بسم الله الرحمن الرحيم

Sudan University of Science and Technology

Faculty of computer science and information technology

# Redesigning the States' Budget Management System Using Model View Controller to Increase Maintainability

إعادة تصميم نظام إعداد وتنفيذ موازنة الولايات بإستخدام نمط التصميم (Model View Controller) لزيادة قابلية الصيانة

A Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Information Technology

By:

Omer Abdelmajeed Idrees Mohammed

Supervisor:

Dr. Tariq Omer Fadl Elsid

July 2018

## DEDICATION

To my father and my mother, my wife and children, my brothers and sisters, and my second family abdeldayem’s family, who are the most valuable people in my life, To all my friends my honest mirror. To all of you a lot of thanks and you will always be remembered with your support and kindness.

## ACKNOWLEDGEMENTS

First of all praise and thank to Almighty Allah the Lord of the universe before and after, who granted me success to accomplish this study and for everything in my life.

Second, I would like to thank Dr. Hisham Mansour for the great effort that he done in teaching us new web technologies during this study.

I would like to express my appreciation to my supervisor Dr. Tariq Omer for his support and guidance.

"Peace and Blessings be upon our Prophet Mohamed"

## ABSTRACT

In 2013 the ministry of finance and economic established the budget management system, following the Sudanese national plan toward the E-government, this system is now absolutely outdated and hard to adapt any new changes as a result for its poor design quality. A new version of the system is redesigned implementing the Model View Controller (MVC) pattern. This new version of the system is evaluated by comparing the architecture of the current and the redesigned versions of the system in terms of their maintainability, this comparison is done by the Analysis method for Software Architecture Modifiability (ASAM), analyzing the single responsibility principal (SRP) for both versions and reporting any violations that any version has. For the current system the study found that the current system is not following a known design pattern, or was designed in ad-hoc pattern, after applying the analysis scenario  found that all current system components are violating the SRP, the reason why is lacking for maintainability. For the redesigned version, the study found that all its components following the SRP which is evaluating it to have a higher maintainability. Further research is recommended to assess the system Scalability, where the system is expected to experience a huge traffic and handling a great load of data after is fully adopted.

## المستخلص

في العام 2013 دشنت وزارة المالية والإقتصاد نظام إعداد وتنفيذ الموازنة تحقيقاً للخطة القومية السودانية نحو حكومة إلكترونية. هذا النظام حالياً اصبح غير فعال نسبة لصعوبة إجراء أي تعديلات عليه وذلك نتيجة لتدني جودة تصميمه. تم تصميم نسخة جديدة من النظام بإتباع نمط التصميم (Model View Controller) والمعروف إختصاراً بـ(MVC). حيث انه تم تقييم خاصية قابلية الصيانة للنسخة المعاد تصميمها والنسخة الحالية من النظام بإجراء مقارنة بين معمارية كل منهم. وهذه المقارنة  تمت بإستخدام طريقة تحليل قابلية معمارية البرمجيات للتعديل (Analysis method for Software Architecture Modifiability) والمعروفة إختصاراً بـ(ASAM), وذلك بتحليل مبدأ المسؤولية الواحدة المعروف ( Single Responsibility Principal) والمعرف إختصاراً بـ(SRP) لكل من النظامين وتقرير اي إنتهاكات لهذا المبدأ يحدثها اي منهم. حيث وجدت الدراسة أن النظام الحالي لا يتبع أي نمط في تصميمه, وبعد تطبيق سناريو التحليل عليه وجد أن جميع مكوناته تنتهك مبدأ المسؤولية الواحدة وبالتالي إنخفضت لديه قابلية الصيانة كثيراً. اما بالنسبة للنسخة المعاد تصميمها وجدت الدراسة أن جميع مكوناته تتبع لمبدأ المسؤولية الواحدة وبالتلي تم تقييمها بأنها تملك قابلية صيانة أعلى من النظام الحالي. وبعد أن يتم إعتماد النظام بصورة كاملة سيكون هنالك ضغط عالي عليه, وبالتالي يقترح القيام بالبحث أكثر في كيفية رفع قيمة قابلية النظام للتوسع والتي من شأنها جعل النظام يعمل بصورة مستقرة ومستمرة ومستوعبة لكل اعباء التشغيل الأضافية التي قد تطرأ مع زيادة الإستخدام.

# Table of Contents

# List Of Tables

# List Of Figures

# List Of Abbreviations

| Acronym | Definition |
|---|---|
| M.V.C | Model View Controller |
| S.R.P | Single Responsibility Principal |
| A.S.A.M | Analysis software architecture for modifiability |

# List of Appendices

# CHAPTER I

# INTRODUCTION

## 1.1 Background:

In Software Engineering, a software have many attributes, those attributes determine its quality. Maintainability is the most interesting of these attributes, since any future changes, enhancements and faults correction would be categorized under the maintainability attribute. The software with a high maintainability will decrease the cost of maintenance, which is a basic aim of the software quality programs (Stavrinoudis, et al., August 1999).

The maintainability of software is the degree to which it can be understood, corrected, adapted and/or enhanced (Pressman, 1997). Software maintenance is the costliest phase of the software life cycle, where requirements are always changing to add a new feature or to change the business logic. Since requirements are always changing, the software would be in need to be continuously maintained.

It’s all about design. A software design quality plays a great role in determining the software maintainability. For this reason the software design needs to be as good as possible, and to accomplish this task there are many design patterns defined to improve the software design. Furthermore, applying design patterns is also considered good since design patterns can speed up the development process by providing tested and proven development paradigm. As concluded by Farooq “good maintainability can be achieved by implementation of Design Patterns in software development” (Abdullah, 2017)

In this research, the Model-View-Controller (M.V.C.) pattern is used to improve the current system maintainability by enhancing the Ease of Modification measured by the Single Responsibility Principal.

## 1.2 **Problem Statement:**

The current system is badly designed, no design pattern was adopted. Hence the system became hard and expensive to maintain (fixing bugs and adding new Features). Furthermore, the current system seem to be violating the Single Responsibility Principal—which also decreased the system maintainability. The Implementation of the M.V.C. Designing Pattern was proposed to solve this problem by eliminating those violations to increase the system's maintainability.

## 1.3 **Approach:**

This research investigates the current system and the re-designed system against the Single Responsibility Principal violations. Apply some changes on the system and report the SRP violations status for each system in the Results and discussion chapter. Finally, deciding which system has a higher maintainability by comparing the number of SRP violations found in each one. For simplicity, the Items Management Module in both systems only will be evaluated.

## 1.4 **Research Aims and Objectives**

The aim of this study is to redesign the budget management system to enhance its maintainability. Given the problem statement presented previously, the objectives of this study are to:

- Implement the MVC architectural design pattern.
- Apply the SRP principal.
- Compare the architecture of the current and the redesigned systems.

## 1.5 **Scope and limitation:**

This research was conducted to increase the maintainability of the “budget Management System” of the “River Nile State” during the 2017 – 2018 year.

Only the single responsibility principal is studied to measure the system maintainability, observing the interactions of the three responsibilities: data handling, user interaction control and view representation.

For simplicity and clearance, only the Budget Items Management module is studied.

## 1.6 **Evaluation and Results:**

This research evaluating the maintainability of the current and the redesigned systems by comparing the architecture of both of them, adopting the SRP violation as a comparison factor, analyzing each architecture by apply a scenario of changes and reporting any violations found. Finally evaluating the system with the minimum number of violations as the system with the higher maintainability.

.

## 1.7 **Thesis Structure**

This thesis comprises five chapters, followed by references and appendices.

Chapter 1: gives an introduction to the research subject.

Chapter 2: reviews the literature of maintainability, SRP principal, MVC pattern and the Scenario-Based software architecture analysis.

Chapter 3: presents the research methodology.

Chapter 4: presents the results and discussions.

Chapter 5: presents the conclusions of the work and some recommendations.

# CHAPTER II

# LITERATURE REVIEW

Software maintainability enhancement is a very interesting topic in the research community, since maintainability affects almost every stage of the software development stages. What is really challenging is finding a model that covering a wide range of the characteristics that affects the maintainability. In this chapter we reviewing the software maintainability attribute, studying how design patterns enhance maintainability covering the SRP principal, MVC pattern and the software architecture analysis methods.

## 2.1 **Software Maintainability:**

Software maintainability is a software quality factor that defined as the degree to which an application is understood, repaired, or enhanced.

McCall, Richards, and Walters defined many factors that affect the software quality, categorized in many aspects of a software product: product operation, product revision and product transition. These three categories and their related factors are shown in Figure (1):

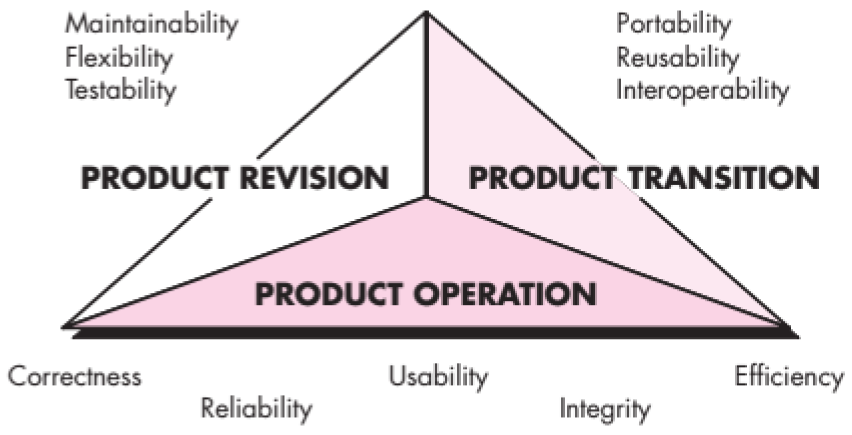


*Figure 1: McCall software quality factors*

This model categorize the maintainability in line with flexibility and testability under the product revision, where all those three factors at some point affected by the developers' abilities. Further, the maintainability is divided into four sub-attributes according to the ISO 9126, shown in figure (2), they: Analyzability, Changeability, Stability and Testability.

