影音管理軟體

HW# 7

四電資四 101820302 施帛辰

四電資四 101820340 鄒令業

目錄

1 Requirement Document

1.1 Change History

Iteration I		
Version	**Description**	**Date**
1	Cover Page Problem Statement The Development Language	2016.2.28
2.1	Change History System Feature Use Case 1	2016.3.10
2.2	Modify Use Case 1 Add Use Case 2 & 3	2016.3.15
2.3	Add Use Case 4	2016.3.17
3.1	Domain Diagram Design	2016.3.29
4.1	Logic Architecture & SSD	2016.4.12
4.2	Contract & Operation Sequence Diagram SSD modified	2016.4.26
4.3	Class Diagram	2016.4.27
5.1	Add Initialize Seq and Destructor Seq and Implementation Class Diagram	2016.5.03
5.2	Add the difference between implementation and design and the information of source code	2016.5.05
6.1	Modify Use Case 1 and Use Case 2 Modify Domain Model Design Add SSD of Use Case 2	2016.5.17
7.1	Update Domain Model, Sequence Diagrams, Contract, and Design Class Diagram of Use Case 1	2016.5.31
7.2	Add Contracts of Use Case 2	2016.6.02
7.3	Edit Document	2016.6.13

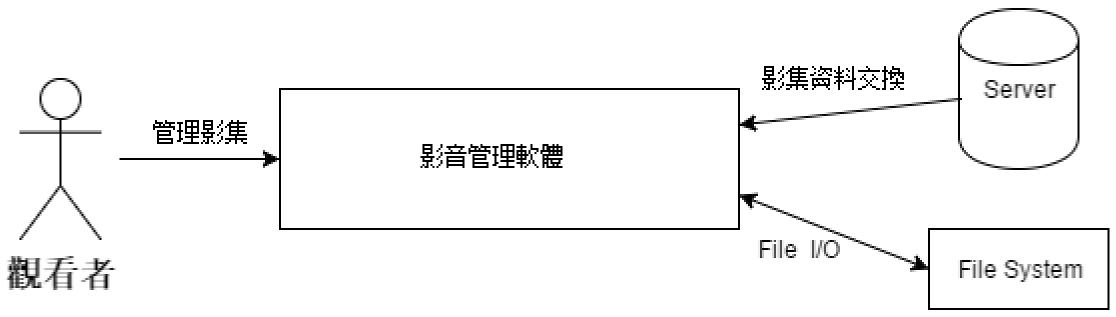
1.2 Problem Statement

日前電視劇風靡各年齡層，每周推出的新集數與每季上映的新劇，往往令那些電視劇觀賞者眼花撩亂，也讓那些想嘗試新劇的人不知如何下手，於是我們想到利用一套軟體記錄使用者的觀賞進度與喜好程度，同時管理使用者的影集，並可利用軟體推薦使用者可能喜歡的新劇，而使用者可在自家使用任意設備、平台進行記錄。

因為電視劇眾多且集數與觀賞記錄個不同，因此使用者容易遺失自己的影集進度，也不容易清楚究竟哪些新劇符合自己的胃口，透過此軟體，在每一次觀看前便可查詢先前的進度，而觀看後做上記錄與評論，就可以避免進度的遺失，也可藉評論讓軟體歸納使用者可能喜歡的類型，進而推薦新劇給有需要的人。

本軟體提供 GUI，讓使用者快速、簡便、直覺地操作，並記錄影集的觀看進度，也讓使用者可清楚自己喜好的類別與觀看歷史。

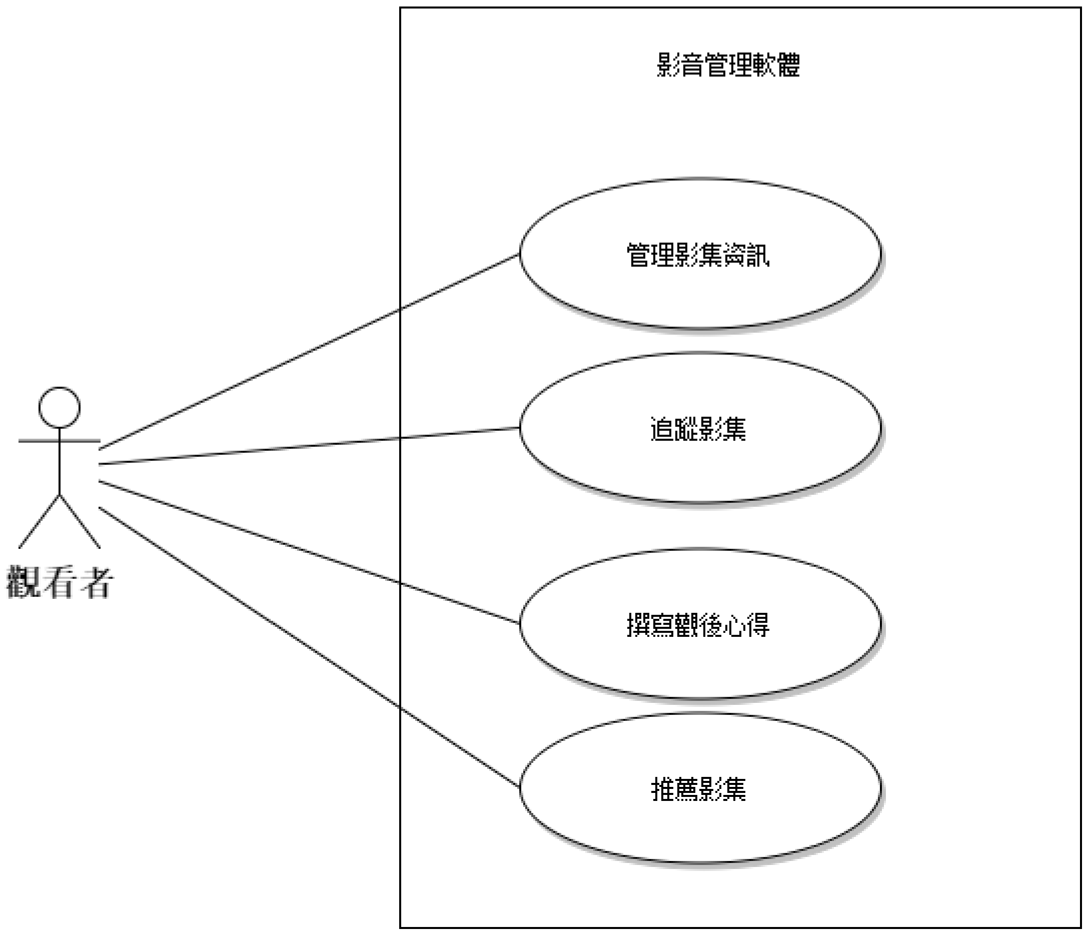
1.3 System Context Diagram



1.4 System Features

1. 管理影集資訊
2. 追蹤影集
3. 撰寫觀後心得
4. 推薦影集

1.5 Use Case Diagram



1.6 Use Case

1. Use Case 1 (New)

Use Case Name	管理影集資訊
Scope	影音管理軟體
Level	User Goal
Prime Actor	觀看者
Stakeholder and Interests	觀看者：管理影集的資訊，包含取得、新增、修改、刪除影集資訊
Preconditions	觀看者已安裝軟體
Success Guarantee	觀看者能成功管理影集資訊，並看見結果
Main Success Scenario	1. 觀看者啟動軟體 2. 軟體讀取 local 的存檔 3. 軟體告訴使用者正在更新資訊 4. 軟體自 server 抓取最新資訊 5. 軟體顯示更新完成 6. 觀看者手動對影集資訊進行管理

	7. 顯示正確結果
Extensions	2a. 若 local 沒有存檔，建立空的影集資訊列表 2b. 若檔案讀取失敗，通知觀看者，檔案損毀 4a. 沒有網路的情況下，通知觀看者，目前裝置尚未連接網路 4b. 如果沒有新的影集資訊，通知觀看者 6a. 若觀看者要新增影集資訊 1. 觀看者使用新增功能 2. 觀看者輸入影集資訊 3. 觀看者完成新增 6b. 若觀看者要匯入影集資訊 1. 觀看者使用匯入功能 2. 觀看者選擇要匯入的檔案 3. 軟體完成匯入 6c. 若觀看者要修改影集資訊 1. 觀看者選擇影集並修改 2. 觀看者輸入修改的資訊 3. 觀看者完成修改 6d. 若觀看者要刪除影集資訊 1. 觀看者選擇影集並刪除 2. 軟體再次確認影集的刪除 2a. 若觀看者確認刪除，軟體刪除影集資訊 6e. 若觀看者要手動更新網路資訊 1. 觀看者使用更新功能 2. 跳到步驟二 7a. 若觀看者想繼續管理影音資訊，回到步驟六 7b. 若觀看者關閉軟體，軟體把目前的影集資訊儲存到 local 中
Special Requirements	NFR-01、NFR-02、NFR-03
Technology and Data Variations List	網路影集資訊與私人影集資訊擁有個別的獨立編號。 影集資訊中，包含描述與類別。
Frequency of Occurrence	每次啟動後一定會發生至少一次
Open Issue	1. 影集資料格式尚未決定 2. 各部影集的獨立編號產生方式尚未決定

	3. 伺服器是要租用還要自己架設 4. 匯入的資料格式尚未決定

2. Use Case 2

Use Case Name	追蹤影集
Scope	影音管理軟體
Level	User Goal
Prime Actor	觀看者
Stakeholder and Interests	觀看者：對影集進行追縱，包含新增追蹤的影集、修改追蹤進度、取消追蹤
Preconditions	觀看者已安裝軟體
Success Guarantee	觀看者能成功追蹤影集，並看見結果
Main Success Scenario	1. 觀看者選擇影集 2. 軟體顯示影集資訊 3. 觀看者使用追蹤功能 4. 顯示正確結果
Extensions	3a. 若觀看者要追蹤新的影集，軟體紀錄開始追蹤 3b. 若觀看者要新增已追蹤的影集集數 1. 觀看者使用新增集數功能 2. 觀看者輸入集數資訊 3. 軟體顯示該影集的集數資訊 重複 2、3 步驟，直到觀看者不再新增集數 3c. 若觀看者要修改已追蹤的影集進度 1. 觀看者使用修改進度功能 2. 觀看者紀錄觀看的集數 2a. 若集數不存在，觀看者新增集數 3. 軟體要求輸入評論 3a. 若觀看者取消輸入，則不新增評論 3b. 若觀看者輸入評論，則新增一筆評論 3d. 若觀看者要取消已追蹤的影集 1. 觀看者使用取消追蹤的功能 2. 軟體再去確認影集取消追蹤 2a. 若觀看這確認，軟體取消影集的追蹤 3e. 若觀看者要恢復已取消追蹤的影集 1. 軟體恢復開始追蹤 2. 軟體讀取先前的集數資訊，並顯示
Special	NFR-03、NFR-01

Requirements	
Technology and Data Variations List	NA
Frequency of Occurrence	經常發生
Open Issue	NA

3. Use Case 3

Use Case Name	撰寫觀後心得
Scope	影音管理軟體
Level	User Goal
Prime Actor	觀看者
Stakeholder and Interests	觀看者：可以記錄自己的觀後心得
Preconditions	觀看者至少有一部已追蹤的影集
Success Guarantee	觀看者能成功紀錄下觀後心得
Main Success Scenario	1. 觀看者選擇影集 2. 軟體顯示影集資訊 3. 觀看者開始撰寫心得 4. 軟體定期儲存當前的心得資訊 5. 觀看者結束心得的撰寫
Extensions	3a. 若軟體發現上次沒有正確儲存的心得資料 1. 軟體詢問觀看者是否重新載入上次的心得 1a. 若觀看者確認，則軟體顯示上次心得 1b. 若觀看者取消，則軟體清除上次心得 5a. 若觀看者取消心得撰寫 1. 軟體詢問觀看者是否保留目前的心得 1a. 若觀看者確認，軟體保留當前心得記錄 1b. 若觀看者取消，軟體清除定期儲存的心得資訊 5b. 若觀看者完成心得 1. 軟體詢問觀看者是否儲存心得 1a. 若觀看者確認，軟體儲存並完成心得 1b. 若觀看者取消，觀看者可以繼續編輯心得
Special	NFR-03、NFR-01

