國立屏東大學 資訊工程學系 物件導向軟體工程 (last updated September 6, 2021)

2. 軟體生命週期與塑模

軟體的開發從構思開始，歷經設計、實作、測試、上線等階段，直至最後不再繼續使用為止的過程，被稱之為軟體的生命週期(software development life cycle，SDLC)

<note> 軟體生命週期又稱為「軟體生存週期」或「軟體開發生命週期」 </note>

傳統的軟體工程，將一個軟體的生命週期概分為問題定義、可行性分析、總體描述、系統設計、實作、測試、運行、維護到廢棄等不同階段(phase)，在每個階段應該按部就班依序進行、推進。為了確保軟體的質量，在軟體生命週期中的每個階段都應該要有明確的定義，才能在推動相關工作時有所依據。為了呈現或描述軟體在生命週期中的過程，我們可以使用適當的模型來加以表示 – 我們將其稱為軟體塑模(modeling)，本課程後續將就軟體塑模以及常見的軟體生命週期模型加以介紹。

2.1 軟體塑模(Software Modeling)

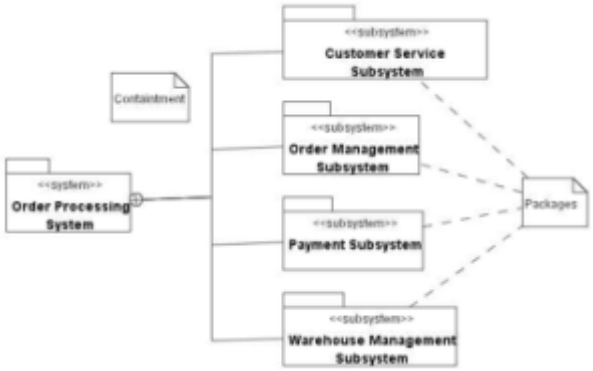
大部份的人應該都有「玩過模型」的經驗吧？就算沒有，至少也看過「模型」吧！模型通常是「真實或虛擬世界」中物品的「某種程度」的縮影，例如F16戰機或是鋼彈的縮小比例模型。依據不同的目的，模型所呈現出的是「真實或虛擬世界」中物品的某些不同面向；例如最簡單的「鋼彈模型」可能呈現的只是外在的樣貌，但是還可以「活動、變身」的「變型鋼彈模型」則除了外觀外，還必須考慮到部份的機械設計原理。建築業也習慣在開始動工前，先做出標的物的模型，好讓人瞭解到所要蓋出的是什麼樣的房子；而且依據不同的需求或對象設計也會使用不同的模型加以表達，例如一般的消費者(購屋者)只需要縮小比例(甚至比例不完全正確也沒關係)的房屋外觀模型以及樣品屋(針對室內空間規劃與裝璜擺設的展示)就可能足夠了；但是，對於實際要參與施工的營造團隊來說，模型的比例不但要「絕對正確」，同時連內部的管線配置、機電配置等不同工務，也應該要在模型內加以呈現 – 這又引申了另一個事實：「對於一個標的物來說，一個模型通常不能呈現所有不同的觀點」，所以我們必須針對不同的目的，建構出不同的模型才能正確地表述。在這段說明裡，我們使用了「模型」一詞，但要注意的是，不是所有的「模型」都長得像「模型」一樣！在很多情況下，模型可能只是一張圖表、甚至只是一段文字的描述，例如針對一個物品，可以使用其「平面圖」、「立面圖」、「俯視圖」來組合呈現其真實的面貌。

在軟體的開發過程中(也就是其生命週期中的每個階段)，也需要使用各種「模型」來呈現各種不同的觀點，才能滿足不同人的不同需求(也就是讓不同人理解軟體的不同面向)。所以，在軟體開發的過程中，軟體工程的重要工作就是「製作出不同的模型」來滿足不同的需求，我們把這個過程稱之為軟體塑模(software modeling)

<note> 模型的英文是model，而打造模型的過程可以用其動名詞型式表達為modeling，所以我們取其塑造模型之意，將modeling譯為「塑模」 </note>

