*Florida International University*

*School of Computing and Information Sciences*

CIS 4911 Senior Capstone Project

Test Case Management System

**Deliverable 4 – Final Deliverable**

December 2014

**Group Members**

Justin Phillips  
Karina Harfouche

**Mentors**Tariq King  
Dionny Santiago

**Instructors**Masoud Sadjadi  
Juan Caraballo

**Executive Summary**

This Document contains the entire summary of the projects activities. This includes the feasibility study and recommendations, the requirements elicitation and analysis, the system design process and artifacts generated, the implementation details and artifacts generated, and finally the verification stage. The project centered on the synchronization and storage of test plans across various 3rd party test management systems and the viewing and running of automated tests.

The entire process was performed in a disciplined, quantifiable, and iterative manner starting with the feasibility study and requirements elicitation in September of 2014, followed by user interface design and development in October 2014, and finally iterative cycles of implementation and verification in November of 2014.

Table of Contents

[**1.**  **Introduction** 5](#_Toc406080351)

[**1.1. Problem definition** 5](#_Toc406080352)

[**1.2. Scope of system** 6](#_Toc406080353)

[**1.3. Over all development methodology** 6](#_Toc406080354)

[**1.4. Definitions, acronyms, and abbreviations** 6](#_Toc406080355)

[**1.5. Overview of document** 6](#_Toc406080356)

[**2.**  **Feasibility Study** 7](#_Toc406080357)

[**2.1. Description of current system.** 7](#_Toc406080358)

[**2.2. Description of alternative solutions considered.** 8](#_Toc406080359)

[**2.2.2. Selection Criteria** 8](#_Toc406080360)

[**2.2.3. Analysis of Alternatives** 8](#_Toc406080361)

[**2.3. Recommendation** 9](#_Toc406080362)

[**3.**  **Project Plan** 9](#_Toc406080363)

[**3.1. Project Organization** 10](#_Toc406080364)

[3.1.1. Project Personnel 10](#_Toc406080365)

[**3.1.2.**  **Hardware and Software Resources** 10](#_Toc406080366)

[**3.2. Identification of Tasks, Milestones and Deliverables** 11](#_Toc406080367)

[**3.3. Cost of the Project** 12](#_Toc406080368)

[**4.**  **System Requirements** 14](#_Toc406080369)

[**4.1. Functional and Nonfunctional Requirements** 14](#_Toc406080370)

[**4.2. Requirements Analysis** 16](#_Toc406080371)

[**5. System Design** 16](#_Toc406080372)

[**5.1. Overview** 16](#_Toc406080373)

[**5.2. Subsystem Decomposition** 18](#_Toc406080374)

[**5.3. Hardware and Software Mapping** 18](#_Toc406080375)

[**5.4. Persistent Data Management** 20](#_Toc406080376)

[**5.5. Security/Privacy** 21](#_Toc406080377)

[**6. Detailed Design** 21](#_Toc406080378)

[**6.1. Overview** 21](#_Toc406080379)

[**6.2. Static mode** 22](#_Toc406080380)

[**6.3. Dynamic model** 22](#_Toc406080381)

[**6.4. Code Specification** 22](#_Toc406080382)

[**7.**  **System Validation** 22](#_Toc406080383)

[7.1. Subsystem Tests 23](#_Toc406080384)

[7.2. System Tests 23](#_Toc406080385)

[7.3. Evaluation of Tests – evaluate how successful the tests were. Use a tabular form. 23](#_Toc406080386)

[8. Glossary 23](#_Toc406080387)

[9. Appendix 23](#_Toc406080388)

[**9.1. Appendix A - Project schedule (Gantt chart or PERT Chart)** 23](#_Toc406080389)

[**9.2. Appendix B – All use cases with nonfunctional requirements.** 24](#_Toc406080390)

[**9.3. Appendix C – User Interface designs.** 35](#_Toc406080391)

[**9.4. Appendix D – Analysis models (static and dynamic)** 37](#_Toc406080392)

[**9.5. Appendix E – Design models (static and dynamic)** 41](#_Toc406080393)

[**3.3. Dynamic model** 46](#_Toc406080394)

[**9.6. Appendix F – Documented Class interfaces (code) and constraints.** 47](#_Toc406080395)

[**9.7. Appendix G – Documented code for test drivers and stubs.** 53](#_Toc406080396)

[**9.8. Appendix H – Diary of meeting and tasks for the entire semester.** 62](#_Toc406080397)

[9.9 Appendix I - Feasibility Matrix 68](#_Toc406080398)

[10. References 68](#_Toc406080399)

# **1. Introduction**

This document describes in detail the processes, artifacts, and code generated within this project. The document is covered in order of the work done starting with feasibility study, requirements elicitation, system design, implementation, and finally verification.

## **1.1. Problem definition**

Software testing is a core decision stage within a functional software’s development and operation life cycle. Testing determines the correctness, completeness, and overall compliance with a client’s requested features and requirements. Companies such as Ultimate Software depend heavily on efficient and practical management of software testing in order to confidently validate their software products with their customer requirements. While Ultimate Software’s testing team manages large sets of test cases using Microsoft Test Manager (MTM) in conjunction with Microsoft’s Team Foundation Server (TFS), several shortcomings have been found in MTM. Additionally, MTM cannot connect to 3rd party systems so Ultimate test engineers must use multiple user interfaces in order to maintain their test plans. The current software testers’ experience when creating, editing and managing the tasks related to testing in general could improve in order to significantly and directly enhance the time it takes to fully test software as well as the overall assertiveness and ease of spotting blocking issues that ultimately affect the customer’s satisfaction with a given software product.

The Test Case and Automation application will help overcome some of the difficulty with software testing and the reviewing of test results. A software tester that runs dozens of test cases a day relies heavily on endless amounts of logs scattered across different applications to accurately keep track of failures and its regular tendencies. Common causes of legitimate or false failures are often overlooked due to the large amount of details required to analyze test results and conclude its resolution with the correct form of action on a timely matter.

Testers require the ability to very quickly and easily create a new test project, link test cases to their associated code repository files, and notify the tester to changes in these files; in addition to viewing the results is a quick, easy to comprehend format.

## **1.2. Scope of system**

The proposed Test Management system’s requirements and development will be solely based and personalized on the needs and specifications of the Solution Testing Team at Ultimate Software. As the requirements and use case sections of this document specifies, the scope of the system includes the general tasks involved with the creation, edit, update and deletion of test case documents coupled with providing a mechanism for persistent storage of the information required in addition to interoperability TSF,MTM, and expandability for other management systems. The system shall be responsible for synchronizing its own model with 3rd party management systems such as Phoenix and maintaining overall integrity of the data wherein. The system shall also include metrics for test case failure rates, failure types, and tie these metrics into suites and code files. These metrics should be displayed in a graphical, easy to read format. The automation aspect of the system will be pursued only if permissible by the time constraints and resources for this project. In the case that time constraints impede the full development of the application’s ability to automate test cases, analysis and proper planning will still be made accordingly in order to develop an application that allows the addition of an automation component.

## **1.3. Over all development methodology**

The overall design methodology employed for this project is agile, where design, development, and verification happen in an iterative cycle. The initial development iterations consisted of user interface design and development. Once the user interface was determined to be a usable state a bottom up approach was taken for developing the rest of the system’s components starting with the adapter interfaces within the DataStore.

## **1.4. Definitions, acronyms, and abbreviations**

**COCOMO:** Constructive Cost Model

**MTM:** Microsoft Test Manager

**SCM:** Source Code Management

**SW**: Software

**TFS:** Team Foundation Server

**WA:** Work Activity

## **1.5. Overview of document**

This document provides a detailed description of the projects activities including the feasibility study, requirements elicitation, system design, implementation, verification, and final documentation.

# **2. Feasibility Study**

This chapter presents the feasibility analysis including the current system, its limitations and constraints, the purpose of the new system, and proposed alternatives. The purpose of the new system includes high level requirements for the proposed new system which were elicited from the client. The alternative solution was developed by analyzing the limitations and constraints of the current system in regard to the problem provided by the client. The chapter closes providing recommendations as to which proposed system would be the best investment for the client.

## **2.1. Description of current system.**

The current system, within the scope of this project, is centered around MTM utilized in conjunction with TFS, in addition to the Phoenix test automation system. These applications and services are used to create test suites, cases, and steps in addition to providing autonomous build, test, and deployment services; as well as reporting services. Also, MTM supports manual and exploratory testing via reporting and rich media capturing capabilities. The system provides for the ability to then query reports and tests.

The most major limitations of the current system are the usability governing creation or cloning of test suites, cases, the accessibility of test metrics data, and steps and the duplication of effort required in test planning. The usability limitations exist within the structural limitations of test cases, drill-down navigation, and the complexity/limited feature set of the user interface, and having multiple user interfaces. The client would like the ability to create two levels of test cases, one that is high level and one at an implementation details level; which the current system does not support. MTM provides a drill- down navigation of test suites and cases that allow for attachments to be added. However, viewing these attachments using the drill-down navigation creates for an arduous process. Creating similar test plans requires much duplication of effor, especially across multiple systems. Phoenix test result data, specifically in the area of test failure causes, is not readily accessible from the test planning and maintenance interfaces and not currently stored in a human readable format. Lastly, in many areas test case creation is too complex, with too busy of a user interface while also not providing necessary features such as copying of cases and steps.

Another limitation of the current system is that MTM cannot associate test suites, cases, or steps with the code repository (SVN, Git, CVS, etc…). Thus, there does not exist a method for which the tester can be notified about file changes associated with a given test element.

When it comes to automating and storing test case results, Phoenix and TeamCity softwares at Ultimate are very efficient; yet the display of this results are not as direct and efficient for a Software Tester as it could be. Since a software tester runs dozens of test cases a day, he or she relies heavily on endless amounts of logs scattered across different applications to accurately keep track of failures and its regular tendencies. Common causes of legitimate or false failures are often overlooked due to the large amount of details required on logs provided by the current system. In effect, to analyze test results and conclude its resolution with the correct form of action on a timely matter turns to be difficult.

In addition to viewing the test results in a quick and easy to comprehend format; testers require the ability to very quickly and easily create a new test project, link test cases to their associated code repository files, and notify the tester to changes in these files. Currently, the previous tasks may not be accomplished within one single user friendly application.

Finally, MTM does not provide a mechanism in which the results of automated tests may be displayed. This provides for a fragmented testing system which requires multiple applications to determine the state of the tests and displaying test associations.

## **2.2. Description of alternative solutions considered.**

**Web Application**

A web application would be developed which would communicate with TFS, Phoenix, and the code repository. This web application would provide a web accessible interface for creating test suites, cases, and steps as well as the other required features.

**Visual Studio Plug-in**

Visual Studio provides a plug-in API which would leverage the Microsoft platform, communicating with the TFS server. The plug-in would expose the TFS test suites, cases, and steps as well as other requested features within the Visual Studio UI.

**ClearQuest Synchronization**

A simple Visual Studio plug-in would be developed which would shadow copy test suites, cases, and steps between ClearQuest and TFS. This would allow for the mature ClearQuest test creation UI to be used in conjunction with TFS.

### **2.2.2. Selection Criteria**

Given that the primary requirement relies on usability, the operational feasibility was one of the most weighed portions of the feasibility analysis. Due to the time constraints on the project the technical feasibility is also very important, including the practicality of the overall solution; developer expertise with the solution implementation; and current licensed software available from the client. This time consideration coincides with measuring the scheduling feasibility. Finally, the economic feasibility was also measured using a cost-benefit analysis.

### **2.2.3. Analysis of Alternatives**

In the analysis of the operational feasibility, the current system was compared to the proposed systems’ capabilities in terms of usability and user acceptance. The Visual Studio plug-in provides an inflexible interface dependent upon the existing Visual Studio constructs, this limitation detracts from the usability of this proposed system. Additionally, many testers may not be familiar with Visual Studio which may cause adoption resistance. Conversely, the ClearQuest solution provides for a mature and well-understood user interface compared to creating a new interface for creating test elements. However, ClearQuest’s interface is dated and does not take advantage of many new usability techniques and narrow the extendibility of the user interface options. The web based solution provides for a highly tailored and flexible user experience, in addition to being very extendable. These benefits make the web-based solution the most usable and least resistant to user adoption in terms of operational feasibility.

The technical feasibility was limited when researching the Visual Studio plug-in and ClearQuest solutions due to a lack of developer expertise with developing Visual Studio plug-ins and lack of pre-existing ClearQuest experience in addition to licensing. Whereas the developers both have experience with the development of web applications, there is no licensing, and the extensibility of web applications is very broad.

Scheduling feasibility of the Visual Studio plug-in and ClearQuest applications would not require as much development as they would only require extending existing applications, rather than developing a new system from the ground up, as is the case of the web application. In contrast, there would be a greater period of time required for the developers to become familiarized with developing Visual Studio plug-ins. In addition, the Visual Studio plug-in and ClearQuest applications would require more training time for testers than the more usable web application.

Economic feasibility was determined by the sum of the development cost, future benefit, and licensing costs. Both the Visual Studio plug-in and ClearQuest application incur licensing cost, however the development costs are lower due to less development time being required. The future benefit is inflexible with these applications due to their tight technology coupling. While the web application requires more development cost, it does not have the direct licensing cost and provides the most future-proof solution being loosely coupled and very extensible.

Appendix C presents a feasibility matrix which utilizes the feasibility analysis to score each application based on the four mentioned areas of feasibility. This matrix clearly indicates the best investment for the client in regard to the three solutions proposed.

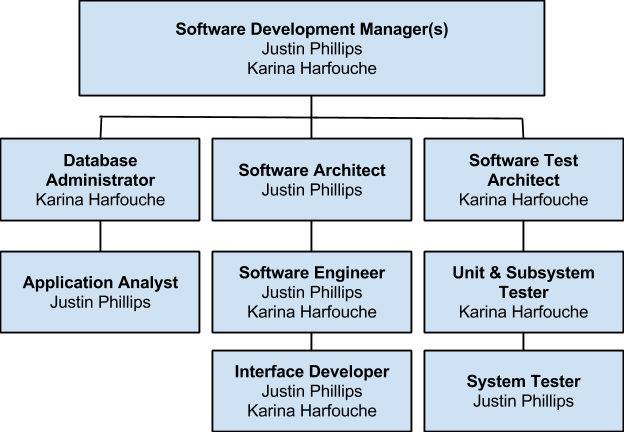
## **2.3. Recommendation**

Based on the feasibility analysis and feasibility matrix scoring, the project would be best suited with a web application solution. The web application provides the ability to create a much richer and simplified user experience for testers, is less tightly coupled to one technology, and is far more extensible than the alternative solutions. This provides for a more operational and economically feasible solution while also incorporating the client’s need for a highly usable system. The feasibility matrix in Appendix C presents the solution with the best investment benefit as being the web application, scoring 6 points more than the next solution. Moving forward, it is recommended that the web application solution be chosen as the path forward.

# **3. Project Plan**

The project organization section focuses on the visualization and analysis of the general software organization as well as the software support and service organization. A hierarchical chart is presented in order to illustrate the proposed structure for the Test Case Management System’s working personnel.

