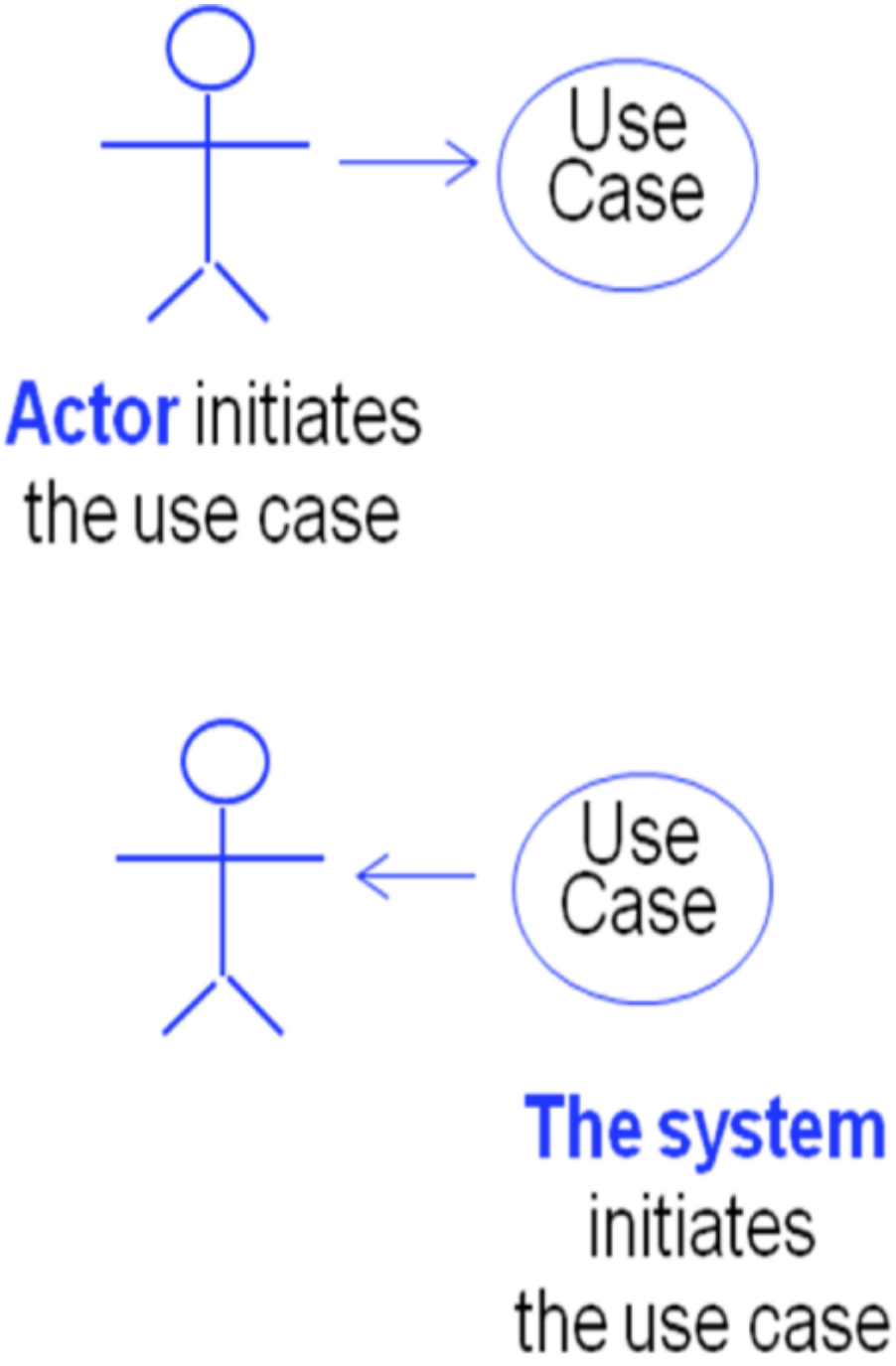

Fig. 9: 使用Use Case的範例2

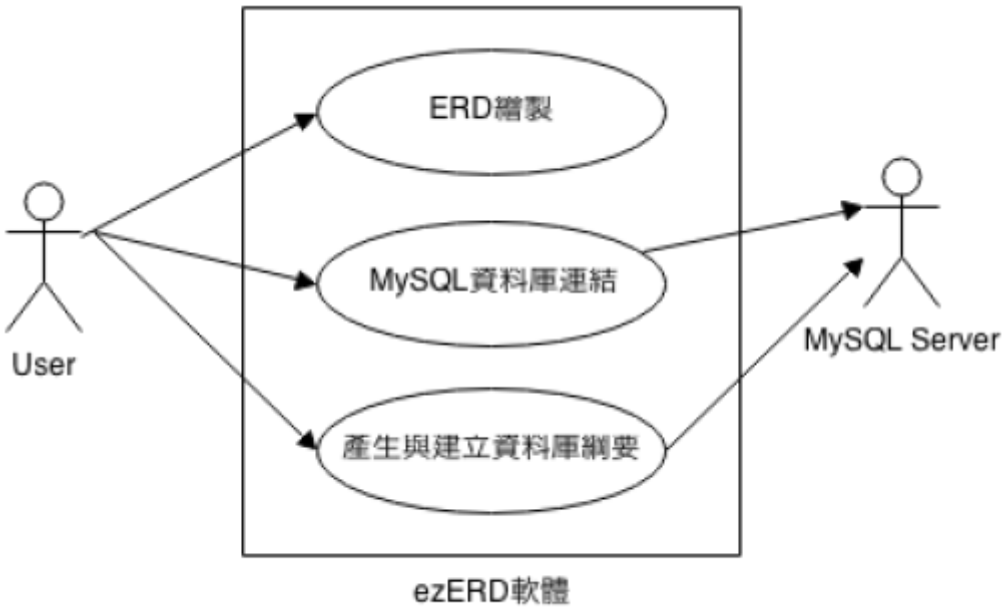

Fig. 10: ezERD的Use Case圖

3.0.5.4 Use case modeling process

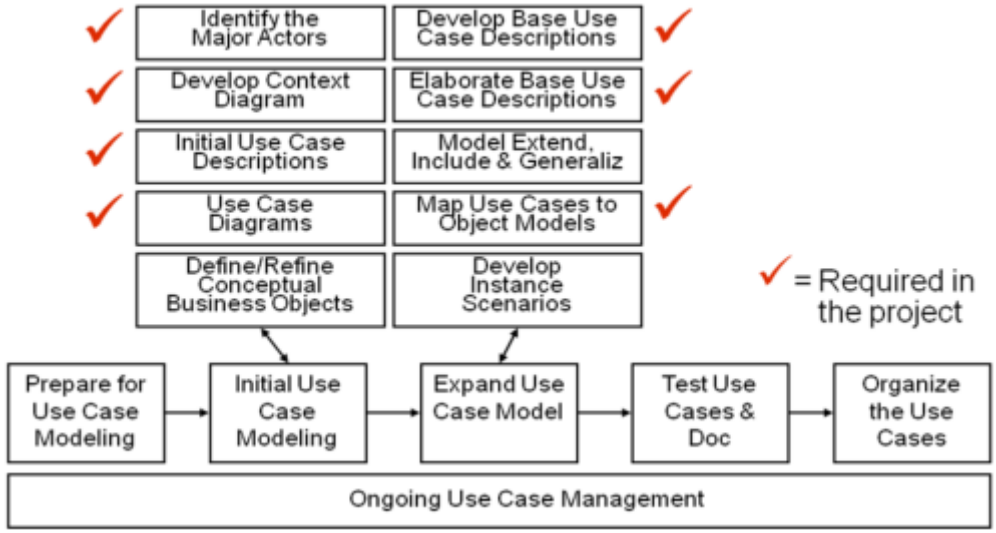

Fig. 11: Use case modeling process

3.0.5.4.1 Initial Use Cases

- Prepare a list of all use cases based on the actor goals identified in all Primary Actor Specification cards
- Per the UP, not all use cases need to be identified at this stage; more use cases will probably be discovered during the elaboration phase
- For each Use Case, fill in a Use Case form to record all the information you have about the responsibilities of that use case
- Focus on general descriptions initially (i.e., Initial Use Cases)
- Per the UP, you will add details later as you further elaborate the use cases
- Use Case Form
    - Contents
    - Use case number and name

- Names of the actor or actors who interact with the system
- Description of the use case (the description can be shortened if needed at this point to avoid redundancy with the flows of events)

Use Case ID	
Use Case	
Actors	
Description	
Priority	(only if you have this information at this point)
Non-Functional Requirements	(only if noteworthy at this point)
Assumptions	(only if noteworthy at this point)
Source	

Example:

Use Case ID	UC-100
Use Case	Withdraw Funds
Actors	(P) Customer
Description	The customer inserts card in the ATM, logs in with a pass code, and makes a selection from the available choices to withdraw funds. Once in the funds withdrawal screen, the customer is prompted to enter the amount to withdraw. After the amount is entered, the system will check for availability of funds for that customer. Provided that funds are available, the system will dispense the amount requested in cash and then debit that amount in the customer's bank account records.
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English
Source	Bank's Operational Procedures Manual

Example for the ezERD:

Use Case ID	ezUC-001
Use Case	ERD繪製
Actors	(P) 使用者
Description	使用者可以透過ezERD軟體進行ERD的繪製。
Priority	Medium
Non-Functional Requirements	必須採用WYSIWYG的方式繪製的向量的ERD且必須符合ERD的規則。
Assumptions	假設使用者不需要存檔的功能
Source	使用者訪談(附訪談記錄)

