易用性與可維護性於智慧環境之研究
Analysis and Design on Usability and Maintainability Issues in a Smart Home System

廖介綱 劉立頌 吳宗錫 游榮聿 盧柏亘
國立中正大學 電機工程學系
jack710050@gmail.com
aliu@ee.ccu.edu.tw

摘 要

本論文實作「行程提醒暨家電控制系統」之功能在於整合並顯示智慧家庭內已連線的家電資訊，本文將由此角度切入探討品質屬性中的易用性其本質與意義、介紹爲何易用性對於使用者而言如此重要，以及何種方式的呈現能夠有效提升。另一方面，軟體系統除了該提供良好的使用者體驗，更要能夠隨著使用者需求的變化而進行有效率的修改，本論文依照這些情境探討可維護性問題。爲了更有效的提升易用性，本系統的操作界面捨棄了複雜的家電開關設計，改爲透過擬物化風格表示家電狀態。爲了兼顧開發方便與可維護性，採用 MVC 架構將動畫顯示與資訊擷取分開實作。透過核對設計清單的方法，在確認滿足多數清單之建議事項後，系統之易用性與可維護性具有明確的改善。

關鍵字：易用性、可維護性、智慧家庭

一、前言

當智慧家庭與物聯網(IoT)迅速發展的同時，使用者所接收到的資料訊息更多且複雜，故本論文實作「行程提醒暨家電控制系統」之功能在於整合顯示智慧家庭內已連線的家電資訊，使得使用者能夠根據這些家電狀態直接進行遠端操作。然而整合顯示家電資訊之外，也必須考量到使用者體驗。在設計圖形化界面之前，未經過提示的普通使用者可能無法直覺理解文字界面上所呈現的顯示內容與家電設備之間的關係。根據此種問題，本文將由此角度切入探討品質屬性中的易用性(usability)其本質與意義、介紹爲何易用性對於使用者而言如此重要，以及何種方式的呈現能夠有效提升。爲了進一步的實現對於使用者體驗的提升，本系統決定採用圖形化界面顯示搭配上直覺的操作邏輯，希望能夠最大程度的降低使用者學習成本。

另一方面，軟體系統除了該提供良好的使用者體驗，更要能夠隨著使用者需求的變化而進行有效率的修改。本論文也會依照這些情境探討可維護性(maintainability)問題。以本系統之功能而言，需要實作多個家電設備，爲確保後續系統維護方便進行，本系統的規劃朝向物件獨立實作方法。而根據物件導向技術的特性：以物件爲出發點，並藉由物件與物件之間的互動完成問題的解答；由於物件之間的獨立性，單一物件的修改並不會影響到其它物件內容，由此加以改善可維護性問題。本系統在設計與實作階段即是應用此概念，例如：多個家電物件繼承自單一個家電類別。如此一來，不但方便開發還能檢視是否能夠滿足本系統對於可維護性的需求。

二、易用性與可維護性問題探討

所謂的易用性大致上被分爲五個能夠個別探討的議題[1]，其中分別爲 learnability, efficiency, memorability, errors 與 satisfaction。除此之外，對於真正好用的系統，效用性(utility)與易用性，兩者皆爲重要因素，效用性代表系統是否能夠達成使用者需求，而易用性則是評估這些需求有多輕易被使用。

對於商業化的軟體系統而言，易用性的程度將會是決定此系統是否能夠創造利潤的關鍵，當使用者浪費越多時間在冗贅的操作，能夠達成的效用就越低。理論上對於易用性的提升方法最直觀就是使用者測試，大致上分爲三個方面：與系統的直接使用者持續溝通並收集反饋、要求使用者執行經過特定設計的任務並觀察使用者是否能夠成功操作，以及哪些步驟遭遇困難。

軟體系統的維護一直以來都被認爲是軟體生命週期當中花費成本與資源最多的一個階段，對於物件導向軟體系統而言，個別軟體元件在設計時還需考慮整合的問題，因此系統對於可維護性的需求有顯著提升。而可維護性即是軟體系統被修改的難易度，當今有許多軟體工程的方法與技巧，其最主要目標皆是可維護性，例如：重複使用(reuse)、產品線方法(product-line approach)等[2]。