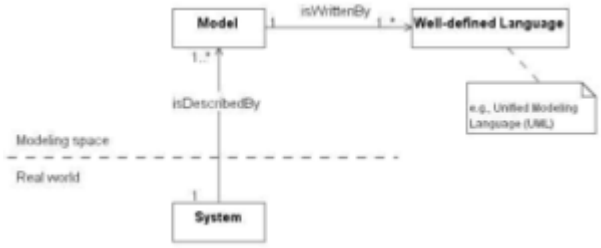
我們可以把模型視為是軟體系統的整體或某一部份的「簡化(simplified)」表達方式，且在生命週期中的不同階段需要不同程度或不同觀點的簡化 – 我們將其稱之為「抽象化(abstration)」，透過這些軟體模型，我們將可以在開發階段中更為瞭解「未來所要開發出來的軟體系統」的樣貌；當然，模型的選擇對於軟體的開發非常重要，因為不適當的模型不但無法讓我們理解軟體的樣貌，甚至還有可能會帶來誤解。請參考figure 1，一個預計要開發的軟體系統通常還可以再分解成數個子系統(Subsystem)，換句話說，一個軟體系統可以被視為是由多個子系統所組合，其中子系統也還可以再由更小的子系統組成。

Fig. 1: 系統與子系統



每一個系統(以及子系統)，又可以使用代表不同觀點或不同對像的多個模型來加以呈現，請參考figure 2:

Fig. 2: 系統可以擁有不同的模型



如果能使用適當的模型來表達軟體生命週期中的過程，那麼就可以將軟體的構思、設計、開發、測試與維護等工作加以抽象化、簡單化，讓人們能夠正確地理解軟體的樣貌。所以，針對在軟體生命週期中的每個階段，提出應該要使用那些不同的模型，就被稱為是「軟體生命週期模型(software development life cycle model)」，我們將在接下來的下一小節進行說明。

2.2 軟體生命週期模型(Software Development Life Cycle Model)

軟體生命週期模型(software development life cycle model)是定義用以描述與軟體開發相關的需求收集、分析、設計、執行及測試、操作與維護等生命週期階段的模型，應達成以下的7個主要的目標[1)]：

- Effectiveness(有效性) – 必須能夠幫助我們產出正確的成果;
- Maintainability(可維護性) – 可以快速且簡易地找出與改稱錯誤；
- Predictability(可預測性) – 必須要能夠幫助我們規劃或評估工作內容；
- Repeatability(可重複性) – 應該要能夠應用在未來的其它專案中；
- Quality(品質) – 必須要能產出高品質的軟體
- Improvement(可改進) – 應該要能夠隨著快速改變的開發環境的改變與軟體的需求而加以改進
- Tracking(可稽核) – 應該要能讓軟體開發相關人員用以追蹤專案的狀態

本節後續將介紹一些著名的軟體生命週期模型：

2.2.1 瀑布模型(Waterfall Model)

瀑布模型(waterfall model)是由Winston W. Royce於1970年提出[2)]，他將軟體工程師的工作視為一系列的階段，在每個階段完成前，必須通過驗證來確保品質 – 包含verification(確認工作正確)與validation(確認工作滿足需求)。瀑布法所列舉的階段包含：

- Requirements Gathering(需求收集階段):收集軟體需求 – 瞭解使用者要些什麼？
- Analysis(分析階段)：依據需求產生模型、制定處理流程、邏輯與商業法則 – 定義該做些什麼？
- Design(設計階段)：產生軟體架構 – 決定該如何作！

- Implementation(實作階段):發展軟體 – 動手寫程式，把軟體開發出來！
- Testing(測試階段):找出軟體缺陷並加以修正 – 確認做出來的是正確的！
- Operations and maintenace (操作與維護階段):包含安裝、建置、整合讓軟體能開始運作，以及進行維護 – 拿去用吧！