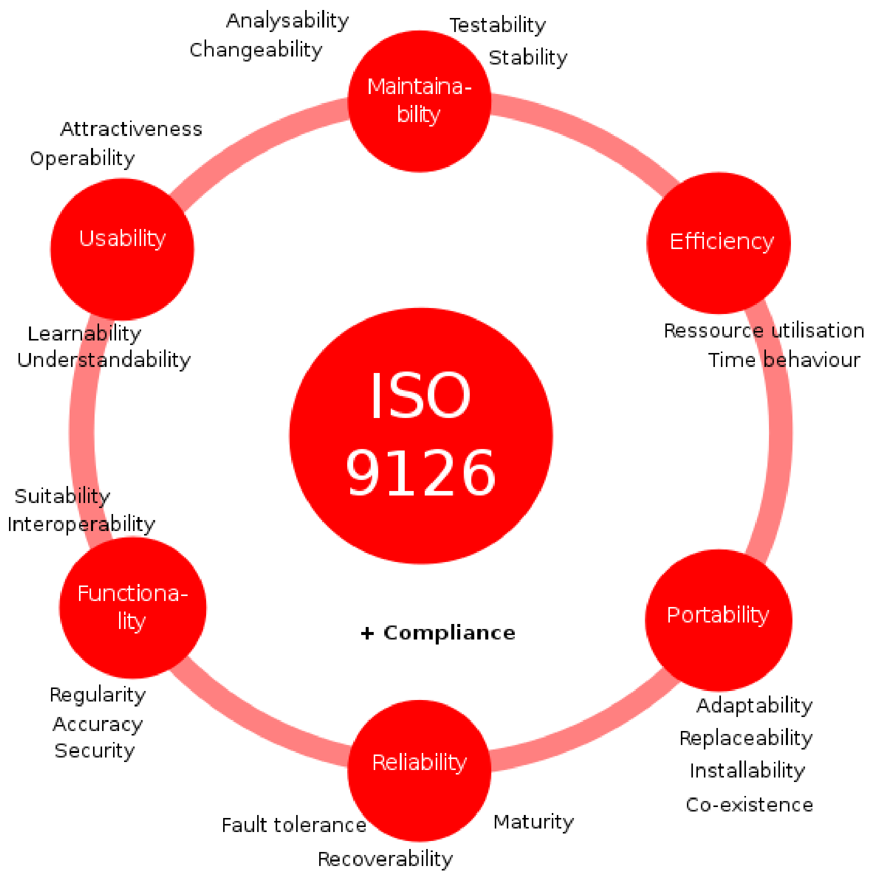


*Figure 2: ISO 9126 software quality factors*

The sub-attribute changeability of the maintainability is the targeted quality factor that needs to be addressed by this work, where the current system is hard and expensive to be changed.

## 2.2 **Design Pattern and Maintainability Enhancement:**

Abdullah said that Design Patterns are known to provide more maintainable and reusable code, as also in his research "Evaluating Impact of Design Patterns on Software Maintainability and Performance" concluded that good maintainability can be achieved by implementation of Design Patterns in software development (Abdullah, 2017). Similarly, Bass claimed that: "an effective architecture is one in which the most likely changes are also the easiest to make" (Bass, et al., 2003). Moreover, Buschmann defines two patterns to help designing for change, as he said that "Design for change is therefore a major concern when specifying the architecture of a software system" (Buschmann, et al., 1996). Also Fredrik Hoffman claim that M.V.C may enhance some aspects of maintainability, e.g. understandability, and describing that the way of evaluating this aspects of maintainability "could be to use a group of developers as subjects to experiment on" (Hoffman, 2001). All the previous studies indicates that applying design patterns on the software development improves software quality, specifically the software maintainability. Hence this work conducted to redesign the current system following the M.V.C pattern.

## 2.3 **Complexity Measurement:**

Software maintainability is linked to the software complexity, as Zuse said: "the term complexity measure is a misnomer. The true meaning of the term software complexity is the difficulty to maintain, change and understand software. ", which it can be divided into two parts: 1) Computational complexity: "refers to algorithm efficiency in terms of the time and memory needed to execute a program" and 2) Psychological (or cognitive) complexity: "refers to the human effort needed to perform a software task, or, in other words, the difficulty experienced in understanding or performing such a task" (Zuse, 1991), the following is a list of related work to the software complexity measurement, grouped based on Zuse's complexity classification:

### 2.3.1 **Computational complexity**:

Also known as Essential complexity (Brooks & Frederick, 1987), which complexity that caused by the characteristics of the problem to be solved, the following is a list of related work in this type of complexity:

- **The Cyclomatic complexity** (McCabe, 1976), introduce a quantitative measure of the number of linearly independent paths through a program's source code.
- **Halstead complexity measures** (Halstead & H., 1977), Introduced this measures which are computed statically from the code.
- **Software Structure Metrics Based on Information Flow** (Henry & Kafura, 1981), introduced these metrics which measure the procedure complexity by calculating number of local flows into that procedure plus the number of data structures from which that procedure retrieves information and calculating the number of local flows out of that procedure plus the number of data structures that the procedure updates.
- **A Metrics Suite for Object Oriented Design** (Chidamber & Kemerer, 1994), introduced these metrics weighting methods per class, coupling between object classes, response for a class, number of children, depth of inheritance tree and lack of cohesion of methods.

### 2.3.2 **Psychological complexity:**

Also known as Accidental complexity (Brooks & Frederick, 1987), is complexity that Relates to difficulties a developer faces, the following is a list of related work in this type of complexity:

- **The cognitive weight**, introduced in 2003 (Shao & Yingxu Wang, APRIL 2003 ), which is defined as the is the degree of difficulty or relative time and effort required for comprehending a given piece of software modelled by a number of basic control structures.
- **Task complexity model**, introduced by Wood (R.E, n.d.), is generally representative of task complexity and defined four theoretical frameworks: 1)

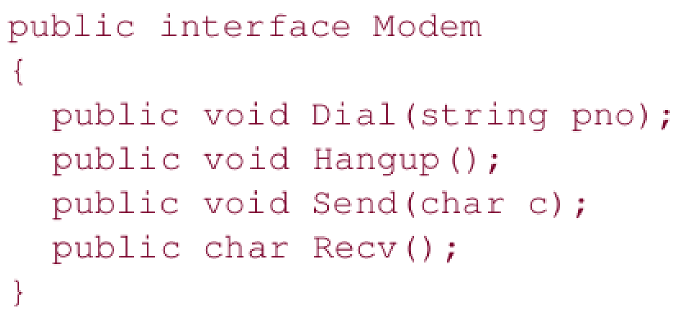
task qua task, 2) task as behavior requirements, 3) task as behavior description and 4) task as ability requirements.

## 2.4 **The Single Responsibility Principal:**

The Single Responsibility Principle (SRP), is one of the five design principles intended to make software designs more understandable, flexible and maintainable (Wikipedia, 2014). And this principal states that each component of the software module must be responsible for only one responsibility. Robert C. Martin defines this principle as following: "A class should have only one reason to change." (Martin, 2005)

### **2.4.1 Responsibility definition:**

The responsibility is defined as “a reason for change” according to Robert C. Martin, Describing that a class or component with more than one motive for changing is a class with more than one responsibility. As an example, Martin considered a modem interface (Figure 1), has two responsibilities. The first is connection management. And the second is data communication.



```
public interface Modem
{
  public void Dial(string pno);
  public void Hangup();
  public void Send(char c);
  public char Recv();
}
```

*Figure 3:* Modem Interface, by Robert C. Martin

The dial and hangup functions manage the connection of the modem; the send and recv functions communicate data.

### 2.4.2 The principal limitation:

The current S.R.P is concerned with the class or the software component structure only (cohesion and coupling), based on the functionality of that class or component. In this there may be a class or a component with only one functionality, but it may requires more than one developing skills in multiple technologies, this issue of having more than one development skills and technologies required to develop the same class or component will lead to a complicated maintenance tasks, where more communication needed, multiple-technologies developers and reducing the chances of the paralleling development ability which leads to consume more time in the development.

To get over this limitation this study extends the single responsibility principal by defining another dimension to it. This dimension takes the required development skills into consideration, this way responsibilities are defined also based on the development abilities. Hence the extended SRP may be defined also as: “a component should have only one reason to develop”. Where the responsibility now is defined as “a reason for development”. This extension defines three types of responsibilities that this work study, are: 1) data handling responsibility, 2) user interaction control responsibility and 3) view representation responsibility. These responsibilities are facilitated by the MVC architectural design pattern which is presented in the following section.