根據 SWEBOK 定義[3]，將可維護性細分爲五個特性：modularity, reusability, analyzability, modifiability 與 testability，其中 analyzability 與 testability 可以被廣泛的涵蓋爲軟體錯誤能夠被重現的難易度。而 modularity 則是評估改變系統中單一元件時，對於其他元件的影響大小，並且可以由此引導出模組耦合性，即軟體元件之間的獨立與相依性。無論是開發人員或利益關係人，爲了更有效率的節省系統後續維護與修改所花費的時間與金錢，對於系統可維護性之重要性皆有共識，尤其是在時間與經濟壓力下，對於大型軟體系統的維護或修改，更是突顯了系統可維護性的影響。

可維護性對於軟體系統的影響可分爲三個層

次[2]，系統、架構與元件，本論文主要針對系統層面進行探討，然而決定可維護性的實現方式必須釐清下列問題：

1. 軟體未來的可用程度與用途
2. 瞭解軟體的執行平臺
3. 在開發過程中，軟體根據何種標準進行修改
4. 預期系統未來的擴充功能
5. 根據產品生命週期評估維護成本

透過上述問題的討論，對於決定何種實現方式能夠提供最大效益有著實質幫助。

當開發人員針對品質屬性設計系統時，無可避免的必須先理解品質屬性所影響的層面。就如同 L. Bass 針對各屬性所提出的通用劇本(general scenario)[4]，明確闡述了觸發來源(source of stimulus)、觸發狀況(stimulus)、系統環境(environment)、系統(artifact)、回應(response)與回應量測(response measure)在事件發生時所代表的意義。在此，本論文以易用性與可維護性爲例：

1. 易用性

首先，易用性的改善主要是針對使用者體驗，故此部分劇本(表 1)皆是以使用者爲觸發來源。當使用者想開啟或關閉電燈裝置時，本系統能夠透過圖形化界面的呈現(如圖 1)，將此行爲簡化至兩個操作步驟以內。若是使用者想知道當下的日期或時間，根據本系統之圖形化界面，使用者能夠在畫面之固定且顯眼位置得知。即使是使用者不常操作本系統的情況下，也能夠因爲本系統所呈現的裝置圖標，而有效率的記住各圖標之意涵。透過上述品質屬性劇本可知，本系統至少能夠改善易用性中的可學習性、滿足性、效率與可記憶性。

表 1 易用性之品質屬性劇本

Scenario	情境一	情境二	情境三
Source	使用者	使用者	使用者
Stimulus	適應系統	使用者覺得自在	能記住圖標之意涵
Artifact	系統	系統	系統
Environment	設定時間	執行時間	執行時間
Response	想開關電燈	想知道日期與時間	想記住圖標之意涵
Response measure	開關電燈功能可在 2 個步驟內完成	在畫面固定位置顯示時間日期	圖標以擬物化呈現

2. 可維護性

可維護性的重點在於，能夠改善開發者或系統管理員針對特定功能調整之效率，請參考表 2。開發人員若在實作過程中想新增可連接的家電裝置，物件導向的開發概念對於解決此種需求能夠提供有效幫助，在既有的家電類別之下，本系統可讓開發者在 30 分鐘內完成實作。當開發人員依據客戶需求欲將溫度單位由攝氏改爲華氏時，也能夠不因爲修改溫度計算方式而影響其它家電裝置。而良好的軟體架構也能使系統管理員在操作系統時，快速修改常用的調整選項。是故，達成這些品質屬性劇本後，本系統能夠改善可維護性中的模組化、易修改性與可分析性。

表 2 可維護性之品質屬性劇本

Scenario	情境一	情境二	情境三
Source	開發人員	開發人員	系統管理員
Stimulus	新增家電裝置	更改溫度單位	修改行事曆提醒時間
Artifact	程式碼	程式碼	程式碼
Environment	實作時間	實作時間	實作時間
Response	不會造成既有功能失效	不影響其它裝置	能有效調整
Response measure	可在半小時內完成	可在10分鐘內完成	可在 10 分鐘內完成

圖 1 本系統的使用者界面

三、分析與設計

在本系統之開發初期，為了有效率的完成系統開發便決定採用敏捷開發模式，先迅速的釐清具體的開發目標[5]，有了明確需求，才能夠列舉完整的設計。首先進行分析問題，找出需要的功能。此系統開發主軸為「行程提醒暨家電控制系統」，而系統所需的功能，是考慮使用者在智慧家