## **3.1. Project Organization**



### 3.1.1. Project Personnel

Justin Phillips - Cross functional and software engineer

Karina Harfouche - Cross functional and software engineer

### **3.1.2. Hardware and Software Resources**

The following tentative table specifies all the hardware and software resources needed throughout the planning, development, implementation and testing of the Test Case Management system:

|  |
| --- |
| **Tools** |
| Server with public IP or domain, minimum of 4GB of RAM and 100GB free of hard disk space. |
| Visual Studio 2013 Premium or better |
| GIT Client |
| Microsoft Office Home |
| Team Foundation Server 2013 |
| Microsoft Windows Server or Windows 7 |
| StarUML |
| Microsoft Test Manager |
| Microsoft SQL Server Standard 2012 |

## **3.2. Identification of Tasks, Milestones and Deliverables**

The following table identifies the major tasks, milestones and deliverables along with each respective duration dates visually defined on the Gantt Chart presented in Appendix A:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task Name** | **Start Date** | **End Date** | **Duration (days)** |
| **Requirement Elicitation** | **09/04/14** | **09/15/14** | **8** |
| **Meet with client** | **09/04/14** | **09/04/14** | **1** |
| **Determine Requirements** | **09/04/14** | **09/15/14** | **8** |
| **System Analysis** | **09/05/14** | **09/15/14** | **7** |
| **Determine System Feasibility** | **09/05/14** | **09/10/14** | **4** |
| **Compare with Alternatives** | **09/10/14** | **09/15/14** | **4** |
| **Present Deliverable** | **09/15/14** | **09/15/14** | **1** |
| **Software Project Planning** | **09/26/14** | **09/30/14** | **3** |
| **Scheduling** | **09/26/14** | **09/26/14** | **1** |
| **Resource Allocation** | **09/27/14** | **09/27/14** | **1** |
| **Risk Plan** | **09/27/14** | **09/30/14** | **3** |
| **Software Project Organization** | **09/29/14** | **10/06/14** | **6** |
| **Personnel Structurization** | **09/29/14** | **09/29/14** | **1** |
| **Obtaining Tools** | **10/01/14** | **10/01/14** | **1** |
| **Define Metrics** | **10/02/14** | **10/03/14** | **2** |
| **Acceptance of Project Organization** | **10/04/14** | **10/06/14** | **2** |
| **System Design** | **10/06/14** | **10/10/14** | **5** |
| **Create Use Cases** | **10/06/14** | **10/06/14** | **1** |
| **Subsystem Decomposition** | **10/07/14** | **10/07/14** | **1** |
| **Determine Security and Privacy** | **10/08/14** | **10/08/14** | **1** |
| **Create DB Schema** | **10/09/14** | **10/09/14** | **1** |
| **Hardware & Software Mapping** | **10/10/14** | **10/10/14** | **1** |
| **Object Design** | **10/12/14** | **10/24/14** | **11** |
| **Create Class Diagram** | **10/12/14** | **10/12/14** | **1** |
| **Create OCL statements** | **10/13/14** | **10/13/14** | **1** |
| **Create Sequence Diagram** | **10/24/14** | **10/24/14** | **1** |
| **System Implementation** | **10/10/14** | **11/13/14** | **25** |
| **Create/Configure Database** | **10/10/14** | **10/14/14** | **3** |
| **Implement User Interface** | **10/14/14** | **10/24/14** | **9** |
| **Implement System Logic** | **10/25/14** | **11/13/14** | **15** |
| **Present Deliverable** | **10/28/14** | **10/29/14** | **2** |
| **Testing** | **11/13/14** | **12/02/14** | **14** |
| **Create & Implement Test Cases** | **11/13/14** | **11/17/14** | **3** |
| **Run test suites** | **11/13/14** | **11/19/14** | **5** |
| **Evaluate & Fix Defects** | **11/19/14** | **12/02/14** | **10** |
| **Present Final Deliverable** | **12/15/14** | **12/16/14** | **2** |
| **Present Deliverable to Client** | **12/15/14** | **12/15/14** | **1** |
| **System Release** | **12/16/14** | **12/16/14** | **1** |

## **3.3. Cost of the Project**

The following cost matrices represent an estimate of the hours, cost per person and cost of tools added into a total cost for the completion of the Test Case and Automation System.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Task Name** | **Work** | **Cost Per Person** | **Total Cost** |
| 0 | Planning | 20 hrs | $600.00 | $1,200.00 |
| 1 | Requirement Elicitation | 5 hrs | $150.00 | $300.00 |
| 2 | Meet with the client | 4 hrs | $120.00 | $240.00 |
| 3 | Determine Functional Requirements | 4 hrs | $140.00 | $280.00 |
| 4 | Determine Non-functional Requirements | 4 hrs | $140.00 | $280.00 |
| 5 | Create use cases | 24 hrs | $720.00 | $1,440.00 |
| 6 | System Analysis | 24 hrs | $720.00 | $1,440.00 |
| 7 | Determine system feasibility | 16 hrs | $480.00 | $960.00 |
| 8 | Compare to current system | 8 hrs | $240.00 | $480.00 |
| 9 | Present Deliverable | 1 hrs | $30.00 | $60.00 |
| 10 | Present deliverable to client | 1 hrs | $30.00 | $60.00 |
| 11 | System Design | 40 hrs | $1,200.00 | $2,400.00 |
| 12 | Object Design | 40 hrs | $1,200.00 | $2,400.00 |
| 13 | System Implementation | 301 hrs | $3,750.00 | $7,500.00 |
| 14 | Present Second Deliverable | 4 hrs | $120.00 | $240.00 |
| 15 | Present second deliverable to client | 4 hrs | $120.00 | $240.00 |
| 16 | Unit Testing | 40 hrs | $1,200.00 | $2,400.00 |
| 17 | Integration Testing | 80 hrs | $2,400.00 | $2,800.00 |
| 18 | Fix defects | 16 hrs | $450.00 | $900.00 |
| 19 | System Testing | 80 hrs | $2,400.00 | $4,800.00 |
| 20 | Fix defects | 16 hrs | $450.00 | $900.00 |
| 21 | Present Final Deliverable | 4 hrs | $120.00 | $240.00 |
| 22 | Present final deliverable to client | 4 hrs | $120.00 | $240.00 |
|  | **Total** | 620 hrs | $14,050.00 | **$31,560.00** |

Below are the tools that will be needed in order to successfully complete the project:

|  |  |  |
| --- | --- | --- |
| **Tools** | **Number of Units** | **Budget** |
| Server  Public IP  4GB Ram  100GB HD free space | 1 | $2,199.99 |
| GIT Client | 2 | $0.00 |
| Microsoft Office Home | 2 | $139.99 |
| Team Foundation Server 2013 | 1 | $499.00 |
| Visual Studio 2013 Premium or better | 2 | $2,569.00 |
| StarUML | 2 | $0.00 |
| Microsoft Test Manager | 2 | $0.00 |
| Microsoft Windows Server 2012 Dedicated Hosting (3 months) | 1 | $600.00 |
| Microsoft SQL Server Standard 2012 | 1 | $2,499.99 |
| **TOTAL** |  | **$11,417.94** |

# **4. System Requirements**

The new system shall provide the client with the ability to create test plans; that is the creation of test suites, cases, and steps, efficiently. The system shall interface with TFS and be configurable for both TFS communication and code repository linking, in addition to publishing test plans to a standalone database. Furthermore, the system shall allow test elements to be associated with a code repository. The system shall also support adding attachments to test elements. The system shall support a human readable format of the test result data using a user friendly system of charts. This entire system shall be secured behind a single sign-on login which is linked to domain accounts. Each feature within the proposed system shall provide a easy to use, simple interface to the system actors.

## **4.1. Functional and Nonfunctional Requirements**

The implementation of the Test Management System will include fifteen main functionalities that the system shall perform:

1. The system shall allow for testers within the configured Windows domain to have single sign-on access using the account the tester is signed into a Windows machine with.
2. The system shall provide a mechanism to configured the test management backend system.
3. The system shall provide an interface for creation of test suites, cases, and steps with fluid, intuitive authoring experience.
4. The system shall provide testers with the ability to add attachments and associate them with test suites, cases, and steps. The attachments should then be viewable with ease and with as few clicks as possible.
5. The system shall provide a mechanism to associate test suites, cases, and steps with a code repository or automation.
6. The system shall provide an interface to configure the interaction with a code repository.
7. The system shall display test suites, cases, and steps in a clean explorer or tree view throughout the user navigation within the system.
8. The system shall provide a means to display data categorically, organically, and in a logical view.
9. The system shall provide a simple search interface for searching the test suites, cases, and steps.
10. The system shall provide a logout mechanism so that the current user may be logged out and a user other than the user logged into the current Windows machine may login to the system.
11. The system shall be secured behind the login mechanism. Guests will only be able to visit the login feature.
12. The system shall provide an access table with a list of approved users from within the domain user group, this access table shall be configurable by pre-defined users on an individual basis.
13. The system shall provide functionality for extending the test element model to other 3rd party management systems.
14. The system shall provide a set of metrics which track test case failures, failure types, and source code attached to failures.
15. The system shall provide a visual representation of the failure metrics.

## **4.2. Requirements Analysis**

This chapter covers the analysis of the system requirements and presents the findings in UML diagrams. The UML diagrams referenced in this chapter are provided in Appendix B through Appendix D.

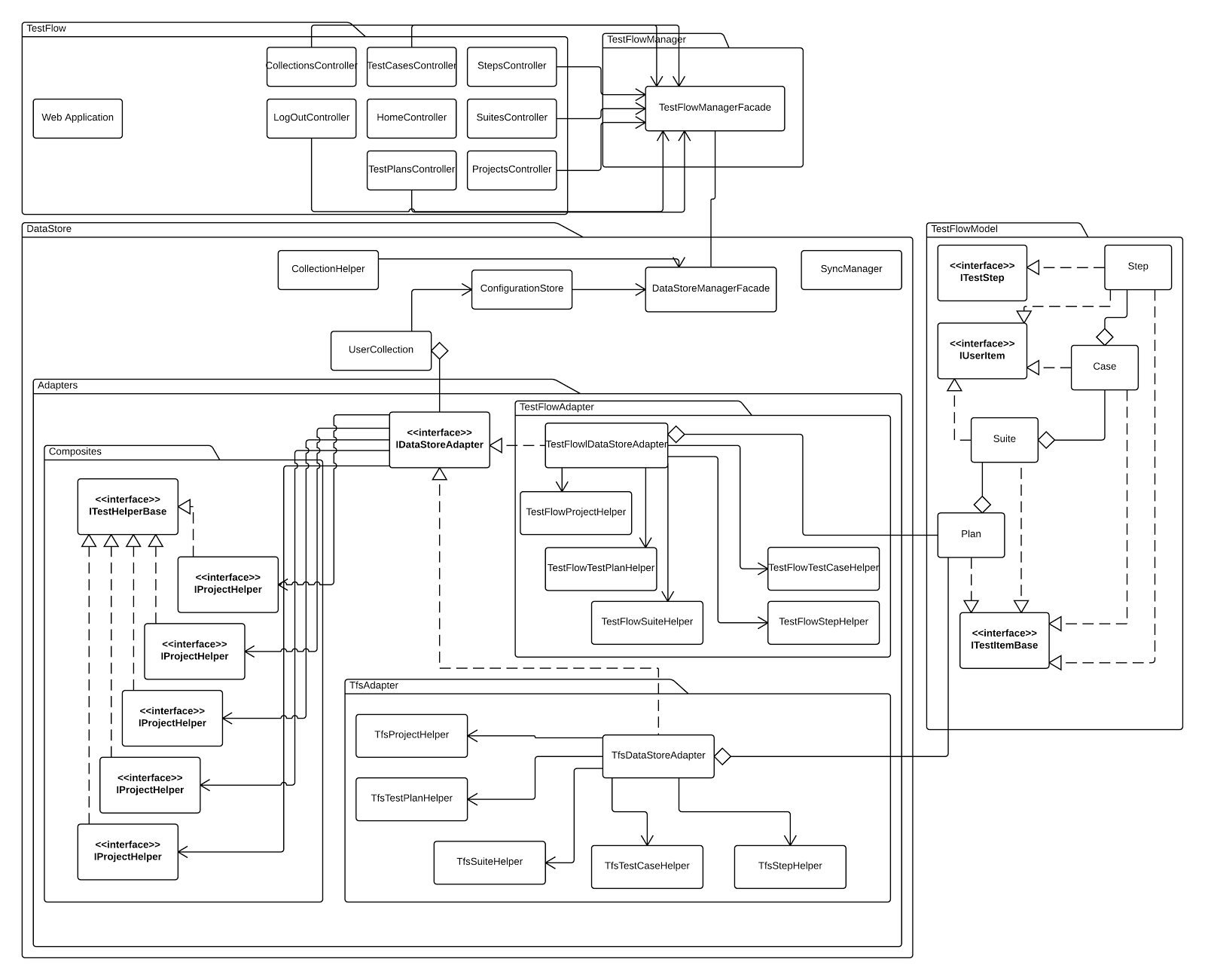
# **5. System Design**

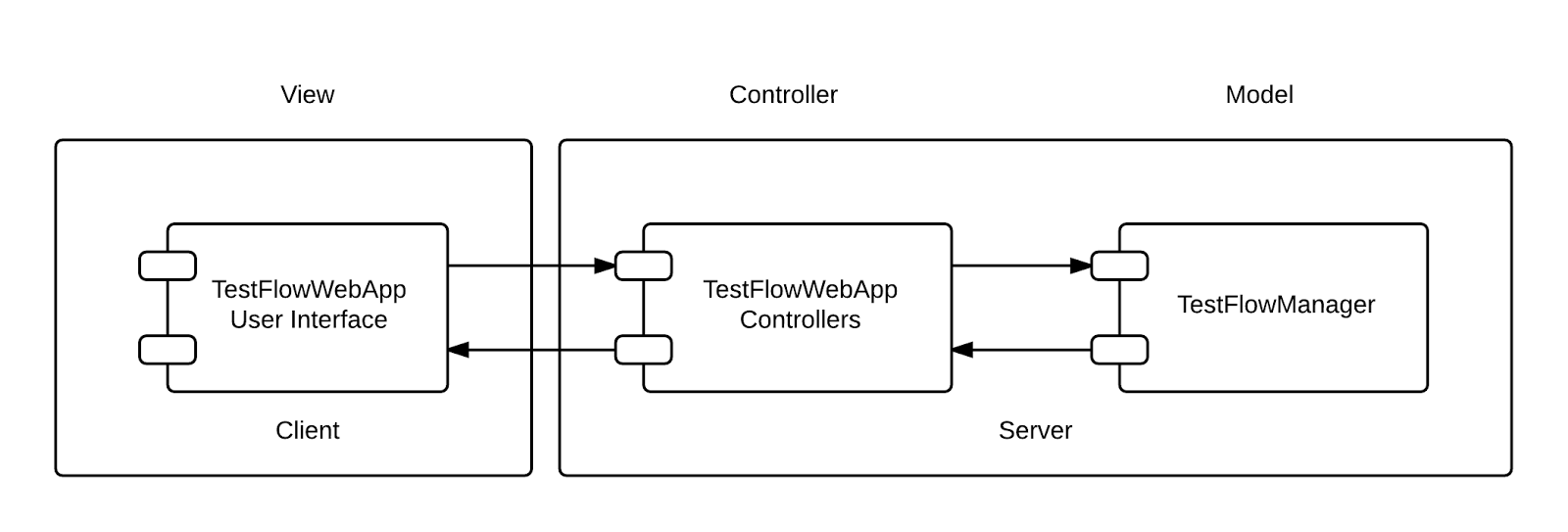
This chapter covers the overall system design and decomposition in great detail starting with a overview of the entire system. Subsystem decomposition describes each major subsystem and hardware and software mapping provides a picture of the systems implementation. These sections are followed by persistent data management, security, and finally detailed design where details specific to each object in the system are presented.

## **5.1. Overview**

The system’s overall architectural pattern is composed of a client / server model where a backend system provides connectivity to data and operations on the data while a client provides

access to the data to an end user. Below the package diagram shows the structure of the system’s design, starting with the test flow web application which hosts a varity of controllers and view followed by the point of access to the libries developed for operating on the test plan data. This system is exposed by TestFlowManager, which is a facade patterned object for managing the test flow system. The DataStore represents a series of adapters which provide connectivity to databases and 3rd party test management systems. This is where the test plans are provided persistant storage. Finally there is the TestFlowModel which realizes the testplan model data in code.



**Package Diagram of system, readable version in UML directory.** 

**Architectural Component Diagram**

The above component diagram depicts the system’s two architectures. The first being MVC where the HTML user interface of the web application servers as the view while the TestFlowWebApp’s controllers serve to decouple the model from the user interface. The model is provided by the TestFlowManager. The second architecture employed is client / server, where the user interface is located on a client browser and the rest of the system is provided on the server.

## **5.2. Subsystem Decomposition**

There are four major components of the system, the DataStore, TestFlowManager, TestFlowWebApp, and Metrics. The DataStore and TestFlowManager are libraries which form the connectivity and logic behind managing the data both on the system’s database and 3rd party management systems.

The DataStore is the central component to satisfying the projects requirements regarding the distribution of test between the system and other test management systems. The DataStore is composed of adapters which facilitate communication via a common interface with test management systems, these adapters are accessed through the DataStore facade pattern based manager object.