## 2.5 **The M.V.C. architectural design pattern:**

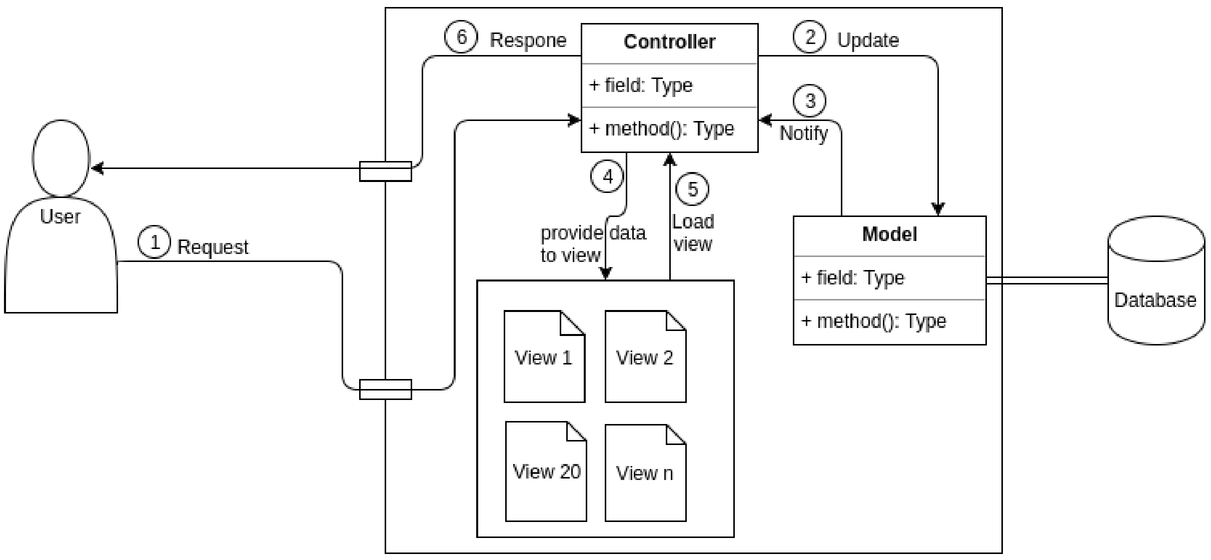


*Figure 4: Illustrates the M.V.C. architectural design pattern*

The M.V.C. in short, is an architectural design pattern invented at Xerox Parc in the 70's, probably by Trygve Reenskaug (Wiki, 2014). It divides a given application into three interconnected parts (Burbeck, 1992). This is done to separate the internal representations of information from the ways information is *presented to* and *accepted from* the user (Reenskaug & James, 2009).

M.V.C. became one of the first approaches to describe and implement software constructs in terms of their responsibilities (Wiki, 2014). In this approach, each component has its own responsibility, these responsibilities are defined according to Steve Smith (Smith, 2018) as following:

### 2.5.1 Model Responsibilities

The Model in an M.V.C. application represents the state of the application and any business logic or operations that should be performed by it. Business logic should be encapsulated in the Model, along with any implementation logic for persisting the state of the application.

### 2.5.2 View Responsibility:

The View is responsible for presenting content through the user interface. There should be minimal logic within the View. Otherwise, any logic in the View should be related to presenting content.

### 2.5.3 Controller Responsibility:

The Controller is the component that handles user interaction, works with the model, and ultimately selects a View to be rendered. In an M.V.C. application, the View only displays information; the controller handles and responds to user input and interaction. In the M.V.C. pattern, the controller is the initial entry point, and is responsible for selecting which Model types to work with and which View to render. Hence, its name the Controller; it controls how the app responds to a given request.

## 2.6 **How the MVC improves the maintainability:**

The MVC separates the responsibilities between the three parts, this delineation of responsibilities increases the maintainability in terms of complexity because it’s easier to code, debug, and test (i.e. the model, view, or controller) that has a single job (that follows the Single Responsibility Principle) (Smith, 2018). Furthermore, the MVC supports the extended SRP principal, by offering a solution that best fits the system under study. Where the system components are supported by the MVC as following:

- Data handling component supported by the Model layer
- User interaction component supported by the Controller layers
- View representation component supported by the View layer

## 2.7 **Architecture analysis methods:**

A software architecture describes the structure of a software system on an abstract implementation independent level. (Züllighoven H., 2006). And to compare alternative software architectures, the candidate architectures must be analyzed first, and then using the analysis results in determining the architecture that is the best choice. There are many methods serve this purpose, the earliest on is the architecture tradeoff analysis (ATAM) method (Rick, et al., 2000), analyzing various quality attributes and identifying a tradeoff points between them. Also, the Software Architecture Analysis Method (SAAM) method (Kazman, et al., 2007), this method is used to provide a method for determining which architecture supports an organization's needs. Moreover, the Software Architecture Comparison Analysis Method (SACAM), which provides organizations with a rationale for an architecture selection process by comparing the fitness of software architecture candidates being used in envisioned systems, this method requires the availability of architectural documentation to perform the comparison criteria analysis (Christoph, et al., 2003). Furthermore, the Analysis of software architecture for maintainability (ASAM) method (Bengtsson, et al., 2000) was proposed to analyze the software architecture for modifiability. The first three methods ATAM, SAAM and SACAM are used to analyze any quality attribute or attributes of interest. But ASAM is only specific for the modifiability attribute. Hence, the ASAM method it will be used to compare the architecture of the current and the redesigned systems in terms of maintainability.

In this chapter we looked deep in the maintainability attribute in both "McCall" and the "ISO 9126" models, identifying the correlated attributes and sub attributes, as also managed to capture the knowledge of how design pattern improves the maintainability. In terms of measurement we reviews both the computational and psychological complexity measurements, extending the SRP principle to bet fits the purpose of this study. Furthermore, we decided software architecture analysis is works for this work

# CHAPTER III

# RESEARCH METHODOLOGY

Software architecture analysis to predict quality attributes of the system under design is a very important activity in every stage of the development. Most methods used to predict quality attributes are based on code metrics, which it will makes the prediction available after the code is present, which may discover problems too late. Hence, scenario-based methods were proposed to address this limitation where they can be conducted in the early stages of the development. In this chapter we will present the ASAM method, determine the goal of the analysis, defining scenarios elicitation, scenario evaluation, scenario result expression, scenario results interpretation techniques and describe the software architecture.

## 3.1 **The Analysis software architecture for modifiability method (ASAM):**

This is a Scenario-Based architecture analysis method, proposed by Bengtsson. That can be used for three different goals: cost prediction, risk assessment and software architecture selection. Consisting of five steps: setting the analysis goal, describing software architecture, eliciting scenarios, evaluating scenarios and interpreting results. Using certain techniques in the following steps: scenario elicitation, scenario evaluation and architecture description, based on the relationship between those techniques, as shown in the figure (5).

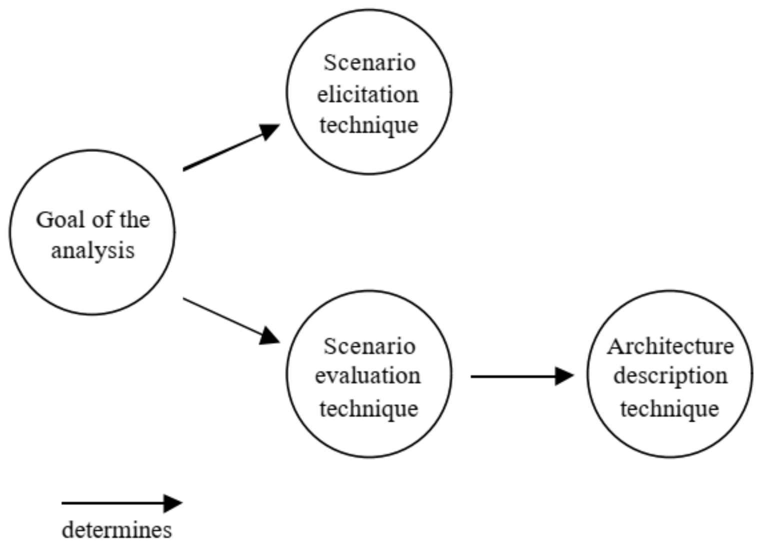

Goal of the analysis
Scenario elicitation technique
Scenario evaluation technique
Architecture description technique
determines

*Figure 5: ASAM different stages techniques relationship*

Based on the goal of the analysis, a technique for scenario elicitation and the scenario evaluation techniques are selected. Subsequently, architecture description technique is selected. The goal that best fits the purpose of this study is the software architecture selection, to find the architecture with the higher maintainability.

## 3.2 **Scenario elicitation:**

Change scenario elicitation is the process of finding and selecting the change scenarios that are to be used in the evaluation step of the analysis. There are two approaches for selecting a set of scenarios: top-down and bottom-up. This study use the top-down approach, by using some predefined classification of change categories to guide the search for change scenarios, those categories were derived from the domain of interest, which is the component responsibility, concluded into three categories:

- Data Handling (DH) category: where all changes classified under this category are being developed to handle the system data.
- User Interaction (UI) category: where all changes classified under this category are being developed to affecting the control of the user interaction with the system.

- View representation (VR) category: where all changes classified under this category are affecting the way the user interface.

## 3.3 **Scenario Evaluation:**

As a subsequent to the scenario elicitation, the scenario effect on the architecture should be evaluated. Since the change scenario sets were elicited according to the SRP principal, we assume that there are no ripple effects, where each change will affect only one component. This study evaluates the change scenario by determining which component was effected by the change.

## 3.4 **Results expression:**

According to this study scope which is adopting only the SRP principal as comparison criterion, we expressing the results of the scenario evaluation by ranking the candidate architectures depending on their violations to the SRP principal.

## 3.5 **Results Interpretation:**

Since the goal is to compare candidate architectures, results interpretation is aimed at deciding which candidate have higher maintainability. Whereas, the results expression was ranking the candidate architectures, then based on the minimum violation criterion we interpret that the candidate architecture with the minimum violation is the candidate with the higher maintainability.

## 3.6 **Architecture description:**

To be possible to evaluate the scenario sets and make the comparison between the current system and the redesigned, we need to describe both systems' architectures, which should provide information about each component decomposition and the relationships between the system components.

