

## **CHAPTER 2**

### **THEORITICAL FOUNDATION**

In this section, the relevant theories is summarized and presented comprehensively and in accordance with the problem. The theories which support the solution of the problem can be referenced from the latest research in the field or from textbook and other references.

#### **2.1 Performance Management System**

According to Susan M. Heathfield [5], performance management is the process of creating a work environment or setting in which people are enabled to perform to the best of their abilities. Performance management is a whole work system as needed that will occur as soon as a job is defined. It ends when an employee leaves your organization. Furthermore, Heathfield explains that performance management should include the following actions:

- Develop clear job descriptions.
- Select appropriate people with an appropriate selection process.
- Negotiate requirements and accomplishment-based performance standards, outcomes, and measures.
- Provide effective orientation, education, and training.
- Provide on-going coaching and feedback.
- Conduct quarterly performance development discussions.
- Design effective compensation and recognition systems that reward people for their contributions.

- Provide promotional/career development opportunities for staff.
- Assist with exit interviews to understand WHY valued employees leave the organization

## **2.2 Balanced Scorecard**

According to Kaplan and Norton [2], the balanced scorecard is a strategic planning and management system that is used extensively in business and industry, government, and nonprofit organizations worldwide to align business activities to the vision and strategy of the organization, improve internal and external communications, and monitor organization performance against strategic goals. Old balance scorecard only retains from financial perspective which is past record and directly involved to revenue of the organization.

The following figure is proposed by Kaplan and Norton; the balanced scorecard suggests that we view the organization from four perspectives, and to develop metrics, collect data and analyze it relative to each of these perspectives:

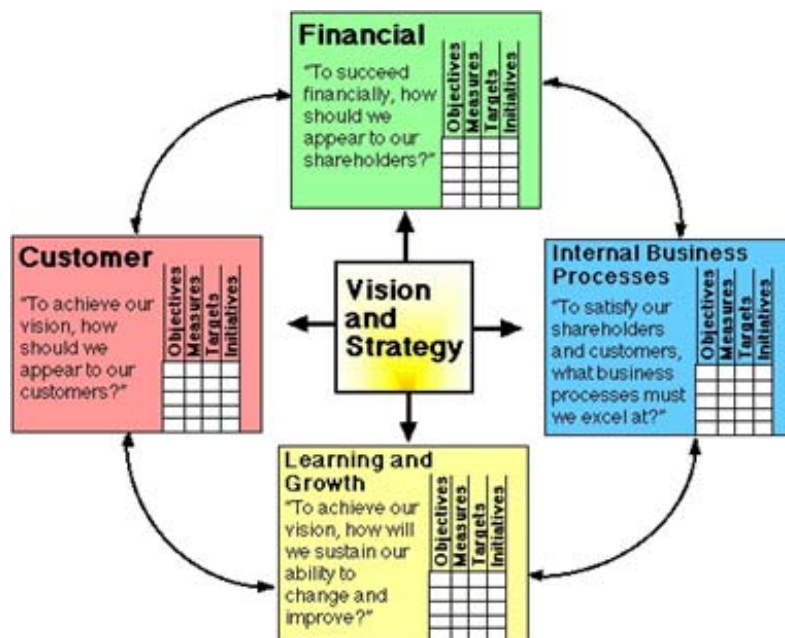


Figure 1: Balance scorecard

1. The learning and growth perspective.

This perspective includes employee training and corporate cultural attitudes related to both individual and corporate self-improvement.

2. The business process perspective.

Metrics based on this perspective allow the managers to know how well their business is running, and whether its products and services conform to customer requirements (the mission).

3. The customer perspective.

Customer focus and satisfaction will be important factors for business. The matrices will be based in terms of kinds of customers and the kinds of processes for which we are providing a product or service to those customer groups

4. The financial perspective.

Historical data of finance will always take into account for managers for performance. The point is that the current emphasis on financials leads to the "unbalanced" situation with regard to other perspectives.

### **2.3 KPI Table**

According to F. John Reh from About.com [1], Key Performance Indicators are quantifiable measurements, agreed to beforehand, that reflect the critical success factors of an organization. KPI for any organization is different depending on their focus. A Key Performance Indicator for a social service organization might be number of clients assisted during the year. In KPI system, each performance for each staff must be able to be measured. In sales, for instance, we need to address the performance by units sold or dollars gained per month.