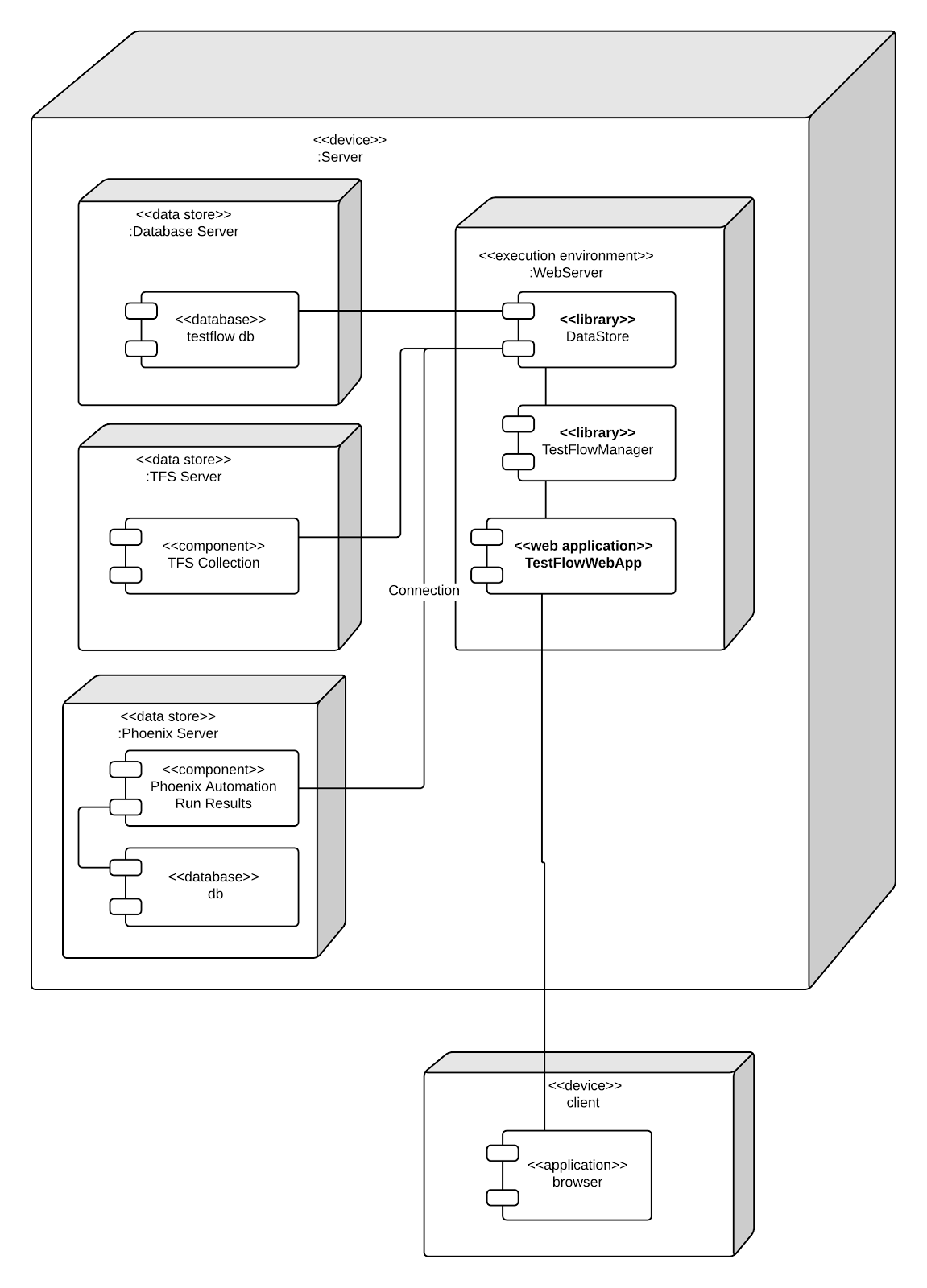
The TestFlowManager component is a facade pattern which allows for the operations exposed by the various libraries to be utilized in one object. The component controls all access to the DataStore and any future components.

The TestFlowWebApp component is the point of access to the system, it is MVC patterned website. It utilizes services which allow for access to the TestFlowManager.

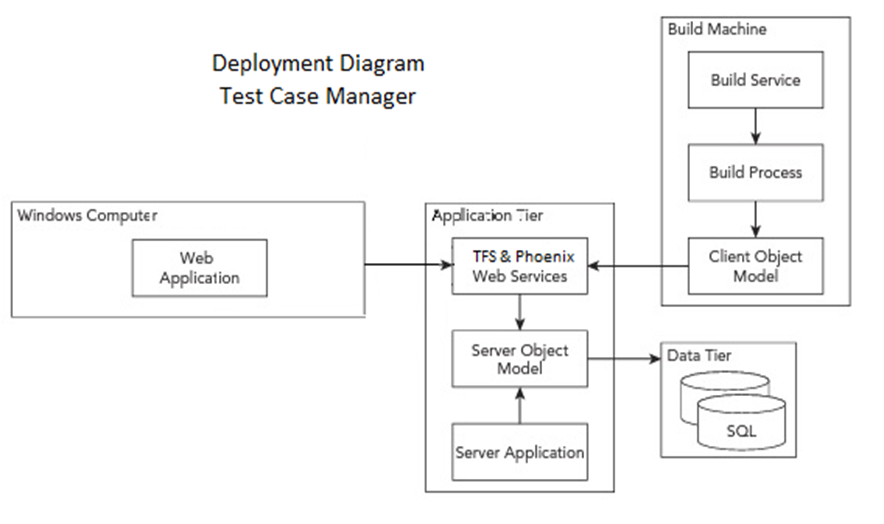
The Metrics component accesses the mock Phoenix server in order to run automation utilizing the data on the Phoenix database. The results are then transferred via web services to the TestFlowManager metrics subsystem.

## **5.3. Hardware and Software Mapping**

A Microsoft Windows server houses a web server as an execution environment for the system’s components which connect and store data on two data stores, a database system and a Team Foundation Server. A client connects to the server execution environment via a web browser. Finally, the system requires a Phoenix automation service to connect with.

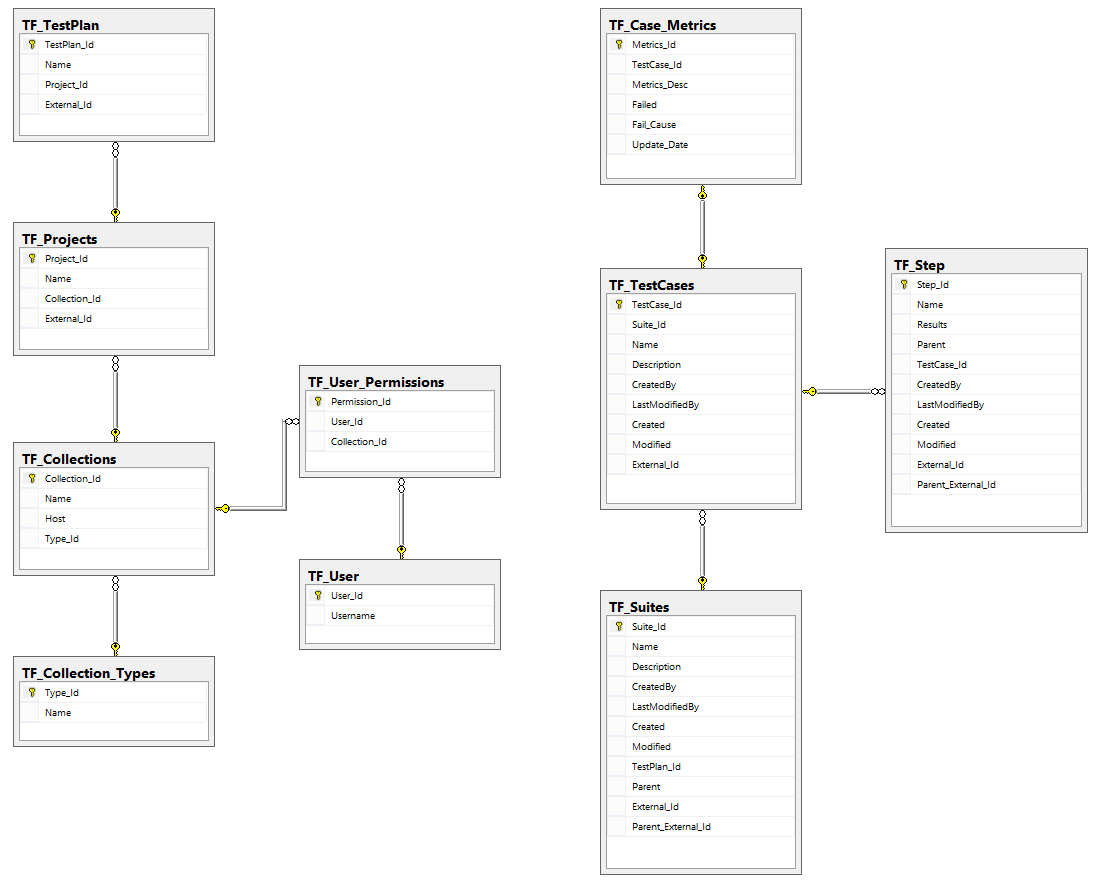


In the deployment diagram featured above, the entire system resides on a central server, although the TFS server, Phoenix server, and DB server could be hosted elsewhere. The web server components must be hosted on a single web server.



## **5.4. Persistent Data Management**

The persistent data is managed via a database server and stored in a database. This data includes user permission information, external storage mapping using IDs, and test plan data.



**Figure 1.2 ERD of database**

## **5.5. Security/Privacy**

Users are authenticated via Windows user identity, anonymous authentication will not be allowed. The Windows identity will be stored in the database and any projects in TFS which the user has permission to access are mapped to the user via the TF\_User\_Permissions table.

# **6. Detailed Design**

This chapter presents the Test Case Manager’s object design. First, an overview of the entire system is depicted followed by the details of the object design process. A minimal class diagram shows the complete system which will provide the appropriate context to introduce object interactions represented on the sequence diagrams. The detailed class description and each class’ purpose and functions have also been included in order to effectively portray and explain the overall object design.

# **6.1. Overview**

**DataStore**

The DataStore component is the core of the system, it provides interoperability via a suite of adapters made of up composite interfaces to external management systems and the test flow database. The adapter is an interface which provides common functionality and usage across different test management platforms. There is also a synchronization manager which ensures the test plan stays synchronized across platforms.

**TestFlowManager**

The TestFlowManager is a simple facade component which provides accessibility to the other test flow libraries as to create a single point of access to testflow.

TestFlowModel

The TestFlowModel component realizes the data as objects in code from the database, these objects are used to create, view, and edit the test elements.

**TestFlowWebApp**

The WebApp is the entry point to using testflow, it provides the user interface and operation endpoints to users. The WebApp is two MVC architectures layered on one another, the first provides most of the features on the server side, with it’s view providing a client side MVC application.

## **6.2. Static mode**

The static models stored in Appendix D describe each component in detail. In each class diagram all classes from the current code revision are present along with every method, signature, and attribute of each class.

## **6.3. Dynamic model**

There is one state machine diagram and one activity flow diagram in Appendix D, the state machine diagram shows the various states possible with the key binding on the user interface. The activity flow diagram is of more interest where is shows the process os syncronization between external test managers and test flow.

## **6.4. Code Specification**

The code stored in appendix C represents the interfaces from each of the components. The interfaces within DataStore represent the IDataAdaper’s potential realization. This adapter is formed from a series of helper interfaces that, as composites, realize the IDataAdapter’s functionality in manipulating test plans. The interfaces within the TestFlowModel reduce code redundancy and increase the consistency of the elements.

# **7. System Validation**

This section covers the processes used for system validation, each adapter in the system was unit tested and system wide test were done on each component and the system as whole.

## 7.1. Subsystem Tests

The code for the sub system tests is automated via Visual studio and was run after each change. These test ensure that all of the data manipulation features do not corrupt the data, cross the data, or do any other invalid things with the data.

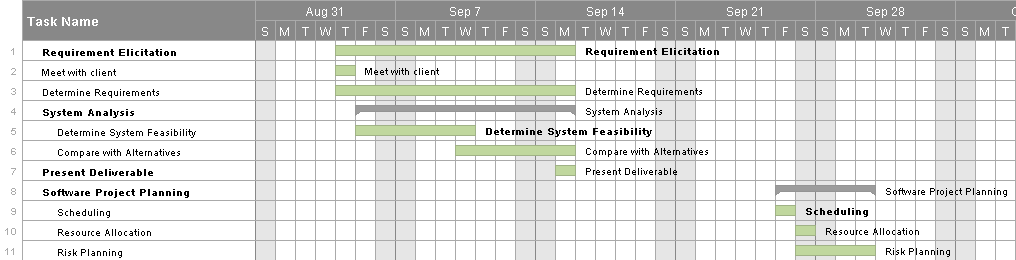
## 7.2. System Tests

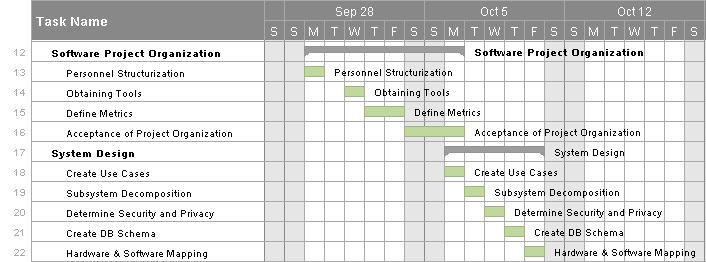
## 7.3. Evaluation of Tests – evaluate how successful the tests were. Use a tabular form.

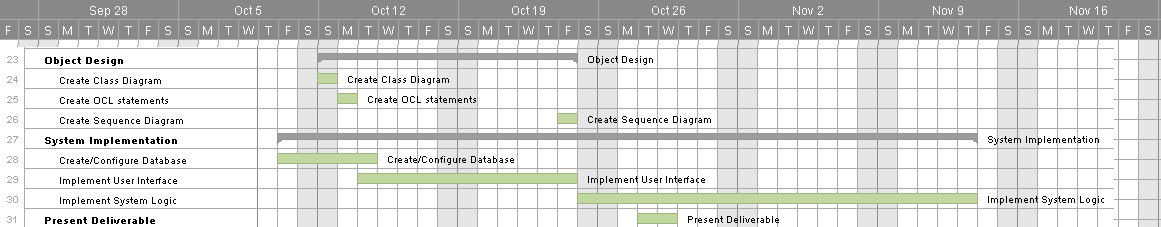
# 8. Glossary

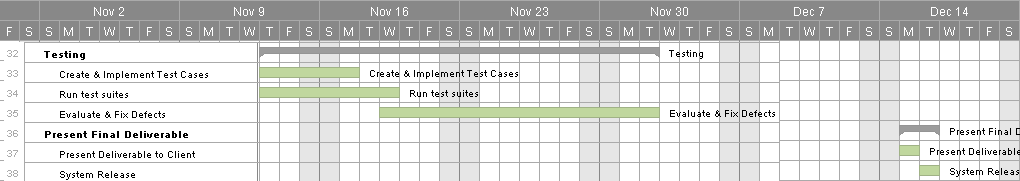
# 9. Appendix

## **9.1. Appendix A - Project schedule (Gantt chart or PERT Chart)**









## **9.2. Appendix B – All use cases with nonfunctional requirements.**

**USE CASE ID:** TS001 – Test Case & Automation Manager – Log In

**Level:** High Level

**Details:**

**Actors** - Tester, Administrator, Guest

**Preconditions**

1. Actor is logged in a Windows machine providing a Windows account to the Test Manager
2. Actor has accessed the web application’s login page

**Description**

1. The use case begins when the actor enters and submits login username and password. (Trigger)
2. The system responds by validating the username and password.
3. The use case ends when the account’s main page is succesfully loaded and displayed.

**Post-Conditions**

1. The Actor gets access to the Test Case & Automation Management page.

**Exceptions**

System is down.

**Decision Support:**

Frequency – The most frequent type of actor is the tester who will log into the system daily from Monday to Friday.

Critically – High, allows all actors to log into the system.

Risk – Medium

**Constraints:**

* Single Sign On is required, the user must be able to sign in with a windows account only.
* Username has to start with a letter character.
* Username characters may contain lowercase & uppercase letters, numbers and underscores.

**Usability:**

* No previous training time required

**Reliability:**

* Mean time to failure – 1% failures for every twenty four hours of operation is acceptable.
* Availability – Down time for login back-up 10 minutes in a twenty four hour period.

**Performance:**

* On average, the acotr should be able to login to the system in less than 30 seconds.

**Supportability:**

* System should deliver clear request for the second check point.

**Modification History:**

Owner: TC Senior Project  
Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS002 – Test Case & Automation Manager – Create Test Suite

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager
2. Actor has accessed the Test Case page

**Description**

1. The use case begins when the tester clicks on the plus sign from the suites left panel. (Trigger)
2. The user then inputs a Suite Case title.
3. The use case ends when the tester hits enter to add the suite with the provided information

**Post-Conditions**

1. The Actor gets access to the new suite on the hierarchical menu list.
2. The Actor may now add test cases to this suite.

**Exceptions**

A test suite with the provided name has already been created

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – High, allows tester to add suites.

Risk – Medium

**Constraints:**

* Test Suite file name should be created with acceptable ASCII characters.
* Tester creating test suite should have privilege granted.
* Test Suite name should be unique.

**Usability:**

* No previous training time required

**Reliability:**

* Mean time to failure – 1% failures for every twenty four hours is acceptable.
* Availability – Always.

**Performance:**

* On average, the system should be able to create a new Suite in less than 20 seconds after tester’s request.

**Supportability:**

* System should deliver clear request.

**Related Use Cases:**

**Modification History:**

Owner: TC Senior Project  
Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS003 – Test Case & Automation Manager – Create Test Case

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager
2. Actor has accessed the Test Case page
3. Actor has chosen the Test Suite where he wants to create a new test case

**Description**

1. The use case begins when the tester clicks on add sign beside a Test Suite on the main panel to add a test case to this suite. (Trigger)
2. The system responds by loading an editable tab to name a test case.
3. The tester fills the tab with a test case title
4. The use case ends when the tester hits enter to add the case with the provided information

**Post-Conditions**

1. The new test case will be added on the hierarchical menu list under its correspondent test suite.
2. The Actor may now access and edit this test case.