### 3.6.1 Current system architecture:

The current system is a web based application, built on html, css, php and sql technologies, is running on apache web server and its database is running on a mysql dbms. Consists of the following components:

**Add_item.php component:**

This component used to create a new item, and contains all steps required to do that starting by retrieving the last ID and all available items from database, representing the new item form view for the user to be able to enter the new item information, receiving the submitted data which contains the item information, and make some validations and security checks, retrieving the item data from the database using the ID, saving the item information by saving the submitted data to the database and representing the status message view which will show either success message when the item inserted successfully or error message when system fails saving the item. This description can be encoded into the following
Which is responsible for the following:

- Getting the next id from database (getNextIdFromDatabase)
- Getting list of available items from database (getListOfAvailableItems)
- Represent the new item form view (representNewItemFormView)
- Receiving and processing submitted data (receivingAndProcessingSubmittedData)
- Save the new item to database (saveItemToDatabase)
- Represent save status message view (representSaveStatusMessageView)

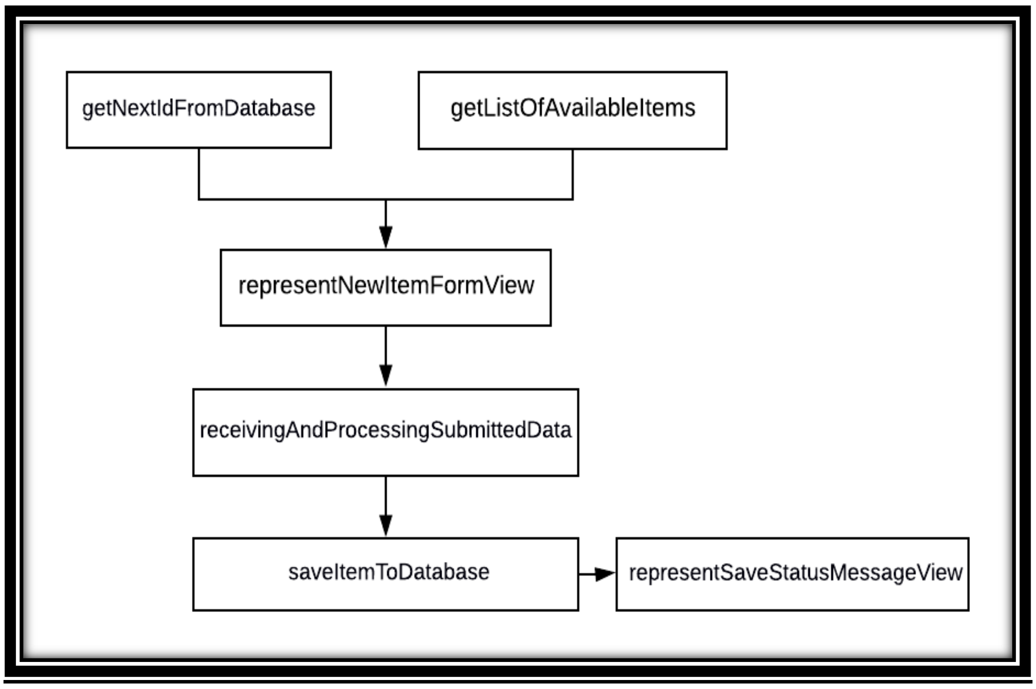


*Figure 6: add_item.php component*

**Edit_item.php component:**

This component used to update an existing item, and contains all steps required to do that starting by retrieving all available items from database, representing the list of items view for the user to be able to select which item to edit, receiving the submitted data which contains the selected item ID, and make some validations and security checks, retrieving the item data from the database using the received ID, representing the edit form view for the user to be able to edit the item information, receiving and make some validations and security checks on the submitted data which is the update information of the selected item, updating the item information by saving the submitted data to the database and representing the status message view which will show either success message when the item updated successfully or error message when system fails saving the updated item. This description can be encoded into the following:

- Get list of available items from database (getListOfAvailableItems)
- Represent items list view (representItemsListView)
- Receiving and process submitted data (receivingAndProcessSubmittedData)
- Retrieving select item data (retreivingSelectItemData)
- Representing item edit form view (representingItemEditFormView)
- Receiving and processing received edited data (receivingAndProcessingRecievedEditedData)
- Save edited item to database (saveEditedItemToDatabase)
- Represent update status message view (representUpdateStatusMessageView)

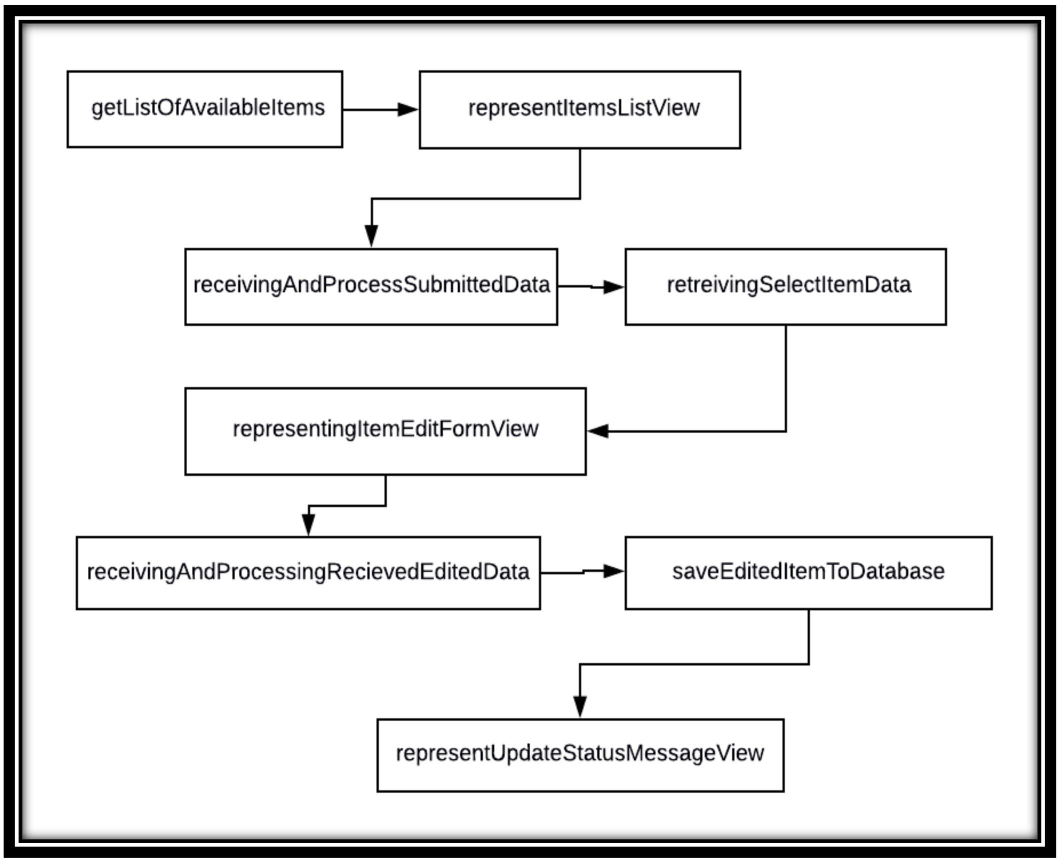

getListOfAvailableItems
representItemsListView
receivingAndProcessSubmittedData
retreivingSelectItemData
representingItemEditFormView
receivingAndProcessingRecievedEditedData
saveEditedItemToDatabase
representUpdateStatusMessageView

*Figure 7: edit_item.php component*

**Delete_item component:**

This component used to delete an item, and contains all steps required to do that starting by retrieving all available items from database, representing the list of items view for the user to be able to select which item to delete, receiving the submitted data which contains the item ID and make some validations and security checks, deleting the item from the database using the received ID and representing the status message view which will show either success message when the item deleted successfully or error message when system fails deleting the item. This description can be encoded into the following:

- Which is responsible for the following:Get list of available items (getListOfAvailableItems)
- Represent items list view (representItemsListView)
- Receiving and process submitted data (receivingAndProcessSubmittedData)
- Delete selected item from database (deleteSelectedItemFromDatabase)
- Represent delete status message view (representDeleteStatusMessageView)

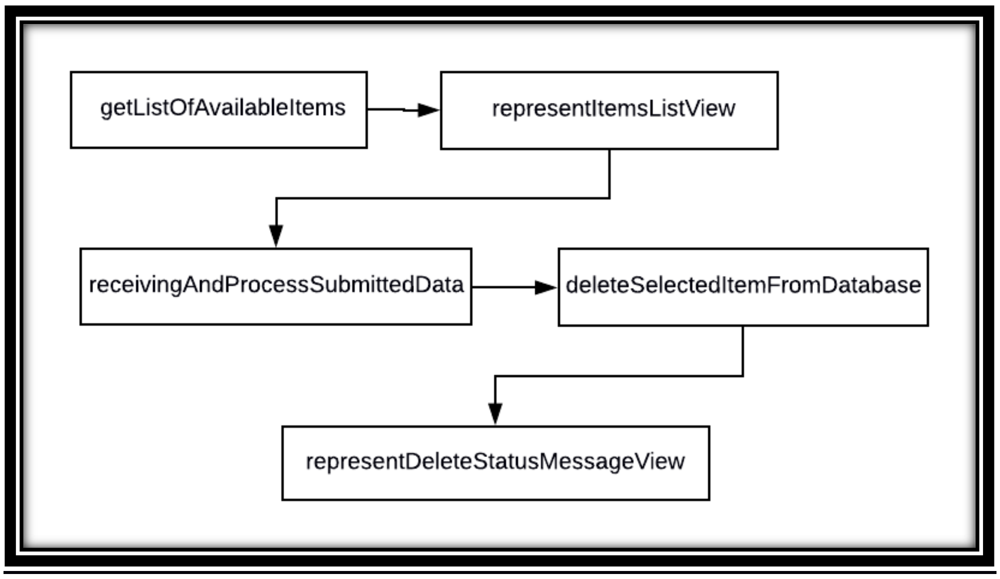

getListOfAvailableItems
representItemsListView
receivingAndProcessSubmittedData
deleteSelectedItemFromDatabase
representDeleteStatusMessageView

*Figure 8: delete_item.php component*

There are no relationships between this module components, where each component work isolated containing all the required logic to do its job as shown in figure (9).