Requirements	
Technology and Data Variations List	NA
Frequency of Occurrence	經常發生
Open Issue	NA

4. Use Case 4

Use Case Name	推薦影集
Scope	影音管理軟體
Level	User Goal
Prime Actor	軟體
Stakeholder and Interests	觀看者：希望可以看見軟體所推薦的影集 軟體：推薦影集給觀看者
Preconditions	觀看者可以有過去影集的追蹤紀錄
Success Guarantee	影集被推薦給觀看者
Main Success Scenario	1. 觀看者使用推薦功能 2. 軟體顯示數個推薦影集 3. 觀看者對推薦影集操作
Extensions	2a, 若軟體無法取得觀看者的資料 1. 軟體通知觀看者，無法推薦影集 2b. 若軟體的推薦影集皆被觀看者列入黑名單 1. 軟體通知觀看者，無非黑名單的推薦影集 3a. 若觀看者對某推薦影集有興趣 1. 觀看者對該部影集進行追蹤 3b. 若觀看者對某推薦影集不感興趣 1. 觀看者取消推薦該影集 2. 軟體將該影集列入黑名單 3c. 若觀看者希望再次推薦 1. 觀看者使用再次推薦功能 2. 回到步驟二 3d. 若觀看者不進行任何操作，直接離開
Special Requirements	NFR-01、NFR-03
Technology and Data Variations List	依照影集類別，軟體一次最多推薦 5 部影集
Frequency of	偶爾發生

Occurrence	
Open Issue	NA

1.7 Non-Functional Requirement and Constraints

NFR ID	Category	Description
NFR-01	Performance	資料讀寫需要在一秒內完成
NFR-02	Performance	伺服器要在 0.5 秒內回應
NFR-03	Usability	通知要夠大夠清楚
NFR-04	Usability	UI 要足夠友善
NFR-05	Reliability	資料讀寫必須正確無誤

1.8 Glossary

Term	Definition and Information	Format	Validation Rules	Aliases
影集	以單集為播放單位而長期放映的影片			影劇、Series
集數	一部影集的最小單位			Episode

1.9 Software Environments

The program will be written in C# language with Visual Studio.

2 Domain Class Model

2.1 Domain Class Diagram Showing Only Concepts

2.1.1 Class Identified (New)

- Business Transaction : Series、Episode、Blacklist、Tracing_List、Abandoned_List
- Products : Review、Command
- Description : Series_Description、Episode_Description
- Catalogs : Catalog
- Collaborating System : Server、Software、FileManager

P.S. 以上為使用類別清單 (Catalog List) 所找出來的 Concepts，其中有些 Concepts 並沒有劃入 Domain Model 中。

2.1.2 Bad Class

以 Attribute 的方式取代：

Series_Description, Episode_Description, BlackList, Tracing_List, Abandoned_List

2.1.3 Good Class (New)

Series, Episode：Domain 基礎物件

Review, Command：功能之一

Category：分類影音的物件

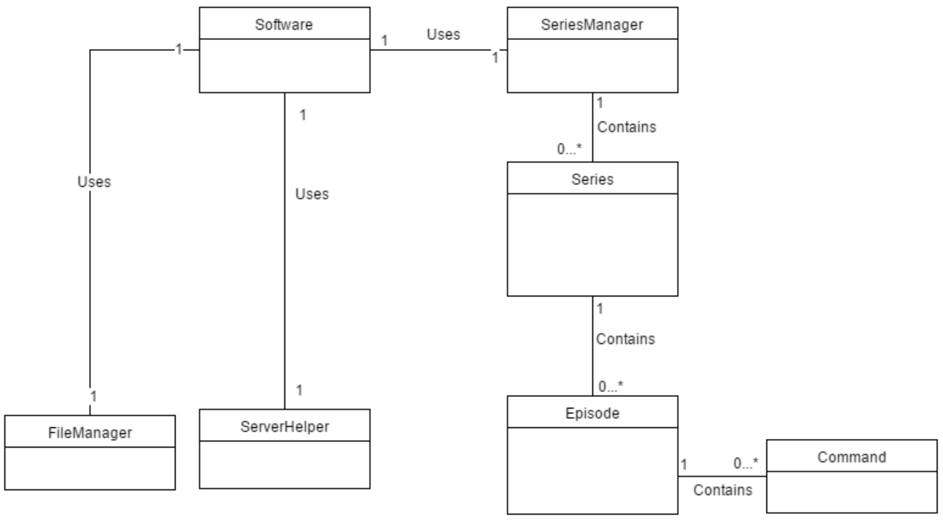
Server：負責外部連結的物件

Software：Root Object

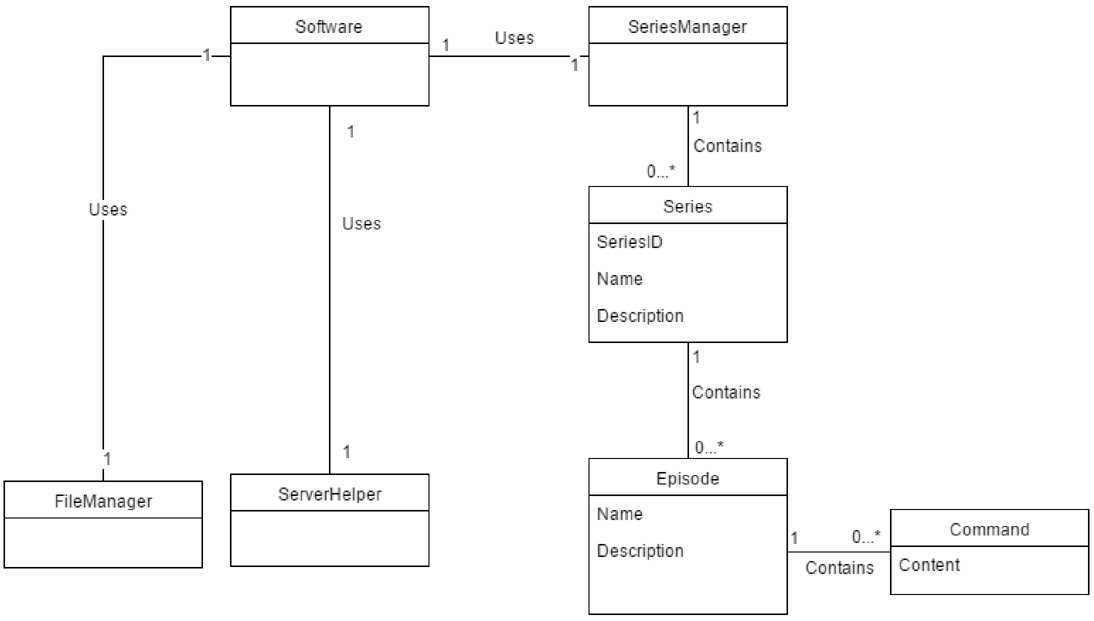
FileManager：負責管理軟體的存讀檔

2.2 Add Associations (New)

- ➢ **Software uses FileManager**
- ➢ **Software uses SeriesManager**
- ➢ **Software uses ServerHelper**
- ➢ **SeriesManager contains Series**
- ➢ **Series contains Episode**
- ➢ **Episode contains Command**

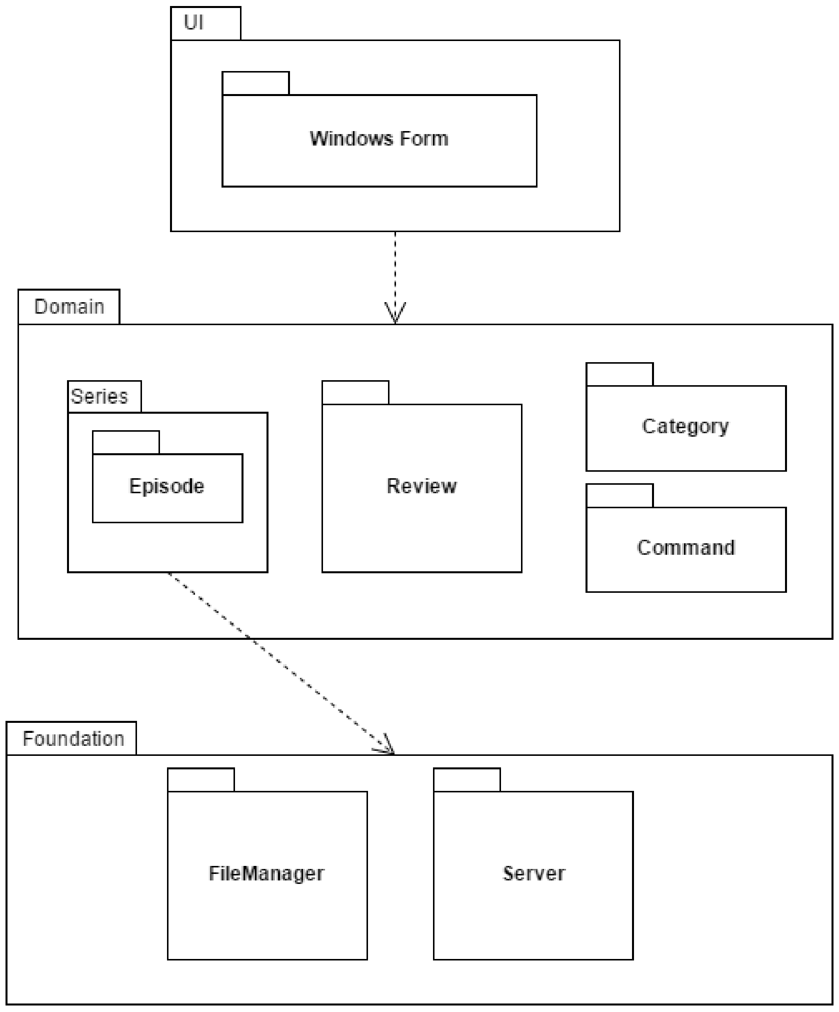


2.3 Add Attributes (New)



3. Design

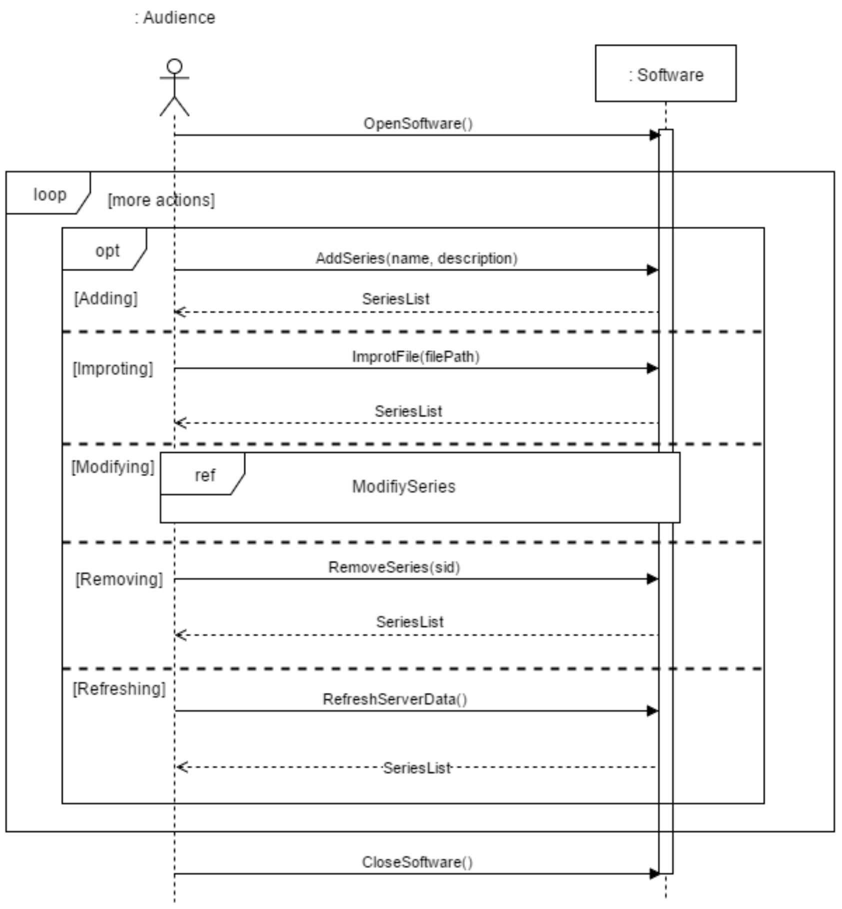
3.1 Logic Architecture (New)