**Exceptions**

A test case with the provided name has already been created under this test suite

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – High, allows tester to add suites.

Risk – Medium

**Constraints:**

* Test Case name should be created with acceptable ASCII characters.
* Tester creating test case should have privilege granted.
* Test Case name should be unique.

**Usability:**

* No previous training time required

**Reliability:**

* Mean time to failure – 1% failures for every twenty four hours is acceptable.
* Availability – Always.

**Performance:**

* On average, the system should be able to create a new case in less than 20 seconds after tester’s request.

**Supportability:**

* System should deliver clear request.

**Modification History:**

Owner: TC Senior Project  
Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS004 – Test Case & Automation Manager – Create Test Steps

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester has created a test case

3. Tester has chosen the test case to be added steps

**Description**

1. The use case begins when the tester clicks on pull down test case

2. The system responds by loading the corresponding test case’s steps document

3. The user clicks on th plus sign button and types new step

4. The use case ends when the tester hits enter and changes have been saved on the test case

**Post-Conditions**

1. All actors are able to see the newly modified steps

2. A previous version of the test case steps has been backed up

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – High, allows tester to add steps.

Risk – Medium

**Constraints:**

* Tester creating steps should have privileges granted.

**Usability:**

* No previous training time required

**Reliability:**

* Mean time to failure – 1% failures for every twenty four hours is acceptable.
* Availability – Always.

**Performance:**

* On average, the system should be able save modified steps in less than 20 seconds after tester’s request.

**Supportability:**

* System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS005 – Test Case & Automation Manager –Duplicate Test Steps

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester has created a test case

3. Tester has chosen the test case to be added steps

**Description**

1. The use case begins when the tester clicks on edit test case
2. The system responds by loading the corresponding test case’s steps document
3. The user clicks on “Add Existing Steps” option
4. The tester enters on search a test case or opens the test case menu
5. The tester selects the test case where the desired steps exists
6. The test case end when the tester selects the steps and clicks on “Add to current test case”

**Post-Conditions**

1. All actors are able to see the newly modified steps duplicated on the test case chosen

2. A previous version of the test case steps has been backed up

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – High, allows tester to add steps.

Risk – Medium

**Constraints:**

* Tester duplicating test steps should have privileges granted.
* Tester should be able to duplicate steps from existing use cases in the current suite

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able save modified steps in less than 20 seconds after tester’s request.

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS006 – Test Case & Automation Manager –Link Test Suite to a Code Repository

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite

**Description**

1. The use case begins when the tester clicks on Link Test Suite

2. The system responds by loading a list of existing code repositories

3. The user selects a code repository

4. The use case ends when the tester clicks add repository

**Post-Conditions**

1. Changes have been saved on the test suite and the linked repository can be seen under the “Test Suite’s Repository’s List”

2. This repository can be unlinked now from this test suite

3. Tester may now click on this repository and be directly transferred to the code if available for viewing.

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium, allows tester to link repositories.

Risk – Medium

**Constraints:**

* Tester should have privileges to access and link code repositories
* Multiple repositories may be linked to a Test Suite

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able save modified links in less than 20 seconds after tester’s request.

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS008 – Test Case & Automation Manager –Add and Link Attachments

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite & a Test Case

**Description**

1. The use case begins when tester clicks on “Attachments” at the Details section of the current test case

2. The system responds by loading an upload window

3. The user selects one or multiple files to be attached

4. The use case ends when the tester clicks OK.

**Post-Conditions**

1. Changes have been saved on the test case and the attachment has now been added to this test case

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium, allows tester to add attachments.

Risk – Medium

**Constraints:**

* Tester should have privileges to add and link attachments
* Tester should be able to add and link any attachments from the local pc

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able save attachments, including compressed videos and folders.

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS009 – Test Case & Automation Manager –View and download Attachments

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite & a Test Case

**Description**

1. The use case begins when the tester clicks on “Attachments”

2. The system responds by loading the available attachments for that test case

3. The user selects the attachments to be viewed

4. The use case ends when attachment is loaded and the tester clicks on “download”

**Post-Conditions**

1. Attachment has been downloaded to local computer

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium, allows tester to download/view attachments.

Risk – Medium

**Constraints:**

* Tester should have privileges to view & download Attachments

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be download attachments, including compressed videos and folders.

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014



**USE CASE ID:** TS010 – Test Case & Automation Manager –Associate Test to Code Repository

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite

**Description**

1. The use case begins when the tester clicks on “Associate Suite”

2. The system responds by acquiring from the server all the test repository files available

3. The user selects the repository to be associated

4. The use case ends when tester clicks on “OK”

**Post-Conditions**

1. The code repository chosen is now available as an associated code to the specific test Suite

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium

Risk – Medium

**Constraints:**

* Tester should have privileges to associate Test Suite

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able to associate multiple repositories to a single Test Suite

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014

**USE CASE ID:** TS011 – Test Case & Automation Manager –Associate Test to Automation

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite

**Description**

1. The use case begins when the tester clicks on “Associate Suite”

2. The system responds by acquiring from the server all the test repository files available

3. The user selects the repository to be associated

4. The use case ends when tester clicks on “OK”

**Post-Conditions**

1. The code repository chosen is now available as an associated code to the specific test Suite

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium

Risk – Medium

**Constraints:**

* Tester should have privileges to associate Test Suite

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able to associate multiple repositories to a single Test Suite

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014

**USE CASE ID:** TS012 – Test Case & Automation Manager – View Code Repository

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite

**Description**

1. The use case begins when the tester clicks on “View Associations”

2. The system responds by acquiring from all available links to repositories and loads options.

3. The user selects the repository for viewing.

4. The use case ends when tester clicks on “View Selection”

**Post-Conditions**

1. The code repository chosen is now visible for the user.

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – High

Risk – Medium

**Constraints:**

* Tester should have privileges to view repository

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· On average, the system should be able to load multiple repositories to select viewing

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014

**USE CASE ID:** TS013 – Test Case & Automation Manager – Search

**Level:** High Level

**Details:**

**Actors** - Tester

**Preconditions**

1. Tester is logged in the test manager

2. Tester Opened the Test Suites Menu & Selected a Suite

**Description**

1. The use case begins when the tester click on the search tab

2. The tester types a title or keyword to search for a test case

3. The use case ends when tester clicks on “Search”

**Post-Conditions**

1. The system returns all search hits contained in the Test Suite

**Exceptions**

N/A

**Decision Support:**

Frequency – Daily from Monday to Friday.

Critically – Medium

Risk – Medium

**Constraints:**

* Search is not case sensitive
* Should search matching test cases and suites

**Usability:**

· No previous training time required

**Reliability:**

· Mean time to failure – 1% failures for every twenty four hours is acceptable.

· Availability – Always.

**Performance:**

· The system should be able to return all test cases matching the search

**Supportability:**

· System should deliver clear request.

**Modification History:**

Owner: TC Senior Project

Initiation Date: 9/16/2014

Date last modified: 9/16/2014

**USE CASE ID:** TS014 – Test Case & Automation Manager – Log Out

**Level:** High Level

**Details:**

**Actors** - Tester, Administrator, Guest

**Preconditions**

1. Actor is logged in the Test Manager

**Description**

1. The use case begins and ends when the actor clicks on the logout button

**Post-Conditions**

1. The Actor gets access to the Test Case & Automation Management page.

**Exceptions**

System is down.

**Decision Support:**

Frequency – The most frequent type of actor is the teste who will log into the system daily from Monday to Friday.

Critically – High, allows all actors to log out the system.

Risk – Medium

**Constraints:**

* System should automatically save the current state for the next login

**Usability:**

* No previous training time required

**Reliability:**

* Mean time to failure – 1% failures for every twenty four hours of operation is acceptable.
* Availability – Down time for login back-up 10 minutes in a twenty four hour period.

**Performance:**

* On average, the actor should be able to logut to the system in less than 20 seconds.

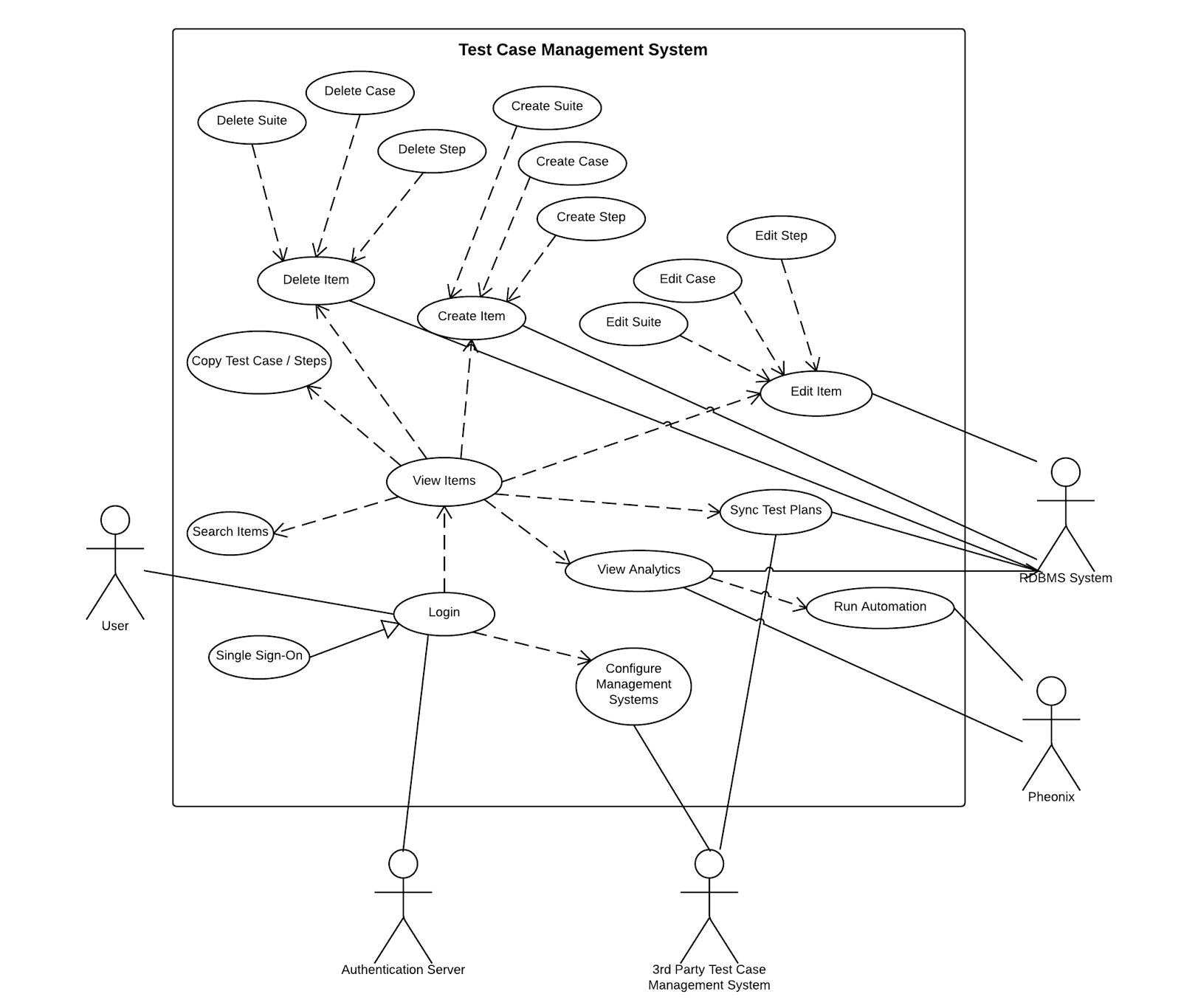
**Supportability:**

* System should deliver clear request for the second check point.

**Modification History:**

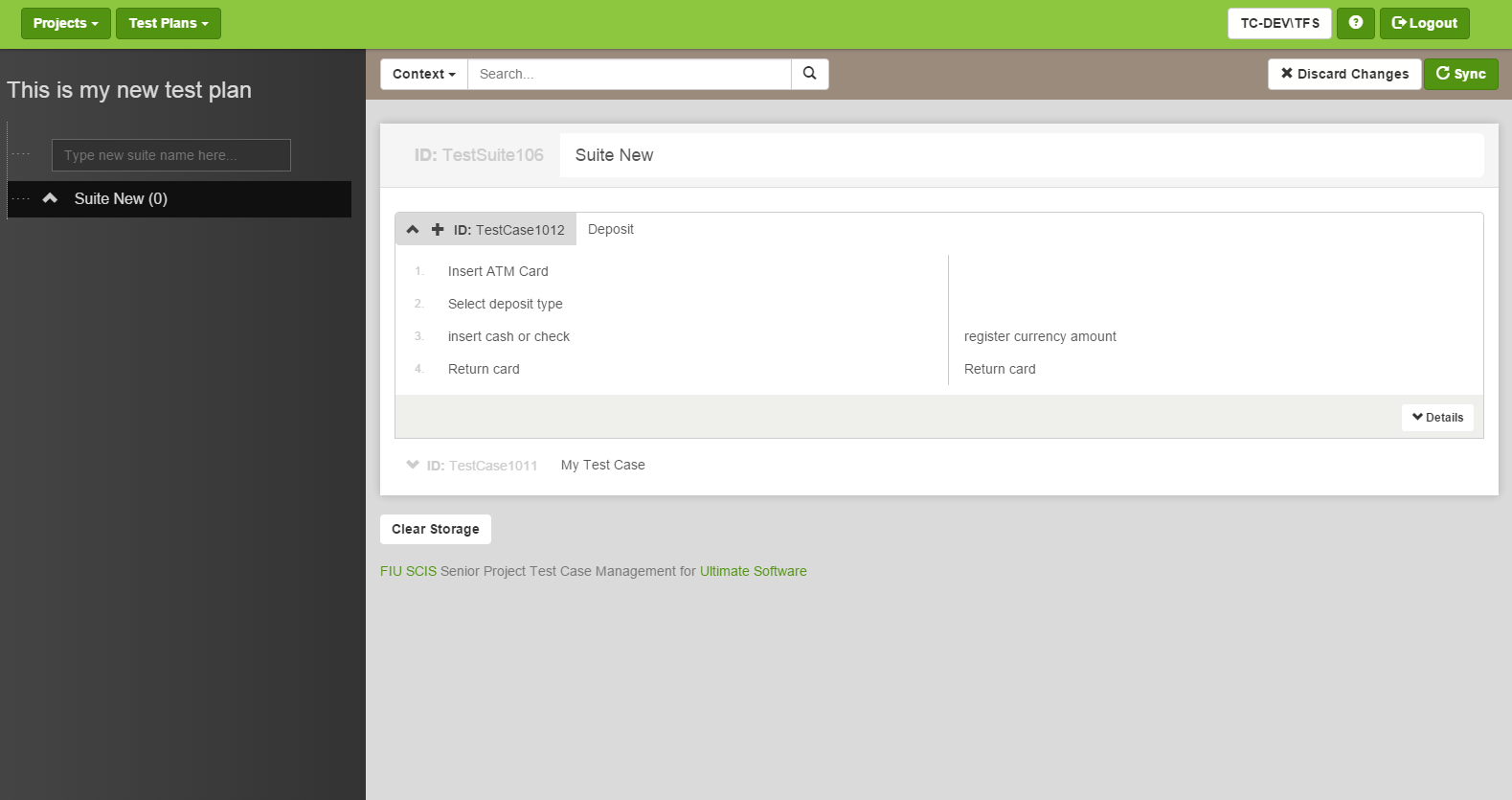
Owner: TC Senior Project  
Initiation Date: 9/16/2014

