《系統分析》

試題評析	今年系統分析的考題主要反應軟體發展最新的趨勢，第一題是目前熱門的 UML 問題，考各種工具圖形的意義與作用；第二題探討演進式雛型與丟棄式雛型之差異和優缺點；第三題屬於結構化設計的問題，雖不是新趨勢的考題，但必須對結構化設計的原則很清楚才能回答完整；第四題的模型驅動架構(MDA)算是最新出來的問題，考的是對物件導向軟體發展的重要架構。綜合而言，今年的考題除第四題較偏僻一點之外，前三題都算是一般性問題，但也要回答完整才能拿高分，普通的考生約可得 40~50 分左右，程度好者可望拿 60~70 分。

一、明統一塑模語言（Unified Modeling Language）中，下列名詞之意義與作用：使用案例圖（Use Case Diagram）、互動圖（Interaction Diagram）、活動圖（Activity Diagram）、元件圖（Component Diagram）、部署圖（Deployment Diagram）。（三十分）

【擬答】

(一)使用案例圖(Use Case Diagram)：主要是由行爲者(actor)和使用個案兩種元件所組成，行爲者是環境中與系統有互動關係的人或事物，而使用個案是系統中可完成某一特定工作的一系列交易有組織的組合。使用個案圖可標示行爲者與使用個案間之互動以及使用個案與使用個案間之關係或合作。從外部觀點來看，其可描述使用個案做什麼(What)，從內部觀點來看，則可描述使用個案如何運作(How)。

(二)互動圖(Interaction Diagram)：循序圖與合作圖統稱互動圖，主要用於描述許多物件在單一使用個案中之互動行爲。

1.循序圖(Sequence Diagram)用來描述一個使用個案中之參與物件及物件間的互動行爲，強調以時間發生之先後順序來表達物件間的訊息傳遞與處理程序。循序圖之重要元件包括類別之物件、訊息、操作、生命線與控制焦點等。

2.合作圖(Collaboration Diagram)：亦用來描述一個使用個案中之參與物件及物件間的互動行爲，但合作圖強調以物件的結構化組織來表達物件間的訊息傳送/接收與處理程序。合作圖之重要元件包括類別之物件、連結、訊息與操作等。

(三)活動圖(Activity Diagram)：是一種塑模工具，它可被用於表達一個物件、一個使用個案、許多使用個案間或一個系統在其生命週期中之循序或同步的操作、作業流程或行爲，例如在其生命週期中之所有活動及其轉換關係。尤其是當互動圖中無法清楚表達物件生命週期之行爲時（例如複雜的平行處理或多執行緒處理），可以用活動圖更詳細的描述之。活動圖可以表達實體、物件或系統之活動、資訊流與控制等，因此可以把活動圖視爲是流程圖之擴充，因爲活動圖除了能表達流程圖的資訊外，還可以表達同步之行爲。

(四)元件圖(Component Diagram)：爲 UML 中第一種實體圖，用以說明系統設計過程中各類別與物件之配置，以及敘述軟體元件間的組織架構和相依關係。元件是開發與執行過程之實際物件的類別，將可分解的實際基本單位模組化，這些基本單位稱爲模組。元件圖主要以視覺化方式表達系統中一群實體元件與實體元件間的靜態結構關係，及說明其建構之細節。

(五)部署圖(Deployment Diagram)：爲 UML 中第二種實體圖，主要元件包括節點和連接線。部署圖亦以視覺化方式表達系統中一群實體節點與實體節點間的靜態結構關係，及說明其建構之細節。例如，可用以表達一個系統中軟硬體元件間的實體關係，尤其是表達元件或物件在一個分散式系統中如何制定路徑與移動。

二、何謂系統雛型（System prototype）？說明演化式雛型（Evolutionary Prototyping）與拋棄式雛型（Throw-away Prototyping）之差異及其優缺點。（二十五分）

【擬答】

(一)系統雛型（System prototype）：爲了讓使用者能界定他們對資訊系統的需求，系統開發人員先行盡快地完成一個系統模型，這個系統的功能並不完整，通常只具備一些主要功能，就開始交給使用者使用。使用後，使用者會不斷提出改進意見，據此不斷對系統模型進行反覆修改，直到符合與使用者之約定。此種系統模型稱爲系統雛型。

(二)演化式雛型法(Evolutionary Prototyping)是將所有需求看成一個整體，從需求最清楚的部分先快速經歷一個系統開發週期(如系分、設計、實施)，以完成初版雛型系統之開發，再利用該雛型與使用者溝通，以確定、修改和擴充需求，並藉以作爲下一階段雛型演進之依據。此種週期不斷進行，直到演進爲最終系統爲止。

(三)拋棄式雛型法(Throw-away Prototyping)是以一種快而粗糙的方式建立雛型，以促使使用者能夠盡快藉由與雛型之互動來決定需求項目，或資訓人員藉以研發問題之解決方法與資訊科技之應用等。這種雛型因爲用過即丟，所以不需要考慮雛型系統之運作效率與可維護性，也不需要容錯的能力。