Key performance indicators must reflect the organizational goals. For instance, a company that has one of its goals "to be the most profitable company in our industry" will have key performance indicators that measure profit and related fiscal measures. Furthermore, KPI need to be quantifiable which means that any value must be able to be defined and measured. For example, "generate more repeat customers" will not work since without some way to distinguish between new and repeat customers. Also, KPI must be the key to organizational success. There are various factors that affects an organization and only few are essential for the organization to reach its goal.

## 2.4 Database

According to Connolly and Begg[6], database is shared collection of logically related data, and a description of this data, signed to meet the information needs of an organization. The database is a single, possibly large repository of data that can be used simultaneously by many departments and users. Instead of disconnected files with redundant data, all data items are integrated with a minimum amount of duplication. The database is no longer owned by one department but is shared corporate resource. The database hold not only the organization's operational but also a description of this data. For this reason, a database is also defined as a self-describing collection of integrated records. The description of the data is known as the system catalog or data dictionary. Because all the database data are logically related, it consists of 3 main elements as information providers which are:

1. Entity (referred as a distinct object like name, place, things, concept used in organization that is represented in database).
2. Entity relationship (The association between one entity to another or more than one entity).
3. Attributes (An information or property of some aspect of entity that people wish to record).

### 2.4.1 Structured Query Language

Structured Query Language (SQL) is a standard of 4<sup>th</sup> generation database computer language (4GL) designed for management and retrieval of database in DBMS (Database management system software). SQL model was firstly introduced by Dr. F.E Codd in his research paper "A Relational Model of Data for Large Shared Data

Banks” in 1970 which later on was developed by Donald D. Chamberlin and Raymond F. Boyce from IBM Company with the name of SEQUEL (Structured English Query Language) and later on changed into SQL [9]. There are four types of standard syntaxes need to be followed for SQL programming language:

### 1. Queries

This is the most commonly used SQL syntax which performs a declarative of SELECT statement to retrieve the data (columns and rows) in database.

While often considered as part of Data Manipulation Language (DML), the standard query is considered separate from DML because it has no persistent effects on the data stored in a database. With queries, user can specify which field or column of table in database need to be retrieved or using a “wildcard” (\*) asterisk to retrieve all columns inside tables. It also allows user to retrieve and filter the result set of database in any manners depend on the declaration of the query in which make the query is the most complex one. Numerical process of column data can also be performed like **SUM**, **COUNT**, **MAX**, **MIN**, or **AVG** if the data is numeric. Logic of data retrieval can be filtered with additional clauses of optional query statement after SELECT which are:

- **FROM**

This clause determine from which table the data retrieval will take place. This clause optionally use **JOIN** clause to join the same related table based on user-specific criteria.

- **GROUP BY**

This clause allows user to perform aggregated function (eliminate duplicated rows in result set) and combine or group row with the same related value into elements of a smaller set of rows.

– **WHERE**

This clause allows user to add comparison predicate or additional logic to specific column use to restrict or filter the result set returned by query.

– **HAVING**

Similar with WHERE clause where this clause allows user to add user specified conditional logic to filter the result set, but it is done after the GROUP BY is applied to result set.

– **ORDER BY**

This clause function is to sort or rearrange the data of user specified column in result set and the order type that they should be sort either descending (**DESC**) or ascending (**ASC**).

## 2. **DDL (Data Definition Language)**

SQL DDL is a SQL process to construct and control of new table with its associated elements including rows, columns, tables, indexes, and database specifics such as file locations. The syntaxes in SQL DDL are CREATE, ALTER, RENAME, TRUNCATE, and DROP statement.

## 3. **DML (Data Manipulation Language)**

SQL DML takes parts to control the insertion, updates and removal of table data inside database. The SQL DML syntaxes are INSERT, UPDATE and DELETE. The use of DML only has a persistence effect to the data inside table, not the table's information.

#### 4. DCL (Data Control Language)

The last category of SQL syntax is DCL known as Data Control Language whose purpose is controlling the data authorization specified by user. With DCL, user are able to control the user permission and restriction of certain access or modification of data inside database. It involves two main syntaxes,

