1. 改變底層Hotel、Room資料型態

原本的概念是Hotel跟Room是獨立、沒有連結的個體，各自放在不同的List裡面。

後來決定刪除RoomList，建立Hotel和Room的從屬關係(has a)，也就是將Room放入Hotel的attribute中。

1. 改變底層Book資料型態

原始的Book不太符合常理訂單的邏輯；因此將他更改成一個完整訂單會有的樣子。（防呆：）

1. 優化使用者操作的直覺度

(1)原先是系統幫旅客計算出住宿人數、房間數之間我們所認為的最佳房型組合，例如最便宜或以最少房間數為原則；現在新增一個下拉式選單，提供旅客從我們找到的所有房型組合中，選擇最適合自己的方案。

1. 將Hotel資料存成資料庫

原先Hotel的資料僅存於一個不可更動的ArrayList；現在改成存放於資料庫中，未來可以對飯店列表做增減、改動資料等等，使系統能有了更多操作上的彈性。

1. 優化系統速度（精簡化）

(1)原本為了貪圖方便，使用了許多的ArrayList，導致系統反應速度較慢；如今更改成盡可能地使用基本資料型態及其陣列，取代過去較為繁冗的自定義class及ArrayList。

(2)訂單的價錢不再到處算，統一在最後須要列出價錢的地方(Book)再計算。

1. 更改使用的資料庫

原本使用僅能在單機上操作的sqLite資料庫；現在轉為使用MySQL。

1. 團隊溝通合作

(1)統一coding格式：盡量統一函數變數之命名原則、以及參數之排序順序，方便團隊內組員之間的分工、理解。

1. 劃分Operation

之前是站在設計者思考的角度，將所有我認為是在進行操作的函數，全部放在同一個Operation Class；現在使用Use Case的概念，經過設計UML的階段之後，先打散原本的Class、method，重新規劃能達到相同目的且經過整理的package、class。

1. 整合UserCase

原先在Use cass設計過程中，是將Search和Book，分作不同的Use case、有不同的Control cass；但是我們在畫communication diagram的過程中，發現他們有重複性極高的行為，在use case diagram中事實上Book可以是Search的extend。因此最後決定合併為SearchAndBook這一個Class。

1. 10