Key points

Software design and implementation are inter-leaved activities. The level of detail in the design depends on the type of system and whether you are using a plan-driven or agile approach.

軟體設計和實現是交叉活動。 設計中的詳細程度取決於系統的類型以及你是使用計劃驅動還是敏捷開發。

The process of object-oriented design includes activities

to design the system architecture,

identify objects in the system,

describe the design using different object models

and document the component interfaces.

物件導向設計的過程包括設計系統體系結構，識別系統中的對象，使用不同對像模型描述設計和記錄組件接口的活動。

A range of different models may be produced during an object-oriented design process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models).

在物件導向的設計過程中可以產生一系列不同的模型。 這些包括靜態模型（類別模型，泛型模型，關聯模型）和動態模型（序列模型，狀態機模型）。

Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces.

元件的接口必須定義的恰恰好，以便其他對象可以使用它們。 UML接口構造型可用於定義接口。

When developing software, you should always consider the possibility of reusing existing software, either as components, services or complete systems.

在開發軟體時，始終考慮重用現有軟體的遠景，無論是作為元件，服務還是完整系統。

Configuration management is the process of managing changes to an evolving software system. It is essential when a team of people are cooperating to develop software.

配置管理是管理對不斷發展的軟體系統的更改的過程。 當一個團隊合作開發軟體時，這一點至關重要。

Most software development is host-target development. You use an IDE on a host machine to develop the software, which is transferred to a target machine for execution.

大多數軟體開發都是本地目標來開發。 您可以在主機上使用IDE來開發軟體，該軟體將傳輸到目標主機以供執行。

host-target development：開發環境與實際的不同(ex Android app)

Open source development involves making the source code of a system publicly available. This means that many people can propose changes and improvements to the software.

開源開發涉及使系統的開源代碼可公開使用。 這意味著許多人可以提出對軟體的更改和改進。

HW3

3.Describe the 5 key activities in an object-oriented design process.

(1) Specify object interfaces. 指定對象。

(2) Develop design models. 開發設計模型。

(3) Identify the principal system objects. 識別主要的系統。

(4) Design the system architecture. 設計系統架構。

(5) Define the context and modes of use of the system. 定義系統的上下文以及使用模式。

#### **S ingle-Responsibility Principle（單一責任原則）**

一個 Class 只應該擁有一個改變的理由。

當一個 Class 擁有多個責任：

* 受到改變的機會較大，容易出BUG（容易因為多種原因壞掉）
* 程式碼又臭又長（體積太大）
* 程式碼難以維護（不好打磨）
* 程式碼可讀性比較低（被路人誤會）
* 名字會比較難取（功能太多，出現一堆爛大街的名字）
* 還有一堆缺點…

會違反單一責任原則，代表你對於這個 Class 的概念不清，無法用一個具體的形象來描述你的 Class，對之後程式碼的維護或是交接都是個大危機。

#### **O pen-Closed Principle（開放封閉原則）**

軟體應該對擴展開放，但是對修改封閉。

一份好的 Code 應該要保留彈性，讓我們之後可以方便地擴充新功能，以及修改既有的功能。

如果新增功能時，影響到了其他的 Code，甚至造成程式出現BUG（垮掉），代表程式裡面 Class 之間的耦合度（Coupling）過高，你依賴我、我依賴你、你依賴它、它又依賴我，牽一髮而動全身，將造成程式難以擴充與更新。

#### **L iskov Substitution Principle（里氏替換原則）**

程式中的對象應該是可以在不改變程式正確性的前提下被它的子類別所替換的。

#### **I nterface Segregation Principle（介面隔離原則）**

客戶不應該被逼著依賴它們沒有使用的方法， 應該根據客戶的需要，給他最正確的東西。

#### **D ependency-Inversion Principle（依賴反轉原則）**

高階模組不應依賴低階模組，兩個都應該依賴在抽象概念上。

抽象概念不依賴細節，而是細節依賴在抽象概念。

說話不要說的太死，盡量講一些概念性的東西。