### 2.4.2 Database Management System

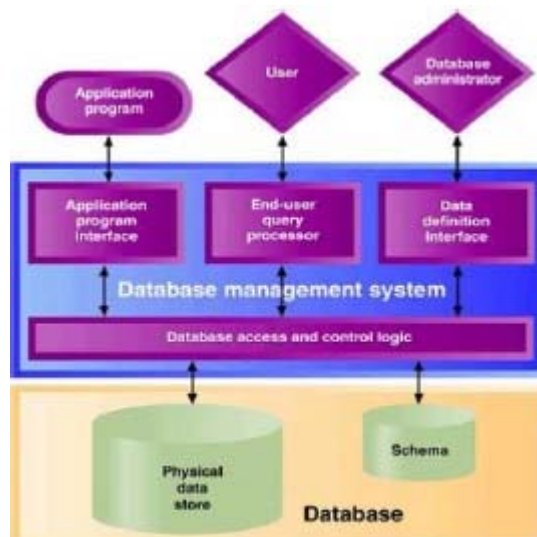


Figure 2: Database Management System Architecture

Database Management System (DBMS) is a software that enables users to define, create, maintain, and control access to the database. The DBMS is the software that interacts with the users' application programs and the database. DBMS is also designed as a set of complex computer application that controls the organization, storage, management and retrieval of data inside database. As defined by Dr. F.E



Codd (1982), the full-scale DBMS should have the capability to provide services to handle the querying, DDL (Data Definition Language), DML (Data Modification Language), and DCL (Data Control Language) processes combined with the capability of database concurrency control (to ensure that database is updated correctly when multiple user updating database concurrently), integrity system (maintain the consistency of stored data) and database backup and restore if the case of data conflict.

### **2.4.3 MySql**

It is one of well known open-source, multithreaded and multi-users DBMS (Database Management System) application developed by MySQL AB – the subdivision of Sun Microsystems acquired in February 2008 written under C and C++ programming language and designed for cross-platform operating system. The first initial version was released on May 1995 and the last update final version of 5.0 released on October 2005. The project's source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements (EULA).

### **2.4.4 Database Normalization**

Normalization is a database design technique which begins by examining the relationships (called functional dependencies) between attributes. By using a series of tests (described as normal forms), optimal grouping of attributes will make a suitable relations that support data requirements.

According to Ian Gilfillan [7], a table that contains one or more repeating groups is called Unnormalized form (UNF). A first normal form table should have the following requirements; there are no repeating groups, all the key attributes are defined and all attributes are dependent on primary key. The following table is in UNF:

Project number	Project name	Employee number	Employee name	Rate category	Hourly rate
1023	Madagascar travel site	11	Vincent Radebe	A	\$60
		12	Pauline James	B	\$50
		16	Charles Ramoraz	C	\$40
1056	Online estate agency	11	Vincent Radebe	A	\$60
		17	Monique Williams	B	\$50

The data structure would be:

Project number

Project name

1-n Employee numbers (1-n indicates that there are many occurrences of this field - it is a repeating group)

1-n Employee names

1-n Rate categories

1-n Hourly rates

The 1NF solution is implemented and the table would have primary keys as

following:

<i>Project number</i>	Project name	<i>Employee number</i>	Employee name	Rate category	Hourly rate
1023	Madagascar travel site	11	Vincent Radebe	A	\$60
1023	Madagascar travel site	12	Pauline James	B	\$50
1023	Madagascar	16	Charles	C	\$40

	travel site		Ramoraz		
1056	Online estate agency	11	Vincent Radebe	A	\$60
1056	Online estate agency	17	Monique Williams	B	\$50

**Employee project table**

Project number - primary key

Project name

Employee number - primary key

Employee name

Rate category

Hourly rate

A table in second normal form (2NF) should have what it is in first normal form and includes no partial dependencies (where an attribute is dependent on only a part of a primary key). Employee project table thus becomes:

**Employee project table**

Project number - primary key

Employee number - primary key

**Employee table**

Employee number - primary key

Employee name

Rate category

Hourly rate

**Project table**

Project number - primary key

Project name

The third normal form (3NF) table should have what it is on second normal form and contains no transitive dependencies (where a non-key attribute is dependent on another non-key attribute). As a result, the following tables are created:

**Employee project table**

Project number - primary key

Employee number - primary key

**Employee table**

Employee number - primary key

Employee name

Rate Category

**Rate table**

Rate category - primary key

Hourly rate

**Project table**

Project number - primary key

Project name

Normalization however is not necessary to all database structure. Still, all database designers need good old common sense whether it is normalized or not. Most tables are actually already on 3NF and therefore normalize it will give more cost and slow process down.