3.0.5.4.2 Base Use Cases

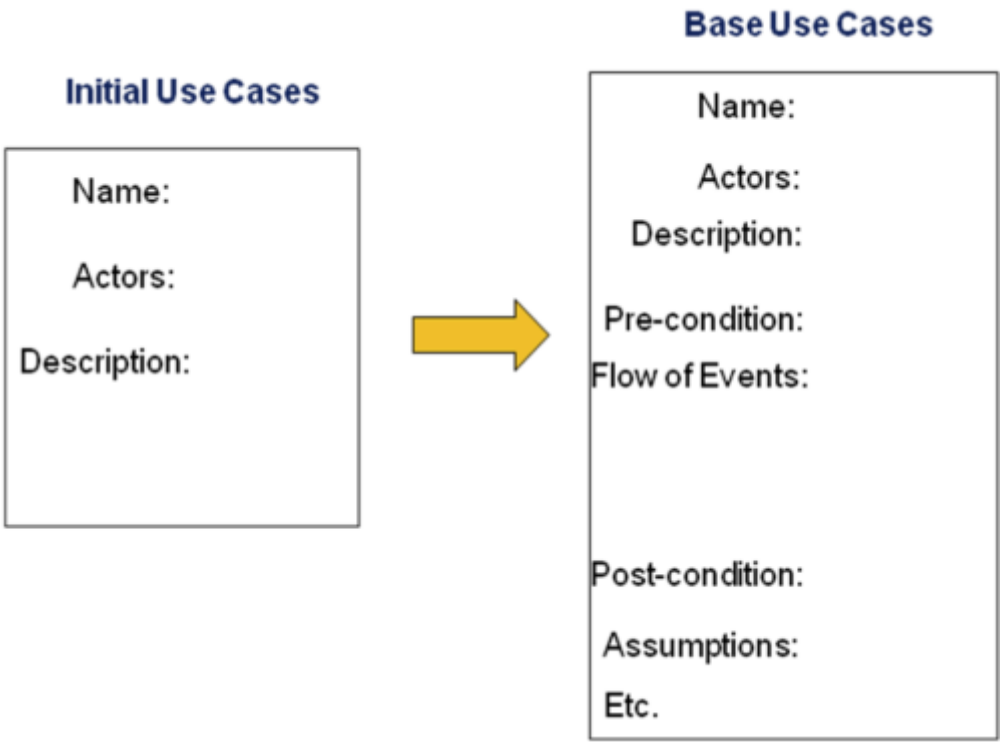

Fig. 12: Base use cases與 initial use case

- The base use cases describe the specific behaviors and interactions that take place between actors and the system, within the scope of the use case.
- An base use case provides a complete description of the normal set of primary interactions between the actors and the system, within a use case
  - i.e., “sunny day”, “success” or “optimistic” scenarios only, i.e., the scenarios that fulfill actors’ goals
  - No need to model “rainy day”, “failure” or “pessimistic” scenarios yet – this comes later
- Purpose
  - Understand the primary interactions between the system and user as well as system behaviors in the use case
  - Provide detailed description of “sunny day” scenarios
  - Begin to identify/document exceptions or alternative scenarios, but don’t model these yet
  - Begin to identify non-functional requirements and assumptions associated with the use case
  - Document the priority of each use case in the development effort
- Contents
  - Same as with initial Use Cases:
    - Use case number and name
    - Names of the actor or actors who interact with the system
    - Description of the use case (the description can be shortened if needed at this point to avoid redundancy with the flows of events)
  - Plus:
    - “Optimistic” flow of events of the use case, i.e., “sunny day” scenarios
    - Pre-Conditions and Post-Conditions of the use case
    - Priority of the use case
    - Known non-functional requirements (e.g., performance, security) specifically associated with the use case
    - Any other assumptions concerning this use case
    - A list of exceptions or alternatives found during the definition of activities in the flow of events

Use Case ID	
Use Case	
Actors	
Description	(brief)
Pre-conditions	
Flow of Events	1.
	1.1
	1.2
	2.
Post-conditions	
Alternative Flows	(briefly described)
Priority	(High, Medium or Low)
Non-Functional Requirements	(only those associated with this use case, if any)
Assumptions	
Outstanding Issues	
Source	

- Developing Base Use Cases
  - Take each initial use case description and expand it
  - Focus on defining the basic or ideal processing flow of an event
  - Usually there is a mapping of one base use case for each initial use case description
  - However, as more is understood about the requirements, multiple base use cases may be discovered as the details of a single initial use case description are better understood
  - The use case model can be simplified by splitting complex and long use cases into smaller, simpler and more manageable use cases
- The Flow of Events
  - The flow of events describes the basic activities (i.e. behaviors and interactions) that occur during the interactions between the actors and the use case
  - It records the order in which activities are performed
  - Activities are documented as a series of steps
  - Some use numbered steps, others use free text
  - Numbered steps establish reference and sequence
  - They describe the primary activities in the use case
  - The use case should be written so that users can easily understand and validate them
- Pre-conditions and Post-conditions
  - Important to know the functional SCOPE of a use case’s responsibilities and the implications of a complete use case execution.
  - Use cases do not stand alone, they represent functionality that is performed within the context of other use cases
  - Some use cases depend on others, with one use case leaving the system in a state that is a precondition for another use case to be able to execute
  - The system states that delimit a use case’s functional scope and its place within a larger set of use cases are known as pre-conditions and post-conditions
  - Preconditions describe the state or status the system must be in, prior to the execution of a use case. What must be true before the use case can execute?
  - Postconditions describe the state or status of the system as a result of the execution of the use case.
  - Examples

- To evaluate a loan request, a loan request must have been submitted
- To withdraw funds from an ATM, the customer must have logged in and authenticated
- To process an order cancellation, the order must have been received, but not shipped
- To process a merchandise return, the merchandise must have been shipped

Use Case ID	UC-100
Use Case	Withdraw Funds
Actors	(P) Customer
Description	Customer logs in, selects withdraw funds, enters an amount and receives cash and receipt
Pre-conditions	Welcome screen is on
Flow of Events	1. Customer slides card in and out
	2. Machine prompts customer for password
	3. Customer enters password
	4. System authenticates customer
	5. System presents user with a choice menu
	6. Customer selects Withdraw Funds option
	7. System asks customer to select account
	8. Customer selects account
	9. System asks customer for amount to withdraw
	10. Customer enters amount
	11. System dispenses cash and prints receipt
	12. System logs customer out
Post-conditions	Welcome screen is back on
Alternative Flows	Customer may not be authenticated; customer may not have sufficient funds; machine may not have enough cash
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English
Source	Bank’s Operational Procedures Manual

以我們的ezERD為例，其base use case如下：

Use Case ID	ezUC-001
Use Case	ERD繪製
Actors	(P) Customer
Description	Customer logs in, selects withdraw funds, enters an amount and receives cash and receipt
Pre-conditions	Welcome screen is on

Flow of Events	1. Customer slides card in and out
	2. Machine prompts customer for password
	3. Customer enters password
	4. System authenticates customer
	5. System presents user with a choice menu
	6. Customer selects Withdraw Funds option
	7. System asks customer to select account
	8. Customer selects account
	9. System asks customer for amount to withdraw
	10. Customer enters amount
	11. System dispenses cash and prints receipt
	12. System logs customer out
Post-conditions	Welcome screen is back on
Alternative Flows	Customer may not be authenticated; customer may not have sufficient funds; machine may not have enough cash
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English
Source	Bank’s Operational Procedures Manual

3.0.5.4.3 Fully Elaborated Use Cases

- Elaborating on the Base Use Cases
  - Base use cases provide an excellent perspective of the system, but we need to add more detailed information about the system’s behavior to complete the analysis
  - Base Use Cases describe “sunny day” scenarios that fulfill the goals of the actor(s) involved (e.g., get cash)
  - During the execution of a use case there will be variations such as alternatives and exceptions that occur as a result of the interactions between the actors and the system.
  - Elaborated Use Cases also include “rainy day” scenarios, some of which don’t fulfill actors’ goals (e.g., insufficient funds, no cash in the ATM machine)
  - Later, we will also add Included and Extended Use Cases, and also Generalizations
- What is added in the Elaborated Use Case
  - More details about the activities performed during the flow of events of a Base use case, as needed
  - Conditional and alternative flow of events to document exception and alternative processing as needed
  - Splitting the Base Use Case into two or more narrowly focused Elaborated Use Cases, when the Base Use Case is too broad or complex
  - Adding non-functional requirements specifically associated with the Use Case (e.g., performance, capacity, availability, security), as needed
  - Adding constraints imposed on the Use Case (e.g., must operate on Linux OS and Oracle database)
- Conditional vs. Alternative Flow of Events
  - The are both very similar
    - Conditional flow of events are described within the use case