Date last modified: 9/16/2014

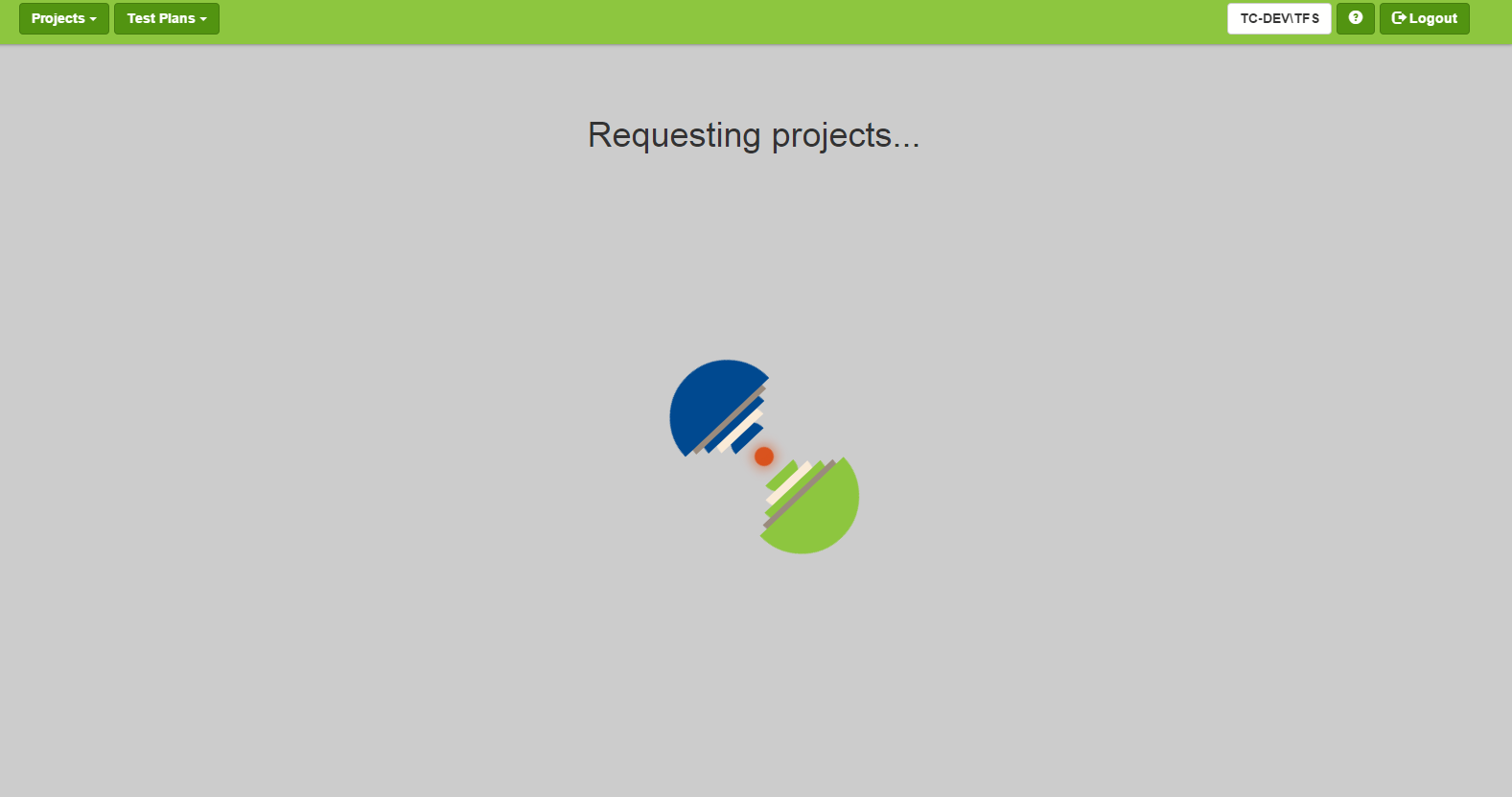


**Use Case Diagram of implemented use cases.**

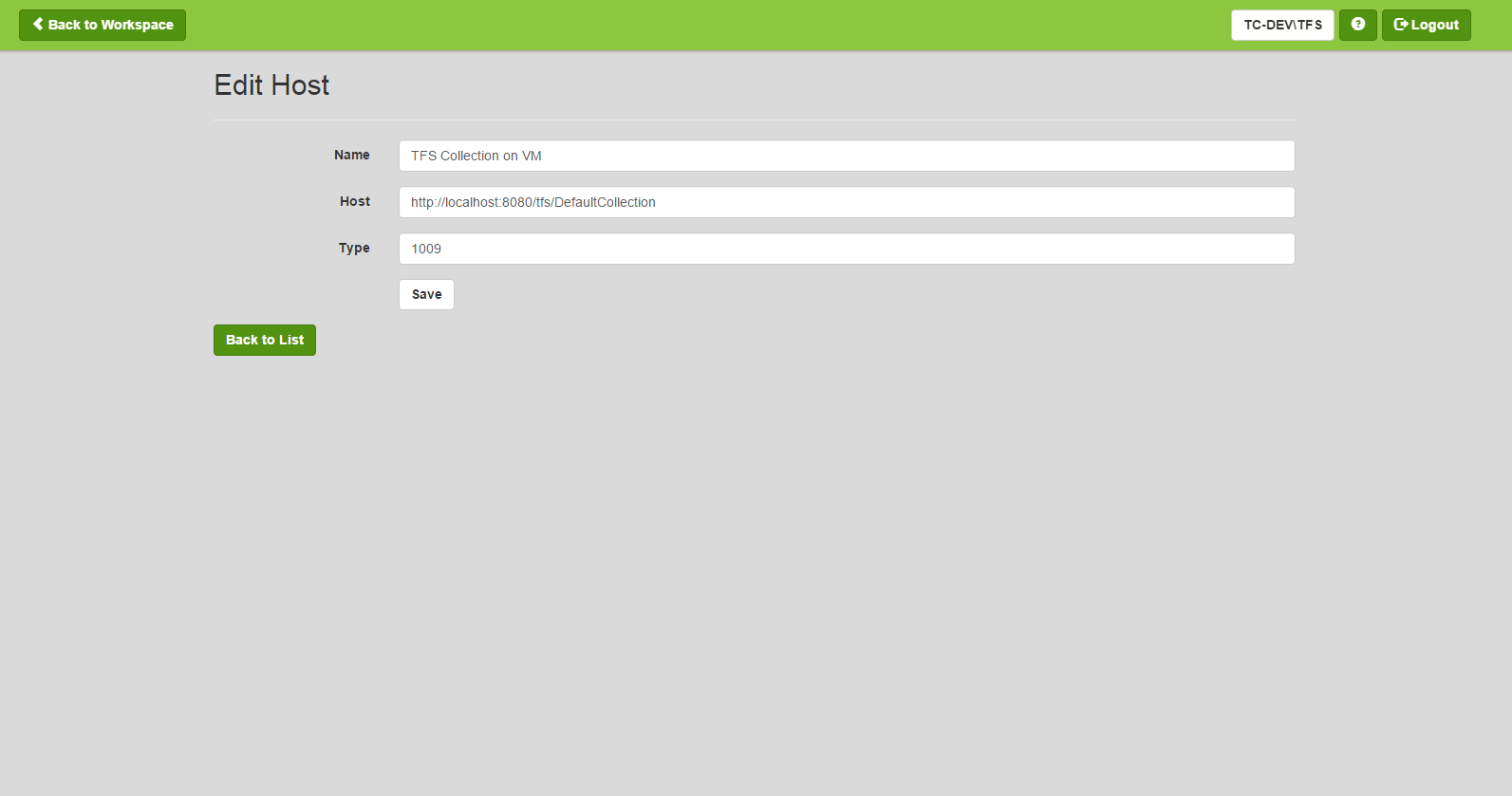
## **9.3. Appendix C – User Interface designs.**