## 2.5 PHP

PHP is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. PHP stands for *PHP: Hypertext Pre-processor* a recursive acronym which originally called Personal Home Page Tools known as PHP Construction Kit created by Rasmus Lerdorf, a software engineer as well as Apache team member in late 1994 [11]. The first development of PHP began as a set of Common Gateway Interface (CGI) binaries written in the C programming language used by Rasmus for his personal tools to maintain personal homepage before he released it for public. PHP has run through several improvements and expansions to make it more global into several versions by PHP Teams and currently the PHP has reached version 5.3.0 on

July 2009. PHP has become one of the most famous influential web programming languages used by many developers to build their website because of its free to use, portability and easy for maintenance which make it as the third most popular computer programming language, ranking behind Java and C[12].

Position Sep 2009	Position Sep 2008	Delta in Position	Programming Language	Ratings Sep 2009	Delta Sep 2008	Status
1	1	=	Java	19.383%	-1.33%	A
2	2	=	C	16.861%	+1.48%	A
3	5	↑↑	PHP	10.156%	+0.91%	A

Figure 3: Top Programming language index.

### 2.5.1 PHP Advantages

The advantages of using PHP are:

- PHP code is inserted directly into the HTML that makes up a website. When a visitor comes to the website. The code is executed. Because PHP is a server side technology, the user does not need any special browser or plugins to see the PHP in action
- The beauty of PHP lies in its simplicity. It is easy to understand and learn, especially for those with backgrounds in programming such as C, javascript and HTML. The language is similar to C and Perl and PHP also runs on just about every platform including most UNIX, Macs and Windows version.
- PHP does not use a lot of the system's resources so it runs fast and does not tend to slow other processes down. PHP is fairly stable and open source and

the PHP community also works together to fix any bugs. The community offers technical support and continuously updates the code further expanding PHP's capabilities.

- PHP offers many levels of security to prevent malicious attacks. These security levels can be adjusted in the .ini file.
- PHP uses a modular system of extensions to interface with a variety of libraries such as graphics, XML, encryption, etc. Programmers can extend PHP by writing their own extensions and compiling them into the executable or they can create their own executable and load it using PHP's dynamic loading mechanism.
- In addition, PHP has many server interfaces, database interfaces and other modules available.
- The main PHP source repository is loaded with modules and interfaces that users have written and contributed. There you can find modules for flash movies, PDF files, calendars, and more.
- PHP community offers sharing PHP project since PHP is open source. If you are looking for a particular script, chances are another user has already created something similar.

### **2.5.2 PHP Data Processing**

PHP code is executed on the server instead on the client side like Java-Script (JSP) does. When a user tries to open a php page using a web browser, the user's browser sends out a request to the web server. The web server then calls the PHP script on that page. The PHP module executes the script, which then sends out the result in

the form of HTML back to your browser, which you see on the screen. PHP hides all complex stuff such as processing, math calculation, file operation, etc and only gives the user the part that it needed to show such as content and pictures.

The following is the PHP process diagram:

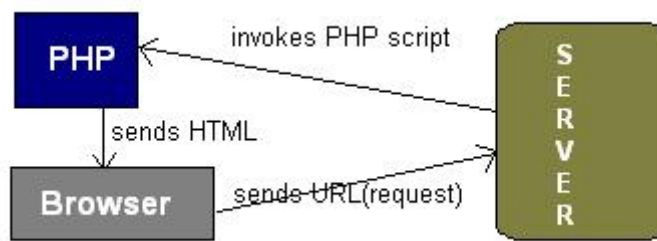


Figure 4: PHP Data Processing


## 2.6 UML Model Diagram

The Unified Modeling Language (UML) is a standard language used for modeling software application needs and creating software blueprints. It is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [23]. In practice, generally UML applies graphical notations to express the design of software projects. By applying UML, project teams could communicate, explore potential designs, and validate the architectural design of the software with less difficulty.


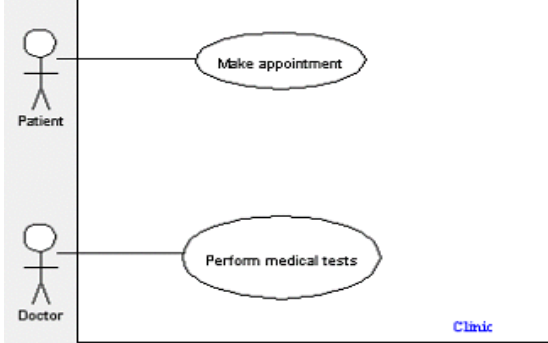
### 2.6.1 Use Case Diagram

Use case diagram is a diagram that depicts the interactions between the system and external systems and users. In other words, it graphically describes who will use the system and in that ways the user expects to interact with the system. A use case

diagram captures the functional aspects of a system and normally domain experts and business analysts should be involved in writing use cases for a given system. A use case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes.