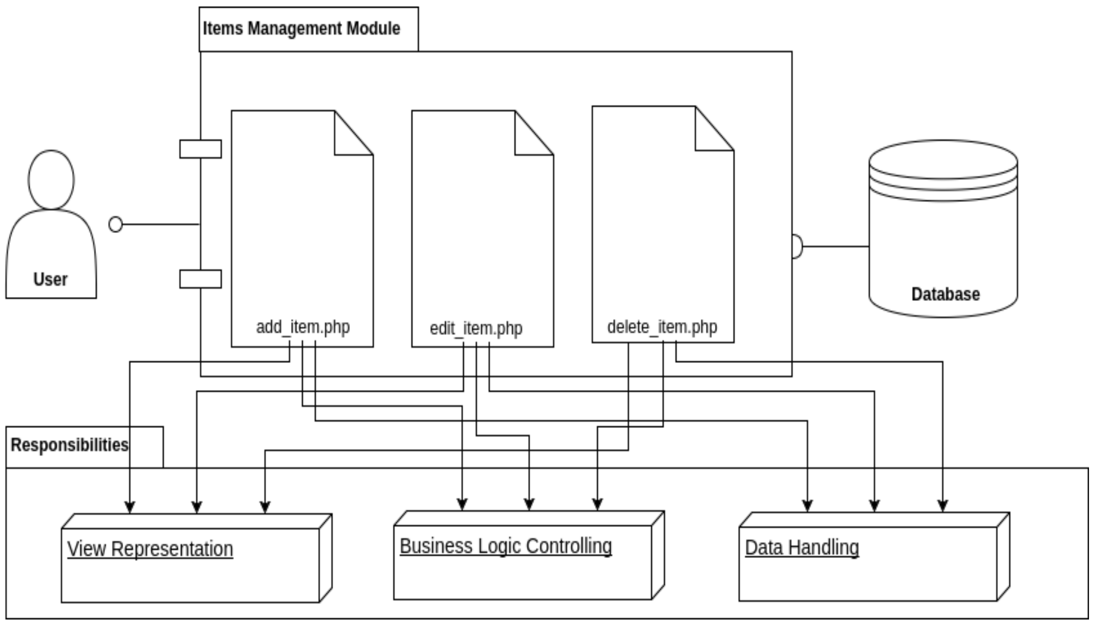


*Figure 9: the current system's architecture, Items Management Module*

### 3.6.2 Redesigned system architecture:

The system components are a combination of Classes and PHP files those grouped logically to form a module. As the (Figure 12) illustrates, that the Items Management Module consists of the following four components:

**Item component:**

This component is classified as a controller, which described in the MVC pattern in section (2.5.3). At the first time the user access the Item module this component calls the “getAll” method which exposed by the Item_model component and loading the index.php and view.php components to the user interface passing them the retrieved items. In addition, when the user enter ad submit new item information this controller will receive the submitted data and make some validations and security checks and then pass it to the “store” method that exposed by the Item_model component, also when the user update or delete an item the data will be passed to this controller and it will do the validations and security checks as usual and then call the “update” method for the update action or the “delete” method for the delete action, for modeling purposes this functionality will be encoded as following:

- Receiving and processing Submitted data (receivingAndProcessingSubmittedData)
- Passing new item data to the Item_model component (passNewItemDataToItemModel)
- Retrieve available items from Item_model component (retreiveAvailableItemsFromItemModel)
- Passing updated item data to Item_model component (passUpdatedItemDataToItemModel)
- Pass deleted item data to Item_model component (passDeletedItemDataToItemModel)
- Load index.php component and pass available items data (loadIndexAndpassAvailableItemsData)
- Load view.php component and pass available items data (loadViewAndpassAvailableItemsData)

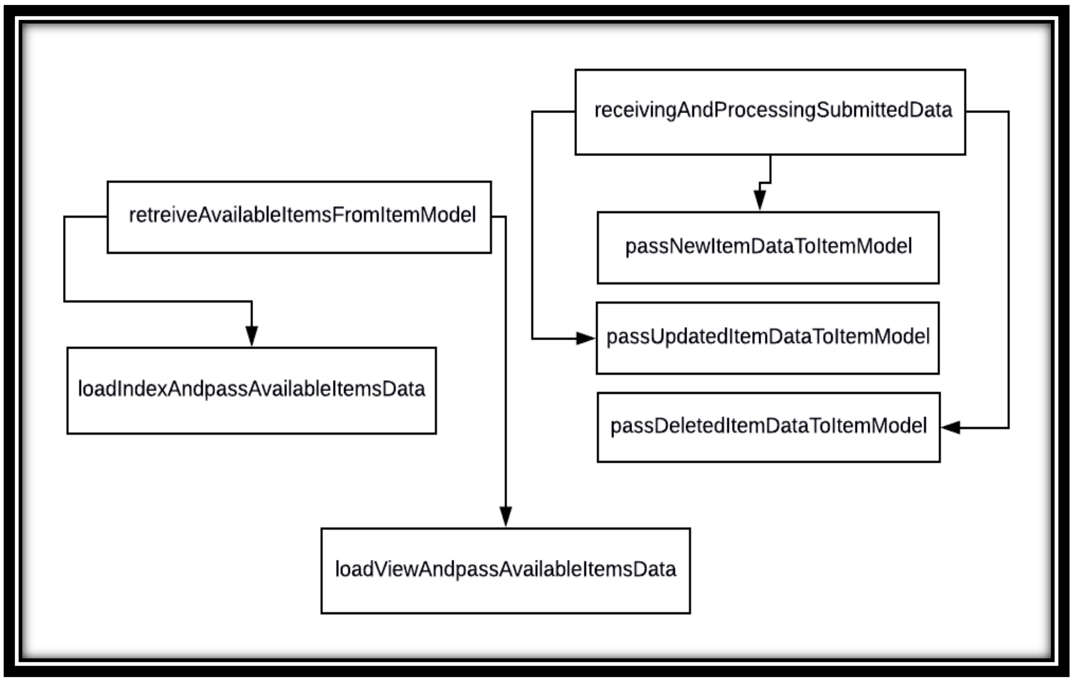

receivingAndProcessingSubmittedData
retreiveAvailableItemsFromItemModel
passNewItemDataToItemModel
passUpdatedItemDataToItemModel
loadIndexAndpassAvailableItemsData
passDeletedItemDataToItemModel
loadViewAndpassAvailableItemsData

*Figure 10: Item Component*

**Item_model component:**

This component is classified as a Model, which described in the MVC pattern in section (2.5.1), and is exposes the "store" method that receiving item data as parameters and store this data in the database, "getAll" method that retrieves and returns all items from database, "update" method that receives update parameters (ID, field name and the new value) and update the item information on the database and return either true when success or false when fails and the "delete" method that receives an ID and deleting the item with that ID from the database, for modeling purposes this functionality are encoded as following:

- Save item to database (saveItemToDatabase)
- Retrieve available items from database (rereiveAvailableItemsFromDatabase)
- Update item on database (updateItemOnDatabase)
- Delete item from database (deleteItemFromDatabase)

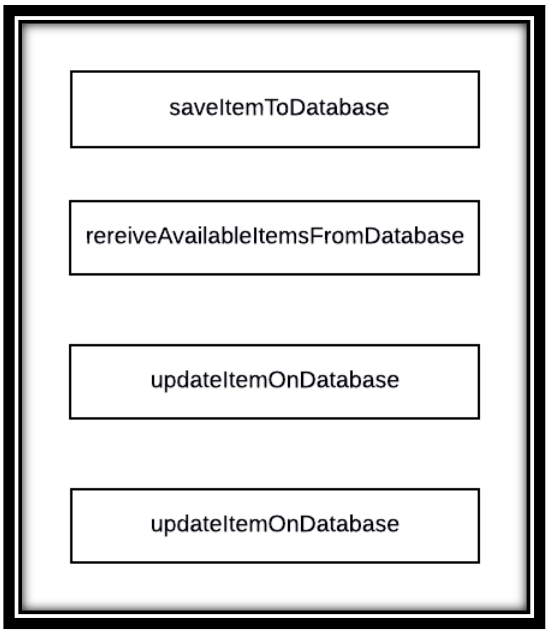


*Figure 11: Item_model component*

**Index.php component:**

This component is classified as a View in the MVC pattern, which described in the MVC pattern in section (2.5.2), and is representing the new item form where the user enter the new item information as also represent the view that display the status message of adding new item, for the modeling purposes this functionality encoded as following:

- Represent new item form view (representNewItemFormView)
- Represent item save status message view (representItemSaveStatusMessageView)

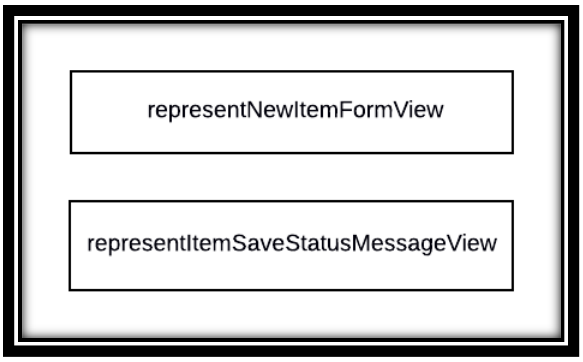


*Figure 12: index.php component*

**view.php component:**

This component is also classified as a View in the MVC pattern, which described in the MVC pattern in section (2.5.2), and is represents the following views: items list view, item edit form view, item delete form view, item update status message view and the delete item status message view, for the modeling purposes this functionality encoded as following:

- Represent items list view (representItemsListView)
- Represent item update form view (representItemUpdateFormView)
- Represent item delete form view (representItemDeleteFormView)
- Represent item update status message view (representItemUpdateStausMessageView)
- Represent item delete status message view (representItemDeleteStatusMesageiew)