庭內可能發生的情境，根據以上論述，可以歸類出系統需要提供五個關鍵的功能，如表 3 所示。

表 3 功能需求 (Functional Requirements)

FR001	本系統能夠分析使用者語意。
FR002	本系統能根據使用者語意自動登錄行事曆。
FR003	本系統能根據行事曆主動提醒使用者。
FR004	本系統可以即時查詢天氣狀態。
FR005	本系統能夠自動調整電器用品之狀態。

有了明確的功能需求後，開始分析何種使用者介面設計對於易用性具有較大的正面效益[4]。本系統採用了多種方法，例如：將相關資訊集中顯示於臨近區域、降低物件之間的依賴性與保持界面修改之彈性。

爲了更有效的提升易用性，本系統的操作界面捨棄了複雜的家電開關設計，改爲透過擬物化風格表示家電狀態，並且允許使用者直接點擊圖標進行操作，具體的實現了「所見即所得」設計原則，詳見表 4。而天氣資訊顯示的部分，也會根據不同的天氣狀態自動切換天氣動畫。

表 4 介面需求 (Interface Requirements)

IR001	使用者可以自然語言輸入待辦事項。
IR002	系統可提供圖形化介面做為視覺反饋。
IR003	系統以圖形化介面提醒使用者。

本系統圖形化界面實作方法是基於微軟 XNA 遊戲引擎[6]，搭配自行設計的家電圖標。在軟體架構的設計過程中，爲了兼顧開發方便與可維護性，決定利用 C#程式語言中的物件導向概念，並採用 MVC 架構將動畫顯示與資訊擷取分開實作，並且由 dashboard 元件整合並統一顯示所對應的資訊，如此一來不但能夠根據功能的不同而拆分成單元化模組、縮小各個功能模組還可以提升模組之間的凝聚力，達到 “Loosely coupled, Highly cohesive.” 的效果。除了單元化開發，本系統爲了有效提升系統管理員在後續操作時對於新增裝置的效率，在設計階段也定義了模組之間能夠正確溝通的統一界面。

表 5 設計與實作限制
(Design and Implementation Constrains)

DIC001	以自然語言做為輸入內容。
DIC002	介面端採用 C#語言編寫。
DIC003	系統後端採用 Java 語言編寫。
DIC004	系統提醒之呈現以圖形介面為主。

在非功能性需求方面，由於本系統的主要目標是提供使用者一個具有易用性的操作界面，因此非功能性需求對於操作效率或者可用性的要求上是具有一定水平的。如表 6 所示，在語音登錄行事曆代辦事項的部分，本系統必須要先具備足夠的語音辨識準確率，才能夠完成接續任務。此外，為了不造成使用者等待時間太久，運作效率必須要有所要求。本系統亦具備根據使用者行程而自動關閉家電設備的功能，爲了避免電力資源的浪費，進而要求能夠在使用者離開之一分鐘內完成。

表 6 非功能性需求 (Non-functional Requirement)

PR001	系統具有的功能 之語意判斷 準確率達到 90% 。
PR002	系統能在 10 秒內完成行事曆登錄 。
PR003	家電能在使用者離開 1 分鐘內關閉 。
PR004	燈光顏色變換能在 10 秒鐘內達成。

在此根據功能性需求的描述與分析整理成使用者案例圖(use case diagram)，如圖 2 所示。透過此使用案例圖，可以有效的幫助開發者與客戶更具體釐清本系統的運作方式，促進雙方溝通。

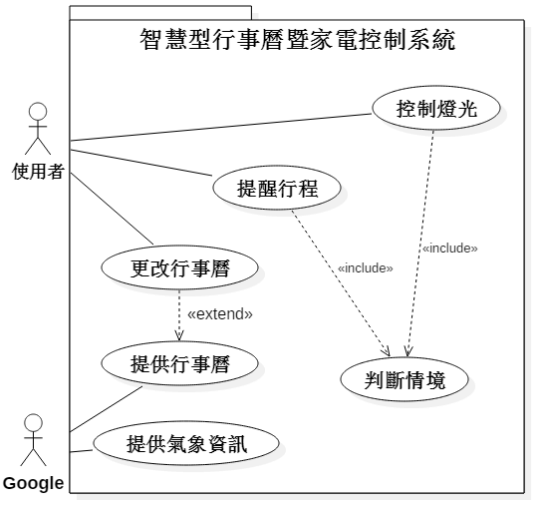

圖 2 行程提醒暨家電控制系統之使用案例圖

接下來嘗試著找出參與者與使用案例、描述使用案例與描述使用案例與其它案例之間的關係，並透過如表 7 與表 8 件範例的方式紀錄。依照使用者案例規格樣板的描述記載，不但能夠更簡潔的描述系統實際執行步驟，也可以在未來的驗收