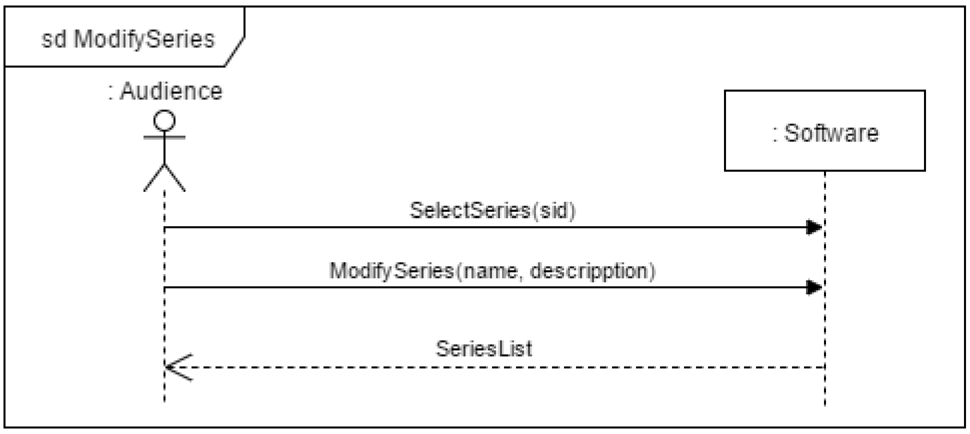


3.2 Use-Case Realizations with GRASP Patterns

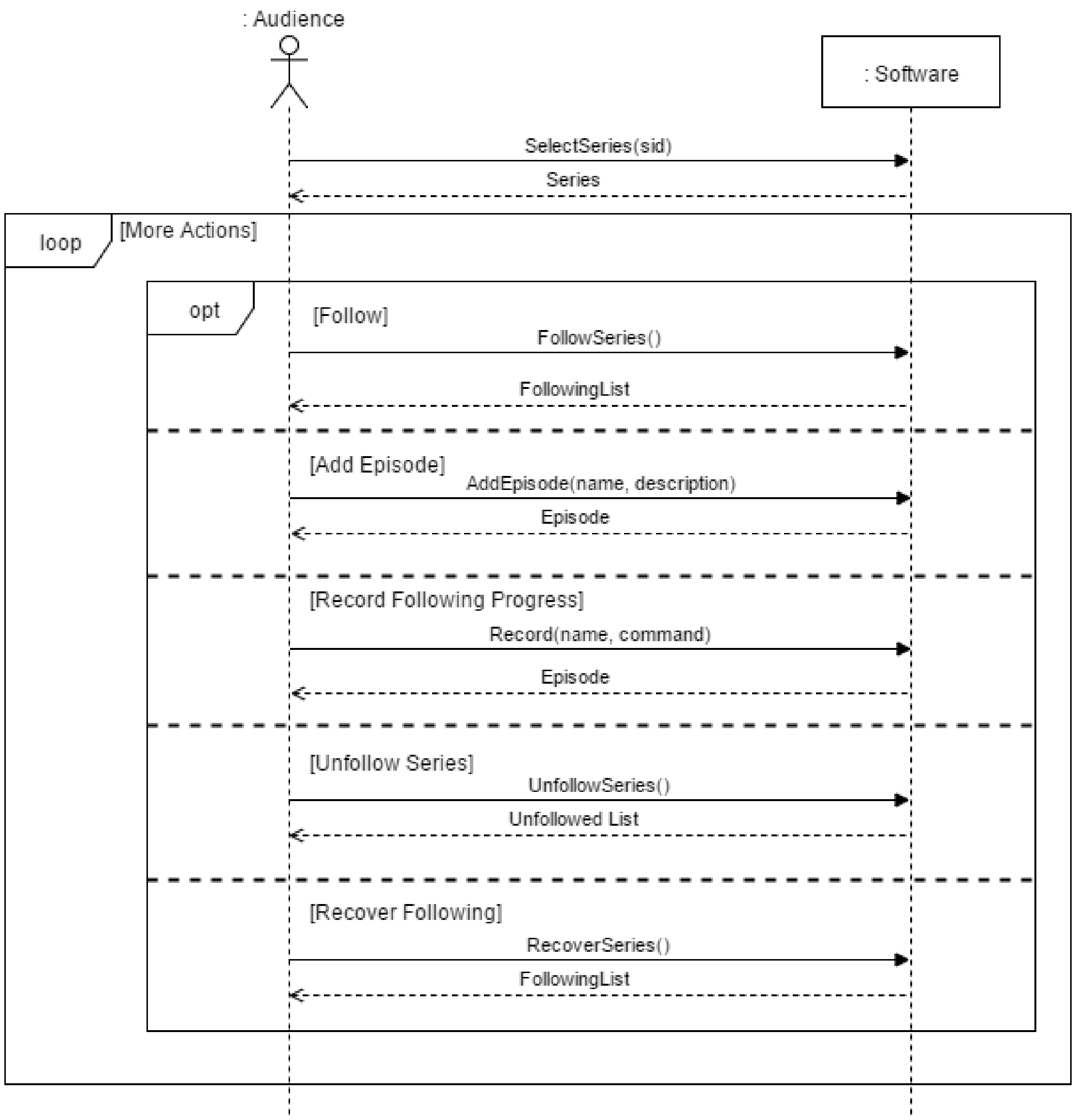
3.2.1 System Sequence Diagram

3.2.1.1 管理影集資訊(Use Case 1)





3.2.1.2 追蹤影集(Use Case 2) (New)



3.2.2 Contract

Contract ID	Operation Name
CO-01	AddSeries
CO-02	ImportFile
CO-03	SelectSeries
CO-04	ModifySeries
CO-05	RemoveSeries
CO-06	OpenSoftware
CO-07	CloseSoftware
CO-08	RefreshServerData
CO-09	FollowSeries
CO-10	AddEpisode

CO-11	Record
CO-12	UnfollowSeries
CO-13	RecoverSeries

3.2.2.1 AddSeries

Operation	AddSeries(name:string, description:string)
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	A new series instance s was created.(instance creation) S was added into series list of series manager.(association formed)

3.2.2.2 ImportFile

Operation	ImportFile(filePath:string)
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	A list of new series sl was created.(instance creation) Sl was added into series list of series manager.(association formed)

3.2.2.3 SelectSeries

Operation	SelectSeries(sid: int)
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	SeriesManager.selectedSeries became a series s.(attribute modification)

3.2.2.4 ModifySeries

Operation	ModifySeries (name: string, description: string)
Cross Reference	Use Case1
Preconditions	A series s has been selected.
Postconditions	s.name was modified.(instance creation) s.description was modified.(attribute modification)

3.2.2.5 RemoveSeries

Operation	RemoveSeries (sid: int)
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	A series s was removed from the list of series of series manager. (attribute modification)

3.2.2.6 OpenSoftware

Operation	OpenSoftware ()
Cross Reference	Use Case1
Preconditions	None
Postconditions	A serverHelper sh was created (instance creation) A fileManger fm was created (instance creation) A software s was created by sh and fm (instance creation & association informed) A seriesManager sm was created (instance creation) sm was associated by s (association informed) Server data and local data was added into sm (attribute modification)

3.2.2.7 CloseSoftware

Operation	CloseSoftware ()
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	Data was saved into local file system

3.2.2.8 RefreshServerData

Operation	RefreshServerData ()
Cross Reference	Use Case1
Preconditions	Software was opened
Postconditions	Data from Server was added into series manager (attribute modification)

3.2.2.9 FollowSeries

Operation	FollowSeries()
Cross Reference	Use Case2
Preconditions	Series has been selected
Postconditions	The selected series was added into the following list

	(attribute modification)

3.2.2.10 AddEpisode

Operation	AddEpisode(name : string, description : string)
Cross Reference	Use Case2
Preconditions	Series has been selected
Postconditions	An episode ep was created (instance cteation) Ep was added into the selected series (attribute modification)

3.2.2.11 Record

Operation	Record(name : string, command : string)
Cross Reference	Use Case2
Preconditions	Series has been selected
Postconditions	A command c was created (instance creation) A episode ep was found by name. (instance found) ep.isRead became true (attribute modification)

3.2.2.12 UnfollowSeries

Operation	UnfollowSeries()
Cross Reference	Use Case2
Preconditions	Series has been selected
Postconditions	The selected series was moved into unfollowing list (attribute modification)

3.2.2.13 RecoverSeries

Operation	FollowSeries()
Cross Reference	Use Case2
Preconditions	Series has been selected
Postconditions	The selected series was moved into following list from unfollowing list (attribute modification)