figure 3顯示了上述這些階段的流程：

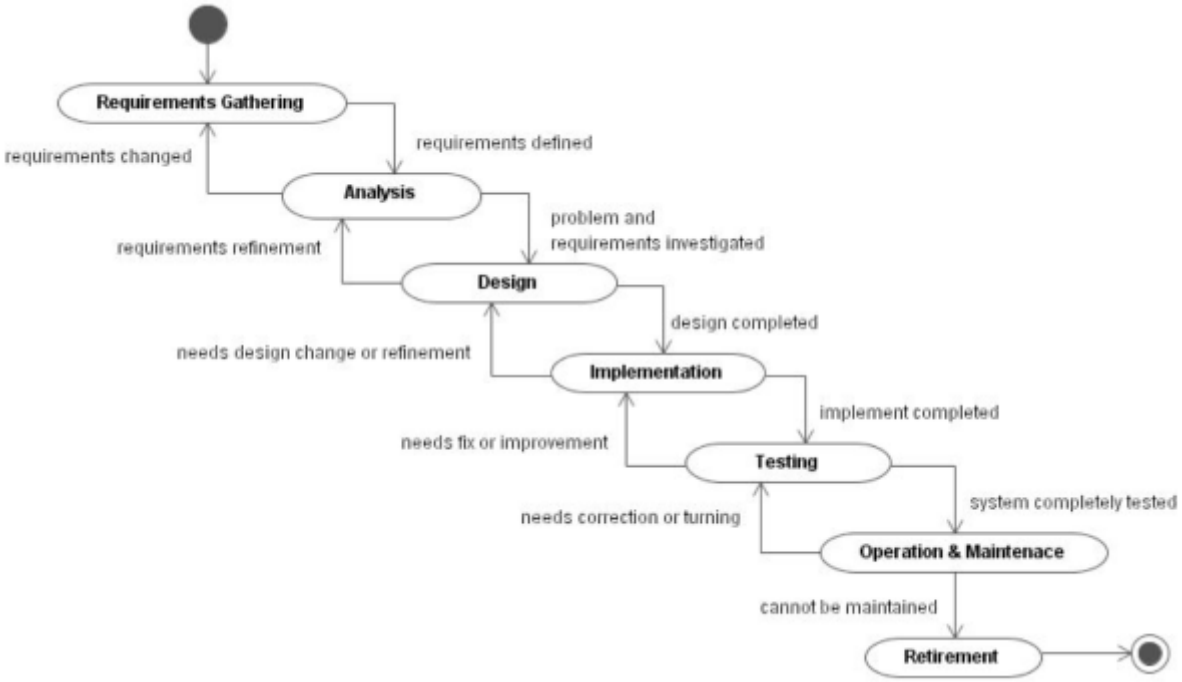


Fig. 3: Waterfall Model

瀑布模型只是一個參考模型，並沒有硬性規定該如何進行，例如figure 4顯示了一個類似(但有一些細節上的差異)的模型，其中的V&V代表的就是在每個階段結束時應該要進行的verification(確認工作正確)與validation(確認工作滿足需求)：

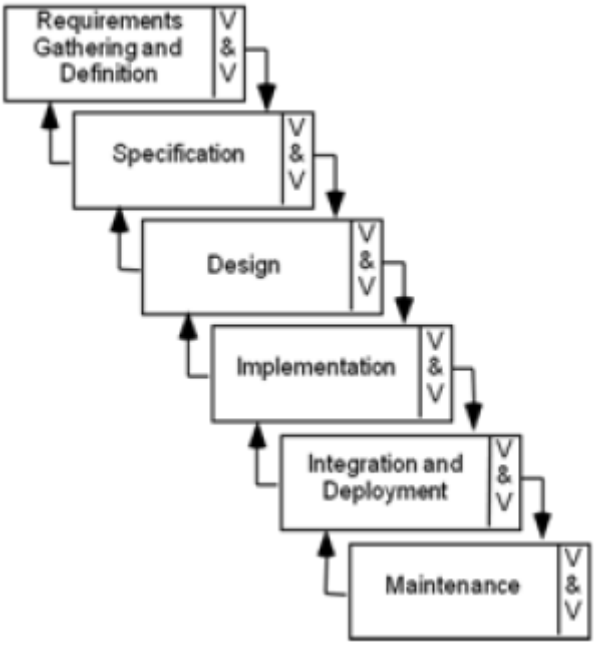


Fig. 4: Waterfall Model

使用瀑布模型的好處是顯而易見的，由於將軟體的開發明確地區分為數個階段，且每個階段都有明確的產出並進行結果的驗證(V&V)，所以只有在確保達成每個階段目標後，才能進行新的階段；否則將回到前一階段進行修正。我們也將這種階段性的驗證稱為milestone apporval(里程碑核可)。但另一方面，瀑布模型也存在著一些缺點：