測試時，根據這些規格產生測試案例。

根據表7所示，此使用案例是規範使用者操作界面的運作方式。在前置動作完成後(即系統開機)，一旦使用者啓動程式，系統便能夠顯示使用者介面，並且等待使用者執行其它功能。

表 7 使用案例規格樣板之範例一

Use case name	操作界面
Summary	提供使用者操作界面
Actor	使用者
Precondition	系統開機完成
Description	Actor Actions： 開啟程式 System Responses： 顯示使用者介面
Post-condition	確認使用者輸入之資料

而表8這個案例，是定義氣象服務的運作方式。其中，使用者能夠輸入地區名稱，系統將會根據所輸入的名稱進行分析，並且收集該地區之天氣資訊，從而整合並顯示到圖形界面上。

表 8 使用案例規格樣板之範例二

Use case name	提供氣象服務
Summary	收集指定地區之氣象資料
Actor	Google 服務
recondition	系統完成分析
Description	Actor Actions： 輸入指定地區 System Responses： 查詢氣象資料
Post-condition	等候使用者指令

當使用案例圖完成後，開發者與客戶對於系統需要達成哪些需求就有明確共識。為了能夠落實目前的規劃，還需進一步完成設計系統細節的工作。因此，開發者需進行流程的分析，輔助描述捉使用案例的執行細節、建構出系統中邏輯的處理程序與檢驗案例互動描述的正確性。

圖3的活動圖中，在使用者執行系統的情況下，如果有新的待辦事項被新增到行事曆當中，系統會自動識別此一待辦事項之預訂時間是否即將到期。若即將到期，系統則進一步收集行事曆待辦事項內容與事件發生地之天氣預報，整合之後一併顯示於圖形界面上，達到提醒使用者的效果。在使用者離開智慧家庭之後，爲了避免使用者忘記關閉電器所造成的電力資源浪費，本系統將會主動關閉可控制的家電設備。

從整體系統運作的狀態流程來看，詳見圖 4，系統在通過待機狀態後，依序處理行事曆資訊與家電資訊兩種狀態。在行事曆資訊的部分，如同前面所述分爲接收事件時間、接收事件內容與收集天氣資訊三個狀態。之後系統會同步目前家電的運作狀態。將兩方面的資料整合之後，顯示於使用者介面，並根據使用者狀態關閉電器設備。