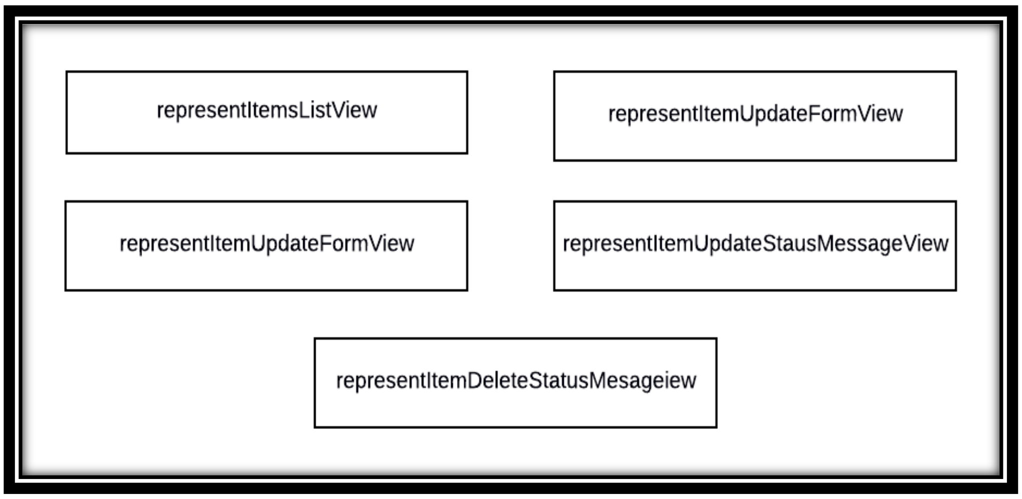

representItemsListView
representItemUpdateFormView
representItemUpdateFormView
representItemUpdateStausMessageView
representItemDeleteStatusMesageiew

*Figure 13: view.php component*

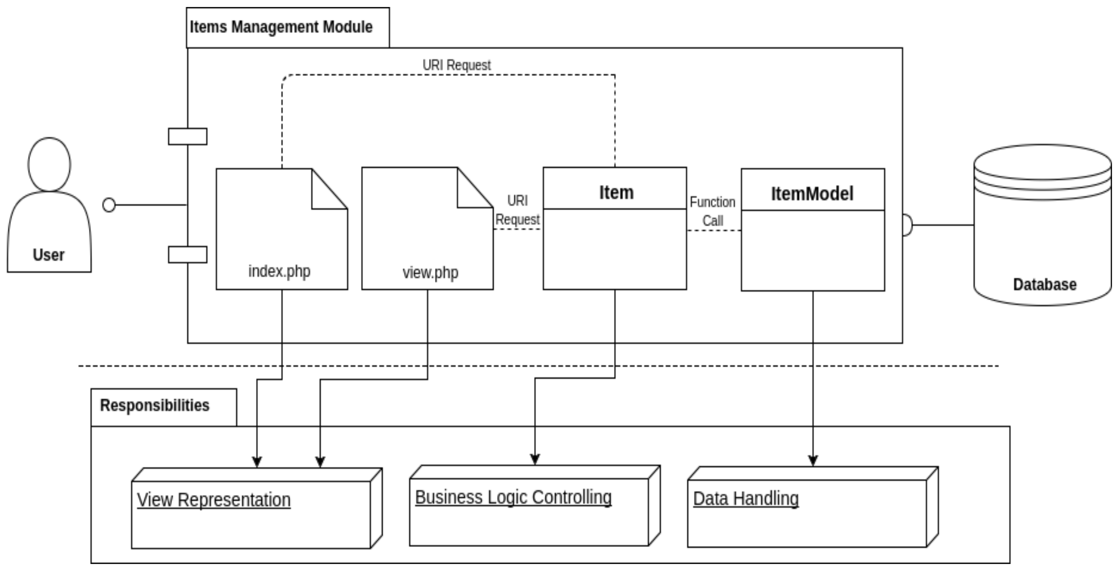


*Figure 14: the re-designed system architecture*

As show in figure (14), there are many relationships among these components, where the Item component calling the methods that exposed by Item_model component and also load views to the user interface according to the requested URI.

In this chapter we reviewed the ASAM method and selected the required techniques required to conduct the comparison of the architecture of both current ad redesigned systems. Furthermore, we described both systems' architectures.

# CHAPTER IV

# RESULTS AND DISCUSSION

Deciding which system has the highest maintainability will help us to decide if the redesigned system is increasing the maintainability. In this chapter the change scenario sets are evaluated and their effects are presented and interpreted by using simple logic rules, the decision is made according to the effects interpretation.

## 4.1 Elicited Change Scenario Sets:

Since the goal of the analysis is to compare different architectures, then the scenario elicitation technique followed is, to concentrate on scenarios that highlight differences between those candidates. A list of elicited scenarios shown in table (), after following the selected technique.

*Table 1: elicited change scenario sets*

| **Category** | **Code** | **Description** |
|---|---|---|
| DH | D1 | Adding a new field (created_by) to store the creator's user ID |
| | D2 | Adding a new field (authority_level) to store the authority level to access this item |
| | D3 | Adding a new field (last_update) to store the last (date and time) for any update occurs on the item record |
| | D4 | Changing the Database connectivity using the PDO connection rather than the legacy mysql_connect |
| UI | U1 | Change the business logic of creating a new item to consider storing the current (user_id) and the (item_authority_level), when adding a new item |

| | | |
|---|---|---|
| | U2 | Change the business logic of deleting items, to authorize or prevent users from deleting items, according to the (user_authority_level) for the current user |
| | U3 | Change the business logic of retrieving items, to filter out the list of items according to the (user_authority_level) for the current user |
| | U4 | Logging the current event |
| VR | V1 | Add a new field to the New Item form. to select the authority level required to access the item under creation |
| | V2 | Add a validation, to prevent the user from submitting Null Values |
| | V3 | Adding a confirmation message for updating item information |
| | V4 | Change the response message to be able to handle the authorization messages |

## 4.2 **Current system results:**

*Table IV-2: current system – scenario evaluation*

| | | add_item.php | edit_item.php | delete_item.php |
|---|---|---|---|---|
| Data Handling | D1 | True | False | False |
| | D2 | True | True | False |
| | D3 | True | True | False |
| | D4 | True | True | True |
| User Interaction Control | U1 | True | False | False |
| | U2 | False | False | True |
| | U3 | True | True | True |
| | U4 | True | True | True |
| View Representation | V1 | True | False | False |
| | V2 | True | True | True |
| | V3 | False | True | False |
| | V4 | True | True | True |

*Table IV-3: Current System - Data Handling Responsibility truth table*

| | D1 | D2 | D3 | D4 | DH = (D1 + D2 + D3 + D4) |
|---|---|---|---|---|---|
| add_item.php | T | T | T | T | T |
| edit_item.php | F | T | T | T | T |

| | | | | | |
|---|---|---|---|---|---|
| delete_item.php | F | F | F | T | T |

In the previous table, the add_item.php component was changed in D1, D2, D3 and D4, when applied the logical OR, its final state was (Changed for the reason DH). The edit_item.php component was changed in D2, D3 and D4, when applied the logical OR its final state was (Changed for the reason DH). The delete_item.php component was changed in D4, when applied the logical OR its final state was (Changed for the reason DH).

*Table IV-4: Current System - User Interaction Control Responsibility truth table*

| | U1 | U2 | U3 | U4 | UI = (U1 + U2 + U3 + U4) |
|---|---|---|---|---|---|
| add_item.php | T | F | T | T | T |
| edit_item.php | F | F | T | T | T |
| delete_item.php | F | T | T | T | T |

In the previous results the add_item.php component was changed in U1, U3 and U4, when applied the logical OR its final state was (Changed for the Reason UI). The edit_item.php component was changed in U3 and U4, when applied the logical OR its final state was (Changed for the Reason UI). The delete_item.php component was also changed in U2, U3 and U4, when applied the logical OR the final state was (Changed for the Reason UI).

*Table IV-5: Current System - View Representation Responsibility truth table*

|  | V1 | V2 | V3 | V4 | VR = (V1 + V2 + V3 + V4) |
|---|---|---|---|---|---|
| add_item.php | T | T | F | T | T |
| edit_item.php | F | T | T | T | T |
| delete_item.php | F | T | F | T | T |

In the previous results the add_item.php component was changed in V1, V2 and V4, when applied the logical OR its final state was (Changed for the Reason VR). The edit_item.php component was changed in V2, V3 and V4, when applied the logical OR its final state was (Changed for the Reason VR). The delete_item.php component was changed in V2 and V4, when applied the logical OR its final state was (Changed for the reason VR).

*Table IV-6: Current System - S.R.P. Violations truth table*

|  | DH | UI | VR | SRP-V1 (DH.UI) | SRP-V2 (DH.VR) | SRP-V3 (UI.VR) |
|---|---|---|---|---|---|---|
| add_item.php | T | T | T | T | T | T |
| edit_item.php | T | T | T | T | T | T |
| delete_item.php | T | T | T | T | T | T |

In the previous table we find that the add_item.php component is violating the SRP principal in SRP-V1, SRP-V2 and SRP-V3. And the edit_item.php component is violating the SRP principal in SRP-V1, SRP-V2 and SRP-V3. As also the component delete_item.php component is violating the SRP principal in SRP-V1, SRP-V2 and SRP-V3.

## 4.3 **Re-designed system results:**

*Table IV-7: redesigned System - scenario evaluation*

| | | Items | Items_model | index.php | view.php |
|---|---|---|---|---|---|
| Data Handling | D1 | False | True | False | False |
| | D2 | False | True | False | False |
| | D3 | False | True | False | False |
| | D4 | False | True | False | False |
| User Interaction Control | U1 | True | False | False | False |
| | U2 | True | False | False | False |
| | U3 | True | False | False | False |
| | U4 | True | False | False | False |
| View Representation | V1 | False | False | True | True |
| | V2 | False | False | True | True |
| | V3 | False | False | False | True |
| | V4 | False | False | True | True |

*Table IV-8: redesigned System - Data Handling Responsibility truth table*

|  | D1 | D2 | D3 | D4 | DH = (D1 + D2 + D3 + D4) |
|---|---|---|---|---|---|
| Items | F | F | F | F | F |
| Items_model | T | T | T | T | T |
| Index.php | F | F | F | F | F |
| View.php | F | F | F | F | F |

In the previous table the ItemsModel component was changed in D1, D2, D3 and D4, its final state was (changed for the reason DH). The components Item, Index.php and View.php not effected by those change scenarios.