<b>Actors</b>	An actor portrays any entity (or entities) that performs certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system. An actor in a use case diagram interacts with a use case.
<p>An actor is shown as a stick figure in a use case diagram depicted "outside" the system boundary, as shown below.</p> <div style="text-align: center;">  </div> <p>To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the business roles and can be easily identified as actors in the system.</p>	
<b>Use case</b>	A use case in a use case diagram is a visual representation of a distinct business functionality in a system. To choose a business process as a likely candidate for modeling as a use case, we need to ensure that the business process is discrete in nature. As the first step in identifying use cases, we should list the discrete

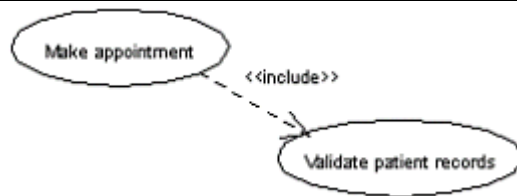


	<p>business functions in your problem statement. Each of these business functions can be classified as a potential use case. A use case is shown as an ellipse in a use case diagram.</p>
	<p>It shows two uses cases: "Make appointment" and "Perform medical tests" in the use case diagram of a clinic system. Discovering such implicit use cases is possible only with a thorough understanding of all the business processes of the system through discussions with potential users of the system and relevant domain knowledge.</p>
<p><b>System boundary</b></p>	<p>A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system.</p>
	

It shows the system boundary of the clinic application. The use cases of this system are enclosed in a rectangle. Note that the actors in the system are outside the system boundary.

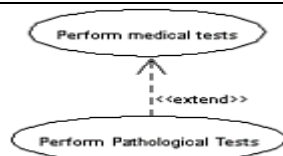
### Relationships in Use Cases.

<b>Include</b>	When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an <i>include</i> relationship.
----------------	--



For example, the functionality defined by the "Validate patient records" use case is contained within the "Make appointment" use case. Hence, whenever the "Make appointment" use case executes, the business steps defined in the "Validate patient records" use case are also executed.

<b>Extend</b>	In an <i>extend</i> relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.
---------------	---



It shows an example of an extend relationship between the "Perform medical

tests" (parent) and "Perform Pathological Tests" (child) use cases. The "Perform Pathological Tests" use case enhances the functionality of the "Perform medical tests" use case. Essentially, the "Perform Pathological Tests" use case is a specialized version of the generic "Perform medical tests" use case.

### Generalizations

A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case.



The "Store patient records (paper file)" (parent) use case is depicted as a generalized version of the "Store patient records (computerized file)" (child) use case. Defining a generalization relationship between the two implies that you can replace any occurrence of the "Store patient records (paper file)" use case in the business flow of your system with the "Store patient records (computerized file)" use case without impacting any business flow. This would mean that in future you might choose to store patient records in a computerized file instead

of as paper documents without impacting other business actions.

A use case specification document should cover the following areas

<b>Actors</b>	List the actors that interact and participate in this use case.
<b>Pre-conditions</b>	Pre-conditions that need to be satisfied for the use case to perform.
<b>Post-conditions</b>	Define the different states in which you expect the system to be in, after the use case executes.
<b>Basic Flow</b>	List the basic events that will occur when this use case is executed. Include all the primary activities that the use case will perform.. This description of actions and responses are the functional requirements. These will form the basis for writing the test case scenarios for the system.
<b>Alternative flows</b>	Any subsidiary events that can occur in the use case should be listed separately. Each such event should be completed in itself to be listed as an alternative flow. A use case can have as many alternative flows as required.

<p><b>Special Requirements</b></p>	<p>Business rules for the basic and alternative flows should be listed as special requirements in the use case narration. These business rules will also be used for writing test cases. Both success and failure scenarios should be described here.</p>
<p><b>Use case relationships</b></p>	<p>For complex systems, it is recommended to document the relationships between use cases. If this use case extends from other use cases or includes the functionality of other use cases, these relationships should be listed here. Listing the relationships between use cases also provides a mechanism for traceability.</p>