*The Workspace of a small test plan*



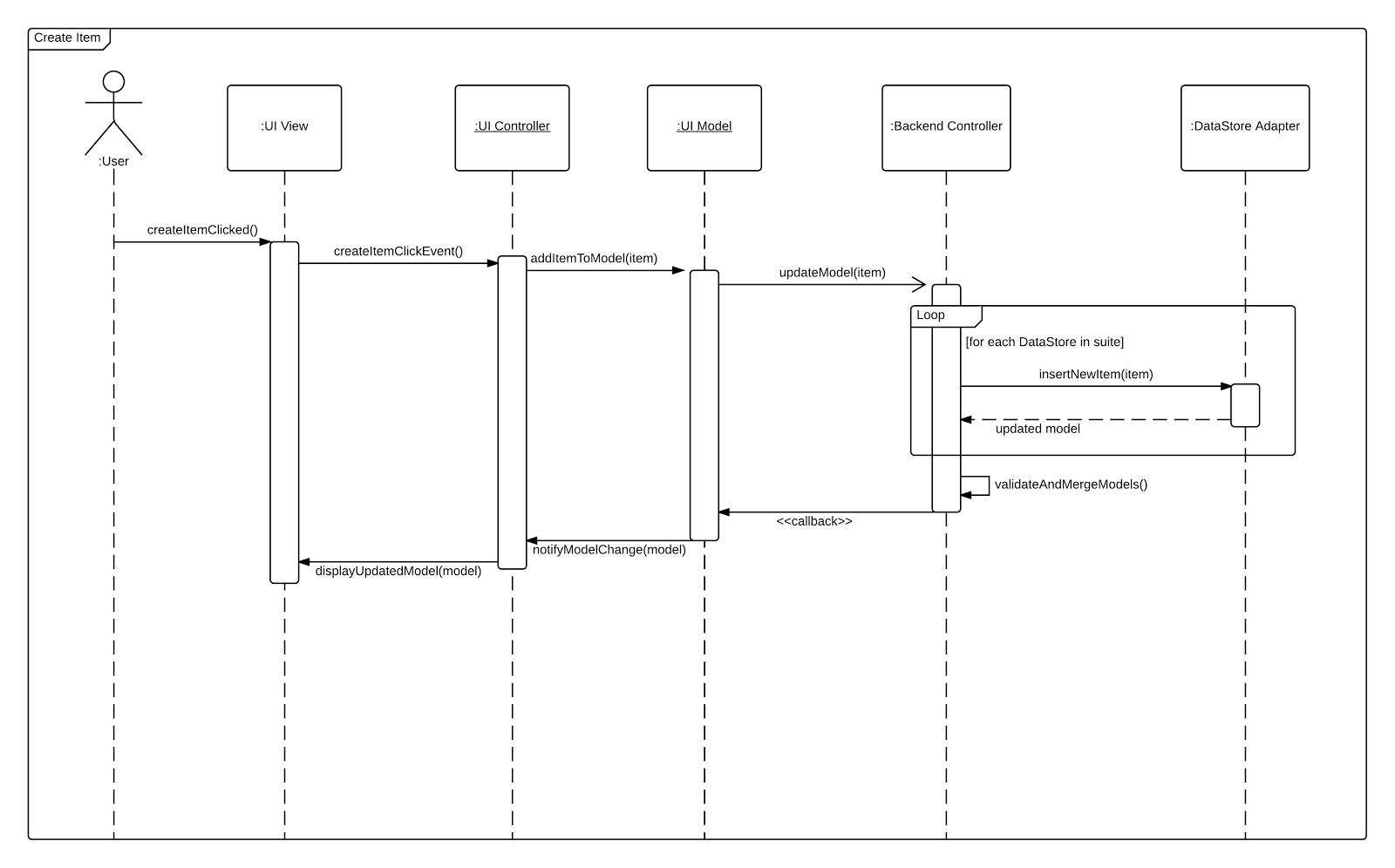
*Requesting Projects*



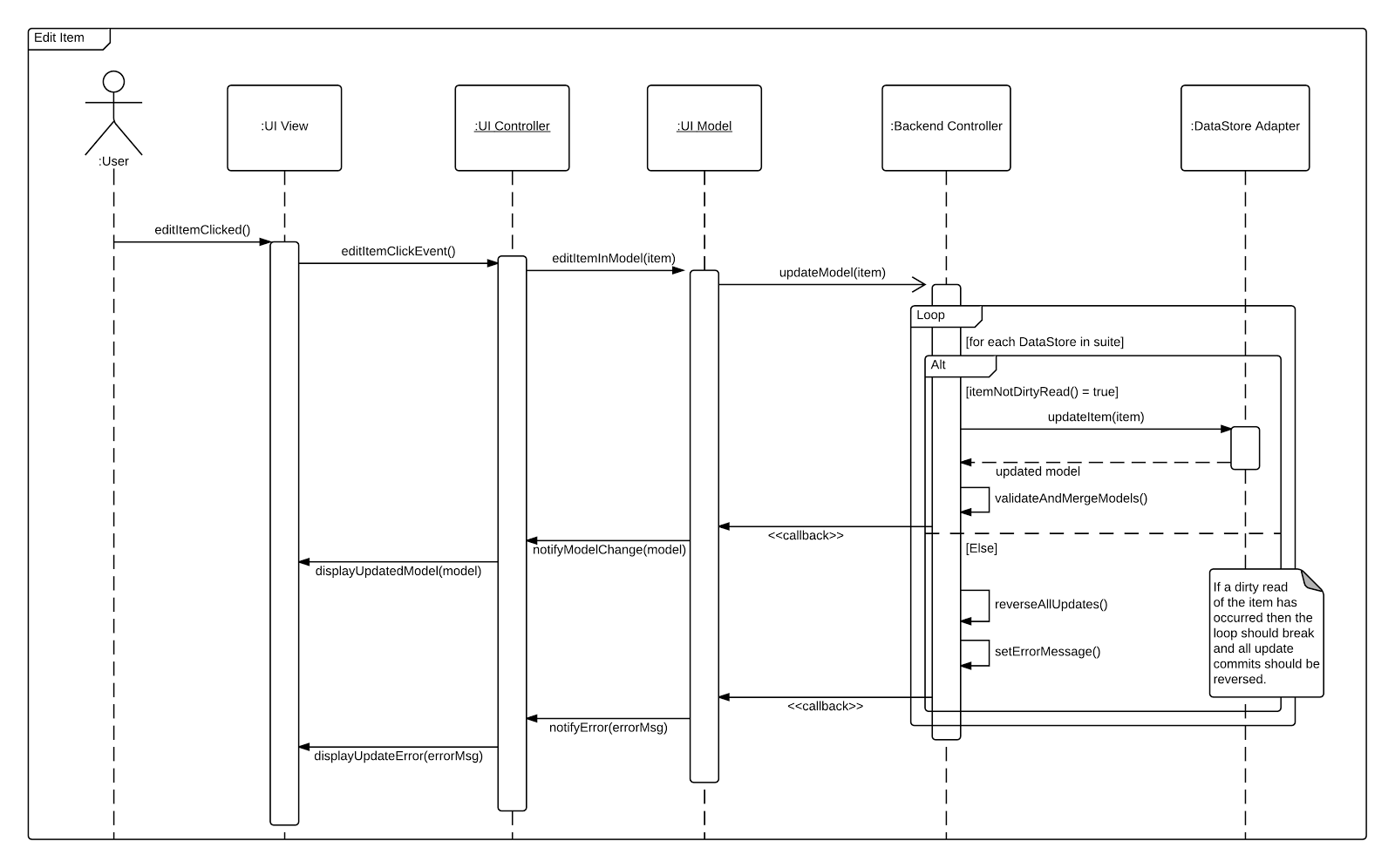
*Editing a TFS collection*

## **9.4. Appendix D – Analysis models (static and dynamic)**

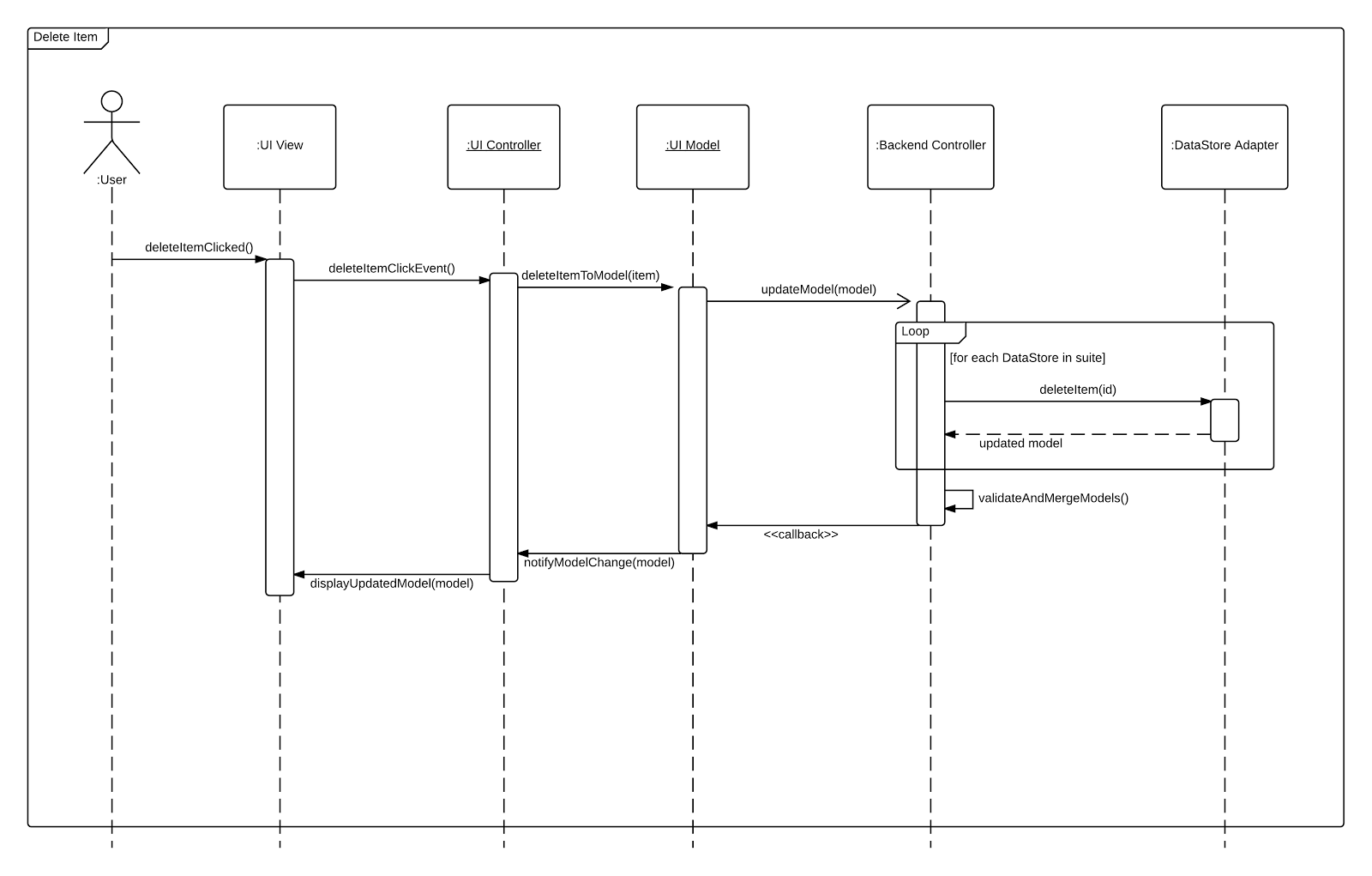
Create Item Use Case Sequence Diagram



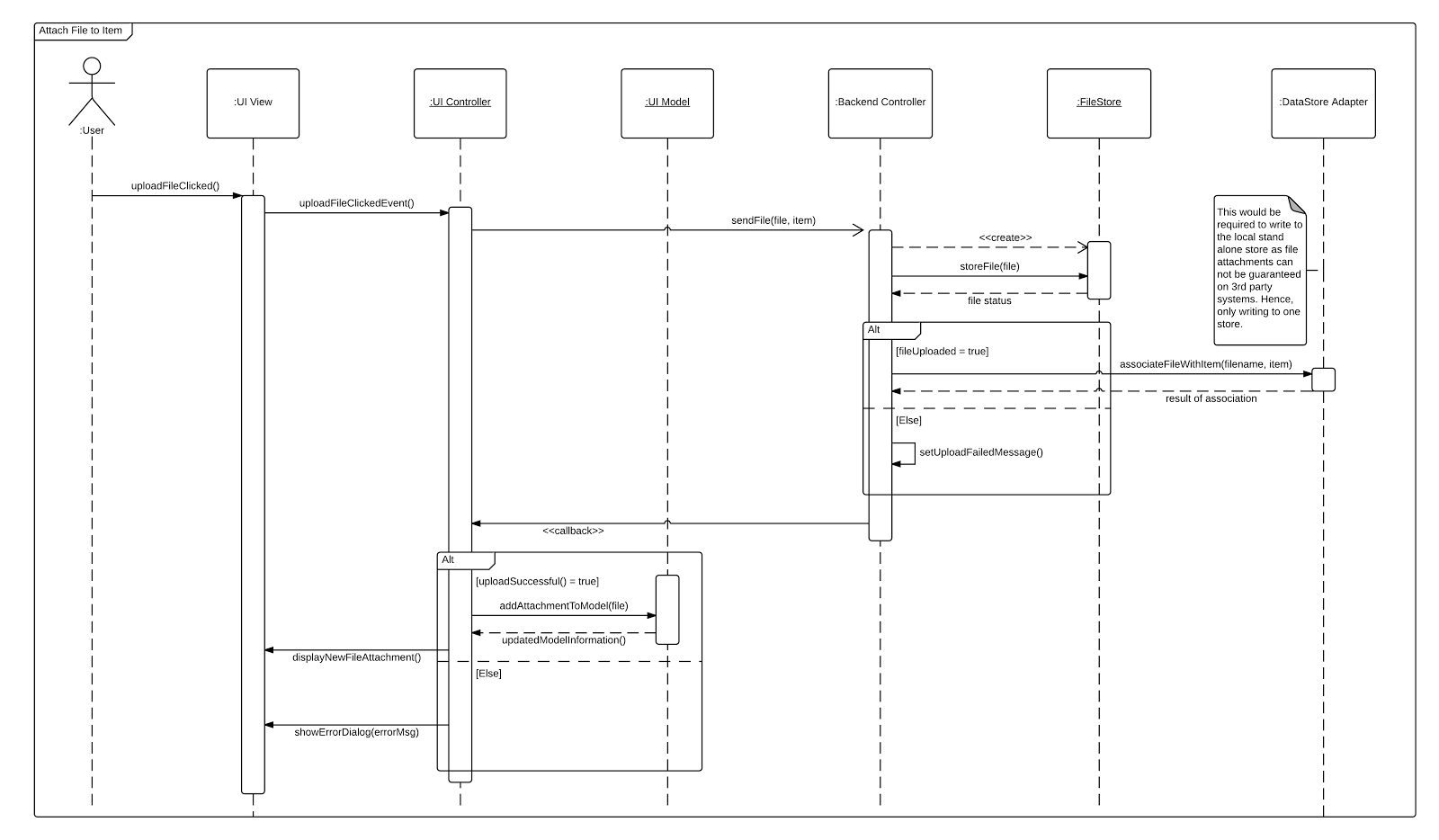
Edit Item Use Case Sequence Diagram



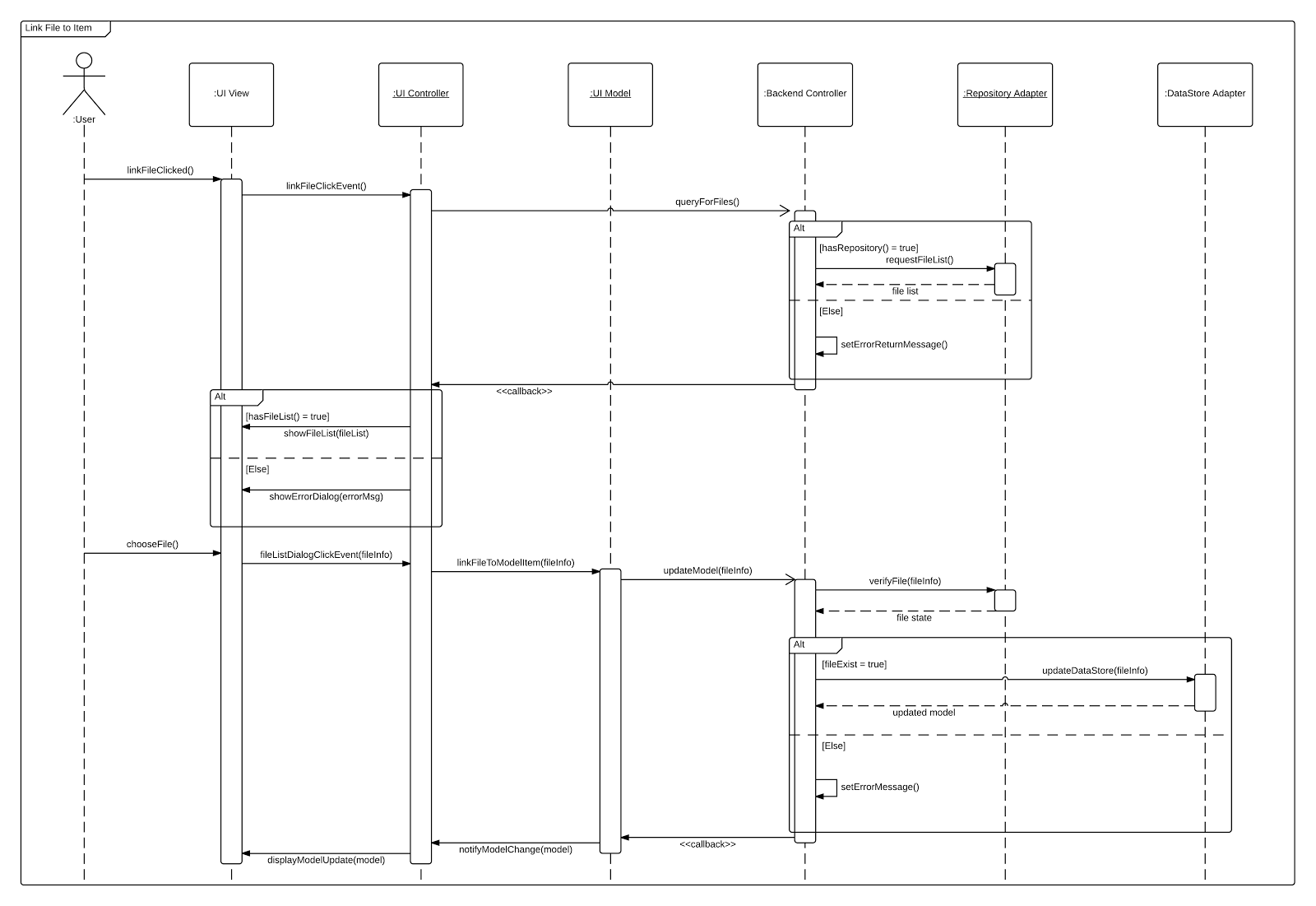
Delete Item Use Case Sequence Diagram



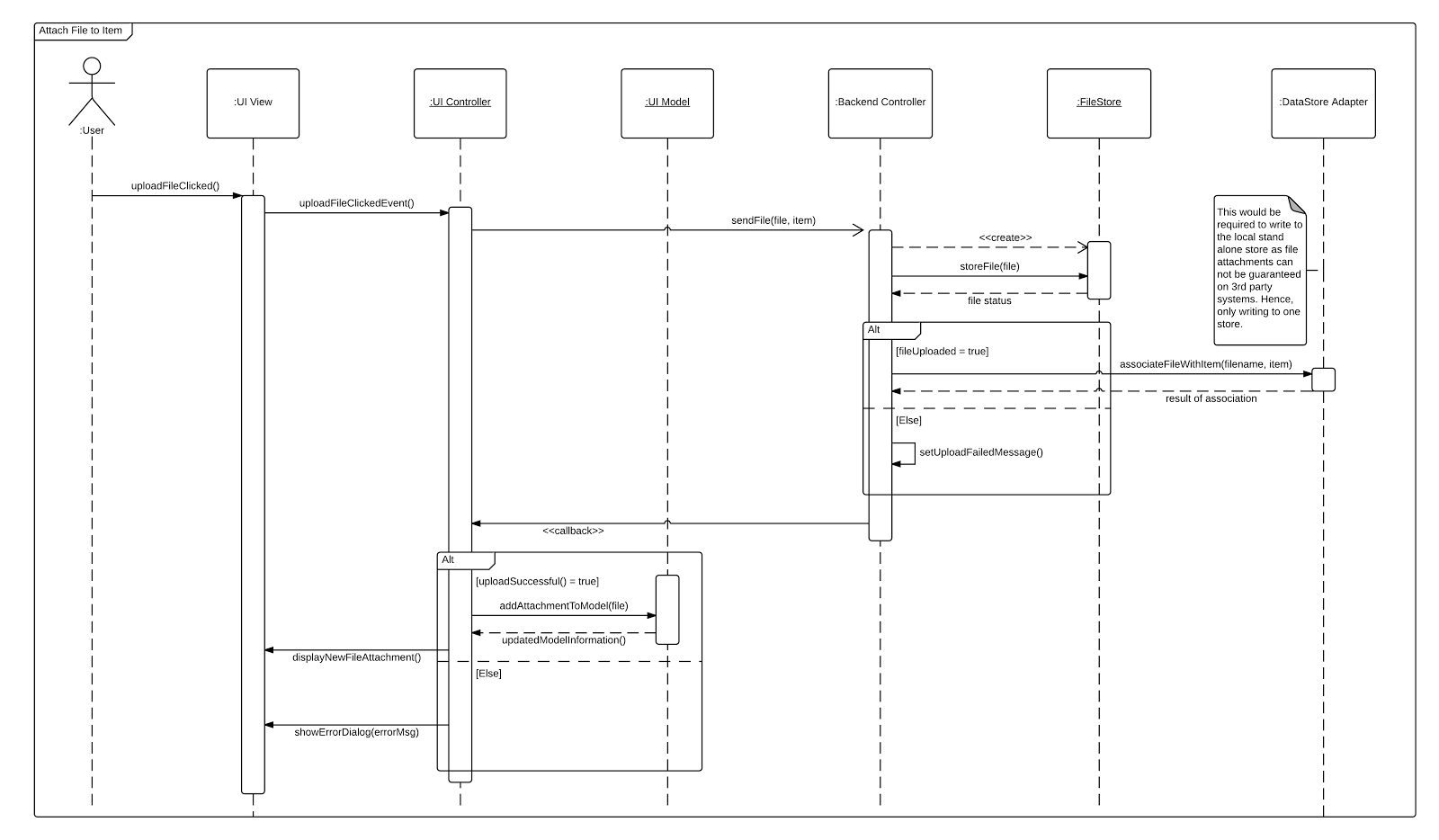
Copy Item Use Case Sequence Diagram

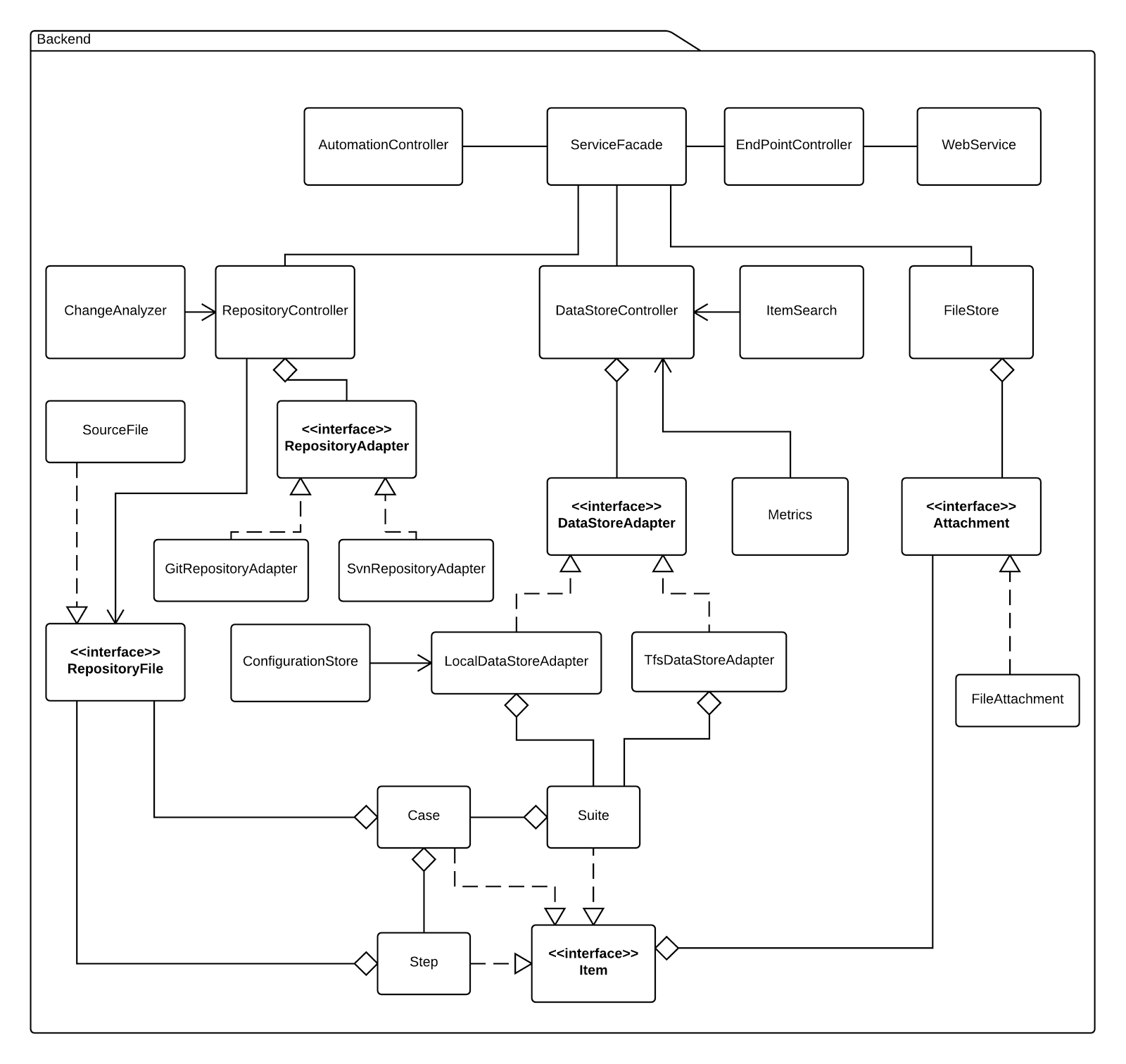


Link File to Item Use Case Sequence Diagram



Attach File to Item Use Case Sequence Diagram

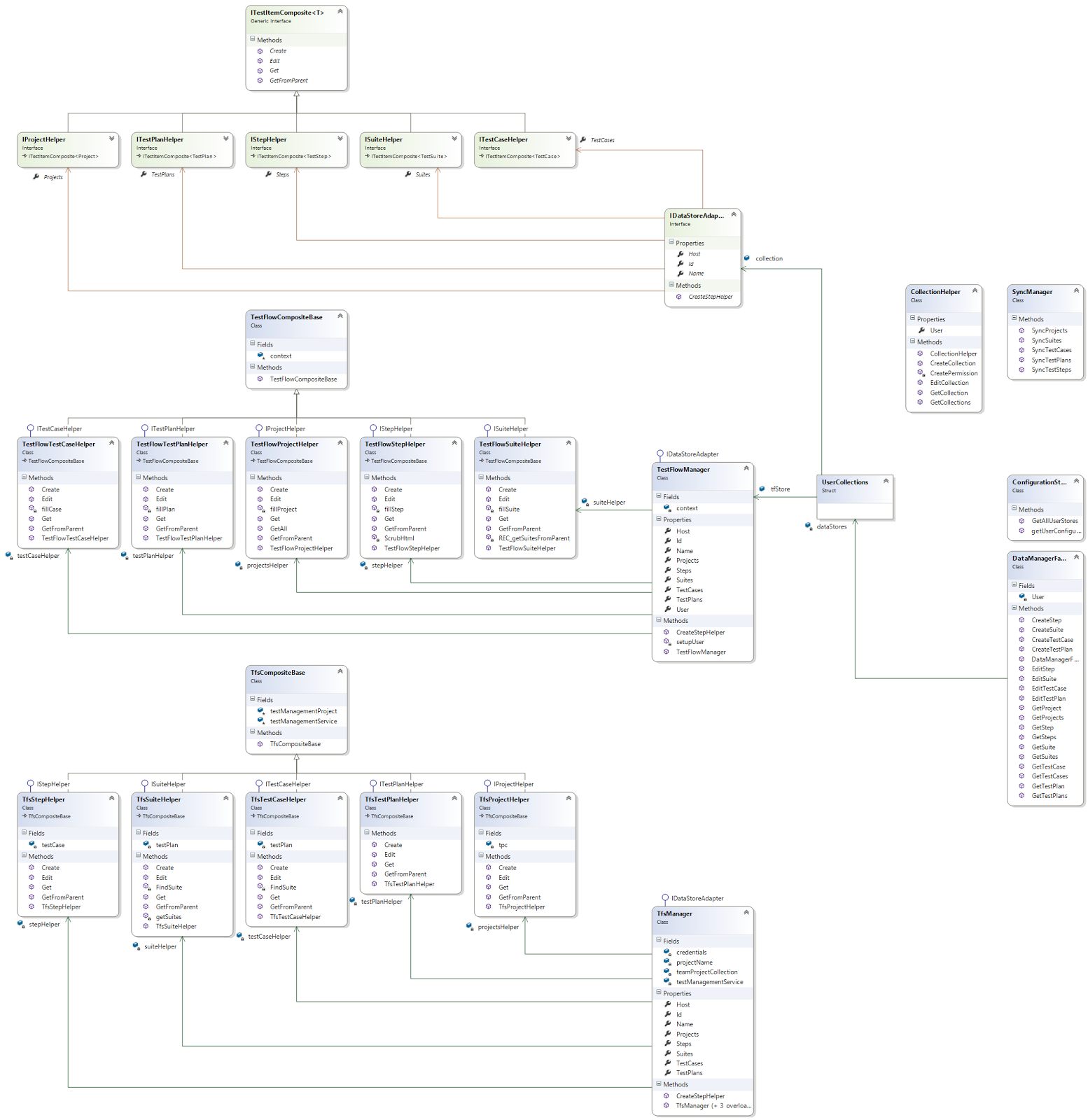




Minimal Class Diagram

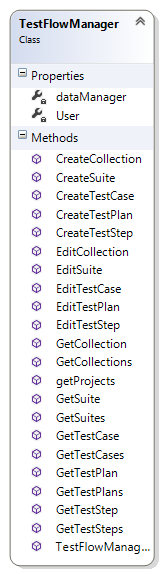
## **9.5. Appendix E – Design models (static and dynamic)**

DataStore

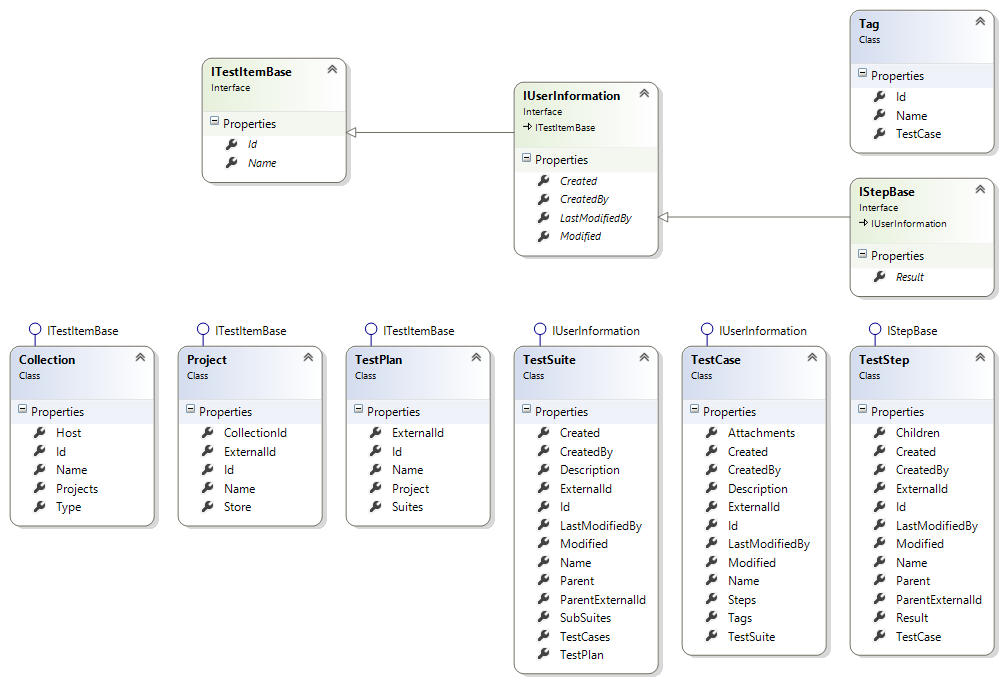


**Note that a more readable version can be found in the GIT files under UML.**

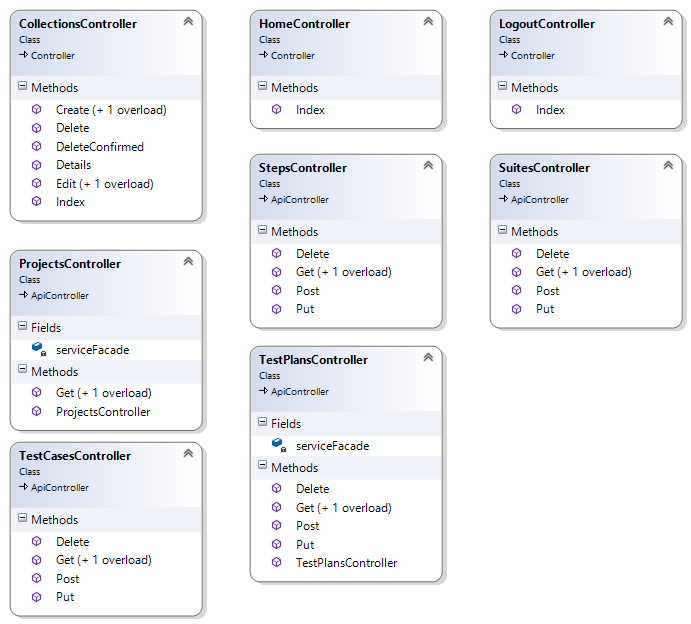
TestFlowManager



TestFlowModel



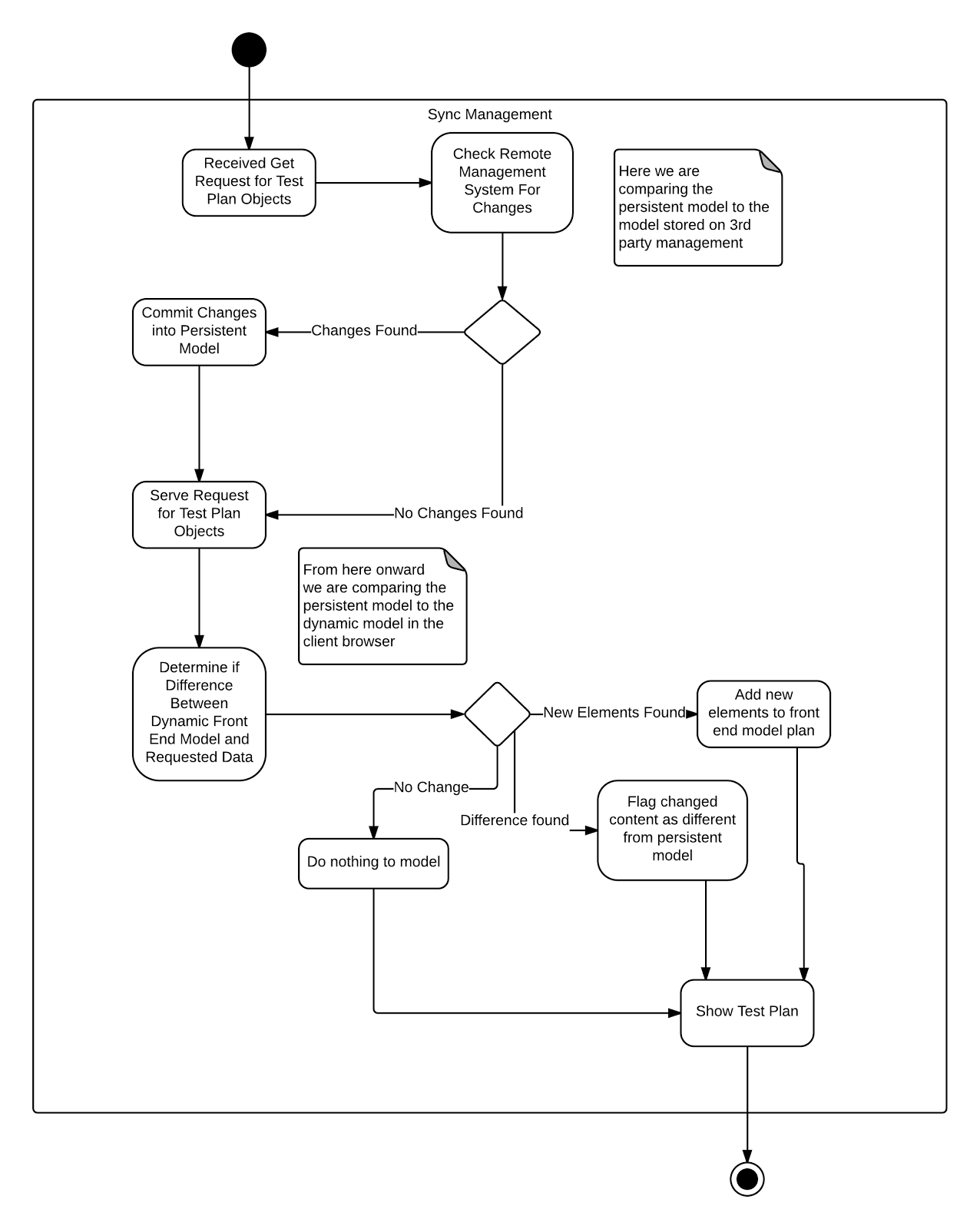
TestFlowWebApp



**Note that the majority of the TestFlowWebApp is implemented in JavaScript which does not have formal diagrams.**

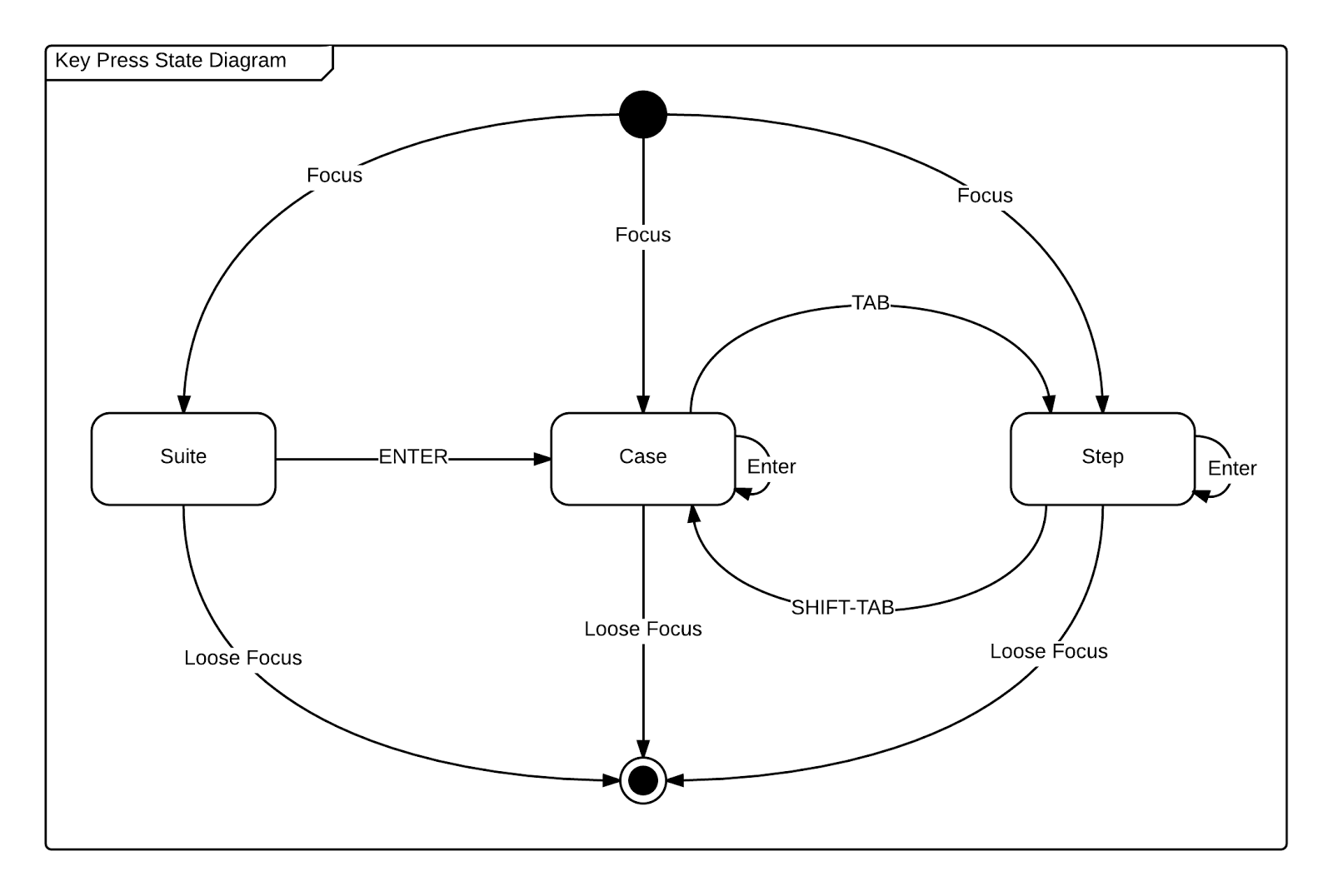
## **3.3. Dynamic model**

*DataStore*



The above activity diagram demonstrates the flow of information and changes through the system, essentially how the data is synchronized across platforms. It begins with a request for test plan data and ends with the data being shown after synchronizing with external sources and any changes in the user interface model.