*Table IV-9: redesigned System - User Interaction Control Responsibility truth table*

|  | U1 | U2 | U3 | U4 | UI = (U1 + U2 + U3 + U4) |
|---|---|---|---|---|---|
| Items | T | T | T | T | T |
| Items_model | F | F | F | F | F |
| Index.php | F | F | F | F | F |
| View.php | F | F | F | F | F |

In the previous table the Item component was changed in U1, U2, U3 and U4, its final state was (changed for the reason UI). The components Item_model, Index.php and View.php not effected by those change scenarios.

*Table IV-10: redesigned System - View Representation Responsibility truth table*

| | V1 | V2 | V3 | V4 | VR = (V1 + V2 + V3 + V4) |
|---|---|---|---|---|---|
| Items | F | F | F | F | F |
| Items_model | F | F | F | F | F |
| Index.php | T | T | F | T | T |
| View.php | T | T | T | T | T |

In the previous table Index.php component was changed in V1, D2 and D4, its final state was (changed for the reason VR). And the View.php component was changed in V1, V2, V3 and V4, its final state is (changed for the reason VR). The components Item and Item_model not effected by those change scenarios.

*Table IV-11: redesigned System - S.R.P. Violations truth table*

| | DH | UI | VR | SRP-V1 (DH.UI) | SRP-V2 (DH.VR) | SRP-V3 (UI.VR) |
|---|---|---|---|---|---|---|
| Items | F | T | F | F | F | F |
| Items_model | T | F | F | F | F | F |

| | | | | | | |
|---|---|---|---|---|---|---|
| Index.php | F | F | T | F | F | F |
| View.php | F | F | T | F | F | F |

In the previous table, we found that all components are following the SRP principal with no violations.

## 4.4 **Current system vs Re-designed System:**

*Table IV-12: Current system vs Re-designed System*

| Results Expression | Current System | Re-designed System |
|---|---|---|
| S.R.P-V1 | 3 | 0 |
| S.R.P-V1 | 3 | 0 |
| S.R.P-V1 | 3 | 0 |
| Total S.R.P Violations | 9 | 0 |

In this chapter all change scenario sets evaluated against candidate architectures, and any SRP violations countered in each component was ranked, this resulted in 9 violations in the current system and 0 violations in the redesigned system. Since the current system has violations more than the redesigned system, then the redesigned system is evaluated as the system with the highest maintainability.

# CHAPTER V

# CONCLUSIONS AND RECOMMENDATIONS

## 5.1 Conclusion:

This study concludes that, successful implementation of MVC design pattern made each component of the software domain focused and distributed the development teams into three domains based on their skills (frontend developers, backend developers and database developers). Hence, the developers work closely to their skills and experience to produce a high quality components. Furthermore, the separation of the software components based on developers' skills that required to develop making the parallel development easier and more efficient, which increases the system maintainability.

The researcher found that "Laravel" framework is the most efficient PHP framework, that fully providing stable scaffolding tools to generate M.V.C applications out of the box.

## 5.2 Recommendations:

This study have increased one of the most important software attributes (Maintainability), when the system also after the fully adoption will experience a huge traffic and handling a great load of data, where the system Scalability will need to be enhanced. We encourage more enhancement in the system scalability.

## REFERENCES

Abdullah, F., 2017. *Evaluating Impact of Design Patterns on Software Maintainability and Performance,* s.l.: s.n.

Bass, L., Clements, P. & Kazman, R., 2003. *Software Architecture in Practice, Second Edition.* s.l.:Addison Wesley.

Bengtsson, P., Nico, L., Jan , B. & Hans , v. V., 2000. *Analyzing Software Architectures for Modifiability,* s.l.: s.n.

bishop, j., 2007. *c# 3.0 design patterns: use the power of c# 3.0 to solve real-world problems.* s.l.:oreilly.

Brooks & Frederick, 1987. *No Silver Bullet – Essence and Accident in Software Engineering,* s.l.: s.n.

Burbeck, S., 1992. *How to use Model-View-Controller (MVC).* s.l.:s.n.

Buschmann, F. et al., 1996. *pattern oriented software architecture: A system of patterns.* s.l.:Chichester Wiley: Addison-Wesley.

Chidamber, S. & Kemerer, C., 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering,* 20(6).

Christoph, S., Felix, B. & Chris, V., 2003. *SACAM: The Software Architecture Comparison Analysis Method,* s.l.: s.n.

Halstead & H., M., 1977. *Elements of Software Science. Amsterdam: Elsevier North-Holland.* s.l.:s.n.

Hazzan, O. & Hadar, I., 2008. Why and how can human-related measures support software development processes?. *The Journal of Systems and Software.*

Henry, S. & Kafura, D., 1981. Software Structure Metrics Based on Information Flow. *IEEE Trans. Software Eng..*

Hoffman, F., 2001. *Architectural software patterns and maintainability: A case study.* s.l.:s.n.

Kazman, R., Len, B., Gregory, A. & Mike, W., 2007. *SAAM: A Method for Analyzing the Properties of Software Architectures,* s.l.: Texas Instruments.

Martin, R. C., 2005. *Agile Software Development, Principles, Patterns, and Practices.* s.l.:s.n.

McCabe, 1976. A Complexity Measure. *IEEE Transactions on Software Engineering.*

Pressman, R., 1997. *Software Engineering. A Practitioner's Approach.* s.l.:McGraw Hill.

R.E, W., n.d. *Task Complexity: definition of the construct, organisational behaviour and human decision processes.* s.l.:s.n.

Reenskaug, T. & James, C., 2009. *The DCI Arcitecture: A New Vision of Object-Orinted Programming.* [Online]
Available at: http://www.artima.com/articales/dci_vision.html
[Accessed 3 11 2017].

Rick, K., Mark, K. & Paul, C., 2000. *ATAM: Method for Architecture Evaluation,* s.l.: s.n.

Roads, B. D., 2003. *Domain-Driven Design.* s.l.:s.n.

Shao, J. & Yingxu Wang, APRIL 2003 . *A new measure of software complexity based on cognitive weights,* s.l.: s.n.

Smith, S., 2018. *Overview of ASP.NET Core MVC.* [Online]
Available at: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview
[Accessed 23 2 2018].

Smith, S., 2018. *Overview of ASP.NET Core MVC.* [Online]
Available at: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnet
[Accessed 2 11 2017].

Stavrinoudis, D., Michalis Xenos & Dimitris Christodoulakis, August 1999. *RELATION BETWEEN SOFTWARE METRICS AND MAINTAINABILITY,* s.l.: s.n.

Tarvainen, 2008. Adaptability Evaluation at Software Architecture Level. *The Open Software Engineering Journal,* pp. 1-30.

Tran-Cao, D., Ghislain Lévesque & Jean-Guy Meunier , n.d. *Software Functional Complexity Measurement with the Task Complexity Approach.* s.l.:s.n.

Wiki, 2014. *Model View Controller History.* [Online]
Available at: wiki.c2.com/?ModelViewControllerHistory
[Accessed 21 2 2018].

Wikipedia, 2014. *SOLID.* [Online]
Available at: https://www.en.m.wikipedia.org/wiki/SOLID
[Accessed 12 11 2017].

Züllighoven H., L. C. B. M., 2006. Software Architecture Analysis and Evaluation. *International Conference on the Quality of Software Architectures.*

Zuse, H., 1991. *Software Complexity Measures and Methods.* s.l.:s.n.

## Appendix A: The Current System (Items Management Module):

### **A-1:** The “add_item.php” component:

```
<?php
session_start();include("connect.php");
include("user_style.php");
$name=$_POST[name];
$data=mysql_query("
          select max(id) from items
");
$no=mysql_fetch_array($data);
$no=$no[0]+1;
?>
<html>
<head>
<title>last</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1256">
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0"
marginheight="0">
<form action='save_item.php' method='post'>
<table align='center' border='1' dir="rtl">
<tr>
<?php
echo"
<th>رقم البند<th><input type='button' value='$no'><tr>
<input type='hidden' name='no' value='$no'><tr>
";
$data=mysql_query("
select * from items
");
echo"<th>البند الرئيسي<th><select name='upper'>
<option value='0'>بند رئيسي</option>
";
while($info=mysql_fetch_array($data))
{
echo"<option value='$info[id]'>".$info[name]."</option>";
}
echo"</select><tr>";
?>
<th>اسم البند<th><input type='text' name='name'><tr>
<th>الكود<th><input type='text' name='code'>
</table><br><input type='submit' value='إضافة'></td>
</body>
</html>
```

## **A-2:** The “edit_item.php” component**:**

```
<?php
session_start();include("../connect.php");include("../user_style.php");$step=$_GET[step];
?>
<html>
<head>
<title>Edit Item</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1256">
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0
<?php
echo"<center>";if(!isset($step)){$data=mysql_query("select * from items");echo"
<form action='item_edit.php?step=1' method='post' name='myform' dir='rtl'>
<select name='id' id='unit'>
<option value='0'>أSelect Item</optio>";
while($info=mysql_fetch_array($data))
{
echo "<option value='$info[id]'
style='backgroundcolor:black;color:white'>".$info[name]."</option>";
}
echo"<input type='button' value=”Continue”  onclick='edit()'>";
}else{
switch($step){
case 1:
$id=$_POST[id];
$data=mysql_query("select * from items where id='$id'");
$info=mysql_fetch_array($data);
echo"<form action='item_edit.php?step=2' method='post'>
<input type='hidden' name='id' value='$id'>
<table dir='rtl' align='center' border='1'>
<tr><th>اسم البند<th><input type='text' name='name' value='$info[name]'>
<tr><th>الكود<th><input type='text' name='code' value='$info[code]'>
</table>
<input type='submit' value='حفظ التعديلات'>
";
break;
case 2:
$id=$_POST[id];
$name=$_POST[name];
$code=$_POST[code];
$sql=mysql_query("update items set name='$name',code='$code' where id='$id'");
if($sql)
{
$access_date=date("Y-m-d");
$access_time=date("h-i-s");
mysql_query("
insert into events_file values('','$_SESSION[myuser]',11,'$id','$access_date','$access_time')
");
echo"<p dir='rtl' align='center'><font color='green'><b> تم حفظ التعديلات
بنجاح</b></font></p>";
}
else
{
echo"
<p dir='rtl' align='center'>
<font color='red'><b>لم يتم حفظ التعديلات</b></font></p>";
}
}//switch
}// if second step or more
?>
</body>
</html>
```