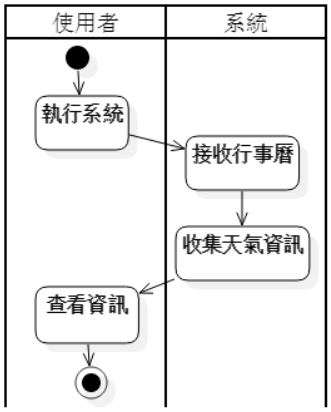

圖 3 服務分析處理活動圖

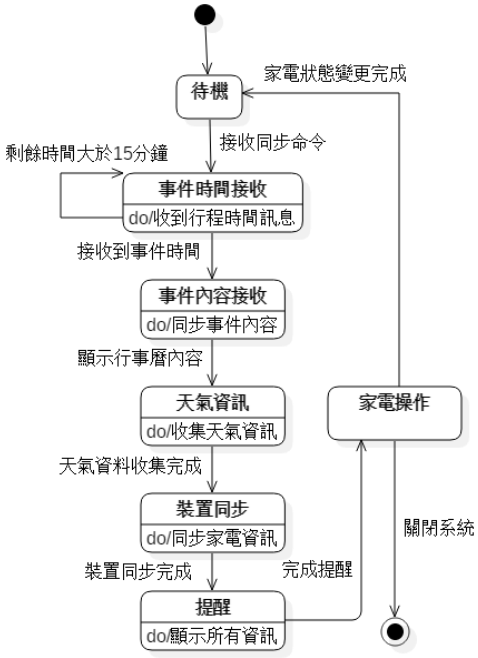

圖 4 系統運作狀態圖

圖5為本系統的類別圖，主要分為三大類別：操作界面、資訊儀表與家電狀態顯示。資訊儀表類別負責顯示系統日期、時間、地區天氣與待辦事項，而這四種功能是透過各自獨立的物件實作而成。家電狀態類別負責實作藍牙傳輸界面並且透過藍牙與各家電裝置進行溝通，在彙整之後顯示於圖形界面上。而操作界面類別則是涵蓋了資訊儀表與家電狀態兩個類別，從系統的角度而言，以這兩個類別爲基礎再加上簡單與直覺的操作才能算是完整的使用者界面。

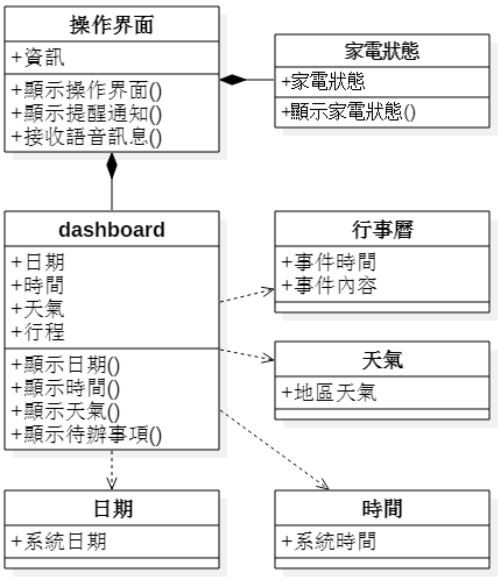

圖 5 本系統的類別圖

透過這樣的類別設計，將資訊儀表與家電狀態整合至操作界面之中，不但可以完整顯示使用者所需的資訊，提升易用性；還可以根據兩個類別用途的不同將其模組化，保持可維護性。

如圖 6 所示，本系統分為四個部分：操作端、資訊端、伺服器端、控制端。操作端包含了圖形化界面與語音輸入功能，圖形化界面是透過簡單易懂的圖標方便使用者理解與操作各家電。語音輸入功能則是提供使用者以自然語言的方式遠端操作電器設備。

資訊端主要是接收並且初步處理前景所需的相關資訊，如詞性分析、行事曆事件與天氣資訊。詞性分析屬於語音輸入功能中的重要步驟，能夠在使用者輸入的自然語言被轉換成文字後，將字句根據不同詞性進行分段。而此處的行事曆是聯絡與同步 Google 行事曆的元件。

伺服器端是本系統的核心功能區塊，是各個重點功能的計算中心。例如：此區塊中的資料庫保存了本系統所有資料，任何對於家電狀態或其它即時資訊的應用皆需要對資料庫進行存取。而此區塊中的情境判斷元件將針對使用者的個人化資訊做出不同的反應。舉例而言，當系統得知使用者於行事曆登錄外出時，系統會進行提醒之外，更會進一步的替使用者關閉家中電器，此一功能的實作則必須與控制端的控制家電元件進行充分的溝通。

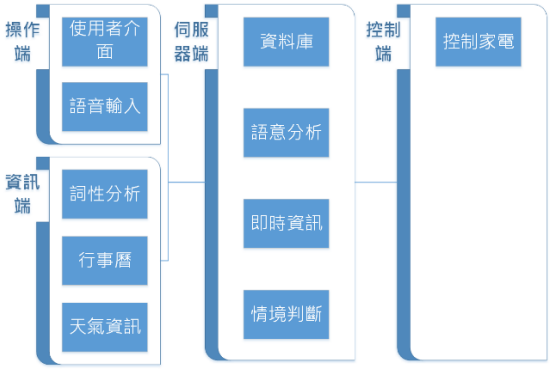

圖 6 本系統的佈署圖

四、系統驗收

4.1 測試案例分析

本系統在實作過程中也參考了測試導向的開發模式(test-driven development)，而測試導向開發模式之特點在於，能夠於系統實作之前先設計出測試方法與案例。在此根據表 3 的描述將功能性需求轉化爲測試案例設計文件，如表 9~13 敘述。因此需撰寫出可完成檢驗這些單元測試的程式碼。如此一來，當所有的測試案例都逐一通過驗收之後，可以確認所有的功能性需求都將被實現。