        - Alternative flow of events are described in separate but related Use Cases
    - When to use Conditional Flow of Events:
        - Variations are key to understanding the Use Case
        - Variations occur frequently
        - Variations are short and simple
    - When to use Alternative Flow of Events:
        - Variations are long and complex
        - Variations have different priorities
- Conditional Flow of Events
    - As more elaborated functionality is discovered, conditional logic can be a useful approach to represent the functional complexity
    - Be sure to keep the conditional logic at a manageable level of detail.
    - The objective is to understand the functionality, not to write the software code
    - Use IF statements to initiate conditional flows; Ex.:

```
7. If ATM machine is out of cash
    7.1 Notify customer of no cash availability
    7.2 Log customer out
    7.3 Notify ATM service group
```

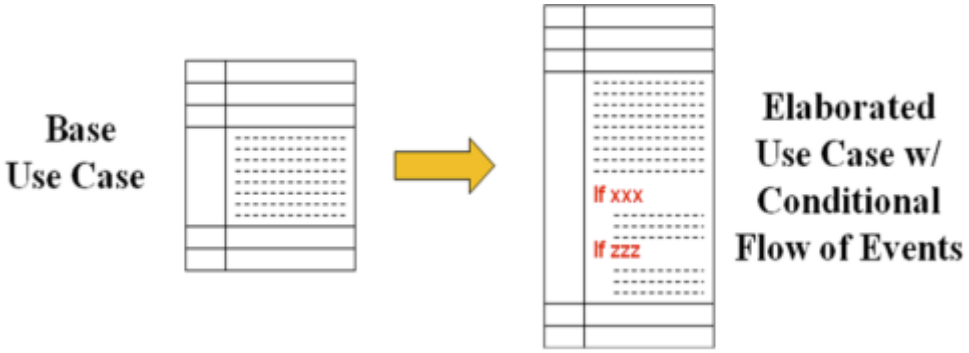

Fig. 13: 從base use case延展至elaborated use case (with conditional flow of events)

- Use case form with conditional flows:

Use Case ID	
Use Case	
Actors	
Description	
Pre-conditions	
Flow of Events	1.
	2. If xx go to 8 (conditional flow)
	3.
	4. If yy go to Use Case UC 101-A1 (alternative flow)
	5. Etc
Conditional Flows	8. (list/describe relevant events and return to 3)
	9. Etc.
Post-conditions	
Alternative Flows	(if simple, briefly describe it; if long or complex, move to a separate Use Case)
Priority	(High, Medium or Low)

Non-Functional Requirements (only those associated with this use case, if any)	
Assumptions	
Outstanding Issues	
Source	

- Another way to describe conditional flows:

Use Case ID	
Use Case	
Actors	
Description	
Pre-conditions	
Flow of Events	1.
	2. If xx (conditional flow listed when it occurs)
	2.1
	2.2
	3.
	4. if yy go to Use Case UC 101-A1
	5. etc
Post-conditions	
Alternative Flows	(if simple, briefly describe it; if long or complex, move to a separate Use Case)
Priority	(High, Medium or Low)
Non-Functional Requirements	(only those associated with this use case, if any)
Assumptions	
Outstanding Issues	
Source	

- Example: Elaborated Use Case with Conditional Flows:

Use Case ID	UC-100
Use Case	Withdraw Funds
Actors	(P) Customer
Description	Customer logs in, selects withdraw funds, enters amount, gets cash
Pre-conditions	Welcome screen is on

Flow of Events	1. Customer slides card in and out
	2. Machine prompts customer for password
	3. Customer enters password
	4. If password is incorrect
	4.1 go back to step 2
	4.2 if password is incorrect 3 times
	4.2.1 Retain card and notify user
	4.2.2 go to step 13
	5. System authenticates customer
	6. System presents user with a choice menu
	7. Customer selects Withdraw Funds option
	8. System asks customer to select account
	9. Customer selects account
	10. System asks customer for amount to withdraw
	11. Customer enters amount
	12. System dispenses cash and prints receipt
	13. System logs customer out
Post-conditions	Customer is logged out an welcome screen is back on
Alternative Flows	Customer may not have sufficient funds; machine may not have enough cash
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English
Source	Bank's Operational Procedures Manual

- Alternative Use Cases

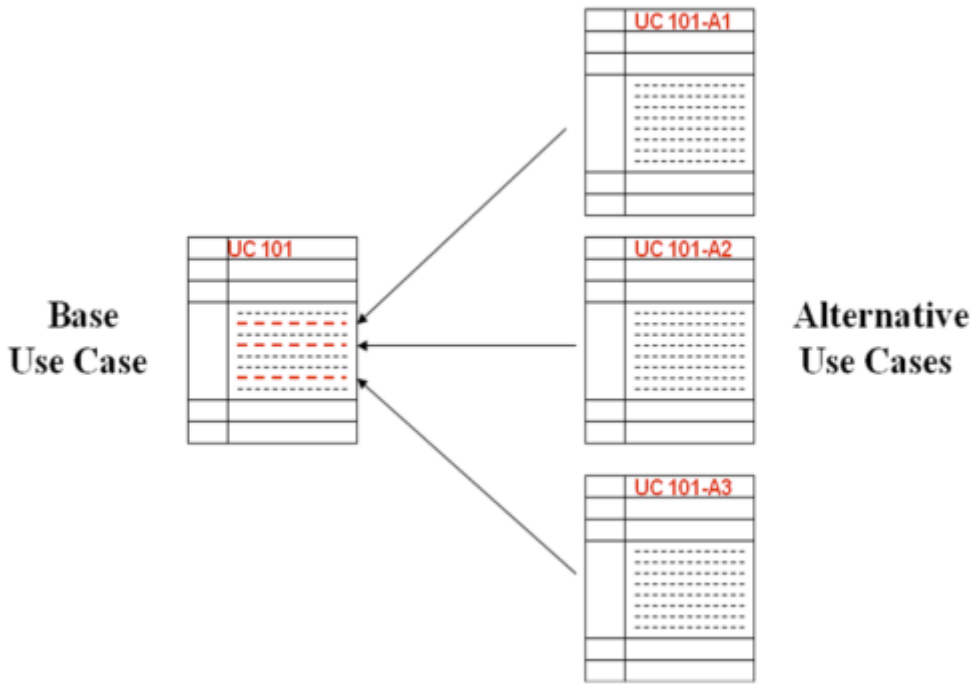

Fig. 14: alternative use cases

- Normally there are several possible variations, alternatives and exceptions that can occur when