3.2.3 Operation Sequence Diagram

3.2.3.1 AddSeries

建立 Series

3.2.3.2 ImportFile

3.2.3.3 SelectSeries

3.2.3.4 ModifySeries

3.2.3.5 RemoveSeries

3.2.3.6 OpenSoftware

建立 SeriesManager

3.2.3.7 CloseSoftware

3.2.3.8 RefreshServerData

3.2.3.9 FollowSeries

3.2.3.10 AddEpisode

建立 Episode

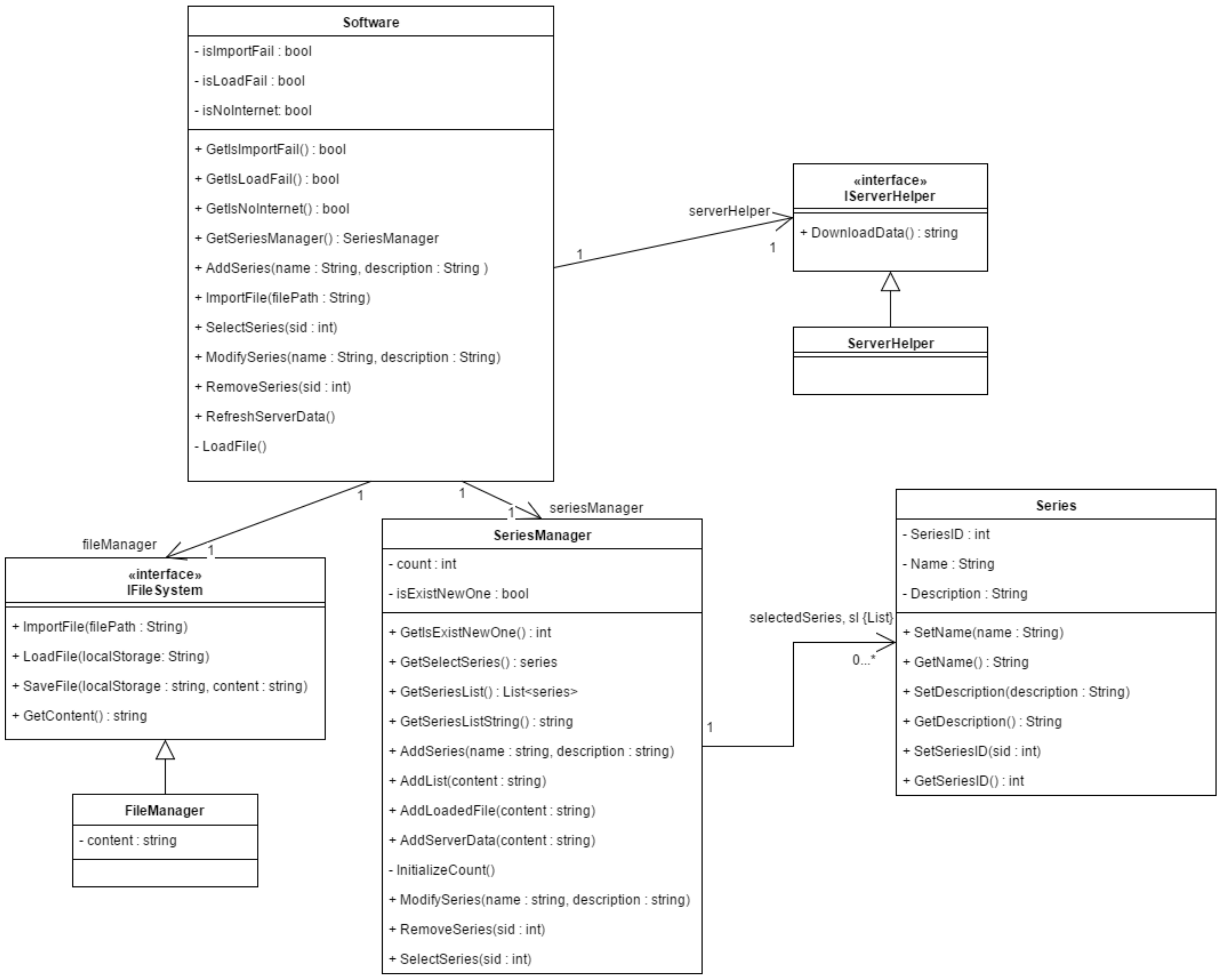
3.2.3.11 Record

建立 Command

3.2.3.12 UnfollowSeries

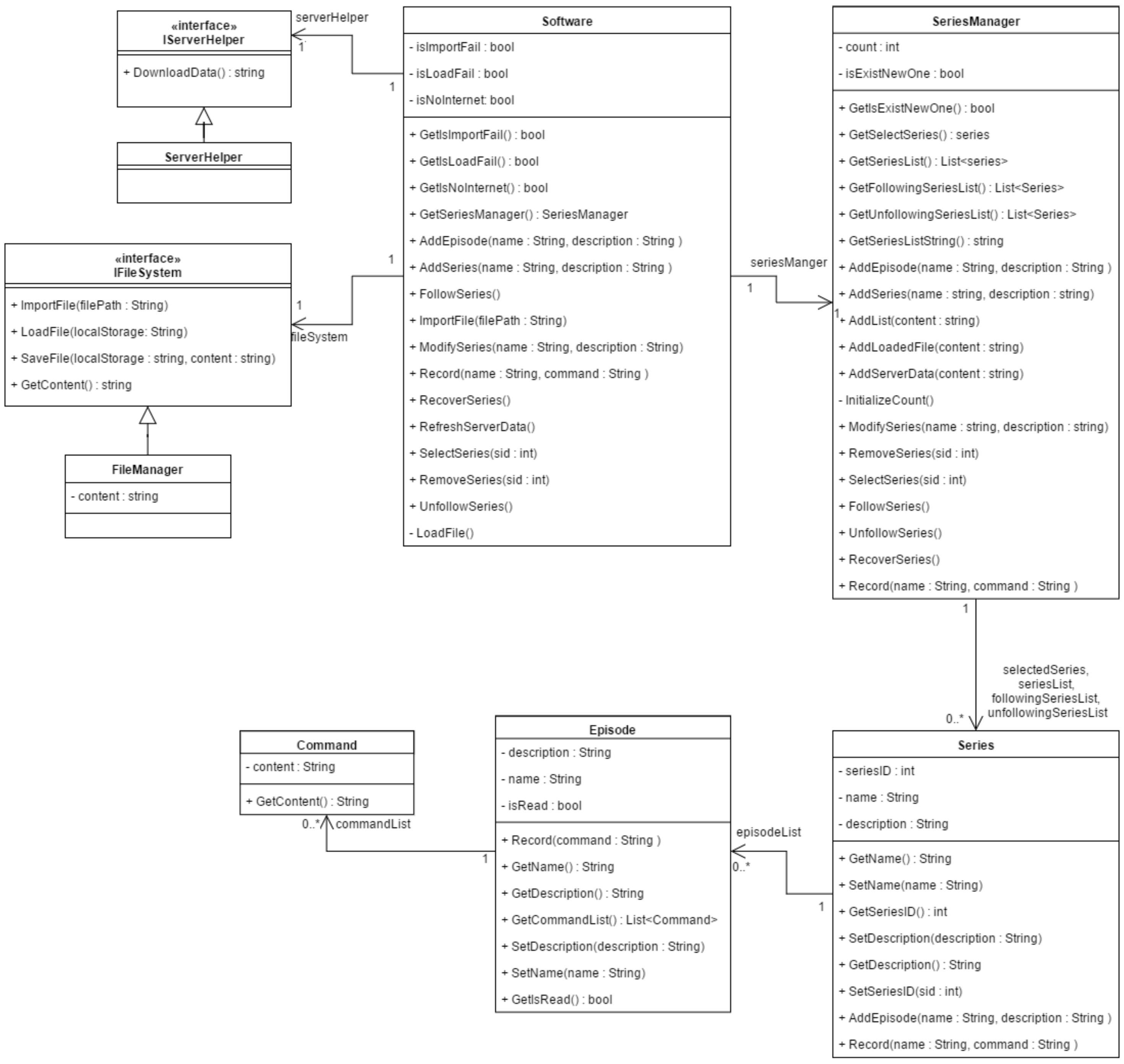
3.2.3.13 RecoverSeries

3.3 Design Class Diagram


Software
- isImportFail : bool
- isLoadFail : bool
- isNoInternet: bool
+ GetIsImportFail() : bool
+ GetIsLoadFail() : bool
+ GetIsNoInternet() : bool
+ GetSeriesManager() : SeriesManager
+ AddSeries(name : String, description : String)
+ ImportFile(filePath : String)
+ SelectSeries(sid : int)
+ ModifySeries(name : String, description : String)
+ RemoveSeries(sid : int)
+ RefreshServerData()
- LoadFile()
«interface»
IServerHelper
+ DownloadData() : string
ServerHelper
serverHelper
1
1
fileManager
1
1
seriesManager
1
1
«interface»
IFileSystem
+ ImportFile(filePath : String)
+ LoadFile(localStorage: String)
+ SaveFile(localStorage : string, content : string)
+ GetContent() : string
FileManager
- content : string
SeriesManager
- count : int
- isExistNewOne : bool
+ GetIsExistNewOne() : int
+ GetSelectSeries() : series
+ GetSeriesList() : List<series>
+ GetSeriesListString() : string
+ AddSeries(name : string, description : string)
+ AddList(content : string)
+ AddLoadedFile(content : string)
+ AddServerData(content : string)
- InitializeCount()
+ ModifySeries(name : string, description : string)
+ RemoveSeries(sid : int)
+ SelectSeries(sid : int)
selectedSeries, sl {List}
0...*
1
Series
- SeriesID : int
- Name : String
- Description : String
+ SetName(name : String)
+ GetName() : String
+ SetDescription(description : String)
+ GetDescription() : String
+ SetSeriesID(sid : int)
+ GetSeriesID() : int

4 Implementation Class Model (New)

4.1 Implementation Class Diagram (New)


«interface»
IServerHelper
+ DownloadData() : string
ServerHelper
serverHelper
1
1
Software
- isImportFail : bool
- isLoadFail : bool
- isNoInternet: bool
+ GetIsImportFail() : bool
+ GetIsLoadFail() : bool
+ GetIsNoInternet() : bool
+ GetSeriesManager() : SeriesManager
+ AddEpisode(name : String, description : String)
+ AddSeries(name : String, description : String)
+ FollowSeries()
+ ImportFile(filePath : String)
+ ModifySeries(name : String, description : String)
+ Record(name : String, command : String)
+ RecoverSeries()
+ RefreshServerData()
+ SelectSeries(sid : int)
+ RemoveSeries(sid : int)
+ UnfollowSeries()
- LoadFile()
«interface»
IFileSystem
+ ImportFile(filePath : String)
+ LoadFile(localStorage: String)
+ SaveFile(localStorage : string, content : string)
+ GetContent() : string
1
1
fileSystem
FileManager
- content : string
seriesManger
1
1
SeriesManager
- count : int
- isExistNewOne : bool
+ GetIsExistNewOne() : bool
+ GetSelectSeries() : series
+ GetSeriesList() : List<series>
+ GetFollowingSeriesList() : List<Series>
+ GetUnfollowingSeriesList() : List<Series>
+ GetSeriesListString() : string
+ AddEpisode(name : String, description : String)
+ AddSeries(name : string, description : string)
+ AddList(content : string)
+ AddLoadedFile(content : string)
+ AddServerData(content : string)
- InitializeCount()
+ ModifySeries(name : string, description : string)
+ RemoveSeries(sid : int)
+ SelectSeries(sid : int)
+ FollowSeries()
+ UnfollowSeries()
+ RecoverSeries()
+ Record(name : String, command : String)
1
selectedSeries,
seriesList,
followingSeriesList,
unfollowingSeriesList
0..*
Series
- seriesID : int
- name : String
- description : String
+ GetName() : String
+ SetName(name : String)
+ GetSeriesID() : int
+ SetDescription(description : String)
+ GetDescription() : String
+ SetSeriesID(sid : int)
+ AddEpisode(name : String, description : String)
+ Record(name : String, command : String)
1
episodeList
0..*
Episode
- description : String
- name : String
- isRead : bool
+ Record(command : String)
+ GetName() : String
+ GetDescription() : String
+ GetCommandList() : List<Command>
+ SetDescription(description : String)
+ SetName(name : String)
+ GetIsRead() : bool
1
commandList
0..*
Command
- content : String
+ GetContent() : String

4.2 The Difference between Implementation and Design Class Model (New)

4.2.1 Comparison with Design and Implementation Class (New)

Class	Method	Design	Imp.
Software(New)	AddSeries	Yes	Yes
	ImportFile	Yes	Yes
	SelectSeries	Yes	Yes
	ModifySeries	Yes	Yes
	RemoveSeries	Yes	Yes
	(New)FollowSeries	Yes	Yes
	(New)AddEpisode	Yes	Yes
	(New)Record	Yes	Yes
	(New)UnfollowSeries	Yes	Yes
	(New)RecoverSeries	Yes	Yes
FileManager	ImportFile	Yes	Yes
	GetList	Yes	Yes
	SaveFile	No	Yes
Series(New)	SetName	Yes	Yes
	SetDescription	Yes	Yes
	SetSeriesID	No	Yes
	GetName	Yes	Yes
	GetDescription	Yes	Yes
	GetSeriesID	Yes	Yes
	(New)GetEpisodeList	Yes	Yes
	(New)AddEpisode	Yes	Yes
	(New)Record	Yes	Yes
SeriesManager (New)	GetSeriesList	No	Yes
	GetSelectedSeries	No	Yes
	AddSeries	No	Yes
	AddList	No	Yes
	SelectSeries	No	Yes
	ModifySelectedSeries	No	Yes
	RemoveSeries	No	Yes
	InitializeCount	No	Yes
	(New)GetFollowingList	Yes	Yes
	(New)GetUnfollowingList	Yes	Yes
	(New)FollowSeries	Yes	Yes
	(New)AddEpisode	Yes	Yes

	(New)Record	Yes	Yes
	(New)UnfollowSeries	Yes	Yes
	(New)RecoverSeries	Yes	Yes
ServerHelper	DownloadData	No	Yes
Command(New)	(New)GetContent	Yes	Yes
Episode (New)	(New)GetName	Yes	Yes
	(New)GetCommandList	Yes	Yes
	(New)Record	Yes	Yes
	(New)GetDescription	Yes	Yes
	(New)SetName	Yes	Yes
	(New)SetDescription	Yes	Yes
	(New)GetIsRead	Yes	Yes