- 一定要先完成當前階段，才能進入下一階段
- 未能反映不斷改變的需求
- 未完成最後階段前，使用者都無法使用軟體的任何功能(並沒能先產生雛形系統供使用者使用或確認)

<note tip> 請同學討論一下，瀑布模型還有哪些其它的優點或缺點 </note>

2.2.2 階段性發佈模型(Phased-Release Model)

階段性發佈模型(phased-release model)採用漸進式開發(incremental development)，在軟體開發期間，就可以讓使用者先使用部份的軟體功能 – 當前期的需求收集、分析與設計完成後，軟體專案就可以再分解為數個子專案(亦可稱為階段phase)進行開發，且每個子專案完成後就可以先提供給使用者使用。相較於瀑布模型，使用階段性發佈模型可以讓使用者更早地先使用部份功能，但仍然受限於必須在前期先完成需求的確認。階段性發佈模型可以參考figure 5，但其中前期階段被稱為requirement gathering and definition，specification與planning。

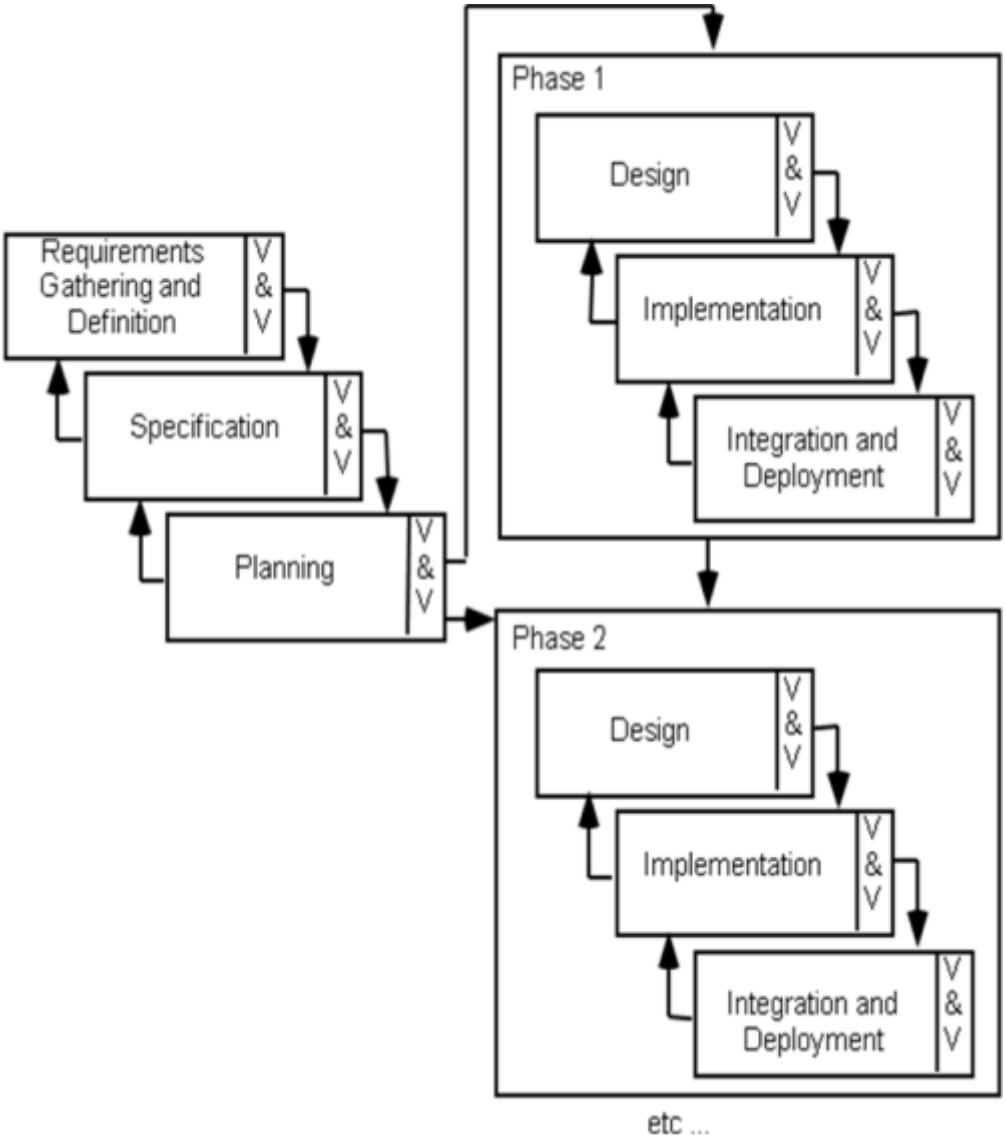


Fig. 5: Pahsed-Release Method

<note tip> 若將階段性發佈模型所切割出的子專案(或階段)，採用平行或是pipeline的方式運作，軟體開發還可以更有效地使用人力資源，進而縮短開發期程 </note>

2.2.3 快速雛形模型(Rapid Prototyping Model)

快速雛形模型(rapid prototyping model)又稱為快速原型模型，在軟體專案開發的初始階段，儘可能快速地建造出軟體雛形，藉以和使用者確認其需求；在使用雛形和使用者確認的過程中，不符合需求的雛形將會被捨棄、部份符合的雛形則反覆修改直到符合使用者的需求為止。雛形的功能與內部結構並不重要，重要的是必須快速地建立雛形，並且在確認的過程中，隨著需求的改變快速地修改以反映客戶的需求為何。