本系統之家電遠端操控功能是藉由語音輸入來達成。由於整個語義辨識過程複雜，為了提升系統對於語義的成功辨識率，目前提供給使用者多個特定關鍵字，方便使用者成功進行操作。語音系統之重要性在於，此步驟爲遙控操作功能之基礎，系統必須順利擷取並且正確辨識，方能夠執行接續步驟。是故將此功能列爲功能性需求驗收之重點項目，如表 9。

表 9 測試案例一

Identification	FRT001
Name	語音系統
Tested target	辨識語意
Reference	FR001
Severity	1
Instructions	對手機講話
Expected result	正確辨識出語意
Cleanup	清除語音資料

除了語音輸入達成遠端操作家電之外，本系統亦提供系統自動與使用者手動操作之功能。系統自動操作的部分，是在使用情境中爲了避免使用者因故忘記關閉家電，造成電力資源的浪費，也希望透過此系統進一步提升系統易用性。而手動操作模式則是期望能透過提供給使用者簡單易懂的圖形介面與交互邏輯，降低使用者初次操作時的學習成本。兩者皆屬於本系統之基本功能，也非常容易影響系統之易用性，亦陳列於測試案例之中。

表 10 測試案例二

Identification	FRT002
Name	開關電器
Tested target	是否手動與自動皆可操作
Reference	FR005
Severity	1
Instructions	1. 系統控制開關電器 2. 手動控制開關電器
Expected result	正確開啟、關閉電器
Cleanup	正常

本系統整合 Google 行事曆功能，使用者於其它設備(如：個人電腦與手機)輸入待辦事項或事件時，本系統能夠同步接收至資料庫中。測試案例通過條件爲，使用者之原始輸入與資料庫內容必須完全符合。

表 11 測試案例三

Identification	FRT003
Name	登錄 Google 行事曆
Tested target	準確登錄行事曆
Reference	FR002
Severity	1
Instructions	檢視行事曆內容是否與預期指令相同
Expected result	相同
Cleanup	保持狀態數據，直到下次狀態改變為止

本系統整合 Google 行事曆功能，藉由使用者輸入預訂事件，本系統將確認該事件是否於短時間內到期，並且在事件到期 15 分鐘內主動顯示於圖形界面，提醒使用者。在此所提供之測試方法爲確認系統能夠在正確的時間之內顯示正確的事件提醒，方可通過測試案例。

表 12 測試案例四

Identification	FRT004
Name	提醒使用者
Tested target	準時提醒使用者
Reference	FR003
Severity	1
Instructions	1. 登錄指令 2. 等待系統提醒
Expected result	能夠準時提醒行事曆內容
Cleanup	清除

當使用者在登錄行事曆事件時，若有提供事件位置，本系統將根據該位置主動收尋天氣資料，並且同步呈現於圖形界面上，方便使用者根據此天氣資訊準備雨具。是故此功能重點在於天氣資訊的正確性，而驗收項目則是確認顯示資訊與真實天氣情況相符合。

表 13 測試案例五

Identification	FRT005
Name	查詢天氣
Tested target	系統是否及時查詢天氣資訊
Reference	FR004
Severity	1
Instructions	詢問當下當地天氣資訊
Expected result	能正確提供天氣資訊
Cleanup	保持天氣資訊

4.2 品質屬性驗證

爲了驗收本論文所探討的品質屬性是否在經過系統實作之後能夠有所改善，本論文將根據 L. Bass 等人所提出的設計清單(design checklist) [4]，採取核對的方式確認是否完成。設計清單的目的主要是針對各種品質屬性提醒開發者多個必須注意之實作細節。本論文也將在此環節檢視易用性與可維護性之設計清單，達到驗收的效果。