(四)演進式雛型法的優點是可從初始雛型演進成最終系統，可縮短系統開發時間；缺點是因雛型開發過程中重視時效，雛型的結構與品質(可用性、可靠性、可維護性、績效等)往往較差。丟棄式雛型法的優點爲適用於具高困難度技術或設計的專案，可以藉由快速的雛型開發與檢討，探索出問題的解決方法或資訊科技應用之可行性；缺點則是因爲用過即丟，將導

致成本之浪費。

三、系統設計品質（System Design Quality）之衡量，除了正確性（Correctness）外，可維護性（Maintainability）是主要的考量，說明下列四種軟體設計特性之意義及其如何影響可維護性：內聚力（Cohesion）、連結力（Coupling）、可理解性（Understandability）、可調整性（Adaptability）。（三十分）

【擬答】

(一)內聚力(Cohesion)是對模組內各成員之間，其功能結合強度之度量；此處的成員是指模組內完成某項特定功能的一條或一群指令。內聚力由強至弱可分爲：功能內聚、順序內聚、溝通內聚、程序內聚、暫時內聚、邏輯內聚及巧合內聚等七種。模組的內聚力越強，系統的可維護性越好；內聚力越弱則可維護性越差。

(二)連結力(Coupling)是模組與模組之間相關性強度之度量，耦合力由弱到強可分爲：無直接耦合、資料耦合、戳記耦合、控制耦合、外在耦合、共同耦合及內容耦合等七種。模組間的耦合力越弱，表示其相關性越低，系統的可維護性越好；耦合力越強則可維護性越差。

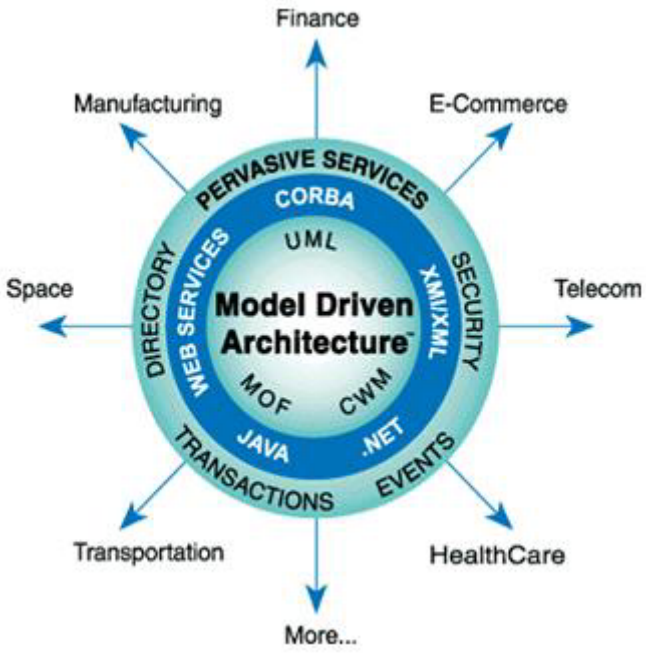
(三)可理解性(Understandability)是指軟體架構中各模組之功能及彼此間的相關性容易理解的程度。會影響可理解性的因素包括：模組是否進行適當分解、工作與管理是否分開、決策是否分裂、是否避免使用靜態記憶體、程式結構與資料結構間是否出現結構衝突、模組功能是否過於廣泛、模組之扇出是否太大等。因此可理解性越高，則系統的可維護性越好；反之則越差。

(四)可調整性(Adaptability) 是指軟體架構是否能隨外在環境或使用者需求改變而隨之調整的容易程度。會影響可調整性的因素包括：工作與管理是否分開、系統形狀是否平衡、是否使用資料叢集、起始動作是否盡量延後、終止工作是否儘早執行、是否設計高扇入模組等。因此，可調整性越高，系統的可維護性亦越好；反之則越差。

四、何謂模型驅動架構（Model Driven Architecture）？說明它對軟體開發之影響。（十五分）

【擬答】

(一)模型驅動架構(Model Driven Architecture, MDA)是 2001 年被世界物件管理組織(OMG)所採用作爲軟體發展的一種標準框架(framework)，MDA 提供一種開放式、廠商中立的方法來因應軟體互動性(Interoperability)之挑戰，它可以支援 OMG 所建立的各種模型標準：UML(Unified Modeling Language)、MOF(Meta-Object Facility)、XML(Extensible Meta Language)、XMI(XML Meta-Data Interchange)、CWM(Common Warehouse Meta-Model)及 CORBA(Common Object Request Broker Architecture)等，如下圖：



(二)MDA 對軟體開發的影響：

1.它提供一種架構，讓使用者能夠將過去已完成的軟體、目前正開發的軟體和未來要發展的軟體整合起來。

2.能在固定的基礎建設之上保留軟體發展的彈性。

3.它能延長軟體使用的生命週期、降低維護的成本及提高投資報酬率。