綜合上述，雛形的主要目的是用以展示我們對使用者需求的認知，並且用以和使用者進行確認。雛形通常僅會完成部份的軟體功能，尤其是圖形使用者介面(graphic user interface GUI)可以幫助使用者瞭解我們打算開發的軟體長得怎麼樣？是否和使用者的想法一致？這是一種有效且實際的方式以得知或確認使用者的需求。

快速雛形模型的好處是對於使用者的需求可以有比較正確的掌握，同時經確認後的雛形的使用者介面與功能，可以保留到最終所開發出的軟體裡；且經雛形確認後的使用者需求通常在後續的開發中較不會再變動，不用像瀑布模型或階段發佈模型一樣，常會因需求的改變而反覆地回到前面的階段。但是缺點也非常明顯：

快速雛形模型必須投入許多資源在製作雛形(不正確的雛形就等於錯誤的資源投入)，而且雛形的製作並沒有什麼一定的方法與標準，就好像是「邊射箭、邊找靶」一樣，也很容易陷入「找不到靶」的困境。不過軟體產業界還是偏好使用快速雛形模型(尤其是使用者介面的雛形)，因為經確認後的雛形可以幫助我們正確地理解使用者的需求，也能夠幫助使用者理解最終開發出的軟體之樣貌 – 可以減少軟體結案時的糾紛。

2.2.4 螺旋模型(Spiral Model)

螺旋模型(spiral model)是由 Barry Boehm 於 1986 年所提出[3]，為瀑布模型的替代方法，它兼顧了快速雛形模型反覆修改雛形的漸進式方法與瀑布模型的分階式嚴格管控特性。與前述模型最大的差異是，螺旋模型引入風險分析，可以在必要時停止專案進行停損 – 在每個反覆階段建構雛形並加以評量就是其用以降低風險的手段。

螺旋模型在開始進行軟體專案開發時，先製作小型的雛形，再使用瀑布模型的方法，但僅進行其中的需求收集階段(因此又稱為微瀑布mini-waterfall)並對初次建構的雛形進行驗證；接著依據前次所得到的使用者需求修改雛形，但這次要進行包含需求收集、設計、實作與驗證等較完整的瀑布模型階段；後續就依照前述的方法，反覆進行，不過在每此展開一個新的回合時，必須先進行風險分析。請參考figure 6，我們將每一個回合稱為一個迭代(iterative)並且可以將其視為是一種演化式迭代的軟體開發模型。

螺旋模型的優點是每一次迭代所產生的雛形，可以在下一個迭代中再應用，且每次迭代都經過驗證，因此最終的軟體不需要再進行太多的測試。不過要注意的是螺旋模型僅適用於較大規模的軟體開發。