關於易用性之設計清單如表 14 所示，根據責任分配之內容，此清單明確定義出所謂的易用性應該在產品端協助使用者學習使用、提升操作效率、適應且調整系統以及協助使用者在錯誤中復原。此設計清單對於系統之元件屬性(如即時性與正確性)有所要求，並檢驗系統是否加入抽象化資料(data abstraction)的概念以便使用者完成上述操作。除此之外，還要決定架構中的元件對於使用者而言是否可見(visible)、避免因爲使用者限制系統資源之使用後，造成負面影響、方便使用者決定連結時間(binding time)以及確認所使用的開發技術能有效達成易用性情境。而本系統正是透過圖形化界面降低使用者之學習成本、提升操作效率，擬物化圖標與使用者操作時所顯示的動畫都一再地強調了本系統能根據互動提供即時與一致的視覺回饋。本系統之家電裝置連結部分，以電冰箱爲例，會在系統執行時不斷的接收溫溼度資訊，並且在背景以抽象化資料的方法處理。根據前述之系統架構，系統執行時以圖形界面具有較高的優先權，故硬體效能遭遇瓶頸時，會以圖形界面能流暢運行爲優先考量。

表 14 易用性之設計清單

類別	內容
責任分配	幫助使用者完成下列事項： 1. 學習如何使用 2. 有效率的完成操作 3. 適應且調整系統 4. 在錯誤中復原
協調模型	確認系統元件之屬性是否影響使用者完成上述動作。
資料模型	將使用者互動的資料抽象化(data abstraction)。確認這些資料抽象化之屬性可幫助使用者完成上述動作。

規劃架構	決定哪些架構元件對於使用者而言是可見的，並評估這些可見元件對於使用者操作是否有幫助。
資源管理	釐清使用者對於系統資源調度的方式，確保使用者在調整系統之資源使用限制後，不會影響使用者操作。
連結時間	在使用者的控制下決定連結時間，並且確定此種連結時間有助於提升易用性。
技術選擇	確認所使用的開發技術能夠協助達成針對易用性而規劃的情境。避免所挑選的技術對易用性造成負面影響。

表 15 呈現了可維護性中的可修改性(modifiability)設計清單，其核心概念便是釐清何種功能需要新增、修改與刪除，以及執行這些操作之後其他部分的功能有哪些受到影響。透過系統架構的改進，確保系統中可能受到影響的模組盡量減少。並且針對系統的變更分析其來源是終端用戶、系統管理員或開發者，若是來自用戶，則需確保用戶知道系統中的必要屬性不能被修改。而規劃架構的部分，會先行確認系統在開發時期、設計時期、編譯時期與執行時期是否需要被修改，並設計易於修改的架構。透過資源管理與技術選擇，控管每一次修改所花費的成本與資源，確保進行修改後，剩餘的資源還足夠系統達到需求。由於本系統重點功能在於家電裝置之整合，是故在設計初期便是以可維護性爲系統開發主軸，在實作物件導向技術後，每一次針對程式碼的修改，都能夠清楚掌握該修改內容對於模組或系統的影響範圍，達到有效控制的效果。且根據物件導向技術之概念，對於單一物件之修改，並不會影響其它物件之功能，故無論於任何開發時期，皆能明顯的降低修改成本。關於系統與裝置之間的溝通方式，本系統採用模組化設計搭配藍牙(Bluetooth)傳輸，舉例而言，若是想將藍牙改變爲無線網路(Wi-Fi)傳輸，針對傳輸界面模組進行修改即可。

表 15 可維護性之設計清單

類別	內容
責任分配	針對每一次的修改： 1. 釐清哪些功能任務需要被增加、修改或刪除。 2. 確認哪些功能任務受到影響。
協調模型	確定哪些功能與品質屬性在系統執行時可以改變，以及其對協調性的影響。
資料模型	確認哪些關於抽象化資料的改變可能會發生。針對這些變化分析來自於終端用戶、系統管理員或開發者。若來自用戶，確保用戶知道系統的必要屬性。
規劃架構	確認功能與計算元件的溝通方式在開發時期、設計時期、編譯時期與執行時期是否需要被修改。
資源管理	分析功能的新增、修改或刪除對於資源的使用。 確保進行修改後，資源還足夠系統達到需求。
連結時間	針對每一次的修改： 1. 確認最後一次的修改時間 2. 使用 defer-binding 機制 3. 確認修改所花的成本
技術選擇	分析系統的技術是否會造成修改難度提高。 1. 系統技術是否有助於增加、測試與實作修改？ 2. 系統修改之難易度？