4.2.2 Summary of Implementation Class / Method Changed (New)

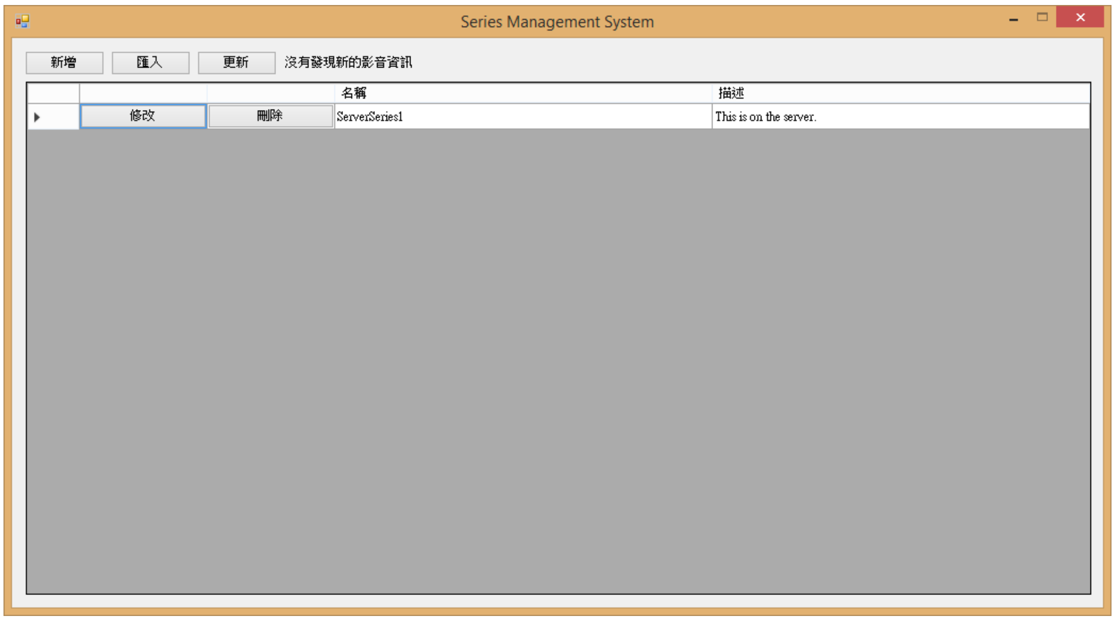
	Number of Added	Number of Removed	Number of Modified
Class	0	0	0
Method	0	0	0

4.3 The Lines of Code (New)

No	Class Name	Number of Methods	Line of codes without comment
1	Software	10	44
2	FileManger	3	13
3	Series	9	22
4	SeriesManager	15	54
5	ServerHelper	1	11
6	Episode	1	18
7	Command	7	4

5 Programing

5.1 Snapshot of System Execution



沒有發現新的影音資訊

			名稱	描述
▸	修改	刪除	ServerSeries1	This is on the server.

5.2 Source Code Listing

5.2.1 Softeware

```
using Newtonsoft.Json;
using SeriesManagementSystem.Foundation;
using System;
using System.Collections.Generic;
using System.Net;

namespace SeriesManagementSystem.Domain
{
    public class Software
    {
        private SeriesManager _seriesManager;
        private IFileSystem _fileManager;
        private IServerHelper _serverHelper;
        private bool _isNoInternet = false;
        private bool _isImportFail = false;
        private bool _isLoadFail = false;
        private const string LOCAL_STOREAGE = "./dat/data.dat";

        public Software(IServerHelper serverHelper, IFileSystem fileManager)
        {
            _serverHelper = serverHelper;
            _fileManager = fileManager;
            LoadFile();
            RefreshServerData();
        }

        private void LoadFile()
        {
            _isLoadFail = false;
            try
            {
                _fileManager.LoadFile(LOCAL_STOREAGE);
                _seriesManager = JsonConvert.DeserializeObject<SeriesManager>(_fileManager.Content);
            }
            catch (Exception)
            {
                _seriesManager = new SeriesManager();
                _isLoadFail = true;
            }
        }

        public void RefreshServerData()
        {
            _isNoInternet = false;
            try
```

```
		{
			_seriesManager.AddServerData(_serverHelper.DownloadData());
		}
		catch (WebException)
		{
			_isNoInternet = true;
		}
	}

	// Add a new series with name and description.
	public void AddSeries(string name, string description)
	{
		_seriesManager.AddSeries(name, description);
	}

	//Import series data from a file.
	public void ImportFile(string filePath)
	{
		_isImportFail = false;
		_fileManager.ImportFile(filePath);
		try
		{
			string content = _fileManager.Content;
			_seriesManager.AddList(content);
		}
		catch
		{
			_isImportFail = true;
		}
	}

	public void SelectSeries(int sid)
	{
		_seriesManager.SelectSeries(sid);
	}

	public void ModifySeries(string newName, string newDescription)
	{
		_seriesManager.ModifySelectedSeries(newName, newDescription);
	}

	public void RemoveSeries(int sid)
	{
		_seriesManager.RemoveSeries(sid);
	}

	public void FollowSeries()
	{
		_seriesManager.FollowSeries();
	}

	public void UnfollowSeries()
	{
		_seriesManager.UnfollowSeries();
	}

	public void RecoverSeries()
	{
		_seriesManager.RecoverSeries();
	}

	public void AddEpisode(string name, string description)
	{
		_seriesManager.AddEpisode(name, description);
	}

	public void Record(string name, string command)
	{
		_seriesManager.Record(name, command);
	}

	~Software()
	{
		string list = _seriesManager.SeriesListString;
		_fileManager.SaveFile(LOCAL_STOREAGE, list);
	}

	public SeriesManager SeriesManager
	{
		get
		{
			return _seriesManager;
		}
	}

	public bool IsNoInternet
```

```
        {
            get
            {
                return _isNoInternet;
            }
        }

        public bool IsImportFail
        {
            get
            {
                return _isImportFail;
            }
        }

        public bool IsLoadFail
        {
            get
            {
                return _isLoadFail;
            }
        }
    }
}
```

5.2.2 SeriesManager

```
using System;
using System.Collections.Generic;
using Newtonsoft.Json;
using System.Runtime.Serialization;

namespace SeriesManagementSystem.Domain
{
    [JsonObject(MemberSerialization.OptIn)]
    public class SeriesManager
    {
        [JsonProperty]
        private List<Series> _series = new List<Series>();
        [JsonProperty]
        private List<Series> _followingList = new List<Series>();
        [JsonProperty]
        private List<Series> _unfollowingList = new List<Series>();
        private Series _selectedSeries;
        private int _count = 0;
        private bool _isExistNewOne = false;

        #region Public Object
        public List<Series> SeriesList
        {
            get
            {
                return _series;
            }
        }

        public List<Series> FollowingList
        {
            get
            {
                return _followingList;
            }
        }

        public List<Series> UnfollowingList
        {
            get
            {
                return _unfollowingList;
            }
        }

        public Series SelectedSeries
        {
            get
            {
                return _selectedSeries;
            }
        }

        public string SeriesListString
        {
            get
            {
                return JsonConvert.SerializeObject(this);
            }
        }
```

```
public bool IsExistNewOne
{
    get
    {
        return _isExistNewOne;
    }
}
#endregion

public void AddSeries(String name, String description)
{
    Series series = new Series(name, description, _count++);
    _series.Add(series);
}

public void AddList(string content)
{
    List<Series> list = JsonConvert.DeserializeObject<List<Series>>(content) as List<Series>;
    foreach (Series series in list)
    {
        series.SeriesID = _count++;
    }
    _series.AddRange(list);
}

public void AddServerData(string content)
{
    List<Series> list = JsonConvert.DeserializeObject<List<Series>>(content);
    _isExistNewOne = false;
    foreach (Series series in list)
    {
        if (_series.Find((s) => s.SeriesID == series.SeriesID) == null)
        {
            _series.Add(series);
            _isExistNewOne = true;
        }
    }
}

public void SelectSeries(int sid)
{
    _selectedSeries = _series.Find((x) => x.SeriesID == sid);
}

public void ModifySelectedSeries(string newName, string newDescription)
{
    _selectedSeries.Name = newName;
    _selectedSeries.Description = newDescription;
}

public void RemoveSeries(int sid)
{
    Series series = _series.Find((s) => s.SeriesID == sid);
    _series.Remove(series);
}

public void FollowSeries()
{
    _followingList.Add(_selectedSeries);
}

public void UnfollowSeries()
{
    _followingList.Remove(_selectedSeries);
    _unfollowingList.Add(_selectedSeries);
}

public void RecoverSeries()
{
    _unfollowingList.Remove(_selectedSeries);
    _followingList.Add(_selectedSeries);
}

public void AddEpisode(string name, string description)
{
    _selectedSeries.AddEpisode(name, description);
}

public void Record(string name, string command)
{
    _selectedSeries.Record(name, command);
}

[OnDeserialized]
private void InitializeCount(StreamingContext context)
```

```
        {
            if (_series.Count != 0)
            {
                _series.Sort((s1, s2) =>
                {
                    return s1.SeriesID - s2.SeriesID;
                });
                int count = _series[_series.Count - 1].SeriesID + 1;
                if (count > 0)
                    _count = count;
            }
        }
    }
}
```

5.2.3 FileManager

```
using System;
using System.IO;
using System.Text;

namespace SeriesManagementSystem.Foundation
{
    public class FileManager : IFileSystem
    {
        private string _content = "{\"_series\":[],\"_followingList\":[],\"_unfollowingList\":[]}";

        public void ImportFile(string filePath)
        {
            String fileContext;
            using (var streamReader = new StreamReader(filePath, Encoding.UTF8))
            {
                fileContext = streamReader.ReadToEnd();
            }
            _content = fileContext;
        }

        public void LoadFile(string localStorage)
        {
            try { ImportFile(localStorage); }
            catch (Exception e)
            {
                if (e is FileNotFoundException | e is DirectoryNotFoundException)
                    _content = "{\"_series\":[],\"_followingList\":[],\"_unfollowingList\":[]}";
            }
        }

        public void SaveFile(string localStorage, string content)
        {
            using (var streamReader = new StreamWriter(localStorage, false))
            {
                streamReader.Write(content);
            }
        }

        public string Content
        {
            get { return _content; }
        }
    }
}
```

5.2.4 ServerHelper

```
using SeriesManagementSystem.Properties;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;

namespace SeriesManagementSystem.Foundation
{
    public class ServerHelper : IServerHelper
    {
        const string SERVER_URL = @"https://script.google.com/macros/s/";

        public string DownloadData()
        {
            string data;
            string url = SERVER_URL + Resources.GoogleWebAppID;
            HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(url);
            request.Method = "GET";
            using (WebResponse wr = request.GetResponse())
            {
                using (StreamReader sr = new StreamReader(wr.GetResponseStream(), Encoding.UTF8))
                {
                    data = sr.ReadToEnd();
                }
            }
            return data;
        }
    }
}
```

5.2.5 Series

```
using Newtonsoft.Json;
using System.Collections.Generic;

namespace SeriesManagementSystem.Domain
{
    public class Series
    {
        private int _seriesID;
        private string _name;
        private string _description;
        private List<Episode> _episodes;


        public Series(string name, string description)
        {
            _name = name;
            _description = description;
            _episodes = new List<Episode>();
        }

        [JsonConstructor]
        public Series(string name, string description, int seriesID) :
            this(name, description)
        {
            _seriesID = seriesID;
        }

        #region Public Properties
        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }

        public string Description
        {
            get
            {
                return _description;
            }
            set
            {
                _description = value;
            }
        }

        public int SeriesID
        {
            get
            {
                return _seriesID;
            }
            set
            {
                _seriesID = value;
            }
        }

        public List<Episode> Episodes
        {
            get
            {
                return _episodes;
            }
        }
        #endregion

        public void AddEpisode(string episodeName, string episodeDescription)
        {
            _episodes.Add(new Episode(episodeName, episodeDescription));
        }

        public void Record(string name, string command)
        {
            Episode episode = _episodes.Find((e) => e.Name == name);
            episode.Record(command);
        }
    }
}
```