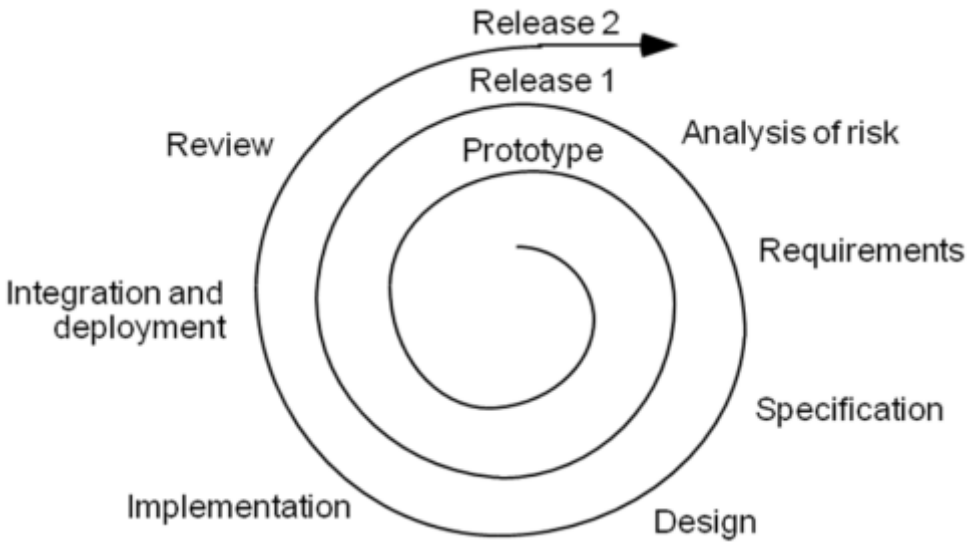


Fig. 6: Spiral Model

2.2.5 同步工程模型(Concurrent Engineering Model)

同步工程模型(concurrent engineering model)是採用分治法(divide and conquer又譯做分割與征服)的概念，請參考figure 7，在初始階段將軟體切割為多個元件(或層次)後，由多個團隊負責不同的元件同時加以開發(可以各自使用相同或不同的開發方法)，最終經整合(或定期進行整合)後，完成軟體的開發。對於縮短軟體開發時程相當有幫助，但缺點是所需的人、物力資源甚鉅。

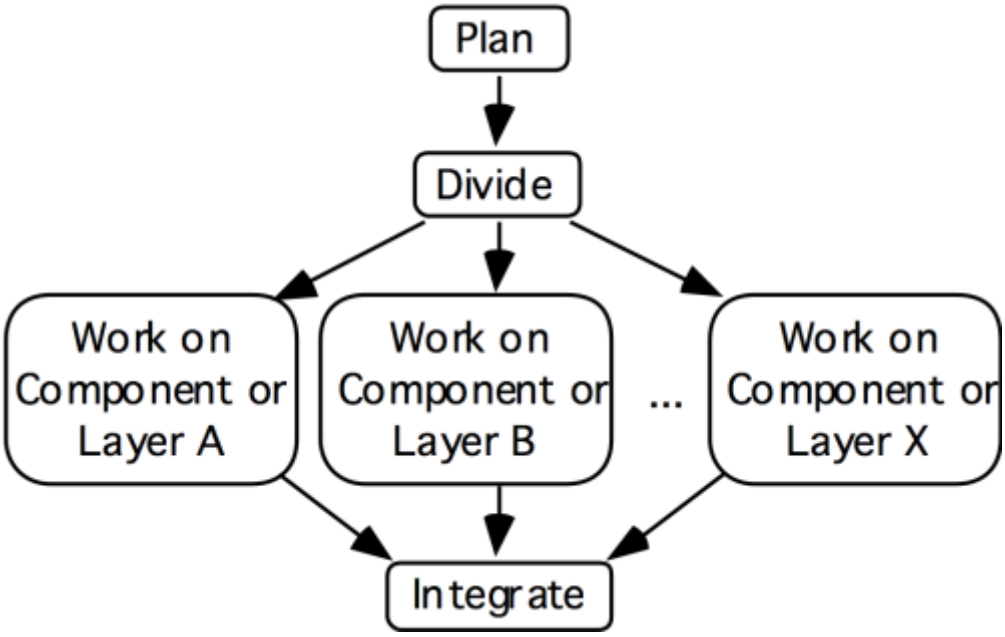


Fig. 7: Concurrent Engineering Model

2.2.6 統一軟體開發程序(Unified Software Development Process)