*TestFlowWebApp*



The above state diagram represents focus and keypress action states within the user interface. Ultimate software required rapid type changing of elements via key presses. When an element has focus in the user interface this state diagram is used to determine what action is performed when certain keys are pressed.

## **9.6. Appendix F – Documented Class interfaces (code) and constraints.**

**DataStore Interfaces**

**ITestItemComposite**

/// <summary>

/// This composite pattern allows for the creation, editing, retrieving, and maping of test items

/// </summary>

/// <typeparam name="T"></typeparam>

public interface ITestItemComposite<T>

{

int Create(T item);

bool Edit(T item);

T Get(int id);

List<T> GetFromParent(int parentId);

}

**IProjectHelper**

public interface IProjectHelper : ITestItemComposite<Project>

{

}

**ITestPlanHelper**

public interface ITestPlanHelper : ITestItemComposite<TestPlan>

{

}

**ISuiteHelper**

public interface ISuiteHelper : ITestItemComposite<TestSuite>

{

}

**ITestCaseHelper**

public interface ITestCaseHelper : ITestItemComposite<TestCase>

{

}

**IStepHelper**

public interface IStepHelper : ITestItemComposite<TestStep>

{

}

**IDataStoreAdapter**

/// <summary>

/// Adapter interface providing common functionality for any data store connectivity.