5.2.6 Episode

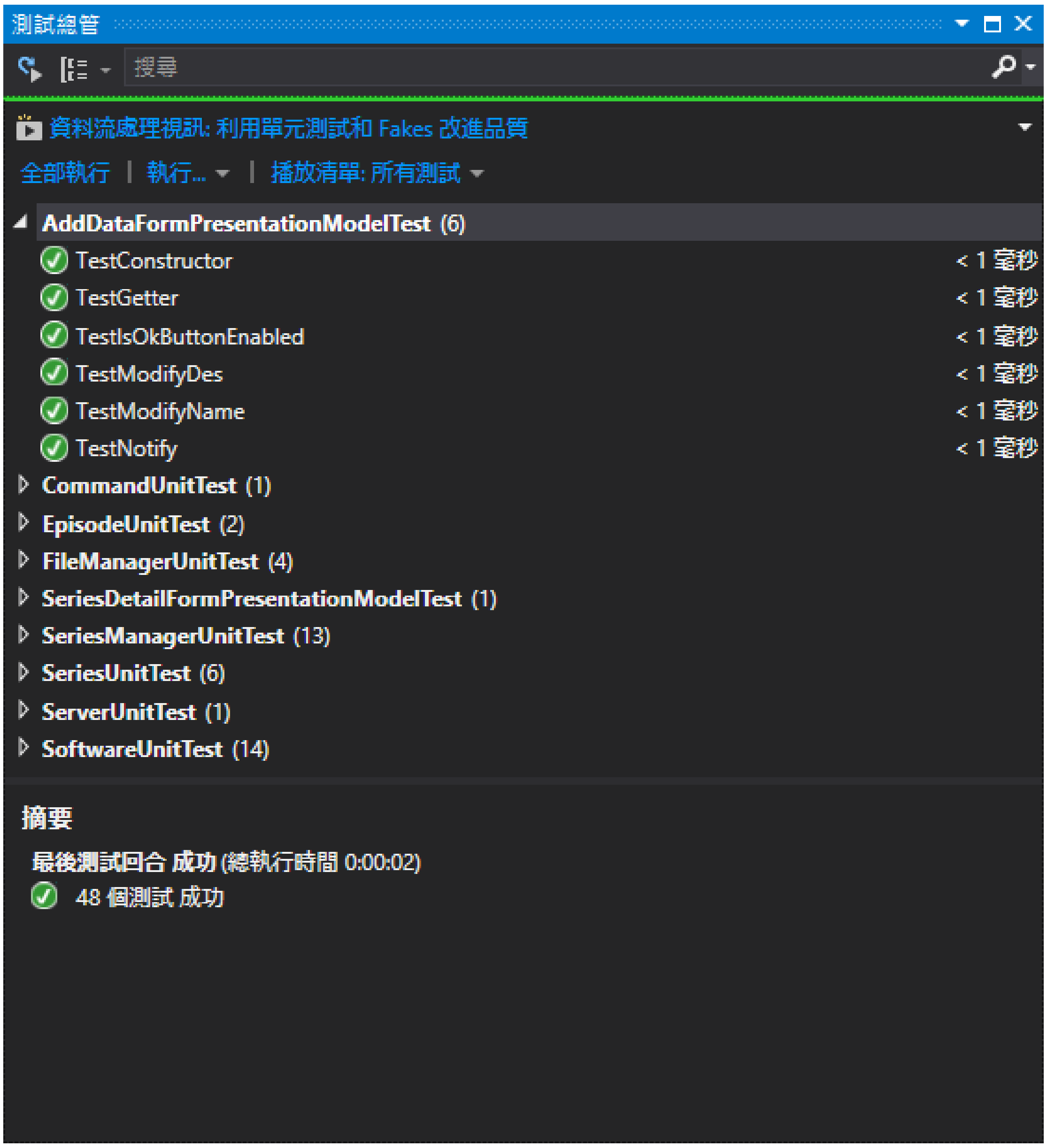
```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SeriesManagementSystem.Domain
{
    [JsonObject(MemberSerialization.OptIn)]
    public class Episode
    {
        [JsonProperty]
        private string _name;
        [JsonProperty]
        private string _description;
        [JsonProperty]
        private bool _isRead;
        [JsonProperty]
        private List<Command> _commandList = new List<Command>();

        [JsonConstructor]
        public Episode(string name, string description)
        {
            _name = name;
            _description = description;
        }

        #region Public Properties
        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }

        public string Description
        {
            get
            {
                return _description;
            }
            set
            {
                _description = value;
            }
        }

        public bool IsRead
        {
            get
            {
                //return _commandList.Count != 0;
                return _isRead;
            }
        }

        public List<Command> CommandList
        {
            get
            {
                return _commandList;
            }
        }
        #endregion

        public void Record(string command)
        {
            _isRead = true;
            if (command != String.Empty)
            {
                Command c = new Command(command);
                _commandList.Add(c);
            }
        }
    }
}
```

5.2.7 Command

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SeriesManagementSystem.Domain
{
    public class Command
    {
        private string _content;

        public Command(string content)
        {
            _content = content;
        }

        public String Content
        {
            get
            {
                return _content;
            }
        }
    }
}
```

6 Unit Test

6.1 Snapshot of Testing Result



資料流處理視訊: 利用單元測試和 Fakes 改進品質

全部執行 | 執行... | 播放清單: 所有測試

AddDataFormPresentationModelTest (6)
- ✓ TestConstructor < 1 毫秒
- ✓ TestGetter < 1 毫秒
- ✓ TestIsOkButtonEnabled < 1 毫秒
- ✓ TestModifyDes < 1 毫秒
- ✓ TestModifyName < 1 毫秒
- ✓ TestNotify < 1 毫秒

CommandUnitTest (1)
EpisodeUnitTest (2)
FileManagerUnitTest (4)
SeriesDetailFormPresentationModelTest (1)
SeriesManagerUnitTest (13)
SeriesUnitTest (6)
ServerUnitTest (1)
SoftwareUnitTest (14)

摘要

最後測試回合 成功 (總執行時間 0:00:02)

✓ 48 個測試 成功

6.2

6.3 Unit Test Code Listing

6.2.1 Softeware

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SeriesManagementSystem.Domain;
using System.IO;
using System.Text;
using System.Collections.Generic;
using SeriesManagementSystemUnitTest.FakeItem;

namespace SeriesManagementSystemUnitTest
{
	[TestClass]
	public class SoftwareUnitTest
	{
		Software _software;
		PrivateObject _privateObject;
		FakeFileSystem _fakeFileSystem;
```

```
        FakeServer _fakeServer;
        const string SeriesName = "Test Series";
        const string SeriesDescription = "This is a test description";
        const string ModifiedSeriesName = "modifiedSeries";
        const string ModifiedSeriesDescription = "this is a modified description";
        const string FILE_PATH = "./dat/data.dat";

        [TestInitialize()]
        public void Initialize()
        {
            _fakeFileSystem = new FakeFileSystem();
            _fakeServer = new FakeServer();
            _software = new Software(_fakeServer, _fakeFileSystem);
            _privateObject = new PrivateObject(_software, new PrivateType(typeof(Software)));
            for (int i = 0; i < 3; i++)
            {
                _software.AddSeries(SeriesName + i.ToString(), SeriesDescription + i.ToString());
            }
        }

        [TestMethod]
        public void TestAddSeries()
        {
            String name = "First Movie";
            String description = "The first movie in the world.";
            _software.AddSeries(name, description);
            Series s = GetLastSeries();
            Assert.AreEqual(name, s.Name);
            Assert.AreEqual(description, s.Description);
        }

        [TestMethod]
        public void TestImportFile()
        {
            String name = "First Movie";
            String description = "The first movie in the world.";
            int seriesID = 1;
            String fileContext = "[{ \"Name\":\"" + name + "\", \"Description\":\"" + description + "\", \"SeriesID\":"
+ seriesID + "}]";
            _fakeFileSystem.PrepareImportFile(fileContext);
            _software.ImportFile(FILE_PATH);
            Series s = GetLastSeries();
            Assert.AreEqual(name, s.Name);
            Assert.AreEqual(description, s.Description);
            // Fail route.
            _fakeFileSystem.PrepareImportFile("[{\"Name\"");
            Assert.IsFalse(_software.IsImportFail);
            _software.ImportFile(FILE_PATH);
            Assert.IsTrue(_software.IsImportFail);
        }

        [TestMethod]
        public void TestSelectSeries()
        {
            _software.SelectSeries(1);
            Assert.AreEqual(SeriesName + 1, GetSeriesManager().SelectedSeries.Name);
            Assert.AreEqual(SeriesDescription + 1, GetSeriesManager().SelectedSeries.Description);
        }

        [TestMethod]
        public void TestModifySeries()
        {
            _software.SelectSeries(1);
            Assert.AreEqual(SeriesName + 1, GetSeriesManager().SelectedSeries.Name);
            Assert.AreEqual(SeriesDescription + 1, GetSeriesManager().SelectedSeries.Description);
            _software.ModifySeries(ModifiedSeriesName, ModifiedSeriesDescription);
            Assert.AreEqual(ModifiedSeriesName, GetSeriesManager().SelectedSeries.Name);
            Assert.AreEqual(ModifiedSeriesDescription, GetSeriesManager().SelectedSeries.Description);
        }

        [TestMethod]
        public void TestRemoveSeries()
        {
            SeriesManager seriesManager = GetSeriesManager();
            List<Series> seriesList = seriesManager.SeriesList;
            Assert.AreEqual(4, seriesList.Count);
            _software.RemoveSeries(1);
            Assert.AreEqual(3, seriesList.Count);
            Assert.IsNull(seriesList.Find((s) => s.Name == SeriesName + 1));
        }

        [TestMethod]
        public void TestGetSeriesManager()
        {
            SeriesManager seriesManager = GetSeriesManager();
            Assert.AreEqual(seriesManager, _software.SeriesManager);
```

```
}

[TestMethod]
public void TestDestructor()
{
	string seriesListString = GetSeriesManager().SeriesListString;
	string expected = FILE_PATH + seriesListString;
	_software = null;
	_privateObject = null;
	GC.Collect();
	GC.WaitForPendingFinalizers();
	Assert.AreEqual(expected, _fakeFileSystem.Content);
}

[TestMethod]
public void TestAddServerData()
{
	Assert.IsFalse(_software.IsNoInternet);
	_fakeServer.IsDownloadFail = true;
	_software.RefreshServerData();
	Assert.IsTrue(_software.IsNoInternet);
	_fakeServer.IsDownloadFail = false;
	_software.RefreshServerData();
	Assert.IsFalse(_software.IsNoInternet);
}

[TestMethod]
public void TestLoadFile()
{
	Assert.IsFalse(_software.IsLoadFail);
	_fakeFileSystem.IsLoadFail = true;
	_privateObject.Invoke("LoadFile");
	Assert.IsTrue(_software.IsLoadFail);
	_fakeFileSystem.IsLoadFail = false;
	_privateObject.Invoke("LoadFile");
	Assert.IsFalse(_software.IsLoadFail);
}

[TestMethod]
public void TestFollowSeries()
{
	GetSeriesManager().SelectSeries(2);
	_software.FollowSeries();
	Series s = GetLastFollowingSeries();
	Assert.AreEqual(1, GetSeriesManager().FollowingList.Count);
	Assert.AreEqual(SeriesName + 2, s.Name);
	Assert.AreEqual(SeriesDescription + 2, s.Description);
}

[TestMethod]
public void TestUnfollowSeries()
{
	GetSeriesManager().SelectSeries(2);
	GetSeriesManager().FollowSeries();
	_software.UnfollowSeries();
	Assert.AreEqual(1, GetSeriesManager().UnfollowingList.Count);
	Assert.AreEqual(0, GetSeriesManager().FollowingList.Count);
	Series s = GetLastUnfollowingSeries();
	Assert.AreEqual(SeriesName + 2, s.Name);
	Assert.AreEqual(SeriesDescription + 2, s.Description);
	int index = GetSeriesManager().FollowingList.IndexOf(s);
	Assert.AreEqual(-1, index);
}

[TestMethod]
public void TestRecoverSeries()
{
	GetSeriesManager().SelectSeries(2);
	GetSeriesManager().FollowSeries();
	GetSeriesManager().UnfollowSeries();
	_software.RecoverSeries();
	Assert.AreEqual(0, GetSeriesManager().UnfollowingList.Count);
	Assert.AreEqual(1, GetSeriesManager().FollowingList.Count);
	Series s = GetLastFollowingSeries();
	Assert.AreEqual(SeriesName + 2, s.Name);
	Assert.AreEqual(SeriesDescription + 2, s.Description);
	int index = GetSeriesManager().UnfollowingList.IndexOf(s);
	Assert.AreEqual(-1, index);
}

[TestMethod]
public void TestAddEpisode()
{
	GetSeriesManager().SelectSeries(2);
	Series s = GetSeriesManager().SelectedSeries;
	string eName = "e1", eDesc = "how it is going?";
```

```
            Assert.AreEqual(0, s.Episodes.Count);
            _software.AddEpisode(eName, eDesc);
            Assert.AreEqual(1, s.Episodes.Count);
            Episode e = s.Episodes[s.Episodes.Count - 1];
            Assert.AreEqual(eName, e.Name);
            Assert.AreEqual(eDesc, e.Description);
        }

        [TestMethod]
        public void TestRecord()
        {
            string eName = "goodEp", eDesc = "Hero is dead.";
            string command = "So suprise!";
            GetSeriesManager().SelectSeries(1);
            Series s = GetSeriesManager().SelectedSeries;
            s.AddEpisode(eName, eDesc);
            Episode e = s.Episodes[0];
            _software.Record(eName, command);
            Assert.AreEqual(1, e.CommandList.Count);
            Assert.IsTrue(e.IsRead);
        }

        #region Get Private Object
        private Series GetLastSeries()
        {
            SeriesManager seriesManager = GetSeriesManager();
            Assert.IsNotNull(seriesManager.SeriesList);
            Assert.IsTrue(seriesManager.SeriesList.Count > 0, "No any series in the list!");
            return seriesManager.SeriesList[seriesManager.SeriesList.Count - 1];
        }

        private Series GetLastFollowingSeries()
        {
            SeriesManager seriesManager = GetSeriesManager();
            Assert.IsNotNull(seriesManager.FollowingList);
            Assert.IsTrue(seriesManager.FollowingList.Count > 0, "No any series in the following list!");
            return seriesManager.FollowingList[seriesManager.FollowingList.Count - 1];
        }

        private Series GetLastUnfollowingSeries()
        {
            SeriesManager seriesManager = GetSeriesManager();
            Assert.IsNotNull(seriesManager.UnfollowingList);
            Assert.IsTrue(seriesManager.UnfollowingList.Count > 0, "No any series in the following list!");
            return seriesManager.UnfollowingList[seriesManager.UnfollowingList.Count - 1];
        }

        private SeriesManager GetSeriesManager()
        {
            return _privateObject.GetField("_seriesManager") as SeriesManager;
        }
        #endregion
    }
}
```