4.3 討論

如易用性之設計清單所述，易用性之基礎原則為幫助使用者學習與提升使用效率，並確認系統之協調性是否會對這兩個原則造成影響，而本系統採用之圖形界面將使用者所需要的資訊統一顯示，達到了化繁爲簡的效果，相較於傳統的文字界面，本系統之界面更爲美觀與方便使用者學習操作。搭配上使用者操作時所呈現的過場動畫，能夠給使用者更明確的視覺回饋，如此一來，具有一致性與即時性的動畫效果讓使用者可以清楚知道自己的操作是否正確。關於架構規劃的部分，清單中提示了對於使用者不需要的資訊應該使用特定的軟體架構加以隱藏，這也是本系統所應用之 MVC 架構所能夠達到的效果，使用者在操作系統時只需要知道介面的操作方法，而不用考慮系統的運作細節。另一方面，清單要求確定系統資源的限制並不會影響使用者操作，其它搭配圖形化界面的系統常因爲硬體效能的不足或使用者過度限制效能而導致畫面延遲或卡頓，反而降低了使用者體驗，爲解決此問題，本系統將圖形界面的畫面更新設定在最高優先權，確保界面顯示的流暢。

可維護性設計清單中要求開發者釐清有哪些功能會被修改以及其修改後的影響範圍，本系統在需求分析時考慮到不同裝置所需要的實作時間不同，在各個裝置實作完成之後，必須多次整合系統與裝置之溝通，於是採取敏捷開發模式搭配物件導向概念。將負責與外部裝置溝通的功能模組化，每次新增裝置都能將程式的修改掌握在可控的範圍內，相較於傳統開發流程必須等待所有裝置實作完成才能進行整合測試，本系統之方法提升了開發效率與可維護性。而協調模型與資料模型的部分，以易用性與可維護性為開發重點，

採用的 MVC 架構確實的分離了介面端與資料端的實作，並且避免使用者或系統管理員修改任意資料導致系統出錯。此外，物件導向概念也有助於其它開發者或系統管理員在後續流程中針對特定功能之修改，達到良好的維護效果。

本論文透過核對設計清單的方法，確認系統滿足多數清單之建議事項以及所強調之品質屬性在實際應用中能夠發揮作用後，有效解釋系統之易用性與可維護性具有明確的改善。

五、結論

根據本系統案例，可以瞭解到品質屬性對於軟體系統之價值與重要性。軟體系統不僅僅是完整的呈現功能性需求，還需針對沒有統一標準的非功能性需求付出更多努力。另一方面，在驗收軟體系統時，無法量化測試的非功能性需求，容易成爲品質屬性的漏洞[7]。然而這也就是一系列品質屬性劇本所要達到的目的：藉由統一格式的劇本，來表達開發者或客戶對於非功能性需求的描述。本論文即是應用了核對清單，並且有效證明本論文之設計方法能夠改善或維持軟體系統的易用性與可維持性。

此外，在規劃軟體系統時，不同的品質屬性之間會互相影響。在有限的資源之下，如何能夠最大限度的滿足各個品質屬性，而無法兼顧的部分又要如何取捨，這將會是後續能夠研究的問題。

參考文獻

[1] J. Nielsen. (2016, April. 23). Usability 101: Introduction to Usability [Online]. Available: https://www.nngroup.com/articles/usability-101-introduction-to-usability/

[2] M. Mari and N. Eila, “The impact of maintainability on component-based software systems,” in Proceedings of Euromicro Conference, 2003. Proceedings. 29th, 2003, pp. 25–32.

[3] P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org.

[4] L. Bass, P. Clements, and R. Kazman, “*Software Architecture in Practice*,” 3rd edition, Addison Wesley, 2012, pp. 61-269.

[5] F. Paetsch, A. Eberlein, and F. Maurer, “Requirements engineering and agile software development,” in Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings, 2003, pp. 308–313.

[6] “XNA Tutorial for Game Development, Game Programming and Game Design | XNA Framework,” xnatutorial.com. [Online]. Available: http://www.xnatutorial.com/. [Accessed: 13-Jun-2016].

[7] L. Chung and J. C. S. do P. Leite, “On Non-Functional Requirements in Software Engineering,” in Conceptual Modeling: Foundations and Applications, A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Springer Berlin Heidelberg, 2009, pp. 363–379.

[8] E. Folmer and J. Bosch, “Architecting for usability: a survey,” Journal of Systems and Software, vol. 70, no. 1–2, pp. 61–78, Feb. 2004.