a use is executed. Ex.:
    - What if loan applicant has a bad credit history?
    - What if loan applicant doesn't qualify?)
- These alternative behaviors can represent significant functionality, so it is necessary to model the implications of these alternatives
- We do this in Alternative Use Cases
- Which describe the Flow of Events that are triggered when such exceptions occur in the Use Case
- An alternative Use Case is not independent; it is tied to the Use Case that originated it
- Alternative Flow of Events
- Documents the specific behaviors that will occur in an Alternative Use Case
- An alternative Use Case can involve such things as:
- A different processing option based on user input
- A decision taken within the use case flow of events, or
- An exception condition that triggers a different set of behaviors
- Not all alternative events need to go in separate Use Cases; they can be documented quickly and briefly in the base use case description
- The Alternative Use Case has a new field indicating the "Insertion Point" (where it starts executing in the Base Use Case
- Use Case Document for Alternative Flows

- Use Case Form for Alternative Flows

Use Case ID	(same ID as the base Use Case ID, but with suffix A1, A2, etc.; e.g., UC 101-A1)
Use Case	
Actors	(same as Base Use Case's Actors)
Description	
Insertion Point	(step in Base Use Case where this alternative flow should be inserted)
Pre-conditions	(clearly indicate under which conditions trigger the alternative flow)
Alternative Flow of Events	1.
	2.
	3.
Conditional Flows	(within the Alternative flow)
Post-conditions	
Priority	
Non-Functional Requirements	
Assumptions	
Outstanding Issues	
Source	

Example:

Use Case ID	UC-100
Use Case	Withdraw Funds
Actors	(P) Customer
Description	Customer logs in, selects withdraw funds, enters amount, gets cash
Pre-conditions	Welcome screen is on

Flow of Events	1. Customer slides card in and out
	2. Machine prompts customer for password
	3. Customer enters password
	4. If password is incorrect
	<html> </html> 4.1 If card used was reported stolen execute UC-100-A1
	<html> </html> 4.2 go back to step 2
	<html> </html> 4.3 If password is incorrect 3 times
	<html> </html> 4.3.1 Retain card and notify user
	<html> </html> 4.3.2 go to step 13
	5. System authenticates customer
	6. System presents user with a choice menu
	7. Customer selects Withdraw Funds option
	8. System asks customer to select account
	9. Customer selects account
	10. System asks customer for amount to withdraw
	11. Customer enters amount
	12. System dispenses cash and prints receipt
	13. System logs customer out
Post-conditions	Welcome screen is back on
Alternative Flows	Customer may not have sufficient funds; machine may not have enough cash.
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English
Source	Bank's Operational Procedures Manual

Use Case ID	UC-100-A1
Use Case	Withdraw Funds Alternative for Stolen Card Use
Actors	(P) Customer
Description	Notify security when use of a stolen card is detected
Insertion Point	UC-100, flow 4.1
Pre-conditions	User entered incorrect password and card had been reported stolen
Flow of Events	1.Immediately notify bank security
	2.Place a call to local authorities
	3.Take multiple pictures with ATM camera
	4.Interact with user as if it were a normal transaction
	5.Disable cash dispensing functions temporarily
	6.Introduce processing delays to keep customer distracted
	7.Retain card
	8.Timeout if user stops interacting with ATM 30 seconds or more
	9.Timeout if new user inserts a legitimate card
	10.System logs customer out
Post-conditions	Welcome screen is back on
Alternative Flows	

Priority	High
Non-Functional Requirements	
Assumptions	System has local authority notification feature
Source	Bank’s Operational Procedures Manual

3.0.5.4.4 Extended Use Cases

- When developing a use case model, there will be times when significant behaviors will be identified, which extend the behavior of the base use cases
- Extended behaviors are commonly used to model:
  - Future extensions to the system (it is a good development practice to include all known future extensions in the analysis model)
  - Extended versions of the system (e.g., “Premium”, “Professional”)
  - Possible extensions, which can be added or removed later
- These additional behaviors are documented using Extend Relationships and Extended Use Cases
- Advantages of the Extend Approach
  - Helps represent the system’s extended behavior in the Use Case without cluttering, complicating or enlarging the base flow of events.
  - Rather, significant extensions can be drawn out and represented separately
  - The Base Use Case flow does not have to be re-written and the Use Case Model does not need to be re-drawn to reflect this new behavior.
  - As new extensions are discovered they can be added to the model
  - As extensions are discarded they can be easily removed from the model
- Adding Extended Use Cases
  - The Base Use Case is extended at specific points in its Flow of Events named “Extension Points”
  - The extending behaviors are executed at these points when the required conditions for extension are met
  - Control is returned to Base Use Case at the same point in Flow of Events where the extension was triggered