統一軟體開發程序(unified software development process)常簡稱為unified process(UP)，是一種應用物件導向技術的軟體開發程序架構(software development process framework)[4] – 使用UML做為發展軟體的模型，是一個迭代(iterative)且漸進的(incremental)軟體發法程序架構。要特別注意的是，UP本身並不是一個程序，而是一個程序架構：是一個可針對特定組織或軟體專案設計客製化的程序的一個架構。由於一開始UP是由Rational軟體公司所提出，所以又常被稱為Rational Unified Process(RUP)。

<note> 如前述，UP只是一個架構，任何人都可以依據UP提出自己的版本，RUP是由Rational軟體公司(自2003年起，為IBM所併購)所提出的一個UP版本，同時也是最知名的一個版本。可千萬不要被UP與RUP相近的名稱給混淆了，UP是一個軟體開發程序的架構，我們可以使用UP架構來發展自己的軟體開發程序，RUP就是一個例子，它是由Rational公司所提出的一個UP架構實作。 </note>

UP架構是由階段(phases)與程序(processes)所組成，其中階段分為：

- Inception(初始): 訂定專案願景(Vision)(專案範圍)
- Elaboration(細化): 決定要做什麼與需要什麼，以及決定系統架構
- Construction(建構): 進行產品開發建構
- Transition(轉移): 將產品發佈給使用者

figure 8顯示了UP架構的四個階段所需的資源與耗費的時間。figure 9進一步顯示了UP架構的程序(processes)，同時它也將elaboration，construction與transition階段切割成數個迭代(iterations)(對於較大型的專案來說，inception階段也可以切割為數個迭代)，每個迭代將會以漸進式的方式，以前一個迭代的產出為基礎進行修改或增加功能。