**A-3:** The “delete_item.php” component:

```
<?php
session_start();
include("../connect.php");
include("../user_style.php");
$step=$_GET[step];?>
<html>
<head>
<title>Edit Item</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1256">
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0"
marginheight="0">
<?php
echo"<center>";
if(!isset($step)){
$data=mysql_query("select * from items");
echo"<form action='item_del.php?step=1' method='post' name='myform' dir='rtl'>
<select name='id' id='unit'>
<option value='0'>Select Item</optio>";
while($info=mysql_fetch_array($data)){
echo "<option value='$info[id]' style='background-
color:black;color:white'>".$info[name]."</option>";
}
echo"<input type='button' value='حذف'  onclick='del()'>";
}else
{
$id=$_POST[id];
$sql=mysql_query("delete from items where id='$id'");
if($sql){
$access_date=date("Y-m-d");
$access_time=date("h-i-s");
mysql_query("insert into events_file
values('','$_SESSION[myuser]',16,'$id','$access_date','$access_time')");
echo"<p dir='rtl' align='center'><font color='green'><b>تم الحذف بنجاح</b></font></p>";
}else{
echo"<p dir='rtl' align='center'><font color='red'><b>لم يتم الحذف</b></font></p>";
}}// if second step or more
?>
</body></html>
```

## Appendix B: The redesigned System (Items Management Module):

**B-1:** The "ItemModel" component:

```
<?php
class ItemModel extends CI_Model {
	Public $name;
	Public $code;
	Public $parent;
	private $tableName;

	public function __construct()
	{
		parent::__construct();
		$this->tableName = "items";
	}

	public function create($name, $code, $parent) {
		$this->name = $name;
		$this->code = $code;
		$this->parent = $parent;
		$this->status = 0;

		if($this->db->insert($this->tableName,$this)) {
			return $this->get($this->db->insert_id());
		} else {
			return false;
		}
	}

	public function update($changed_id, $field_name, $new_value, $data_type) {
		if($data_type == 'varchar') {
			$query = "update ".$this->tableName." set ".$field_name." = 
'".$new_value."' where id=".$changed_id;
		} else {
			$query = "update ".$this->tableName." set ".$field_name." = 
".$new_value." where id=".$changed_id;
		}
		return $this->db->simple_query($query);
	}

	public function delete($id) {
		$query = "delete from ".$this->tableName." where id=".$id;
		return $this->db->simple_query($query);
	}

	public function getAll() {
		return $this->db->get($this->tableName)->result();
	}

	public function getById($id) {
		$this->db->where('id',$id);
```

```
                return $this->db->get($this->tableName)->row();
        }
}
?>
```

**B-2:** The “Item” component**:**

```
<?php
class Item extends CI_Controller {

        private $path;

        public function __construct()
        {
                parent::__construct();
        }

        public function index()
        {
    $this->load->model('ItemModel','self');
    $data['items'] = $this->self->getAll();
                $this->load->view($this->index, $data);
        }

        public function create()
        {
                $this->load->model('ItemModel','self');

                $name = $this->input->post('name');
                $code = $this->input->post('code');
                $parent = $this->input->post('parent');

                $newId = $this->self->create($name, $code, $parent);

                if(is_object($newId))
                {
                        $parms = Array('id'=>$newId->id);
                        $response = Array(
         'success' => true,
         'message' => 'Item created successfully!'
      );
                }
                else
                {
                        $response = Array(
         'success' => false,
         'message' => 'Item was not created'
      );
                }
                print_r(json_encode($response));

        }
```

```
          public function update()
          {
                    $changed_id = $this->input->post('changed_id');
                    $field_name = $this->input->post('field_name');
                    $new_value = $this->input->post('new_value');
                    $data_type = $this->input->post('data_type');

                    $this->load->model('ItemModel','self');
                    $result = $this->self->update($changed_id, $field_name, $new_value,
$data_type);

                    if($result) {
                              $response = Array(
           'success' => true,
           'message' => 'Item updated successfully'
        );
                    } else {
                              $response = Array(
           'success' => false,
           'message' => 'Item was not updated'
        );
                    }
                    print_r(json_encode($response));
          }

          public function delete()
          {
                    $id = $this->input->post('id');

                    $this->load->model('ItemModel','self');
                    $result = $this->self->delete($id);

                    $items = $this->self->getAll();

                    if($result == 1)
                    {
                              $response = Array(
                                                  'success' => 'true',
                                                  'message' => 'Item deleted successfully',
                                                  'items' => $items
                                                  );
                    }
                    else
                    {
                              $response = Array(
                                                  'success' => false,
                                                  'message' => 'Item was not updated'
                                                  );
                    }
                    print_r(json_encode($response));
          }
}
```

**B-3:** The “index.php” component:

```
<?php
   $itemsList = "";
   foreach($items as $item) {
      $itemsList .= "<option value='".$item->id."'>".$item->name."</option>";
   }
?>
<html>
<meta charset="utf-8"/>
<head>
<script language="javascript">
$( document ).ready(function () {
            loadView('../items/getAll');
});
</script>
</head>
<body>
<div class="panel panel-primary" style="position:fixed;left:0%;right:0%;top:33px;">
<div class="panel-heading" dir="rtl">
            <h3 class="panel-title header-font" align="right">
            <i class="glyphicon glyphicon-paperclip"></i> Items
            <i         class='glyphicon         glyphicon-remove         alone'         style="float:left;"
onclick="$('#generalViewer').hide();" title="close"></i>
            </h3>
 </div>
            <div                  class="panel-body"                                     style="background-
color:rgba(162,162,211,0.8);overflow:auto;">
<div style="text-align:right;position:relative;width:100%;">
<div       id="form"       style="display:inline-block;width:57%;padding:0px;vertical-align:top;"
dir="rtl">
<div class="panel panel-primary" style="width:100%;">
             <div class="panel-heading" dir="rtl">
                        <h3 class="panel-title header-font"><i onclick="loadView('../items/getAll');"
class="glyphicon glyphicon-list-alt"></i>
                        Current Items</h3>
             </div>
             <div class="panel-body" id="tank"  style="background-color:rgba(0,0,0,0.2);">
                        <div>
                                   <div                       id="viewLoader"                       dir="rtl"
style="height:400px;overflow:auto"></div>
                        </div>
             </div>
            </div>
</div>
<div id="form" style="display:inline-block;width:40%;padding:0px" dir="rtl">
<div class="panel panel-primary" style="width:100%;">
            <div class="panel-heading" dir="rtl">
                        <h3 class="panel-title header-font">
                        <i class="glyphicon glyphicon-plus-sign"></i> New Item</h3>
             </div>
```

```
        <div class="panel-body" id="accountFormHolder" style="background-color:rgba(0,0,0,0.2);">

<form>
<div class="form-group">
  <input type="text" class="form-control" id="code" name="code" placeholder="Item Code">
</div>
<div class="form-group">
  <input type="text" class="form-control" id="name" name="name" placeholder="Item Name">
</div>
<div class="form-group">
  <select class="form-control" id="parent" name="parent" placeholder="Parent Item">
    <option value="0">Select Parent Item</option>
    <?php
      echo $itemsList;
    ?>
  </select>
</div>
</form>

<div dir="rtl">
  <div class="form-group">
  <button id="add" class="btn btn-default button-font" onclick="create('../items/create','','');">Save</button>
  <button id="res" class="btn btn-warning button-font" onclick="cancelCreation();">Cancel</button>
  </div>
</div>
<div style="background-color:#ccc;" id="status"></div>
</div>
</div>
</div>
<div style="width:96%;text-align:right;margin-left:4%;" dir="rtl">
</div>
</div>
</div>
</div>
</div>
```

**B-4:** The “view.php” component**:**

```
<?php
$path = "../items/update";
$deletePath = "../items/delete";
?>
<table class="table table-hover editableTable" width="80%" align="center" dir="rtl">
        <tr style="background-color:#cadfdf;">
  <td align="right">#
  <td align="right">Code
  <td align="right">Name
```

```
   <td align="right">Parent
   <td align="center" width="120px" colspan="2">Action
          </tr>
<?php
foreach($objects as $object) {
   $itemsList = "";
   foreach($items as $item) {
      if($item->id == $object->id) {
         $itemsList .="<option value='".$object->id."' selected>".$object->name."</option>";
      } else {
         $itemsList .="<option value='".$object->id."'>".$object->name."</option>";
      }
   }
          echo "
          <tr id='".$object->id."'>

                    <td align='right'>".$object->id."</td>
                    <td align='right' ondblclick=\"editableTd(this,'".$path."','code','".$object-
>id."','varchar')\">".$object->code."</td>
                    <td align='right' ondblclick=\"editableTd(this,'".$path."','name','".$object-
>id."','varchar')\">".$object->name."</td>

          <td align='right'>
                    <select onchange=\"updateCell(this,'".$path."','parent','".$object-
>id."','int','".$object->parent."')\">";
      <?=$itemList?>
          echo"
                    </select>
                    </td>

                    <td width='60px' align='center'>
                              <i class='glyphicon glyphicon-remove alone'
onclick=\"deleteObject('".$deletePath."',".$object->id.")\" title='Delete'></i>
                    </td>
          ";
}
?>
</table>
```