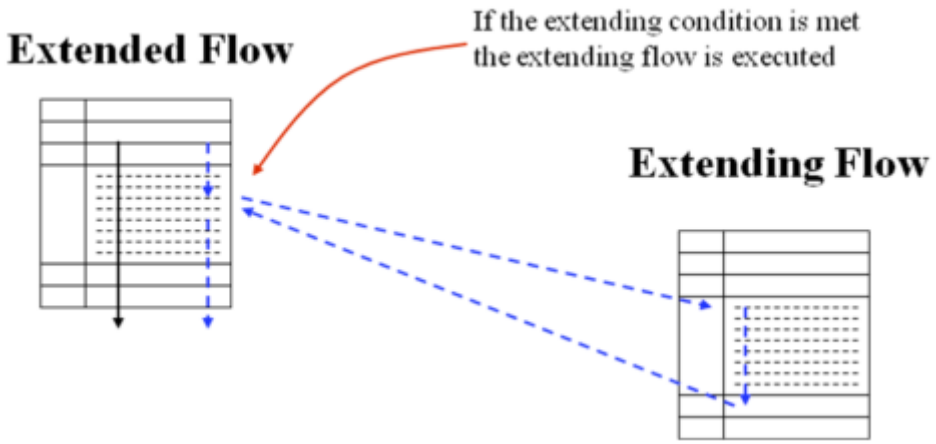

Fig. 15: Extended Flows

- Notation for the Extended Relationship

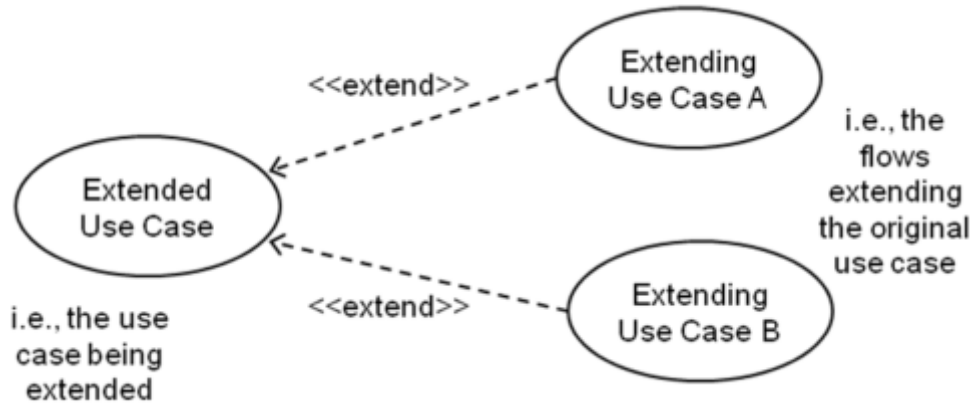

Fig. 16: Extended Use Case 1

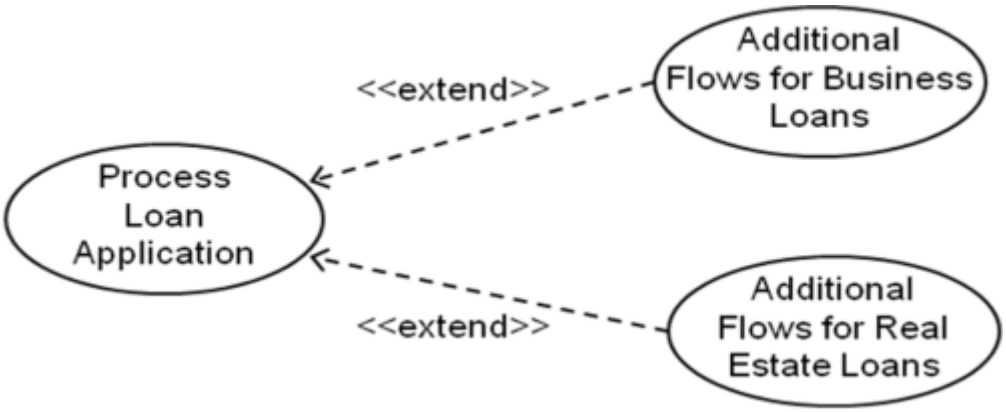

Fig. 17: Extended Use Case 2

- Use Case Form for Extended Flows

Use Case ID	(same as the base Use Case ID, but with suffix E1, E2, etc.; e.g., UC 101-E1)
Use Case	
Actors	(same as Base Use Case's Actors)
Description	
Extended Use Case	
Extension Point	(step in Base Use Case where this extension occurs)
Guard Conditions	(precondition in the Extended Use Case that triggers the extension)
Alternative Flow of Events	1.
	2.
	3.
Conditional Flows	(within the Extended flows)
Post-conditions	
Priority	
Non-Functional Requirements	
Assumptions	
Outstanding Issues	
Source	
Use Case ID	UC-100-E1
Use Case	Print bank statement for a fee
Actors	(P) Customer
Description	Allows customer to print a bank statement with balances and transactions in the last 30 days for a $1 fee
Extended Use Case	UC-100 Withdraw Funds

Extension Point	UC-100; after flow step 12
Guard Conditions	Statement printing option is implemented and enabled
Flow of Events	1. Ask customer if he/she would like a printed bank statement
	2.If customer says yes
	2.1 Notify customer of a $1 charge for this service
	2.2 Prompt customer to continue or cancel
	2.3 If customer selects continue
	2.3.1 Print statement
	2.3.2 Debit $1 from customer's account
	2.3.3 Display thank you note
Post-conditions	Return to UC-100 and continue on flow step 13
Alternative Flows	
Priority	Low
Non-Functional Requirements	Statement must print within 20 seconds
Assumptions	
Source	Bank's Operational Procedures Manual

3.0.5.4.5 Refactoring and Included Use Cases

- The concept of "refactoring" comes from software programming – as software gets corrected, maintained, updated, etc. its code often becomes disorganized and/or suboptimal
- Re-factoring: "is a technique to restructure code in a disciplined way" – Martin Fowler
- It involves re-organizing the software code without changing its functionality, to make it easier to understand and maintain
- Refactoring is applied today to many aspects of system development, not just programming
- In Use Cases, it involves the "Included" Use Cases
- Included Use Cases
    - Included Use Cases provide a way to model similar behaviors that can be used across multiple use cases
    - As the Use Case is elaborated, you may identify flow of events that are repeated in several Use Cases
    - You can extract them into generic "Included" Use Cases, and then "Include" them whenever you need them
    - Examples:
        - User logs in and gets authenticated
        - Check product availability in inventory
    - Advantages of Included Use Cases
        - Identify commonalities among use cases
        - Manage redundancy within the use case model
        - Facilitate change in the use case model – when things change in the Included case, you only need to make the necessary changes once
        - Begin to assemble Included Use Case "Libraries" of common behaviors that can be re-used in other projects