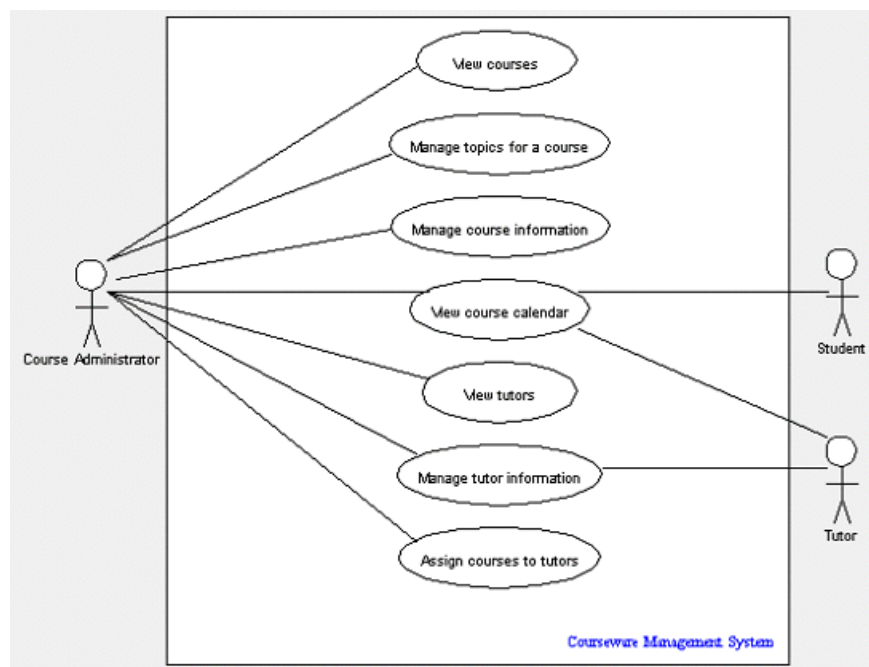


Figure 5: The use case diagram example for the Courseware Management System

## 2.6.2 Context Diagram

The context diagrams describe the way the computer system interacts with the environment, mainly focus on the external entities. Context diagram explains the data flow from the external entities which describe the whole big system. This diagram is the highest level among all the data flow diagrams which only contain a process. This diagram does not have data store notations and looks very simple.

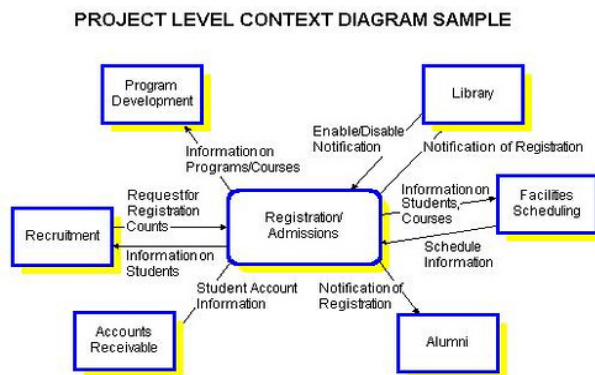


Figure 6: Context Diagram Sample

## 2.6.3 Entity Relationship Diagram

Object-based data model use concepts as entities, attributes, and relationships. This model describes data at the view and conceptual level including with its constraints. One of well known object-based logical model concept is the entity relationship diagram (ERD). These are the elements that build the ERD:

- **Entity**

Entity is an object which is the place where the data is stored.

- **Attributes**

Attributes defines the characteristic and properties of an entity.

- **Relationships**

Relationships show the way the entity shares their information in the database structure.

- **Links**

Links connect the entities, attributes, and relationships.

There are 2 kinds of relationships in an entity relationship diagram

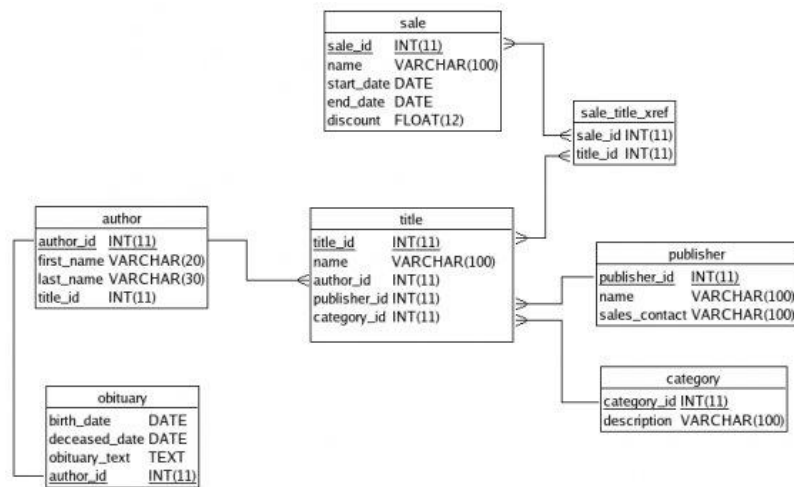


Figure 7: ERD Sample

### 2.6.4 Data Flow Diagram

A data flow diagram (DFD) is a process model used to depict the flow of data through a system and the work or processing performed by the system

The guidelines for constructing DFDs include the following:

- Choose meaningful names for processes, flows, stores, and terminators.
- Number the processes.
- Redraw the DFD as many times as necessary for aesthetics.