6.2.2 SeriesManager

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json;
using SeriesManagementSystem.Domain;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class SeriesManagerUnitTest
    {
        SeriesManager _seriesManager;
        Series[] _series;
        const int SeriesID = 0;
        const string SeriesName = "manager's series";
        const string SeriesDescription = "it is a series' description of manager";
        const string ModifiedSeriesName = "modifiedSeries";
        const string ModifiedSeriesDescription = "this is a modified description";

        [TestInitialize]
        public void Initialize()
        {
            _seriesManager = new SeriesManager();
            _series = new Series[3];
            for (int i = 0; i < 3; i++)
            {
                _series[i] = new Series(SeriesName + i.ToString(), SeriesDescription + i.ToString(), SeriesID + i);
            }
```

```
}

[TestMethod]
public void TestInitializeCount()
{
    PrivateObject privateObject = new PrivateObject(_seriesManager);
    Assert.AreEqual(0, privateObject.GetFieldOrProperty("_count"));
    privateObject.SetField("_series", new List<Series>(_series));
    privateObject.Invoke("InitializeCount", new StreamingContext());
    Assert.AreEqual(3, privateObject.GetFieldOrProperty("_count"));
}

/// <summary>
/// Test function of AddSeries
/// </summary>
[TestMethod]
public void TestAdd()
{
    // test AddSeries function with one parameter, series
    _seriesManager.AddSeries(_series[0].Name, _series[0].Description);
    List<Series> seriesList = GetSeriesList();
    Series series = seriesList.Last();
    Assert.AreEqual(series.Name, _series[0].Name);
    Assert.AreEqual(series.Description, _series[0].Description);

    // test AddSeries function with two parameter, name ,and description
    Initialize();
    _seriesManager.AddSeries(SeriesName, SeriesDescription);
    seriesList = GetSeriesList();
    series = seriesList.Last();
    Assert.AreEqual(series.Name, SeriesName);
    Assert.AreEqual(series.Description, SeriesDescription);
}

/// <summary>
/// test the function of AddRange with a parameter, List<Series>
/// </summary>
[TestMethod]
public void TestAddRange()
{
    List<Series> content = GetSeriesList();
    Assert.IsTrue(content.Count == 0, "the initialized series list is not empty");
    List<Series> seriesList = new List<Series>(_series);
    string contentString = JsonConvert.SerializeObject(seriesList);
    _seriesManager.AddList(contentString);
    content = GetSeriesList();
    Assert.IsTrue(content.Count != 0, "the series list is still empty after adding a list of series");
}

/// <summary>
/// Test function of SelectSeries
/// </summary>
[TestMethod]
public void TestSelectSeries()
{
    // add series into series manager
    List<Series> seriesList = GetSeriesList();
    seriesList.AddRange(new List<Series>(_series));

    // test initialization of selected series is empty
    Assert.IsNull(_seriesManager.SelectedSeries);

    // test selected series after selecting the series
    _seriesManager.SelectSeries(2);
    Assert.AreEqual(_series[2], _seriesManager.SelectedSeries);
    _seriesManager.SelectSeries(1);
    Assert.AreEqual(_series[1], _seriesManager.SelectedSeries);

    // test if manager does not find the series in the list, it returns null
    _seriesManager.SelectSeries(10);
    Assert.IsNull(_seriesManager.SelectedSeries);
}

[TestMethod]
public void TestModifiedSelectedSeries()
{
    GetSeriesList().AddRange(new List<Series>(_series));

    _seriesManager.SelectSeries(2);
    Assert.AreEqual(_series[2], _seriesManager.SelectedSeries);

    _seriesManager.ModifySelectedSeries(ModifiedSeriesName, ModifiedSeriesDescription);
    Assert.AreEqual(ModifiedSeriesName, _seriesManager.SelectedSeries.Name);
    Assert.AreEqual(ModifiedSeriesDescription, _seriesManager.SelectedSeries.Description);

    Assert.AreEqual(ModifiedSeriesName, GetSeriesList().Find((x) => x.SeriesID == 2).Name);
```

```
            Assert.AreEqual(ModifiedSeriesDescription, GetSeriesList().Find((x) => x.SeriesID == 2).Description);
        }

        [TestMethod]
        public void TestRemoveSeries()
        {
            List<Series> seriesList = GetSeriesList();
            seriesList.AddRange(new List<Series>(_series));
            Assert.AreEqual(3, seriesList.Count);
            _seriesManager.RemoveSeries(1);
            Assert.AreEqual(2, seriesList.Count);
            Assert.AreEqual(-1, seriesList.IndexOf(_series[1]));
        }

        [TestMethod]
        public void TestAddServerData()
        {
            List<Series> seriesList = GetSeriesList();
            string content = "[{\"Name\":\"ServerSeries1\",\"Description\":\"This is on the server.\",\"SeriesID\":-
256}]";
            Assert.IsFalse(_seriesManager.IsExistNewOne);
            _seriesManager.AddServerData(content);
            Assert.IsTrue(_seriesManager.IsExistNewOne);
            Assert.AreEqual(1, seriesList.Count);
            _seriesManager.AddServerData(content);
            Assert.IsFalse(_seriesManager.IsExistNewOne);
            Assert.AreEqual(1, seriesList.Count);
        }

        [TestMethod]
        public void TestFollowSeries()
        {
            PrivateObject privateObject = new PrivateObject(_seriesManager);
            privateObject.SetField("_series", new List<Series>(_series));
            privateObject.SetField("_selectedSeries", _series[2]);
            _seriesManager.FollowSeries();
            List<Series> followingList = privateObject.GetField("_followingList") as List<Series>;
            Series s = followingList[followingList.Count - 1];
            Assert.AreEqual(1, followingList.Count);
            Assert.AreEqual(SeriesName + 2, s.Name);
            Assert.AreEqual(SeriesDescription + 2, s.Description);
        }

        [TestMethod]
        public void TestUnfollowSeries()
        {
            PrivateObject privateObject = new PrivateObject(_seriesManager);
            List<Series> followingList = privateObject.GetField("_followingList") as List<Series>;
            List<Series> unfollowingList = privateObject.GetField("_unfollowingList") as List<Series>;
            followingList.AddRange(new List<Series>(_series));
            privateObject.SetField("_selectedSeries", _series[2]);
            Assert.AreEqual(0, unfollowingList.Count);
            Assert.AreEqual(3, followingList.Count);
            _seriesManager.UnfollowSeries();
            Assert.AreEqual(1, unfollowingList.Count);
            Assert.AreEqual(2, followingList.Count);
            Series s = unfollowingList[unfollowingList.Count - 1];
            Assert.AreEqual(SeriesName + 2, s.Name);
            Assert.AreEqual(SeriesDescription + 2, s.Description);
            int index = followingList.IndexOf(s);
            Assert.AreEqual(-1, index);
        }

        [TestMethod]
        public void TestRecoverSeries()
        {
            PrivateObject privateObject = new PrivateObject(_seriesManager);
            List<Series> followingList = privateObject.GetField("_followingList") as List<Series>;
            List<Series> unfollowingList = privateObject.GetField("_unfollowingList") as List<Series>;
            unfollowingList.AddRange(new List<Series>(_series));
            privateObject.SetField("_selectedSeries", _series[2]);
            Assert.AreEqual(3, unfollowingList.Count);
            Assert.AreEqual(0, followingList.Count);
            _seriesManager.RecoverSeries();
            Assert.AreEqual(2, unfollowingList.Count);
            Assert.AreEqual(1, followingList.Count);
            Series s = followingList[followingList.Count - 1];
            Assert.AreEqual(SeriesName + 2, s.Name);
            Assert.AreEqual(SeriesDescription + 2, s.Description);
            int index = unfollowingList.IndexOf(s);
            Assert.AreEqual(-1, index);
        }

        [TestMethod]
        public void TestAddEpisode()
        {
            PrivateObject privateObject = new PrivateObject(_seriesManager);
            List<Series> followingList = privateObject.GetField("_followingList") as List<Series>;
            Series s = _series[2];
            string eName = "e1", eDesc = "how it is going?";
            followingList.AddRange(_series);
            privateObject.SetField("_selectedSeries", s);
            Assert.AreEqual(0, s.Episodes.Count);
            _seriesManager.AddEpisode(eName, eDesc);
            Assert.AreEqual(1, s.Episodes.Count);
            Episode e = s.Episodes[s.Episodes.Count - 1];
            Assert.AreEqual(eName, e.Name);
            Assert.AreEqual(eDesc, e.Description);
        }

        [TestMethod]
        public void TestRecord()
        {
            string eName = "goodEp", eDesc = "Hero is dead.";
            string command = "So suprise!";
            PrivateObject privateObject = new PrivateObject(_seriesManager);
            List<Series> followingList = privateObject.GetField("_followingList") as List<Series>;
            Series s = _series[1];
            s.AddEpisode(eName, eDesc);
            Episode e = s.Episodes[0];
            followingList.Add(s);
            privateObject.SetField("_selectedSeries", s);
            _seriesManager.Record(eName, command);
            Assert.AreEqual(1, e.CommandList.Count);
            Assert.IsTrue(e.IsRead);
        }