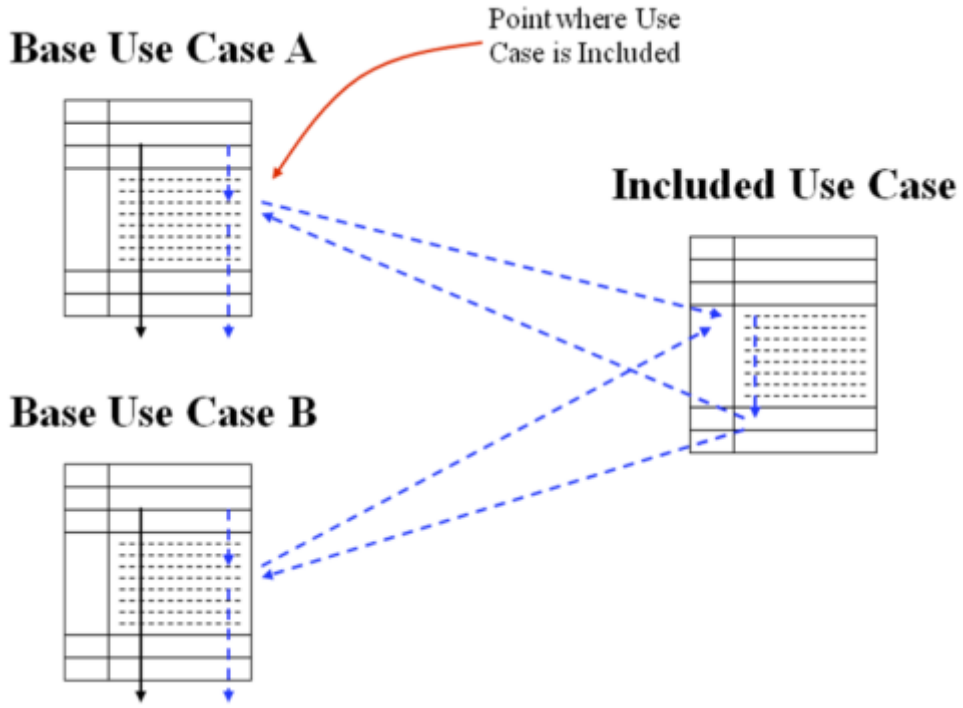

Fig. 18: Included Use Case

* Notation for the Included Relationship

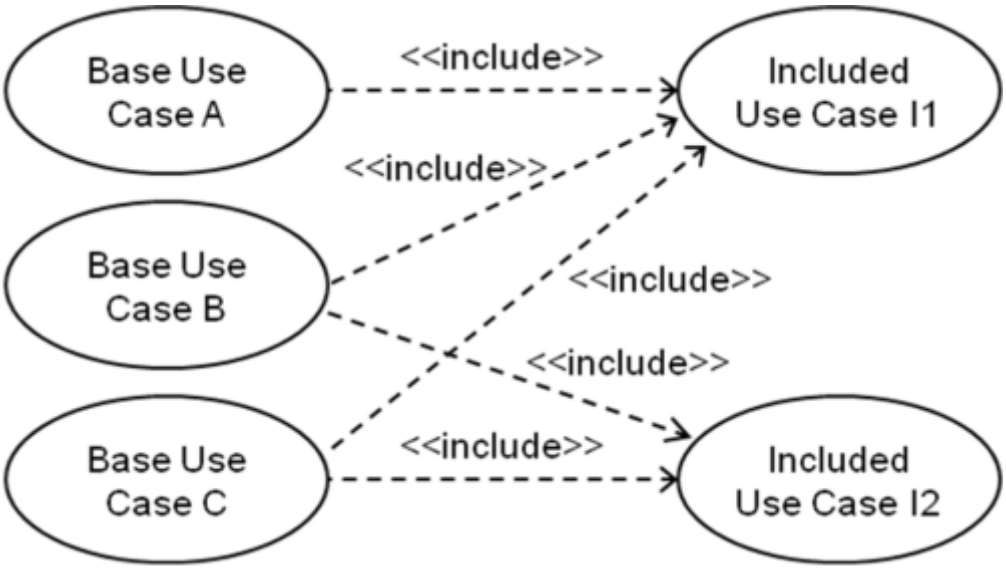

Fig. 19: Included Relationship 1

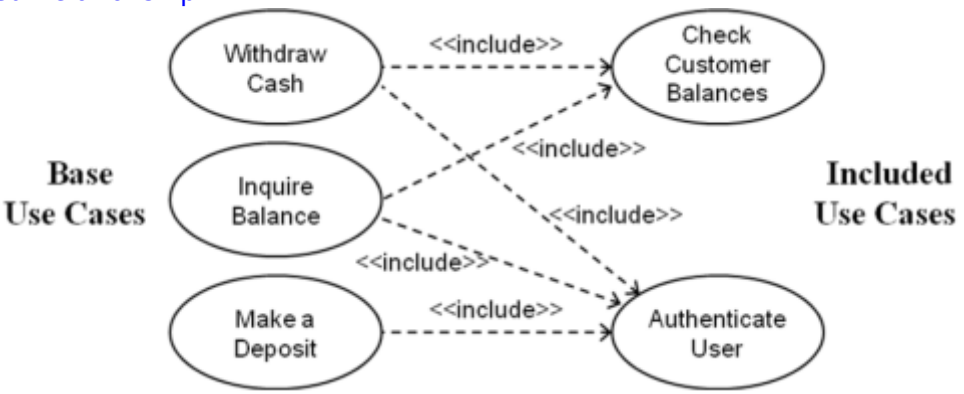

Fig. 20: Included Relationship 2

- Using Included Use Cases

- Identify all Included Use Cases with the prefix “IUC” instead of “UC”
- Indicate in the Base Use Case the point in the Flow of Events where the included use case is included
- In the Included Use Case, document all Base Use Cases that use that Included Uses Case

- Use Case Form for Included Flows

Use Case ID	(use IUC suffix instead of UC; e.g., IUC 001)
Use Case	
Description	
Including Use Cases	(list Use Cases that use in this included Use Case)
Pre-conditions	
Alternative Flow of Events	1.
	2.
	3.
Conditional Flows	(within the included flows)
Post-conditions	
Priority	
Non-Functional Requirements	
Assumptions	
Outstanding Issues	
Source	

- Example

Use Case ID	UC-100
Use Case	Withdraw Funds
Actors	(P) Customer
Description	Customer logs in, selects withdraw funds, enters amount, gets cash
Pre-conditions	Welcome screen is on
Flow of Events	1. Include IUC-001 Log in and Authenticate Customer
	2. If not successful, go to step 10
	3. System presents user with a choice menu
	4. Customer selects Withdraw Funds option
	5. System asks customer to select account
	6. Customer selects account
	7. System asks customer for amount to withdraw
	8. Customer enters amount
	9. System dispenses cash and prints receipt
	10. System logs customer out
Post-conditions	Welcome screen is back on
Alternative Flows	Customer may not have sufficient funds; machine may not have enough cash.
Priority	High
Non-Functional Requirements	Cash dispensed within 10 seconds after amount is entered
Assumptions	Customer speaks English