- Avoid overly complex DFDs.
- Make sure the DFD is internally consistent and consistent with any associated DFDs.

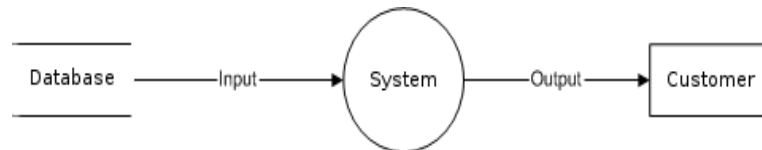


Figure 8: Data flow diagram example.

#### Elements of a DFD:

Process entity	<p>The Process entity identifies a process taking place, it must have at least one input and output.</p> <p>Each process has the following:</p> <ul style="list-style-type: none"> <li>A Number</li> <li>A Name (verb phrase)</li> <li>A Description</li> <li>At least one input</li> <li>At least one output</li> </ul>
Data flow entity	<p>The Data Flow entity identifies the flow of data between processes, data stores &amp; external entities. Each data flow has the following:</p> <ul style="list-style-type: none"> <li>A Name (Noun)</li> <li>A Description</li> </ul>



	One or more connections to a process.
Data Store entity	<p>The Data Store entity identifies stores of data, both manual and electronic. Electronic or digital stores are identified by the letter D, and manual filing systems by the letter M. Each data store has the following:</p> <ul style="list-style-type: none"> <li>A Number</li> <li>A Name</li> <li>A Description</li> </ul> <p>One or more input data flows.</p> <p>One or more output data flows.</p>
External entity	<p>The External Entity identifies external entities which interacts with the system, usually clients but can be within the same organization. Each external entity has the following:</p> <ul style="list-style-type: none"> <li>A Name (Noun)</li> <li>A Description</li> </ul>
The Feedback and Control data	The Feedback and Control data identifies a special purpose. Only the first four elements are needed to create a data flow diagram (DFD).

## DFD Levels

- Context Level

This level shows the overall context of the system and its operating environment and shows the whole system as just one process.



Figure 9: A context level DFD created using Select SSADM.

- Level 0

This level shows all processes at the first level of numbering, data stores, external entities and the data flows between them. The purpose of this level is to show the major high level processes of the system and their interrelation. A process model will have one, and only one, level 0 diagram. A level 0 diagram must be balanced with its parent context level diagram, i.e. there must be the same external entities and the same data flows, these can be broken down to more detail in the level 0, e.g. the "enquiry" data flow could be split into "enquiry request" and "enquiry results" and still be valid.

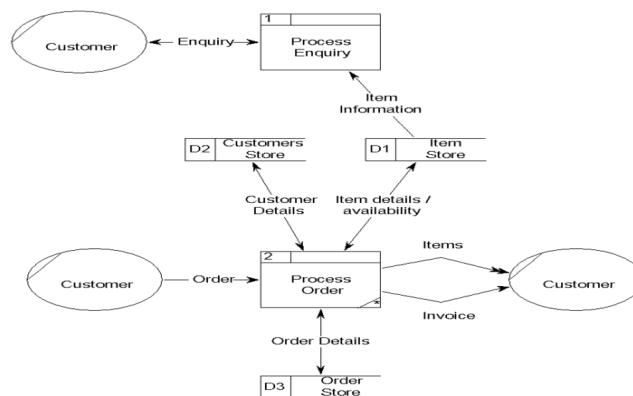


Figure 10: A Level 0 DFD for the same system.

- Level 1

This level is a decomposition of a process shown in a level 0 diagram, as such there should be a level 1 diagram for each and every process shown in a level 0 diagram. In this example processes 1.1, 1.2 & 1.3 are all children of process 1, together they wholly and completely describe process 1, and combined must perform the full capacity of this parent process. As before, a level 1 diagram must be balanced with its parent level 0 diagram.