/// </summary>

public interface IDataStoreAdapter

{

Uri Host { get; set; }

int Id { get; set; }

string Name { get; set; }

// helper objects for each test item type

IProjectHelper Projects { get; }

ITestPlanHelper TestPlans { get; }

ISuiteHelper Suites { get; }

ITestCaseHelper TestCases { get; }

IStepHelper Steps { get; }

// create instances of helpers after manager is created.

void CreateStepHelper(int testCaseId);

}

**TestFlowModel**

**ITestItemBase**

public interface ITestItemBase

{

int Id { get;set; }

string Name { get;set; }

}

**IUserInformation**

public interface IUserInformation : ITestItemBase

{

int LastModifiedBy { get; set; }

int CreatedBy { get; set; }

DateTime Created { get; set; }

DateTime Modified { get; set; }

}

**IStepBase**

public interface IStepBase : IUserInformation

{

string Result { get;set; }

}

**TestFlowManager**

**ITestFlowManager**

public interface ITestFlowManager

{

private DataManagerFacade dataManager;

private IPrincipal User;

public TestFlowManager(IPrincipal user, bool collectionManagement);

public TestFlowManager(IPrincipal user, int projectId = -1);

public TestFlowManager(IPrincipal user, int projectId, int testPlanId);

public TestFlowManager(IPrincipal user, int projectId, int testPlanId, int testCaseId);

// Project Stuff

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public List<Project> getProjects();

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end Project Stuff

// TestPlan Stuff

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public TestPlan GetTestPlan(int testPlanId);

public List<TestPlan> GetTestPlans(int projectId);

public void EditTestPlan(TestPlan testPlan);

public void CreateTestPlan(TestPlan testPlan);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end TestPlan Stuff

// Suite Stuff

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public TestSuite GetSuite(int suiteId);

public List<TestSuite> GetSuites(int testPlanId);

public bool EditSuite(TestSuite suite);

public int CreateSuite(TestSuite suite);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end Suite Stuff

// TestCase Stuff

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public TestCase GetTestCase(int testCaseId);

public List<TestCase> GetTestCases(int suiteId);

public bool EditTestCase(TestCase testCase);

public int CreateTestCase(TestCase testCase);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end TestCase Stuff

// Step Stuff

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public TestStep GetTestStep(int testStepId);

public List<TestStep> GetTestSteps(int projectId);

public bool EditTestStep(TestStep step);

public int CreateTestStep(TestStep step);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end Step Stuff

// collection management metods

// Collections stuff

public void EditCollection(Collection collection);

public Collection GetCollection(int id);

public List<Collection> GetCollections();

public void CreateCollection(Collection collection, int type);

}

## **9.7. Appendix G – Documented code for test drivers and stubs.**

// these are dependant on your database

string username = @"JUSTIN-DESKTOP\Justin";

string projectId = "HelloWorld";

int testPlanId = 6;

int testCaseId = 52;

int suiteId = 51;

[TestMethod]

public void TestPlanTestMethod()

{

TfsManager tfManager = new TfsManager(new Uri("http://tc-dev.cis.fiu.edu:8080/tfs/DefaultCollection"), projectId);

TestPlan createdPlan = new TestPlan();

createdPlan.Name = "Unit Test " + DateTime.Now.ToString();

createdPlan.ExternalId = 0;

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.TestPlans.Create(createdPlan), -1);

// get what was created

TestPlan check = tfManager.TestPlans.Get(createdPlan.Id);

// check that it was the right name

Assert.AreEqual(createdPlan.Name, check.Name);

// edit the name

createdPlan.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.TestPlans.Edit(createdPlan));

// get the edit from db

check = tfManager.TestPlans.Get(createdPlan.Id);

// check if they are equal

Assert.AreEqual(createdPlan.Name, check.Name);

}

[TestMethod]

public void SuiteTestMethod()

{

TfsManager tfManager = new TfsManager(new Uri("http://tc-dev.cis.fiu.edu:8080/DefaultCollection"), projectId, testPlanId);

TestSuite createdPlan = new TestSuite();

createdPlan.Name = "Unit Test " + DateTime.Now.ToString();

createdPlan.TestPlan = testPlanId;

createdPlan.ParentExternalId = 0;

createdPlan.Modified = DateTime.Now;

createdPlan.Created = DateTime.Now;

createdPlan.ExternalId = 0;

createdPlan.Parent = 0;

createdPlan.ParentExternalId = 0;

createdPlan.Description = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.Suites.Create(createdPlan), -1);

// get what was created

TestSuite check = tfManager.Suites.Get(createdPlan.Id);

// check that it was the right name

Assert.AreEqual(createdPlan.Name, check.Name);

// edit the name

createdPlan.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.Suites.Edit(createdPlan));

// get the edit from db

check = tfManager.Suites.Get(createdPlan.Id);

// check if they are equal

Assert.AreEqual(createdPlan.Name, check.Name);

}

[TestMethod]

public void TestCaseTestMethod()

{

TfsManager tfManager = new TfsManager(new Uri("http://tc-dev.cis.fiu.edu:8080/DefaultCollection"), projectId, testPlanId);

TestCase createdCase = new TestCase();

createdCase.Name = "Unit Test " + DateTime.Now.ToString();

createdCase.TestSuite = new TestSuite();

createdCase.TestSuite.Id = suiteId;

createdCase.Modified = DateTime.Now;

createdCase.Created = DateTime.Now;

createdCase.ExternalId = 0;

createdCase.Description = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.TestCases.Create(createdCase), -1);

// get what was created

TestCase check = tfManager.TestCases.Get(createdCase.Id);

// check that it was the right name

Assert.AreEqual(createdCase.Name, check.Name);

// edit the name

createdCase.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.TestCases.Edit(createdCase));

// get the edit from db

check = tfManager.TestCases.Get(createdCase.Id);

// check if they are equal

Assert.AreEqual(createdCase.Name, check.Name);

}

[TestMethod]

public void TestStepTestMethod()

{

TfsManager tfManager = new TfsManager(new Uri("http://tc-dev.cis.fiu.edu:8080/DefaultCollection"), projectId, testPlanId, testCaseId);

TestStep createdStep = new TestStep();

createdStep.Name = "Unit Test " + DateTime.Now.ToString();

createdStep.TestCase = testCaseId;

createdStep.Modified = DateTime.Now;

createdStep.Created = DateTime.Now;

createdStep.ExternalId = 0;

createdStep.Result = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.Steps.Create(createdStep), -1);

// get what was created

TestStep check = tfManager.Steps.Get(createdStep.Id);

// check that it was the right name

Assert.AreEqual(createdStep.Name, check.Name);

// edit the name

createdStep.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.Steps.Edit(createdStep));

// get the edit from db

check = tfManager.Steps.Get(createdStep.Id);

// check if they are equal

Assert.AreEqual(createdStep.Name, check.Name);

}

// these are dependant on your database

string username = @"JUSTIN-DESKTOP\Justin";

int projectId = 14;

int testPlanId = 11;

int testCaseId = 1003;

int suiteId = 96;

[TestMethod]

public void TestPlanTestMethod()

{

TestFlowManager tfManager = new TestFlowManager(username);

TestPlan createdPlan = new TestPlan();

createdPlan.Name = "Unit Test " + DateTime.Now.ToString();

createdPlan.Project = new Project();

createdPlan.Project.Id = projectId;

createdPlan.ExternalId = 0;

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.TestPlans.Create(createdPlan), -1);

// get what was created

TestPlan check = tfManager.TestPlans.Get(createdPlan.Id);

// check that it was the right name

Assert.AreEqual(createdPlan.Name, check.Name);

// edit the name

createdPlan.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.TestPlans.Edit(createdPlan));

// get the edit from db

check = tfManager.TestPlans.Get(createdPlan.Id);

// check if they are equal

Assert.AreEqual(createdPlan.Name, check.Name);

}

[TestMethod]

public void SuiteTestMethod()

{

TestFlowManager tfManager = new TestFlowManager(username);

TestSuite createdPlan = new TestSuite();

createdPlan.Name = "Unit Test " + DateTime.Now.ToString();

createdPlan.TestPlan = testPlanId;

createdPlan.ParentExternalId = 0;

createdPlan.Modified = DateTime.Now;

createdPlan.Created = DateTime.Now;

createdPlan.ExternalId = 0;

createdPlan.Parent = 0;

createdPlan.ParentExternalId = 0;

createdPlan.Description = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.Suites.Create(createdPlan), -1);

// get what was created

TestSuite check = tfManager.Suites.Get(createdPlan.Id);

// check that it was the right name

Assert.AreEqual(createdPlan.Name, check.Name);

// edit the name

createdPlan.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.Suites.Edit(createdPlan));

// get the edit from db

check = tfManager.Suites.Get(createdPlan.Id);

// check if they are equal

Assert.AreEqual(createdPlan.Name, check.Name);

}

[TestMethod]

public void TestCaseTestMethod()

{

TestFlowManager tfManager = new TestFlowManager(username);

TestCase createdCase = new TestCase();

createdCase.Name = "Unit Test " + DateTime.Now.ToString();

createdCase.TestSuite = new TestSuite();

createdCase.TestSuite.Id = suiteId;

createdCase.Modified = DateTime.Now;

createdCase.Created = DateTime.Now;

createdCase.ExternalId = 0;

createdCase.Description = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.TestCases.Create(createdCase), -1);

// get what was created

TestCase check = tfManager.TestCases.Get(createdCase.Id);

// check that it was the right name

Assert.AreEqual(createdCase.Name, check.Name);

// edit the name

createdCase.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.TestCases.Edit(createdCase));

// get the edit from db

check = tfManager.TestCases.Get(createdCase.Id);

// check if they are equal

Assert.AreEqual(createdCase.Name, check.Name);

}

[TestMethod]

public void TestStepTestMethod()

{

TestFlowManager tfManager = new TestFlowManager(username);

TestStep createdStep = new TestStep();

createdStep.Name = "Unit Test " + DateTime.Now.ToString();

createdStep.TestCase = testCaseId;

createdStep.Modified = DateTime.Now;

createdStep.Created = DateTime.Now;

createdStep.ExternalId = 0;

createdStep.Result = "This is a unit test generated element.";

// attempt to create, -1 if failed

Assert.AreNotEqual(tfManager.Steps.Create(createdStep), -1);

// get what was created

TestStep check = tfManager.Steps.Get(createdStep.Id);

// check that it was the right name

Assert.AreEqual(createdStep.Name, check.Name);

// edit the name

createdStep.Name.Replace("Unit Test ", "");

// attempt to make edit

Assert.IsTrue(tfManager.Steps.Edit(createdStep));

// get the edit from db

check = tfManager.Steps.Get(createdStep.Id);

// check if they are equal

Assert.AreEqual(createdStep.Name, check.Name);

}

## **9.8. Appendix H – Diary of meeting and tasks for the entire semester.**

**Diary Entry 1:**

**Date:** Thursday, September 5, 2014

**Location:** ECS Lab 212

**Start time**: 3:00 pm

**End time:** 4:35 pm

**In Attendance:** Dionny Santiago, Tariq King, Justin Phillips, Karina Harfouche

**Late:** N/A

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *09/17/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Project Complexity**  Time Allotted: 15 minutes  Decision: The model aspect of the system shall be fully implemented by the team and an adapter interface shall be used to leverage 3rd party management systems and migration.  Responsible Individuals: N/A |
| **Item 2: Functional Requirements and System Actors**  Time Allotted: 15 minutes  Decision: Review of the functional requirements of the system based on the use case diagram. The client asked that the use case diagram show added complexity of the system, such as the login begin required and that all use cases be visible, perhaps through generalizations or other relationship notation.  Responsible Individuals: N/A |
| **Item 6: Finalize Functional Requirements**  Time Allotted: 5 minutes  Decision: Based on the use case diagram feedback the requirements were finalized.  Responsible Individuals: Justin / Karina |
| **Item 6: Present Project Timeline**  Time Allotted: 10 minutes  Decision: It was decided the timeline was acceptable, though optimistic. It may need to be refactored as we end the requirements phase. Rushing the requirements phase was highly discouraged.  Responsible Individuals: Justin / Karina |

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *09/22/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Feedback on Use Cases**  Time Allotted: 15 minutes  Decision: Still pending review from the client  Responsible Individuals: Karina Harfouche |
| **Item 2: Feedback on use case diagram and sequence diagrams**  Time Allotted: 15 minutes  Decision: Mentor suggested more relationships on the use cases and further complexity in the sequence diagrams.  Responsible Individuals: Justin Phillips |
| **Item 3: Logistics**  Time Allotted: 5 minutes  Decision: Server capacity, hosting, etc...  Responsible Individuals: Justin / Karina |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *09/29/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Feedback on use case diagram and sequence diagrams**  Time Allotted: 15 minutes  Decision: Final diagrams were moved to done.  Responsible Individuals: N/A |
| **Item 2: Discuss Potential Features**  Time Allotted: 15 minutes  Decision: Mentor recommended a stand alone model to be used along side TFS which Justin will implement for two way flow of data. Karina recommended metrics to the client which they accepted.  Responsible Individuals: Justin & Karina |
| **Item 3: Project Summary**  Time Allotted: 5 minutes  Decision: Summarized where we are at and what we have done so far.  Responsible Individuals: Justin / Karina |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *10/08/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Show Demo**  Time Allotted: 15 minutes  Decision: Feedback from client was very positive, client may schedule company meeting for us to do the demo in.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *10/15/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Review of system architecture**  Time Allotted: 15 minutes  Decision: Great feedback from Dionny and some architectural pattern suggestions.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *10/22/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Review of minimal class diagram**  Time Allotted: 15 minutes  Decision: Feedback was that diagram was clean and a good start.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *10/29/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Discussion of what is required for the company demo**  Time Allotted: 15 minutes  Decision: Feedback from client was very positive, client may schedule company meeting for us to do the demo in.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *11/05/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: New company demo schedule**  Time Allotted: 15 minutes  Decision: New deadline for our presentation at Ultimate’s company wide QA meeting.  Responsible Individuals: N/A  **Item 1: Two features to be implemented before demo**  Time Allotted: 15 minutes  Decision: File upload (Karina) and drag-and-drop / new keybind system (Justin)  Responsible Individuals: Justin and Karina |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Impromptu Weekend Meeting*** | Date: *11/8/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Show Demo**  Time Allotted: 15 minutes  Final demo for QA meeting, great response from Tariq and Dionny.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Ultimate Company Wide QA Meeting*** | Date: *11/11/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: QA Meeting Presentation**  Time Allotted: 15 minutes  Decision: Great feedback from mentor about our presentation.  Responsible Individuals: N/A |

**Meeting Minutes**

|  |  |
| --- | --- |
| Meeting: ***Regularly Schedule Meeting*** | Date: *11/19/2014* |
| Meeting Minutes Taken By: *Justin Phillips and Karina Harfouche* |  |
| Attendance: *Tariq King, Dionny Santiago* |  |

|  |
| --- |
| **Minutes:** |
| **Item 1: Moving forward**  Time Allotted: 15 minutes  Decision: Received some suggestions on path forward.  Responsible Individuals: N/A |

## 9.9 Appendix I - Feasibility Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Characteristics** | **Option 1**  **Web Application** | **Option 2**  **VS plug-in** | **Option 3 IBM ClearQuest** |
| **Technical** | Low Risk, Flexible, Future enhancement possible  *Score: 5* | High Risk, Inflexible, Personalized Enhancements not possible  *Score: 3* | Low Risk, Inflexible, Personalized Enhancements not possible, redundant data  *Score: 2* |
| **Operational** | No User Resistance  *Score: 5* | User Resistance  *Score: 2* | User Resistance  *Score: 3* |
| **Economic** | $31,560 | $35,560.00 | $20,160 |
| **Expiration Time** | Indefinite  Score: 5 | Dependent on Visual Studio Licensing  Score: 3 | 12 Months  *Score: 1* |
| **Estimated Configuration & Completion Time** | 3 to 4 Months  *Score: 3* | 2.5 Months  *Score: 4* | 2 Months  *Score: 5* |
| **Total Score** | **18/20** | **12/20** | **11/20** |

# 10. References

1. COCOMO II:  
   <http://csse.usc.edu/tools/COCOMOII.php>
2. IBM Rational ClearQuest:  
   <http://www-03.ibm.com/software/products/en/clearquest>
3. Microsoft Test Manager  
   <http://msdn.microsoft.com/en-us/library/jj635157.aspx>
4. Microsoft Team Foundation Server  
   <http://msdn.microsoft.com/en-us/library/ms181238(v=vs.90).aspx>
5. Wikipedia: Software Testing  
   <http://en.wikipedia.org/wiki/Software_testing>

Test planning using IBM Rational Quality Manager  
<http://www.ibm.com/developerworks/rational/library/09/0203_kelly/>