Source	Bank's Operational Procedures Manual
Use Case ID	IUC-001
Use Case	Login and Authenticate Customer
Description	Customer logs, gets authenticated, card is checked against stolen reports
Including Use Cases	UC-101 Withdraw Funds; UC-102 Deposit Funds; UC-103 Other transactions
Pre-conditions	ATM has detected a card in the slot
Flow of Events	1. Customer slides card in and out
	2. Machine prompts customer for password
	3. Customer enters password
	4. If password is incorrect
	<html> </html> 4.1 If card used was reported stolen execute IUC-001-A1
	<html> </html> 4.2 go back to step 2
	<html> </html> 4.3 If password is incorrect 3 times
	<html> </html> 4.3.1 Retain card and notify user
	<html> </html> 4.3.2 System logs customer out and ends this transaction
	5. System authenticates customer
Post-conditions	System is back on the next flow step right after this included case was invoked
Alternative Flows	
Priority	High
Non-Functional Requirements	Authentication should take place within 20 seconds after password entered
Assumptions	Customer speaks English
Source	Bank's Operational Procedure Manual

3.0.5.4.6 Generalization Relationships

- “implies that the child Use Case contains [i.e., inherits] all the attributes, sequences of behavior, and extension points defined in the parent use case, and participates in all relationships of the parent use case.”
- Like classical generalization and inheritance
- A more powerful version of “Include”
- Notations

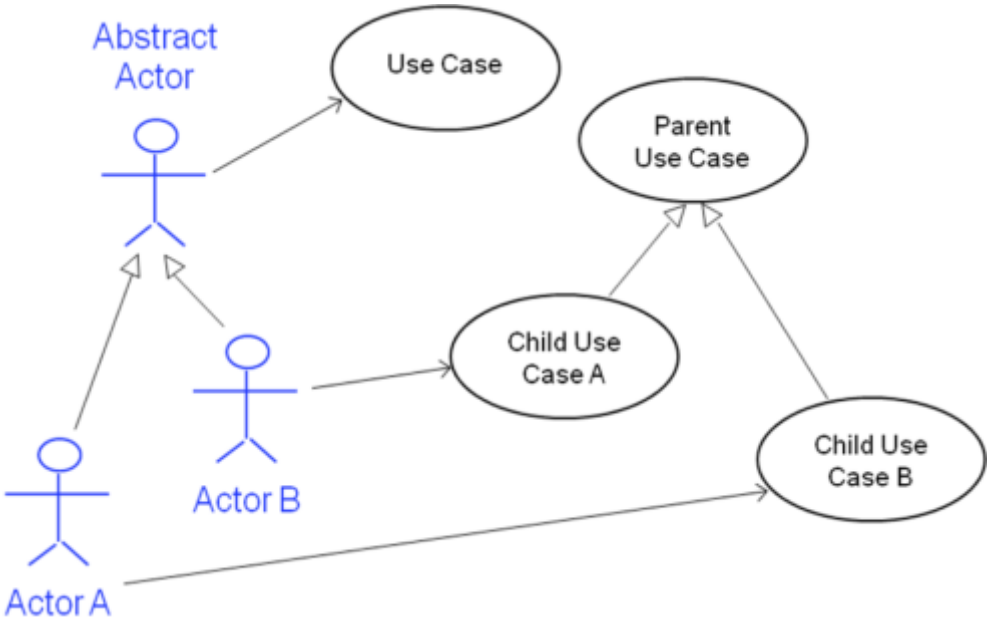

Fig. 21: Generalized Use Case 1

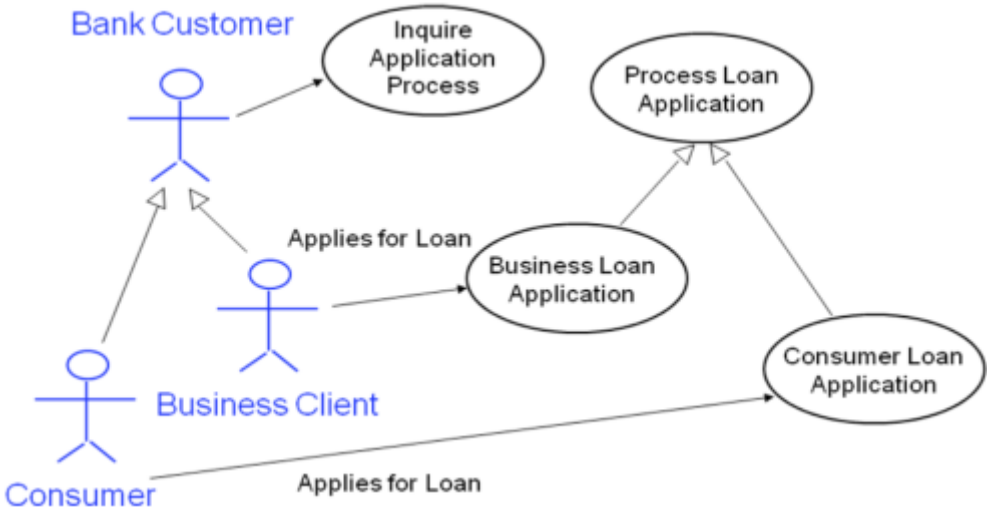

Fig. 22: Generalized Use Case 2

3.0.5.5 Important Rules about Use Case Models

- Alternative Use Cases don't need to be diagrammed –> they an integral part of the originating Use Case (if you decide to diagram them, apply the same rules for Extended Use Cases)
- Use Cases should not connect with other Use Cases in the diagram
- Actors should not connect with other Actors in the diagram
- Only Actors connect (interact) with Use Cases
- But there are 3 exceptions:
  - Extended Use Cases connect with their respective extending Use Cases
  - Included Use Cases connect with the Use Cases that include them
  - Use cases can connect to other use cases and actors can connect to other actors when there is a "generalization" relationship
- Extended and Included Use Cases may or may not connect directly with Actors
- Extended and Included Use Cases don't "stand alone" à they are always associated with another use case.