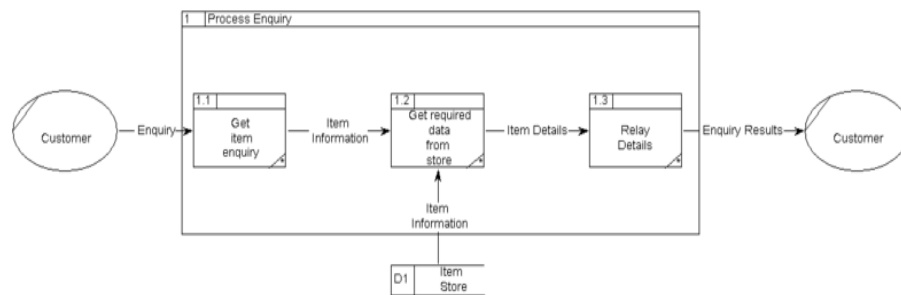


Figure 11: Level 1 DFD showing the "Process Enquiry" process for the same system.

## 2.7 Software Testing

According to Glenford J. Myers[16], software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended. To consider the economics of software testing, it is impractical often impossible to test a program to find all of its errors. Therefore, strategies should be developed with the purpose of finding the most errors with given time and effort. A number of software testing strategies provide

software developer a template for testing with the following general characteristics [18]:

1. To perform effective testing, a software team should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
2. Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
3. Different testing techniques are appropriate at different points in time.
4. Testing is conducted by the developer of the software and (for large projects) an independent test group.
5. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

### **2.7.1 Unit Testing**

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented and the step can be conducted in parallel for multiple components.

### **2.7.2 Integration Testing**

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.

The objective is to take unit tested components and build a program structure that has been dictated by design.

### **2.7.3 User Acceptance Testing**

User Acceptance Testing (UAT) is a process to obtain confirmation by a Subject Matter Expert (SME), preferably the owner or client of the object under test, through trial or review, that the modification or addition meets mutually agreed-upon requirements. In software development, UAT is one of the final stages of a project and often occurs before a client or customer accepts the new system.

## **2.8 Theoretical Frameworks**

Based on the presented theory in the above section, a coherent model or formula which shows the relationships between variables should be formulated to seek for the solution. This model or formula should clarify how the design of solution may be constructed.

### **2.8.1 Iteration Development**

Iterative development slices the deliverable business value (system functionality) into iterations. In each iteration a slice of functionality is delivered through cross-discipline work, starting from the model/requirements through to the testing/deployment [20]. Iteration process purpose is to let developers evaluate the project result based on end users perception. After implement the project design to the real system, generally speaking there always be changing in development due to various factors such as business requirement or new technology implemented for

particular problems. The following figure describes how the iterative process takes place:

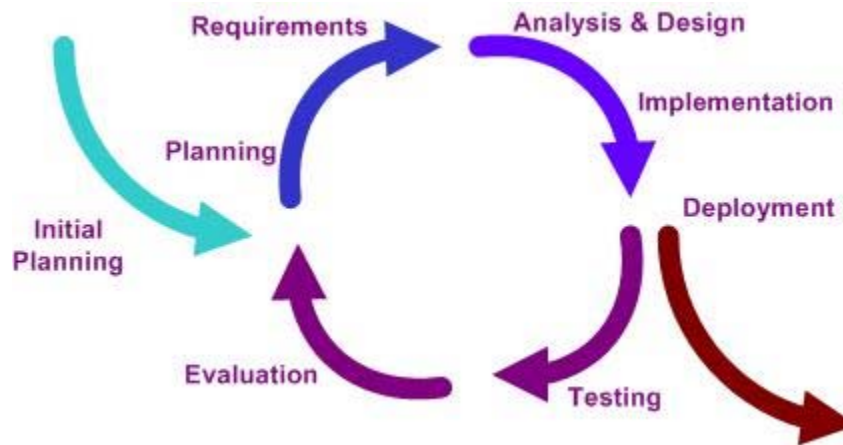


Figure 12: Iterative process

The figure describes a circular process from planning, requirements, analysis & design, and implementation. After testing the result of the project to the client, developers have the evaluation for the first iteration and so on until the project has all required functionalities.

### 2.8.2 Agile Process

According to Ivar Jacobson[19], agility is the key to make the project succeeded. An agile team is able to appropriately respond to changes. Software development is always about changes such as change because of new technology, changes to team members, and changes of all kinds that may have an impact on the product they built or the project that creates the product. The writer adapt agile concept while doing the project in order to meet the project requirement that keep changing to changes.