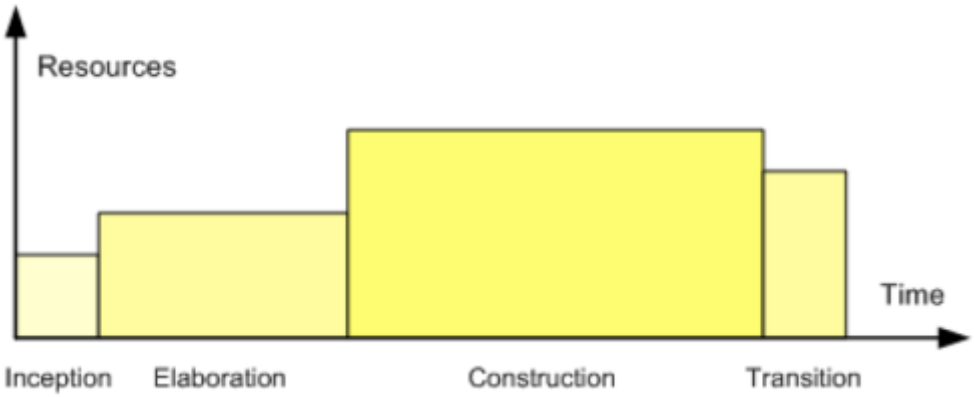


Fig. 8: Four Phases of the Unified Process

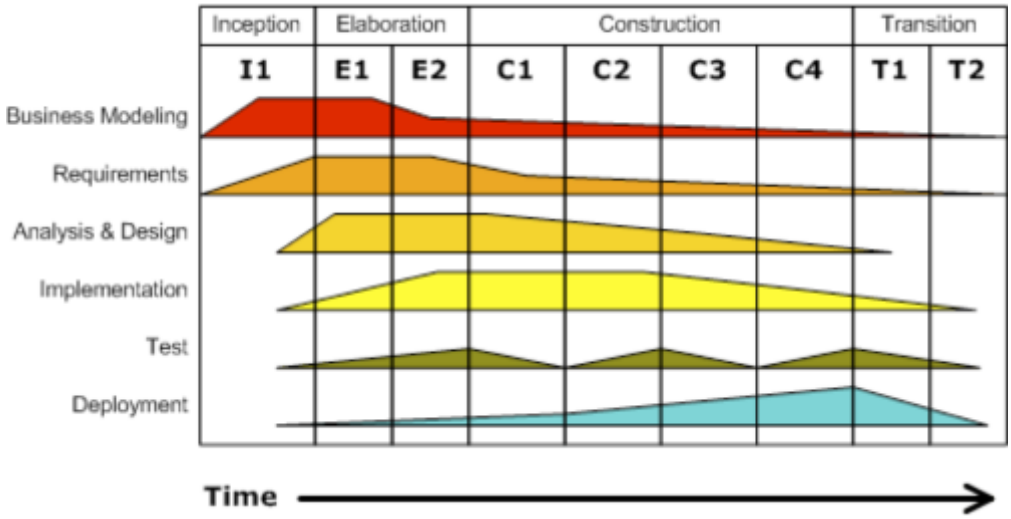


Fig. 9: Iterative Development of UP

我們把各個程序的內容規範(process disciplines)列舉如下：

- Business Modeling (企業塑模/塑模)
  - 瞭解企業的環境
  - 建立系統的願景
  - 建立企業的模型
- Requirements (需求)
  - 蒐集詳細的資訊
  - 定義功能性需求
  - 定義非功能性需求
  - 訂定需求的優先等級
  - 發展使用者介面
  - 與使用者評估需求
- Analysis and Design (分析與設計)
  - 分析與設計支援服務的架構與部署環境
  - 分析與設計軟體的架構
  - 分析與設計使用案例的實現
  - 分析與設計資料庫
  - 分析與設計系統與使用者介面
  - 分析與設計系統的安全與控管機制
- Implementation (實作)
- Test (測試)
  - 定義與執行單元測試

    - 定義與執行整合測試
    - 定義與執行可用性測試
    - 定義與執行使用者驗收測試
- Deployment (部署)
    - 取得硬體與系統軟體
    - 封裝與安裝元件
    - 教育訓練
    - 資料的轉換與初始化

UP使用UML(unified modeling language)做為模型，其中包含有如figure 10所列示的模型[5]。

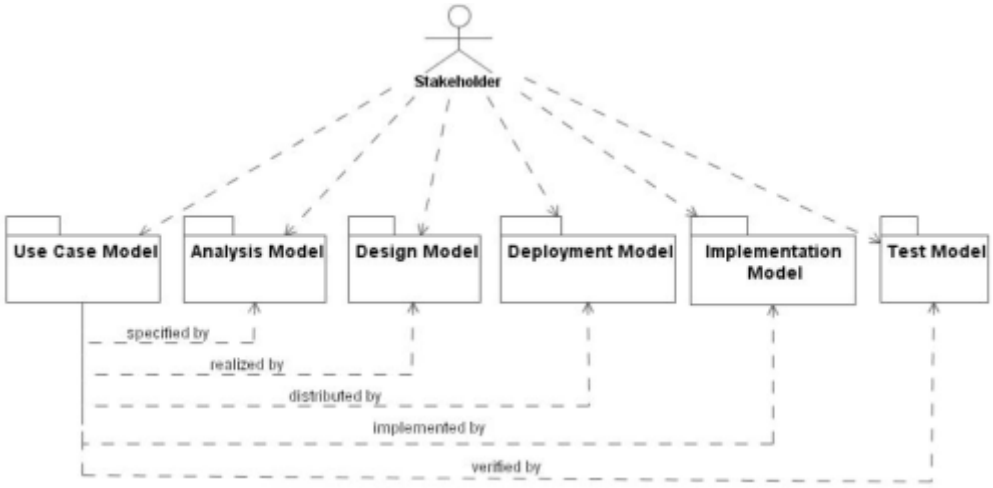


Fig. 10: UP的Model集合

在後續的課程中，我們將逐步地介紹在每個階段裡所需要使用的UML模型。

1)
Sebastiá Tyrrell, “The Many Dimensions of the Software Process”, ACM Crossroads, 2001.
2)
Winston W. Royce,“Managing the Development of Large Software Systems” in: Technical Papers of Western Electronic Show and Convention (WesCon) August 25–28, 1970, Los Angeles, USA, 1970.
3)
B. Boehm , “A Spiral Model of Software Development and Enhancement”, ACM SIGSOFT Software Engineering Notes, 11(4):14-24, August 1986.
4) , 5)
Ivar Jacobson, Grady Booch, and James Rumbaugh, “The Unified Software Development Process”, Addison-Wesley Professional, 1 edition, ISBN: 978-0201571691, February 14, 1999.