        [TestMethod]
        public void TestToJson()
        {
            Series s = new Series("s1", "456");
            s.AddEpisode("e1", "sad");
            _seriesManager.SeriesList.Add(s);
            var jSetting = new JsonSerializerSettings();
            jSetting.Formatting = Formatting.Indented;
            String json = JsonConvert.SerializeObject(_seriesManager, jSetting);
            SeriesManager sm = JsonConvert.DeserializeObject<SeriesManager>(json);
            Assert.AreEqual("s1", sm.SeriesList[0].Name);
            Assert.AreEqual("456", sm.SeriesList[0].Description);
            Assert.AreEqual("e1", sm.SeriesList[0].Episodes[0].Name);
            Assert.AreEqual("sad", sm.SeriesList[0].Episodes[0].Description);
        }

        /// <summary>
        /// get the series list of series manager
        /// </summary>
        /// <returns></returns>
        private List<Series> GetSeriesList()
        {
            return _seriesManager.SeriesList;
        }
    }
}
```

6.2.3 FileManager

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SeriesManagementSystem.Foundation;
using System;
using System.IO;
using System.Text;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class FileManagerUnitTest
    {
        private FileManager _fileManager;
        private PrivateObject _privateObject;
        private const string LOCAL_STORAGE = "./testFileManager.txt";
        private const string EMPTY_CONTENT = "{\"_series\":[],\"_followingList\":[],\"_unfollowingList\":[]}";

        [TestInitialize()]
        public void Initialize()
        {
            _fileManager = new FileManager();
            _privateObject = new PrivateObject(_fileManager, new PrivateType(typeof(FileManager)));
            Assert.AreEqual(EMPTY_CONTENT, _privateObject.GetField("_content"));
        }

        [TestCleanup]
        public void CleanUp()
        {
            if (File.Exists(LOCAL_STORAGE))
                File.Delete(LOCAL_STORAGE);
        }

        [TestMethod]
        public void TestLoadFile()
        {
            string testString = "Gorira parrrrrrty";
            _fileManager.LoadFile(LOCAL_STORAGE);
            Assert.AreEqual(EMPTY_CONTENT, _privateObject.GetField("_content"));
            PrepareFile(LOCAL_STORAGE, testString);
            _fileManager.LoadFile(LOCAL_STORAGE);
            Assert.AreEqual(testString, _privateObject.GetField("_content"));
        }

        [TestMethod]
        public void TestGetContent()
        {
            string testString = "Banana usually drop.";
            _privateObject.SetField("_content", testString);
            Assert.AreEqual(testString, _fileManager.Content);
        }

        [TestMethod]
        public void TestImportFile()
        {
            string testString = "Why monkey can't talk?";
            PrepareFile(LOCAL_STORAGE, testString);
            _fileManager.ImportFile(LOCAL_STORAGE);
            Assert.AreEqual(testString, _privateObject.GetField("_content"));
        }

        [TestMethod]
        public void TestSaveFile()
        {
            string testString = "Super monkey fly bat.";
            PrepareFile(LOCAL_STORAGE, "[]");
            _fileManager.SaveFile(LOCAL_STORAGE, testString);
            // test the file contains the string
            String fileContext;
            using (var streamReader = new StreamReader(LOCAL_STORAGE, Encoding.UTF8))
            {
                fileContext = streamReader.ReadToEnd();
            }
            Assert.AreEqual(testString, fileContext);
        }

        /// <summary>
        /// this function is used to prepare a file with some setting
        /// </summary>
        /// <param name="path">the file's location</param>
        /// <param name="content">file's content</param>
        private void PrepareFile(string path, string content)
        {
            using (var streamReader = new StreamWriter(path, false))
            {
                streamReader.Write(content);
            }
        }
    }
}
```

6.2.4 ServerHelper

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json;
using SeriesManagementSystem.Domain;
using SeriesManagementSystem.Foundation;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class ServerHelperUnitTest
    {
        ServerHelper _server;

        [TestInitialize]
        public void Initialize()
        {
            _server = new ServerHelper();
        }
        [TestMethod]
        public void TestGetData()
        {
            string data;
            data = _server.DownloadData();
            List<Series> series = JsonConvert.DeserializeObject<List<Series>>(data);
            Series s = series[0];
            Assert.IsTrue(series.Count > 0);
            Assert.AreEqual(-20, s.SeriesID);
        }
    }
}
```

6.2.5 Series

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SeriesManagementSystem.Domain;
using System;
using System.Collections.Generic;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class SeriesUnitTest
    {
        Series _series;
        List<Episode> _episodes;
        const int SeriesID = 10;
        const string SeriesName = "testSeries";
        const string SeriesDescription = "this is a test Series Description";
        const string ModifiedSeriesName = "modifiedSeries";
        const string ModifiedSeriesDescription = "this is a modified description";
        static readonly string[] EPISODE_NAMES = new string[] { "episode 0", "episode 1" };
        static readonly string[] EPISODE_DESCRIPTIONS = new string[] { "episode description 0", "episode
description 1" };

        [TestInitialize]
        public void Initialize()
        {
            _series = new Series(SeriesName, SeriesDescription);
            _episodes = new List<Episode>();
            _episodes.Add(new Episode(EPISODE_NAMES[0], EPISODE_DESCRIPTIONS[0]));
            _episodes.Add(new Episode(EPISODE_NAMES[1], EPISODE_DESCRIPTIONS[1]));
        }

        [TestMethod]
        public void TestName()
        {
            Assert.AreEqual(SeriesName, _series.Name);
        }

        [TestMethod]
        public void TestDescription()
        {
            Assert.AreEqual(SeriesDescription, _series.Description);
        }

        [TestMethod]
        public void TestSetName()
        {
            Assert.AreEqual(SeriesName, _series.Name);
            _series.Name = ModifiedSeriesName;
            Assert.AreEqual(ModifiedSeriesName, _series.Name);
        }

        [TestMethod]
        public void TestSetDescription()
        {
            Assert.AreEqual(SeriesDescription, _series.Description);
            _series.Description = ModifiedSeriesDescription;
            Assert.AreEqual(ModifiedSeriesDescription, _series.Description);
        }

        [TestMethod]
        public void TestAddEpisode()
        {
            Assert.AreEqual(0, GetEpisodes().Count);
            _series.AddEpisode(EPISODE_NAMES[0], EPISODE_DESCRIPTIONS[0]);
            Assert.AreEqual(1, GetEpisodes().Count);
        }

        [TestMethod]
        public void TestRecord()
        {
            string command = "Very well.";
            GetEpisodes().AddRange(_episodes);
            _series.Record(EPISODE_NAMES[0], command);
            Assert.AreEqual(1, _episodes[0].CommandList.Count);
            Assert.IsTrue(_episodes[0].IsRead);
            command = "";
            _series.Record(EPISODE_NAMES[1], command);
            Assert.AreEqual(0, _episodes[1].CommandList.Count);
            Assert.IsTrue(_episodes[1].IsRead);
        }

        private List<Episode> GetEpisodes()
        {
            return new PrivateObject(_series).GetFieldOrProperty("_episodes") as List<Episode>;
        }
    }
}
```

6.2.6 Episode

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SeriesManagementSystem.Domain;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class EpisodeUnitTest
    {
        Episode _episode;
        const string EPISODE_NAME = "episode name";
        const string EPISODE_DESCRIPTION = "epispode description";
        const string MODIFIED_NAME = "modified name";
        const string MODIFIED_DESCRIPTION = "modified description";

        [TestInitialize]
        public void Initialize()
        {
            _episode = new Episode(EPISODE_NAME, EPISODE_DESCRIPTION);
        }

        [TestMethod]
        public void TestProperties()
        {
            Assert.AreEqual(EPISODE_NAME, _episode.Name);
            Assert.AreEqual(EPISODE_DESCRIPTION, _episode.Description);

            _episode.Name = MODIFIED_NAME;
            _episode.Description = MODIFIED_DESCRIPTION;
            Assert.AreEqual(MODIFIED_NAME, _episode.Name);
            Assert.AreEqual(MODIFIED_DESCRIPTION, _episode.Description);
        }

        [TestMethod]
        public void TestRecord()
        {
            string command = "So suprise!";
            _episode.Record(command);
            Assert.AreEqual(1, _episode.CommandList.Count);
            Assert.IsTrue(_episode.IsRead);
            command = "";
            _episode.Record(command);
            Assert.AreEqual(1, _episode.CommandList.Count);
        }
    }
}
```

6.2.7 Command

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SeriesManagementSystem.Domain;

namespace SeriesManagementSystemUnitTest
{
    [TestClass]
    public class CommandUnitTest
    {
        private Command _command;
        private const string DEFAULT_CONTENT = "It's good.";

        [TestInitialize()]
        public void Initialize()
        {
            _command = new Command(DEFAULT_CONTENT);
        }

        [TestMethod]
        public void TestGetContent()
        {
            Assert.AreEqual(DEFAULT_CONTENT, _command.Content);
        }
    }
}
```

Measurement

101820302 施帛辰		**101820340 鄒令業**		**備註**
HW #1				
16/02/23 14:10~15:15	**65 min**	16/02/23 14:10~15:15	**65 min**	**Meeting**
		16/02/28 13:30~14:00	**30 min**	**Doc. Writing**
Total	**65 min**	**Total**	**95 min**	
HW #2				
16/03/10 10:10~12:10	**120 min**	16/03/10 10:10~12:10	**120 min**	**Meeting & Discussion**
16/03/15 14:15~17:05	**170 min**	16/03/15 14:15~17:05	**170 min**	**Meeting & Discussion**
16/03/17 10:10~11:30	**80 min**	16/03/17 10:10~11:30	**80 min**	**Meeting & Discussion**
Total	**370 min**	**Total**	**370 min**	
HW #3				
16/03/29 14:10~15:30	**80 min**	16/03/29 14:10~15:30	**80 min**	**Meeting**
Total	**80 min**	**Total**	**80 min**	
HW #4				
16/04/26 14:20~17:10	**170 min**	16/04/26 14:20~17:10	**170 min**	**Meeting**
		16/04/27 13:00~14:00	**60 min**	**Coding**
16/04/27 16:10~17:20	**70 min**	16/04/27 16:10~17:20	**70 min**	**Meeting**
Total	**240 min**	**Total**	**300 min**	
HW #5				
16/05/01 19:00~20:30	**90 min**	16/05/02 12:00~12:30	**30 min**	**Coding**
16/05/03 14:00~17:00	**180 min**	16/05/03 14:00~17:00	**180 min**	**Meeting**
		16/05/04	**30 min**	**Coding**

		11:00~11:30		
16/05/05 15:10~17:00	**110 min**	16/05/05 15:10~17:00	**110 min**	**Meeting**
16/05/05 19:00~19:30	**30 min**			**Coding**
Total	**410 min**	**Total**	**350 min**	
HW #6				
16/05/17 14:10~17:00	**170 min**	16/05/17 14:10~17:00	**170 min**	**Discussion**
Total	**170 min**	**Total**	**170 min**	
HW #7				
16/05/31 14:10~18:00	**230 min**	16/05/31 14:10~18:00	**230 min**	**Discussion**
16/06/02 15:00~17:30	**150 min**	16/06/02 15:00~17:30	**150 min**	**Discussion & Coding**
16/06/11 9:00~12:00	**180 min**	16/06/12 20:00~22:00	**120 min**	**Coding**
16/06/13 1:00~3:00	**120 min**			**Coding**
		16/06/12 22:00~22:30	**30 min**	**Edit Document**
16/06/13 14:00~16:00	**120 min**	16/06/13 14:00~16:00	**120 min**	**Discussion & Edit Doc.**
Total	**800 min**	**Total**	**650 min**	
All Efforts	**2135 min**	**All Efforts**	**2015 min**	